Congestion Management for Non-Blocking Clos Networks

Nikolaos I. Chrysos
Institute of Computer Science,
ICS, FORTH – Hellas
member of HiPEAC
http://archvlsi.ics.forth.gr
nchrysos@ics.forth.gr

ABSTRACT

We propose a distributed congestion management scheme for non-blocking, 3-stage Clos networks, comprising plain buffered crossbar switches. VOQ requests are routed using multipath routing to the switching elements of the 3rd-stage, and grants travel back to the linecards the other way around. The fabric elements contain independent single-resource schedulers, that serve requests and grants in a pipeline. As any other network with limited capacity, this scheduling network may suffer from oversubscribed links, hotspot contention, etc., which we identify and tackle. We also reduce the cost of internal buffers, by reducing the data RTT, and by allowing sub-RTT crosspoint buffers. Performance simulations demonstrate that, with almost all outputs congested, packets destined to non-congested outputs experience very low delays (flow isolation). For applications requiring very low communication delays, we propose a second, parallel operation mode, wherein linecards can forward a few packets eagerly, each, bypassing the request-grant latency overhead.

Categories and Subject Descriptors

C.2.6 [**Computer-Communication Networks**]: Internetworking—*routers*

General Terms

Algorithms, Design, Performance

1. INTRODUCTION

The demand for high-radix switching fabrics is driven by the market for Internet routers, server systems, data-centers, SAN clusters, and high-performance computers. Beyond 32 or 64 ports, single-stage crossbars switches are quite expensive, and *multi-stage switching fabrics* become preferable. Multi-stage fabrics are made of multiple, smaller-radix crossbars, and can provide the same non-blocking performance with their single-stage counterparts at significantly lower cost. In this paper we consider *buffered, 3-stage Clos fabrics* [1], and we study how to provide quality-of-service (QoS), using inherently distributed, and pipelined control. Our objective is robust operation during overload periods, with fair (and full) allocation of congested link bandwidth, and minimized latency for lightly loaded destinations.

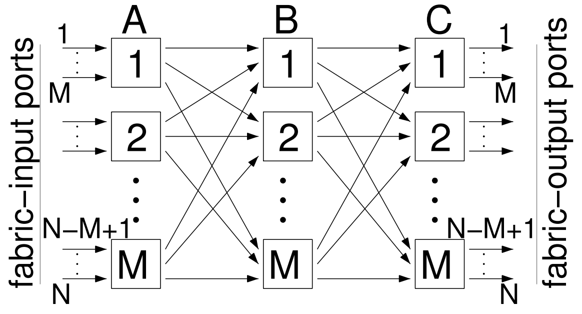

Figure 1: A non-blocking, 3-stage, Clos fabric, with no internal speedup;m=n=M;N=$M\times M$.

1.1 Three-stage (non-blocking) Clos fabrics

Practical non-blocking fabrics are made using 3-stage Clos networks, shown in Fig. 1. By A, B, and C, we denote the 1st, 2nd, and 3rd fabric stage, respectively. One of the parameters of Clos networks, m/n, controls the *speed expansion* ratio –something analogous to the "internal speedup" used in combined input-output queueing (CIOQ) switches: the number of middle-stage switches, m, may be greater than or equal to the number of input/output ports per 1st/3rd-stage switch, n. In this paper, we assume m=n=M, i.e. *no speedup*: the aggregate throughput of the middle stage is no higher than the aggregate throughput of the entire fabric. In this way, the fabrics considered here are the *lowest-cost practical non-blocking fabrics*, also referred to as *Benes* [3]. For non-blocking performance, we use *inverse multiplexing (multipath routing)* [2]: each input-output pair flow is distributed among all middle-stage switches, so as to equalize the rates of the resulting sub-flows. In effect, if the load is feasible at ports, it will also be feasible at any internal link.

1.2 The problem: output port contention

Output contention is a primary source of the difficulties in multi-stage networks: input ports, unaware of each other's decisions, may inject traffic for specific outputs that exceeds those outputs' capacities. The excess packets, which must wait in buffers, may prevent other packets from moving toward their outputs, thus leading to poor performance.

Bufferless fabrics overcome this difficulty by holding these excess packets at the ingress linecards, in virtual-output-queues (VOQs) implemented using large, off-chip memories: as no packet ever halts inside the fabric, the "road is always clear" for new connections. This congestion-free property of bufferless networks comes however at the expense of either (potentially heavy) packet loss, or, for lossless operation, at the expense of a far too complicated central scheduler, which admits one packet per output, per packet time [4].

In our view, *buffered fabrics* are preferable because they require simpler control, and because they can offer higher performance compared to their bufferless counterparts. The internal buffers can be made small, so as to avoid off-chip memory at the switching elements and limit delays through the fabric. In order for the small buffers not to overflow, *backpressure* must be used. *Indiscriminate* backpressure stops all flows sharing a buffer when that buffer fills up; it leads to poor performance due to buffer hogging and head-of-line (HOL) blocking. *Per-flow* buffer reservation and backpressure signaling overcomes these shortcomings by isolating congested from well-behaved packets in separate buffer and queue resources per flow [5], or per output [6]. A recently proposed compromise is to *dynamically* allocate *set-aside queues (SAQs)* for the presently congested outputs only [7]. This solution is cost-effective for up to a few concurrent congested outputs. However, for every "new" congested output, additional resources are needed.

In this paper, we adopt an alternative strategy. Instead of allowing inputs to eagerly forward traffic, we coordinate inputs' injections, in order to prevent excessive packet contention inside the fabric [8] [9] [10]. This coordination resembles the scheduling function in bufferless fabrics, but is much easier here because some amount of packet conflicts can be tolerated inside the fabric buffers.

Next, we ask ourselves the following question: what type of schedulers would be appropriate for Clos networks? Crossbar schedulers typically employ a "crossbar-like" interconnect between input and output *arbiters*. This approach may be suitable for single-stage crossbars, but does not scale to the port counts achievable by multi-stage networks. What we need here is a scheduling interconnect that scales as well as the Clos data network that we wish to schedule, thus a Clos scheduling interconnect.

2. BACKGROUND

In [9], we proposed an alternative congestion management for non-blocking, 3-stage Clos fabrics. Before injecting a packet into the fabric, input ports have to first request and be granted permission from a control unit. This scheme obviates the need for per-flow data queues, while providing excellent protection against congestion expansion, by enforcing the following rule.

The packets inside the fabric that are destined to a given fabric-output never exceed the buffer space in front of that output.

Figure 2(a) depicts an abstract model of a 3-stage Clos network. Each buffer shown is assumed to be in front of an internal or fabric-output link. As the left figure shows, indiscriminate backpressure may stop packets targeting the filled buffer in front of output 1; these stopped packets, outstanding inside intermediate buffers as they are, block other packets behind them, spreading congestion out. As shown in Fig. 2(b), our congestion control rule eliminates congestion in the sense that the data entering the fabric steadily move to the buffer of their output: output buffers never fill up to the point where data already inside the network cannot proceed into them, and they never exert backpressure on the upstream (2nd) stage. This avoids HOL blocking in 2nd stage. Combined with inverse multiplexing, this rule additionally guarantees that intermediate buffers only rarely fill up (only due to unlucky routing), thus preventing HOL blocking throughout the fabric –see [9] or [10, Sec. 4.2].

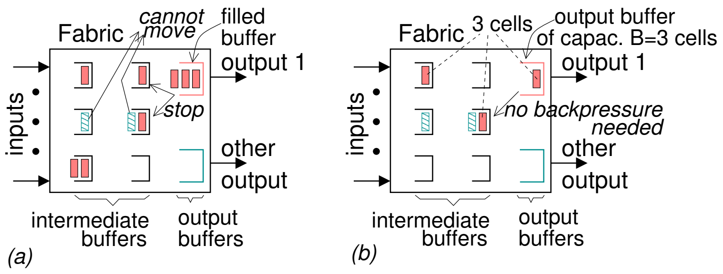

Figure 2: Our method against congested outputs.

Observe that this congestion control does not eliminate packet conflicts. Our scheme benefits from the fact that input-output matches do not need to be exact, since thanks to internal buffers, several inputs may match the same output at the same time. Such approximate matches can be produced by loosely coordinated independent schedulers, which makes it possible to pipeline the operations in the control subsystem as in buffered crossbars [11].

A comparison is appropriate here with the "akin" congestion management style in [8], which regulates the long term rate at output ports. Reference [12] extends that work with work conserving guarantees (in theory, a speedup of 2 is needed, but smaller speedups perform well in simulations). The systems in [8] [12] foresee a complex and lengthy communication and computation algorithm; to offset that cost, rate adjustments are made fairly infrequently (e.g. every 100 μs). Such long adjustment periods *(i)* hurt the delay of new packets arriving at empty VOQs; and *(ii)* do not completely prevent HOL blocking. Our scheme uses no speedup, and operates at a much faster control RTT, with much simpler algorithms, basically allocating buffer space, and only indirectly regulating rates. The result is low latencies and prevention of HOL blocking: for any number of congested links, non-congested packets are hardly affected [9].

Our initial work in [9] targets a 1024-port Clos fabric, made of 96, single-chip, 32-port buffered crossbar switches. It employs a *central control chip*, containing N output *credit schedulers*, as shown in Fig. 3(a). These schedulers allocate space in buffers in front of fabric-outputs to the requesting VOQs, thus enforcing the targeted congestion control rule.

2.1 Limitation of previous work

The system in [9] suffers from three serious drawbacks.

1. **Central scheduler:** The central control chip comprises $2 \cdot N^2$ request/grant counters (queues), N^2 distribution counters (pointers), and a $N \times N$ crossbar scheduling interconnect, with 1-bit lines. Another difficulty pertains to the I/O bandwidth of this central chip, which must accept N requests and issue N grants per packet time.

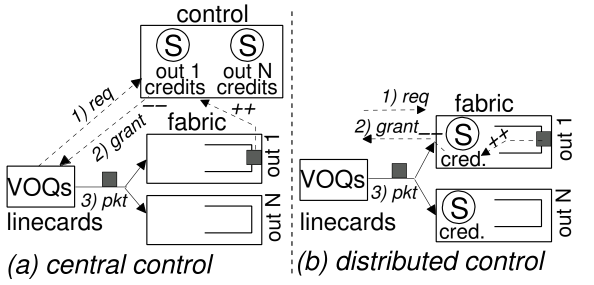

Figure 3: Alternative scheduler placements.

2. **Large crosspoint buffers:** In [9], each crosspoint buffer is sized proportionally to the end-to-end RTT. The RTT becomes a stringent constraint as it directly translates to on-chip memory, and in [9] it comprises 4 linecard-fabric signal propagation delays, which can be immense in multi-rack systems with fast links [13].

3. **Cold-start request-grant latency:** The request-grant adds a delay overhead to the fabric response time. For lightly loaded flows (i.e destinations), it is desirable to avoid this overhead in latency-sensitive applications, e.g. cluster/multiprocessor interconnects.

2.2 Contents & contribution

The present paper *distributes* the N credit schedulers evenly over the M switches in the 3rd-stage, as abstractly shown in Fig. 3(b). Requests (and grants) are routed to (and by) the credit schedulers through the Clos fabric using inverse multiplexing (Section 3). Thus, the new scheduling architecture alleviates the bandwidth and area constraints of the central control chip. What is rather challenging in this decentralized approach is the management of contention among the requests inside the scheduling network. If proper methods are not put to use, requests of congested flows can monopolize the queues in the scheduling network (just as without proper congestion management, congested packets can monopolize packet queues), reducing scheduling and data throughput. This paper identifies this problem, proposes feasible solutions, and studies performance (Section 3.5).

Regarding the second drawback of the architecture in [9], *(a)* in Section 3.4 we decouple the flow control of crosspoint buffers from that of reorder buffers, reducing the effective RTT by a factor of approximately two; and *(b)* in Section 5.1 we show that we can operate the system with sub-RTT crosspoint buffer space. The crosspoint buffers are now practically independent of the linecard-fabric propagation delay, and $\sim 1/4$ (new) RTT space for each suffices. Combining these methods, we reduce the memory requirements of [9] approximately eight (8) times.

Section 4 investigates methods that avoid the request-grant latency overhead. Each linecard is initially allowed to forward a few packets eagerly (without requests and grants) in order to minimize their delay; this privilege is renewed every time an *acknowledgment (ACK)* from one such packet arrives to the linecard. In addition, congested destinations issue congestion notifications, and inputs serve eagerly lightly loaded flows only, whose ACKs tend to return fast. This section also estimates the percentage of eagerly forwarded cells as a function of the ACK delay at load ρ.

Section 5 examines crosspoint buffer size under realistic packaging assumptions, for 10 and 40 Gb/s links; it also estimates the bandwidth and area overheads of the control subsystem. Finally, Section 6 concludes the paper.

3. SYSTEM DESCRIPTION

Throughout this paper, flows are identified by distinct input-output pairs, and schedulers implement the *round-robin (RR)* discipline. For simplicity, we consider only fixed-size packets, called *cells.* (When external packets have variable size, we can directly switch *variable-size* multipacket segments, thus eliminating the padding overhead, and the associated need for speedup [10, Sec. 5.2.5, 5.4.6][1].) The cell time is defined as the duration of a cell on an external or internal link. There is *no internal speedup.*

3.1 Cell datapath

Storage for cells is provided in VOQs maintained at ingress linecards. The data fabric is a $(N\times N)$ 3-stage Clos network, made of $(M\times M)$ buffered crossbar switches, as shown in Fig. 1. We consider small, on-chip crosspoint buffers; their size is denoted by b, and is measured in cells. Hop-by-hop, credit-based backpressure between adjacent fabric stages prevents buffer overflow. Crosspoint credits are sent upstream at a peak rate of 1 credit per cell time, per (local) input; outstanding credits are stored using M (=2 in Fig. 4) counters at each switch input, organized per switch output.

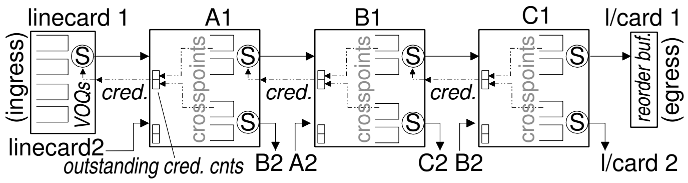

Figure 4: Cell datapath of the fabric.

3.2 Distribution of scheduling functions

We assume that the buffered crossbar switches contain the circuity required to implement the distributed functions of request-grant scheduling[2].

The core of the scheduling unit comprises N output *credit schedulers.* These single-resource schedulers (sometimes called link arbiters) are *distributed* evenly over the M switches in the last fabric stage: there are M of them in every C-switch, one for each corresponding fabric-output port –see Fig. 6. VOQ requests travel through the Clos network, from their ingress linecard to the C-switch that hosts the targeted output credit scheduler; grants travel the other way around.

The request-grant scheduling network, being incorporated inside the Clos fabric, is itself a three-stage (bidirectional) Clos. We route requests and grants using *inverse multiplexing*, much as we use inverse multiplexing to route cells in

[1]Each request-grant transaction always reserves space for a maximum-size segment (similar to cell operation in this paper), which is ≥ 2 minimum-size packets; upon receiving a grant, a VOQ injects a variable-size segment, with size that ranges from a minimum-size packet to a maximum-size segment. Reassembly is performed at egress linecards.

[2]This is not a requirement: the scheduling functions can also be distributed in separate (private) control chips, interconnected in a Clos arrangement.

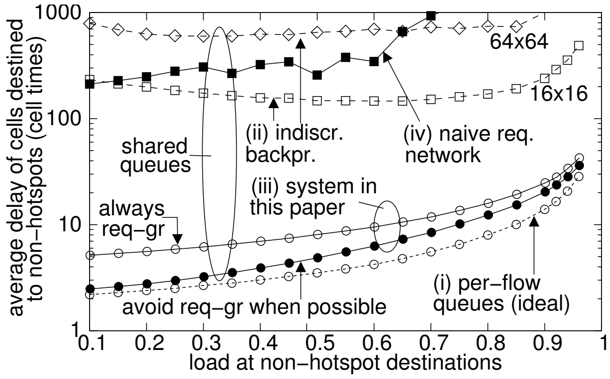

Figure 5: Delay performance in the presence of hotspots. In this paper, we present the average delay of several independent simulation runs, with 10% confidence intervals, and 95% confidence;in some plots, e.g. plot (ii) and (iv) here, we were not always able to obtain such confidence intervals. We gather results after a warm up period, which increases with the load, and switch size, $\sim$ 90 M cells for N=256.

the data network. Due to inverse multiplexing, each particular B-switch conveys requests from as many as N^2 flows. To isolate requests from different flows, we would normally need N^2 request queues (counters) inside each B-switch. We can circumvent this quadratic cost if we use *shared* request queues, managed by (indiscriminate) hop-by-hop backpressure. However, this alternative has the same drawbacks with indiscriminate backpressure in the data network.

3.2.1 Request contention management is needed

Figure 5 depicts the delay of cells destined to non-hotspot fabric-outputs (destinations), when the demand for (hotspot) destinations 1 and 2 is 1.1 cells per cell time. Non-hotspots receive uniform traffic, from all inputs, at a load which we vary along the x-axis. The figure has separate plots for *(i)* the ideal system that employs per-flow data queues[3] (N=16); *(ii)* a system with one queue per crosspoint managed using indiscriminate backpressure (N= 16, 64); *(iii)* the same as (ii), but with the scheduling mechanisms that we will present in this paper (N= 16); and *(iv)*, the same as (iii), but with naive (indiscriminate) flow control of request queues (N= 16). We can see that in systems (ii) and (iv), well-behaved cells suffer in the presence of hotspots, due to HOL blocking in the data and scheduling network, respectively. Additionally, the delay of well-behaved cells gets worse with increasing N, as shown for system (ii).

In system (iii), no HOL blocking appears in the data or scheduling networks, and well-behaved flows are virtually unaffected by the presence of hotspots. We have two plots for this system. In the first plot, we always use the request-grant protocol. This adds a request-grant latency overhead to every cell, rendering the delay at low loads approximately two times higher than that of per-flow queueing (minimum cell delay 4 vs. 2 in this experiment). In the other plot, we avoid the request-grant protocol for non-congested cells,

[3]There are as many (statically allocated) buffers per crosspoint, as the number of destinations reachable through the corresponding crossbar output, i.e. N^2, or $M \cdot N$, crosspoint buffers, in every A-switch, or B-switch, respectively.

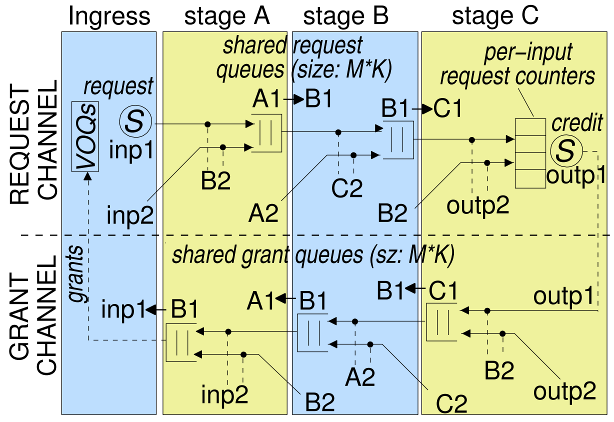

Figure 6: The request and grant channels.

using the methods that we describe in Section 4; now, the delay at low loads is minimized.

Not shown in Fig. 5 is the throughput at hotspot outputs. In all systems, except (ii), this throughput is full (100%). In system (ii), 16×16 fabric, throughput starts from about 75%, and increases almost proportionally with the load at the non-hotspots, approaching 100% at the right end of the x-axis[4]. Hence, with poor congestion control, the delay and throughput performance can degrade dramatically, even if we "pay" for a non-blocking network, as the Clos.

3.3 Scheduling architecture

Our scheduling architecture is depicted in Fig. 6. It comprises request and grant channels; the figure shows a single path from each, connecting an ingress linecard with an output credit scheduler. In reality, there are M such paths for each input-output pair, one per route (B-switch) in the fabric. Each link in the request (grant) channel transfers one request (grant) per cell time. As shown in the figure, requests from different inputs (and maybe targeting different outputs) are multiplexed on intermediate $A \rightarrow B$, and $B \rightarrow C$ links, and sink in per-flow request counters, in front of the targeted output credit scheduler.

3.3.1 Request channel

A request scheduler inside each linecard issues requests from non-empty VOQs. Consecutive requests from the same VOQ are routed through consecutive B-stage switches. The request scheduler limits the number of requests that a VOQ may have outstanding inside the scheduling network to an upper bound u (note[5]). Effectively, using this *hierarchical flow control*, the request counters in front of credit schedulers

[4]The reason for the poor utilization in the system with shared queues and indiscriminate backpressure must be the interference between the overloaded flows: when a buffer in front of a hotspot fills up, cells destined to the other hotspot can be blocked upstream. This behavior is more pronounced when the load for the non-hotspots is low, i.e. when hotspot cells dominate in fabric buffers; as the load of other outputs increases, the population of non-hotspot cells inside fabric buffers grows against that of hotspot ones, and the HOL blocking (between overloaded flows) diminishes.

[5]The linecard maintains a pending request counter, per flow. This counter is incremented when a request, per-flow, is sent, and decremented when a per-flow, grant is received.

will never wrap around (overflow) if they are at least $\lceil log_2 u \rceil$-bit wide, each. To support persistent connections, u must be ≥ 1 req_RTT, which spans from the time a linecard issues a request until it receives the grant (assuming no contention); for simplicity, req_RTT is measured in cell times.

As shown in Fig. 6, there is a *shared request queue* in front of each $A{\rightarrow}B$ or $B{\rightarrow}C$ link in the request channel, for a total of M queues in every A- or B-switch. Every such queue can accommodate $M \cdot K$ requests, and may accept up to M new requests per cell time, one from each input of the hosting switch. (To reduce queue speed, the shared request (and grant) queues can be partitioned per-upstream, with size K per subqueue.) We prevent overflow in these queues by maintaining M "request-credit" counters per $inp{\rightarrow}A$, or $A{\rightarrow}B$, link, all of which are initialized at K: an ingress linecard, or A switch, forwards a new request only if the corresponding request-credit counter is non-zero, and decrements this counter afterwards. For every request sent from stage A or B, a request-credit is generated, which may be stored in the present switch before being sent upstream, similarly with crosspoint (data) credits. Requests can freely depart from stage B, since space for them in stage C has already been reserved via the hierarchical request control.

3.3.2 Output credit scheduling

Upon reaching stage C, each request, say $i{\rightarrow}o$, increments the $i{\rightarrow}o$ request count, which is maintained in front of the credit scheduler for output o. The credit scheduler maintains M *credit counters*, initialized at b, one per crosspoint queue at its output, and N *distribution counters (pointers)*, one per flow arriving to this output. Initially, every per-flow distribution pointer points the same B-switch with its corresponding cell distribution pointer, maintained at the corresponding ingress linecards[6]. The output credit schedulers implement the *random-shuffle* round-robin discipline, which avoids severe scheduler synchronization[7] [10, Sec. 3.7].

Since we assume buffered crossbar switches, when the credit scheduler serves a flow, it must choose a particular B-switch (route), and reserve space in the corresponding crosspoint buffer. This route is selected by the distribution pointer of each candidate flow. Flow $i{\rightarrow}o$ is eligible for service at its (output) credit scheduler when its request counter is non-zero and the credit counter selected by the distribution pointer $i{\rightarrow}o$ is also non-zero. After service, this credit counter is decremented, the distribution pointer advances to the next B-switch, and a grant is issued to the corresponding input. This grant will be routed through the route selected by the distribution pointer[8]. Throughout this paper, each credit scheduler issues at most one grant per cell time.

3.3.3 Grant channel

The grant channel (reverse path) contains a *shared grant queue* in front of each ($C{\rightarrow}B$, $B{\rightarrow}A$, and $A{\rightarrow}inp$) link. These grant queues are managed using a "credit-based" type of flow control, similar to request queues[9]. Due to multipath grant routing, consecutive grants for the same flow may be routed out-of-order to the ingress side, but there is no problem with that since the per-flow grants are interchangeable with each other.

3.3.4 Cell injection

Upon receiving a grant issued from output o, ingress linecard i injects the HOL cell of VOQ $i{\rightarrow}o$. The route of this cell is given by the cell distribution pointer $i{\rightarrow}o$; after cell injection, this distribution pointer advances to the next B-switch. Observe that cell injection is subject to credit-based backpressure exerted from the crosspoint buffers inside the downstream A-switch, and thus it may not be possible at the time the grant arrives. Instead, the per-flow grants are registered, and a VOQ scheduler in each linecard forwards cells from granted VOQs, subject to credit availability.

Forwarded cells reach their C-switch subject to similar hop-by-hop ("local") backpressure. This local backpressure is indiscriminate (not per-flow), but as explained in [9] (and briefly outlined in Section 2), it does not introduce harmful blocking. When a cell leaves the fabric, it travels to its egress linecard, where cell resequencing is performed, using small, on-chip reorder buffers. Note that there is backpressure from stage C to stage B, in order to permit cell injections, without having first secured end-to-end credits.

3.4 Round-trip time & crosspoint buffer size

We can return the credit reserved for a cell, either when the cell departs from the C-switch (to the nearby credit scheduler), or when it becomes in order in its reorder buffer. Using the latter approach means that, by allocating buffer space for a cell in stage C, the credit scheduler also implicitly allocates space for that cell in the reorder buffer [9]; effectively, with space for $M \cdot b$ cells, the reorder buffer will never overflow. Unfortunately, this method increases the *data round-trip time (RTT)*, which is used to dimension crosspoint buffers, by the added delay C-switch to end of resequencing and back.

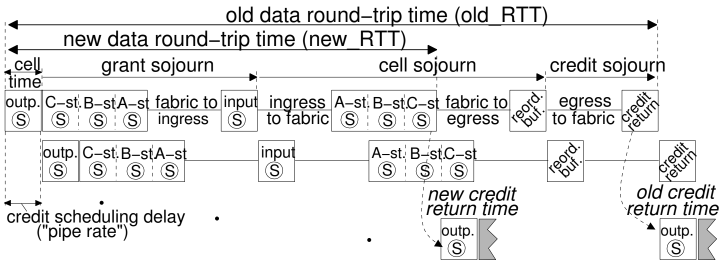

Figure 7: Pipelined scheduling operations & RTTs.

This data RTT (old_RTT in Fig. 7) spans from the beginning of a credit scheduling operation that reserves a credit, to the time that this credit returns back to its credit scheduler. It consists of credit scheduling delay, *plus* grant sojourn delay from stage C to ingress, *plus* cell sojourn delay from ingress to egress (reorder buffers), *plus* credit sojourn delay from egress linecard to C-switch (assuming no contention). For simplicity, we measure the RTT in cell times, and we refer to 1 RTT worth of space as 1 RTT buffer.

In large multi-rack systems, the signal propagation delay from (ingress or egress) linecard to fabric, or vice-versa, is

[6]We use coordinated load distribution [9], in order to eliminate route identifiers from grant messages.

[7]Each output scheduler, o, "scans" inputs (request counters $i{\rightarrow}o$, $i \in [1, N]$), using a private, random, but fixed order.

[8]This eliminates the need for grant distribution pointers.

[9]Effectively, one additional condition must be met when a credit scheduler issues a new grant: grant-credits must be available for the C-stage grant queue in the route selected by the distribution pointer of the candidate flow.

expected to surpass the other components of the RTT [13]. We can eliminate two such propagation delays from the effective data RTT, if we decouple the flow control of crosspoint buffers in stage C, from the flow control of reorder buffers. For this purpose, each output credit scheduler, besides the M credit counters used up to now, maintains one additional *reorder credit counter*. Both types of credits must be available in order to issue a grant. The decremented reorder credit counter is incremented when a cell becomes in order –for this purpose at most one "reorder credit" is sent upstream from each egress linecard per cell time–, whereas the crosspoint credit counter is incremented immediately when a cell departs from the fabric. Effectively, the reorder buffer needs one old_RTT space, and each crosspoint buffer one new_RTT space –see Fig. 7. In Section 5.1 we reduce crosspoint area even further, by allowing sub-RTT buffers.

3.5 Managing the contention among requests

The scheduling network consists of the request and grant channels. Both channels employ shared (not per-flow) queues, thus they are both vulnerable to HOL blocking.

For the following reasons, we do not need to worry about the grant channel. First, grants constitute a *second pass* through the fabric, after requests have traversed it once from ingress linecards to C-stage. The grant channel has the same capacity with the request channel, and the grants that use it will be no more than the requests that made it through the request channel: the request channel "filters out" many excessive requests. Second, in the worst-case, each credit scheduler may have up to $M \cdot b$ grants pending inside the grant channel. Considering all N outputs, this gives us a total of $N \cdot M \cdot b$ pending grants. On the other hand, the total grant queue space per switch is $M \cdot M \cdot K$ grants, for a total of $N \cdot M \cdot K$ grants per stage. Thus, for $K \geq b$, we expect no filled grant queues, thus no HOL blocking either. Indeed, our simulations have shown that grant queues are usually empty, even at high loads with many hotspots present.

Contention is more severe in the request channel, because requests are eagerly forwarded, and also because there can be many pending requests per flow. For congestion management purposes, limiting the maximum number of pending requests to 1 req_RTT (per flow) via the hierarchical request flow control has the following benefits: *(a)* with per-input request counters in stage C, there is no need for $B{\rightarrow}C$ request flow control; thus, there is no HOL blocking in $B{\rightarrow}C$ request queues (every non-empty $B{\rightarrow}C$ request queue always drains 1 request per cell time); and *(b)*, in the mid-term, the aggregate request rate for a specific output, from any number of inputs, is equalized to the grant rate of this output, i.e. $\leq$ 1 request (grant) per cell time.

Figure 8 depicts the delay of well-behaved cells in the presence of a varying number of congested fabric-outputs (hotspots). Hotspots are randomly selected among fabric-outputs, and each one is 100% loaded, uniformly from all inputs; h/• specifies the number of hotspots. For comparison, we also plot cell delay when no hotspot is present (h/0), i.e. uniform traffic. Cell arrivals are driven by independent Bernoulli processes. We have separate plots for our system, denoted by *distributed scheduled fabric (DSF)*, for indiscriminate backpressure, and for ideal per-flow queueing.

To see how well DSF isolates flows, observe that the delay of DSF-h/32 –delay of well-behaved cells when 32 outputs are congested– is very close to that of DSF-h/0, i.e. when

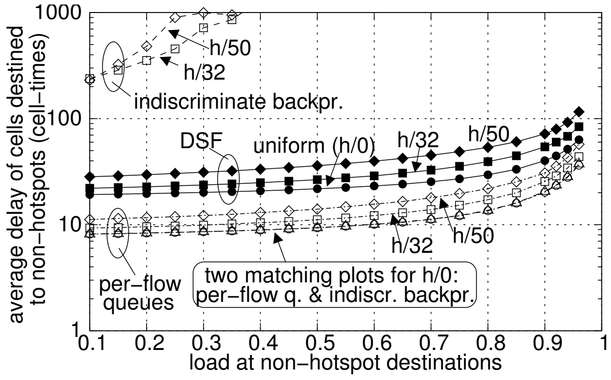

Figure 8: N=64, new_RTT=12; b=12,K=22,u=31; 48 cells reorder buffer per output.

traffic is uniform. The delay of DSF increases marginally when many hotspots are present, but this is not due to HOL blocking. Simply put, hotspot traffic increases the contention along the shared paths inside the fabric and the scheduling network. This effect is also present in the per-flow queueing system, and was also observed in [9]. (Observe that the delay of DSF-h/50 at 0.1 input load is approximately equal to the delay of DSF-h/0 at 0.8 input load; this happens because in h/50, at 0.1 load for non-hotspot destinations, the effective load at which inputs inject traffic is $\frac{0.1 \cdot 14 + 50 \cdot 1}{64} \approx 0.8$. At low loads, DSF delay is higher than that of per-flow queues, due to the request-grant latency.

Hence, we see that DSF operates robustly even when almost all fabric-outputs are oversubscribed. This happens because there is no HOL blocking in the data network, and also because there is no HOL blocking in $B{\rightarrow}C$ request queues, thanks to the hierarchical request control. However, as we describe next, under particular (rare) situations, HOL blocking may develop in $A{\rightarrow}B$ request queues.

3.5.1 Congested request links: problem & solutions

Multipath request routing brings the average load for each individual $A{\rightarrow}B$ request link below 1 request per cell time; it can be above that only in the short-term, due to unlucky routing decisions, but is not affected by output contention.

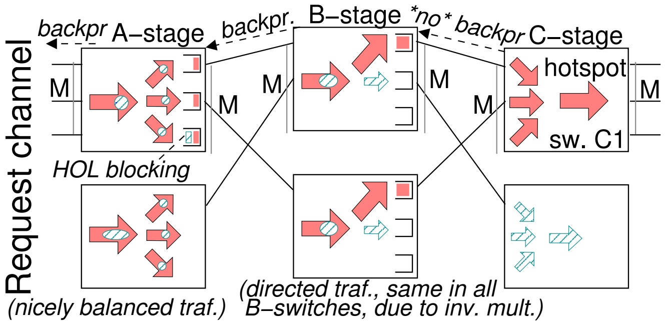

Figure 9: HOL blocking in $A{\rightarrow}B$ request queues.

Whereas request traffic is nicely balanced up to stage B, it can become more "directed" when it approaches stage C. Consider Fig. 9. If the instant request rate, from all inputs, towards a particular C-switch, say switch $C1$, exceeds M – i.e. the aggregate capacity of all $B{\rightarrow}C1$ request links, which

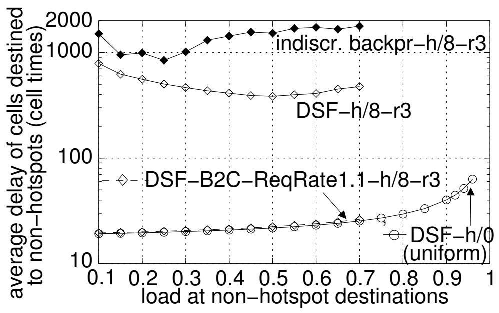

Figure 10: N=64, new_RTT=12; b=12,u=31,K=22; 48 cells reorder buffer per output.

is equal to the aggregate capacity of all outputs in $C1$–, all $B{\rightarrow}C1$ request queues will grow. Eventually, some of these queues may fill up with congested requests, because there are up to $M \cdot N \cdot u$ pending requests for switch $C1$ whereas all $B{\rightarrow}C1$ request queues together can store only $M \cdot M \cdot K$ of them. Once one queue fills up, it will indiscriminately "stop" cells in $A{\rightarrow}B$ request queues, which can induce HOL blocking and congestion expansion.

Normally, this condition cannot last for long. If there is at least one non-overloaded output in $C1$, the hierarchical request flow control will reduce the aggregate load of all $B{\rightarrow}C1$ request queues to a value below M; hence these queues will tend to drain, stopping HOL blocking. However, in the corner case, with *all* outputs in $C1$ overloaded, the request queues can remain full in the long term. This problem did not appear in the previous experiment (Fig. 8), because the aggregate request rate for any C switch was always safely below M: at least one output of each C switch was not overloaded. But this only happened by chance.

To avoid HOL blocking, we can *(a)* size $B{\rightarrow}C$ request queues so that they never fill up permanently. If each such queue can hold $\geq N \cdot u$ requests, then all pending requests for $C1$ will fit in the request queues in front of $B{\rightarrow}C1$ links, drastically diminishing in this way the HOL blocking effect. However, increasing by (at least) M times the request buffer storage in each B-switch, as required by this method, is not practical. Another solution is *(b)* to limit the number of requests that each ingress linecard may have pending towards each particular C-switch. If each linecard is allowed up to u pending requests per C-switch, we can achieve the same goal as method (a), but with the existing request buffer storage (assuming $K = u$). However an ingress linecard may consume its allowable requests for a particular C-switch, sending them to congested outputs, thus not being able afterwards to request other, possibly lightly loaded outputs in the same C-switch. A different approach is *(c)* to use separate request queues for the flows destined to different C-switches: the request queues in the A-stage must be organized per C-switch, and they must exert discriminative backpressure on ingress linecards. This method requires costly modifications.

The following solution is simpler. If the capacity of $B{\rightarrow}C1$ request links is *slightly higher* than the long-term demand for them, the request queues will almost always be empty: they may temporarily fill in the start of a congestion epoch, but hierarchical request control (request-grant rate equalization) ascertains that they will drain shortly after.

In the following experiment, the demand for each of outputs 1 to 8 (64-port fabric) is 3 cells per cell time –i.e. 300% loaded–, in order to overload all request links ending to $C1$. Figure 10 plots the delay of cells destined to non-hotspot outputs, under uniform traffic (h/0), and under the presence of the 8 hotspots described above. In the latter case, the maximum normalized load at each of the non-hotspot outputs is approximately (64-3·8)/56 ≈ 0.7. We have separate plots for the default DSF, and for DSF with $B{\rightarrow}C$ request link rate equal to 1.1 requests per cell time, i.e. a speedup of just 10% in request rate. All other request or grant links in the scheduling network operate at their default rate. As can be seen in the figure, well-behaved cells suffer in the default DSF under these stressing conditions. This happens due to HOL blocking that develops in the first stage of the request channel. However, with the 1.1 speedup on $B{\rightarrow}C$ request links, performance is excellent.

4. AVOIDING REQUEST-GRANT DELAY

In this section we eliminate the request-grant latency overhead for non-congested cells.

4.1 Limiting the number of rush cells

We have seen that forwarding cells eagerly, i.e. without first requesting and receiving grant, runs the danger of filling internal buffers. For this reason, we allow every ingress linecard to forward only a *few* such cells, which we denote as *rush cells*. When a linecard forwards one rush cell, it increments a *rush_out* counter. (This counter is shared by all VOQs, irrespective of the output that they target.) When a rush cell exits the fabric, an ACK message is issued upstream[10], which decrements the rush_out counter again.

Up to now, the scheduler that forwards VOQ cells from an ingress linecard considered as eligible only granted VOQs. For rushy transmission, we will also consider some VOQs that *have no grant* when the *rush_out count is below a threshold, U*; these non-granted VOQs must additionally *(a)* be non-empty, *(b)* have no request pending, and *(c)* their HOL cell must have been waiting in the ingress linecard for less than 3/2 new_RTTs (note[11]). On the other hand, only VOQs non-eligible for rushy transmission can issue new requests.

In this paper, we set U equal to the *ack_RTT*, measured in cell times, which spans from the time an ingress linecard injects a rush cell, till the time it receives the respective ACK (assuming no contention). Although more conservative values for U are also possible, in this way we provision for continuous flows of back-to-back rush cells. Indeed, our simulations have shown that non-conflicting input-output connections are served eagerly, thus with minimized delays. Under heavy load, ACKs do not return fast enough, and request-grant transmissions dominate[12].

Using a simple model, we can estimate the frequency of

[10]ACK messages are routed in place of grant messages.

[11]We choose to forward eagerly only very "recent" cells. Note that the 3/2 new_RTTs time is close to the minimum delay of request-grant cells.

[12]Note that rush cells use multipath routing (linecards use N, separate distribution pointers for rush cells), but don't have reorder buffer space reserved. In our simulations, we have not observed reorder buffer overflow, basically because under high load, and especially under bursty traffic, when cells tend to arrive out-of-order, most cells use the request-grant mode, which explicitly reserves reorder buffer space.

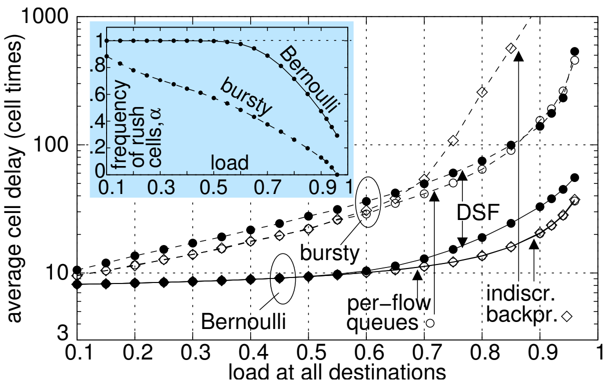

Figure 11: N=64, new_RTT=12; b=12,u=31,K=22; 68 cells reorder buffer per output.

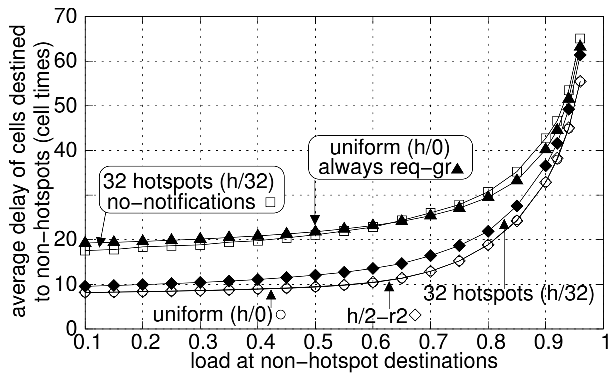

Figure 12: N=64, new_RTT=12;b=12,u=31,K=22; 68 cells reorder buffer per output; DSF.

rush cells, α. Denote by $\overline{T(\rho)}$ the average ACK RTT, when inputs inject cells at rate ρ (contention is present now). Frequency α can be expressed as $\min(1, \frac{U}{\rho \cdot \overline{T(\rho)}})$ –i.e. how many rush cells an input injects before receiving the "first" ACK, divided by the total number of cells it injects. If we break $\overline{T(\rho)}$ to its constant component, ack_RTT, and its variable component (queueing delays), $\overline{t(\rho)}$, α=min$(1, \frac{U}{\rho \cdot ack_RTT + \rho \cdot \overline{t(\rho)}})$. As expected, α decreases with increasing ack_RTT. But if we set U=ack_RTT, this relationship is reversed. Now $1/\alpha$ becomes $\max(1, \rho + \frac{\rho \cdot \overline{t(\rho)}}{ack\_RTT})$. Assuming that $\overline{t(\rho)}$ only loosely depends on ack_RTT (with $b \approx 1$ new_RTT), with increasing ack_RTT, $1/\alpha$ will decrease hence α will increase. These trends are verified in our simulations. The frequency of rush cells, α, *increases* with ack_RTT; it *decreases* with increasing load, ρ, and with increasing queueing delays, e.g. Bernoulli vs. bursty arrivals. Obviously, the $lim_{\rho \to 1}(\alpha) \approx 0$, since as input injection rate approaches unity, $\overline{t(\rho)}$ becomes immense –not infinite though because the fabric buffer space is finite.

4.2 Congestion notifications

A problem with the aforementioned method, alone, is that it cannot discriminate well between congested and non-congested flows. A congested flow, "unaware of its congestion", may forward some cells in rushy mode. The corresponding ACKs will delay, but will eventually arrive. If we are lucky, the renewed (through these ACKs) rushy "quota" might not be used for the same congested flow again, because this flow will very likely have requests pending. However, it is possible that a congested flow continues to grab rushy quota. Congested cells experience significant delays, hence the rushy mode is not essential for them. Additionally, congested rush cells tend to flood data buffers, thus delaying all cells, congested or not. To prevent congested flows from appropriating rushy "quota", congested fabric-outputs can notify the active ingress linecards. In response, linecards can avoid forwarding new rush cells to congested outputs.

At every fabric-output, we monitor the aggregate occupancy of the respective crosspoint buffers and request counters, using a single occupancy counter, OC. We also use one bitmask of width equal to N, in order to identify ingress linecards that have been notified of this output's (possible) congestion. When a rush cell is forwarded out of the fabric while the respective OC is greater than a high threshold, H, and that cell's source has not been notified, a congestion notification is piggybacked to the ACK message. On the other hand, when a request is served from a credit scheduler while the respective OC is below a low threshold, L, and the respective source has been notified that the output is congested, a ceased-congestion notification is piggybacked to the issued grant. In this way, ingress linecards can discriminate between congested and non-congested outputs[13].

Figure 11 depicts delay for uniform traffic, with Bernoulli and bursty cell arrivals (10 cells average burst size). The minimum cell delay of a rush cell is 8 cell times, and 19 that of a request-grant cell. The high and low OC thresholds, H and L, are equal to 2 new_RTTs and 1 new_RTT, respectively[14]. As can be seen, the delay of DSF at low load is virtually identical to that of per-flow queueing. Indeed, as the subfigure included in Fig. 11 shows, most cells are forwarded in rushy mode. As the load increases above 0.6, the frequency of rush cells for Bernoulli arrivals decreases. Under bursty arrivals, this decrease starts from lower loads.

In Fig. 12, we measure the delay of non-congested cells, in the presence of 100% loaded hotspots. Under low loads, non-hotspot cells are sent in rushy mode, and their delay is minimized, while hotspots are fully utilized by request-grant cells. The *no-notifications* plot shows the behavior when inputs do not conform to congestion notifications. In this case, we observed that many hotspots receive rush cells, depriving rushy mode opportunities from non-congested cells. For this reason, non-congested cells are frequently penalized with the request-grant latency (compare with *always req-gr* plot).

5. IMPLEMENTATION ISSUES

This section analyzes the components of the new_RTT, shows that we can operate the system with crosspoint buffer size significantly lower than 1 new_RTT, and estimates the bandwidth and area overheads of the control subsystem.

Assume that signals cover a 100 meters distance to go from linecard to fabric, or vice-versa, i.e. ~ 500 ns propaga-

[13]Observe that grants and ACKs may arrive out-of-order, thus notifications may also arrive out-of-order. If an ingress linecard receives a ceased-congestion notification from an output which is currently considered non-congested, it will mark this output as congested, and vice-versa.

[14]When the 2 new_RTTs threshold is crossed upwards, we are sure that the first new request-grant cells will find the output busy. When the 1 new_RTT threshold is crossed downwards, the output can be idle when new rush cells arrive.

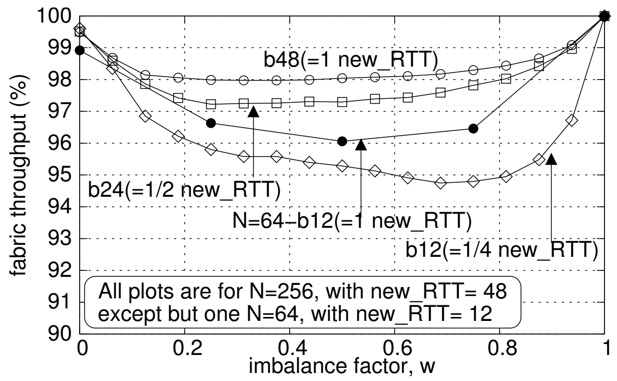

Figure 13: DSF; N=256, u=63, K= 31; 148 cells reorder buffer per output; Bernoulli arrivals.

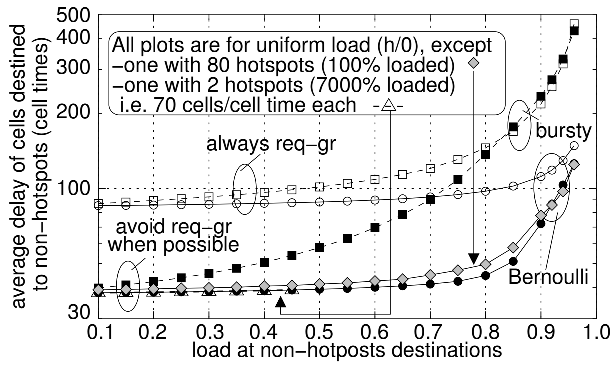

Figure 14: DSF; N=256, new_RTT=48; b=12,u= 63,K=31; 148 cells reorder buffer per output; 10 cells average burst size.

tion delay, for 0.2 meter/ns speed-of-light. At *10 Gb/s*, and for 64-byte cells, the 500 ns correspond to $\sim$ 10 cell times. As shown in Fig. 7, the new_RTT comprises two (2) such propagation delays. Additionally, it comprises four (4) signal propagation delays between adjacent fabric stages (A, B, C): assume that each one lasts two (2) cell times. Another significant contribution to new_RTT is due to the ($\sim$ 1 cell time) SERDES delay, which occurs every time a signal goes in or out of a chip. The new_RTT in our system contains 12 SERDERS delays, and, finally, 8 scheduling delays: credit scheduling, grant scheduling (C, B, A), VOQ scheduling, and crosspoint scheduling (A, B, C), one (1) cell time each.

Under these (rather pessimistic) assumptions, the new_RTT is 48 cell times, hence the crosspoint buffer size, b, must be $\geq$ 48, 64-byte cells. We significantly reduce this size below.

5.1 Sub-RTT crosspoint buffers

The following property of proper inverse multiplexing can reduce internal memory requirements. In the long term, and for any traffic pattern, the inverse multiplexing distributes evenly the load to the M buffers in front of any fabric-output. But an aggregate buffer space of 1 new_RTT per fabric-output suffices for full utilization, hence the crosspoint buffer size, b, can be as small as $1/M$ new_RTT.

Figure 13 depicts DSF throughput for N= 256, with b that corresponds to different fractions of the aforementioned 48 cell times new_RTT. (There is also a plot for new_RTT=12, b=12, and N= 64.) All inputs are 100% loaded. The normalized load from input i to output j is given by $w + \frac{1-w}{N}$, when $i = j$, and by $\frac{1-w}{N}$ otherwise: traffic is uniform when w= 0, and completely unbalanced –persistent $i \rightarrow i$ connections– when w= 1. As can be seen, for b= 48, 24, and 12 cells, DSF throughput is $\geq$ 94% for any w value, and very close to full when w=0 or 1. In this experiment, no rushy transmissions were allowed. We have also performed the same experiment with rushy mode activated, and results are practically the same as in Fig. 13. This happens because at high loads, almost all cells use the request-grant mode.

In Fig. 14, we plot the delay of DSF with b= 1/4 new_RTT= 12, for various traffic patterns. As can be seen in the figure, performance is unaffected by the presence of (up to 7000% loaded) hotspots, while delays at low loads are minimized. Observe that U, H, and L are all set as in Fig. 11. However, the ack_RTT (hence U) here is 47 cell times, i.e. > than the 11 cell times ack_RTT in Fig. 11. Effectively the frequency of rush cells, α, is higher here than in Fig. 11, e.g. $\sim$ 0.99 vs. 0.75 for Bernoulli arrivals at 0.8 load[15] –see Sec. 4.1.

Observe that the minimum allowable value for b is $1/M$ new_RTT, or 3 cells in the example here. Our results, not presented here, show that with b= 3, throughput is approximately 27.2% when w= 1 (Fig. 13). The reason is the intra-fabric (local) backpressure from stage B to stage A. When w= 1, the total traffic volume from each A-switch is directed to a particular C-switch, thus only one crosspoint buffer is active at each output of every B-switch. For full throughput, this crosspoint buffer must have size greater than the local data *rtt* between stages A and B, which is 11 cell times in this experiment[16]. Considering this *rtt* also, the buffer size $b \geq$ *max((1/M)$\cdot$ new_RTT, local A-B rtt)*.

Now assume that we wish to support *40 Gb/s* links. Would that increase the aforementioned buffer spaces by four? Consider that, practically, for large M ($\geq$ 32 for $N \geq$ 1024), the crosspoint buffer size will be dictated by the local, *A-B rtt*, which is independent of the large, linecard-fabric propagation delay. Also, many of the delays described above are overestimated. For example, crosspoint scheduling delay can be even smaller than 12.8 ns [11], which is the cell time at 40 Gb/s. If the local *rtt* between adjacent fabric stages A and B can be engineered below 200 ns, the required crosspoint space will be less than 16 (64-byte) cells, even for 40 Gb/s links.

5.2 Bandwidth & area overheads

Requests and grants need only identify a flow ID, and possibly the route that they will follow. The information required to identify a flow changes from hop to hop. The requests that go out from an ingress linecard indicate the targeted fabric-output port, and the path (B-switch) in the request channel; the grants sent to ingress linecards identify the fabric-output port that they come from. At a link between an A-switch and a B-switch, there are M inputs combined with N outputs to identify and separate from each other, and symmetrically at a link between a B-switch and a C-switch. Effectively, the size of a request or grant message is always smaller than $log_2M + log_2N$, or $3/2{\cdot}log_2N$ bits.

When one request (grant) is forwarded, there may also

[15]In Fig. 5, U= ack_RTT= 2, and $\alpha \approx 0.30$ at 0.8 load.

[16]Cell scheduling in A and B, plus (outstanding) credit scheduling in B, plus 2 A-B propagations, plus 4 SERDES.

be a request- (grant-) credit issued upstream. These credits need to identify a port ID in the present switch, hence they are $log_2 M$-bit wide, each. Summing up, the bandwidth overhead is (1 req+1 req-credit+1 gr+1 gr-credit) $4 \cdot log_2 N$ bits, per cell: for N= 4096, it corresponds to 9.4% for 64-byte, 4.6% for 128-byte, and 2.4% for 256-byte cells. For N= 16K, the respective numbers are 11%, 5.5%, and 2.8%.

Next we estimate the area overhead of the control subsystem. Every A- or B-switch has M request queues, and each such queue has a capacity of $M \cdot K$ requests, $3/2 \cdot log_2 N$-bit wide, each; hence the request storage is $3/2 \cdot N \cdot K \cdot log_2 N$ bits, per switch. Since we need an equal amount of grant storage, the area overhead per switch is $3 \cdot N \cdot K \cdot log_2 N$ bits, or $18 \cdot N \cdot K \cdot log_2 N$ transistors (xtors), assuming 6 xtors per memory bit. Thus, for K= 32 (note[17]), we need 27 M xtors if N= 4096, and 126 M xtors if N= 16 K.

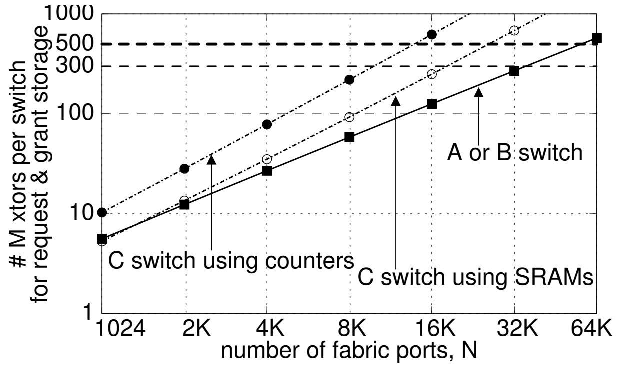

Figure 15: Millions transistors per switch.

Every C-switch contains $M \cdot N$ request counters. Assuming that each counter can store up to 127 (=u) requests, the counter width is 7 bits. The C-switch additionally comprises $M \cdot N$, $\log_2 M$-bit wide, distribution counters. For 20 xtors per counter bit, the total cost of counters per C-switch is $20 \cdot M \cdot N \cdot (7 + log_2 M)$ xtors. To reduce the transistor count, several counters can be implemented in an SRAM block, with external adders for increments and decrements. Considering that an SRAM bit consumes about 3 times less area than a counter bit, there is room for significant savings. Figure 15 depicts the number of transistors for request and grant storage, per switch, for various fabric sizes, N.

6. CONCLUSIONS

We proposed a distributed congestion management scheme for buffered, 3-stage Clos fabrics, comprising buffered crossbar switches. Performance simulations demonstrated high utilization, and very good flow isolation, comparable to that of pure per-flow queueing, even under hard traffic conditions, with many, highly congested outputs. For applications requiring very low latencies, we showed how linecards can avoid the request-grant latency overhead for lightly loaded flows. The scheduling unit exploits basic hardware primitives, i.e. distributed, single-resource (link) RR arbiters (loosely coordinated by plain backpressure signals) that operate in a pipeline, thus being scalable to thousands of ports.

Certainly, as with any new solution, some new practical problems come to shore, and many needs to be done before we can start thinking of actual implementation. We note two open issues here: routing of upstream messages –e.g. grants–, and control error recovery.

Extension to 5 or more stages forms a new challenge. An in-depth study of transient behavior is also interesting.

[17]K must be $\geq$ the local rtt, between adjacent stages.

7. ACKNOWLEDGMENTS

This work was supported by the European Commission in the context of the SARC (Scalable Computer Architecture) integrated project #27648 (FP6), and the HiPEAC network of excellence, and in part by IBM Phd Fellowship. The author would like to thank Manolis Katevenis for many stimulating discussions and ideas. Cyriel Minkenberg is thanked for helping estimate the RTT in Section 5.1. Finally, Wladek Olesinski and the anonymous reviewers are thanked for helping improve the quality of this paper.

8. REFERENCES

[1] C. Clos: "A Study of Non-Blocking Switching Networks", *Bell Systems Tech. Journal*, vol. 32, March 1953.

[2] L. Valiant, G. Brebner:"Universal Schemes for Parallel Communication", *Proc. ACM Symp. on Theory of Computing (STOC)*,Milwaukee, USA, May 1981.

[3] V. Benes: "Optimal Rearrangeable Multistage Connecting Networks", *Bell Systems Tech. Journal*, vol. 43, July 1964.

[4] T. Anderson, e.a.: "High-Speed Switch Scheduling for Local-Area Networks", *ACM Trans. on Computer Systems*, vol. 11, Nov. 1993.

[5] M. Katevenis: "Fast switching and Fair Control of Congested Flow in Broad-Band Networks", *IEEE J. Select. Areas in Commun.*, vol. 5, Oct. 1987.

[6] G. Sapountzis, M. Katevenis: "Benes Switching Fabrics with O(N)-Complexity Internal Backpressure", *IEEE Commun. Magazine*, vol. 43, Jan. 2005.

[7] J. Duato, e.a.: "A New Scalable and Cost-Effective Congestion Management Strategy for Lossless Multistage Interconnection Networks", *Proc. IEEE Symp. High-Perf. Computer Arch. (HPCA)*, San Francisco, USA, Feb. 2005.

[8] P. Pappu, e.a.: "Distributed Queueing in Scalable High Performance Routers", *Proc. IEEE INFOCOM*, San Francisco, USA, March 2003.

[9] N. Chrysos, M. Katevenis:"Scheduling in Non-Blocking Buffered 3-Stage Switching Fabrics", *Proc. IEEE INFOCOM*, Barcelona, Spain, April 2006.

[10] N. Chrysos: "Request-Grant Scheduling for Congestion Elimination in Multi-Stage Networks", *Ph.D dissertation*, Univ. of Crete, Dec. 2006.

[11] M. Katevenis, e.a.: "Variable Packet Size Buffered Crossbar (CICQ) Switches", *Proc. IEEE ICC*, Paris, France, June 2004.

[12] P. Pappu, J. Turner, K. Wong: "Work-Conserving Distributed Schedulers for Terabit Routers", *Proc. of ACM SIGCOMM*, Portland, USA, Sept. 2004.

[13] C. Minkenberg, e.a.: "Current Issues in Packet Switch Design", *Proc. HOTNETS*, Princeton, USA, Oct. 2002.