2007 IEEE International Conference on Signal Processing and Communications (ICSPC 2007), 24-27 November 2007, Dubai, United Arab Emirates

HYBRID SCHEDULING IN HIGH-SPEED PACKET SWITCHES

Lotfi Mhamdi *Stamatis Vassiliadis*

TU Delft
Computer Engineering Lab
Delft, The Netherlands

ABSTRACT

The crossbar fabric switching architecture is popular due to its low cost and scalability. However, crossbar fabric switches require centralized and highly complex scheduling. Fully buffered crossbars overcome the scheduling complexity, by means of distributed scheduling. However, the scheduling simplicity comes at the expense of a complex fabric, where internal buffering is required in each crosspoint. This article proposes a novel architecture, namely the Hybrid Buffered Crossbar (HBC) switching architecture, where only few internal buffers are maintained per output. Our goal is to design a HBC switch having the performance of buffered crossbars and the cost of unbuffered crossbars. We propose a round robin scheduling algorithm for the HBC switching architecture. Our algorithm is a combination of unbuffered as well buffered crossbar scheduling. Simulation results show interesting trade offs in choosing the optimal number of internal buffers when designing a hybrid crossbar switch.

Index Terms— Scheduling, Buffered Crossbar Fabric.

1. INTRODUCTION

The crossbar-based architecture is the dominant architecture for today's high-performance packet switches (IP routers, ATM switches, Ethernet switches) for at least three reasons. First, they are more scalable than their direct competitors, shared-bus and shared memory. This is due to the limitation in bus transfer bandwidth and/or the limitation in the memory access bandwidth. Second, they provide simple point-to-point connections, which means they can operate at very high-speed (up to 40Gb/s). Third, they can support multiple I/O transactions simultaneously. This can increase the aggregate bandwidth of the system, which can be in the hundreds of Gbps. There are two main variants of the crossbar fabric: unbuffered and internally buffered.

Generally, unbuffered crossbar switches use input queues at the inputs, known as Input Queued (IQ) switches. The most popular input queuing architecture is the Virtual Output Queuing (VOQ), as depicted Figure 1-(a). The VOQs are employed to avoid the Head of Line (HoL) blocking phenomenon [1]. However, the sophisticated VOQ architecture requires a scheduler to arbitrate packet transfer across the crossbar fabric. Because the inputs, the crossbar fabric core and the outputs run at the same speed as the external line, the scheduler must resolve input and output port contentions. In every time slot, every input can send at most one packet and every output can receive at most one packet. This is similar to bipartite graph matching. Optimal solutions existed, such as Maximum Weight Matching algorithms [2]. Practical algorithms have also been proposed such as [3][4]. 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 the nature of the unbuffered crossbar switching architecture.

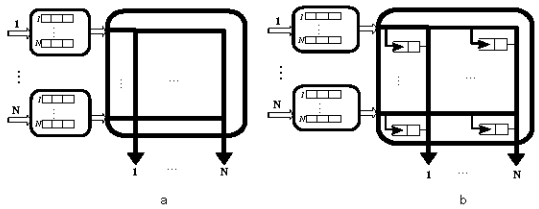

Figure 1. Crossbar Fabric Variants: (a) Unbuffered Crossbar Fabric. (b) Buffered Crossbar Fabric, with N^2 Internal Buffers.

As IQ unbuffered crossbar switches reach their practical limitations due to higher port numbers and data rates, buffered crossbar switches (CICQ) are gaining a lot of interest due to their great potential in solving the scheduling complexity and scalability issues faced by their bufferless predecessors [5]. The CICQ switching architecture is identical to an IQ switch, where small buffers are added inside the crossbar fabric as depicted in Figure 1-(b). The existence of internal buffers avoids the need for a synchronized contention free transfer of packets between inputs and outputs, as in IQ switches, by allowing conflicting packets to enter the crossbar core simultaneously [6]. Buffered crossbars use distributed and independent schedulers (one per input/output port) to switch packets from the input to the output ports of the switch. One major drawback of CICQ switches is their expensive crossbar fabric. A CICQ is required to maintain up to N^2 internal buffers, where N is the number of input/output port of the switch. This high number of internal buffers grows quadratically with the switch size and linearly with round trip delays (RTT), limiting the CICQ scalability.

In this article, we propose a new architecture where a small number of internal buffers is maintained per each output of the crossbar fabric. We refer to our novel architecture as the Hybrid Crossbar Switching architecture (HBC), as depicted in Figure 2. Our goal is to design a HBC with a fixed small number of internal buffers irrespective of the switch size, N. The envisioned HBC is required to exhibit the performance of buffered crossbars while having a cost comparable to unbuffered crossbars. We also proposed an appropriate scheduling algorithm for the HBC architecture. Our proposed scheduling scheme is based on round robin policy and termed, Partial Round Robin (PRR). The scheduling is a combination of unbuffered scheduling as well as buffered scheduling. We conducted an extensive simulation study and showed that a HBC architecture with only 4 internal buffers per output can replace an unbuffered crossbar employing iSlip [3] with 4 iterations. On the other hand, using a HBC with only 8 internal buffers per output can replace a fully buffered crossbar irrespective of its port count, N.

The remainder of this article is structured as follows: Section 2 presents the HBC architecture and its scheduling. We introduce our PRR algorithm and its properties in Section 3. We also discuss the requirement for an output scheduling policy that ensures in-order cell delivery. Section 4 presents a detailed simulation study under various settings and illustrates different trade offs. Finally, Section 5 concludes the paper.

2. THE HYBRID BUFFERED CROSSBAR ARCHITECTURE (HBC)

This section introduces the Hybrid Buffered Crossbar switching architectural organization along with its scheduling. We consider the Hybrid Buffered Crossbar switching architecture (HBC) depicted in Figure 2. The switch operates on fixed size packets (cells). Variable size packets are segmented into fixed size cells while inside the switch and reassembled back into packets upon their exit. The HBC 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, one per input port that control the transfer of cells from the input line cards to the internal fabric buffers. The input scheduler decision is coordinated with a grant scheduler that manages the internal buffers availability for each output. The input and grant schedulers communicate throughout a request grant accept (RGA) handshaking protocol, as in iSlip [3]. The only, yet significant, difference in our architecture is that each grant scheduler is permitted to send more than one (up to B) grants every time slot.

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. Each output port contains an output scheduler (OS) that arbitrates cell departures from the internal buffers to the output queues. There are N credit queues

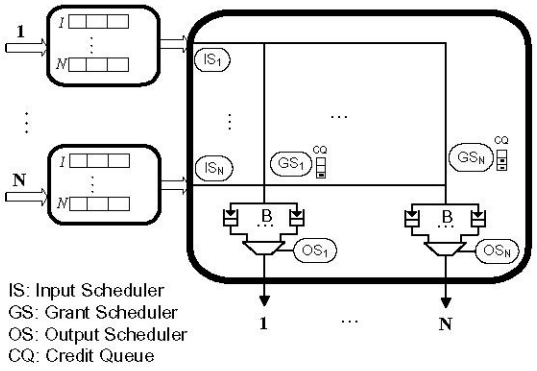

Figure 2. The Hybrid Buffered Crossbar (HBC) Switching Architecture.

(CQ), one per output, to avoid internal buffers overflow. 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.

3. THE PARTIAL ROUND ROBIN ALGORITHM

The scheduling process in the HBC 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 in HBC is similar to the iterative matching performed by unbuffered crossbar scheduling. However, maintaining a small number of internal buffers makes a significant difference. 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 (see Figure 3). Unbuffered crossbar schedulers resort to multiple iterations to improve the match size.

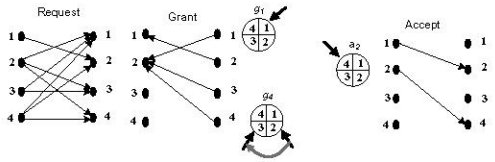

Figure 3. The iSlip Scheduling Algorithm.

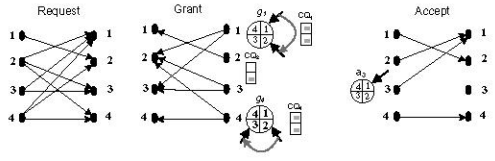

Figure 4. The PRR Scheduling Algorithm.

The HBC input scheduling, while enforcing the input contention constraint, 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 granting to B inputs instead of just one. Figure 4 depicts a PRR input scheduling cycle. As we can see, the grant scheduler at output 1, g_1, sends two grants (to input 2 and 3) because its credit queue, CQ_1, has two available credits. The same process applies to g_3 and g_4. However, g_2 sends only one grant because its associated output buffers have only one location free ($CQ_2 = 1$). Increasing the number of grants per output results in a higher acceptance rate. If we consider an unbuffered crossbar-based random scheduling policy such as PIM [7] with one iteration, the probability that an input will remain ungranted is $(\frac{N-1}{N})^N$, where N is the switch port count [3]. As N increases, this probability tends to $\frac{1}{e}$. If we use the same random scheduling policy in the HBC with B internal buffers per output and assuming that cells 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 $B \geq 4$). Hence, higher acceptance rate and throughput. The PRR algorithm uses fully unsynchronized grant pointers settings, similar to [8]. The Specification the PRR input scheduling is as follows:

PRR:

Grant Phase:

All output pointers, g_j, are initialized to different positions.

For each output, j, do

. 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.
  - Decrement CQ_j by one.

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

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 received grant, g_j, and send the HoL cell of $VOQ_{i,j}$ to the internal buffer.

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

Since the number of internal buffers, B, per output is much smaller than the number of competing inputs, N, one has to be careful on 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 address this, we employed a First-Come-First-Serve (FCFS) output scheduling to ensure in-order cell delivery [9]. A cell departure from the internal buffers at output j, causes CQ_j to increment by one. When a grant scheduler, GS_j, sends a grant to a requesting input, a credit must first be reserved and, consequently, CQ_j is decremented by one.

4. PERFORMANCE ANALYSIS

In this section, we analyze the performance of the HBC 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 [10]. Simulations run for 1 million time slots and statistics are gathered when fourth of the total simulation length had elapsed. We tested different HBC switch sizes, each with different internal buffer sizes [1]. However, due to space limitation, we limited the results to switch sizes of 16×16, 32×32 and 64×64.

We compared the average cell latency of the PRR algorithm for 32×32 HBC 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 uniform Bernoulli and Bursty arrivals. Figure 5-(a) depicts the performance of PRR with different internal buffers, iSlip (with 1 and 4 iterations), RR_OCF and OQ. Irrespective of whether the input traffic is Bernoulli or Bursty, PRR(1) (1 refers to $B = 1$) exhibits similar behavior to 1Slip. The slight better performance of PRR comes from its pointers update mechanism, where they are fully unsynchronized. As B increases, the delay of PRR significantly decreases. It approaches that of an ideal OQ with just 8 internal buffers per output ($B = 8$). Similar performance is observed under bursty arrivals (Figure 5-(b)).

Figure 5-(c) depicts the delay performance of PRR with different numbers of internal buffers (B) and compares it to the iSlip unbuffered algorithm as well as the fully buffered crossbar algorithm (RR_OCF). We can see that, irrespective of the switch size, PRR(4) has the same delay as 4Slip. On the other hand, PRR(8) has a similar delay to that of a fully buffered crossbar. These results suggest that an HBC 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 afford 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 an HBC switch with just 4 internal buffers per output. However, if performance is the main target with little flexibility in cost, one may employ an HBC switch with 8 internal buffers per output, for its high performance.

We studied the stability of a 32×32 HBC switch under unbalanced traffic arrivals. We employed the unbalanced traffic proposed in [10]. Figure 6 depicts the performance of the HBC switch with PRR algorithm and different internal buffer settings and compares it to that of a

[1] Extensive simulations were conducted out for switch sizes of 8×8, 16×16, 32×32 and 64×64. Depending on each HBC switch size, different internal buffer sizes, B, were used (1,2,3,4,5,6,8,10,12,16,32).

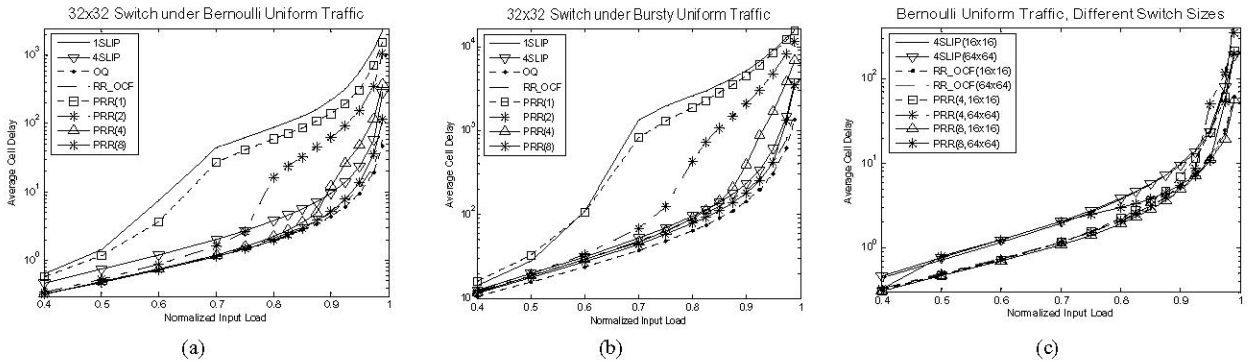

Figure 5. Average Cell Delay Performance under Uniform Arrivals. (a): Bernoulli uniform traffic (b) Bursty uniform traffic (c): Bernoulli uniform with different switch sizes of 16×16 and 64×64.

fully buffered crossbar (RR_OCF). We can see that, with $B = 4$ internal buffers per output, we can achieve comparable throughput to a fully buffered crossbar when the unbalanced coefficient, w, is higher than 0.4. The ideal throughput of the HBC is reached when using 8 internal buffers per output.

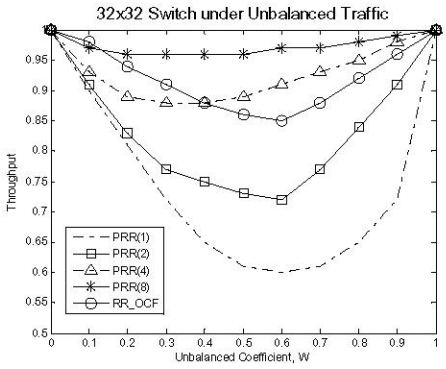

Figure 6. Switch throughput under Unbalanced Traffic

5. CONCLUSIONS

This article proposes a novel switching architecture, namely the Hybrid Buffered Crossbar (HBC). The HBC switch is designed to be a good compromise between unbuffered crossbars and 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 small number of internal buffers per output irrespective of N. On the other hand, it overcomes the multiple iterations required by unbuffered crossbar scheduling algorithms. We proposed a round robin algorithm for the HBC architecture, named the Partial Round Robin (PRR). Simulation results showed that, using 4 internal buffers per output ($B = 4$), the HBC can behave as an unbuffered crossbar scheduling with 4 iterations. Setting B to 8 internal buffers per output results in the HBC approaching the performance of fully buffered crossbars, irrespective of the switch size, N.

6. REFERENCES

[1] McKeown. N., *Scheduling algorithms for input-queued cell switches*, Ph.D. thesis, University of California at Berkeley, May 1995.

[2] A. Mekkittikul, *Scheduling Non-Uniform Traffic In High Speed Packet Switches and Routers*, Ph.D. thesis, Stanford University, Nov 1998.

[3] N. McKeown, "islip scheduling algorithm for input-queued switches," *IEEE Trans. On Networking*, vol. 07, no. 02, pp. 188–201, Apr. 1999.

[4] D.N. Serpanos and P. I. Antoniadis, "Firm: A class of distributed scheduling algorithms for high-speed atm switches with input queues," *IEEE INFOCOM*, March 2000.

[5] S. Chuang, S. Iyer, and N. McKeown, "Practical algorithms for performance guarantees in buffered crossbars," *IEEE INFOCOM*, March 2005.

[6] L. Mhamdi and M. Hamdi, "MCBF: A High-Performance Scheduling Algorithm for Buffered Crossbar Switches," *IEEE Communications Letters*, vol. 07, no. 09, pp. 451–453, Spet. 2003.

[7] T. Anderson, S. Owicki, J. Saxe, and C. Thacker, "High speed switch scheduling for local area networks," *ACM Transactions on Computer Systems*, pp. 319–352, Nov. 1993.

[8] Y. Jiang and M. Hamdi, "A fully desyncronized round-robin matching scheduler for a voq packet switch architecture," *IEEE HPSR*, pp. 407–411, 2001.

[9] Roberto Rojas-Cessa and Ziqian Dong, "Combined Input-Crosspoint Buffered Packet Switch with Flexible Access to Crosspoint Buffers," *IEEE ICCDCS*, Apr. 2006.

[10] R. Rojas-Cessa, Z. Jing E. Oki, and H. J. Chao, "Cixb-1: Combined input one-cell-crosspoint buffered switch," *IEEE HPSR*, pp. 324–329, 2001.