Discharging the Network From Its Flow Control Headaches: Packet Drops and HOL Blocking

Nikolaos Chrysos, Lydia Chen, Christoforos Kachris, and Manolis Katevenis

***Abstract*—Congestion control becomes indispensable in highly-utilized consolidated networks running demanding applications. In this paper, proactive congestion management schemes for Clos networks are described and evaluated. The key idea is to move the congestion avoidance burden from the data fabric to a scheduling network, which isolates flows using per-flow request counters. The scheduling network comprises per output arbiters that grant data packets after reserving space for them in the buffer memories in front of fabric outputs. Computer simulations show that this strategy eliminates head-of-line (HOL) blocking and its adversarial effects throughout the fabric, without having to drop packets. In particular, a simplified model describes this result as a synergy between proactive buffer reservations and fine-grained multi-path routing.**

Two alternative designs are presented. The first one places all arbiters in a central control unit, is simpler, and has superior performance. The second is more scalable by distributing the arbiters over the switching elements of the Clos network and by routing the control messages to and from endpoint adapters via multiple paths. Computer simulations of the complete system demonstrate high throughput and low latency under any number of congested outputs. Weighted max-min fair allocation of fabric-output link bandwidth is also demonstrated. Furthermore, delay breakdowns show that the time that packets wait in fabric and re-sequencing buffers is minimized as a result of the reduced (and equalized across all fabric paths) in-fabric contention. Finally, the high throughput capability of the system is corroborated by a Markov chain analysis of output buffer credits.

***Index Terms*—Computer networks, multi-stage switching fabrics, interconnection networks, flow control, scheduling.**

1 . Introduction

As datacenter applications become more sophisticated, there is a growing need for efficient and scalable switching fabrics to cope with the increasing volume and the divergent requirements of ("east-west") inter-server traffic that these applications generate. These new fabrics are expected to offer a high-end alternative to current installations that are based on standalone, off-the-shelf switches. In parallel, network consolidation results in more flows, protocols and applications being multiplexed on a higher utilized infrastructure. For this to succeed, the bottlenecks that any stream of traffic may have should not detriment the global network performance. The common practice today is to overprovision the network by adding switches and cables, sometimes in an ad-hoc manner. Although overprovisioning can be successful in certain roles, any size of pipe may occasionally fill up. The role of congestion management is to ensure that the network operates robustly under such events.

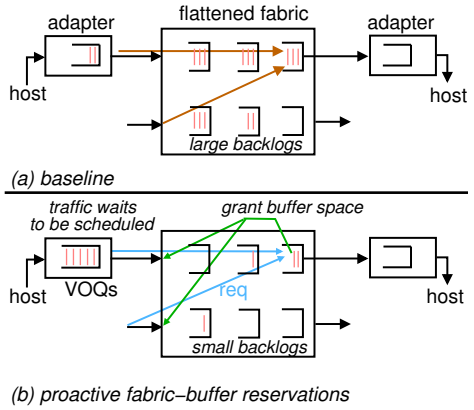


Fig. 1. Proactive buffer reservations vs. indiscriminate flow control in datacenter fabrics.

For single-stage switches, congestion has been successfully tackled by using virtual output queues (VOQs) to isolate flows for different outputs and to prevent head-of-line (HOL) blocking. Extensions of the same method for multi-stage fabrics do exist, but require order $O(p \cdot N)$ queues inside the switching elements, where N and p denote the number of fabric-ports and service classes respectively—see for instance [1] for a full-fledged solution and [2] for a hybrid approach. In practice, one can afford VOQs only at the inputs of the fabric, and therefore one has to deal with the interference between flows sharing fabric queues.

At the same time, scale-out, latency-sensitive applications are typically characterized by intensive inter-server communication, with high fan-in, and also require small in-fabric delays and delay variations [17], [18]. As a response, the industry is now shifting from the hierarchical datacenter networks of the past to *flattened fabrics* that provide full bisection bandwidth. Eventually, the same trend will invalidate the present-day congestion management that relies on lossy switches with large buffers (and queues) and on sluggish (TCP-like) software re-transmissions. What comes next is networks with small buffers that use flow control to prevent packet drops. This transition is seriously hampered by the fear of ensuing head-of-line (HOL) blocking and saturation trees, which may also have detrimental effects to packet delays—see Fig. 1(a).

In this paper, we propose and evaluate fabrics with small,

Manuscript submitted 30 November 2012.

Nikolaos Chrysos is with Foundation for Research and Technology (FORTH); a large portion of this work was performed while he was with IBM Research–Zurich; (email: nchrysos@ics.forth.gr).

Lydia Chen is with IBM Research–Zurich (email: yic@zurich.ibm.com)

Christoforos Kachris is with Athens Information Technology (email: kachris@ait.edu.gr)

Manolis Katevenis is with Foundation for Research and Technology (FORTH), and University of Crete (email: kateveni@ics.forth.gr)

cleverly-scheduled buffers inside the switching elements. As shown in Fig. 1(b), in our method, a scheduler selectively pulls packets from the input VOQs inside the network, ensuring that injections do not overshoot network buffers. Our results demonstrate that this solution frees the fabric from buffer overflows and from HOL blocking, thus minimizing the in-fabric packet delays under the most demanding traffic conditions.

1.1 Overview of proactive fabric-buffer reservations

We consider the Clos datacenter network (or fabric) shown in Fig. 2, where each link is associated with a small buffer in front of it. We define a flow as a fabric-input/fabric-output/priority triple. Arriving packets are stored in per-flow VOQs, located in network adapters in front of the fabric.

The proposed congestion management scheme is proactive, in the sense that it takes measures before the onset of congestion. In particular, the VOQs first issue a request and reserve space in *all* buffers across a fabric route, and then let a data packet go through it[1]. Buffer credits are granted by a scheduling sub-network that comprises pipelined credit arbiters, one for each link of the fabric. Buffer reservations are made in *reverse order*, moving from outputs to inputs, one stage at a time. In this way, internal buffers are reserved only for packets that are feasible for downstream links. Effectively, this scheme eliminates the need for hop-by-hop backpressure, which can lead to HOL blocking, and, at the same time, it prevents packet drops.

Our scheme lifts the congestion avoidance burden from the data network and places it into the scheduling network. This is a reasonable trade-off, as requests are significantly smaller in size than payload data, and can also be combined in per-flow counters. Thus, it is easier to handle a flood of requests in the scheduling network than a flood of data packets in the data fabric.

Extensive performance evaluations of this method, combined with packet-level multi-pathing, demonstrate that the resulting networks are immune to congestive episodes and that they successfully compete against single-stage architectures. The new systems however scale to tens of thousands of ports, about two orders of magnitude larger than their single-stage counterparts. Additionally, by preventing HOL blocking, the packet delays and the delay variations inside the fabric and in re-sequencing buffers become marginal.

Besides isolating non-congested flows, the proposed scheme also enables fair allocation of congested links, without requiring per-flow queues inside the network. Having eliminated the intra-fabric (*i.e.* hop-by-hop) backpressure, there is no problem if flows share the fabric queues, since the fabric now steadily forwards data at the rate that these are admitted into it by the schedulers.

The cost to be paid is additional *control* logic and buffers that can readily be distributed over multiple chips and be run by independent, asynchronous hardware units, plus some fixed bandwidth to accommodate for the control messages.

[1] A detailed description can be found in Section 3, and Fig. 3.

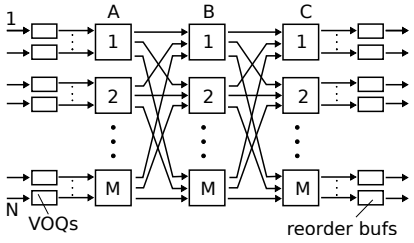


Fig. 2. A three-stage Clos fabric with N-ports, made of M×M switches.

1.2 Contents

We start in Section 2 with a review of related work. Section 3 describes a generic scheduler for buffered multi-stage fabrics that reserves fabric buffers ahead of packet injections to eliminate HOL blocking. In Section 3.2, we simplify the scheduler by exploiting the non-blocking properties of three-stage Clos networks that perform fine-grained load balancing. Then, using a fluid model, we show that the simplified scheduler prevents HOL blocking at any fabric stage much like the generic scheduler does. Our first computer simulation results in Section 3.3 highlight the benefits gained: the time that packets wait in the fabric and re-sequencing buffers drops by more than two orders of magnitude in the presence of hotspots.

Section 4 uses a Markov-chain analysis to study the throughput as a function of buffer size and switch size. With modest buffer sizes, in the range of 7 to 10 packets per switch output, nearly full throughput is obtained independent of switch size, N.

In Section 5, two alternative designs are presented. The first one employs a central scheduler, its operation is simpler, and has superior performance, *i.e.* smoother transient behavior at the onset of congestion. The second design improves scalability by distributing the control arbiters over the fabric elements, and by routing control messages over multiple paths. Computer simulations of detailed system models in Section 6 demonstrate high throughput and low delays, under adverse, uniform and non-uniform traffic, for both systems. Most notably, robust operation is sustained in extreme traffic conditions, where all inputs synchronously overload one output after the other, resembling incast congestion [28].

Section 7 draws hardware cost projections and addresses implementation issues. Finally, we conclude with discussions in Section 8 and a summary in Section 9.

2 . Related work

2.1 Proactive congestion control

Proactive congestion control is quite different and in certain aspects much more efficient than reactive congestion control. Reactive control allows congestion to develop and subsequently tries to break it. However, once you allow too many packets into the network (like too many cars into a city junction), but lack the capacity to store them all across the fabric, HOL blocking can develop, thus affecting delay and throughput performance. Reactive algorithms do not prevent the initial formation of buffer backlogs, and are frequently

over-conservative, slowing down flows more than necessary and/or delaying their recovery. In effect, they perform sub-optimally under the rapidly changing traffic conditions generated by bursty workloads.

From research, the benefits of proactive buffer reservations were first examined in earlier versions of this work [3], [4]. A similar proactive congestion control that regulates the long term rate at fabric output ports was even earlier presented in [9]. This system foresees a complex and lengthy communication and computation algorithm. To offset that cost, rate adjustments are made fairly infrequently (*e.g.* every 100 μs), thus increasing the delay of new packets arriving at empty VOQs, and not preventing HOL blocking in between adjustment times. More recently, proactive congestion control for on-chip networks was presented in [10], and multicast traffic support for a server-rack fabric using a request-grant/credit protocol was examined in [8].

Some recent switch products use a request-grant/credit protocol in their fabrics, in order to reserve buffers at end-point linecards [5] [6] [7]. These schemes regulate the per-output flow rates over a long time window, and do not offer guarantees on the filling level of internal buffers, which is crucial to avoid HOL blocking. For these schemes, there are no performance evaluation results or detailed system descriptions publicly available.

Another recently proposed scheme that proactively avoids saturation trees is *Speculative Reservation Protocol (SRP)* [16]. SRP does not reserve network buffers, but regulates flows by serializing the injections using a reservation schedule for each output endpoint. A similar method, named *implicit rate regulation*, was also examined in [15]. However, our results in Section 6.5 suggest that sources may miss their time schedule because of in-fabric contention. In this case, outputs may be overloaded, increasing the in-fabric contention and thus bringing even more out-of-schedule injections.

2.2 Routing induced congestion

There has been extensive work on optimal routing for fat-trees (Clos-like) datacenter or HPC fabrics. This branch of research tries to prevent internal hotspots by selecting a favorable routing function [19], [20] or by dynamically rerouting traffic away from congested minimal paths [21]. In the present work, we target output-induced congestion and use per-packet multi-path routing to avoid routing inefficiencies [22].

2.3 Is flow completion time a universal metric?

Many recent research papers focus on *flow completion time*, as a central measure of merit for datacenter networks. This point is particularly valid for production datacenters that use a MapReduce framework to support data-intensive scale-out applications, such as web search or data analytics. Typically, these applications run on lossy networks using TCP. Consequently, there have been attempts to modify the TCP transport in order to minimize the average flow completion time and to suppress the tail of its distribution [17]. Recently, lossless networks were shown to offer significant benefits along in this direction [18], [24]. The zFabric further extends this mechanism to include flow control inside the virtual switches of supervisors for virtualized networks [29].

Of particular interest are the cross-layer proposals in [25], [26]. These view the datacenter network as a complete, integral function aiming at end-to-end optimization. Similarly, in this paper we show that a switching fabric built with end-to-end performance in mind can perform considerably better than a (flattened) datacenter fabric made of standalone, off-the-shelf switches. Nevertheless, it is not yet the time to completely abandon traditional metrics of interest, such as bandwidth fairness, high utilization and flow isolation. These are still very important for many typical workloads, and, because of their endurance in time, they are likely to continue being important in the future. Furthermore, if a network fails in one or more of the above "traditional" metrics, it will be very difficult to deliver low flow completion times and run multiple services at the same time. The scheme that we propose in this paper reduces the in-fabric packet latency by more than one order of magnitude under extremely challenging traffic conditions. This directly translates into lower completion times for all flows.

3 . Proactive fabric-buffer reservations

We consider the three-stage Clos network shown in Fig. 2. For simplicity, we assume fixed-size packets[2]. All links operate at rate λ, such that the internal bisection bandwidth matches that of input or output ports. Packet time is defined as the time it takes to transmit a packet at rate λ.

By A, B, and C, we denote the 1st, 2nd, and 3rd stage, respectively. There are M (=$\sqrt{N}$), switches per stage, with M ports each. Every A and C switch connects to M ingress and M egress network adapters. Ingress adapters contain VOQs, which in this study are assumed to have infinite size. Egress adapters contain the buffers used for the re-sequencing, whose size can be bounded by the scheduling network. The $M \times M$ switching elements in this study comprise a buffer for b packets per (local) output, and can in practice be implemented by combined-input-output-queuing (CIOQ) or buffered crossbar chips [14]. In the baseline system, local, hop-by-hop backpressure is used to prevent buffer overflows.

3.1 First conceptual scheduler

Figure 3 shows a first generic scheduler that reserves fabric buffers thus removing the need for intra-fabric backpressure. Every packet in a VOQ issues a request and waits for the grant before it can proceed towards the fabric. Requests are processed by a pipeline of *credit arbiters*. There is one such arbiter for every output or internal link that maintains a private buffer-credit counter and private request buffers that hold outstanding requests. The credit arbiter serves a request only when its credit counter is non-zero. On service, it decrements that count (*i.e.* reserves buffer space) and forwards the (granted) request to the appropriate arbiter in the next pipeline stage. Eventually the request will reach the issuing VOQ, after buffer

[2]Variable-size packets have to be segmented at ingress and reassembled at egress.

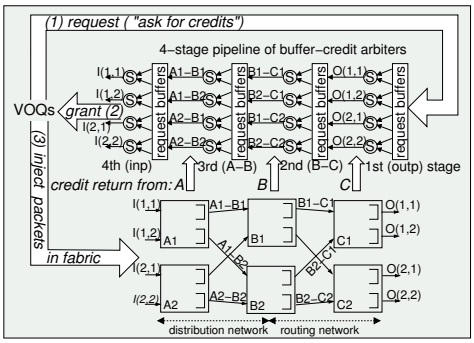


Fig. 3. The scheduling sub-unit (shown at the top) for a Clos fabric (bottom), using intermediate credit arbiters; N= 4, M= 2.

space has been reserved along a fabric path. Credits return to the credit arbiter, terminating a *credit cycle*, once the injected packet departs from the corresponding fabric buffer. Thus, the rate of data departures indirectly regulates the rate of grants.

First-stage arbiters allocate space for the buffers of stage C, enforcing feasible rates for fabric outputs; we call them *output arbiters*. Output arbiters must also choose a B switch (path) and direct requests to the appropriate 2nd-stage arbiter. The 2nd and 3rd stage arbiters reserve space for the buffers of stages B and A, respectively; we call those *intermediate arbiters*. Finally, 4th-stage arbiters simply forward grants to VOQs; we call them *input* arbiters.

Buffer reservation order: As shown in Fig. 3, we start buffer-space reservations from the last (output) fabric stage, moving left (to the inputs), one stage at a time. This is precisely opposite to how data progress and buffers get reserved under traditional backpressure protocols. The reservation direction chosen prevents packets that will later not be able to move on from needlessly occupying buffers: each reservation, when performed, is on behalf of a packet that has already reserved space in the next downstream buffer.

Of course, when reserving buffers in the upstream direction, as we do here, the danger is for many outputs to simultaneously reserve credits for packets to come from the same input or internal link. The contention at these upstream links may delay the credit recycling time, yielding buffer and link underutilization. Observe, however, that whereas output contention, which we tackle with this reservation order, can be severe, input contention has a short-term character because the (long-term) load at an input cannot exceed the capacity of a single link.

3.2 Simplifications owing to load balancing

For non-blocking performance, we use per-flow inverse multiplexing as in [1]. We use per-flow pointers to distribute the load of each flow equally among all B-switches, on a per-packet basis. This uniform (oblivious) load balancing equalizes the load on each path. The following theorem shows that, in a fluid model, if we combine this load balancing with the injection policy enforced by the output arbiters, then the admitted data will also fit in internal fabric buffers. Thus we can safely remove intermediate credit arbiters, and radically simplify the scheduling network.

Theorem **1.** *Assume a fluid model of a three-stage network where (i) there is a buffer in front of every switch output that can hold an amount of traffic equal to b, measured in arbitrary units; (ii) the traffic inside the fabric destined to a particular output never exceeds b, and is evenly distributed on per-flow basis over all possible paths; and (iii), there is no hop-by-hop backpressure between adjacent switch stages. Then the output buffers of switching elements will never overflow, i.e. the backlog in any of them will be $\leq b$.*

Proof: Consider a particular fabric output, j. The corresponding output arbiter makes sure that, at any time instance, there are at most b units of traffic inside the fabric towards output j. Due to perfect load balancing, the portion of this traffic that is assigned to a particular B-switch is at most $\frac{b}{M}$. Now, for all M outputs in the same C-stage switch, their collective traffic steered on a tagged $B{\rightarrow}C$ link is at most $\sum_{\nu=1}^{M} \frac{b}{M} = b$, hence the output buffers of stage B never overflow.

Because of the ideal traffic distribution in this fluid model, the maximum load on each $A{\rightarrow}B$ link (distribution network) is at most λ, where λ is also the capacity of the link. Therefore, since there is no hop-by-hop backpressure to prevent a busy buffer from draining, the backlog in front of $A{\rightarrow}B$ links will be less than b. ■

The following corollary becomes evident.

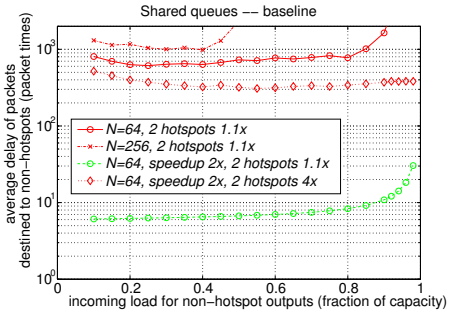
Corollary **1.** *In a fluid system as described in Theorem 1 but which implements hop-by-hop flow control to prevent buffer overflow, the flow control will never be activated.*

Note that for a real (non-fluid) system, we can guarantee that output adapters never exert backpressure on fabric-outputs, and that all injected packets fit into fabric-output buffers, hence fabric-output buffers also never exert backpressure. We cannot guarantee the same for the other buffers because load balancing on a per-packet basis may not be perfect in the short term (quantization imbalances). In order to prevent occasional buffer overflows, we use hop-by-hop backpressure to flow control A and B stage buffers—C stage buffers are flow controlled by output arbiters and do not need hop-by-hop backpressure. Our simulation results indicate that the hop-by-hop flow control is only rarely activated, due to random short-term conflicts.

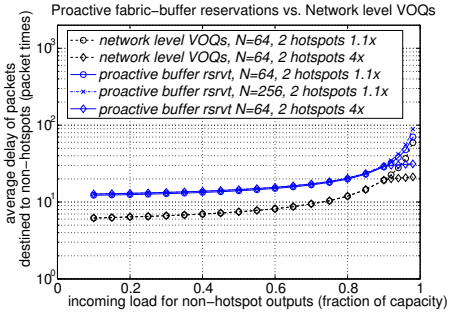
3.3 Tolerance to hotspots in computer simulations

Figure 4 presents the quality-of-service (QoS) levels obtained by the simplified scheduler. Subfigures (a) and (b) depict the average delay that packets experience from the time they enter the VOQ at the ingress adapter till the time they depart on the targeted destination[3]. We plot the delay of non-congested packets only, as a function of the aggregate load at their destinations. In Fig. 4(a), the progress of these packets is clearly inhibited by (congested) packets heading to (hotspot) destinations, as manifested by their large delays (observe the logscale of axis y). Note that the unwanted interference disappears when we speedup the fabric by a factor

[3]The simulation model and the traffic patterns are presented in Section 6.



(a) Baseline system



(b) Proactive buffer reservations vs. network-level VOQs. Centralized scheduler.

Fig. 4. Delay of packets going to non-congested destinations in the presence of hotspots; hotspots are overloaded by a factor of 1.1×. The proactive buffer reservations yield slightly higher delays at low loads because of the request-grant latency overhead.

of 2×, but shows up again when the oversubscription factor ramps up to 4.

Congested packets are also present in the experiments of Fig. 4(b), just as in (a), but, now, their presence does not impede the progress of non-congested ones. This figure depicts the performance of two systems. The first one uses network-level VOQs, *i.e.* O($N \cdot M$) number of queues per switch, thus preventing the interference by congested flows. The second one uses shared queues, as the baseline does, but proactively reserves fabric buffers using the simplified scheduler. The proposed solution isolates flows as well as network-level VOQs[4].

3.4 Intra-fabric delays in computer simulations

Next, we measure the delays that packets undergo in fabric queues and in re-sequencing, as a function of input load. Figure 5 presents the *average* and *maximum* queuing delays, under uniform and hotspot traffic. Comparing subfigures (a) and (b), one may see that, for the baseline scheme, the in-fabric delays increase dramatically in the presence of hotspots, starting from low loads. This is due to saturation trees. What is surprising is how similar are the corresponding delays with proactive buffer reservations: even with highly congestive traffic, non-congested packets flow seamlessly through the fabric, as if nothing happens. This manifests that internal fabric queues do not build up any significant backlog that could cause HOL blocking.

Comparing Fig. 5(a) with (b), *i.e.* the average with the maximum in-fabric delay, also manifests a large spread (jitter) of delays inside the fabric for the baseline system. This gets reflected into higher waiting times at the re-order buffers, as shown in subfigures (c) and (d). One may see that, in baseline, the waiting time in re-sequencing increases by up to two orders of magnitude when hotspots are present: hotspots can induce saturation trees inside the fabric and delay the "next-expected-packet" for arbitrarily long. (This result narrows the scope of the findings in [23], which states that packet spraying (multi-pathing) in fat-tree networks does not reorder packets significantly: this statement is false for networks with flow control.) On the other hand, with proactive buffer reservations, the re-sequencing delays are negligible and unaffected by hotspots; they are similar or better than in baseline even with uniform traffic.

4. Analysis of System Throughput

Having shown through analysis and simulations that the proposed mechanism protects innocent flows from congested ones, next we analytically quantify its throughput performance. For uniformly-destined traffic, such that all input VOQs are constantly full, we find that the throughput is nearly full even for small output buffers, in the range of 7-10 packets. Due to space limitations, below we outline the model and the main results [13].

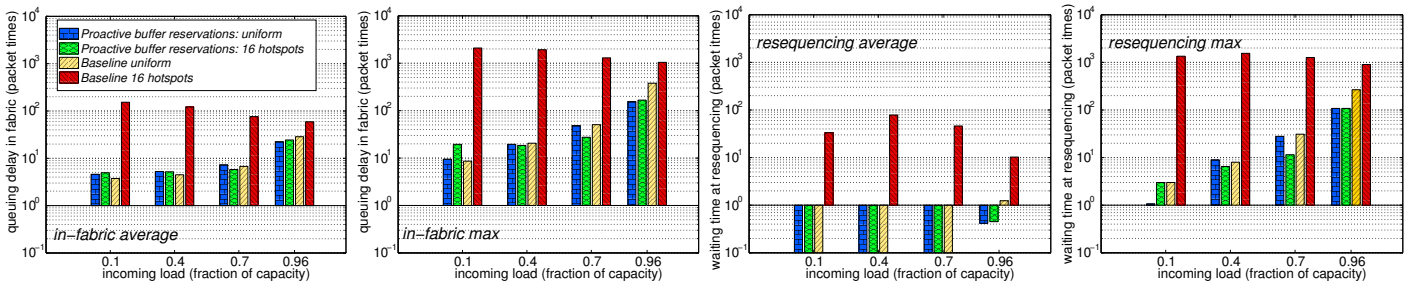
4.1 Model used in analysis

To ease the analysis, we will draw simple models for the network and the scheduling unit. In particular we consider an $N \times N$ single-stage switch, with VOQs at the inputs and small buffers in front of the outputs. This model ignores the first and second switch stages of the Clos network. This simplification is justified by the analysis of the previous section: after reserving space in output buffers and load balancing the traffic on a per-packet basis, the first and second stages merely steer traffic to fabric outputs, virtually without any contention or blocking.

The model that we study in this section performs the following operations in every packet slot:

1) Each unmatched input sends a request to every output i for which it has a packet in its VOQs.
2) If output i receives a request from m inputs, it chooses $k = \min(cr[i], m)$ of them to *grant randomly*, where cr[i] is the current credit at output i, and notifies each input whether its request was granted.
3) If an input receives one or more grants, it chooses *one* to *accept randomly*, notifies the corresponding output, and inserts the HOL packet from the corresponding VOQ in the targeted output buffer.
4) Each output decrements its credit counter by one for every accept signal it receives, and increments it by one for each packet that departs from the corresponding output buffer.

[4]Observe that for $N = 64$, and 2 hotspots overloaded by a factor of 4, the delay curves flatten out at loads ≥ 0.90. This happens because the load of non-congested destinations cannot increase beyond $\frac{64-2\cdot4}{64} = 0.875$.



(a) Aver. queuing delay in fabric (b) Max. queuing delay in fabric (c) Aver. waiting time at resequencing (d) Max. waiting time at resequencing

Fig. 5. Queuing delay of packets inside fabric-buffers and waiting time at re-sequencing: only packets to non-hotspots are considered.

5) Each non-empty output buffer forwards its HOL packet on the output link.

The affinity of this scheduling mechanism to PIM [11] is well pronounced. The main difference is that in PIM each output issues at most one grant per packet time, whereas here an output can have as many grants pending as the available output buffer slots.

We denote by U the steady state (input and output) utilization. Following the derivation in [11], the input utilization of PIM is equal to the probability that an input port receives at least one grant, $Pr(y \geq 1)$. This probability is equal to one (1) minus the probability that all N outputs grant one of the remaining $N-1$ inputs, *i.e.* $1-(\frac{N-1}{N})^N$. As $N \to \infty$

$$U = Pr(y \geq 1) = 1 - \left(\frac{N-1}{N}\right)^N \cong 1 - e^{-1} \qquad (1)$$

When $b > 1$, it's not as straightforward to obtain $Pr(y \geq 1)$, as the number of credits, x, available at every output may vary between 1 and b. In steady state, each output holds an average number of credits, $\beta = E[x]$. As a result, we obtain:

$$U = Pr(y \geq 1) = 1 - \left(\frac{\binom{N-1}{\beta}}{\binom{N}{\beta}}\right)^N = 1 - \left(\frac{N-\beta}{N}\right)^N \qquad (2)$$

One can obtain the upper bound of U as:

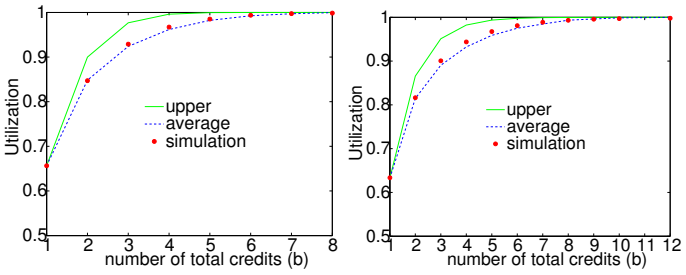
Lemma 4.1. *When* $b > 1$,

$$U \leq 1 - \left(\frac{N-b}{N}\right)^N$$

Proof: From Eq. 2, $Pr(y \geq 1)$ is monotonically increasing with β, and is maximized when $\beta = b$. As a result, $1-(\frac{N-b}{N})^N$ is the upper bound of $P(y \geq 1)$ and of U. ■

We can find β using an iterative procedure [13]. Next, we compare the average throughput computed using Eq. 2 with the upper bound of throughput from Lemma 4.1 and with the average throughput obtained from three independent computer simulations runs, using uniformly-destined traffic at 100% load.

In Fig. 6, we depict the throughput for buffer sizes b in 1-12 and switch sizes $N = 8$ and 256. The results from the analysis match very closely with those from the simulation, especially for intermediate b values, between 2 and 8 packets. Moreover, the difference between the upper bound and the



(a) N = 8 (b) N = 256

Fig. 6. Throughput vs. buffer size (b) from analysis and simulation for different switch sizes N.

average throughput diminishes with increasing b. As can be seen, when b is small, there is some dependency between buffer size and switch size, but when b is greater than 8, all switch sizes achieve virtually full utilization. Interestingly, the full throughput capability does not depend on desynchronized round-robin pointers as in *i*SLIP, but assumes random arbitration.

5 . Alternative Scheduler Organizations

In this section, we present two alternative organizations for multi-stage fabrics based on the simplified scheduler of Sec 3.2. The first uses a centralized scheduler, and the second a distributed one[5]. Note that in order to limit the write throughput of output buffers, we assume buffered crossbar switching elements; however, CIOQ switches can be used as well—see Section 6.5.

5.1 Central scheduler

In the first design, all output arbiters are placed in a central scheduling chip as shown in Fig. 7, offering a clean separation between control and datapath logic. The central scheduler comprises N output credit arbiters, responsible for reserving buffers (credits), as discussed in Section 3.2. When more than one output arbiter grants the same input at the same time, the corresponding *input arbiter* serializes these grants, sending them to the waiting VOQ. We consider that every output or input arbiter serves one request or grant, respectively, per packet time. To reduce pin count, endpoint adapters send and

[5]The design descriptions in this section match carefully the operation of our computer simulation models.

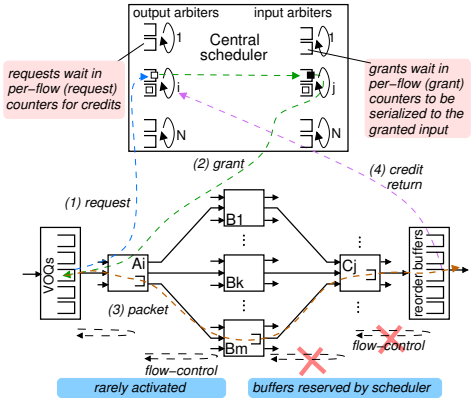


Fig. 7. A three-stage fabric with centralized control. The grants from (requests to) the scheduler are aggregated through the corresponding C (A) switch so as to reduce the pin count of the central scheduler; for simplicity the grants in the figure are shown as routed through their corresponding A switch.

receive control messages to and from the scheduler through their corresponding first- and third-stage switch elements, respectively.

As mentioned in Section 1, the proposed congestion control lifts the congestion avoidance burden from the data fabric and moves it to the scheduling network. A key point is that the per-flow requests and grants can be safely isolated inside per-flow counters. The central scheduler uses N^2 of them to store outstanding requests, and N^2 to store outstanding grants. The large number of counters limits the scalability of this design.

In order to reduce the counters' size, we throttle the per-flow requests. Specifically, each adapter limits the number of requests that a flow may have outstanding to an upper bound u (note[6]). As a favorable side effect, this request throttling equalizes the request rate of each flow with its grant rate.

Every adapter maintains a *pending-requests (pr)* and a *received-grants (rg)* variable for every corresponding flow, initialized to zero. Variable pr is incremented on every (per-flow) request sent, and decremented on every received grant when rg is incremented; counter rg is decremented when a new (per-flow) packet is forwarded into the fabric. A request arbiter visits the eligible VOQs and issues at most one request message per packet time. Each request message specifies the issuing flow and the number of requested credits.

Upon reaching the central scheduler, a request from flow $i \to j$ increments request count $i \to j$, in front of output arbiter j. Since we assume buffered crossbar switches, the output arbiter must choose a particular B switch (route), and reserve space in the corresponding crosspoint buffer. For this purpose, every output arbiter has N distribution counters, one for each input, selecting the B switch through which to route the next packet from each flow, and M credit counters, one for each corresponding crosspoint buffer. A flow is eligible for service when its request count is non-zero, and the credit count selected by its distribution counter is also non-zero. Once flow $i \to j$ gets served, its request count and the selected credit count get decremented by one. Then a grant is routed to input arbiter i, where it increments grant counter $i \to j$ by one. Non-zero grant counters are always eligible for service. When grant counter $i \to j$ is served, it is decremented by one, and a grant is sent to adapter i.

Observe that packet injections are subject to backpressure from stage A, thus it may not be possible to inject the HOL packet of VOQ $i \to j$ immediately after a grant arrives. Instead, an input link arbiter forwards packets from VOQs with $rg > 0$, subject to the availability of A-stage credits.

The route (*i.e.* a B-switch identifier) for each packet can be indicated by the grant message. To reduce the message size, the route can also be indicated by a per-flow distribution pointer inside adapter i, which is synchronized with the distribution pointer used by output arbiter j. A sequence number is included in the header of injected packets, which is used to re-sequence the out-of-order packets at the egress.

The injected packets travel to the targeted B switch subject to hop-by-hop backpressure, based on local, next-hop credits. Each input adapter or switch output maintains M credit counters, one for each crosspoint buffer along the corresponding input line of the downstream crossbar. The switch (or ingress adapter) can forward a packet only if credits for the targeted downstream crosspoint buffer are available. After serving a packet, the switch generates a credit to notify the upstream switch (or ingress adapter) about the now released buffer space. Local, hop-by-hop credits are sent upstream at a peak rate of one credit per upstream, per packet time. There is no backpressure from stage B to C. When the packet becomes in-order in the egress adapter, an (end-to-end) credit is returned to the central scheduler[7]. A link arbiter forwards the in-order packets from the adapter on the egress link. We do not send the end-to-end credit when the packet departs from the C-stage buffer, but when it becomes in-order in the egress adapter, to upper bound the buffers needed for reordering purposes.

Statistical pointer desynchronization: The output arbiters serve the per-flow request counters in a round-robin fashion. For improved desynchronization, each output arbiter uses a *different, random (but fixed) order* at which it visits inputs. In this way, we avoid the *persisting pointer synchronization* that we have observed in simulations with common-order output round-robin pointers [12]: for uniform traffic, multiple outputs may grant inputs *in phase* over extended periods of time, thus rendering unbalanced credit allocations and low throughput (80% or less).

5.2 Distributed scheduler

In this section, we overcome the scalability limitations of the central scheduler by distributing the scheduling functions over the switches of the Clos network[8]. Although this distributed design may seem more complex than the centralized one, our computer simulation results show that it yields nearly the same performance levels.

The network of the distributed scheduler is depicted in Fig. 8. There are M output arbiters in every C switch, one for each

[6]For continued transmissions, u must be dimensioned according to the request-grant rtt, which spans from the time an adapter issues a request till the time the adapter receives the corresponding grant.

[7]At most one credit is sent per egress, per packet time.

[8]Separate control chips, interconnected in Clos structure, can also be used.

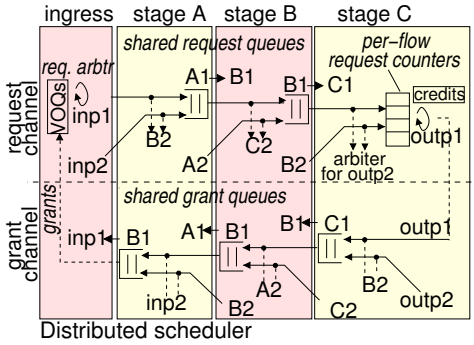


Fig. 8. Distributed scheduler for a 4×4 fabric.

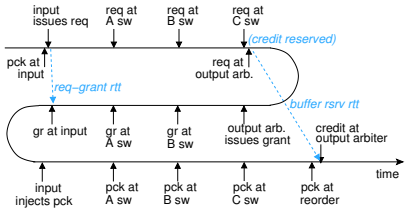


Fig. 9. Scheduling a packet using the distributed scheduler. Shown are the request-grant rtt, and the (credit) buffer reservations rtt.

fabric output. The ingress adapters communicate with them through a three-stage scheduling network. Each link in this network transfers one request or grant per packet time. We route requests and grants using inverse multiplexing (multi-path routing) much as we do for packets in the data network. The figure shows a single path from the request channel and a single path from the grant channel, connecting an ingress adapter with an output arbiter. In reality, there are M such paths for every input/output pair, one for each B switch. The per-flow requests (or grants) may now arrive out-of-order, and get merged in the receiving outstanding request or grant counter. There is no problem with that, since the per-flow request (grants) are interchangeable with each other.

Continuing with the request network in Fig. 8, there is a shared request queue in front of every $A \rightarrow B$ or $B \rightarrow C$ link, with space for $M \cdot K$ requests, each, where K is set equal to one local rtt worth of requests. Requests are issued from adapters as in the centralized design, but now every next (per-flow) request is routed through the next B switch, selected by a per-flow request distribution counter. Request queues are flow controlled using M *request-credit* counters per ($inp \rightarrow A$ and $A \rightarrow B$) link, each one initialized at K. Ingress adapters (A switches) forward a new request only if the corresponding request-credit count is non-zero, and decrement this count afterwards. For every request sent from stage A (or B), a request-credit is sent upstream. Although there is hop-by-hop, backpressure in the request channel, requests can freely depart from stage B, since space for them in per-flow request counters at stage C has already been reserved via request throttling.

Output arbiters operate as in the central scheduler, but now they must route every next (per-flow) grant through the next B switch. For economy, the grant path is selected by the corresponding packet distribution counter of the output arbiter. The grant channel (reverse path) includes a shared grant queue in front of each $C \rightarrow B$, $B \rightarrow A$, and $A \rightarrow inp$ link. Grant queues are managed using a credit-based type of flow control, similar to request queues[9]. The timing of operations in the scheduling network is depicted in Fig. 9.

5.2.1 Managing contention inside the scheduling network: The use of shared request queues[10], alone, has the same drawbacks with indiscriminate backpressure in the data network: requests of a congested flow can populate the shared request queues and reduce the scheduling (hence also the data) throughput. However, request throttling acts as *hierarchical flow control*, alleviating this problem to a large extent. Because there is no backpressure on the B switches of the request channel, there is no HOL blocking in $B \rightarrow C$ request queues.

One implicit assumption made so far is that the request counters inside the C switches are able to absorb the requests coming from any number of B switches. Assume that the packet size is P Bytes and that network links run at C Gb/s. C and P together determine the packet time on a link as $PT = 8 \cdot \frac{P}{C}$ ns. For P= 512B, this yields PT= 408, 102 and 40 ns, for C= 10, 40 and 100 Gb/s, respectively. Assume a clock speed of 500 MHz, *i.e.* a clock cycle, CC, of 2 ns, and dual-ported SRAMs that keep the requests counts, allowing one read/modify/write memory operation per clock cycle. We can allocate half of the SRAM bandwidth to write new requests and half to read requests (for output arbitration). Then, at each output, we can write $\frac{PT}{(2 \cdot CC)}$ new requests per packet time. This gives us an upper bound on how large the size of the C stage switches may be: 100, 25 and 10 ports for 10, 40 and 100 Gb/s links, respectively.

With P= 512B, bandwidth may be wasted for small packets. A possible solution is to use variable-size internal packets that hold fragments or the entire payload from more than one external packets [15]. Effectively, an intensively loaded VOQ or one that carries large IP packets will mostly produce full 512B internal packets—link overload situations typically involve large (MTU) packets [17], [28]. Nevertheless, the demand for read/write operations on the flow counters may increase when many small packets ($<$ 512B), coming from a large number of inputs, overload an output. In such an event, we may expect some short-term blocking inside the scheduling network, but this blocking will stop growing once the backlogs build up enough such that the internal packets become full; then, we can clear the backlogs using some internal speedup in the data network, or in the scheduler as in Sec. 6.4.

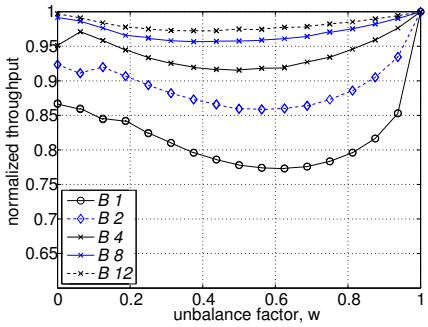
6 . Performance Evaluations

In this section, we use compute simulations to evaluate the performance of the proposed system. The computer models are built on top of an event-driven simulator, written in C++, and follow closely the system descriptions of Section 5. Targeting a thorough evaluation, we use the following traffic patterns that cover a wide range of workloads.

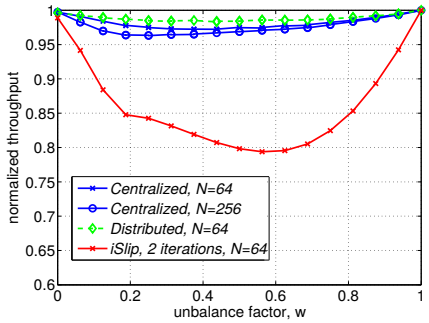
1) Uniformly destined traffic/ unbalanced traffic

[9]Output arbiters issue grants subject to grant queue flow control.

[10]Because of multi-path request routing, every B switch conveys the requests from as many as N^2 flows, therefore it is not scalable to use per-flow request counters to isolate the flows.



(a) Throughput of centralized scheduler vs. buffer size, B



(b) Throughput of buffer reservations vs. iSLIP.

Fig. 10. Throughput comparisons under unbalanced saturated traffic.

2) Bernoulli / bursty packet arrivals
3) Multiple concurrent hotspots
4) Fan-in/incast

Each performance point that we present, be it a number for latency or throughput, is the average of at least (3) independent simulations, run using different random seeds.

Parameter settings: For fabric-buffer reservations, all arbiters in the scheduling unit, switching elements and adapters implement the round-robin discipline. Packet cut-through operation is assumed for switching elements. The credit round-trip time of buffer reservations (credit-reserve till credit-return in Fig. 9) is 12 packet times, and all crosspoint queues can store 12 packets. The occupancy level of the re-sequencing buffers is explicitly upper bounded by the output arbiters. We set their maximum size to 300 packets. Finally, we do not use the intermediate credit arbiters shown in Fig. 3.

The minimum packet delay for both the centralized and the distributed system is 12 packet times. For baseline and network-level VOQs, which do not incur the additional request-grant latency overhead, the minimum packet delay is 6 packet times. Finally, for *i*SLIP, the minimum packet delay is zero (0).

The schemes that we compare are summarized below.

- *Shared queues with indiscriminate hop-by-hop backpressure (baseline)*
- *Per-flow packet buffers throughout the fabric (network-level VOQs)*
- *Baseline with endpoint acknowledgments that regulate VOQ injections (end2end acks)*
- *Single-stage crossbar running the iSLIP algorithm [12]*

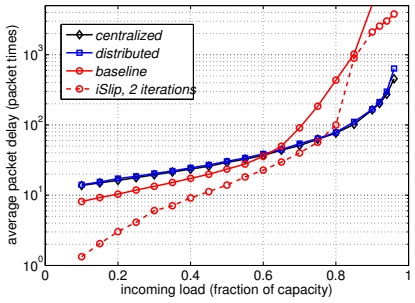


Fig. 11. Packet delay under bursty traffic; aver. burst size = 12 packets.

- *Speculative reservations (SRP) [16]*
- *Buffer reservations using the centralized scheduler*
- *Buffer reservations using the distributed scheduler*

6.1 Throughput under unbalanced traffic

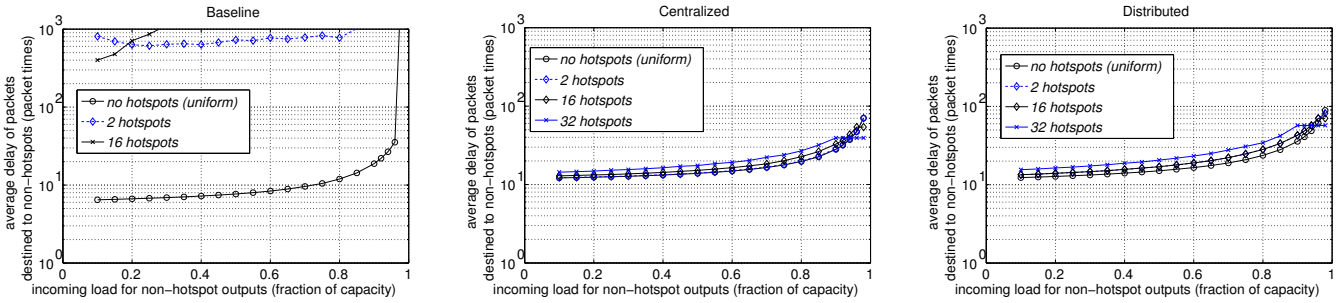
In this experiment, the unbalance degree of the traffic is controlled by parameter $w \in [0, 1]$, as in [14]: the normalized load $\rho_{i,j}$, from input i to output j is given by $w + \frac{1-w}{N}$, when $i = j$, and by $\frac{1-w}{N}$ otherwise. Traffic is uniformly-destined when w= 0, and directed (N non-conflicting, input $i \rightarrow$ output i, connections) when w= 1. For intermediate values of w, the traffic is a weighted mix of uniform and directed traffic (*i.e.* unbalanced traffic).

In Fig. 10(a), we plot the throughput of the system using the centralized scheduler for various buffer sizes at the crosspoints, B. The throughput increases with buffer size, approaching the maximum possible across the entire spectrum of w for B= 12 packets As can be seen in Fig. 10(b), the throughput of the centralized system is only marginally affected if we increase the number of ports from 64 to 256. Also, the distributed scheduler is shown to achieve equally high throughput. To improve the legibility of the figure, we omit throughput of the distributed scheduler for N= 64: this matches closely that of the centralized scheduler. As can be seen, both designs provide considerably higher throughput than the *i*SLIP, single-stage switch.

6.2 Bursty traffic

Next, we consider uniformly-destined bursty traffic, with exponentially distributed burst sizes. Under bursty traffic, any output may get overloaded during a transient period of time, and impede the progress of packets heading elsewhere; thus we expect this effect to show up in the baseline scheme.

In Fig. 11, we plot the performance of baseline, of *i*SLIP with 2 iterations, and of the centralized and distributed schedulers. At low loads, the delay of *i*SLIP is close to zero, that of the baseline around eight (8), and of the centralized and distributed schedulers around 12 packet times. As the load increases, the comparisons go the other way around: *i*SLIP has the higher delay, next comes the baseline, and the proposed systems incur by far lower delays.



(a) No buffer reservations (baseline) (b) Centralized scheduler (c) Distributed scheduler

Fig. 12. Delay of non-congested packets in the presence of a varying number of hotspots.

6.3 Multiple concurrent hotspots

Our next experiment tests the tolerance of the system to hotspots. Under hotspot traffic, each destination belonging to a designated set of k "hotspots" is oversubscribed by a factor of h, uniformly from all sources; other destinations receive a smaller load. To determine the destination of a packet, we pull a uniform random variable, $z \in (0,1)$. If $z < h \cdot \frac{K}{N}$, the destination of the packet is uniformly selected among the hotspots; otherwise, it is uniformly selected among the non-hotspots. To generate a load ρ for the non-hotspot outputs, the effective load at which inputs generate traffic is $\rho \cdot \frac{(N-K)}{N} + h \cdot \frac{K}{N}$.

The aim of these experiments is to see how the delay of packets going to non-hotspots is affected in the presence of congested packets. Some first results for the centralized scheduler were presented in Section 3.3. Here we include the performance of the distributed scheduler as well, and examine k= 0-32 hotspots, each oversubscribed by a factor h= 1.1×. Note that k= 0 corresponds to uniformly-destined traffic.

In Fig. 12(a), we plot the throughput of the baseline system. Baseline performs well under uniform traffic, but its delay ramps up by at least two orders of magnitude when hotspots are present. Figures 12(b,c), present the corresponding results for the centralized and distributed scheduler respectively. It can be seen that, thanks to proactive buffer reservations, non-congested packets are *virtually unaffected by the presence of any number of hotspots*. The size of switch-element buffers required to achieve this result is independent of N. The scheduling network needs request buffers with size that grows with N, but these are implemented using cheap SRAM-based counters.

6.4 Transient behavior under fan-in (incast) traffic patterns

In this experiment, we examine an adversarial fan-in traffic pattern, similar to what has been observed in datacenters running MapReduce, as well as in distributed storage clusters [28]. In particular, we warm up the system by having each input send uniform traffic to 58 non-hotspot destinations, loading them at 50%. This will be our background (non-hotspot) traffic, which lasts until the end of the simulation. In the following, time is measured in packet times. At time 5000, all inputs generates additional packets, which are all destined to hotspot1. The aggregate demand for hotspot1 is 30× higher than its capacity. At time 10000 the traffic to hotspot1 seizes, and inputs generate traffic for hotspot2. Hotspot 2 is loaded as much as hotspot 1, and for the same duration. In the sequel, hotspot3, hotspot4, hotspot5, and hotspot6, are similarly overloaded one after the other, starting from time 15000, 20000, 25000 and 30000, respectively.

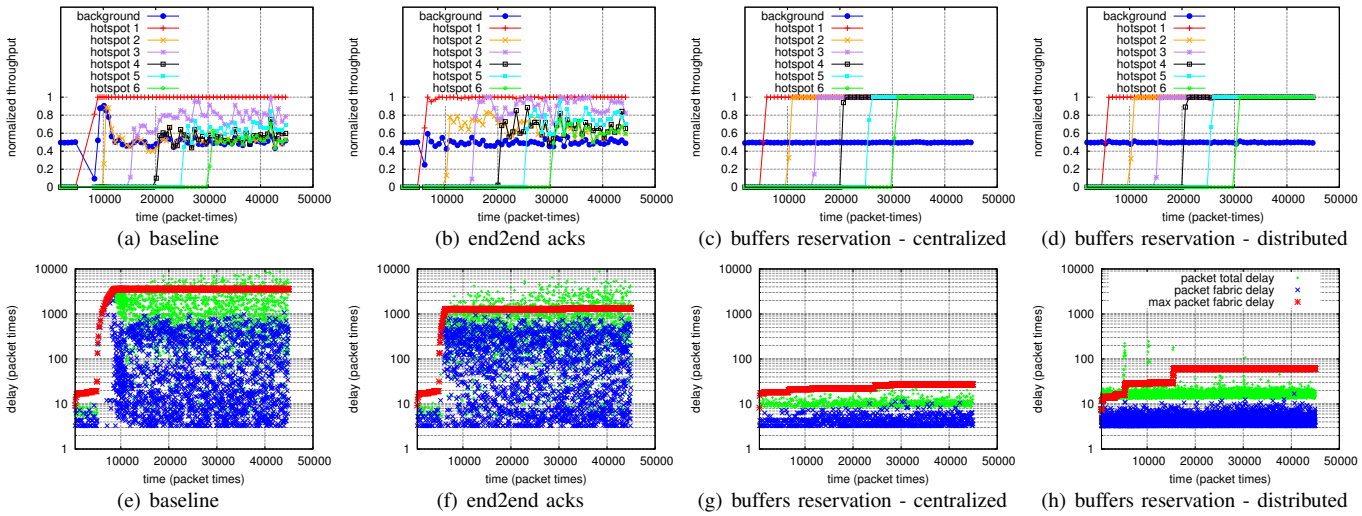
We performed this experiment for the baseline, the centralized scheduler, the distributed scheduler, as well as for an *end2end acknowledgments* scheme. The latter scheme is based on baseline, but allows W unacknowledged packets per-flow, which can fit in buffers reserved at output adapters. Whenever a fabric output forwards an in-order packet, it issues an ACK message to the sourcing input adapter. In our experiments, we set W= 16 packets, which slightly exceeds the end-to-end acknowledgment rtt (12 packet times). For simplicity, ACKs are sent-out-of-band, incur zero delay, and do not interfere with data traffic. Apart from that, all other parameters for end2end acks (*e.g.* switching element type, buffer sizes and link speeds) are the same as in the other schemes that we examine.

The following results show that although the end2end acks scheme can regulate the traffic to each destination, it's not as robust as fabric-buffer reservations. The reason is that all inputs may collectively send $N \cdot W$ packets towards a congested destination before being throttled. These packets can fill up the network buffers and induce saturation trees[11].

In Figs. 13(a,b,c,d), we plot the time series of the per-hotspot throughputs and of the average throughput at the remaining non-hotspot outputs (background traffic). Also, in Figs. 13(e,f,g,h), we depict the time series of the delays of packets received at non-hotspots. We have separate plots for their total delay, their in-fabric delay, as well as for the maximum in-fabric delay. As can be seen in Figs. 13(a,e), the baseline performs quite poorly. The end2end acks scheme, shown in Figs. 13(b,f), yields better throughput than the baseline, but does not improve the delay performance. Although this scheme regulates the rate of injections to each output, it does not avoid nor resolve persistent backlogs that can form inside the switching fabric at the onset of a congestive event.

Figures 13(c,g) depict the performance of the centralized scheduler. For comparison, the performance of the distributed scheduler is shown in Figs. 13(d,h). Both systems maximally

[11]The performance of end2end acks can improve by dimensioning switch buffers to fit $N \cdot W$ packets. However, this solution does not scale well.



(a) baseline (b) end2end acks (c) buffers reservation - centralized (d) buffers reservation - distributed

(e) baseline (f) end2end acks (g) buffers reservation - centralized (h) buffers reservation - distributed

Fig. 13. The evolution of throughput and delay performance in time, under synchronized inputs overloading one after the other a set of randomly selected outputs by a factor of 30×. The remaining (non-hotspot) outputs receive uniform traffic at 0.5 load. The congestive episodes start from time 5000, and each one lasts for 5000 packet times. We plot the delay of packets heading to non-hotspot outputs.

utilize the hotspot and non-hotspot outputs, while keeping the delays virtually unaffected in the presence of hotspots.

Careful examination shows that the total delays of the distributed scheduler spike at the on-set of each hotspot. These transients, which last for less than 1000 packet times, are due to hotspot requests filling up $B \rightarrow C$ request queues. The aggregate flow of requests to each congested output is eventually limited to 1 request per packet time thanks to the hierarchical request flow control. Thus, the duration of the transient is related to the time that it takes for the request network to absorb the excessive requests. This, in turn, depends on the transmission time of requests as compared to their generation rate. Thus, in practice, when an output is overloaded by large messages, the transient behavior will be very short [15]. There are several ways to reduce the duration of these transients even further. For instance, we can speedup the request links $B \rightarrow C$ and the absorption rate of requests inside C switches. In this experiment, this speedup was $1.1\times$. Another way is to reduce u, *i.e.* the number of pending requests per VOQ (32 in this experiment), bringing it closer to (12) one request-grant rtt worth of requests. Request contention can also be reduced further by combining several VOQ requests into one control message and also by piggybacking requests in data messages as examined in [15].

In summary, the results in this section showed that both the centralized and the distributed scheduler incarnations of proactive buffer reservations provide exceptional performance levels under extremely hard traffic condition.

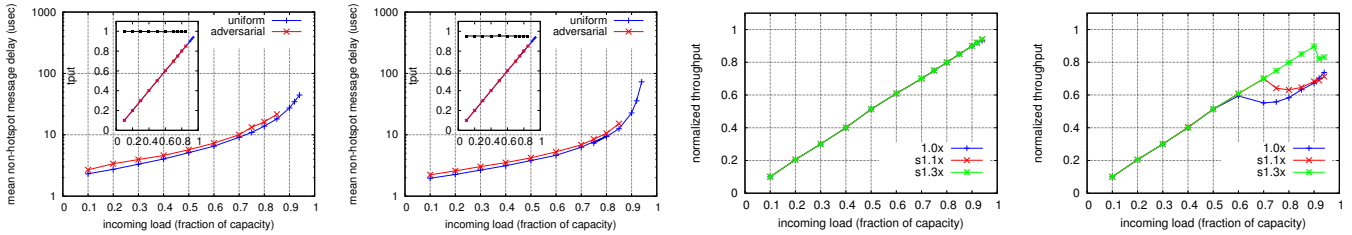
6.5 Comparison with SRP (and CIOQ switching elements)

In this section, we compare SRP (see Section 2) with the proactive buffer reservations variation described in [15]. (We do not compare SRP directly to the systems proposed in this paper, because SRP exploits the large expected size of data messages to reduce the control overhead, whereas our simulation models for the latter systems cannot perform such optimizations.) The system in [15] is based on the principles presented in this paper, but avoids using a separate scheduling network. Instead, the network adapters route request and grants messages over the data network. In addition, the scheme in [15] can combine several requests and grants of the same flow in a single control message, which can also be piggybacked over data packets of the reverse flow to minimize the control overhead.

In our experiments, we consider a 64×64, three-stage Clos network, made of 8×8 CIOQ switches, and 10 Gb/s links. In some experiments, the fabric lines run faster than the external ports by a speedup factor s. CIOQ switches also run faster than the fabric lines by a speedup factor os, and implement four (4) VCs as defined in [16]. Data (application) messages are 2048B long, and are segmented into 256B packets. The size of control messages is set to 8B. The VOQs at each source adapter can store 10,000 2048B messages, and discard new data messages when they are full. Similar to [16], the propagation delay along links is 32 ns, and the CIOQ switches offer a 26 ns cut-through latency and a 16-packet deep queue at each input for each VC. There is credit-based flow control between same-VC queues in adjacent network stages.

There is a noteworthy difference between our model and that used in [16]. The latter considers that CIOQ switches have local VOQs at inputs; this means that there are $4 \cdot M^2$ queues in each switch. Instead, we consider that all packets that arrive at the same input of a switch and belong to the same VC are stored in one FIFO queue, out of which only the HOL packet can participate in arbitration. The CIOQ arbitration is VC aware, serving higher-priority VCs first. In order to mask out first-order HOL effects within the CIOQ switches, we set os= 8, and the size of each VC output buffer to 8×16 packets. Effectively, for the 8×8 switches that we consider in this experiment, each output buffer can absorb up to 8 incoming packets per packet time, thus approaching the behavior of ideal output-queued switches.

For proactive reservations, the output buffers of CIOQ switches in the last fabric stage can store 150 packets. Output



(a) Proactive buffer reservations, Bernoulli arrivals (b) Speculative reservations, Bernoulli arrivals (c) Proactive buffer reservations, 8 hotspots, overload factor $h = 2\times$ (d) Speculative reservations, 8 hotspots, overload factor $h = 2\times$

Fig. 14. Fabric-buffer reservations as in [15] vs. SRP [16].

adapters residing at adapters reserve space in these last-fabric buffers to the requesting VOQs [15]. We also model reorder buffers at the destinations of both systems. As SRP does not enforce an upper bound on the occupancy of the reorder buffers, we set the reorder buffer size to 12K packets. With this empirically selected size, SRP's reorder buffers did not overflow in our experiments. For proactive buffer reservations, each reorder buffer can store 200 packets, each request and grant message can combine up to 20 packets, and each VOQ may have outstanding requests for up to 20 packets [15].

Figures 14(a,b) compare the performance of buffer reservations to that of SRP under uniform and adversarial traffic, similar to that described in Section 6.3. We configured eight (8) hotspots in one C-stage switch, each overloaded by a factor h =2×. As can be seen, the delay of non-hotspot messages in both systems is virtually unaffected in the presence of hotspots. SRP achieves slightly better delay thanks to speculative transmissions that avoid the request-grant latency at low loads. The inset plots the utilization at non-hotspot outputs for both uniform and adversarial traffic, and also the utilization at hotspots. In SRP, the hotspot utilization is close to 0.95 because $\epsilon = 0.05$ [16].

Figures 14(c,d) depict the throughput performance of proactive buffer reservations and of SRP under uniform traffic, driven by bursty message arrivals. Bursty arrivals are generated as described in Section 6.2, with an average burst size of 20, 2048B messages. We have several plots for different fabric speedup factors s. With proactive buffer reservations, throughput goes up to 0.94 for any speedup factor between 1.0 to 1.3×; for the same speedup factors, the throughput of SRP is worse. We attribute these inefficiencies of SRP to missed deadlines. Performance improves as we increase the speedup of the fabric, because the backlogs at input and fabric buffers resolve faster, and thus the output enforced time schedules are more conveniently met.

Similar performance problems with SRP are also present in *implicit rate regulation* [15]. In implicit rate regulation, each output enforces an arrival schedule which is conveyed to inputs based on the time that the output issues the grants—the grants in SRP have an explicit injection-time notification field. In a distributed network, it is very difficult to guarantee that arrivals will keep on following output schedules. Effectively, a Clos network must run significantly faster than input and output links to eliminate out-of-schedule arrivals at outputs. With SRP, this is controlled via parameter ϵ. But the amount of fabric speedup that is required to guarantee conformity to output schedules remains unknown, especially considering the unaccounted burden coming from small messages and from their control overhead. On the other hand, proactive buffer reservations are more robust to random delays. Instead of requiring packets to arrive to destinations at a predefined time, we reserve buffer slots so that packets can queue in front of their destination, without exerting backpressure.

6.6 *Weighted max-min fairness*

With proactive buffer reservations, output (aggregate) flows are protected from each other. The output arbiters that perform the reservations serve per-flow request counters, thus they are able to enforce any output bandwidth allocation they decide. In principle, the final bandwidth allocation will also depend on the arbiters of internal links. However, after buffers have been reserved, the injected packets experience little contention inside the fabric, thus the role of in-fabric arbiters is subordinate.

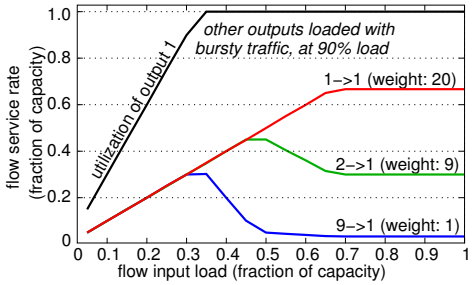


Fig. 15. Weighted max-min fair service for three (3) flows competing for output 1.

In this experiment, we demonstrate *weighted-max-min fair* bandwidth allocation. For this, we replace the RR output arbiters in the centralized design with WRR arbiters. In Fig. 15, we configured three sub-flows: 1→1, 2→1, and 9→1, with weights of 20, 9 and 1, respectively. Each of them is the only active flow at its input, and its load changes along the horizontal axis. Inputs other than 1, 2 and 3, receive a uniform, bursty (background) traffic, at 0.9 load, targeting outputs 2 to 64. The vertical axis depicts flows' normalized service rate, measured at output ports. As long as the aggregate load for output 1 is feasible (up to 0.33 load per flow), all flows' demands are satisfied. At the other end, when all flows are saturated (starting from 0.66 load per flow), each flow gets served at a rate equal to its output fair share, *i.e.* 0.66, 0.30

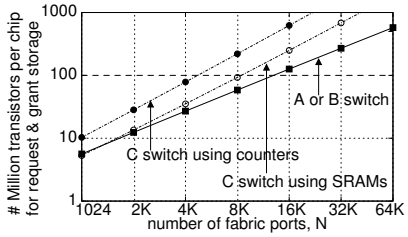


Fig. 16. Millions transistors per switch for the distributed scheduler.

and 0.033. When the load of a flow is below its share, the excess bandwidth gets distributed to other flows, in proportion to their weights.

7 . Implementation considerations

In this paper, we described and evaluated the basic underlying ideas behind proactive fabric-buffer reservations. This section estimates the area overhead of the control subsystem for the centralized and the distributed designs as well as their control bandwidth overheads.

The centralized scheduler requires N^2 request and N^2 grant counters, $log_2(u)$-bit wide each, and N^2 $\log_2(M)$-bit wide distribution pointers. For 20 transistors per counter bit, the central scheduler chip requires $20 \cdot (2 \cdot N^2 \cdot log_2(u) + N^2 \cdot \log_2(M))$ transistors. Moreover, the centralized scheduler faces bandwidth constraints. Effectively, it will be very difficult to scale the centralized alternative beyond one thousand of ports [3].

The distributed organization improves scalability significantly. Every A- or B-switch has M request queues, and each such queue has a capacity of $M \cdot K$ requests. Considering the grant queues as well, we need a storage for $2 \cdot N \cdot K$ messages per switch ($M^2 = N$). The most difficult part is the switches of the last stage (C-switches), which contain $M \cdot N$ request counters and $M \cdot N$, $\log_2(M)$-bit wide, distribution counters. For 20 transistors per counter bit, and $\log_2(u) = 7$, the total cost of counters per C-switch is $20 \cdot M \cdot N \cdot (7 + log_2(M))$ transistors. To reduce the transistor count, several counters can be implemented in an SRAM block, with external adders for increments and decrements. An SRAM bit consumes about 6 transistors, *i.e.* approximately 3 times less than a counter bit.

Figure 16 depicts the number of transistors for request and grant storage, per switch, for various fabric sizes, N. (Each request/grant message requires $\log_2(N) + \log_2(M) = \frac{3}{2} \cdot \log_2(N)$ bits.) As can be seen, fabrics with 16 or 32 K of ports are readily feasible using the integration capabilities of today.

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 adapter indicate the targeted fabric-output port, and the path (B-switch) in the request channel; the grants sent to ingress adapter 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. there are M outputs combined with N inputs to identify and separate from each other. 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 be a request- (grant-) credit issued upstream. These credits need to identify a port ID in the present switch, hence they are log_2M-bit wide, each. Summing up, the bandwidth overhead is (1 req + 1 req-credit + 1 gr + 1 gr-credit) $4 \cdot log_2N$ bits, per packet: for N= 4096, it corresponds to 9.4% for 64-byte, 4.6% for 128-byte, and 2.4% for 256-byte packets. For N= 16K, the respective numbers are 11%, 5.5%, and 2.8%.

The data switches are (re)configured once per port and per time slot, where a time slot is the internal data-packet time. Control switches need to be reconfigured *two* times more frequent (request plus grant vs. packet). Since control messages are significantly smaller than data packets (*e.g.* 20 vs 256B), the cost is not excessively high given the abundance of wires in modern ASICs.

The partitioning of the distributed scheduler in stages and modules (A_i,B_i,C_i) and the flow control of control messages between adjacent scheduling nodes are intended to facilitate distributed multi-rack implementations. In such an implementation, the switch elements would be connected by copper or fiber cables, realizing bidirectional point-to-point channels. This practical arrangement calls for a *two-level fat-tree* topology. If we ignore shortest-path routing (*i.e.* if we always route packets through root nodes), the fat-trees are functionally equivalent to the Clos networks that we consider here.

Also, it would be desirable to route control messages over the same physical links as data [15]. To do so without compromising bandwidth, the links must run faster. However, as described above, the overhead is fairly low. In practice, the overhead becomes significantly small if we consider that one request or grant message may correspond to multiple packets from the same flow, and that control messages can be piggybacked in data packets. Note that it is also possible to logically partition the capacity of links into data and control slices.

8 . Discussions

Proactive buffer reservations bear a few similarities to TCP, in the sense that both schemes control the flow towards endpoint buffers. However, there are important differences between the two, which are critical for high-performance switching fabrics. First, the endpoint buffers that TCP flow controls are located in the network stack of the receiving host; hence TCP cannot make any guarantees about the filling level of the fabric buffers that we bound with proactive buffer reservations. Second, TCP reduces the congestion window upon detecting packet drops. In contrast, the proactive scheme that we propose takes informed proactive decisions, and does not rely on conservative rate control to ensure stability [26]. In addition, by incorporating scheduling in output buffer allocation, the proposed scheme allocates exact fair rates to the competing flows, and can schedule short flows on time. On the other hand, TCP converges to an approximately fair allocation

after several RTTs, and may mistreat badly the delay-sensitive short-lived flows that are blooming in datacenters [27]. Finally, by not relying on packet drops to detect congestion, as TCP does, proactive buffer reservations do not suffer from long timeouts, which can further increase the completion time of delay-sensitive flows.

Having acknowledged the aforementioned shortcomings of TCP in datacenter networks, recent research has been examining efficient TCP variants. These mainly aim at reducing the large in-fabric backlogs and the excessive packet drops of traditional TCP in order reduce packet delays and to tackle TCP incast congestion [17]. With proactive buffer reservations, we minimize the in-fabric delays by keeping the backlogs at the input VOQs, allowing inside the fabric only an admissible portion of traffic. The cost of such informed decisions is a scheduling subsystem which nevertheless can be simplified in an implementation.

It is frequently quoted that the request phase of proactive schemes needlessly increases packet latency. In reality, the extra latency amounts to up to a couple of microseconds, which is a no-issue considering that the deadlines of flow completion times typically measure in milliseconds [17]. Furthermore, a proactive scheme can save many tens of microseconds of queuing, or even milliseconds, if we account for software re-transmissions. For applications that are extra-sensitive to latency, *e.g.* inter-processor communication, it is also possible to bypass the request phase under suitable traffic conditions [4], [16], [30].

In this paper, we separated the data network from the scheduling network to emphasize the underlying principles and some important trade-offs. Such a broader understanding allows one to think of alternatives on how to separate data from control in a cost-effective way. In addition, having separate networks gives optimal results, and thus demonstrates the full potential of the proposed methods. Additional results in Sec. 6.5 showed that, for realistic packet sizes, similar performance levels can be obtained when the two networks are merged together.

9 . Conclusions

We proposed congestion control schemes based on proactive fabric-buffer scheduling schemes. The methods that we described unify the ideas of buffered and bufferless flow control and scheduling. We also described centralized and distributed implementations of the control/scheduling units, suitable for distributed multi-rack implementations.

There are multiple positive interactions between multi-path routing and fabric-buffer reservations. The former avoids persisting contention on unlucky internal links that could otherwise occur due to routing conflicts; this allowed us to simplify the buffer reservations without compromising performance. On the other hand, proactive buffer reservations enable multi-path routing in the first place by bounding the size of re-sequencing buffers. Finally, they both contribute to reducing the delay and delay-jitter inside the fabric (since HOL blocking is minimized equally across all possible paths), thus also minimizing the additional latency incurred at the re-sequencing unit.

Extensive and detailed computer simulations of the complete system demonstrated sophisticated QoS, including *(a)* immunity of packet delay in the presence of multiple concurrent hotspots, *(b)* low in-fabric delays and delay variations, and *(c)* weighted max-min fair allocation on a per-port basis. As demonstrated in Sec. 6.5, using the proposed methods, the delay of non-congested packets slightly exceeds 10 microseconds with fabric utilization up to 80 percent.

As future work, we plan to investigate how to eliminate the need for intra-fabric backpressure from the first two fabric stages.

10 . Acknowledgments

The authors would like to thank the anonymous reviewers for their extensive feedback that greatly helped us to improve the quality of present manuscript. Nikolaos Chrysos would also like to thank Cyriel Minkenberg, Mitch Gusat, Fredy Neeser, Kenneth Valk, Martin Schmatz and Claude Basso for inspiring discussions on realistic traffic patterns and on implementation issues. Finally, Anne-Marie Cromak is thanked for helping us to improve the presentation of the manuscript.

References

[1] G. Sapountzis, M. Katevenis: “Benes switching fabrics with O(N)-complexity internal backpressure", *IEEE Communications Magazine*, vol. 43, no. 1, January 2005, pp. 88-94.

[2] J. Duato, I. Johnson, J. Flich, F. Naven, P. Garcia, T. Nachiondo: “A new scalable and cost-effective congestion management strategy for lossless multistage interconnection networks", *Proc. HPCA*, San Francisco, Feb. 2005

[3] N. Chrysos and M. Katevenis: “Scheduling in non-blocking buffered three-stage switching fabrics", *Proc. IEEE Infocom*, Barcelona, Spain, Apr. 2006.

[4] N. Chrysos: “Congestion management for non-blocking Clos networks", *ACM/IEEE ANCS*, Orland, USA, Dec. 2007.

[5] P. S. Sindhu, R. K. Anand, D. C. Ferguson, B. O. Liencres, "High speed switching device", United States Patent No. 5905725.

[6] O. Iny, "Method, device, and system, of scheduling data transport over a fabric", United States Patent Application No. 20070253439.

[7] CISCO Systems, "A day in the life of a fibre channel Frame: CISCO MDS 9000 family switch architecture", White Paper, 2006.

[8] N. Chrysos, F. Neeser, B. Vanderpool, M. Rudquist, M. Valk, C. Basso: “Integration and QoS of multicast traffic in a server-rack fabric With 640 100G ports", *Proc. ACM/IEEE ANCS*, Los Angeles, CA, USA, Oct. 2014.

[9] P. Pappu, J. Turner, K. Wong: “Work-conserving distributed schedulers for terabit routers", *Proc. ACM SIGCOMM*, Sept. 2004

[10] N. Alfaraj, J. Zhang, Y. Xu, H. J. Chao: “HOPE: Hotspot congestion control for Clos network on chip", *Proc. NOCS*, May 2011.

[11] T. Anderson, S. Owicki, J. Saxe, C. Thacker: “High-Speed switch scheduling for local-area networks", *ACM Trans. on Computer Systems*, November 1993.

[12] N. McKeown: “The *i*SLIP scheduling algorithm for input-queued switches", *IEEE/ACM Trans. on Networking*, April 1999.

[13] L. Chen, N. Chrysos: “Performance of random arbitration in switching fabrics", IBM Research Report, RZ3856, 2013.

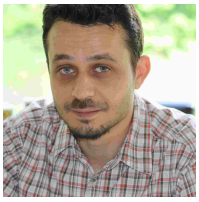
[14] R. Rojas-Cessa, E. Oki, H.J. Chao: “CIXOB-k: Combined input-crosspoint-output buffered switch", *Proc. IEEE GLOBECOM*, San Antonio, Texas, Nov. 2001.

[15] N Chrysos, L. Chen, C. Minkenberg, C. Kachris, M. Kateveni: “End-to-end congestion management for non-blocking, multi-stage switching fabrics using commodity switches", IBM Research Report, RZ3792, 2010.

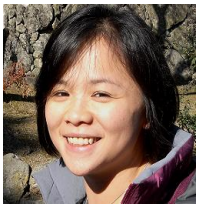
[16] N. Jiang, D. U. Becker, G. Michelogiannakis, W. J. Dally: “Network congestion avoidance through speculative reservations", *Proc. High Performacnce Computer Architecture (HPCA)*, New Orleans, Feb. 2012.

[17] M. Alizadeh, et al.: “Data center TCP (DCTCP)", *Proc. ACM SIGCOMM*, New Delhi, India, Aug. 2010.

[18] D. Zats, et al.: “DeTail: reducing the flow completion time tail in datacenter networks",*ACM SIGCOMM Computer Communication Review*, vol. 42. no. 4, pp. 139-150, 2012.
[19] F. Gilabert, M.E. Gomez, P. Lopez, and J. Duato, “On the influence of the selection function on the performance of fat-trees", *Proc. European Conf. on Parallel Computing* , Aug. 2006.
[20] X. Yuan, W. Nienaber, Z. Duan, and R. Melhem: “Oblivious routing for fat-tree based system area networks with Uncertain Traffic Demands", *ACM SIGMETRICS*, San Diego, CA, 2007.
[21] J. Kim, W. J. Dally, and D. Abts: “Adaptive routing in high-radix Clos networks", *ACM/IEEE Proc. Conference on Supercomputing*, Tampa, FL, USA, 2006.
[22] N. Chrysos, et al. “All routes to efficient datacenter dabrics", *Proc. ACM OCMC*, Berlin, Germany, Jan. 2014.
[23] A. Dixit, et al. “On the impact of packet spraying in data center networks", *Proc. IEEE INFOCOM*, Apr. 2013.
[24] D. Crisan, A.S. Anghel, R. Birke, C. Minkenberg, M. Gusat: “Short and fat: TCP performance in CEE datacenter networks", *IEEE Hot Interconnects*, Santa Clara, Aug. 2010.
[25] C.-Y. Hong, M. Caesar, and P. B. Godfrey: “Finishing flows quickly with preemptive scheduling", *Proc. ACM SIGCOMM*, Aug. 2012.
[26] M. Alizadeh, S. Yang, S. Katti, N. McKeown, B. Prabhakar, S. Shenker: “Deconstructing datacenter packet transport", *Proc.11th ACM Workshop on Hot Topics in Networks*, NY, 2012.
[27] Y-T. Li, D. Leith, and R. N. Shorten: “Experimental evaluation of TCP protocols for high-speed networks" *IEEE/ACM Trans. on Networking*, vol. 15, no. 5, Oct. 2007.
[28] Y. Chen, R. Griffith J. Liu, R. H. Katz, A. D. Joseph: “Understanding TCP incast throughput collapse in datacenter network", *Proc. WREN’09*, Barcelona, Spain, 2009.
[29] D. Crisan, R. Birke, N. Chrysos, C. Minkenberg and M. Gusat: “Fabric: How to Virtualize Lossless Ethernet?", *Proc. IEEE Cluster*, Madrid, Spain, Sept., 2014.
[30] C. Minkenberg and M. Gusat: “Speculative flow control for high-radix datacenter interconnect routers", *Proc. IPDPS*, Long Beach, CA, USA. Mar. 2007.



Nikolaos Chrysos received his Ph.D. degree in Computer Science from University of Crete in 2007. His Ph.D. research on congestion control for multi-stage networks was awarded an IBM Ph.D. Fellowship in 2003 and 2004. He was a visiting assistant professor at Technical University of Crete in 2007 and 2008 and a visiting scientist at FORTH. From 2009 to 2014, he was with IBM Research–Zurich, where he played key decisive roles in the design and ASIC implementation of a high-performance server-rack fabric for 100G Ethernet, and of 136×136×25G (crossbar-like) switch. Since October 2014, he is with FORTH, working on high-speed networks and low-power, holistically-designed systems. He is a member of ACM, and (co-)inventor of four granted patents.



Lydia Chen received her Ph.D. degree in Computer Science from Upen. Dr. Lydia Chen is a performance analyst at IBM Zurich Research Lab since 2007. She received her Ph.D. in Operations Research and Industrial Engineering from Penn State University. She completed her undergraduate studies at National Taiwan University and British Columbia University. She has published papers in several international conferences, such as Sigmetrics, Infocom, Globecom, ICC, CGO, CCgrid, ICPE, Middelware and DSN. She has also served as a TPC member on a number of networking and performance conferences, and she is a member of the ACM and of IEEE.

Christoforos Kachris Christoforos Kachris is a senior researcher at Athens Information Technology (AIT), Greece. He obtained his Ph.D. in Computer Engineering from Delft University of Technology, The Netherlands in 2007, and the diploma and the M.Sc. in Electronic and Computer Engineering from the Technical University of Crete, Greece in 2001 and 2003 respectively. From February 2009 till August 2010 he was a visiting assistant professor at the University of Crete, Greece and associate researcher at the Institute of Computer Science in the Foundation for Research and Technology (FORTH) working in the HiPEAC NoE and the SARC IP European research projects. In 2006 he was a research intern at Xilinx Research Labs, San Jose, CA, working at the Networks Group. His research interests include reconfigurable computing (FPGAs), multi-processor SoC (MPSoC) and network processing.

Manolis Katevenis received his Ph.D. degree in Computer Science from the University of California, Berkeley, in 1983. From January 1984 to March 1985 he was Assistant Professor of Computer Science at Stanford University, Stanford, California, USA. Since September 1985, he is with the University of Crete, Dept. of Computer Science, where he is currently a Professor. Since 1985, he is also with the Institute of Computer Science (ICS), FORTH, Heraklion, Crete, where he is currently the Head of the Computer Architecture and VLSI Systems Laboratory. His interests are in Embedded and Scalable Multiprocessor System Architecture, Interprocessor Communication Mechanisms, Interconnection Network Architecture, Packet Switch Architecture, Computer Architecture, and VLSI Systems.