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/ – {passas, katevenis}@ics.forth.gr

***Abstract*— Buffered crossbars have emerged as an advantageous switch architecture mainly due to their scheduling efficiency and capacity to operate directly on variable size packets. Such operation requires crosspoint buffers at least as large as one maximum packet each. When we cannot afford that large crosspoint buffers, we are forced to segment packets. Although variable-size segments can be used to avoid padding overheads, we are still left with the cost of reassembly buffers and the associated delays. This paper applies *packet mode scheduling* to buffered crossbars in order to remedy these shortcomings: the segments of a variable-size packet are switched consecutively in time. We propose two scheduling schemes: *probabilistic* and *deterministic* packet mode scheduling. The probabilistic case allows cut-through forwarding and operates with independent crossbar output schedulers, but it requires reassembly buffers. Deterministic scheduling sacrifices some scheduler independence in order to eliminate the reassembly buffers. Using simulation we show that it performs very close to buffered crossbars with no segmentation and large buffers at the crosspoints.**

1. Introduction

Crossbars are the building blocks for modern switching fabric 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 segments, called *cells*. As illustrated in Fig. 1, variable-size packets are segmented at the inputs (where virtual-output queues, VOQs, reside), the cells are switched through the crossbar, and the packets are reassembled at the outputs (where real-output queues, ROQs, and 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 the first part of the figure: 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 scheduler decisions cannot be predicted, reassembly buffers are required and *store-and-forward* operation is enforced.

Packet Mode Scheduling [1] [2] (see also [3] [4] [5]) is the alternative illustrated in the second part of the figure: 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 the 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. Fig. 1 illustrates[1] that packet mode scheduling reduces packet delay. The advantages are particularly important in systems requiring low latency (e.g. multi-processor cluster interconnects), and when traffic includes large packets (e.g. jumbo frames [6]).

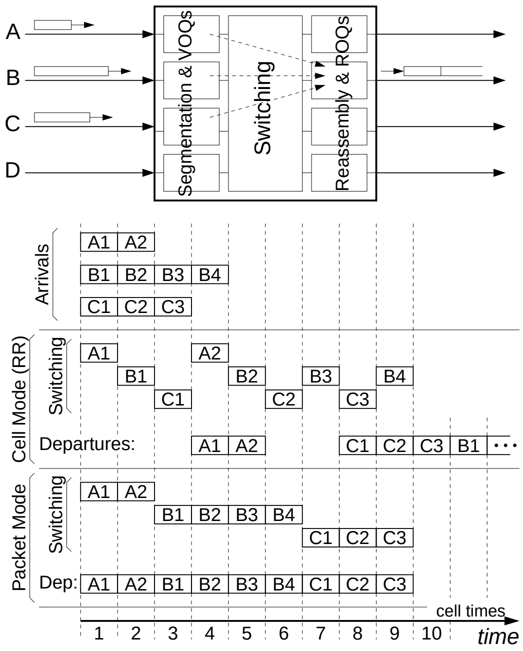


Fig. 1. Packet mode versus cell mode switch scheduling

Recently, *buffered crossbars* (combined input-crosspoint queueing, CICQ) have emerged as an advantageous switch 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

[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.

about buffered crossbars concerned the simplicity and high efficiency of their scheduling [7] [8] [9] [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 [7] [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 have a limitation: the required buffer size per crosspoint is at least one maximum-size external packet [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 this limitation, segmentation and reassembly (SAR) has to be re-introduced in buffered crossbars. However, buffered crossbars can switch variable-size segments, thus padding overheads are avoided and no internal speedup is needed for this purpose [13]. It is a good thing that SAR can be introduced in 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 work resolves these issues applying packet mode scheduling to buffered crossbars. The application is not a trivial extension of the bufferless case. Fig. 2 illustrates the two kinds of crossbar. In bufferless crossbars (left), a central scheduler 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 (A and B in the figure) may send to the same column; two or more outputs (1 and 2 in the figure) may read from the same row. Thus, the scheduling process does *not* necessarily determine input-output pairings.

We describe two scheduling schemes. The first one allows for distributed and independent schedulers, associated to switch input and output ports, as in the classical buffered crossbar architecture; whenever the independent decisions of the schedulers pair an input to an output port, the schedulers synchronize in order to maintain the pairing for the lifetime of the corresponding packet transmission. We call this first scheme *probabilistic packet mode scheduling* and we describe it in Section 3. Section 4 describes our second method, *deterministic packet mode scheduling*, which reduces scheduler independence in order to obtain determinism and hence eliminates reassembly buffers.

Section 5 presents our simulation results, evaluating the performance of our schemes and comparing them to previous systems. We show that deterministic scheduling achieves performance very close to buffered crossbars with very large crosspoint buffers and no SAR. Furthermore, it performs better than packet mode scheduling in bufferless crossbars, even if we factor-out the padding overheads imposed by the bufferless architecture. Probabilistic packet mode scheduling performs even better than the deterministic scheme for some traffic patterns, while it performs similar for the rest.

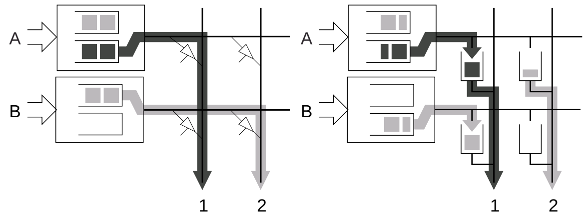


Fig. 2. Bufferless (left) versus buffered crossbar (right)

2. System Architecture

We study a classical buffered crossbar switch with VOQs in the ingress line cards and one, small FIFO buffer at each crosspoint in the switch core (see Fig. 3). External packets are converted into segments in the ingress path; a packet with size P is segmented to $\lfloor P/S \rfloor$ internal segments of fixed size S, and one last, variable-size segment of size $P - \lfloor P/S \rfloor \times S$ and without any padding bytes. The resulting segments are switched through the buffered crossbar.

Schedulers are placed *(i)* on the ingress path of each line card (IS_i), to resolve input contention; *(ii)* at each output port of the crossbar (OS_j), to resolve output contention; and *(iii)* on the egress path of each line card, to serve contending reassembled packets when packet reassembly is required. Egress schedulers are simple round-robin schedulers. The operation of IS_i and OS_j is the subject of this paper, discussed in Sections 3 and 4.

Credit flow control is used between the line cards and the fabric in order to prevent the crosspoint buffers from overflowing. Control lines, separate from data lines, run from the crossbar to the line cards carrying the flow control credits. The bandwidth of each control line suffices for the transmission of one credit during each minimum-external-packet time. Credit queues buffer credits which result from segments that depart at about the same time from the same crossbar row; contenting credits are served round-robin.

Each crosspoint buffer should be at least one maximum internal segment plus one backpressure RTT window large, as shown in [12]. A RTT value around 400ns at 10 Gbps line rate, including scheduling times, propagation times, SERDES delays, etc, is reasonable in realistic systems [12] [13].

We *do not* use internal speedup (core overspeed), since we do not need it to achieve top performance; we consider this to be a key feature of buffered crossbars. Internal speedup increases power consumption, thus limiting port and line-rate scalability. Hence, we always consider that the external and the crossbar links run at the same rate.

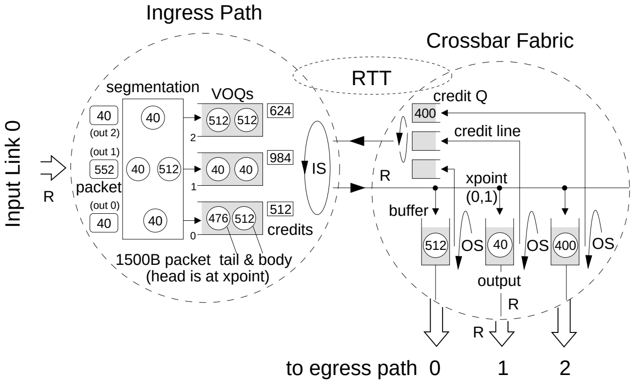


Fig. 3. System architecture studied in the paper. The figure assumes a 3 × 3 switch with 512-byte maximum internal segments, RTT below 512 byte times and 1-Kbyte crosspoint buffers. We only show one line card and its corresponding part in the switch core.

3. Probabilistic Packet Mode Scheduling

3.1. Scheduling Operation

We assume independent round-robin input and output schedulers which serve flows at segment granularity. In the resulting scheduling process, we define that an (i, j) pairing occurs when a maximum segment of a packet P is being written to crosspoint buffer (i, j) by input i while the same or a previous segment of P is concurrently being read by output j from the same crosspoint buffer. When such a pairing occurs, under some appropriate timing constraints, input scheduler IS_i and output scheduler OS_j will keep serving the same flow f_{ij} until the whole packet P has been completely forwarded to the crossbar fabric and to the egress card respectively. We will say that the schedulers operate in *packet mode* for packet P of flow f_{ij}.

OS_j can infer a pairing at the time of its occurrence; it just needs to observe both the read and write enable signals of the crosspoint buffer, while also checking that the same packet is being read and written. Typically, an input scheduler cannot observe a pairing sooner than half RTT from the moment it occurs; although the input sees the output decisions, via the returned flow control credits, for simplicity we assume that it is notified of a pairing by receiving a *special notification* from the crossbar.

We claim that if an (i, j) pairing occurs while the crosspoint reads a (max) segment s_k and writes a (max) segment s_{k+m} of a packet P, it is safe for OS_j to enter packet mode scheduling for P if

$$t_r - t_w \leq \Delta t_s - (t_{IS} + 2 \times PROP), \tag{1}$$

where t_r denotes the time s_k starts being read from buffer (i, j) and t_w denotes the time s_{k+m} starts being written to buffer (i, j); Δt_s is the maximum segment time, t_{IS} is the input scheduling time and $PROP$ is the signal propagation time[2]. We name the time interval $t_r - t_w$ *synchronization distance* (SD). The space-time diagram of Fig. 4 displays the maximum allowable value for a packet mode transmission to take place. If $SD \leq SD_{max}$, the pairing notification reaches input i before IS_i makes the next scheduling decision, thus IS_i synchronizes *in time*. On the other hand, if $SD > SD_{max}$, the notification arrives at the ingress path while IS_i has probably started serving other flows. Notice that when the segment size is smaller than the maximum, it contains the entire tail of the packet, and thus OS_j can enter packet mode scheduling independent of IS_i. Also note that our method works only as long as $t_{IS} + 2 \times PROP < \Delta t_s$.

Summarizing, in the proposed scheme the input and output schedulers toggle between two modes. In *segment mode*, they serve flows round-robin on a per-segment basis. In *packet mode*, they keep serving the same flow until the entire packet has been forwarded. An output scheduler transits from segment to packet mode, when it observes a pairing and the synchronization distance is smaller than the maximum; at the time it transits to packet mode, it sends a notification to the input scheduler it synchronized to and to the egress path. An input scheduler transits from segment to packet mode when it receives a pairing notification from an output scheduler. It is guaranteed that at most one pairing notification arrives at an input when the scheduler is in segment mode, and no pairing notification arrives while it is in packet mode. Both input and output schedulers transit from packet to segment mode scheduling when the current packet has been totally forwarded from the input and the crossbar output respectively. A scheduler at the egress path can start transmitting a packet on the line when the last segment of the packet starts arriving at the egress path or when a packet mode notification for this packet is received.

3.2. Discussion

Under light load, when input and output contention is rare, input-output pairings occur with great probability in (buffered) crossbar scheduling, thus packet mode transmissions are very likely to occur. As the load increases and flows become congested, the schedulers are unlikely to synchronize and our method's effectiveness degrades. Simulation results, presented in Section 5.2, confirmed these hypotheses.

When the schedulers do not synchronize, packets are trans-

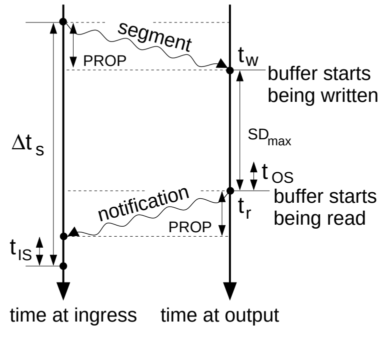


Fig. 4. Maximum allowable synchronization distance

[2]Additional delays, such as memory access and SERDES delays, are not reported for the shake of presentation; consider that they are included in $PROP$.

mitted in segment mode, possibly interleaved with segments of other packets from/to other links. Thus, per-input reassembly buffers are needed at the egress path to collect these segments. Since no guarantee for packet mode transmission is provided, each reassembly buffer needs to be large enough to store a maximum packet.

The first contribution of the proposed scheme is the offered opportunity for cut-through transmission; cut-through is beneficial under light loads, since it greatly reduces packet latency. Second, at heavier loads, our method constricts the packet interleaving in the switch core and thus also reduces the occupancy of the reassembly buffers. Hence, at heavier loads probabilistic packet mode scheduling reduces total queueing delay by reducing the queueing delay at the egress path (Section 5.2).

4. Deterministic Packet Mode Scheduling

4.1. Scheduling Operation

The protocol described in Section 3 preserves scheduling independence at the expense of egress reassembly buffers. Two or more outputs may start reading from the buffers of the same input (at the same crossbar row), even if partly stored packets reside there. Then, it is impossible to set-up packet mode transmissions between that input and all of these outputs, and thus, some (parts of) packets are transmitted in segment mode.

Let ΔP denote the part of the packet stored at crosspoint buffer (i,j), R the line rate, and Δt the time interval within which the remaining part of the packet is known to start arriving at crosspoint (i,j). In order for all packets to be transmitted in packet mode, output scheduler OS_j should start serving a flow f_{ij} only when it is guaranteed that $\Delta t \times R \leq \Delta P$.

To guarantee that always $\Delta t \times R \leq \Delta P$, the buffered crossbar is scheduled as follows, ensuring that all three points are adhered to:

1. A flow f_{ij} is eligible for OS_j if and only if a packet is completely stored in the crosspoint buffer (i,j), or $\Delta P \geq (RTT + \Delta t_s) \times R$; Δt_s represents the max-size segment time. OS_j serves the eligible flows round-robin and when it starts the transmission of a packet whose tail is pending at the ingress path, it requests synchronization with the corresponding input asserting a flag in the first flow control credit released during the packet mode transmission. A credit is released each time a segment starts departing from the crossbar output and corresponds to the size of that segment.
2. Before OS_j starts transmitting a partly stored packet at a crosspoint (i,j), it has to acquire a *lock* associated with input i (see Fig. 5). There is one lock per input, and locks are shared among all output schedulers. If lock acquisition fails, because that input is currently connected to another output, OS_j proceeds serving the next eligible flow in the round-robin schedule. Notice that lock acquisition is an operation entirely *internal* to the buffered crossbar chip, and does not involve any transaction with the ingress line cards. As long as the lock for input i has been acquired by OS_j, all flows $f_{ik}, k \neq j$, having a partly stored packet at crosspoint (i,k) are ineligible until the lock is released. A lock is released when the last segment of the packet starts arriving at the crosspoint.
3. When an ingress line card i receives a synchronization request from output j (as indicated by the respective bit in the credit signal), IS_i synchronizes with OS_j (i.e. enters packet mode scheduling for output j) right after its current segment transmission, if there is one, or right away if it is idle. If no synchronization request is received, IS_i serves flows round-robin on a per-segment basis.

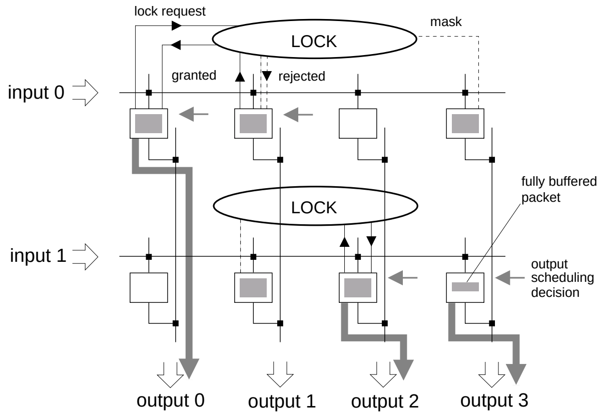


Fig. 5. Proposed (asynchronous) buffered crossbar scheduling. The lock acquisition phase is involved only when an output scheduler attempts to serve a flow whose head packet is partly stored in the crosspoint buffer. A masking operation is used to mark flows as ineligible. Two inputs and four outputs are shown in the figure.

With constraint (1), we guarantee that when an input receives a synchronization request, it may delay honoring it for a max-segment time, in case it is busy transmitting a (maximum) segment to another crosspoint. With constraint (2) we guarantee that no additional requests arrive at an input while that input is paired to another output.

In case RTT is large, it is possible that an input has finished the transmission of the last segment of the packet when it receives the synchronization request for that packet. To avoid synchronizing for different packet transmissions, when an input scheduler receives a synchronization credit corresponding to a packet already departed from the VOQs, it simply discards the synchronization request[3].

4.2. Comparison to Bufferless Crossbar Scheduling

At first glance, deterministic packet mode scheduling resembles the classical request-grant-accept scheduling algorithms for bufferless crossbars. Resemblance concerns "large packet" transmissions: the transmission of packet segments to the crossbar core is analogous to the request phase for packet mode transmissions; output scheduling corresponds to the

[3]Packets and credits should carry ids for this to be possible.

grant phase; lock acquisition is the equivalent of the accept phase.

However, our three-phase matching process yields asynchronous operation, contrary to the synchronous operation of the bufferless crossbar architecture. Concerning transmissions of packets that can be totally buffered at the crosspoints, scheduling reduces to (asynchronous) buffered crossbar scheduling [12] [13]. When large packets are involved, inputs need not synchronize before transmitting to the crossbar because buffers at the crosspoints absorb temporary output conflicts; inputs synchronize in the long-run by the scheduling and flow control protocol. Furthermore, outputs need not synchronize with inputs before they start transmitting, again because there is a buffered segment to read until the input eventually synchronizes. On the other hand, outputs need to be coordinated in order to eliminate input contention when large packets are involved, but this does not imply synchronous operation. As a result, our scheme can switch variable size packets, with fine-grained packet size[4]. By contrast, scheduling in bufferless crossbars assumes cell granularity for packet size –a typical cell-size is 64 or 128 Bytes– incurring padding overheads and requiring internal speedup –by a factor of 2 in the worst case[5].

4.3. Locked Configurations

Packet mode scheduling in bufferless crossbars may "lock" the switch in a fixed configuration because "exact" pairings *at all times* are required. Consider a heavily loaded switch where all input ports are currently connected to one output each, and conversely all output ports are being fed by an input each. Connections are held for an entire packet duration. Consider a case where, when one of the connections is terminated –because the corresponding packet has been delivered in its entirety– *no other* connection happens to have terminated at the same time. Then, there is only a *single input* and a *single output* port that have become available for new pairing(s), hence the scheduler is forced to again pair the same input to the same output. Fig. 6 shows an example of such a traffic pattern. Under such traffic, the scheduler is forced to maintain the crossbar configuration locked into a fixed set of connections (flows $f_{00}, f_{11}, f_{22}, f_{33}$ in the figure), thus starving all other flows.

Although with our proposal a packet-mode pairing between input i and output j may be configured at the time a segment is being transmitted from input i to a different output k –i.e. a (i, k) segment-mode pairing may exist concurrently with a (i, j) packet-mode pairing– packet-mode pairings are still exact –i.e. a (i, k) packet-mode pairing cannot exist concurrently with a (i, j) one. Thus, the problem remains in our method. By contrast, buffered crossbars with maximum-packet crosspoint buffers [12] allow *temporary* situations of "inexact" pairings, i.e. times when multiple inputs forward packets to a same output or multiple outputs read packets

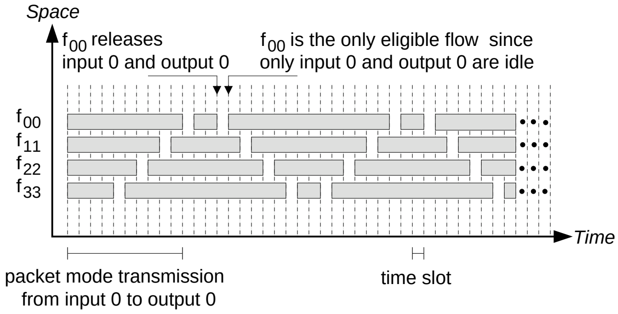


Fig. 6. A staircase-like traffic pattern that leads a packet mode scheduler for a bufferless crossbar to "lock" in a fixed configuration. Assume a 4×4 switch; the figure shows the 4 flows that are constantly served; other flows with non-empty queues exist, but are never served.

from crosspoint buffers that had been fed by a same input at different times in the past. These periods of inexact input-output pairings, allow buffered crossbars with one maximum packet worth crosspoint buffers to escape from the above locked configurations.

5. Performance Study

5.1. Method

We developed a byte-time-accurate simulator to model the buffered crossbar switch. It handles variable size packets at the switch interfaces and variable size segments in the core; it keeps track of time using the discrete event simulation approach [14]. We modelled the probabilistic scheme, labelled *PPM* below, and the deterministic scheme, labelled *DPM*. We compare *PPM* and *DPM* to buffered crossbars with full, variable-size packets (no SAR, round-robin scheduling on a per-packet basis) [12], labelled *VPS*, and to buffered crossbars with SAR and plain segment mode scheduling (round-robin on a per-segment basis, blind of packet boundaries), labelled *SM*.

For all buffered crossbar models, RTT always equals 500 byte times. For the models that use packet segmentation the maximum segment size is 512 Bytes and the crosspoint buffers worth 1 KByte each (one maximum segment plus one RTT window). *VPS* uses 9-Kbyte buffers (one full external packet plus one RTT window). A scheduling operation is initiated a scheduling time before the completion of the current transmission and the scheduling time equals 18 byte times. In *DPM* we assume that the time to acquire a lock equals the output scheduling time and locks are granted to requesting outputs round-robin; output scheduling starts two scheduling times before the completion of the current transmission. We experimented with 16×16 switches.

We also wrote a slotted-time simulator for the bufferless crossbar architecture with input queueing (VOQs) and a central scheduler. The *iSLIP* scheduling algorithm (*IQ-CM*) [15] and its packet mode modification (*IQ-PM*) from [3] were simulated. We assumed 64-byte cells, one scheduling iteration, and a 16×16 switch.

For the buffered crossbar we considered minimum packet size 40 Bytes and maximum 8 KBytes with 1-byte packet size

[4]Size granularity equals to the crossbar datapath width (4 Bytes in [12]).
[5]E.g. with 65-byte packets in a 64-byte-cell switch.

granularity. For the bufferless crossbar we consider minimum packet size 64 Bytes (1 cell) and maximum 8 KBytes (128 cells), with one-cell packet size granularity. Although this integer-cell-size granularity favors *IQ* (padding overhead is a serious disadvantage of *IQ* - Section 4.2), we made this assumption in order to compare pure scheduling efficiency, factoring out padding overheads. Three packet size distributions were used:

- *bimodal* - 95% of the packets are minimum sized and 5% are maximum sized.
- *uniform* - packet size is uniformly distributed in the range between the minimum and the maximum size.
- *constant* - all packets have the maximum size.

For the buffered crossbar we assumed Poisson packet arrivals, while for the bufferless crossbar we modelled packets as bursts of cells, following the same approach as [2]. We chose the traffic models such that they offer insight on the scheduler performance under some "clear and extreme" traffic circumstances, rather than using just a single, "real life" traffic model. The maximum packets were 8 KBytes large in order to show that our methods efficiently switch very large packets. In sections 5.2, 5.3 we assume uniform destinations. In Section 5.4 we present results for non-uniform traffic.

The total queueing delay of a packet in the switch was computed as the time interval between the first byte of the packet arriving to a VOQ and its first byte departing from the reassembly buffer, when the system requires packet reassembly, or departing from the crossbar output port, when it does not[6]. The delay of a packet at the egress path was defined as the time interval between the first byte of the packet arriving and its first byte departing from the reassembly buffer. Constant delays, such as propagation and scheduling times, were subtracted. The delay was averaged over the number of packets with each packet delay contributing the same portion to the average. The reported delay values are in units of 512 byte times (the transmission time of a 512-byte segment). The simulations were run long enough to assure a confidence interval better than 7% with confidence greater than 95%.

5.2. Probabilistic packet mode scheduling (PPM)

We first compare *PPM* to *SM* and report the results for bimodal packet size. With *SM*, the large packets were delayed in the reassembly buffers for around one packet store time (16 segment times) and for loads up to 0.2. In the rest of the load range, this delay increased due to the packet interleaving in the switch core. With *PPM*, the delay of the large packets in the egress buffers was almost zero for loads up to 0.5, because cut-through transmissions were possible at the egress path and the packet interleaving was limited. *PPM* becomes less efficient for higher loads: egress delay reduction, compared to *SM*, goes down from almost 100% for light loads, to 80% for loads around 0.7, and to only 30% for a load of 0.95. The delay of small packets was almost the same in both systems. Fig. 7 shows average total delay of large packets; *VPS* has also been included in the plot as the reference system. Under uniform and constant packet size, similar results were seen with *PPM* being slightly worse as average packet size increases.

[6] We made this assumption in order for the packet delay to be independent of packet size.

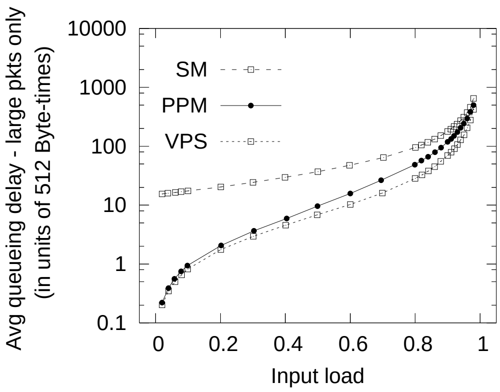


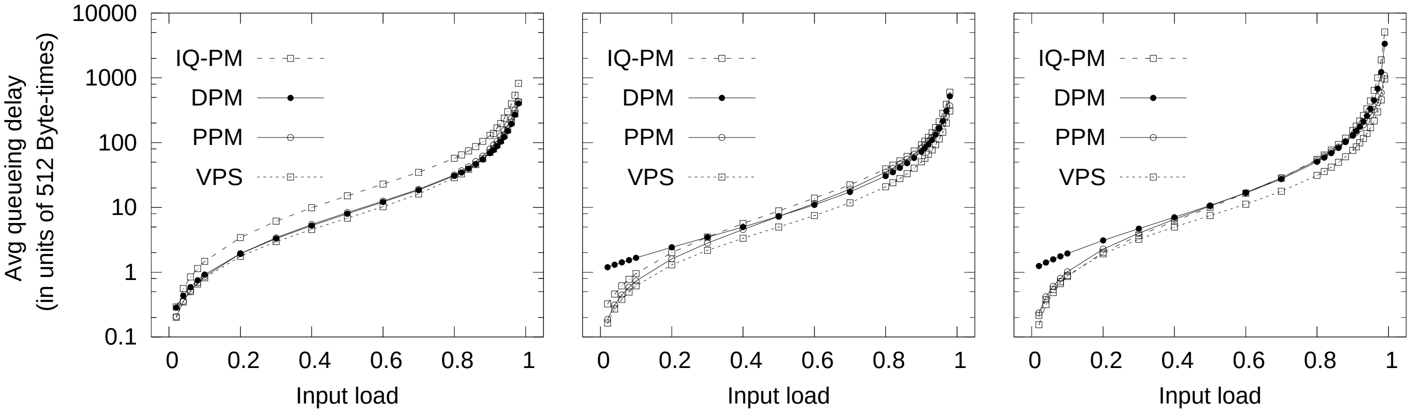
Fig. 7. Performance of probabilistic packet mode scheduling (PPM) compared to segment mode scheduling (SM). The traffic is uniformly destined and the packet size distribution is bimodal.

5.3. Deterministic packet mode scheduling (DPM)

Fig. 8 compares *DPM* to *PPM*, *VPS* and *IQ-PM* assuming all of the three packet size distributions.

1) Comparison to PPM : *DPM* imposes a crosspoint buffer delay which is reflected on the average delay metric for light loads (up to 0.3) and when the average packet size is large (uniform and constant packet size); if desired, one can patch-up this inefficiency by combining *DPM* with *PPM*. For loads between 0.4 and 0.7, average total delay is almost the same for both systems and for all of the packet size distributions. A trade-off appears under heavier loads: with *PPM* packets suffer a delay at the egress path while with *DPM* the delay in the system increases because the matching capabilities of buffered crossbar scheduling are restricted. As a result, under bimodal packet size *DPM* total delay is smaller by at most 20% for loads between 0.85 and 0.97 while at a load of 0.98 *PPM* delay matches *DPM* delay. Under uniform packet size *DPM* and *PPM* are very close for loads up to 0.95 while for greater loads PPM becomes better by at most 30%. Under constant packet size and for loads up to 0.93, total packet delay is the same for both *DPM* and *PPM*; for heavier loads *PPM* becomes better up to 50%. Note that the superiority of *PPM* becomes greater and appears at a lighter load as the average packet size increases. The results confirm the cost of the lock acquisition phase in *DPM*, which is more frequently involved as the percentage of large packets in the traffic mixture increases, restricting the matching capabilities of buffered crossbar scheduling.

2) Comparison to VPS: When packet size is bimodal, the difference between *VPS* and *DPM* is withing the range of statistical error. This is an encouraging result since realistic traffic patterns are usually a mixture of small (control) and large (data) packets [16]. On the other hand, for uniform and constant packet size, *VPS* is better by around 30% and 40% respectively for loads between 0.4 and 0.95; for a load of 0.98 *VPS* becomes better by 40% under uniform packet size and



(a) Bimodal packet size (b) Uniform packet size (c) Constant packet size

Fig. 8. Performance of deterministic packet mode scheduling (DPM) compared to probabilistic packet mode scheduling (PPM), round-robin scheduling in buffered crossbars with full size packets (VPS) and packet mode scheduling in bufferless crossbars (IQ-PM). The traffic is uniformly destined.

60% under constant packet size. Note that *VPS* uses 9-Kbyte crosspoint buffers -which is always greater than the average packet (burst) size- while *DPM* uses only 1-KByte buffers.

3) Comparison to IQ-PM: Last, we compare packet mode scheduling in buffered to bufferless crossbars. When packet size is bimodal, packet delay in the buffered crossbar switch is almost half the delay of the bufferless case for loads between 0.4 and 0.98. For uniform packet size the difference drops to 20% for loads up to 0.98. *DPM* comes closer to input queueing when packet size is constant.

5.4. Non-uniform Traffic

We experimented with non-uniform traffic using a destination distribution model which is based on the Zipf's law and which was proposed in [17]. In a switch with N input/output ports all inputs are 100% loaded and input 0 sends to output i with probability

$$Zipf(i) = \frac{i^{-k}}{\sum_{j=1}^{N} j^{-k}}. \tag{2}$$

The distribution of destinations for each of the rest of the inputs results from a different cyclic shift of the distribution of input 0 so that the traffic is admissible; fig. 9 shows an example for $N = 4$ and $k = 2$. For $k = 0$ the traffic is uniform while for $k \to \infty$ it is totally directional.

Fig. 10 displays results for most of the simulated models when the packet size is bimodal and the Zipf order is in the range from 0 to 6. We also run experiments assuming uniform and constant packet size for the buffered crossbar switch; similar results were seen. The first observation is that the curves corresponding to buffered crossbar models are almost indistinguishable. The second observation is that the packet mode modification of iSLIP yields significantly higher throughput than its original proposal and very close to buffered crossbars. The superiority of packet mode scheduling under non-uniform traffic in bufferless crossbar scheduling was also observed in [1] [2]; packet mode scheduling exploits the locality of traffic reusing already made scheduling decisions and increases the size of the match after each scheduling operation.

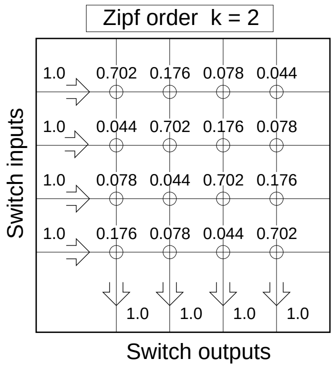


Fig. 9. Example of the distribution of destinations in a 4×4 switch according to the Zipf law and when the Zipf order, k, equals 2.

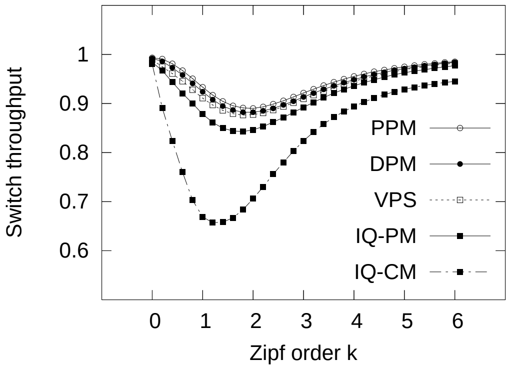


Fig. 10. Switch throughput under non-uniform (Zipf) traffic. Packet size is bimodal.

6. Conclusion

We proposed and evaluated a *probabilistic* and a *deterministic* packet mode scheduling scheme for buffered crossbar switches. The deterministic scheme performs virtually as well as buffered crossbars that use no segmentation, and eliminates egress reassembly buffers like the latter systems do, while using crosspoint buffers whose size is only linked to the crossbar-ingress round-trip time –and not to the maximum packet size– hence can be much smaller than in buffered crossbars without segmentation; performance is always better than bufferless crossbars with packet mode scheduling. Probabilistic packet mode scheduling allows cu-through forwarding and performs even better than the deterministic scheme for some traffic patterns, while it performs similar for the rest; it allows independent output schedulers in the crossbar, but it does need the extra cost of egress reassembly buffers.

Acknowledgments

The authors thank Nikos Chrysos for his early observations on the capacities of central schedulers in buffered crossbar switches, Enrico Schiattarella for his suggestions on simulation traffic patterns, and Alejandro Martinez for valuable discussions during the last few months. This work was performed within the projects "Scalable Intelligent Video Server System (SIVSS)" and "Scalable Architectures (SARC)", supported by the European Union FP6 IST programme.

References

[1] M. Marsan, A. Bianco, P. Giaccone, E. Leonardi, and F. Neri, "Packet scheduling in input-queued cell-based switches," in *Proceedings of the INFOCOM'01 Conf.*, April 2001.

[2] A. Marsan, A. Bianco, P. Giaccone, E. Leonardi, and F. Neri, "Packet-mode scheduling in input-queued cell-based switches," *IEEE/ACM Transactions on Networking*, vol. 10, no. 5, April 2002.

[3] S. H. Moon and D. K. Sung, "High-performance variable-length packet scheduling algorithm for IP traffic," in *Proceedings of the IEEE GLOBECOM'01 Conf.*, November 2001.

[4] X. Zhang and L. Bhuyan, "Deficit round robin scheduling for input-queued switches," *IEEE Jour. Sel. Areas in Communications*, vol. 21, no. 4, pp. 584–594, 2003.

[5] S. Mukherjee, F. Silla, P. Bannon, J. Emer, S. Lang, and D. Webb, "A comparative study of arbitration algorithms for the Alpha 21364 pipelined router," in *Proceedings of the ACM ASPLOS-X Conf.*, October 2000.

[6] J. Hurwitz and W. Feng, "Initial end-to-end performance evaluation of 10-gigabit ethernet," in *Proceedings of the IEEE Hot Interconnects Conf.: 11th Symposium on High-Performance Interconnects*, August 2003.

[7] D. Stephens and H. Zhang, "Implementing distributed packet fair queueing in a scalable switch architecture," in *Proceedings of the IEEE INFOCOM'98 Conf.*, March 1998.

[8] R. Rojas-Cessa, E. Oki, and H. J. Chao, "CIXOB-k: Combined input-crosspoint-output buffered switch," in *Proceedings of the IEEE GLOBECOM'01 Conf.*, November 2001.

[9] F. Abel, C. Minkenberg, R. Luijten, M. Gusat, and I. Iliadis, "A four-terabit packet switch supporting long round-trip times," *IEEE Micro Magazine*, vol. 23, no. 1, pp. 10–24, 2003.

[10] N. Chrysos and M. Katevenis, "Weighted fairness in buffered crossbar scheduling," in *Proceedings of the IEEE Workshop on High Performance Switching and Routing (HPSR'03)*, June 2003.

[11] K. Yoshigoe and K. Christensen, "A parallel-polled virtual output queued switch with a buffered crossbar," in *Proceedings of the IEEE Workshop on High Performance Switching and Routing (HPSR'01)*, May 2001.

[12] M. Katevenis, G. Passas, D. Simos, I. Papaefstathiou, and N. Chrysos, "Variable packet size buffered crossbar (CICQ) switches," in *Proceedings of the IEEE Int. Conf. on Communications (ICC'2004)*, June 2004.

[13] M. Katevenis and G. Passas, "Variable-size multipacket segments in buffered crossbar (CICQ) architectures," in *Proceedings of the IEEE Int. Conf. on Communications (ICC'2005)*, May 2005.

[14] S. M. Ross, *Simulation*. Academic Press, 3rd Edition, 2001, ISBN 0125980531.

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

[16] *Cooperative Association for Internet Data Analysis*, http://www.caida.org.

[17] Networking Processing Forum (NPF), *Fabric Benchmarking Traffic Models*, available from http://www.npforum.org/benchmarking/fabric_bm.shtml.

