Packet Mode Scheduling in Buffered Crossbar (CICQ) Switches

*Georgios Passas and Manolis Katevenis**

Inst. of Computer Science (ICS), Foundation for Research and Technology - Hellas (FORTH) - member of HiPEAC
ICS-FORTH, P.O. Box 1385, Vassilika Vouton, Heraklion, Crete, GR-711-10 Greece
http://archvlsi.ics.forth.gr/bufxbar/ - {katevenis,passas}@ics.forth.gr

***Abstract*— Buffered crossbars can directly switch variable size packets but they require significant crosspoint buffering to do so, especially when the traffic includes large packets. When we cannot afford large crosspoint buffers we are forced to restrict the maximum internal transfer unit by segmenting packets. Packet segmentation implies a reassembly delay cost which is an issue in systems requiring low latency. We drastically reduce reassembly delay by applying packet mode scheduling to the buffered crossbar architecture. Packet mode scheduling has been studied in input queued switches: when the central switch scheduler establishes a connection from a switch input to a switch output port, it maintains that connection until all the cells of the packet are switched. In buffered crossbars the scheduling is distributed at switch input and output ports, thus the extension is not trivial. We synchronize the input and output port schedulers so as whenever their independent decisions result to an input-output port pairing they maintain that pairing for the lifetime of the packet transmission. Using simulation we study our system's performance. We show that reassembly delay is significantly reduced, especially under light loads.**

I. Introduction

Crossbars are the building blocks for modern switching fabrics and router systems. Traditional crossbars are bufferless –with buffering provided only on the ingress and egress line cards– hence transmissions through the switch have to be synchronized with each other, leading to operation with fixed-size cells (segments). As illustrated in figure 1, variable-size packets are segmented at the inputs (where virtual-output queues (VOQ) reside), the cells are switched through the crossbar, and the packets are reassembled at the outputs (where real-output queues (ROQ) / reassembly buffers reside), before they are transmitted on the line.

If the switch is scheduled ignoring which cell belongs to which packet, *Cell Mode* operation results, as illustrated in figure 1: the cells of a packet (say B) arrive at the egress line card interleaved in time with cells of other packets (A, C) arriving from other inputs. Packet transmission on the output line cannot start until the egress line card is certain of receiving the last cell of the packet. Given that future decisions of the scheduler cannot be predicted, store-and-forward operation is enforced, and cut-through cannot be used.

* The authors are also with the Dept. of Computer Science, University of Crete, Heraklion, Crete, Greece.

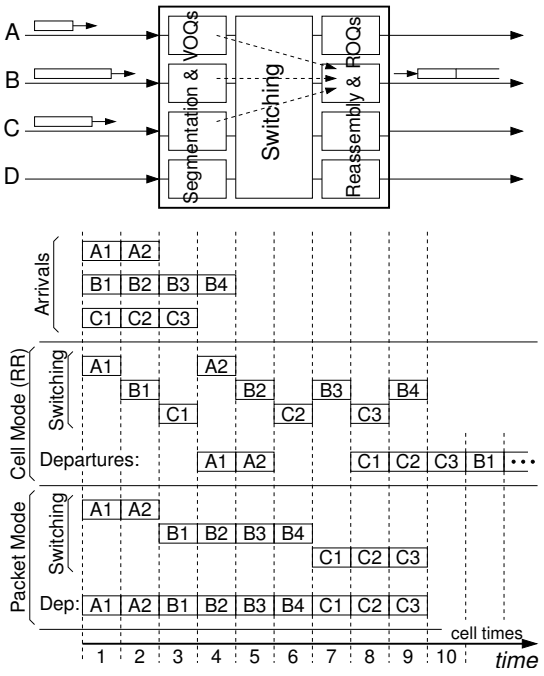

Fig. 1. Packet mode versus cell mode switch scheduling

Packet Mode Scheduling [1], [2], [3] is the alternative illustrated at the bottom of figure 1: when the switch makes a "connection" from an ingress to an egress line card for the first cell of a packet, that connection is maintained until all cells of that packet are switched. Since the egress line card knows that all cells of a packet will arrive consecutively in time, *(a)* it needs no reassembly buffer, and *(b)* it may start transmitting the packet right away, i.e. *cut-through* is allowed. Figure 1 illustrates that, under some circumstances, packet-mode scheduling reduces packet delay[1]; the advantage is particularly important in systems requiring low latency (e.g. multi-processor cluster interconnects), and becomes especially noticed when cut-through is supported and when traffic includes large packets (e.g. jumbo frames).

[1] the timing diagrams of fig. 1 were drawn ignoring the delay (assuming zero delay) of the line card and scheduler logic: a cell can be switched in the same time slot when it arrives; a packet departure may start in the same time slot when its last cell is known to have been scheduled.

To the best of our knowledge, packet mode scheduling has only been studied for bufferless crossbars. Recently, *buffered crossbars* (combined input-crosspoint queueing - CICQ), have emerged as an advantageous architecture; they contain small buffers at their crosspoints, and use backpressure to the ingress line cards to prevent these crosspoint buffers from overflowing. The first observation about buffered crossbars concerned the simplicity and high efficiency of their scheduling [6]-[10]; no internal speedup is needed to compensate for scheduler inefficiencies, thus allowing the increase of port speed. A subsequent observation was that buffered crossbars can directly switch *variable-size* packets [6] [11]. Doing so, without any segmentation, eliminates the need for speedup to cope with cell-padding overhead; in turn, the lack of speedup eliminates egress queueing, and the lack of segmentation eliminates reassembly buffers, thus reducing cost [12].

Buffered crossbars that use no segmentation at all have a limitation: the required buffer size per crosspoint is at least one maximum-size packet plus one round-trip-time (RTT) worth of data [12]. That buffer space becomes expensive in high valency switches (the number of crosspoint buffers equals the square of the port count), and in switches supporting large packets (e.g. jumbo frames).

To overcome that limitation, segmentation and reassembly (SAR) has to be re-introduced into buffered crossbars. However, SAR can be re-introduced in such a way that no padding overhead is incurred (as in bufferless crossbars), hence no internal speedup is needed: in a previous paper [13], we proposed SAR using *variable-size multi-packet segments* to achieve this goal. The maximum segment size can be much smaller than the maximum packet size, thus drastically reducing crosspoint buffer size; the segment size is variable, thus eliminating padding overhead; and multiple (small) packets can be placed in a same segment, thus avoiding small segments so as to reduce relative header overhead. It is a good thing that SAR can be introduced into buffered crossbars without incurring a speedup penalty, but how about the reassembly delay (and preclusion of cut-through) and the reassembly buffer cost that SAR implies?

This paper applies packet-mode scheduling to CICQ switches, so as to reduce the reassembly delay on average, when these switches are forced to use SAR. In particular, cut-through becomes possible, which is especially beneficial for delay under light loads. Packet-mode scheduling in buffered crossbars is not a trivial extension of the bufferless case. Figure 2 illustrates the two kinds of crossbar. In bufferless crossbars (left), there is a central scheduler which determines input-output port pairings (connections). Packet-mode scheduling is simple: once a connection is made, keep it until the last cell of the packet. In buffered crossbars (right), scheduling is distributed and independent: each input selects a non-full crosspoint and sends to it; each output selects a non-empty crosspoint and reads from it. Two or more inputs (1 and 2 in the figure) may send to the same column; two or more outputs (A and B in the figure) may read from the same row. Thus, the scheduling process does *not* necessarily determine input-output port pairings. Packet mode, however, requires such pairings.

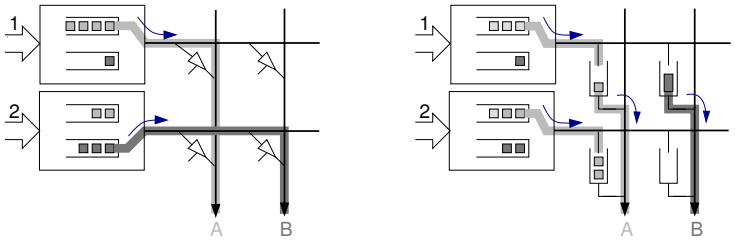

Fig. 2. Bufferless (left) versus buffered (right) crossbar

We perform pairings in a *distributed* way, without any central scheduler: when an output scheduler observes that a pairing is taking place - the enqueue and dequeue signals of the corresponding crosspoint memory are both active - it enters packet mode, until all the segments of the packet are transmitted to the output. Morever, it sends a notification to its input counterpart and commands it to follow on packet mode. The notification is piggybacked to the flow control credit which corresponds to the segment that revealed the occurrence of the pairing. In section III, we present the details of our synchronization protocol. Before that, in section II, we give a precise description of the queueing/scheduling architecture we study. We evaluate our system's performance in section IV using simulation. We show that our method reduces reassembly delay by more than 90% for loads up to 70%. The reduction is between 40 and 80% for the rest range of loads, up to 98%.

Related Work was indicated in the above discussion. Two additional studies on variable size packets in bufferless crossbars are [4], concerning the Alpha 21364 router pipeline and [5], where packet mode scheduling is connected to the provision of QoS guarantees. This is the first study on applying packet mode scheduling on buffered crossbars.

II. Description of the Studied Architecture at the System Level

Figure 3 shows the queueing organization and the placement of the schedulers in the system we study, in a small, for the sake of presentation, configuration with two input and output links. It is a classical CICQ switch architecture with VOQueueing in the ingress datapath, small buffers inside the switch core (one fifo queue at each crosspoint) and small per-input buffers in the egress path.

We consider variable size packets at the switch interfaces and packet segmentation in the ingress path. So, we need egress buffering for packet reassembly. The segmentation method we use is not new, but it is based on our previous work [13] and we prefer it because of its efficiency; we briefly repeat it here for reasons of completeness. We use variable size segments while merging multiple external packets or packet fragments into each segment. The size of the segment is defined by the queue's backlog and the maximum internal transfer unit (S), a system parameter. Whenever, the backlog is greater than S, a chunk of S bytes is candidate for transfer. Whenever the

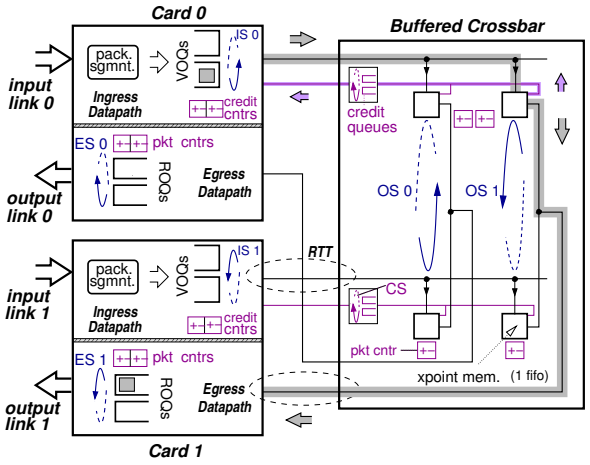

Fig. 3. System Level Architecture

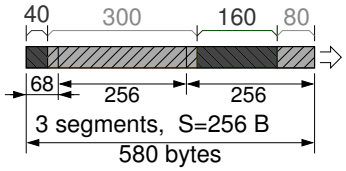

Fig. 4. Segmentation to variable size, multipacket segments

backlog is less than or equal to S, the whole backlog forms a single segment. Thus, the system switches segments ignoring packet boundaries. Segment size variability entails no padding bytes in the segments. The encapsulation of multiple packet fragments into each segment is coherent with the memory management operations when DRAM is used for the buffering at the linecards and, furthermore, it reduces the overhead of the internal headers; DRAM blocks are usually multipacket structures to save throughput and memory space. For more information on how memory management is connected to our segmentation method refer to [13]. Figure 4 shows the resulting segments in an example where the backlog of a VOQ is 580 bytes and S is 256 bytes. Four packets are queued and their sizes are shown in the figure.

An internal switch header is prepended to every segment and comprises three fields: a switch source and destination port id and a *neops* field indicating the number of packets ending inside the segment. In case the external packet headers do not contain information about the packet size, such information must be included in an additional field in the internal headers.

Each crosspoint buffer is at least S bytes large. The egress buffers are one maximum packet long each one. The VOQs are considered to have infinite size.

The input, output and crossbar links run at the same rate. Credit based flow control is used between the inputs and the crosspoints to prevent from crosspoint buffer overflow. The bandwidth of the control lines suffices for the transmission of one credit for the departure of each minimum internal packet segment. Credit queues are used to store credits corresponding to segments which almost simultaneously depart from the same crossbar row and thus the credit bandwidth demand is instantaneously greater than the available.

A scheduler (*IS*) is placed at the ingress path to resolve input contention, at each crossbar output port (*OS*) to resolve output contention and at the egress path (*ES*) to serve contending reassembled packets. Scheduling (*CS*) is also needed at the credit queues. *CS* and *ES* are simple round robin schedulers. The operation of *IS* and *OS* is the subject of this paper and it is determined in section III.

Non zero round-trip time is considered between the cards and the switch core. This time includes packet or credit propagation time, memory accesses, scheduling times etc.

Last, we assume per flow packet and time counters inside the crossbar chip; both of them are needed by the method we describe in section III. The packet counters keep track of the complete packets that are stored in each crosspoint buffer and the time counters indicate the waiting time of the tail segment in each crosspoint buffer. Packet counters are also used in the egress path to indicate the number of the reassembled packets in each Real Output Queue (ROQ). Packet boundaries can be easily recognised by the packet size information contained in the external or internal header. The update of the packet counters is easy considering the *neops* field in the internal headers.

III. Distributed Packet Mode Scheduling

In this section we describe how the input and output schedulers can be synchronized so that packet mode transmissions can be jointly decided. The core idea is that when a pairing (i, j) occurs, both $IS\ i$ and $OS\ j$ decide to maintain it - we say they are synchronized. The packet associated to the pairing is the one at the head of the segment which is being written to output j and the pairing is maintained for as long as that packet's transmission to the output requires. First we probe the "types" of pairings and specify which ones and why are considered to be valid, i.e. result to scheduler synchronization and packet mode transmission. Next we describe how the schedulers actually operate.

A. Input-Output Port Pairings Leading to Synchronization

An (i, j) pairing emerges at the time input i writes to the crosspoint buffer of output j

- *a fragment F of a packet P (or a whole packet P) while output j reads the same fragment F (or whole packet P) (type 1), or*
- *a fragment of a packet P while output j reads a fragment F' of the same packet P (type 2), or*
- *a fragment of a packet P_1 while output j reads from the same crosspoint buffer a fragment of a different packet P_0 (type 3).*

We discard pairings of type (3) because they cause starvation side effects. To see why, suppose that we maintain an (i, j) pairing which results from the write of a segment belonging to packet P_1 and the read of a segment belonging to packet P_0. It is possible that when the transmission of packet P_1 to

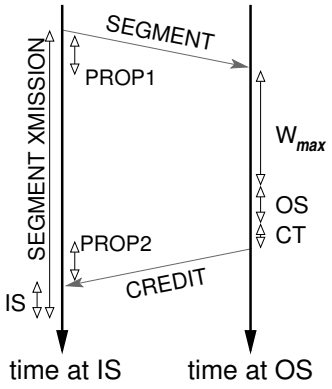

Fig. 5. Maximum allowed time between a read and a write operation to the same corsspoint buffer.

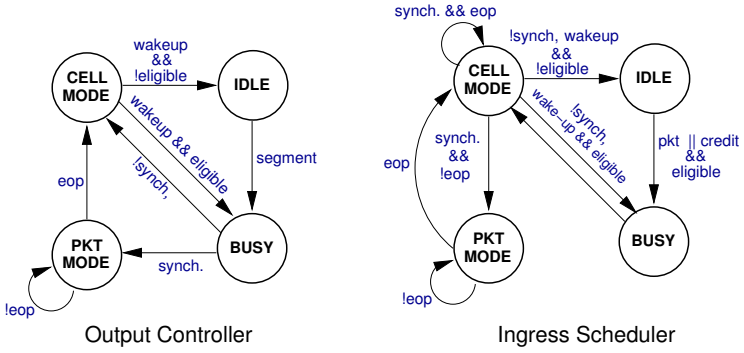

Fig. 6. State machines for IS and OS.

the crossbar is completed, input i starts transmitting a second packet P_2 to output j. In this case a second pairing is possible to emerge during the write of P_2 and the read of P_0 or P_1. Similarly, if input i persists transmitting to output j, output j will keep serving i and the rest of the inputs are going to starve.

Not all pairings of type (1) or (2) are considered valid. Assume that input i writes to output j a segment F_1 of a packet P and output j reads a segment F_0 of the same packet P. We want both IS i and OS j scheduler to become aware of the pairing before they complete the transmission of segments F_0 and F_1 in order to be synchronized. Alternatively, they should have to wait for some time resulting to the insertion of bubbles between the transmission of segments. If one of them, say IS, proceeds serving other flows while OS observing the pairing has already entered packet mode transmission for packet P, buffer underflow occurs.

We define as time window W the time interval between the moment the input starts writing to the crosspoint buffer to the moment that crosspoint buffer starts being read. In the "space-time diagram" of fig. 5 we display the maximum allowed W in order for both the schedulers to observe the pairing in time. The horizontal direction represents space while the vertical one time. The arrows denote messages and the arrow heads events. To save throughput, we start input and output scheduling t_{ST} time before the completion of the segment transmission, where t_{ST} is the scheduling time.

$$W_{max} = t_S - (t_{PROP} + t_{IS} + t_{OS} + t_{CT})$$

t_S stands for the transmission time of the maximum segment. Note that in case the segment size is less than the maximum, no synchronization is needed because the segment already carries the ending bytes of a packet. t_{PROP} is twice the propagation time from the linecards to the crossbar chip. CT stands for the credit transmission time. We need to include this time to account for the delay the credit probably meets at the credit queues. CS always gives priority to credits that are connected to scheduler synchronization, so the delay at credit queueus is at most one credit transmission time.

When $t_S < (t_{PROP} + t_{IS} + t_{OS} + t_{CT})$ none of the pairings is valid, the schedulers are never synchronized and no packet mode transmission is started. In order for the schedulers to be synchronized the *RTT must be smaller than transmission time of a maximum segment*[2]. The latter is a system parameter and it should be set to be greater than the RTT.

B. Scheduler Operation and Synchronization

In fig 6 we show the finite state machines of IS and OS. When all crosspoint buffers are empty, OS remains in the IDLE state. When a segment arrives, scheduling is triggered (BUSY state) and the scheduler decides in a round robin manner which input i to serve. Afterwards, it checks for the occurrence of a valid pairing, which happens when all of the following conditions hold:

- The enqueue signal of buffer i is asserted at the time of check
- The packet counter corresponding to buffer i is zero (this means that the paring has the type (1) or (2) specified in the previous section).
- The waiting time W of the tail segment in buffer i is smaller than the value W_{max} specified in the previous section. The waiting timing is computed by the time counters.

If the above conditions hold, OS transits to packet mode state and keeps transmitting bytes of the same packet. Furthermore, it asserts an ACK flag into the credit corresponding to the first segment transmitted on packet mode and, in the header of that first packet, it asserts a "cut-through flag". The ACK is used to command the input scheduler to follow on packet mode and the cut-through flag is used to notify the egress scheduler that it can start transmitting the packet before it is completely stored. When the last segment of the packet starts being transmitted, OS transits to cell mode.

If the conditions for valid pairings do not hold, OS transmits the current segment (in cell mode state) and wakes up ST time before the completion of the segment transmission in order

[2]For simplicity we assume that the RTT comprises scheduling and propagation times. In a real system more time constants should be included, such as memory access times.

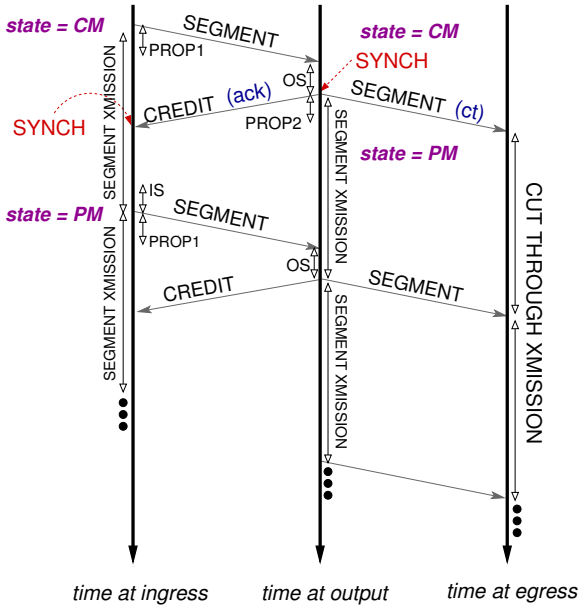

Fig. 7. Scheduler synchronization timing.

to reschedule and serve probably another input, selected in a round robin manner.

IS operates symmetrically. The difference is in the flow eligibility and the way it gets synchronized. Flows are eligible as in the classical crossbar scheduling. IS gets synchronized by receiving an ACK credit when in cell mode.

Figure 7 displays an example of the scheduler synchronization when there is no congestion in the system:

1) Input sends the first segment A of a large packet P.
2) Output serves that segment, after a scheduling delay time and observes a valid pairing. A credit flagged ACK is sent back to the input and a cut-through notification to the output.
3) Input sees the ACK and gets synchronized.
4) The output "sees" the cut-through flag and start transmitting the packet right away.

Note that our connection set up process incurs minimal traffic overheads: one bit per flow control credit. Also note that the traffic is piggybacked on the headers of the flow control traffic.

C. Discussion

There will be cases where the schedulers fail to get synchronized. In these cases, some packets will be transmitted on a cell mode possibly interleaved with segments of other packets from/to other links. Our protocol does not provide deterministic guarantees for packet mode transmission because buffered crossbar scheduling does not necessarily result to input-output port pairings. Our simulations show that under light loads such pairings will almost always happen since congestion is absent. This is important because we care about cut-through transmission especially under light loads for a wide range of traffic patterns. Furthermore, they show that even under heavy loads packet mode connections will occur with a high frequency, resulting to reduced reassembly delay.

Since our proposal does not provide deterministic guarantees for packet mode transmission, egress reassembly buffers must be as large as the respective ones in cell mode scheduling: N maximum packets memory space. In bufferless crossbars [3] only one maximum packet worth memory space is needed for the packet reassembly. On the other hand, our scheduling method is distributed and much more scalable than the central scheduler needed in [3].

A second important advantage of our system compared to [3] is that it can operate with variable size segments and thus no speed up is needed for the segmentation overheads. On the other hand, bufferless crossbars require fixed length cells and segmentation overheads may be high.

IV. Performance Study

A. Method

We developed a byte-time accurate simulator to model the buffered crossbar system described in section II. We model variable size transfer units at the switch interfaces and the core, so in order to keep track of time we follow the discrete event simulation approach [15]. Information relative to the simulator can be found in [12].

The RTT is considered 400 byte times, the default segment size is 512 bytes and the default crosspoint buffer size 512 bytes as well. We experiment with 32 input/output ports.

Packet mode scheduling is compared to cell mode round robin scheduling in buffered crossbars. The delay in the ideal output queueing serves as a lower bound.

The traffic consists of packets with Poisson arrivals and bimodal packet size. 95% of the packets are 40 bytes long and 5% of the packets 10 Kbytes. We use that large packets to model switching of jumbo frames. Switch destination ports are uniformly selected.

We measure the mean delay of the large packets at the queues of our system as the time interval between their first byte arriving to the system to the their first byte departing from it, after subtracting constant delays such as propagation and scheduling times. The reported values are multiples of the transmition time of a maximum segment.

B. Numerical Results

In fig. 8 we show the mean queueing delay at reassembly buffers (ROQs) for our system ("PM_baseline") and the relative one with classical cell mode round robin schedulers ("CM"). For light loads *PM_baseline* presents zero queueing delay (at ROQs) because almost always cut-through transmission is achieved. On the other hand *CM* presents one packet store & forward delay (20 segments). Delay at ROQs increases in *PM_baseline* as loads become heavier because due to congestion cut-through cannot start from the starting bytes of the packets and some segments are transmitted on cell mode. In the *CM* system ROQ delay increases because the interleaving of packets increases with load. For a load of 98% the ROQ delay is reduced by 30% when using packet mode scheduling.

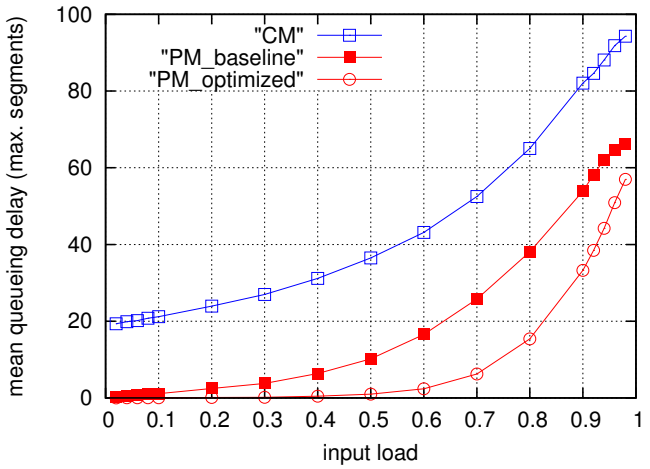

Fig. 8. Queueing delay at ROQs for 10KB packets. Poisson packet arrivals, uniform destinations and bimodal packet size is assumed.

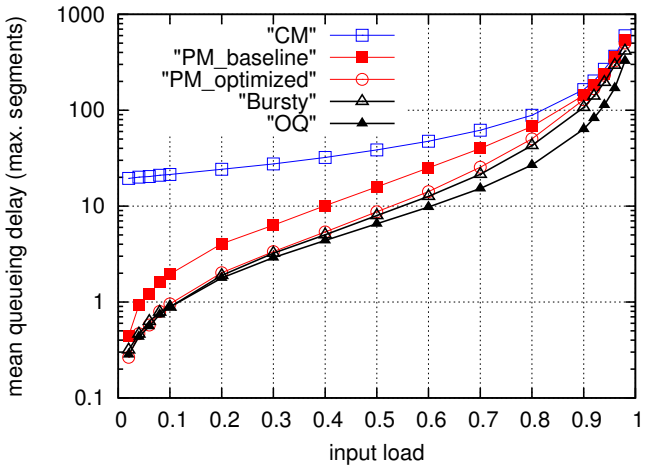

Fig. 9. Total queueing delay for 10KB packets. Poisson packet arrivals, uniform destinations and bimodal packet size is assumed.

The curve *PM_optimized* shows the ROQ delay in our system when the schedulers described in section III are slightly modified: when synchronization fails, instead of polling the next crosspoint buffers in the round robin schedule, the output scheduler revisits the same crosspoint buffer, to check if it is not empty, i.e. to check for a possible respond of the input. In that case it serves the same input and gets synchronized with it. For a load of 0.8 the improvement is around 60%.

Figure 9 displays the total queueing delay: at VOQs, crosspoint buffers and ROQs. The output queueing system has been included in this plot; we assume cut-through at the output buffers. The curve labelled "bursty" shows the queueing delay in out system when the 10K byte streams form 20 discrete packets - with size equal to the segment size - instead of a single one. Under this traffic we have the same burtsiness compared to the traffic described in the previous section and thus the delay at VOQs is identical. However the reassembly delay is almost zero. This curve shows what is the best we can do with packet mode scheduling. We observe that the *PM_baseline* indeed improves the total delay compared to *CM* and with the aforementioned optimization achieves almost optimum performance.

Ongoing work:
Simulations with unbalanced traffic - based on the pattern defined in [8] - are running. Preliminary results suggest a worst-case throughput of around 85% when using one maximum segment crosspoint buffers.

Conclusion

We presented a novel scheduling method for buffered crossbar switches based on the application of packet mode scheduling. We described a method to synchronize the arbiters which are distributed at the switch input and output ports. Using simulation we concluded that with the cut-through opportunity, which is offered by our scheduling method, the queueing delay is significantly reduced at a great range of loads. For loads up to 70% the reassembly delay is reduced by more than 90%. For greater loads the improvement is between 40 and 80%.

References

[1] Sung-Ho Moon, Dan Keun Sung: "High-performance variable-length packet scheduling algorithm for IP traffic", *GLOBECOM 2001*, no. 1, Nov 2001 pp. 2666-2670
[2] M. Ajmone Marsan, A. Bianco, P. Giaccone, E. Leonardi, F. Neri, "Packet scheduling in input-queued cell-based switches", *IEEE INFOCOM 2001*, no. 1, April 2001 pp. 1085-1094
[3] M. Ajmone Marsan, A. Bianco, P. Giaccone, E. Leonardi, F. Neri: "Packet-Mode Scheduling in Input-Queued Cell-Based Switches", *IEEE/ACM Tr. on Networking*, vol. 10, no. 5, October 2002, pp. 666-678.
[4] S. Mukherjee, F. Silla, P. Bannon, J. Emer, S. Lang, D. Webb: "A Comparative Study of Arbitration Algorithms for the Alpha 21364 Pipelined Router", *Proc. of the ACM ASPLOS-X Conf.*, San Jose, CA USA, Oct. 2002, pp. 223-234.
[5] X. Zhang, L. Bhuyan: "Deficit Round Robin Scheduling for Input-queued Switches" , *IEEE Journal of Selected Areas in Communications*, May 2003, pp. 584-594.
[6] D. Stephens, H. Zhang: "Implementing Distributed Packet Fair Queueing in a scalable switch architecture", *Proc. INFOCOM'98 Conf.*, San Francisco, CA, March 1998, pp. 282-290.
[7] R. Rojas-Cessa, E. Oki, and H. Jonathan Chao: "CIXOB-k: Combined Input-Crosspoint-Output Buffered Switch", *Proc. IEEE GLOBECOM*, 2001, vol. 4, pp. 2654-2660.
[8] F. Abel, C. Minkenberg, R. Luijten, M. Gusat, I. Iliadis: "A Four-Terabit Packet Switch Supporting Long Round-Trip Times", *Proc. IEEE Micro Magazine*, vol. 23, no. 1, Jan./Feb. 2003, pp. 10-24.
[9] N. Chrysos, M. Katevenis: "Weighted Fairness in Buffered Crossbar Scheduling", *Proc. IEEE Workshop on High Performance Switching and Routing (HPSR 2003)*, Torino, Italy, June 2003, pp. 17-22.
[10] K. Yoshigoe, K. Christensen: "A Parallel-Polled Virtual Output Queued Switch with a Buffered Crossbar", *Proc. IEEE Workshop High Perf. Switching & Routing (HPSR 2001)*, Dallas, TX, USA, May 2001, pp. 271-275; http://www.csee.usf.edu/∼christen/hpsr01.pdf
[11] M. Katevenis, G. Passas, D. Simos, I. Papaefstathiou, N. Chrysos: "Variable Packet Size Buffered Crossbar (CICQ) Switches", *Proc. IEEE International Conference on Communications (ICC 2004)*, Paris, France, 20-24 June 2004, vol. 2, pp. 1090-1096.
[12] M. Katevenis, G. Passas: "Variable-Size Multipacket Segments in Buffered Crossbar (CICQ) Architectures", Proc. IEEE International Conference on Communications (ICC 2005), Seoul, Korea, 16-20 May 2005, CR-ROM paper ID "09GC08-4", 6 pages.
[13] J. Postel, "RFC 793: TRansmission Control Prtotocol", Sept. 1981.
[14] Sheldon M. Ross: "Simulation", Academic Press, 3rd Edition, 2001, ISBN 0125980531