Scheduling in Switches with Small Internal Buffers

Nikos Chrysos and Manolis Katevenis[‡]

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

Draft version of 22 July 2005

***Abstract*— Unbuffered crossbars or switching fabrics contain no internal buffers, and function using only input (VOQ) and possibly output queues. Schedulers for such switches are complex, and introduce increased delay at medium loads, because they have to admit at most one cell per input *and* per output, during each time slot. Buffered crossbars, on the other hand, contain sufficient internal buffering (N^2 buffers) to allow independent schedulers to concurrently forward packets to the same output from any number of inputs. These architectures represent the two extremes in a range of solutions, which we examine here; although intermediate points in this range are of reduced practical interest for crossbars, they are nevertheless quite interesting for switching fabrics, and they may be of interest for optical switches. We find that tolerating *two* cells per-output per time-slot, using small buffers inside the switch or fabric, sufficies for independent and efficient scheduling. First, we introduce a novel "request-grant" credit protocol, enabling N inputs to share a small switch buffer. Then, we apply this protocol to a switch with N such buffers, one per output, and we consider the resulting scheduling problem. Interestingly, this looks like unbuffered crossbar schedulers, but it is much simpler because it comprises independent schedulers that can be pipelined. We show that individual buffer sizes do not need to grow, neither with switch size nor with propagation delay. Through simulations, we study performance as a function of the number of cells allowed per-output per-time-slot. For one cell, the switch performs very close to the iSLIP unbuffered crossbar with one iteration. For more cells, performance improves quickly; for 12 cells, packet delay under (smooth) uniform load is practically as low as ideal output queueing. Under unbalanced load, throughput is superior to buffered crossbars, due to better buffer sharing.**

1 . INTRODUCTION

Packet networks require low-cost and fast packet switches to keep pace with the increase in communication demand. Packet switch architectures fall in one of the following two principal categories: (a) architectures with unbuffered fabrics, and (b) architectures with buffered fabrics. Most commercial switches of today belong to the former category, but the trend is towards buffered fabrics that exploit advances in IC technology with increased on-chip memory. This paper presents a combining system, which inherits benefits from both categories. The resulting architecture uses considerably fewer packet buffers than other buffered architectures that are currently examined, maintains simplified (independent and inherently pipelined) scheduling, and provides performance which is strictly better than that of practical unbuffered architectures.

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

With unbuffered fabrics, output conflicts must be avoided at the inputs, before packets are switched. Avoidance at the inputs requires a central crossbar scheduler to coordinate the set of input/output pairs (connections) that will be served at each time-slot [1]; this is a complex task that can limit the switch packet rate. Heuristic algorithms that have been adopted today work well only when internal speedup is used to compensate for their scheduling inefficiencies [2]. Because these algorithms operate only on fixed-size units, additional internal speedup is needed when the external network operates on variable-size packets. In effect, the complexity of these algorithms, which under the combined speedup need to run much faster than the line-rate, constitutes a crucial bottleneck of unbuffered architectures in scaling to large port-counts, and in supporting very fast links, as OC768.

Buffered architectures ease scheduling by allowing conflicting packets to enter the fabric. Recently, the buffered crossbar, or combined input crosspoint queueing (CICQ) architecture, receives research attention because it features simple and efficient scheduling [3][4][5][7], and because the speed of each crosspoint queue does not increase with switch size.

Buffered crossbars enable N input and N output schedulers to work independent of each other, in a pipeline arrangement. All inputs may concurrently send packets for the same switch output, and these packets will be held inside the crosspoint queues until they are served by the corresponding output link arbiter. Backpressure flow-control is used to keep the size of the crosspoint queues small, so that they can be implemented inside inexpensive on-chip SRAMs; effectively, backpressure impedes the independent input schedulers from repeatedly making conflicting decisions [5]. It is interesting that simple (and independent) round-robin (RR) schedulers in buffered crossbars yield considerably better delay-throughput performance compared to practical unbuffered crossbar schedulers. An additional advantage of independent scheduling is that it can be performed directly on variable-size packets, eliminating the segmentation overhead as demonstrated in [6].

These benefits come at the expense of a more expensive fabric. The internal memory of a buffered crossbar grows with $N^2 \cdot RTT \cdot \lambda$, where N is the switch size, RTT the round-trip time between the input line-cards and the crossbar, and λ the line-rate. This is a high cost for switches with large numbers of ports; even for modest switch size, the implementation can be expensive when the $RTT \cdot \lambda$ product is large [8].

1.1 Contributions

Unbuffered fabrics, on one hand, and crossbars with one buffer per input-output pair, on the other hand, are the two extremes in a range of architectures that contain some small amount of buffering inside a crossbar or a switching fabric. In this paper, we examine these intermediate design points: what is the minimum amount of buffer space that will enable efficient, independent, and pipelined scheduling? Our main finding is that allowing up to two cells for the same output to concurrently enter the fabric suffices. To make this feasible, we had to go over a sequence of steps which we present in this paper. First, we replace blind-mode transfers (sending without prior notice) with requested admissions (Section 2). This enables the construction of a switch with a $RTT \cdot \lambda$ queue at each output maintaining simplified scheduling (Section 3.1). However, in practical situations ($RTT \cdot \lambda >$ one cell), multicell output buffers supporting concurrent cell arrivals are needed. Next, we remove the dependence of queue size on propagation delay (Section 3.2), and we present a packet switch with one-cell output buffers, allowing up to two conflicting cells per output, which features inherently pipelined and independent scheduling. We further show that this system with RR schedulers has close correspondence with iSLIP, and we prove that, like iSLIP, it can deliver 100% throughput under uniform traffic (Section 3.3). Last, using simulations, we demonstrate the performance increase with multicell output buffers (Section 4): as more inputs are allowed to send cells for the same output in parallel, performance approaches that of buffered crossbars.

These results have both theoretical and practical importance. With buffered architectures in mind, (i) we demonstrate how regulated admissions can reduce the number of queues by a factor on the order of N, while still operating effectively; and (ii), we show how to eliminate the dependence on the propagation delay between the line-cards and the fabric when sizing fabric queues. The other way around, starting from unbuffered crossbar architectures, our system demonstrates that allowing two cells for the same output to concurrently enter the fabric enables *inherently* pipelined and independent scheduling, similar to buffered crossbars scheduling. Pipelined unbuffered crossbar schedulers ([9][10]) typically employ multiple crossbar schedulers, each one comprised of non-independent schedulers, which is only a halfway solution. In addition, we show how the backpressure buffer flow-control can desynchronize the independent schedulers, and we demonstrate how performance increases with buffer storage.

Our results are of interest primarily for fabrics consisting of multiple switches. For a single $N \times N$ switch implemented as crossbar, placing fewer than N^2 buffers "near" its outputs is awkard, because we would have to increase the output throughput of the crossbar, which often costs more than the reduction in memory bits. However, an $N \times N$ switching fabric will often contain less than $O(N^2)$ switching elements; even if each switching element is internally a buffered crossbar, the entire fabric will contain less than $O(N^2)$ buffers. Studying the behavior of an entire fabric would be too complex for a first investigation like this one; however, the main message of this paper is as follows. There are scalable scheduling methods that can be used to control the number of cells that are allowed to enter the fabric. These methods are scalable because they consist of *independent* per-output and per-input schedulers, cooperating in a *pipelined* fashion. Once the number of cells entering the fabric is properly controlled, a limited amount of buffer space inside the fabric suffices to achieve high performance. Our model is crude because it assumes that all buffer space is concentrated at the outputs, however it constitutes a first approximation towards studying the switching fabric itself; the results are so encouraging that they suggest that this can be a promising approach for flow control and congestion management in buffered multistage switching fabrics.

This study can also be of interest for optical switches. Scheduling can be dramatically simplified if the optical switch contains a small (e.g. one-cell), fixed-delay, fiber delay line (FDL) at each output; a cell will need to be stored on an output FDL for one or a few cell times.

1.2 Previous Work

Reference [12] considers a switch with FIFO inputs queues (not VOQs), and infinite output queues accepting a limited number of concurrent arrivals. Our difference from this study is that we consider VOQs and finite output queues, and that we study the emerging scheduling and flow-control problems. Reference [14] describes a request-grant protocol incorporating buffer allocation but applies this scheme to a buffer being fed by a single input. This scheme is intended for unbuffered architectures employing crossbar scheduling; per-input synchronization buffers are used to compensate for differences in the propagation delays. Instead, we use our request-grant protocol for queues shared among multiple inputs. In [13], the authors provide a flow-control scheme with scheduling and routing being performed in advance of packets (data-flits) by special packets (control-flits). Although their scheme results in efficient buffer usage, in our understanding it requires buffers to grow with turn-around delay. Our methods apply in the intra-switch context, not in an interconnection network as theirs, and in addition we do not need buffers to increase with turn-around delay.

We also regard as relevant with ours the work in [16], which demonstrates the PRIZMA shared-memory switch architecture. The output queue "grant" flow-control protocol that they use is actually a type of On/Off backpressure, which is does not to utilize buffers as efficiently as credit-based backpressure. In principal, this protocol, in contrast to classical credit-based backpressure, enables queue sharing among multiple inputs; however it works efficiently only if each queue has designated space for all N inputs. Therefore, as the designers of PRIZMA chip set comment, the fabric requires O(N^2) packet buffers in total, in order for fair and independent scheduling to be realized. In this paper we show how our "request-grant" credit protocol can be used for fair independent scheduling with just O(N) packet buffers.

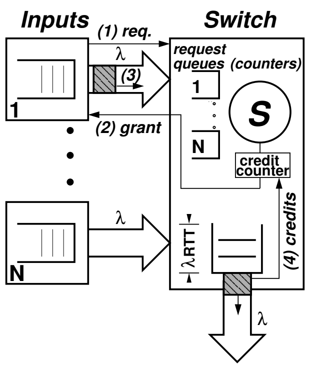

Fig. 1. N inputs sharing a $RTT \cdot \lambda$ switch queue.

Finally, we view as an ancestor of our request-grant credit protocol reference [17] describing the IBM SP2 switch architecture, which is comprised of the Vulcan, shared-memory, switching elements. The Vulcan switch node utilizes requests and grants to schedule accesses to it's single-ported shared-memory, and hence to economize in memory bandwidth rather than economizing in memory size as we do. We believe that in the absence of memory bandwidth limitation, a type of On/Off flow-control could equally well be employed, since the round-trip time within the Vulcan node is very small (one or two clock-cycles); and, 1KB shared-memory is large enough to accomodate concurrent (1-byte) flit arrivals from all input ports to any output port of the 8×8 Vulcan switch chip. (Essentially, the Vulcan shared-buffer contains more than N^2 flit (or chunks of 8 flits) locations). Furthermore, our target besides to memory savings is to utilize this protocol in a multistage fabric enforcing congestion management.

2 . Request-Grant Backpressure

In this Section we present a novel variant of credit-based backpressure, which enables queue sharing among multiple upstream inputs. Under typical credit-based backpressure flow-control, each upstream input is assigned private credits which correspond to a dedicated $RTT \cdot \lambda$ queue inside the fabric. In effect, inputs may forward traffic in "blind-mode", without taking into account what other inputs do, and will only sense output contention through the lack of credits. Hence, even when congestion is present, all inputs may send one $RTT \cdot \lambda$ worth of traffic for a particular destination which will reside inside fabric queues.

A crucial observation, which can lead to buffer savings, is that when the aggregate of these queues is served with peak rate λ, one $RTT \cdot \lambda$ of queued traffic suffices to prevent (aggregate) queue underflow. Though, it is not acceptable to divide $RTT \cdot \lambda$ buffer credits among the inputs, because this will work only when all inputs have traffic; therefore, to achieve this potential economy on buffer space, inputs must share buffer credits.

Consider that the input line-cards upstream to the switch have access to a common credit-counter for a switch queue that they intend to share –see figure 1. As in credit-based flow-control, each input must first secure credits before sending its data downstream. To resolve credit contention, which appears when multiple inputs concurrently need credits, the reservation is performed by first issuing a *request* to a *credit scheduler* authorized to perform credit allocation. Requests wait inside *request queues* for their turn to be served, and the scheduler considers as eligible only those requests demanding credits that are available. When it selects one, it decrements appropriately the buffer credit-counter, and replies to the issuing input via a *grant*. The recipient of the grant, having the required credits reserved, can safely proceed to the corresponding data transfer. For the flow-control to be complete, the shared credit-counter is incremented when traffic departs from the associated packet queue via feedback signals (credits) that travel to the credit scheduler.

The rate R, at which the scheduler hands credits out, must be $R \geq \lambda$. If $R > \lambda$, the credit-scheduler may occasionally produce bursts of grants, but its effective rate will be dictated by the rate that credits are released, ie, λ. The round-trip time in this protocol equals the delay from a cell departure that increases the shared credit-counter, till a cell which reserves the newly released credits reaches the output queue and is ready for transmission. The request corresponding to the latter cell can be in advance (of the credit release) inside the request queues, hence the round-trip time (and the associated queue space) is comparable to that of credit-based backpressure. Observe that it is straighforward to prevent overflow of the the request queues, if each input interprets grants as "credits" to send new requests[1]. With fixed-size cells, the request queues can be implemented using counters.

The advantage of the request-grant backpressure scheme is that *one* $RTT \cdot \lambda$ packet queue suffices to support full line-rate to any input that requests for it, whereas typical backpressure needs N such queues; when the collective demand from the inputs exceeds the line-rate, the excess traffic is effectively backpressured. The drawback is that credits must be requested through a separate opening transaction, therefore packet latency under low traffic is increased.

3 . A Switch with Small Output Queues

In this Section we present a switch with one small queue at each output, managed using request-grant backpressure.

3.1 Switch Description

Figure 2 presents our scheme. The input line-cards contain large virtual output queues, and express their demand for an output by issuing a request to the associated credit scheduler. Outstanding requests are kept in request counters, organized per-input (and per-output), which will be served in subsequent cell times. Unmatched inputs, that wait for grant (credit), are allowed to send new requests to the same or to other outputs; thus, multiple grants from different outputs can be generated concurrently for the same input. A *grant scheduler* associated

[1]To prevent underflow of the request queues, which in turn can result in underflow of the packet queue, each input must be allowed to send a round-trip time worth of requests in-advance, before it receives a new grant.

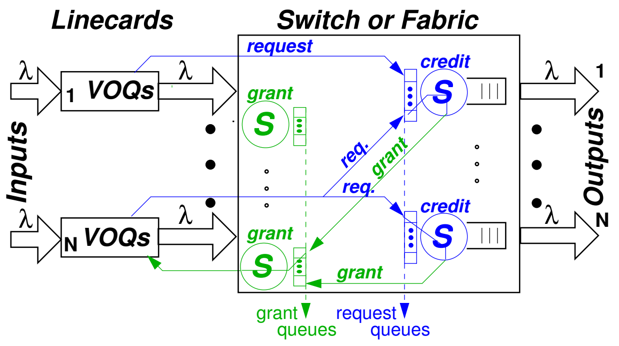

Fig. 2. A switch with small output queues managed using request-grant backpressure. Request and grant queues can be implemented using counters.

with the input selects one among them, sends it to the input line-card, and keeps the remaining grants inside appropriate *grant queues*, organized per-output, which will be served in subsequent cell times. The input line-card responds to an arriving grant by forwarding the corresponding cell, which acknowledges the grant by generated by the credit scheduler; when the cell starts departing from the output queue, the credit assigned to it returns to the credit scheduler.

This organization of credit and grant schedulers resembles schedulers for unbuffered crossbars, like iSLIP, but, by using small output buffers, the present scheme is simpler. There is no need for schedulers to coordinate their decisions on a cell-time basis, as they do in iSLIP; instead, they can operate independently, in a two-stage pipeline: in the first pipeline stage, each credit scheduler independently produces a grant and sends it to the corresponding grant (pipeline) queue; in parallel with the first stage operations, each grant scheduler (second pipeline stage) independently selects one among the grants accumulated up to now inside its grant queues –not considering the concurrent outcomes by the credit schedulers. In this way, the matchings produced can be conflicting but we do not care: if more than one input line-cards receive a grant for the same output at the same time, the output buffer will resolve the resulting conflict. This type of scheduling is as simple as buffered crossbar scheduling.

Although global coordination is not imposed explicitly, and the independent schedulers could synchronize in states of poor throughput, they will tend to desynchronize as they do in the iSLIP architecture [1]. The correspondence between the present scheme and the iSLIP switch originates from the following property. A “pending” grant generated by a credit (output) scheduler that is not selected (accepted) by the associated input (grant) scheduler will persist inside the grant queues until it is matched. This is analogous to what happens in iSLIP: an output continously grants an input until their matching is accomplished.

With multicell output buffers a credit scheduler may produce grants in consecutive cell times, in addition to a first pending grant, thus providing matching opportunities for other inputs as well. This means that multiple requests for the same output can be acknowledged in parallel, and that the output buffers will need to support concurrent cell arrivals. Assuming that the latency of each individual scheduler (credit or grant) is one cell time, the round-trip time will be greater than two cell times, thus at least two-cell output buffers are needed. Of course, the round-trip time additionally includes the propagation delay. We now show how we can eliminate the dependence on this parameter.

3.2 *Making Queue Size Independent of Propagation Delay*

In figure 2, the grant schedulers are positioned on the input side of the switch node; alternatively, they could be placed on the input line-cards. We will show that the placement of the grant schedulers in the same chip with the credit schedulers, as shown in figure 2, can eliminate the propagation delay P, from the effective round-trip time used in dimensioning the output queues.

Essentially, a cell departure from the output buffer is committed when the corresponding grant is sent from the grant scheduler; hence, the output buffer credits maintained by the nearby credit scheduler can be incremented just after the grant scheduler selection. Consider a grant, g, selected at time t by a grant scheduler. Grant g, will drive the enqueue of a cell, p_g, inside the output buffer at time $t+2\cdot P$. Assuming cut-through operation, the output buffer will start draining immediately on the arrival of p_g–if it wasn't draining already–, thus, in time $t+1+2\cdot P$ queue space will be available. This space can be safely reserved for a new cell from time $t+1$. (In other words, instead of waiting for returned credits from the output queues inside the fabric, credits are returned to the output credit schedulers from the nearby input/grant schedulers.) Using the last optimization, the switch operates efficiently with two-cell output queues supporting enqueues at rate $2 \cdot \lambda$: when the demand for an output is high, cells and grants for this output, of aggregate size $2 \cdot P \cdot \lambda$, will be virtually “stored” on the lines between the line-cards and the switch [2].

Lastly, consider that a final economy on buffer space is possible –probably suitable for optical switches. We can implement single-cell output queues, and permit two pending grants per output, as needed in order to compensate for the pipelined schedulers latency. When two cells concurrently arrive at an output, one of them will be stored inside the output buffer, while the other one will bypass the buffer on its way to the

[2]When the credit scheduler hands credits out at rate higher than one (1) credit/cell-time, we must be careful when performing these fast credit updates. Say that at time i, k (>1) grants for the same output are acknowledged by (different) input schedulers in parallel; in this case, the output credit count must not be incremented by k at once, since the output credit scheduler may then drive multiple ($>$1) new cell arrivals in time $t+1+2\cdot P$, whereas only one new cell position in the buffer will be available on that time. Hence, in order to remove the propagation delay from the effective round-trip time, we must ensure either that the credit schedulers serve requests at peak rate λ, or that the credit count is always incremented at that rate. In early stages of our simulator, not performing credit updates correctly lead us to believe that, performance considerably benefits by increasing the credit scheduler rate; however, this was due to output buffers growing beyond the limit that we had placed on them –actually, the credit scheduler rate improves performance, but only under smooth (e.g., Bernoulli) cell arrivals, and medium loads (see Section 4, fig. 5.).

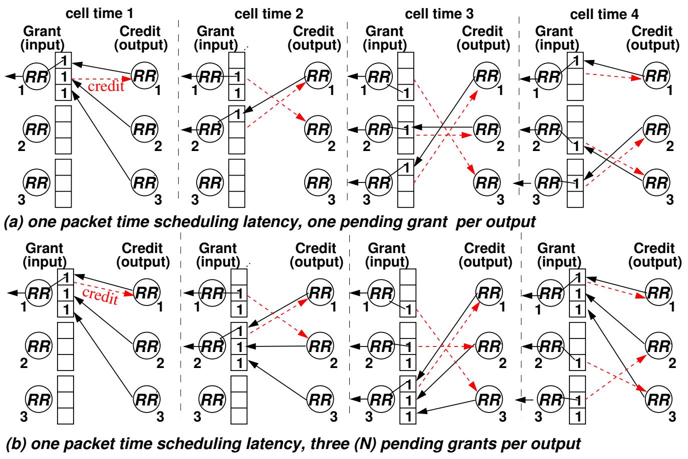
cell time 1
cell time 2
cell time 3
cell time 4
Grant (input)
Credit (output)
RR
credit
(a) one packet time scheduling latency, one pending grant per output
(b) one packet time scheduling latency, three (N) pending grants per output

Fig. 3. The desynchronization effect. The request queues (not shown) are assumed to be continously non-empty.

output. In the next cell time, the stored cell can depart for the output[3].

3.3 Throughput under Uniform Traffic

Let Q denote the number pending grants allowed per output –equivalently, the number of credits per-output–, and SD the pipeline scheduling latency, ie, the sum of credit and grant schedulers delays. With SD=1 and Q=1, it is trivial to prove that the throughput of the switch with RR schedulers, under fixed-size cell uniform traffic can reach 100%. Assuming 100% uniform load, after a point all request queues will be persistent. If $g_{k,i}$ denotes the number of cells inside the grant queues of input i in cell time k, then, it is easy to show that:

$$g_{k+1,i} = \begin{cases} 0, & g_{k,i} \geq 1, g_{k,i-1} = 0 \quad (1) \\ 1, & g_{k,i} > 1, g_{k,i-1} \geq 1 \quad (2) \\ g_{k,i} - 1, & g_{k,i} \geq 1, g_{k,i-1} = 0 \quad (3) \\ g_{k,i}, & g_{k,i} > 1, g_{k,i-1} \geq 0 \quad (4) \end{cases}$$

where $i - 1 = (i + N - 1) \bmod N$. Then, from [11], it follows that at most after $N - 1$ time-slots, all inputs will continously have a non-empty grant queue ($g_{t,i} > 0, \quad \forall \quad t > k+N-1, \quad \forall \quad i \in [1, N]$); hence, the switch will reach 100% throughput.

For the more practical system, with two cell times pipeline scheduling latency, and Q=2, a possible proof for the 100% throughput capability would not be trivial at all. The problem lies in that we cannot easily identify the input that an output will grant after it receives a credit from a grant scheduler, since it may have already granted several subsequent inputs utilizing the second available credit. We have proved that a system with SD=2 and Q=2, can provide 100% throughput when the grants generated by the same output are acknowledged by the grant schedulers in their generation order. However, as figure 3 demonstrates, desynchronization and 100% throughput are achieved even with $Q > 1$, even if no dependency is being enforced between the pending grants –[18] discusses this issue in more depth. Actually, our simulation results indicate performance improvements as Q increases.

[3]Assuming credit scheduler peak rate one (1) grant/cell-time–, at most one new cell will arrive in this next cell time. The new cell can be stored inside the output queue, while the previously stored cell departs.

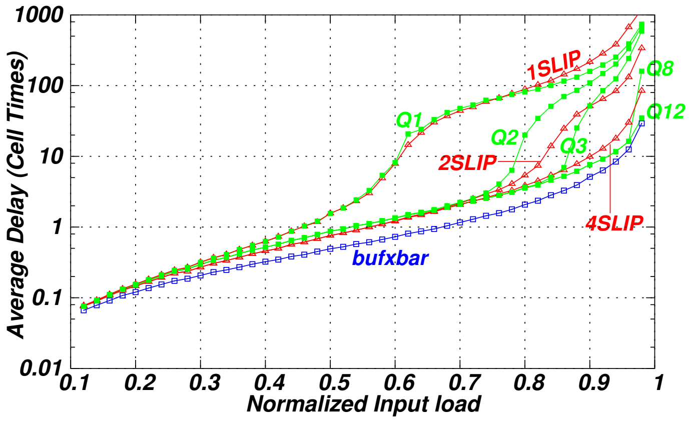

Fig. 4. N=32, P=0 ct, R=1 cell/ct, and SD=1 ct. Uniform Bernoulli cell arrivals.

4 . Simulation Results

The performance of the switch with RR schedulers was evaluated under uniform and unbalanced traffic using simulations. Simulations were run long enough to eliminate the effect of any initial transient, and the confidence intervals achieved were better than 10% around the reported values with confidence 95%[4]. In the plots that follow, we measure cells average delay in number of cell times (cts). Note that the round-trip time equals $2 \cdot P + SD$, and that the minimum recorded cell delay in all systems equals zero (we have removed the request-grant, cold-start delay overhead, as well as scheduling and propagation delays).

First, we use uniform Bernoulli arrivals and we compare the switch proposed here for different Q values, to the *i*SLIP switch (iterations 1, 2 and 4), and to a buffered crossbar with one cell buffer per crosspoint; all simulated switches are 32×32. Our cell delay results are presented in figure 4. A first point is that thanks to the desynchronization effect, for any Q value our system saturates near 1.0 input load. $Q1$ behaves very close to 1SLIP. With increasing Q, instances upon which an input does not receive a grant are expected to occur less frequently, and therefore delay improves. The delay of $Q12$ approaches that of the buffered crossbar ($bufxbar$); under smooth arrivals, we found no benefit in further increasing Q –reference [3, fig. 3] demonstrates that the delay of $bufxbar$ essentially matches ideal output queueing.

Figure 4 shows that at medium loads, for any Q value the delay of our system is slightly higher than $bufxbar$. We believe that these small discrepancies can be ascribed to the following behavior. At medium loads, occasional small bursts of cells for a switch output, from different inputs, enter the

[4]In throughput experiments, confidence intervals where better than 1%

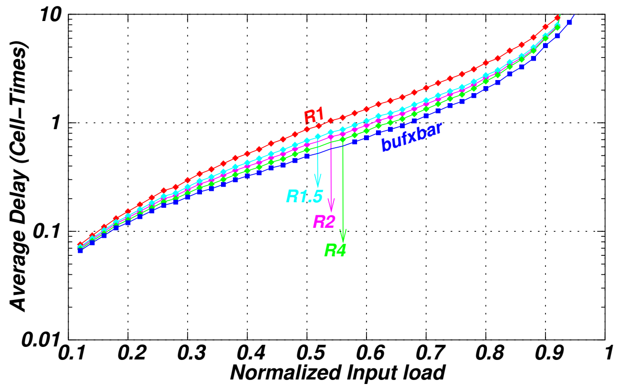

Fig. 5. N=32, P=0 ct, SD=1 ct, and Q=G=4. Uniform Bernoulli cell arrivals.

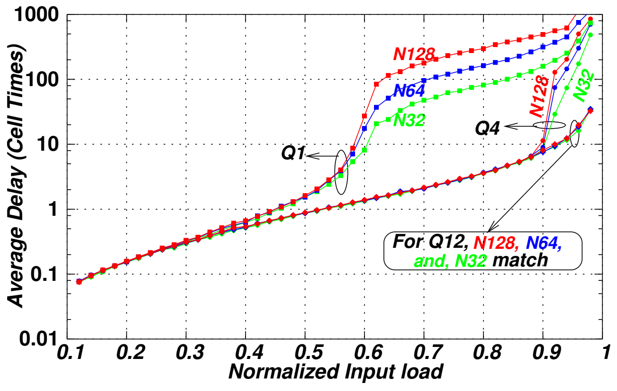

Fig. 6. P=0 ct, R=1 cell/ct, and SD=1 ct. Uniform Bernoulli cell arrivals.

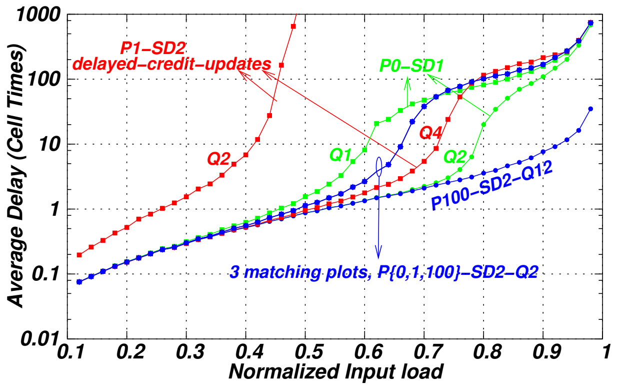

Fig. 7. N=32, R=1cell/ct. Uniform Bernoulli cell arrivals.

fabric of our switch only at the rate the credit scheduler admits cells inside, ie, 1 cell per-ct; in the buffered crossbar, such bursts may enter the fabric immediately, bypassing input contention. These "deferred" admissions in our system increase input contention and thereby cell delay. To support this argument we increase the rate R, at which each independent credit scheduler hands credits out [5]. Figure 5 shows our results for Q=4 and varying R. For $R > 1$, the cell delay at medium loads approaches $bufxbar$ because of more cells skipping input contention.

Next we evaluate the effect of switch size. Now figure 6 presents our results. We see that when Q is small, performance declines with increasing N. This behavior, also present in the iSLIP switch with few iterations [1], should be ascribed to harmful synchronizations among the credit (output) schedulers becoming more severe as the number of switch ports grows; but with increasing Q, the dependence on switch size vanishes because schedulers can desynchronize faster. Under Bernoulli arrivals, and for any switch size N in $\{32, 64, 128\}$, we found no benefit in increasing the output queues beyond 12 cells. This suggests that even from the point of view of performance, output queues' size does not have to increase with switch size, as in buffered crossbars or in the shared-memory architectures [16]. The performance of a 128×128 switch, with 12 cell buffers per output, approaches that of an equal size buffered crossbar (and that of pure output queueing) which requires approximately ten times more buffer space.

Following, we examine how performance behaves with increased scheduling latency and propagation delay. Results are presented in figure 7. A first observation is that $P0$-$SD2$-$Q2$ outpeforms $P0$-$SD1$-$Q1$: it turns out, that the availability of more credits per output in $SD2$-$Q2$ improves delay, even though the individual schedulers communicate their decisions with a cell time latency. Second, we see that our switch, by incorporating early credit updates as explained in Section 3.2, exhibits a performance that is independent of the propagation delay P: with constant Q, the switch performs equally well for all P value that we examine (0, 1, and 100 cts). On the other hand, a switch waiting for cell departures to increment credits needs Q (and output queues) to grow with $2 \cdot P + SD$. This is manifested through the performance curve $P1$-$SD2$-$Q2$ annotated by $delayed\text{-}credit\text{-}updates$: the round-trip time is $2 \cdot P + SD$= 4 cts, hence, with Q=2 the switch saturates at load 0.5.

A final remark is that the conclusions inferred using previous experiments for systems with unit scheduling delay apply equally well for the beneficial from the point of view of implementation system, with SD=2 cts and non-zero propagation delay. We can see in figure 7 that $P100$-$SD2$-$Q12$ attains very small delay. A 32×32 buffered crossbar switch with P=100 cts, achieving comparable performance with $P100$-$SD2$-$Q12$, requires 204 K of cell buffers, whereas $P100$-$SD2$-$Q12$ uses only 384. With increasing switch size, the cost reduction achieved increases even more.

This paper does not deal with bursty cell arrivals; bursty arrivals reveal some issues, which call for additional mechanisms; due to space limitations, these mechanisms are discussed in an extended version of this paper [18].

Last we experiment with unbalanced traffic. We borrow the unbalanced traffic model of [4], where each input, i, sends most of its traffic to a private "favored" output –in our experiments to output i. As in [4], w denotes the unbalanced

[5]The grant schedulers still operate at rate λ, and cells enter the fabric from a given input, and depart from a given output, at peak rate λ. Hence, increases in R become functional only if Q large enough, otherwise the effective credit scheduler rate is limited by the rate that credits are released, ie, 1 credit per-ct

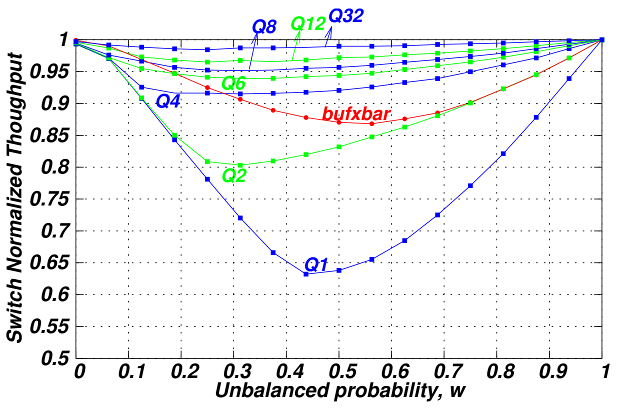

Fig. 8. N=32, P=0 ct, R=1 cell/ct, and SD=1 ct; 100% Input load, Unbalanced Bernoulli cell arrivals.

factor; when w=0 traffic is uniform, whereas when w=1 the switch is loaded by a persistent permutation. Our results, presented in figure 8, show that the throughput of $Q1$ can be as small as 0.63 for intermediate w values, which is also the throughput of the 32×32 1SLIP switch –see [3, fig. 6]. With increasing Q, throughput improves fast; for $Q4$ throughput is higher than 0.9, for $Q12$ higher than 0.97, and for $Q32$ higher than 0.99.

The intuition that throughput increases with buffer size can be better elucidated under this traffic model. Consider a heavily loaded flow, $f_{i \to i}$. Under unbalanced traffic, for intermediate w values, the input neighbors of f are occasionally active, thereby increasing during these periods the input contention that f faces; similarly, the output neighbors of f are occasionally active, increasing output contention. Therefore, output i can be underutilized when f experiences input contention and f's output neighbor flows are not active. A large output buffer (Q) absorbs traffic from f during the output contention periods, and occupies with that traffic the output when f experiences increased input contention. On the other hand, the throughput of the buffered crossbar is relative low due to memory partitioning: even though the crossbar contains 32 cell buffers per output, each heavily loaded flow can access only its private, single-cell, crosspoint queue.

5 . Conclusions

We presented a method to reduce the amount of internal buffer space in a switching fabric by a factor of the order of N; we also showed how the propagation time dependence can be removed when sizing fabric queues. We applied our methods in the design of a switch with small output queues, allowing a limited number of conflicts per output (Q), which features simplified scheduling and good performance. For this switch we showed how performance changes with varying Q: performance is close to that of unbuffered crossbars for Q=1, and increases with Q, approaching that of buffered crossbars for Q=12. A value of $Q \geq 2$ enables independent and inherently pipelined scheduling. Of course, a fabric that supports multiple output conflicts requires output queues running at a speed of multiple λ's. Our results indicate that Q, and therefore queue speed, does not have to increase neither with switch size nor with propagation delay; hence, techniques for high-bandwidth buffers, originating from shared-memory architectures ([15][16]), can be used in our switch in a scalable way.

References

[1] Nick McKeown: "The *i*SLIP Scheduling Algorithm for Input-Queued Switches" *IEEE/ACM Trans. on Networking*, vol. 7, no. 2, April 1999.

[2] P. Krishna, N. Patel, A. Charny, R. Simcoe: "On the Speedup Required for Work-Conserving Crossbar Switches", *IEEE J. Sel. Areas in Communications*, vol. 17, no. 6, June 1999, pp. 1057-1066.

[3] D. Stephens, H. Zhang: "Implementing Distributed Packet Fair Queueing in a scalable switch architecture", *Proc. IEEE INFOCOM Conf.*, San Francisco, CA, March 1998, pp. 282-290.

[4] R. Rojas-Cessa, E. Oki, H. Jonathan Chao: "CIXOB-k: Combined Input-Crosspoint-Output Buffered Switch", *Proc. IEEE GLOBECOM'01*, vol. 4, pp. 2654-2660.

[5] N. Chrysos, M. Katevenis: "Weighted Fairness in Buffered Crossbar Scheduling", *Proc. IEEE HPSR'03*, Torino, Italy, pp. 17-22. http://archvlsi.ics.forth.gr/bufxbar/

[6] Manolis Katevenis, Giorgos Passas, Dimitris Simos, Ioannis Papaefstathiou, Nikos Chrysos: "Variable Packet Size Buffered Crossbar (CICQ) Switches", *Proc. IEEE ICC'04*, Paris, France, vol. 2, pp. 1090-1096. http://archvlsi.ics.forth.gr/bufxbar

[7] N. Chrysos, M. Katevenis: "Multiple Priorities in a Two-Lane Buffered Crossbar", *Proc. IEEE Globecom*, TX, USA, 29 Nov. - 4 Dec. 2004, CR-ROM paper ID "GE15-3".

[8] F. Abel, C. Minkenberg, R. Luijten, M. Gusat, I. Iliadis: "A Four-Terabit Packet Switch Supporting Long Round-Trip Times", *IEEE Micro Magazine*, vol. 23, no. 1, Jan./Feb. 2003, pp. 10-24.

[9] E. Oki, R. Rojas-Cessa, H. J. Chao: "A Pipeline-Based Approach for a Maximal-Sized Matching Scheduling in Input-Buffered Switches", *IEEE Communication Letters*, vol. 5, no. 6, pp. 263-265, June 2001.

[10] C. Minkenberg, I. Iliadis, F. Abel: "Low-latency pipelined crossbar arbitration", *IEEE GLOBECOM'04*, Dallas, Tex., CR-ROM paper ID "GE15-2".

[11] Yihan Li, Shvendra Panwar, H. Jonathan Chao: "On the Performance of a Dual Round-Robin Switch", *IEEE INFOCOM'01* vol. 3, pp. 1688-1697.

[12] S.Q.Li: "Performance of a Nonblocking Space-Division Packet Switch with Correlated Input Traffic", *IEEE Trans. on Communications* , vol. 40, no. 1, Jan. 1992, pp. 97-107.

[13] Li-Shiuan Peh, Willia J. Dally: "Flit-Reservation Flow Control", *Proc. of the 6th Symposium on HPCA*, Toulouse, France, January 2000, pp. 73-84.

[14] PMC-SIERRA: "Linecard to Switch (LCS) Protocol", http://www.pmc-sierra.com/pressRoom/pdf/lcs_wp.pdf

[15] M.Katevenis, P.Vatsolaki, A. Efthymiou: "Pipelined Memory Shared Buffer for VLSI Switches", *Proc. of the ACM SIGCOMM'05 Conf.*, Cambridge, MA USA, pp. 39-48.

[16] R.P.Luijten, T.Engbersen, C.Minkenberg: "Shared Memory Switching + Virtual Output Queuing: a Robust and Scalable Switch" *Proc. of the IEEE ISCAS*, Sydney, Australia, May 2001, pp. IV-274-IV-277.

[17] C.B.Stunkel et. al.: "The SP2 High-Performance Switch", *IBM Systems Journal*, vol 34, no. 2, 1995.

[18] N. Chrysos, M. Katevenis: "Scheduling in Switches with Small Internal Buffers: Extended Version", http://archvlsi.ics.forth.gr/bpbenes