Scheduling in Switches with Small Internal Buffers: Extended Version

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, suffices 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, single-resource 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.

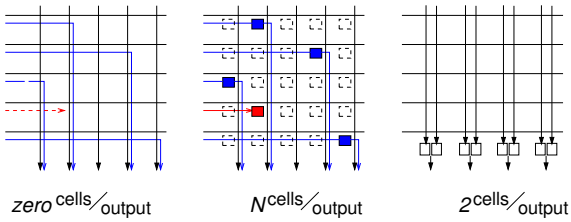

Fig. 1. Placement of buffers in a crossbar. Starting from the left, we have a bufferless crossbar, then a buffered crossbar, containing N^2 buffers, i.e. N per output, and, last, on the right, a system with less than N buffers per output –two cell buffers per output are shown in the figure, but, in general, the number of cells that fit in the aggregate output buffer can be either a constant independent of N, or a sub linear growing function of N. In this paper we consider the latter type of switches (or fabrics), i.e. buffered, with less than N buffers per output. In the figure of the bufferless crossbar, the red cell, shown in dashed line, cannot proceed as its output is occupied by an other cell; in buffered switches however, the cell can proceed, as it can be stored inside an output buffer.

With unbuffered core, output conflicts must be avoided before packets enter the core: in each time-slot[1], only a single cell in the crossbar can use a specific crossbar output, and only one cell in the crossbar can use a specific crossbar input; in graph theory, crossbar scheduling is equivalent to bipartite, input to output, graph matching. This problem requires a central scheduler to coordinate the set of input/output pairs (flows, or connections) that will be in the crossbar in each time-slot [1] [2] [3]; 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 [4]. 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 conflict-

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

[1] a time-slot is equal with a cell time, that is the time it takes to transmit a cell at rate λ.

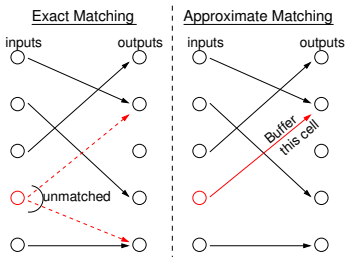

Fig. 2. (a) Exact versus approximate matchings, produced by independent input and output schedulers. In exact matches (left), an input may not be able to serve any cell if the cells it holds are for outputs currently matched with other inputs. In approximate matches (right), two or more inputs can concurrently feed the same output, which, as we show in this paper, increases flexibility and performance.

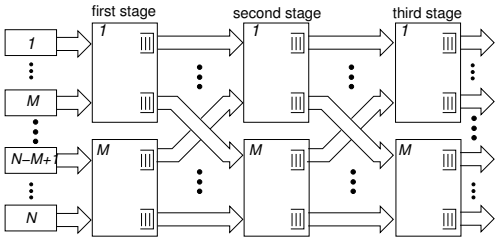

Fig. 3. A $N \times N$ three-stage non-blocking fabric, comprised of $M \times M$ switching elements with small output buffers. Even if each switching element is a buffered crossbar, it will contain (M) less memories per-output than (N) the number of ingress linecards, hence, it is imperative to find methods that will enable the sharing of these queues among the N ingress linecards. This paper studies the above problem in the simpler framework of a single stage switch.

ing 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 [5] [6] [7] [8].

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 [9].

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 [10].

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 –see figure 1 for alternative buffer placements. 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 output suffices for independent pipelined scheduling, that yields decent performance; with a buffer space of 12 cells per output, performance approaches the ideal, 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 crossbars.

Besides their theoretical importance, these results are of interest primarily for *fabrics* made 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 –see figure 1. 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 –see figure 3. 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 [11].

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 traditional credit-based with *request-grant credit* 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 [12] [13] typically employ multiple crossbar schedulers, each one comprised of non-independent schedulers, which is only a halfway solution.

Next, we propose a *credit prediction* scheme, which further reduces buffer size, making it *independent of propagation delay*, 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" can be achieved, and we discuss how the system provides 100% throughput under uniform traffic. Section 4 discusses *grant-rate control*, a method preventing

credit accumulations at (temporarily) congested inputs. The last part of the paper (section 5) 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 prediction scheme.

1.2 Previous Work

In 1992, Li [14] 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 [15] 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 [16] 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) [17] 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 [18] 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

2.1 Motivation

Buffered switching fabrics, i.e. networks comprising buffered switching elements (or switches), are advantageous as they tolerate approximate matchings –matchings that may include link conflicts. Such matchings can be implemented using independent, pipelined single-resource schedulers, one at each contention point of the fabric. It is common practice today to maintain large off-chip (expensive) packet buffers at the ingress stage, and small on-chip buffers inside the fabric. In this case, lossless backpressure needs to be exerted from the fabric queues on the upstream inputs, so as to prevent queue overflow.

An open issue in switching fabric design is the organization, the flow control, and the scheduling of these queues. It is advantageous to place the queues at the outputs of the switches, in order to allow the scheduling of these queues by independent, per-output schedulers. (In a large switching fabric, it is sometimes desired to have additional queues that will host flows destined to different fabric output-ports.) At the same time, it is also advantageous to associate each queue in a switch with an upstream (input) switch, in order to eliminate queue speedup. Credit-based flow control, reserving private credits at each switch for the set of queues fed by its output lines, works with such per input queues. In the end however, the combined effect is that the number of queues needed grows too fast, as we need N queues per output port.

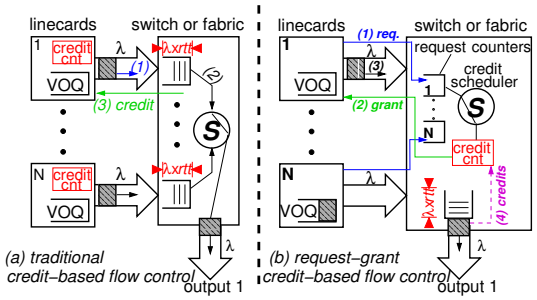

Fig. 4. (a) traditional credit-based flow control needs N window buffers; (b) request-grant credit-based flow control using a single window buffer.

2.2 Protocol

In this section we present a novel variant of credit-based flow control, which enables buffer space sharing among multiple upstream inputs. This offers the possibility of reducing buffer cost in large switching fabrics.

Consider N ingress linecards feeding one egress port of rate λ, as in figure 4. With traditional credit-based backpressure, figure 4.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 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 4.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. (While its credit-counter equals zero, the credit scheduler cannot serve input requests.) 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 (*RTT*) in this protocol equals the delay from a cell departure that increases the shared credit-counter, till a cell which reserves the newly released credit 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. With respect to figure 4.b, the RTT comprises the delays spent on (operations) arrows 4, 2, and 3.

As shown in figure 4.b, with fixed-size cells, the request queues can be implemented using per-connection counters.

Observe that it is straightforward to prevent overflow of the request queues, if each input interprets grants as "credits" to send new requests. VOQs with more than u pending requests[2] are not allowed to sent more requests to their credit scheduler, until they first receive a grant/credit back. Now, the size of a connection request queue (counter) will never exceed u. To prevent underflow of the request queues, which in turn can result in underflow at the packet queue, u has to be set so that each VOQ is allowed to send a round-trip time worth of requests before being notified that its "first" (i.e. earlier in time) pending request has been accepted[3].

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. These issues are not discussed further in this paper.

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 5 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

[2]a VOQ request is pending from the time it is issued from its input linecard (towards its output credit scheduler), until the corresponding grant is received back to the input linecard.

[3]the round-trip time corresponding to the flow-control of the request queues can differ from the round-trip time corresponding to the request-grant backpressure for the packet queue, if for example, the credit-scheduler and the shared packet queue are located in separate chips.

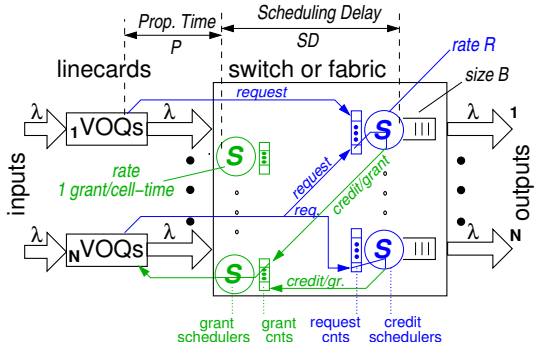

Fig. 5. A switch with small output queues managed using request-grant backpressure. Request and grant queues are implemented as counters.

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, with a complete cell time long pipeline "clock-cycle": 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 output conflict, which will occur after the granted cells reach the fabric, after some fixed propagation delay. This type of scheduling is as simple as buffered crossbar scheduling.

Let B denote the buffer size –i.e. the number of credits per-output–, and SD the pipeline scheduling latency –i.e. the sum of credit and grant schedulers delays.

Additionally, denote by R the peak rate at which any particular credit scheduler hands credits out. In general, R may have any value ≥ 1 credit/cell-time; in this paper however, unless otherwise commented, we assume the minimum allowable rate R, i.e. one (1) credit/cell-time[4]. 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/cell-time by default, for we do not assume any speedup at the fabric ports.

[4]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, i.e. 1 (cell)credit/cell-time. Our simulations show only marginal benefits in increasing R beyond this value.

We assume that each credit or grant scheduler implements pointer-based round-robin scheduling. 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 [3] –see section 3.3. With increasing buffer size per output B, performance improves fast.

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[5]. This means that two or more input (grant) schedulers may select a grant/credit from the same output at the same time, thus, multiple cells may reach an output buffer in the same time.

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 5. 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 the corresponding output buffer 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 [6].

3.3 Operation and Throughput Of Independent & Asynchronous Schedulers

When B=1 cell, our credit scheduling disallows output conflicts, by having at most one credit pending at a time, per output. Thus, no output buffers are actually needed in this case, and the crossbar behaves like bufferless. However, a complete pipeline operation, comprising both (output) credit and (input) grant scheduling, has to complete within time-slot boundaries –$SD \leq 1$ cell time. (It is easy to see that if SD equals two cell times in this bufferless crossbar, its peak throughput will be equal to 50%.) In each such time-slot operation, our credit and grant schedulers "find" a different bipartite graph matching. This process for scheduling looks very much alike crossbar schedulers that use independent input and output link schedulers, and multiple iterations of handshaking between them to improve match size. These schedulers have been studied extensively for more than a decade now [1] [2] [3] [19].

Acceptable performance, using a single iteration, is only achieved when crossbar schedulers fix in some effective matching sequence. For example, desynchronization, as first presented in the *i*SLIP switch, makes each output scheduler always granting a different input scheduler in each time-slot, and different than all other output schedulers, even though all schedulers work independently; however, desynchronization is accomplished only under a limited set of VOQ loads, and, anyhow, it introduces large delays until schedulers find their proper arrangement.

Normally, these algorithms need many "iterations" of handshaking between the independent output and input schedulers, in order to achieve good matches, as, in each time-slot that two or more output (credit) scheduler happen to grant the

[5] a grant/credit is said pending from the time it is generated by its output credit scheduler, until it is returned back to the credit scheduler, i.e. after the corresponding cell departs its output queue; when credit prediction is used, credits return faster –see sec. 3.2.

[6] when R=1 credit/cell-time, which is the default value in this paper, the need for the predict credit counter is removed: each credit scheduler will always allocate only one new credit per-cell-time.

same input, throughput is directly wasted. In each such time-slot, there must be some unmatched input, thus an input with *no* grant/credit to use, thus an underutilized input line, and therefore an underutilized output line –since the number of inputs is equal to the number of outputs.

For instance, in PIM-like crossbar schedulers, inputs request all outputs for which they have cells. Scheduling starts with each output (independent) scheduler granting some input request, and an input-output match is added when some input (indepedent) scheduler accepts an output grant. In these schedulers, all the grants that conflict on some input, except the one that is "accepted", are normally "dropped". This happens as it is either inefficient, or not safe, to store these previous "not-accepted" decisions. Consider that one such grant, name it g, produced by output o, was stored, instead of being dropped. If output o is *not allowed* to produce a new grant (after g) before the g gets accepted by its input scheduler, then, that output will stay underutilized for all that time; on the other hand, if output o is *allowed* to produce a new grant even before g is accepted, then, we run the danger of both grants being accepted concurrently by their input schedulers, thus violating the (bufferless) crossbar constraints. For this reason, not-accepted grants are normally dropped and many iterations or speed-up are used to increase match size, essentially to match those schedulers whose previous decisions have been dropped[7]. By contrast, our architecture, for any B, never drops a grant or request, thus, every decision of a link scheduler will become "operational" immediately or in the short run[8]. In this way, fewer decisions are needed in order to occupy the output lines with same data.

In our system, with large buffers ($B \geq$2 cell time), each single resource scheduler operates independently within time-slot boundaries, producing a new valid "outcome" in each time-slot (SD=2 cell times), without needing to communicate anything with other schedulers in the middle of this cell time operation. Scheduling time-slots, in different schedulers, are essentially asynchronous with each other. Each scheduler picks the state communicated by other schedulers before starting a new operation, and communicates its new "outcome", when that new operation finishes. The scheduler (or schedulers) being notified of this new "outcome" may be in the middle of a scheduling operation, as schedulers operate asynchronous of each other; thus, it will buffer its new "input", and will use it (i.e. produce a new outcome taking that new input under account), in its next scheduling slot, after finishing the current one.

[7]pipelined schedulers for bufferless crossbars [12] [13] reduce timing constraints, but can get quite cumbersome to build, as they normally need to incorporate multiple scheduling iterations for each matching produced. Instead, our pipelining is as simple as it can get, and it does not use speedup. The intrinsic difficulty with bufferless crossbar schedulers relative to our scheduler for multi-cell output buffers should be tracked down at the problems that the two schedulers try to solve: it is much easier to build an approximate match than an exact match.

[8]There is no problem with that, since, each such grant has space reserved to fit inside the output buffer whenever it gets accepted by its input grant scheduler.

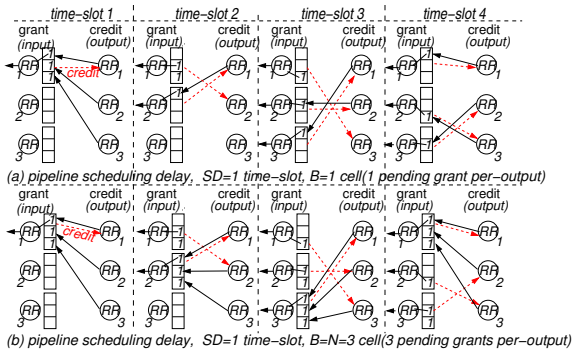

Fig. 6. (a) Desynchronization of output pointers yields 100% throughput when B=1; (b) Buffering yields 100% throughput when B=N. The request queues (not shown) are assumed to be continously non-empty. The figure assumes that credit prediction is employed, so that a credit is replenished just after a grant is selected by its grant scheduler.

Performance improves sharply with increasing B, as schedulers do not have to halt after their first conflicting decision –i.e. when the replenishment of a credit delays– , but, instead, they can use their additional credits to continue serving inputs.

Schedules become feasible thanks to the flow control of the *output* buffers: during any time interval T, the number of cells injected for a given output, from any number of inputs, will always be $\leq \lambda \cdot T + B$. Consider that, if buffers were not associated with outputs, (a) their flow control would not control the congestion of the output lines, and inadmissible traffic could monopolize the input queues inside the crossbar; and (b) exact crossbar matches would still be need.

Summing up, buffer space before the output ports of the fabric acts as a conflict tolerance parameter that increases flexibility, allowing us to build approximate crossbar matches, using independent and pipelined single-resource schedulers with multi-cell-time pipeline delay; the schedules produced by these schedulers may include conflicts in the short term but are feasible and efficient in the long run: in this paper, we show that, without any speedup, this architecture performs very close to ideal, with only 12 cells per output[9].

3.3.1 Single-Cell Buffers: Deterministic Desynchronization

Based on [19], we prove in this section that, under uniform cell arrivals, the throughput of the switch that we propose in this paper, with SD=1 time-slot, B=1, and pointer-based round-robin schedulers, can reach 100%. (This result applies for any propagation delay P, if we employ credit prediction.) A critical assumption is that *all* output credit schedulers use a common ordering of inputs, when they "search" which input is next to serve. To see why, first consider that when B equals one (1), any particular credit scheduler may have only one input granted at any given time –it can produce a new grant only after it is notified that its "first" grant has been accepted[10]. If this output grant is not selected by the grant scheduler, it will reside in its grant queue, waiting to be served, which is

[9]we have made extensive tests for B equal to {1,2,5,7,8,12,16,32}.

[10]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).

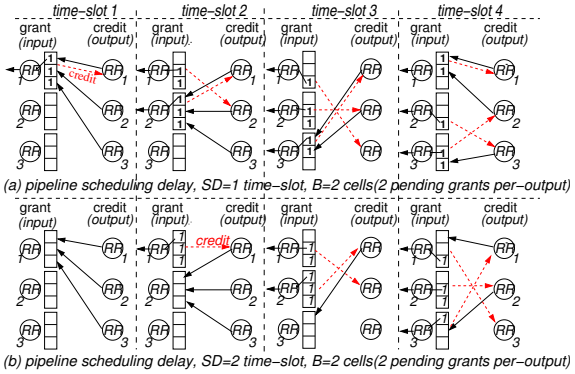

Fig. 7. Desynchronization for intermediate B values; (a) for SD=1; and (b) for SD=2. The request queues (not shown) are assumed to be continously non-empty. The figure assumes that credit prediction is employed, so that a credit is replenished just after a grant is selected by its grant scheduler.

equivalent to what happens in *i*SLIP –and, symmetrically in DRRM: *i*SLIP, instead of storing not-accepted grants, cancels them, but reproduces them in subsequent time-slots until these get accepted.

More formally, 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:

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

where $i - 1 = (i + N - 1) \bmod N$. Then, from [19], 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.

3.3.2 Multi-Cell Buffers: Statistical Desynchronization

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. The problem lies in that we cannot easily identify the input that an output will grant after it receives a credit back from a grant scheduler, since it may have already granted several subsequent inputs utilizing the second available credit.

However, our simulation results indicate that 100% throughput is still achieved with B>1. Actually, the practical system with SD=2 time-slots and B=2 cells, outperforms the one for which we have proved the 100% capability (SD=1, B=1) –see figure 13 in section 5.

As B grows further, delay and throughput improve. The improvement is sharp as B increases from 1 to 4 cells, sligltly smaller as B increases from 4 to 12 cells, and marginal thereafter. With large output buffers, desynchronization of output pointers is not that crucial. For comparison, figure 6 depicts the two extreme cases: (a) when B=1, full utilization is achieved thanks to desynchronization; (b) when B=N, even if all output credit schedulers circularly visit inputs, *in full synchrony*, full utilization can be achieved. As another example of how buffers tolerate conflicting decisions, consider a buffered crossbar switch with one cell buffer per-crosspoint (hence N cell buffers per output), and round-robin input and output schedulers. In this architecture, clashing input pointers may produce output conflicts, however this does not seem as a spot of significant bother: a tagged cell, conflicting with as many as $N-1$ other cells, may have to wait $N-1$ time-slots at its crosspoint; in the meanwhile that cell's input can feed other outputs, returning to the crosspoint of the tagged cell just after that empties (and so on).

For intermediate B values, between 1 and N, the credit schedulers must desynchronize at some extend to achieve full throughput. Figure 7.a shows how with B=2 and N=3, the credit schedulers desynchronize to the extend needed for full utilization; and figure 7.b shows the same behavior, when the pipelined scheduling delay (SD) equals two time-slots. (The 100% throughput capability under moderate buffer sizes is further substantiated through simulations in section 5.)

It should be noted here that, what we call desynchronization under moderate buffers ($\geq$ 1 cell) is not a deterministic, but rather a *statistical* argument. With large output buffers, thus many credits available, each input is expected to have some grant in its grant queues. Since there are $N \cdot B$ total credits in the system, and only N inputs, it can easily be the case that a "busy" input, that did not receive a new grant in some time-slot, will very much probably have some credit from previous time-slots. To this end, we do not need to fix the output pointers in some specific arrangement, as when B=1, or as in *i*SLIP and DRRM. Instead, we simply must avoid persistent degenerate credit distributions, where plenty of output credits get reserved for only a few inputs.

4 . Throttling Grants to a Bottleneck Input

According to our simulation results (section 5), our switch, as described so far, performs very well under smooth (Bernoulli) arrivals with buffers of 4 to 12 cells per-output, independent of the propagation time, under both uniform and non-uniform (unbalanced) traffic. This section discusses system operation under bursty traffic (correlated cell arrivals).

If multiple credit schedulers allocate credits to a grant scheduler for the 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 –remember that each grant scheduler selects one grant per time-slot. Such accumulations of credits in front of grant schedulers correspond to underutilization of the common pool of available credits, hence also underutilization of the available buffer space. Accumulations of this kind may occur even under smooth cell arrivals, but tend not to be too severe, as (output) credit schedulers quickly alternate among the inputs to which they grant, and, because the reserved credits return fast enough to their output credit scheduler, as the variance of the load experienced in different inputs (and different outputs) stays small, due to the smoothness of traffic.

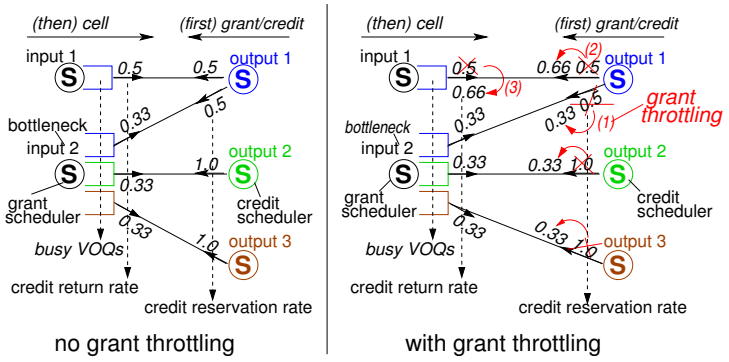

Fig. 8. The VOQs shown are assumed to be persistent, while the rest of VOQs are assumed to be empty. (a) shows the performance of our system under this unbalanced transient with input 2 constituting a bottleneck, when no grant throttling is employed, and (b) shows how we want grant throttling to work.

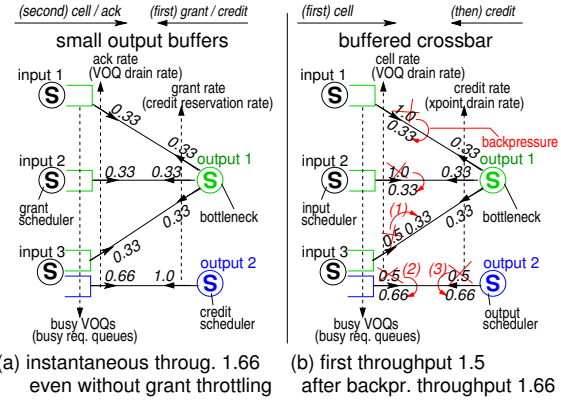

Fig. 9. The VOQs shown are assumed to be persistent, while the rest of VOQs are assumed to be empty. In the depicted traffic scenario, output 1 constitutes a bottleneck. Our system, shown in (a), directly achieves the desired instantaneous rates, whereas, the buffered crossbar, shown in (b), may delay the proper rates until the crosspoint queue of connection 3→2 fills up.

Under bursty traffic, however, requests may have to wait for quite a long time before being served, due to multiple bursts, from different inputs, targeting the same output at about the same time. An input with many such requests pending at multiple outputs, is said congested, and may receive multiple grants at the same time, which will accumulate inside its grant queues, as described above. 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 switch load (higher than 0.9); under the same traffic, buffered crossbars achieve ideal performance.

4.1 Unbalanced Transients with Congested Inputs

Since each independent credit scheduler is informed about the state of the VOQs targeting its associated output but has no direct information whatsoever regarding other VOQs, the overall credit allocation cannot be optimal in the short term. For one, credit schedulers, being oblivious of immediate input contention, allocate credits as if all inputs with non-empty request queues are equally loaded. Under probabilistic traffic, however, inputs are not always equally loaded. Even under otherwise uniform traffic patterns, there can be "moments" when some VOQs are more loaded than other, and, analogously, there can be "moments" when some inputs are more loaded than other inputs, thus constituting a bottleneck. We name such dynamic states after *unbalanced transients*. As discussed above, one may fairly guess that, even though such transients can also appear under smooth arrivals, they will be more notable when arrivals are bursty. The problem is that during such unbalanced transients, credits may accumulate in front of the grant scheduler sending grants back to the congested input at rate 1 grant per time-slot. Obvioulsy, this is the rate that the accumulated credits become available again, and this constitutes a problem, because these credits "steer" the traffic to as many as N outputs.

For instance, consider that the credit scheduler for a tagged output, say output 1, steers credits evenly among two inputs with non-zero request queues. If one of these inputs, say input 2, has many cells destined elsewhere, say for two additional outputs, whereas the other input, say input 1, has cells only for output 1 –an unbalanced VOQ demand with input 2 constituting a bottleneck–, credits will accumulate inside the grant queues for input 1.

Figure 8.a depicts roughly the performance of our switch during the aforementioned unbalanced transient with the congested input. As the figure shows, in our system this transient might bring idling output slots. The figure shows this inefficiency as induced by credit reservation rates. In the very end however, the main problem must be tracked down to the following dynamic behavior[11]: whenever output 1 has some credit available, with probability 1/2 it reserves it for input 2 and with same probability for input 1. However, as the credits from input 2 return 3 times slower than from input 1, and output 1 continues credit reservations using the same (1/2) probabilities, in the end, it can be the case that input 1 usually contains no credit at all for output 1; under this condition, input 1 gets a credit chance of probability 1/2 once in every k time-slots, where $k \approx 3$. As we describe next, this inefficient credit allocation cannot persist.

If demand persists in that same way, request queue 2→1, i.e. that of the bottlenecked input, will empty even if VOQ 2→1 grows, as: initially, by assumption, request queue 2→1 drains at rate 1/2. Being loaded by two additional output credit schedulers, grant scheduler for input i will serve grant queue 2→1 at rate 1/3, thus VOQ 2→1 will receive grants at rate 1/3. Due to request queue flow control, this rate, 1/3, upper bounds the rate of new 2→1 requests (see section 2.2). Now, since its drain rate (1/2) exceeds its load (1/3), request queue 2→1 will eventually empty, and will stay close to being empty for as long as this VOQ state persists. It follows then that the credit scheduler for output 1 will drop the credit rate for input 2 at 1/3, increasing at the same time the credit rate for input

[11] if the problem was that input 1 did not receive credits fast enough, then increasing R would obviously improve performance. On the contrary, simulations showed that increasing R offers marginal improvement.

1 to 2/3, improving throughput[12]. However, the suboptimal allocation may last for some time if the drain time of the request queue is long, which can be the case when u is set large in order to compensate for a large propagation delay P. While the transient is "active", input (virtual-output) queues may marginally grow, and delay may increase[13].

(It is interesting to see that when the bottleneck is an output rather than an input, our system promptly delivers the peak instant throughput, whereas a buffered crossbar may lag behind the ideal until the backpressure is activated, that is until the credits available at the ingress linecards for the congested flows get exhausted[14] –see figure 9.)

4.2 Threshold Grant Throttling

Using these observations, we came up with following simple solution, which, instead of some sophisticated scheduling discipline, uses a type of grant queue "backpressure" that aims at responsively *throttling the grants routed to the bottleneck input*. The key idea is to make credit schedulers stop serving an input that does not return credits fast enough. In this way, credit allocations, besides to output contetion, also take input contention into immediate account; therefore, credit distributions mirror flows fair shares better. (The way that we want grant throttling to work is presented in figure 8.b.)

Grant throttling works 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 at a value below B, we can bring delay down to the levels of ideal OQ, using only 12 cell buffers per-output, and plain round-robin schedulers; these results apply for any switch size (N) in the range of 32 up to 256, and for a wide range of burstiness factors.

5 . 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 effects of initial transients –depending on switch size, input load, and burstiness factor, a few thousands, up to tens of millions of "start" cells where discarded, before gathering statistics–, and the confidence intervals achieved were better than 10% around the reported values with confidence 95%[15]. 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. Finally, to simplify the analysis of our first results, in this paper we assume that each VOQ may have a large number of pending requests –u is set equal to 1000 in all experiments.

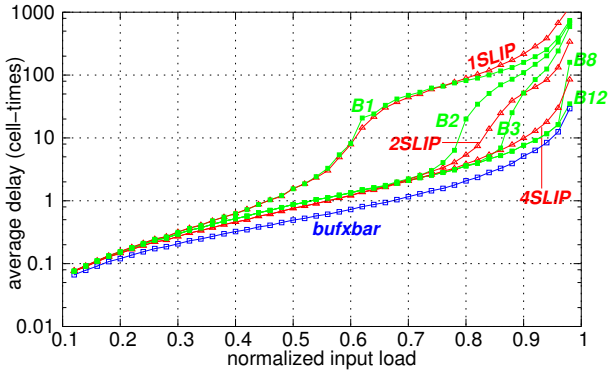

Fig. 10. Performance for varying buffer size, B; N=32, P=0, R=1 credit/time-slot, and SD=1 time-slot; Uniform Bernoulli cell arrivals; Only the queueing delay is shown, excluding all fixed scheduling and propagation delays.

5.1 Uniform Traffic

5.1.1 Effect of Buffer Size, B: 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. 10. $B1$ 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 ($bufxbar$) –see [3, fig. 3] shows that $bufxbar$ virtually matches OQ delay; under smooth arrivals, we found no benefit in further increasing B.

5.1.2 Effect of Credit Rate, R: Figure 10 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, i.e. R= 1 cell

[12] once again, we used a rather rough rate description. The actual phenomenon can be more intricate.

[13] when the long-term traffic is uniform, the long-term throughput will not be affected, as: if such VOQ increments are constantly added in a sequence of such transients, all VOQs will end up being persistent, hence unbalanced transients will disappear and VOQs will stop growing further.

[14] such unbalanced transients do not hurt the performance of buffered crossbars because credits are private to each flow, thus, the conflicting cells are stored at their crosspoint, and do not disturb the flows destined to other outputs; on the contrary, these conflicting cells relief cells targeting other outputs, by not contenting at the input any more. In our system, under congested inputs, conflicting grants are similarly stored inside private (per connection) grant queues; but, by delaying the reuse of the *shared* output buffers, the grants for the congested input implicitly interfere with other inputs, and might even slow down the flow of traffic through them.

[15] in throughput experiments, confidence intervals where better than 1%

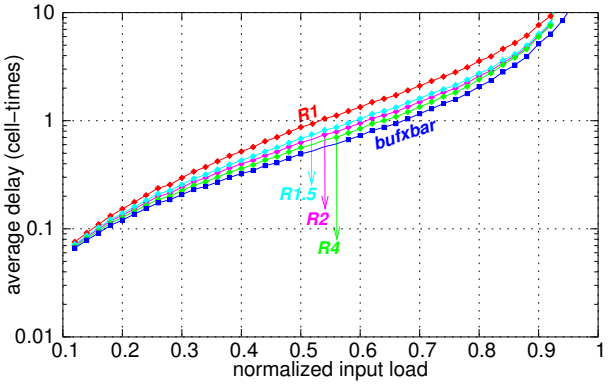

Fig. 11. Performance for varying credit scheduler rate, R; N=32, P=0, SD=1 cell time, and B=4 cells; Uniform Bernoulli cell arrivals; Queueing delay shown, excluding all other fixed delays.

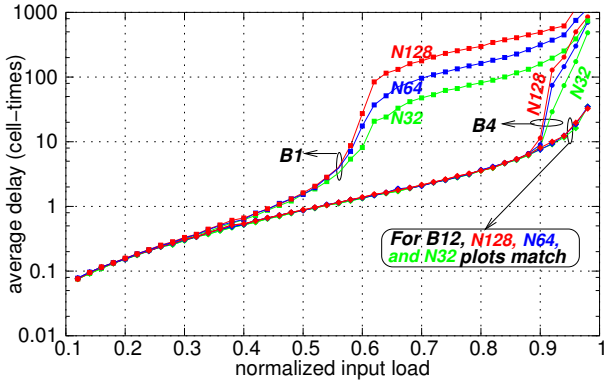

Fig. 12. Performance for varying sw. size, N; P=0, R=1, and SD=1; Uniform Bernoulli cell arrivals; Delays excludes all fixed delays.

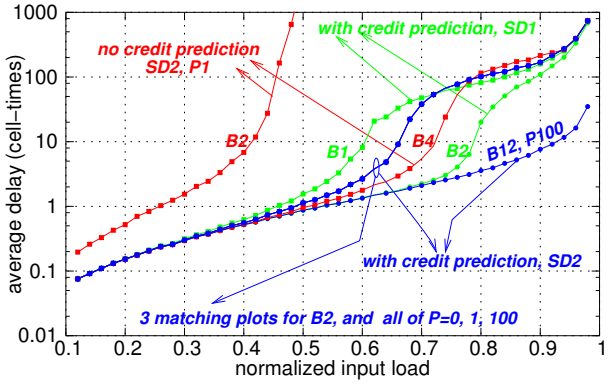

Fig. 13. Performance for varying scheduling delay SD, and varying P, using credit prediction, or without credit prediction; N=32, R=1 credit/cell-time. Uniform Bernoulli cell arrivals; Delays excludes all fixed delays.

(credit) per-cell-time in fig. 10; 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.

To support this argument we increase the rate R, at which each independent credit scheduler operates[16]. Figure 11 shows our results. For $R > 1$, the cell delay at medium loads approaches buffered crossbar, because of more cells skipping input contention. Under high load, increasing R above 1 credit/time-slot, offers only very marginal improvements in performance.

5.2 Under High Load Credits are Usually Pending

We have observed that, when the load is high, soon credit schedulers have all their credits reserved. This happens due to random grant conflicts, which, unavoidably, delay the return of credits; therefore, as credit schedulers are greedy, soon their whole available credit resides in grant queues, or in the output buffers. This explains why increasing the rate R does not improve the delay at high loads. Simply, such increments are not functional, but only when the load is low, in which case, as credits are usually available, a faster credit scheduler can serve the "rare" sets of inputs that request its service at about the same time faster.

5.2.1 Effect of Switch Size, N: In fig. 12 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 [3], 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, 256}, 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.

5.2.2 Effect of Propagation and Scheduling Delays: Figure 13 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 delay, 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 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 13 shows that, when P=100, SD=2, and B=12, the queueing delay is very close to $bufxbar$ –compare to fig. 10. A 32 × 32 buffered crossbar switch,

[16] 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 B large enough, otherwise the effective credit scheduler rate is limited by the rate that credits are released, i.e. 1 credit/cell-time

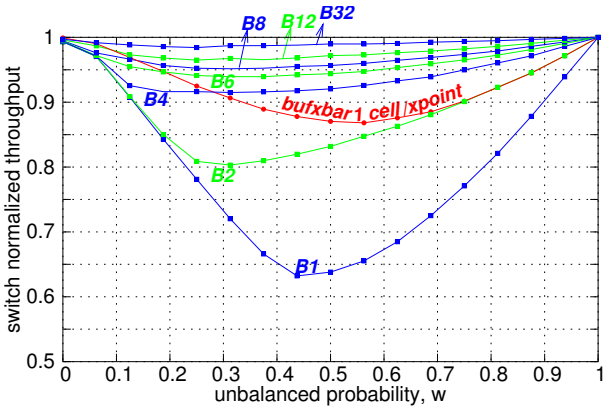

Fig. 14. Throughput under unbalanced traffic for varying buffer size, B; N=32, P=0, R=1 credit/cell-time, and SD=1 cell-time; 100% input load.

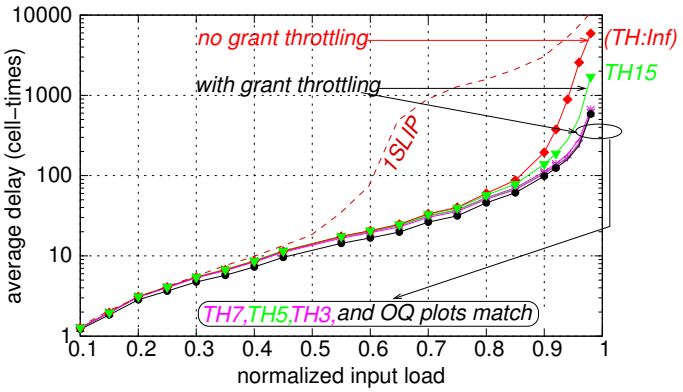

Fig. 15. Performance under bursty traffic, for different grant queue thresholds, TH; N=32, B=12, P=0, R=1 credit/cell-time, and SD=1 cell-time; average burst length 12 cells.

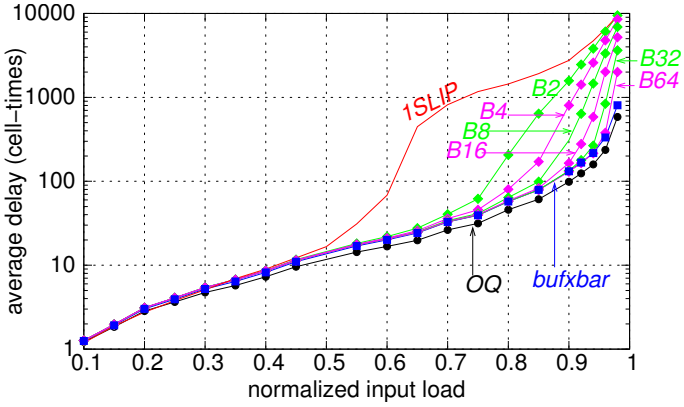

Fig. 16. Performance under bursty traffic, for different buffer sizes, B, *when no grant throttling is used*; N=32, P=0, R=1 credit/cell-time, and SD=1 cell-time; average burst length 12 cells.

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.

5.3 Unbalanced Traffic

Next, we measure switch throughput under unbalanced traffic. We borrow the unbalanced traffic model of [6], where each input, i, sends most of its traffic to a private "favored" output –in our experiments to output i. As in [6], 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. 14, 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.

The intuition that throughput increases with buffer size can be better elucidated under this traffic model. Consider a heavily loaded connection, $f_{i \to i}$. Under unbalanced traffic with 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 connections are not active. A large output buffer (B) 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, with one cell buffer per-crosspoint, is relative low due to memory partitioning: even though the crossbar contains 32 cell buffers per output, each heavily loaded connection can access only its private, single-cell, crosspoint queue.

5.4 Bursty Traffic

Finally, we present results for bursty traffic. Bursty traffic is based on a two-state (ON/OFF) Markov chain. ON periods (consecutive, back-to-back cells arriving at an input for a given output) hold for at least one (1) cell-time, whereas OFF periods may last zero (0) cell-times, in order to achieve 100% loading of the switch. The state probabilities are calibrated so as to achieve the desirable load, giving exponentially distributed burst length around an average which in this experiment equals 12 cells (close to the average packet size in the Internet).

As figure 15 shows, when no grant queue throttling is employed, delay increases at high switch loads. Our results indicated that increasing the buffer size per-output above 12 cells alleviates the delay degradation, though large buffers are needed to approximate the ideal output queueing performance. However, as the figure shows, by using a grant throttling threshold below the buffer credit available per output (B), the delay degradation essentially drops down to zero. We have verified that this results apply for any switch size N up to 256.

Figure 16 demonstrates that, simply increasing the buffer size per output, without applying any grant throttling, will not improve performance; as the figure shows, at high very loads, delay increases above the ideal output queueing even for B=64. Instead, with grant throttling, a buffer size (B) of

12 cells suffices to bring the delay down at the levels of pure output queueing –see fig. 15.

6 . 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. We also discussed the asynchronous operation of this switch, its throughput, and the harmful accumulations of credits during unbalanced transients, as well as a method to prevent them. 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 [20] [16] can be used in our switch in a scalable way.

7 . 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] T. Anderson, S. Owicki, J. Saxe, C. Thacker: "High-Speed Switch Scheduling for Local-Area Networks", *ACM Trans. on Computer Systems*, vol. 11, no. 4, Nov. 1993, pp. 319-352.

[2] R. LaMaire, D. Serpanos: "Two-Dimensional Round-Robin Schedulers for Packet Switches with Multiple Input Queues", *IEEE/ACM Trans. on Networking*, vol. 2, no. 5, Oct. 1994, pp. 471-482.

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

[4] 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.

[5] 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.

[6] 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.

[7] 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/

[8] G. Georgakopoulos: "Few buffers suffice: Explaining why and how crossbars with weighted fair queuing converge to weighted max-min fairness", http://archvlsi.ics.forth.gr/bufxbar/

[9] 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

[10] 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.

[11] 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/

[12] 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.

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

[14] 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.

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

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

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

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

[20] M. Katevenis, P. Vatsolaki, A. Efthymiou: "Pipelined Memory Shared Buffer for VLSI Switches", *Proc. of the ACM SIGCOMM'95 Conf.*, Cambridge, MA USA, Aug-Sep. 1995, pp. 39-48; http://archvlsi.ics.forth.gr/sw_arch/pipeMem.html