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/

***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 *i*SLIP 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

Networks need fast and low-cost packet switches to keep pace with the increase in communication demand. Switches employ ingress and egress linecards, which usually contain sizable buffer memories, and a core, which is a crossbar or a switching fabric. Packet switch architectures belong to two principal categories, depending on their core: *bufferless* or *buffered*. Crossbars were bufferless, but are now evolving to architectures with buffers per-crosspoint, owing to advances in IC technology that allow increased on-chip memory; analogous trends exist for fabrics made of multiple smaller switching elements. This paper studies the spectrum of intermediate solutions between the two extremes of bufferless and buffered crossbars. Our study provides indications that most of the advantages of buffered architectures –simple and efficient, distributed, pipelined scheduling– can be achieved with considerably less total buffer space compared to what buffered crossbars currently employ.

With unbuffered core, output conflicts must be avoided before packets enter the core. This requires a central scheduler to coordinate the set of input/output pairs (flows) that will be served in 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 speedup is needed when external packets have variable size, to compensate for segmentation padding.

Buffered architectures ease scheduling by allowing conflicting packets to enter the fabric. The buffered crossbar has one buffer per crosspoint (combined input crosspoint queueing - CICQ), and has received much research attention recently because it features simple and efficient scheduling [3] [4] [5]. Its single-resource, per-input and per-output schedulers operate independent of each other; loose, long-term coordination comes from backpressure flow-control, which is used to keep the size of the crosspoint queues small enough to fit on-chip. Flow control impedes repeated conflicting decisions by the schedulers, and enforces pipeline-like operation. An additional advantage of independent scheduling is that it can be performed directly on variable-size packets, eliminating the segmentation overhead [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 valency, RTT is the round-trip time between the ingress linecards and the crossbar, and λ is the line-rate. This is a high cost for switches with large numbers of ports, N; even for modest N, the implementation can be expensive when $RTT \cdot \lambda$ is large [7].

1.1 Contributions

Unbuffered fabrics, on one hand, and crossbars with one buffer per crosspoint, 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 least amount of buffer space that allows efficient, independent, and pipelined scheduling? We find that buffer space of 2 cells per

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

output suffices for decent performance, while buffer space of 12 cells per output yields close to ideal performance, almost *independently of switch size* N; these numbers are to be compared to buffer space of $N \cdot RTT \cdot \lambda$ per output in buffered corssbars.

Besides their theoretical importance, these results are of interest primarily for *fabrics* of multiple smaller switching elements. For a single $N \times N$ switch implemented as crossbar, placing fewer than N^2 buffers "near" its outputs is awkward, because we would have to increase the output throughput of the crossbar, which often costs more than the reduction in memory bits. However, any interesting $N \times N$ switching fabric contains less than $O(N^2)$ switching elements, and it is desirable to also limit its total buffer space quite below that value. The message of this paper is positive: There exist *scalable scheduling methods* to control the number of conflicting cells entering a fabric. Once that number is properly controlled, limited buffer space inside the fabric suffices for high performance. Our model is crude because it assumes that all buffers are at the outputs; however, it constitutes a useful first approximation towards a full study of the fabric itself. We are undertaking such a full study in a subsequent, current work [8].

To achieve the above small buffer spaces, we had to go over a sequence of steps which we present in this paper. First, we replace credit-based with *request-grant backpressure* (section 2). This increases latency by 1 RTT, but allows buffer space reduction by a factor of N, in principle. Section 3.1 presents the basic switch architecture, with independent per-output and per-input schedulers operating in a pipelined fashion, similar to buffered crossbar scheduling. Pipelined unbuffered crossbar schedulers [9] typically employ multiple crossbar schedulers, each one comprised of non-independent schedulers, which is only a halfway solution.

Next, we propose a *credit predicition* scheme, which further reduces buffer size, making it *independent of propagation delay (RTT)*, by exploiting the lack of backpressure at the egress ports (section 3.2). The resulting system allows independent, pipelined scheduling with output buffers as small as a single cell (which allows up to two conflicting cells per output). This part of our results can be of interest for optical switches: scheduling can be simplified significantly 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.

Section 3.3 outlines the similarities of our system, when using round-robin schedulers, to *i*SLIP; we discuss how "desynchronization" is achieved, and we show how the system provides 100% throughput under uniform traffic. Section 3.4 discusses grant-rate control. The last part of the paper (section 4) presents our simulation results. Performance improves significantly when buffer space per output increases from 1 to 4 cells, and less so up to 12 cells. We also study how performance depends on credit generation rate, scheduling delay, and switch size, and we demonstrate RTT-independence under the credit predicition scheme.

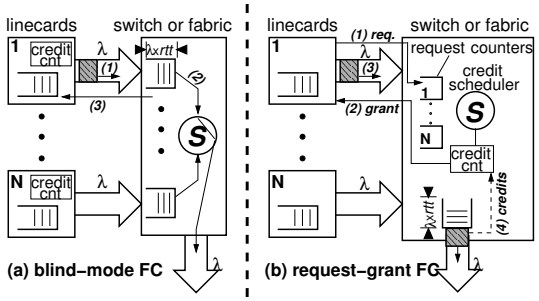

Fig. 1. (a) traditional credit-based flow-control needs N window buffers; (b) request-grant credit-based FC, using one window buffer.

1.2 Previous Work

In 1992, Li [10] considered a switch with FIFO inputs queues (not VOQs) and infinite output queues accepting a limited number of concurrent arrivals. Our study differs because we consider buffer space rather than buffer throughput limitation; also, we assume VOQs, and we study scheduler implementation. In an analogous way, the IBM SP2 Vulcan switch [11] used requests and grants to control the use of the limited throughput of its shared-memory buffer; again, we differ because we control buffer space rather than throughput. Recent PRIZMA work at IBM Zurich [12] considered a switch with VOQs and a limited shared-memory. However, the scheduling and the flow control (On/Off) style used end up requiring O(N^2) buffer space in the shared memory. By contrast, our scheme only uses O(N) buffer space.

Request-grant protocols like the one we use (section 2) are used to communicate with the schedulers of all bufferless crossbars. However, request-grant protocols have rarely been used for flow control, in ensuring that buffers do not overflow. Abrizio (later PMC-Sierra) [13] used the *LCS* request-grant protocol to control the utilization of a buffer fed by a *single* input in a bufferless crossbar system. Instead, we use our request-grant protocol for queues shared among multiple inputs. Finally, *flit-reservation* flow control [14] is reminiscent of our credit prediction scheme (section 3.2). Flit-reservation applies to general interconnection networks and results in efficient buffer usage, but buffer space is still dependent on round-trip time. By contrast, our credit prediction makes buffer space independent of round-trip time, but only applies to cases where there is no backpressure from the egress port.

2 . REQUEST-GRANT BACKPRESSURE

In this section we present a novel variant of credit-based flow control, which enables buffer space sharing among multiple upstream inputs. Consider N ingress linecards feeding one egress port of rate λ, as in figure 1. With traditional credit-based backpressure, figure 1(a), each input is allocated private credits corresponding to a dedicated $RTT \cdot \lambda$ window inside the fabric. This is necessary for individual inputs to be allowed to abruptly step up their transmission rate without needing prior "consultation" with the switch so as to learn about the other inputs' current rate. However, since the aggregate rate of all

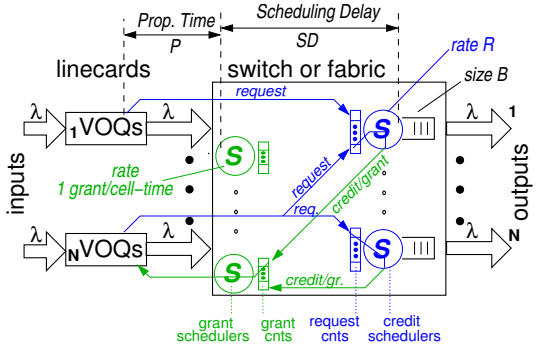

Fig. 2. A switch with small output queues managed using request-grant backpressure. Request and grant queues are implemented as counters.

inputs cannot exceed λ, a single $RTT \cdot \lambda$ window suffices, in principle, for the entire aggregate traffic. The problem with such a small buffer window is that it is not known a priori how to divide the credits for it among the N inputs.

This problem can be solved by making the ingress linecards share access to a *common credit counter* for the buffer space that they intend to share. Figure 1(b) shows how to do this. The shared credit counter is placed in the switch. Inputs must secure credits before transmitting data. To resolve credit contention when multiple inputs concurrently need credits, inputs first *request* credits from a *credit scheduler* authorized to allocate them. Requests wait inside *request queues* for their turn to be served. The scheduler decrements the credit counter when it serves a request, and returns a *grant* to the input being serviced. The recipient of the grant can now safely forward the corresponding data. The shared credit-counter is incremented when data depart from the shared buffer space.

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. As shown in fig. 1, with fixed-size cells, the request queues can be implemented using per-connection counters. The advantage of the request-grant backpressure scheme is that *one* $RTT \cdot \lambda$ window suffices to support full line-rate to any input that requests for it, whereas traditional backpressure needs N such windows. One drawback of the new method is that credits must be requested through a separate initial transaction, thus increasing packet latency under low traffic by one RTT (similar, though not exactly equal to the above RTT); also, requests consume some extra bandwidth.

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

This organization of credit (output) and grant (input) schedulers resembles schedulers for unbuffered crossbars, like *i*SLIP, 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 *i*SLIP; 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 linecards receive a grant for the same output at the same time, the output buffer will absorb the resulting conflict. This type of scheduling is as simple as buffered crossbar scheduling.

Denote by R the peak rate at which any particular credit scheduler hands credits out. In general, R may have any value $\geq$1 credit/time-slot; in this paper however, unless otherwise commented, we assume the minimum allowable rate R, ie one (1) credit/time-slot [1]. On the input side, we assume that each time a grant scheduler grants its corresponding input linecard, a cell is injected inside the fabric: the rate of each grant scheduler is one (1) grant/time-slot by default, for we do not assume any speedup at the fabric ports.

Although global coordination is not imposed explicitly, and the independent schedulers could synchronize in states of poor throughput, for B=1 (allowing one conflicting cell per-time-slot) they will tend to desynchronize as they do in the *i*SLIP architecture [1] –see section 3.3. With multicell output buffers, a credit scheduler may produce grants in consecutive time-slots, in addition to a first pending grant, thus providing matching opportunities for other inputs as well. This means

[1] If R>1, the credit scheduler may produce multiple credit in a single time-slot; however, its effective (long-term) rate will be dictated by the rate that credits are replenished, ie, 1 (cell)credit/time-slot. Our simulations show only marginal benefits in increasing R beyond this value.

that multiple cells may reach an output buffer in the same time-slot, since the grant schedulers work independently of each other.

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. Next, we show how we can eliminate the dependence on this parameter.

3.2 Credit Prediction: Independence from Propagation Delay

Let P be the (one way) propagation delay between a linecard and the switch –see figure 2. We will show how to eliminate P from the effective round-trip time used in dimensioning the output queues. This is possible because egress ports are not subject to external backpressure.

Credits are generated when cells depart through the egress ports. Since there is no external backpressure to these ports, if we know that an output queue will be non-empty at a given time in the future, we can predict that a cell will depart and a credit will be generated (per cell time) at that time in the future. Such predicted "future credits" can be used to trigger cell departures from the ingress linecards, provided we can guarantee that the corresponding cells will not arrive at the buffer before the above future time. In our case, consider a grant g selected at time t by a grant scheduler; g will arrive at its linecard at $t + P$, will trigger the corresponding cell departure, and that cell will arrive into its output buffer at $t+2P$. At time t we know g, hence we also know the output that it refers to; thus, we can safely conclude that output will be non-empty at time $t+2P$, and consequently it will generate a credit at $t+2P+1$. At $t+1$ we can use this predicted credit to generate a grant, given that the latency from grant generation to cell arrival can never be less than $2P$.

Using credit prediction, the switch operates efficiently with two-cell output queues supporting enqueues at rate 2λ, independent of P: 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 linecards and the switch.

For the scheme to work correctly, we must take care of one additional issue. Say that at time-slot t, k (> 1) grants for output o are selected by k grant schedulers in parallel. In this case, under credit prediction, k credits must be returned to the corresponding credit scheduler. Observe that the credit count should not be incremented by k at once in time-slot t, since the credit scheduler for output o may then drive multiple (>1) cell arrivals in time $t + 1 + 2 \cdot P$ –assuming $R>1$–, whereas only one new cell position in the buffer will be available on that time. Thus, we must throttle credit increments so that these occur at a peak rate of one (credit) increment per-time-slot per-output. This can be realized using an intermediate *predict credit counter*, in addition to the actual credit counter used so far. The predict credit counter, which is initialized at zero (0), is incremented every time a grant for that output is sent to an input linecard, and is decremented by one in every time-slot when it is greater than zero; once decremented, the corresponding (actual) credit counter is incremented by one [2].

3.3 100% Throughput under Uniform Traffic

Let B denote the buffer size –ie, the number of credits per-output–, and SD the pipeline scheduling latency, ie, the sum of credit and grant schedulers delays. Based on [15], it is trivial to prove that, under uniform cell traffic, the throughput of the switch that we propose in this paper with SD=1 cell-time, B=1, and pointer-based RR schedulers, can reach 100% [16]. (This result applies even for P greater than zero (0), if we employ credit prediction.) To see why, consider that, when B equals one (1), any particular credit scheduler may have only one input granted at any given time –before being notified that its "first" grant has been accepted[3]. The "first" output grant (credit) that a grant scheduler receives will reside in its grant queue, waiting to be served, which is equivalent to what happens in *i*SLIP –and, symmetrically in 2DRRM: *i*SLIP, instead of storing unaccepted grants, cancels them, but reproduces them in subsequent time-slots until these are accepted. For the more practical system, with two cell-time pipeline scheduling latency (SD=2), and two cell buffers per-output (B=2), a possible proof for the 100% throughput capability would not be trivial at all. However, our simulation results indicate that 100% throughput is still achieved.

3.4 Throttling Grants to a Bottleneck Input

According to our simulation results (section 4), the system performs very well under i.i.d. Bernoulli arrivals with buffers of 4 to 12 cells per-output, independent of the propagation time, under both uniform and unbalanced traffic. This section discusses system operation under bursty traffic (correlated cell arrivals). The corresponding simulation results are not presented in this paper, due to space limitations, but can be found in [16].

If multiple credit schedulers allocate credits to a same grant scheduler (for a same input) at about the same time, that input will not be able to respond as fast to all of them, due to its limited throughput. Such accumulations of credits in front of grant schedulers, waiting for the corresponding grants to be forwarded and used at rate λ, correspond to underutilization of the common pool of available credits, hence also underutilization of the available buffer space. Accumulations like this do not occur under smooth arrivals, because credit schedulers alternate quickly among the inputs to which they grant. Under bursty traffic, however, bursty requests arriving at credit schedulers may cause repeated allocation of credits to a same grant scheduler, hence the above phenomenon may appear. Our simulations showed that, using 12 cell buffers per-output, average cell delay under uniform bursty traffic may get 3 to 4 times higher compared to ideal output queueing at high

[2]Observe that, when R=1 credit/time-slot, the need for the predict credit counter is removed: each credit scheduler will always allocate only one new credit per-time-slot.

[3]either when the grant is selected by its counterpart input scheduler (credit prediction) or when the corresponding cell leaves the output buffer (no credit prediction).

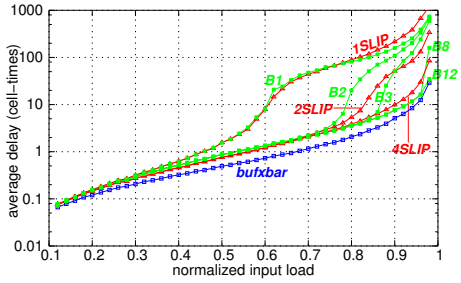

Fig. 3. Performance for varying buffer size, B; N=32, P=0, R=1 credit/cell-time, and SD=1 cell-time; Uniform Bernoulli cell arrivals; Only the queueing delay is shown, excluding all fixed scheduling and propagation delays.

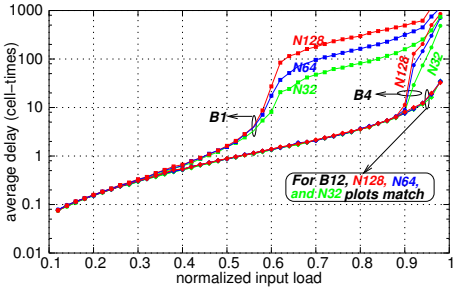

Fig. 4. Performance for varying sw. size, N; P=0, R=1, and SD=1; Uniform Bernoulli cell arrivals; Delays excludes all fixed delays.

switch load (higher than 0.9); under the same traffic, buffered crossbars achieve ideal performance.

In [16], we propose the use of a grant throttling mechanism, in order to maintain as good a delay as that of buffered crossbars. The key idea is for credit schedulers to stop serving an input when that input does not return credits fast enough. One way to achieve is as follows: a request from an input is eligibile at its output (credit) scheduler, iff (i) output buffer credits are available (as before), and additionally (ii) the combined credit/grant queue size before that input's grant scheduler is less than a threshold, TH. This method can be realized by having each input scheduler circulate an *On/Off* signal, common (indiscriminate) for all credit schedulers, that stays *Off* whenever the (grant) backlogs in its corresponding grant queues sum up to TH (or higher). Using simulations, we found that by adjusting TH, we can bring delay down to the levels of ideal OQ, using only 12-20 cell buffers per-output, and plain round-robin schedulers; these results apply for any switch size (N) in the range of 32 up to 128, and for a wide range of burstiness factors.

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 cell propagation delays). Unless otherwise noted, our results do not use neither credit prediction, nor grant throttling.

First, we use uniform Bernoulli cell arrivals and we compare our switch for different values of B –buffer space per-output in numbers of cells–, to the *i*SLIP switch (iterations 1, 2 and 4), and to a buffered crossbar with one cell buffer per crosspoint. Our cell delay results for N=32 are presented in fig. 3. $B1$ behaves very close to 1SLIP for the reasons described in section 3.3. With increasing B, instances upon which a backlogged input does not receive grant are expected to occur less frequently, therefore delay improves. $B12$ approaches the delay of buffered crossbar ($bufxbar$) –ref. [3, fig. 3] shows that $bufxbar$ virtually matches OQ delay; under smooth arrivals, we found no benefit in further increasing B.

Figure 3 shows that at medium loads, for any B value, the delay of our system is slightly higher than $bufxbar$. 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 fabric of our switch only at the rate the credit scheduler admits cells inside, ie, R= 1 cell (credit) per-cell-time in fig. 3; in the buffered crossbar, such small bursts may enter the fabric immediately, bypassing input contention. In our system, these "deferred" admissions increase input contention and thereby cell delay. We found out that, by increasing R to 1.5-2.0 credit/cell-time, the cell delay at medium loads approaches $bufxbar$ because of more cells skipping input contention [16]. At high loads, no considerable improvement were observed.

In fig. 4 we evaluate the effect of switch size, N. We find that, when B is small, performance declines with increasing N. This behavior, also present in the *i*SLIP algorithm using 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 B, the dependence on switch size vanishes because credit schedulers, even if synchronized at some point, they can keep on producing grants. 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 buffer space does not have to increase with switch valency.

Figure 5 examines how performance behaves with increased scheduling latency and propagation delay. A first observation is that, when credit prediction is employed, the queueing delay does not depend on the propagation time P: with constant B, the switch performs equally well for all P values that we examine (0, 1, and 100 cts). On the other hand, a switch waiting for cell departures to increment credits needs B to grow with $2 \cdot P + SD$. This is manifested through the performance curves corresponding to the configurations using $SD2$, $P1$ and *no credit prediction*: the round-trip time is $2 \cdot P + SD$= 4 cts, hence, for B=2 the switch saturates at load 0.5, and performs

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

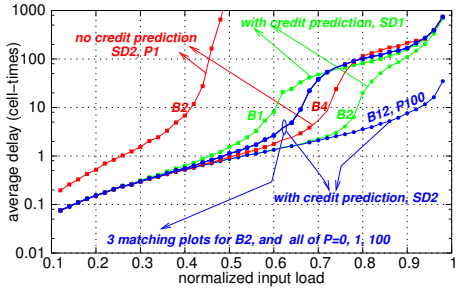

Fig. 5. Performance for varying scheduling delay SD, and varying P, using credit prediction, or without credit prediction; N=32, R=1 credit/cell-time. Uniform Bernoulli cell arrivals; Delays excludes all fixed delays.

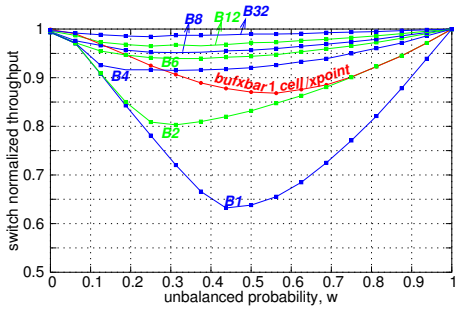

Fig. 6. Throughput under unbalanced traffic for varying buffer size, B; N=32, P=0, R=1 credit/cell-time, and SD=1 cell-time; 100% input load.

satisfactory for B=4. Another point is that the availability of more credits per-output in $SD2$-$B2$ improves delay compared to $SD1$-$B1$ –both using credit prediction–, even though output and input schedulers communicate their decisions with a cell time latency.

A final remark is that the conclusions inferred using previous experiments for systems with unit scheduling delay apply equally well when SD=2. Figure 5 shows that, when P=100, SD=2, and B=12, the queueing delay is very close to $bufxbar$ –compare to fig. 3. A 32×32 buffered crossbar switch, having 100 cell-times propagation delay, requires 204 K of cell buffers, whereas ours uses only 384. With increasing switch valency, the cost reduction achieved increases even more.

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 factor; when w=0 traffic is uniform, whereas when w=1 the switch is loaded by a persistent permutation. Our results, presented in fig. 6, show that the throughput of $B1$ can be as small as 0.63 for intermediate w values, which is also the throughput of the 32×32 1SLIP switch [3, fig. 6]. With increasing B, throughput improves fast; for $B4$, throughput is higher than 0.9, for $B12$ higher than 0.97, and for $B32$ higher than 0.99. Our system achieves better throughput than $bufxbar$ with one cell per-crosspoint, due to better buffer sharing.

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 delay 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 (B), which features simplified scheduling. For this switch we showed how performance changes with varying B: performance is close to that of unbuffered crossbars for B=1, and increases with B, approaching that of buffered crossbars for B=12. A value of $B \geq 2$ is sufficient for independent and inherently pipelined scheduling. Our results indicate that B does not have to increase neither with switch size nor with propagation delay; hence, techniques for high-bandwidth buffers, originating from known shared-memory switch architectures, can be used in our switch in a more scalable way.

Acknowledgments: this work has been supported by an IBM Ph.D. Fellowship. The authors would also like to thank CARV (FORTH) members for stimulating discussions.

References

[1] N. 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] M. Katevenis, G. Passas, D. Simos, I. Papaefstathiou, N. 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] 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.

[8] N. Chrysos, M. Katevenis: "Scheduling in Non-Blocking Buffered Three-Stage Switching Fabrics", FORTH-ICS, Crete, Greece, August 2005, 14 pages, http://archvlsi.ics.forth.gr/bpbenes/

[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] 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.

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

[12] 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.

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

[14] L. Peh, W. Dally: "Flit-Reservation Flow Control", *Proc. of the 6th Symposium on HPCA*, Toulouse, France, January 2000, pp. 73-84.

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

[16] N. Chrysos, M. Katevenis: "Scheduling in Switches with Small Internal Buffers: Extended Version", FORTH-ICS, Crete, Greece, September 2005; http://archvlsi.ics.forth.gr/bpbenes

