A Partially Buffered Crossbar Packet Switching Architecture and its Scheduling

Lotfi Mhamdi
Computer Engineering Laboratory
TU Delft, The Netherlands
lotfi@ce.et.tudelft.nl

Abstract

The crossbar fabric is widely used as the interconnect of high-performance packet switches due to its low cost and scalability. There are two main variants of the crossbar fabric: unbuffered and internally buffered. On one hand, unbuffered crossbar fabric switches exhibit the advantage of using no internal buffers. However, they require a centralized and complex scheduler. Internally buffered crossbar fabric switches, on the other hand, overcome the scheduling complexity by means of distributed schedulers. However, they require expensive internal buffers —one per crosspoint. In this paper we propose a novel architecture, namely the Partially Buffered Crossbar (PBC) switching architecture, where a small number of separate *internal buffers are maintained per output. Our goal is to design a PBC switch having the performance of buffered crossbars and a cost comparable to that of unbuffered crossbars. We propose a class of round-robin scheduling algorithms for the PBC switch. Simulations results show that using as few as 8 buffers per output port and irrespective of the number, N, of input ports of the switch, we can achieve even better performance than buffered crossbars that use N buffers per output port.*

1 Introduction

Numerous proposals for identifying suitable architecture for high-performance packet switches have been investigated and implemented both in academia and industry [1][8][10][15]. These architectures can be classified based on various attributes such as queuing schemes, scheduling algorithms, and/or switch fabric topology. The crossbar-based architecture is the dominant architecture for today's high-performance packet switches because of its low cost and scalability. As a result, the vast majority of the commercially available core switches/routers are based on crossbar fabric with virtual output queuing (VOQ) [7]. The crossbar fabric architecture can mainly be classified into two categories: unbuffered or internally buffered.

Extensive research work has been dedicated to unbuffered crossbar switches for more than two decades. Figure 1(a) depicts an Input Queued (IQ) crossbar fabric switch with VOQs at the inputs. The crossbar of an IQ switch runs at the same speed as external input/output ports. In order to maintain this low bandwidth requirement, an unbuffered IQ switch requires a centralized scheduler to resolve two main blocking problems, namely input and output contention. Input contention results from the constraint that an input can send at most one packet every time slot. Similarly, output contention arises from the constraint that an output can receive at most one packet every time slot. These blockings make the task of the scheduler complex and the packets delay unpredictable. As a result, the switch performance essentially depends on its scheduling algorithm. Different classes of scheduling algorithms have been proposed [5][9][11][12]. Unfortunately, for high-bandwidth IQ switches, almost all scheduling algorithms are either too complex to run at high speed or fail to exhibit satisfactory performance. This is mainly attributed to the centralized design of these schedulers and to the nature of the unbuffered crossbar switching architecture.

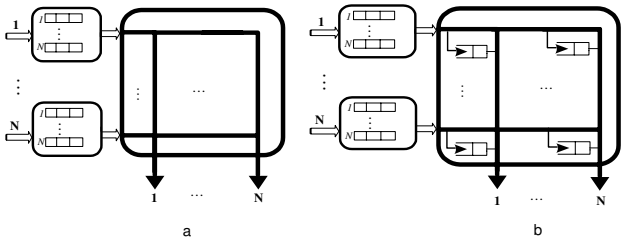


Figure 1. Crossbar fabric variants: (a) Unbuffered crossbar fabric. (b) Buffered crossbar fabric, with N^2 internal buffers.

In order to overcome the scheduling complexity faced by IQ unbuffered crossbar switches, buffered crossbar switches have been proposed [13][16][17]. Figure 1(b) depicts a buffered crossbar switch, a crossbar where a limited amount

978-1-4244-2703-1/08/$25.00 ©2008 IEEE

of memory is added per crosspoint. The existence of internal buffers relaxes the output contention constraint, making the scheduling task much simpler. Buffered crossbar use distributed and independent schedulers (one per input/output port) to switch packets from the input to the output ports of the switch. A scheduling cycle consists of input scheduling, output scheduling and flow control to prevent internal buffer overflow. Efficient scheduling algorithms have been proposed for this architecture [6][13][19]. The scheduling simplification comes at the expense of a costly crossbar. The crossbar has to contain N^2 internal buffers, where N is the number of input/output ports of the switch. The number of internal buffers grows quadratically with respect to the switch size and linearly with round trip delays [1]. This makes buffered crossbar switches highly expensive and hence less appealing.

In this paper, we propose a novel architecture termed the Partially Buffered Crossbar (PBC) switching. The PBC is designed to be a switching architecture having both the performance of buffered crossbars and a cost comparable to unbuffered crossbars. The PBC switch, depicted in Figure 2, contains a small number of separate internal buffers, $B \ll N$, per fabric output. We propose a class of pipelined scheduling algorithms for the PBC switch and study their performance under various traffic patterns. Experiments show that setting the number of internal buffers per output to $B = 8$ is sufficient for the PBC to achieve optimal performance irrespective of the switch size, N. Previous work proposed similar architecture to the PBC switch [3][4]. Our work differs from [3][4] both at the architectural and the scheduling level. While the architecture in [3] relies on internal shared memory per output port, our PBC architecture uses separate internal buffers per output and hence avoiding the requirement of expensive shared buffers. Second, the architecture proposed by [4] was targeting multistage switches whereas our proposed architecture targets single stage switches. On the scheduling level, our proposed algorithms outperform those proposed by [3][4] as will be shown in the experimental results section of this paper.

The remainder of the paper is structured as follows: Section 2 presents the PBC architecture and its scheduling. In Section 3, we introduce our set of scheduling algorithms. The first algorithm is called Distributed Round Robin (DRR). It is based on round robin grant and credit schedulers per input and output port respectively. Because of the credit release delay experienced by DRR, we propose an alternative algorithm named DROP that drops not accepted grants every time slot and consequently minimizing the credit release delay. We also propose an enhanced version of the DROP algorithm, named DROP-PR, that selects grants based on output priority. Section 4 presents the performance study of our algorithms under various settings. Finally, Section 5 concludes the paper.

2 The Partially Buffered Crossbar Architecture (PBC)

This section introduces the Partially Buffered Crossbar switching architectural organization and its scheduling.

2.1 Switch Model

We consider the Partially Buffered Crossbar switching architecture (PBC) depicted in Figure 2. The switch operates on fixed sized packets (cells). Variable size packets are segmented into fixed sized cells while inside the switch and reassembled back to packets upon their exit. The PBC has N input and N output ports. When a cell, destined to output j arrives at input i, it gets queued in $VOQ_{i,j}$ while waiting its turn to be selected by the input scheduler (IS_i). There are N input schedulers, each controls the transfer of cells from one input line card to the internal fabric buffers. The input scheduler decision is coordinated with a grant scheduler that manages the internal buffers availability for each output. All input and grant schedulers are embedded within the buffered crossbar core. When a new cell arrives to $VOQ_{i,j}$, the index of $VOQ_{i,j}$ is forwarded to IS_i inside the fabric. This is achieved using $\log N$ bit signals. When IS_i makes a decision, it sends $\log N$ bit signals to the line card indicating the selected VOQ. In total $2N\log N$ flow control signals are used, in contrast to a fully buffered crossbar that requires N^2 bit signals. The input and grant schedulers communicate throughout a grant queue (GQ) maintained per input. There are N GQs, one per input, and each contains N entries, one per output. When a grant scheduler (GS), j, sends a grant g to input i, $GQ_{i,j}$ is set to one. Once input i accepts g, $GQ_{i,j}$ is reset to zero.

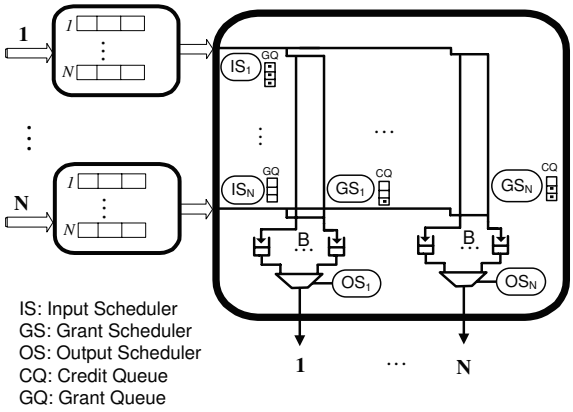


Figure 2. The Partially Buffered Crossbar (PBC) switching architecture.

The crossbar fabric contains a small number of internal buffers. These internal buffers are maintained per output port and there are $B \ll N$ separate internal buffers per output port. The fabric has B internal buses per output, one per internal buffer. These buses run at the same bandwidth as the external line rates. These buses are needed in order to maintain low bandwidth. If we use one bus per output, instead, its bandwidth is required to be B times the external bandwidth in addition to intermediate buffering of cells which is costly. Each output port contains an output scheduler (OS) that arbitrates cells departures from the internal buffers to the output queue. There are N credit queues (CQ), one per output. Each CQ contains B entries and CQ_j records the availability of the internal buffers belonging to output j. A CQ is decremented whenever a grant is sent to the input, and incremented during output scheduling.

2.2 Scheduling Process

The scheduling process in the PBC switch is a combination of unbuffered as well as buffered crossbar scheduling. A scheduling cycle consists of input scheduling and output scheduling phases as in buffered crossbars. The input scheduling phase resembles a scheduling cycle in unbuffered crossbars, as it is based on request-grant-accept handshaking protocol. The input scheduling phase works as follows: During time slot, t, each non empty (eligible) $VOQ_{i,j}$ sends a request to the grant scheduler (GS) corresponding to output port j. Subject to internal buffers availability (CQ_j) and the grant scheduler policy, a grant may be sent back to the input scheduler i and stored in its GQ ($GQ_{i,j}$ set to 1). At the same time, input scheduler (IS_i) picks a VOQ Head of Line (HoL) cell to be transferred to the internal buffers based on its GQ, excluding the current grants of time slot, t. Meaning that the outcome of GS_i at time t is only valid during time slot $t + 1$ or later. This allows a two-stage pipelined scheduling, avoiding the need for synchronized coordination between the grant schedulers and the input (accept) schedulers on a time slot basis as does iSlip [9] and PIM [2].

Because the number of internal buffers, B, maintained per output is much smaller than the number of competing inputs, N, one has to be careful as to how to service cells during output scheduling. The internal buffers are separate and cells from the same VOQ may arrive to different internal buffers during consecutive time slots. In this case, we have to maintain in sequence cell delivery. To this end, we employed a First-Come-First Serve (FCFS) output scheduling to ensure in order cell delivery [18]. A cell departure from the internal buffers at output j, causes CQ_j to increments by one. A cell arrival, from an input, to an internal buffer at output j causes CQ_j to decrement by one. Likewise, the grant queue, at an input i, is incremented whenever a grant scheduler sends a grant to input i, and decremented whenever a cell departs the input port i.

The input scheduling in PBC is similar to the iterative matching performed by unbuffered crossbar scheduling. However, maintaining a small number of internal buffers makes it significantly different. The absence of internal buffers in an unbuffered crossbar switch meant that a grant arbiter can grant at most one input, to avoid output contention. Similarly, an input accept arbiter has to accept at most one grant, to avoid input contention. The PBC scheduling, while keeps the input contention constraint enforced, relaxes the output contention constraint by allowing conflicting cells (up to B) to be admitted to the internal buffers for the same output. This is equivalent to unbuffered crossbars schedulers accepting one grant and storing other $B - 1$ grants instead of discarding all the rest. Unbuffered crossbar schedulers resort to multiple iterations to improve the match size. Using one iteration for a random scheduling policy such as PIM [2], the probability that an input will remain ungranted is $(\frac{N-1}{N})^N$, where N is the port count of the switch [9]. As N increases, this probability tends to $\frac{1}{e}$. If we use the same random scheduling policy in the PBC with B internal buffers per output and assuming that packets are flushed every time slot (memoryless Markov process), the probability that an input remains ungranted is $(\frac{N-B}{N})^N$. With increasing N, this probability tends to $\frac{1}{e^B}$ (almost 0 for $B \geq 4$).

3 Scheduling in PBC

This section introduces our set of scheduling algorithms for the PBC switching architecture. We propose a class of round robin based input scheduling algorithms. The output scheduling we use here is based on FCFS policy, as discussed in the previous section, and will remain the same throughout the whole article. Each time the output scheduler, at output j, performs its FCFS selection and sends a cell to the output queue, it increments CQ_j by one. As for the input scheduling, we propose a set of round robin based schedulers. The first algorithm we propose is named Distributed Round Robin (DRR) and will be described in the the following section.

3.1 The Distributed Round Robin (DRR) Algorithm

The DRR algorithm is similar to the scheme proposed by [3] and its grant scheduler's pointer update is the same as iSlip [9]. This is because, a grant sent by a GS to an input scheduler, if not immediately accepted, is stored and will eventually get accepted in the short run (less than N time slots later). The only difference between DRR and

iSlip lies in DRR's input scheduler pointer update mechanism. Unlike iSlip, DRR's input scheduler pointers are fully unsynchronized, as in [5]. Additionally, the DRR differs in its way of assigning cells to internal buffers when they leave the input $VOQs$. However, this is specific to the PBC architecture. Cell assignment to internal buffers can be realized by maintaining a separate field in each entry of the GQ. Whenever the GS_j grants to input i, it sets the entry $GQ_{i,j} = 1$ and the field corresponding to the internal buffer to the index of the next free internal buffer $B_{k,j}$. This is fairly feasible, especially if we envision embedded schedulers design such as [14].

Algorithm 1 DRR

Grant Phase:
For each output, j, do

. While there are credits in CQ_j do

- Starting from the grant pointer g_j index, send a grant to the first input, i, that requested this output (set $GQ_{i,j} = 1$).
- Decrement CQ_j by one. /** CQ_j is incremented by one during output scheduling phase. **/
- Move the pointer g_j to location $(i + 1)\ (mod\ N)$.

Input Scheduling Phase:
For each input, i, do

. Starting from the input pointer a_i index, select the first non empty $VOQ_{i,j}$ for which $GQ_{i,j} = 1$ and send its HoL cell to the internal buffer.

. Set $GQ_{i,j} = 0$.

. Move the pointer a_i to location $(a_i + 1)\ (mod\ N)$.

The DRR scheme experiences the same credit release delay as with the scheme in [3]. Credit release delay arises when multiple grant schedulers grant to the same input concurrently. Because DRR returns credits one at a time (input contention), credits may not return fast enough. This, consequently, affects the rate at which grants are sent back to other inputs, hence delaying the transfer of cells from the input line cards. In order to relief this delay, a grant throttling mechanism was proposed in [3]. It consists of setting a threshold (TH) for the grant queue and requests from an input are eligible so long as the grant queue relative to their input is less TH. While this mechanism speeds up the credits release, it does not completely eliminate it or minimize it. Additionally, it requires some extra signaling to control the grant queues thresholds.

Our solution to credits release delay is different. We do not want to just lower the credit release delay. Instead, our goal is to completely eliminate it or set it to its absolute minimum. First, we need to quantify this delay. A grant has to wait up to N time slots in each grant queue before its associated credit is released back. Therefore, an input request waits at most $\frac{N^2}{B}$ time slots before it gets granted. That is why when $B = 1$, the performance of DRR is similar[1] to iSlip (see Figure 4(a) and 4(b)). However, as B increases, the credit release delay decreases ($\frac{N^2}{B}$ decreases). Thus, the problem of credit release delay is now reduced to solving the grant queuing delay. Minimizing the credit release delay means altering the grant mechanism to reduce thw grant queuing delay. We, therefore, modified the way DRR allocates grants per input, hence a new grant scheduler. Instead of *storing* the grants, that are not accepted, in a grant queue while they await their turn to be accepted, and therefore credits waiting (delayed) to get released, we simply *drop* them, hence the name of the new scheme DROP.

3.2 The DROP Algorithm

Dropping the not accepted grants implies that the pointer updating scheme of the grant scheduler has to change. This is because, otherwise, using the iSlip pointer updating mechanism results in pointer synchronization similar to RRM [9]. This convergence of iSlip to RRM, under our settings, comes from the two-stage pipeline scheduling relative to the PBC architecture. Recall that DRR stores the not accepted grants, guaranteeing their immediate or soon acceptance, and therefore the grant pointer can safely be updated. This is similar to iSlip, where the grant pointer gets updated only on accept (which is in the same time slot or stage). With the DROP scheme, however, dropping the not accepted grants during the accept phase (second pipeline stage) means that we have to update the grant pointer (first stage) which is already late and out of date (already performed during last time slot). Therefore, using the iSlip pointer update mechanism in DROP means 'blindly' updating the grant pointers, which leads to synchronization and poor performance as in RRM [9]. To overcome this problem, we use a fully unsynchronized grant pointers settings, similar to [5]. The grant pointers are initially set to different positions and are always incremented by one irrespective of the accept/drop outcome. Proceeding this way, a request waits no more than N time slots before it gets granted. This is to be compared to $\frac{N^2}{B}$. The specification of the DROP algorithm is as below.

3.3 The Prioritized DROP Algorithm

We wanted to further improve the performance of the DROP scheme. This need for enhancing DROP stems from two reasons. From one hand, DROP works in a two-stage pipeline meaning that it experience some initial delay. From the other hand, DROP is designed for the PBC architecture that is assumed to have a limited small number of internal

[1]The slightly lower delay of DRR results from its input scheduler pointer update mechanism. Unlike iSlip, DRR input scheduler pointers are fully unsynchronized, as in [5]

Algorithm 2 DROP

Grant Phase:
All output pointers, g_j, are initialized to different positions.
For each output, j, do

. Set CQ_j equals to the number of non full internal buffers for output j.

. While there are credits in CQ$_j$ do

- Starting from g_j index, send a grant to the first input, i, that requested this output (set $GQ_{i,j} = 1$).
- Decrement CQ_j by one.

. Move the pointer g_j to location $(g_j + 1)\ (mod\ N)$.

Input Scheduling Phase:
All input pointers, a_i, are initialized to different positions.
For each input, i, do

. Starting from a_i index, select the first eligible $VOQ_{i,j}$ for which $GQ_{i,j} = 1$ and send its HoL cell to the internal buffer.

. Drop the remaining grants (reset GQ: $GQ_{i,*} = 0$).

. Move the pointer a_i to location $(a_i + 1)\ (mod\ N)$.

buffers per output, B. With these observations, we propose an enhanced version of DROP that services grants based on output priority. We call this version prioritized DROP and refer to it as DROP-PR. The specification of the DROP-PR scheme is as follows:

Algorithm 3 DROP-PR

Grant Phase:
All output pointers, g_j, are initialized to different positions.
For each output, j, do

. Set CQ_j equals to the number of non full internal buffers for output j.

. Set the priority bit, P, to the logic OR of CQ_j entries.

. While there are credits in CQ_j do

- Starting from g_j index, send a grant to the first input, i, that requested this output (set $GQ_{i,j} = 1$ and add bit P).
- Decrement CQ_j by one.

. Move the pointer g_j to location $(g_j + 1)\ (mod\ N)$.

Input Scheduling Phase:
All input pointers, a_i, are initialized to different positions.
For each input, i, do

. Starting from a_i index, select the first non empty $VOQ_{i,j}$ for which $GQ_{i,j} = 1$ and bit $P = 1$ and send its HoL cell to the internal buffer.

. If no HoL cell is selected, Then

- Starting from a_i index, select the first non empty $VOQ_{i,j}$ for which $GQ_{i,j} = 1$ and send its HoL cell to the internal buffer.

. Drop the remaining grants (reset GQ: $GQ_{i,*} = 0$).

. Move the pointer a_i to location $(a_i + 1)\ (mod\ N)$.

When selecting a cell for input scheduling, the DROP-PR scheme takes into account the occupancy of the internal buffers belonging to an output. When a grant scheduler grants an input request, it sends back the grant with an additional priority bit. The priority bit informs the granted input whether or not the grant comes from an output with empty internal buffers (prioritized output). During the input scheduling phase (second pipeline stage), the input scheduler first gives priority to grants for which the priority bit is set to 1. The priority bit is obtained by logically OR-ing the entries of the Credit Queue (CQ).

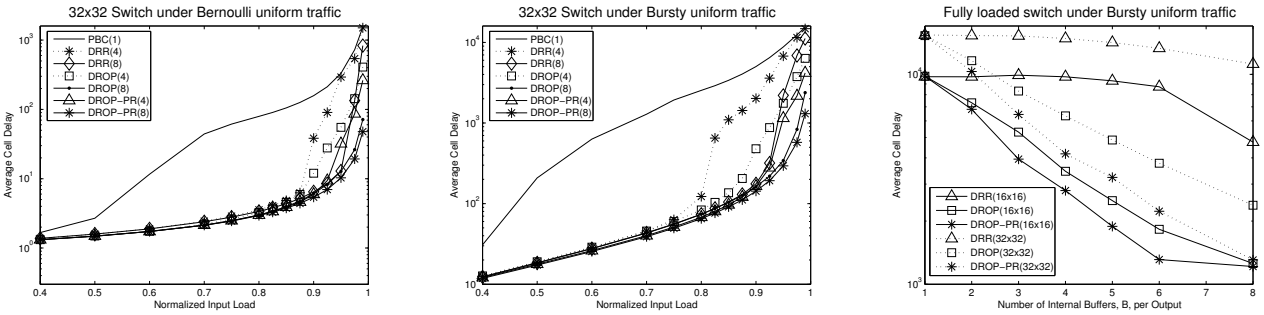
4 Performance Analysis

This section presents the performance study of the PBC switching architecture. The study is aimed at comparing our proposed architecture to both the unbuffered and the buffered crossbar fabric architectures as well as an ideal Output Queued (OQ) switch. The experiments are carried out under three input traffic patterns: Bernoulli uniform, Bursty uniform and Unbalanced traffic [19]. Simulations run for 1 million time slots and statistics are gathered when fourth of the simulation length has elapsed. We tested different PBC switch sizes, each with different internal buffer settings. However, due to space limitation, we restrict the results to switch sizes of 16×16 and 32×32 only.

4.1 Uniform Traffic

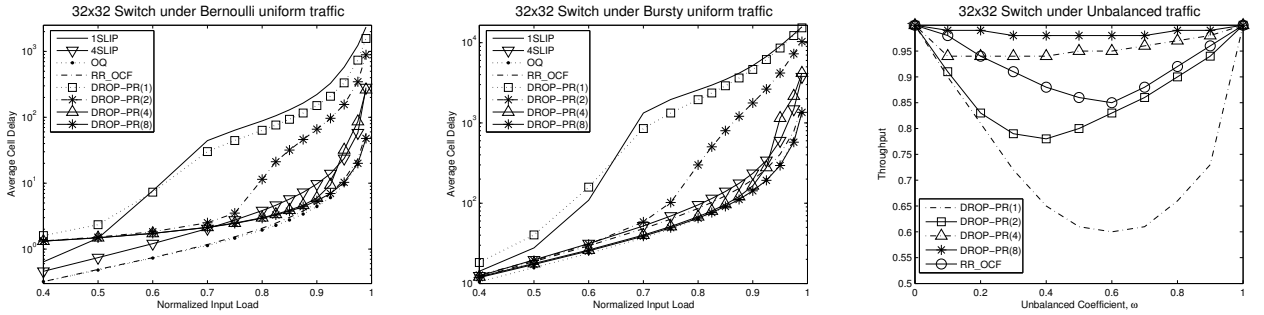
Figure 3(a) illustrates the average cell delay performance of each of our proposed algorithms under Bernoulli uniform traffic. We measured the delay of each of the algorithms with different internal buffers settings. When the number of internal buffers per output, B = 1, the three algorithms have the same delay because there is no credit release delay. For this reason we denote any of the algorithms by "PBC(1)" in the Figure. Note that the performance of DRR is equivalent to that of the algorithm proposed by the related work [3][4]. We can observe throughout the experimental study the improvement of the DROP and DROP-PR algorithms over that of DRR (related work). Increasing B to as few as 4 internal buffers per output boosts up the performance by an order of magnitude. The delay improvement is less sharp for B between 4 and 8. When B = 4 or more, the granting likelihood is almost 100% from each grant scheduler to each input scheduler. The same behavior is observed under Bursty uniform arrivals, as depicted in Figure 3(b).

Assessing the performance of each of our three proposed algorithms (DRR, DROP and DROP-PR respectively) requires tuning different parameters such as switch size, the number of internal buffers per output and the input traffic loads, hence many plots. However, because we are mostly interested in switch cell delays under heavy input loads, in



(a) Bernoulli uniform arrivals (b) Bursty uniform arrivals (c) Bursty uniform arrivals at 99% load

Figure 3. Average cell delay performance of the PBC algorithms under uniform traffic



(a) Bernoulli uniform arrivals (b) Bursty uniform arrivals (c) Throughput performance under Unbalanced traffic

Figure 4. Delay performance comparison between a PBC switch and both unbuffered and fully buffered crossbar switches.

Figure 3(c) we fixed the input load to be 99% and varied the switch size as well as the internal buffer sizes for each algorithm. Figure 3(c) depicts the performance of each of out algorithms under Bursty uniform arrivals for a 16×16 and 32×32 respectively. We observed the average cell delay as function of the number of internal buffers per output, B. We can see that when $B = 1$, unbuffered crossbar switch, the three algorithms have the same delay which is comparable to iSlip with one iteration. This is because when $B = 1$, every request waits the same time (N^2 times slots at most) before it gets served. However, with increasing B, both DROP and DROP-PR have lower cell delays than DRR because of their fast credits release. Recall that the credit release delay (and consequently grant and service delays) of DRR is $\frac{N^2}{B}$. However, both DROP and DROP-PR have a credit release delay of N. As B increases, (especially as B approaches N, not shown in the Figures) all the algorithms have the same delay. However, we are only interested in PBC switches with $B \ll N$. DROP-PR has the overall lowest delay because it prioritizes outputs with empty internal buffers, resulting in more balanced internal buffers occupancies and hence lower cell latencies.

We compared the average cell latency of the DROP-PR algorithm for 32×32 PBC switch to that of an unbuffered crossbar switch, a fully buffered crossbar switch and an ideal OQ switch. The iSlip algorithm is used for the unbuffered crossbar architecture. The fully buffered crossbar switch uses input round robin (RR) scheduling and Oldest Cell First (OCF) output scheduling. The comparison is done under unform Bernoulli and Bursty arrivals. Figure 4(a) depicts the performance of DROP with different internal buffers, iSlip (with 1 and 4 iterations), RR_OCF and OQ. Irrespective of whether the input traffic is Bernoulli or Bursty, DROP-PR(1) (1 refers to $B = 1$) has a similar behavior to 1Slip (and to DRR(1) since $B = 1$), as described earlier (see Section 3.1). As B increases, the delay of DROP-PR significantly decreases. It approaches that of an ideal OQ with just 8 internal buffers per output ($B = 8$). Similar performance is observed under burtsty arrivals (Figure 4(b)). These results suggest that a PBC switch can replace a buffered crossbar, or even an ideal OQ switch with as few as 8 internal buffers per output. These results can also give a switch designer the choice depending on the constraints and needs. For example, if the delay-cost product

is the main target, one may replace an unbuffered crossbar switch employing 4Slip with a PBC switch with just 4 internal buffers per output (see the delays of 4Slip and DROP-PR(4) in Figure 4(a) and Figure 4(b) respectively). However, if performance is the main target with a little flexibility in cost, one may employ a PBC switch with 8 internal buffers per output as it exhibits ideal performance.

4.2 Unbalanced Traffic

In order to further endorse our claims with respect to the PBC performance, we analyzed the stability of a 32×32 PBC switch under unbalanced traffic arrivals. We employed the unbalanced traffic proposed in [19]. We set the switch input load to be 100% and we varied the unbalanced coefficient, w and observed the switch throughput performance. Figure 4(c) depicts the performance of the PBC switch with DROP-PR algorithm and different internal buffer setting and compares it to that of a buffered crossbar (RR_OCF). We can see that, with $B = 2$ internal buffers per output, we can achieve comparable throughput to a fully buffered crossbar when the unbalanced coefficient, w, is higher than 0.6. This translates into saving worth of up to 960 internal buffers. Setting $B = 4$, we can achieve higher throughput than a fully buffered crossbar. The ideal throughput of the PBC is reached when using 8 internal buffers per output.

5 Conclusion

A novel Partially Buffered Crossbar (PBC) switching architecture is proposed in this article. The PBC switch is designed to be the best compromise between unbuffered crossbars and the fully buffered crossbars. From one hand, it overcomes the high cost of fully buffered crossbars that use N^2 internal buffers, by using just a few number of internal buffers per output irrespective of N. From the other, it overcomes the scheduling complexity experienced by unbuffered crossbars by means of distributed scheduling algorithms. We proposed a class of distributed and pipelined round robin scheduling algorithms for the PBC architecture. In particular, the DROP-PR scheme was shown to have optimal performance under different traffic patterns and switch sizes. Our experimental results showed that a PBC switch with 8 internal buffers per output exhibits ideal performance, irrespective of the switch size, N.

References

[1] F. Abel, C. Minkenberg, P. Luijten, M. Gusat, and I. Iliadis. A Four-Terabit Packet Switch Supporting Long Round-Trip Times. *IEEE Micro*, 23(1):10–24, Jan/Feb 2003.

[2] T. Anderson, S. Owicki, J. Saxe, and C. Thacker. High Speed Switch Scheduling For Local Area Networks. *ACM Transactions on Computer Systems*, pages 319–352, November 1993.

[3] N. Chrysos and M. Katevenis. Scheduling in Switches with Small Internal Buffers. *IEEE Globecom*, pages 614–619, November 2005.

[4] N. Chrysos and M. Katevenis. Scheduling in Non-Blocking Buffered Three-Stage Switching Fabrics. *IEEE INFOCOM*, April 2006.

[5] Y. Jiang and M. Hamdi. A Fully Desyncronized Round-Robin Matching Scheduler for a VOQ Packet Switch Architecture. *IEEE workshop on High Performance Switching and Routing*, pages 407–411, May 2001.

[6] M. Katevenis, G. Passas, D. Simos, I. Papaefstathiou, and N. Chrysos. Variable Packet Size Buffered Crossbar (CICQ) Switches. *IEEE International Conference on Communications (ICC 2004)*, 2:1090–1096, June 2004.

[7] N. McKeown. *Scheduling Algorithms For Input-Queued Cell Switches*. PhD thesis, University of California at Berkeley, May 1995.

[8] N. McKeown. A Fast Switched Backplane for a Gigabit Switched Router. *Business Commun. Rev.*, 27(12), 1997.

[9] N. McKeown. iSLIP Scheduling Algorithm for Input-Queued Switches. *IEEE Trans. On Networking*, 07(02):188–201, April 1999.

[10] N. McKeown, M. Izzard, A. Mekkittikul, B. Ellersick, and M. Horowitz. The Tiny Tera: A Packet Switch Core. *IEEE Micro*, pages 26–33, Jan/Feb 1997.

[11] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand. Achieving 100% Throughput in Input-Queued Switches. *IEEE Trans. On Communications*, 47(08), 1999.

[12] A. Mekkittikul. *Scheduling Non-Uniform Traffic In High Speed Packet Switches and Routers*. PhD thesis, Stanford University, November 1998.

[13] L. Mhamdi and M. Hamdi. MCBF: A High-Performance Scheduling Algorithm for Buffered Crossbar Switches. *IEEE Communications Letters*, 07(09):451–453, Spetember 2003.

[14] L. Mhamdi, C. Kachris, and S. Vassiliadis. A Reconfigurable Hardware Based Embedded Scheduler For Buffered Crossbar Switches. *ACM/SIGDA Fourteenth International Symposium on Field Programmable Gate Arrays (FPGA 2006)*, pages 143 – 149, February 2006.

[15] C. Minkenberg and T. Engbersen. A Combined Input And Output Queued Packet-Switched System Based On A Prizma Switch-On-A-Chip Technology. *IEEE Comm. Magazine*, 38(2):70–77, December 2000.

[16] M. Nabeshima. Performance Evaluation of Combined Input- and Crosspoint-Queued Switch. *IEICE Trans. On Communications*, B83-B(3), March 2000.

[17] S. Nojima, E. Tsutsui, H. Fukuda, and M. Hashimmoto. Integrated Packet Network Using Bus Matrix. *IEEE Trans. on Commun.*, 05(08):1284–1291, October 1987.

[18] R. Rojas-Cessa and Z. Dong. Combined Input-Crosspoint Buffered Packet Switch with Flexible Access to Crosspoint Buffers. *IEEE International Caribbean Conference on Devices, Circuits and Systems, Playa del Carmen*, April 2006.

[19] R. Rojas-Cessa, E. Oki, Z. Jing, and H. J. Chao. CIXB-1: Combined Input One-Cell-Crosspoint Buffered Switch. *Proceedings of the 2001 IEEE WHPSR*, pages 324–329, May 2001.