RZ 3792 (# Z1011-002) 11/18/2010
Electrical Engineering 13 pages

Research Report

End-to-End Congestion Management for Non-Blocking, Multi-Stage Switching Fabrics using Commodity Switches

N. Chrysos*, L. Chen*, C. Minkenberg*, C. Kachris‡, M. Katevenis‡

*IBM Research – Zurich
8803 Rüschlikon
Switzerland

‡Institute of Computer Science
Foundation for Research and Technology - Hellas

End-to-End Congestion Management for Non-Blocking, Multi-stage Switching Fabrics using Commodity Switches

Nikolaos Chrysos
IBM Research
Zurich Research Laboratory
cry@zurich.ibm.com

Lydia Chen
IBM Research
Zurich Research Laboratory
yic@zurich.ibm.com

Cyriel Minkenberg
IBM Research
Zurich Research Laboratory
sil@zurich.ibm.com

Christoforos Kachris
Institute of Computer Science
Foundation for Research and
Technology - Hellas (FORTH)
kachris@ics.forth.gr

Manolis Katevenis
Institute of Computer Science
Foundation for Research and
Technology - Hellas (FORTH)
kateveni@ics.forth.gr

ABSTRACT

We propose and evaluate end-to-end, proactive congestion management schemes for non-blocking multi-stage switching fabrics. All control functions are delegated to network adapters, which coordinate among each other through a variable-grain request-grant protocol. Switching elements simply forward packets, regardless to whether these correspond to payload data or control messages. We examine two different schemes. The first one merely regulates the long-term injection rate towards each output. For this scheme we show pathological states that may appear in practice, and conclude that it is stable only if some internal speedup is employed. The second scheme reserves buffer space inside the some switching elements before granting segments. Using a fluid model, we show that this scheme eliminates congestion in Clos/Benes fabrics for any number of stages, and detailed computer simulations demonstrate its robust operation without requiring speedup. Finally, for this distributed set-up, we show how to reduce the size of reassembly buffers, so as to fit them into SRAM buffers at the network adapters.

1. INTRODUCTION

Packet-switched networks are encountered at the heart of scalable network routers and data center (or high-performance computer) interconnects. The overall system performance increasingly depends on the performance of its interconnection network. As these networks scale to larger port counts, and their utilization increases, congestion management becomes indispensable. At the same time, technology constraints rule out monolithic bufferless switches with centralized schedulers, and impose buffered multi-stage switching fabrics with distributed control [1]. These trends have for some time now called forth research and products [2, 3, 4, 5, 6], which applied the request-grant philosophy of bufferless crossbars to make buffered multi-stage switching fabrics practical and efficient. The present work makes some steps further along this direction.

In settings where the network is expected to be highly loaded by workloads that cannot be characterized in advance, a *non-blocking* fabric is often necessary to meet performance targets, where non-blocking means that any conflict-free traffic pattern can be routed without creating contention for any link in the fabric. Typical examples of scalable non-blocking networks encountered in such systems are Clos or Beneš networks [7] [8], and fat trees [9]. In this paper we consider such non-blocking, multi-stage networks, although some of the results may also be applicable to other networks.

As bufferless multi-stage networks require complicated centralized control mechanisms to create conflict-free matchings, which typically scale poorly to large port counts, such large-scale networks are usually built using switching elements (or switches) with some amount of integrated SRAM buffers. To prevent excessive drop rates because of these small buffers overflowing, hop-by-hop *link-level flow control (backpressure)* is employed in between switches. A well-known phenomenon that can seriously degrade the performance of any buffered flow-controlled, multi-stage network is *saturation-tree congestion* [10]. This can occur when a network link is oversubscribed, i.e., the aggregate rate of traffic wanting to cross that link exceeds its capacity. We refer to such a link as a *hotspot*. Congestion can spread because of backpressure, and create a saturation tree of congested links rooted at the initial hotspot, thus deteriorating performance of the entire network.

Such saturation trees can be counteracted by *congestion management* mechanisms. These can coarsely be classified into *reactive* and *proactive* schemes. Reactive schemes do not impose constraints on traffic injection a priori, but wait until some congestive event occurs before they initiate corrective action. Proactive schemes, on the other hand, attempt to shape traffic injections such that congestion cannot occur, or can only occur in a very limited fashion such that no saturation trees are induced. TCP is an example of a scheme that exhibits both proactive (slow start) and reactive (reduce window on duplicate ACK or time-outs) characteristics. The advantage of reactive schemes is that a communication requires no explicit setup (e.g., handshake, request/grant), which avoids the latency penalty of having to do so. On the other hand, proactive schemes can avoid congestion risks by design, and offer better worst-case performance. In this paper, we adopt a hybrid proactive model that skips the request/grant setup at low loads for small messages; however, the proactive spirit of the scheme is preserved, as we reduce such unsolicited injections to one segment per flow and round-trip time.

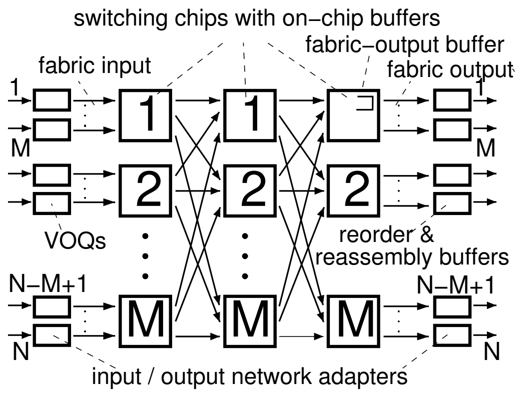


Figure 1: Non-blocking, buffered switching fabric.

The congestion management methods we propose in this paper are particularly suitable for non-blocking multi-stage networks, and are tightly coupled with the load balancing that these networks use. In particular, we treat only fabric-output links (see Fig. 1) as possible hotspots, and try to prevent them from saturating. As implied in [11] [12], and further substantiated here, random load balancing ensures that, when fabric outputs are not saturated, also the internal links of the non-blocking network do not experience congestion, so that no additional measures to reduce the load at internal links are needed.

1.1 Background and contributions

Recent research efforts have demonstrated that proactive (request/grant) congestion management can provide superior performance in non-blocking multi-stage switching fabrics. In essence, the request-grant scheme lifts the congestion avoidance burden from the data network, and places it on the scheduling network. Therefore, it becomes critical that requests for overloaded outputs do not interfere with requests for other outputs, otherwise blocking inside the scheduling network can deteriorate performance as much as blocking in the data network can.

From the industry side, a first attempt in this respect was made in [2], which employs per-flow request counters inside the switches to isolate requests from different flows[1]. Each request, referring to a single VOQ cell, is transferred to the output adapter, which immediately generates a grant and routes it back to the waiting input. As requests for congested outputs are throttled inside the request network, the grants generated also throttle the injections towards congested outputs. From academia, two schemes, published at about the same time, used a request-grant scheme, and a central scheduler responsible for reserving buffer space before issuing the grants: [4] considers a single-stage fabric, and reserves buffer space in output adapters, whereas [12] considers a 3-stage Clos fabric, and reserves space in both the reorder buffers of output adapters and the fabric buffers in front of the targeted fabric output (see Fig. 1). By reserving space in these fabric-output buffers before packets are injected, [12] shows that the fabric behaves transparently to congestion epochs: saturation trees are avoided for any traffic conditions. Both [12] [4] employ a centralized scheduler, which limits scalability.

[1]We define a flow as a distinct input/output adapter pair.

In [13], scalability is improved by distributing the control functions over the last-stage switches of the 3-stage Clos fabric. These switches in addition comprise per-flow request counters (N counters per node). Other switches store requests in queues that are shared among flows heading to different outputs[2]. With these structures, hierarchical flow control is enabled in the scheduling network that reduces request interferences to a great extent. On the downside, control functions are implemented in switches, whereas private links are assumed to convey request-grant messages. Our paper addresses the weaknesses of this scheme:

1. We propose and evaluate end-to-end congestion management. All control circuits are placed at input and output adapters, which coordinate in a variable-grain, request-grant protocol. Switching elements have no duty other than forwarding packets, without knowing whether these are request/grants or payload data. The variable-grain protocol reduces the control bandwidth overhead, as heavy or output-constrained flows converge to coarse grants, and avoids the underutilization and large delays that are pertinent to coarse-grain (or frame-based) scheduling by serving lightly loaded flows with fine-grained grants.

2. For this distributed setup, we examine an *implicit rate regulation scheme*, demonstrate its weak points, and show, by means of computer simulations, that its stable when some internal speedup is used[3].

3. We also revisit the scheme in [13]. To what was already known for this scheme, the present paper *(a)* adds its adapter-based implementation, with variable-size requests and grants; and *(b)* provides new evidence demonstrating that, when coupled with random, per-flow load-balancing, it virtually eliminates saturation trees throughout non-blocking multi-stage Clos/Benes networks with *any number of stages.*

4. Finally, we introduce a coordination protocol between inputs and outputs, that enables fitting the reassembly buffers in small SRAMs, where typically their space grows with N maximum-size packets, with N being the number of fabric ports.

Our results on congestion control: Enforcing injection rates, and reserving buffer space, are two candidate schemes for proactive congestion management. Implicit rate regulation defines a practical way to enforce rates: each output issues grants to the requesting inputs at a peak rate that the corresponding fabric-output can handle. For instance, when an output issues a grant for two (or ten) segments, it waits for two (or ten) segment times before issuing a new grant. The output arbiters in [13] reserved buffer space in order to avoid congestion trees; but also, in order to reduce complexity, these arbiters generated at most one grant per cell time, which corresponds to fine-grain, implicit rate regulation. Thus, it is still left unknown whether we can rely on implicit rate regulation, alone, for congestion management.

In this paper we find that implicit rate regulation works well in many cases, but not always. As there are multiple contention points in the system, we cannot ascertain that granted segments will reach the output, just on the time

[2]Because of load-balancing, O(N^2) flows pass through each of these nodes; thus the implemention of per-flow request counters in them becomes intractable.

[3]While we were working with implicit rate control, we found a paper from industry that appears to be using something similar [6]. This independent work present neither a detailed description, nor provides performance evaluations.

that the output has "prescheduled" for them, i.e. ≈ about one end-to-end round-trip-time (RTT) after the grant has been issued. As we will see, this uncertainty can lead to a destructive, positive feedback loop, where short-term contention brings out-of-schedule segments, overloaded outputs, and finally congestion trees, which intensify contention, and spread the out-of-schedule segments all over the switching fabric. Our results show that one can remedy this behavior by using a speeded-up fabric. Although it is still unclear whether (or which) speedup can avoid the regeneration of saturation trees, our experiments indicate that a small speedup may already suffice. The drawback is that this speedup is off-chip, and thus power-hungry and expensive.

On the other hand, when we reserve space in fabric-output buffers before issuing grants, saturation trees will be avoided even without requiring internal speedup. This method essentially trades buffers for speedup, because when an output issues a coarse grant, e.g. for the payload of Ethernet Jumbo frame, it needs to reserve an equal amount of space in the corresponding fabric-output buffer in the upstream switch of the last stage. Although this might have been prohibitive in the past, it is no longer so as each new generation of CMOS almost doubles the capacity of on-chip memory. Current 32 nm CMOS, or 22 nm in the 2011-2012 time frame and forward, makes it possible to place more than 64 Mbits of SRAM (or eDRAM) on-chip. With this amount of memory, we can allocate more than 50 KBytes for packet buffers per output port of a 128×128 switch. This space fits not only one but *multiple* 9 KBytes Jumbo Frames, allowing some output overprovisioning that can mask out scheduling inefficiencies, i.e. that multiple outputs reserving space for the same input at the same time.

In the next subsection, we present related work, and in Sec. 2 the end-to-end congestion management schemes. Then, in Sec. 3, we describe the organization of switching elements and the operations performed by network adapters. Finally, Sec. 4presents performance evaluation, and Sec. 5 concludes.

1.2 Related work

Per-flow queues: In single-stage crossbars, congestion can easily be controlled by using virtual output queues (VOQs) [14]. The best established congestion management schemes for multi-stage fabrics are based on the same principle [15] [16]. These approaches do not scale well because of the buffers required in the switching elements grow with N, the number of final destinations. However, their performance is unbeatable, and they serve as landmarks that have inspired many of subsequent efforts including the present one.

Reactive flow control: A recent example of a reactive scheme relevant to data center networks is the Quantized Congestion Notification (QCN) scheme [17], which is standardized for 10G Ethernet (Data Center Bridging [18]). Another reactive congestion management scheme is *Regional explicit congestion notification (RECN)* [19]. The key point in RECN is that a shared queue can accommodate all non-congested flows, with no performance losses; therefore, RECN dynamically allocates set-aside-queues (SAQs) on a per congestion tree basis. Both QCN and RECN aim to tackle the difficult and long-standing problem of congestion management in arbitrary (possibly blocking) topologies by relying on congestion notifications issued from the congested points.

For the non-blocking topologies that we consider here, we avoid the need for special hardware in switching elements. Furthermore, by acting proactively, we avoid the mistimed reaction and oscillations that are pertinent to most reactive schemes. On the downside, our scheme requires the exchange of request-grant messages. However it is not clear what is the exact control message (congestion notifications) overhead of the reactive schemes, since this relates to timely reacting on saturation trees.

Explicit rate regulation: Reference [11] presents end-to-end rate congestion management for non-blocking multi-stage switching fabrics that performs similar functions with implicit rate regulation but by enforcing (per-output) injection rates at each input. For rates to be accurate, all inputs may need to be notified when some flow becomes active. Thus, the rate adjustments in [11] are made fairly infrequently (e.g., every 100 microseconds) so as to amortize the communication overhead. These large control delays may hurt the performance. By contrast, in our scheme, inputs do not have to be explicitly informed of a new flow's, f, appearance, and can continue injecting granted data: inputs will "sense" f's arrival when their grant rate gets reduced, i.e., one RTT after f becomes active.

The Trueway Switch: The Trueway switch [20] comprises multiple, stacked three-stage Clos networks, and employs end-to-end, credit-based (not request/grant) flow control to manage the per-input reorder and reassembly buffers at output adapters. The buffer space required at each output adapter is N end-to-end RTTs worth of segments, plus N maximum-size packets. In addition, the Trueway partitions the VOQs per plane and per path. Although this removes first-order HOL blocking, it does not deal with blocking due to congestion trees rooted at fabric outputs. In the present paper, we tackle blocking not by using additional queues, but with end-to-end request-grant schemes.

Centralized Scheduling and the PPS architecture: The PPS [21] is a three-stage fabric in which the large (and expensive) buffers reside in the central stage. First- and third-stage switches each serve a single external port of high bandwidth. The port count of a PPS can be increased by having multiple slower (sub)ports for the first- and last-stage switches. Such a three-stage architecture is actually similar to the NEXUS fabric, a descendant of the MDS, from CISCO Systems [3], which increases the capacity of the MDS by stacking multiple, parallel switching fabrics, switching fabric being a large crossbar. To prevent blocking, these systems employ a central scheduling unit, which seems to be similar to the approach described in [4] [12]. However, critical details regarding the operation of the central scheduler are not publicly available.

2. CONGESTION CONTROL

In this section we first discuss the overall switching and load-balancing techniques that we use, and then describe the end-to-end congestion control methods. The details of the request-grant scheduling protocol are described in the Sec. 3.

2.1 Overall switching architecture

We consider a non-blocking, multi-stage switching fabric. For our evaluations, we will use a (rearrangeably) non-blocking three-stage Clos network (Fig. 1).

For convenience, we sometimes refer to fabric-inputs and fabric-outputs as inputs (or sources) and outputs (destinations), respectively. A flow is defined as a distinct input-output pair. Every network adapter has an ingress path (input adapter), from which it receives packets, and an egress path (output adapter), through which it forwards packets. We assume that each adapter can communicate control messages from its ingress to its egress side, and vice versa, without having to go through the switching fabric.

The main packet buffers are the VOQs, which are implemented in DRAM modules and positioned at the input adapters in front of the fabric. The output adapters also have buffers (mainly for reordering and reassembly), but these are flow-controlled by the request-grant scheme, so that they can be implemented using smal SRAMs, even in scenarios with internal speedup. The switching elements are described in Sec. 3.1; these contain on-chip buffers subject to hop-by-hop flow control and are agnostic to congestion control functions and messages.

We assume that the network traffic comprises variable-size packets up to $\max_p$ in size. When a packet arrives at an input adapter, it is first stored in the VOQ corresponding to the targeted output. The packet will be divided into one or more variable-size segments. Each segment may carry fragments (or the entire payload) from multiple packets [22]. Segments can be as small as a minimum-size packet, $\min_p$, whereas their size can grow up to a maximum, $\max_s$, corresponding to several minimum-size packets. In our evaluations, $\max_s = 256$ B. These variable-size segments avoid padding overhead and reduce header overhead.

The congestion control functions are implemented exclusively at network adapters. Output adapters collect input requests and grant them while trying to ensure that, in the mid to long term, the arrival rate at each fabric-output port does not exceed the capacity of that port. With random load balancing on a per-flow basis, the same guarantee is automatically valid for any link of the multi-stage (non-blocking) fabric. Of course, load-balancing works on discrete, autonomously routed units, and cannot be perfect in the short term. To obtain the best possible results, we uniformly distribute the load from each flow on a per-segment basis along all available paths[4]

Requests and grants carry a size field, enabling an adaptable admission granularity. Effectively, heavy flows, e.g., those targeting overloaded destinations, can request and be granted tens or hundreds of segments in a single transaction. In addition, we piggyback new request messages on the granted data segments that input adapters inject, such that the request traffic volume practically drops to zero for heavily-loaded or output-constrained flows,

One drawback of such a request-grant protocol is that it increases the delay of flows by the request-grant scheduling delay. On the other hand, request-grant efficiently resolves persistent hotspots, which can deteriorate throughput and delay for longer time periods. As in [23] [13], we can avoid this latency penalty for small packets by allowing each flow to inject a few *unsolicited segments* without first requesting and getting permission. These unsolicited segments may carry multiple small packets, and can also convey the first requests that initiate bulky (e.g., a large packet or a train of small ones) or output-constrained transfers. Here, we minimize this "priviledged" traffic to *one* unsolicited segment per flow.

[4]Routing the entire payload of packets through a single path can cause serious load imbalances among network paths and needlessly overload internal links.

2.2 Congestion control methods

The congestion control functions are delegated to output adapters. Every output adapter holds a request counter for every input adapter, which maintains the number of pending requests that this input has issued to this output. An output arbiter in the adapter serves input requests and issues the corresponding grants. In addition, the arbiter maintains the available space in the reorder/reassembly by means of credits and grants segments only after reserving space for them in this buffer. In this way, the maximum number of segments inside the fabric that are destined to a common output adapter never exceeds the size B of the reorder/reassembly buffer in segments[5].

2.3 Implicit rate control

With implicit rate regulation, the output arbiter introduces an idling period after granting an input. The length of this period is proportional to the size of the grant issued (i.e., the size of the corresponding request counter) and to the rate (measured in segments per time unit) L of the corresponding fabric-output port. In this way, the output arbiter can ensure that in a time interval of T segment times (measured for maximum-size segments), it grants at most $G(T) \leq L \cdot T$ requests. An example of implicit rate regulation is shown in Fig. 2(a). Here output 3 first grants the five requests from input 1, and issues the grant. Then, it stays idle for 5 segment times. It does not grant input 2, which in the meanwhile has also presented a request for four segments, but only after this idling period has finished.

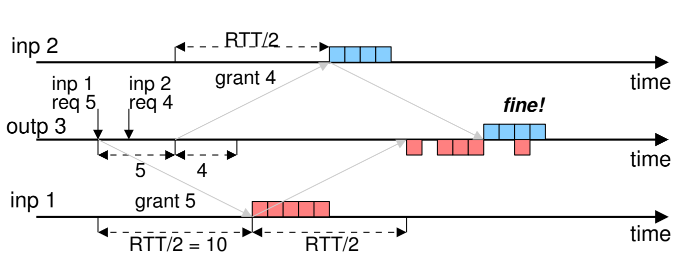
Essentially, the output arbiter programs the arrivals at the buffer in front of the output of the upstream switching element[6] for one RTT time ahead in the future, serializing them at the link rate, so that the corresponding fabric port can immediately fetch them without letting backlogs build up. Note that output adapters never exert backpressure on fabric-output buffers, because space for segments in the reorder and reassembly buffers has already been reserved.

Unfortunatelly, "output schedules" produced in this way are not very robust, and may be violated if different inputs respond to the output "call" with different delays. This may happen when for instance there are inherent differences in the RTTs (due to the packaging of the system), or when the *effective RTTs* vary because of contention. When these discrepancies are small, the fabric-output buffers can mask them out without filling up, as shown in Fig. 2(a). However, as we describe below, discrepancies can be large, and regenerating.

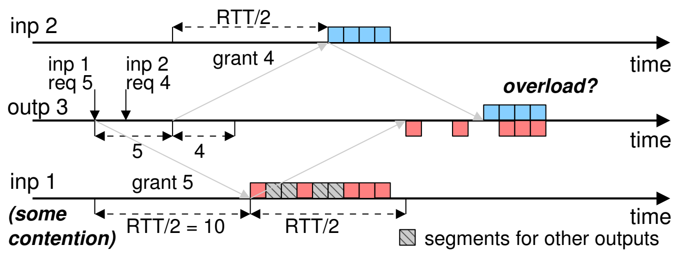
Discrepancies caused by non-uniform RTTs: In a real system, a signal transferred from one adapter to another will experience some in-flight delay. Correspondingly, we define the *rtt* delay as the time it takes for a signal to be transfered from adapter A to adapter B, and return to adapter A. Consider that A is the adapter issuing the grants, and that adapters B and C have requests pending to adapter A. The rtt delay between A and B may differ from the rtt delay between A and C. We assume that the maximum difference among the rtts between any adapter and adapter A is Δrtt.

[5]B is expected to be on the order of a few megabits.

[6]We will use the name fabric-output buffer to distinguish this buffer from the output buffer in the adapter.



(a) A good scenario.



(b) A bad scenario.

Figure 2: Implicit rate regulalion.

Consider now that adapter A first grants adapter B, and then adapter C. According to implicit rate regulation, the arrivals from C to the fabric-output should start just after the arrivals from B have finished. However, the arrivals may be brought closer in time by up to Δrtt, if the largest rtt is between adapters A and B and the smallest rtt between adapters A and C. Conversely, the arrivals may be spread by up to the same Δrtt, when the large rtt is between A and C. We can rigorously show the following:

Lemma 1. *The maximum backlog in fabric-output buffers due to variances in the rtts is less or equal to one Δrtt worth of segments.*

According to the lemma, if Δrtt is 400 ns, and the link rate is 100 Gb/s, then a fabric-output-buffer of 5 KBytes will be able to absorb the discrepancies without filling up.

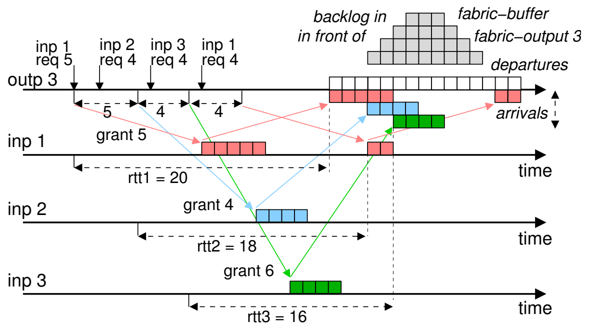


Figure 3: Implicit rate regulation under different rtts. Note that the backlog at the fabric-output buffer never exceeds Δrtt worth of segments, i.e., 4 segments in this example.

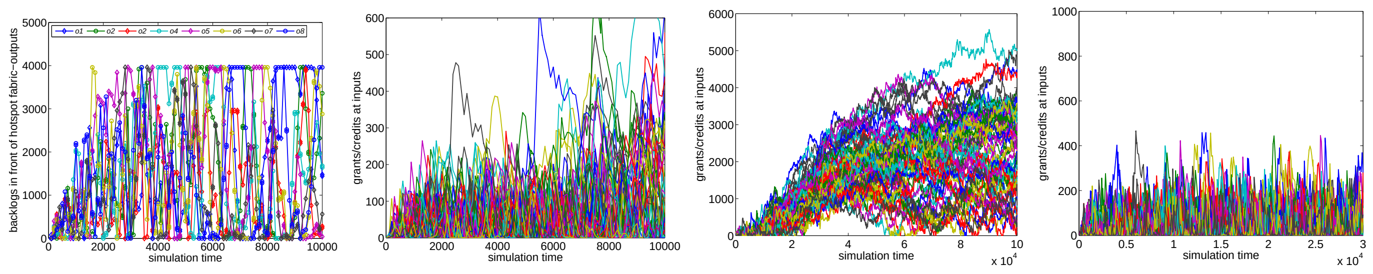
An example showing the formation of a worst-case backlog for $\Delta rtt=$ 4 segment times is shown in Fig. 3. Output 3 first grants inputs in *sequence of decreasing rtts*, thus bringing the respective arrivals closer in time. When the segments from the last input granted, 4, arrive, the backlog in the fabric-output buffer reaches its maximum of 4 segments. This decreasing-rtt sequence is reversed at the end, when output 3 grants input 1 again, which has the largest rtt (20), preceding a grant having been to the input (3) with the smaller rtt (16). Effectively, arrivals are now spread apart, making enough time for the fabric-port to drain that backlog in the fabric-output buffer.

Discrepancies due to contention: Output schedules may be violated because of contention, especially at input adapters. An example is shown in Fig. 2(b). When an input receives many multiple grants from more than one output close in time, it will delay injecting some of the granted segments. Eventually, it may inject out-of-schedule, thus inducing a transient saturation tree. Depending on its severity, the saturation tree may damage the capability of the fabric to bring segments to their output on time, by blocking the injections from *any* input adapter, and eventually causing even higher deviations with regard to *any* output's schedule, and thus new congestion trees, etc., in a destructive positive feedback-loop. In Fig. 4, we depict the results of a similar scenario occurring in computer simulations.

The loop depicted in Fig. 4 was triggered by an input receiving multiple grants from two or more outputs. Obviously, with coarse-grained grants, such "small" accidents are more dangerous than with fine-grained grants. Inputs that are temporarily "congested" in this way return grants at their limited rate, whereas outputs, being oblivious of input contention, may continue to grant them faster, thus increasing the backlog of grants [24]. We have observed that in such scenarios, outputs may run out of credits, which then pile up at inputs in the form of grants. Implicit rate regulation does not work properly with inputs holding a lot of grants, because inputs are then free to inject segments for the same output close in time! Instead, implicit rate regulation assumes that inputs use their grants immediately. This is what speedup tries to enforce, as described in the next section[7].

Speeded-up fabric: So far we have assumed that all links in the switching fabric have the same rate as network-external links, L. With fabric links running faster, namely at rate L_F, output arbiters may grant segments at any rate $L \leq L_G \leq L_F$, see Fig. 5. A good choice in this trade-off is to use a rate $1 < L_G = L_F - \epsilon$. In this way, outputs use a portion of the internal speedup to bring data to the output faster, and clear backlogs, while the fabric still runs faster than the output arbiters. With such conservative admissions, the load-to-capacity ratio, and thus the short-term contention, at inputs and internal links, get reduced, which helps granted segments arrive at the designated time. If occasionally some inputs overshoot some fabric buffer, operating at full speedup rate can remove this backlog. Although it is still unclear whether (or which) speedup can avoid regenerating congestion trees, our experiments indicate that already a small speedup may suffice.

[7]Note that similar grant accumulations may occur even if we reserve space in fabric-output buffers, as observed in [13]. But in those systems, they are normally transient states [24], whereas in implicit rate regulation they cannot be resolved easily because they may also induce congestion trees.



(a) No speedup; backlogs at fabric-output buffers in front of hotspots outputs (first 10000 simulation time units).

(b) No speedup; grants/credits accumulated at each input (first 10000 simulation time units).

(c) No speedup; grants/credits accumulated at each input (first 10^5 simulation time units).

(d) 1.2 speedup; grants/credits accumulated at each input (first 30000 simulation time units).

Figure 4: Situation in which all the outputs of a last-stage switch are hotspots, each one with 2.5x more load than it can handle. Other outputs receive uniform highly bursty traffic at the peak possible rate. The maximum grant size is 128 segments. Initially some inputs hold more than 128 credits (b), obviously from more than two outputs, causing out-of-schedule injections and the first congestion trees. (a) The first fabric-output buffers reach the maximum occupancy (4000 Bytes) at about the same time. This eventually causes more out-of-schedule injections, and more congestion trees. (c) How grants accumulate at inputs in time. (d) With a speedup of 1.2×, credit accumulations are avoided.

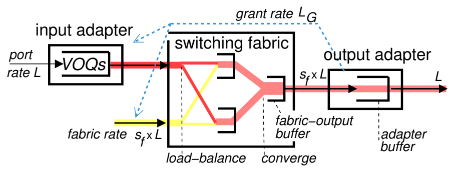


Figure 5: Rate of output grants in a fabric with internal speedup.

.

In Fig. 6, we examine implicit rate regulation when ACKs, grants, and requests are routed out-of-band, thus incurring no bandwidth overhead of their own. We plot the delay of packets going to non-hotspot outputs as a function of the load that these non-hotspots receive. We have three plots, for uniform traffic (no hotspots), 1 hotspot, and 8 hotspots, all reachable through the same last-stage switch. As can be seen, implicit rate regulation performs well when one hotspot is present, but behaves poorly (both in terms of delay and throughput) when 8 hotspots are present. In fact, when no speedup is used, we see reduced throughput even for uniform traffic. But with a speedup $L_F = 1.2{\times}L$ $({\times}L_G = L)$, implicit rate regulation performs well, delivering full throughput and very good delays even with 8 hotspots.

2.4 Reserving fabric-buffers

We can avoid congestion trees rooted at fabric outputs without relying on speedup, if output arbiters, in addition to what they do up to now, also reserve space in fabric-output-buffers for every segment that they grant. In this way, saturation trees cannot build up at fabric outputs. When used with coarse-grants, a drawback of this method is that each fabric-output buffer needs to be dimensioned so as to fit a few maximum-size grants to be able compensate for scheduling inefficiencies[8]. The benefit is increased certainty, possibly at smaller internal speedups. Actually, owing to per-flow load-balancing, we can state an even stronger guarantee. For an non-blocking Clos/Benes network, with *arbitrary number of stages*, the following theorem holds:

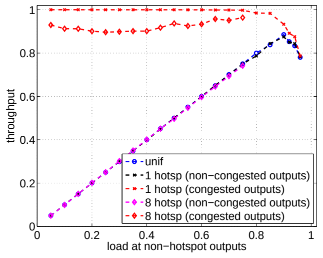
Theorem 1. *Assume a fluid model in which (i) there is a buffer in front of every switch output that can hold an amount of traffic equal to b, measured in arbitrary units; (ii) every switch output buffer has a full write bandwidth to accept the traffic from any number of inputs in the same switch; (iii) there is no flow control between adjacent stages; (iv) each output arbiter ensures that no more than b grants are outstanding; and (v) the traffic admitted by output arbiters is distributed evenly on a per-flow basis over all possible routes. Then, the output buffers in switching elements will never overflow.*

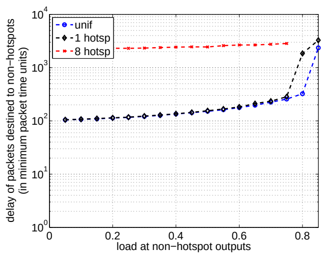
We skip the full proof here, and only present a descriptive outline of it below. Theorem 1 assumes that departures from buffers are never backpressured, and shows that none of them ever overflows. This implies that if buffers are flow controlled, as is the case for a real system, then the following corollary applies:

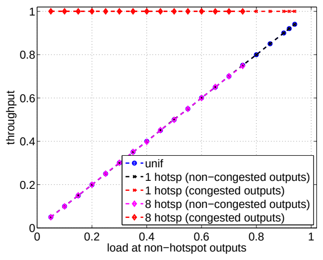
Corollary 1. *In a fluid system as described in Theorem 1, which implements flow control to prevent buffer overflow, flow control will never be activated.*

We now present an intuitive proof of Theorem 1 for the five-stage Clos network depicted in Fig. 7. Each 2×2 switching element has an output buffer for b segments. Consider a time interval of T segment times, which starts with all buffers being empty. Now consider buffer A in front of fabric-output 5. The corresponding output arbiter will grant at most $T + b$ segments in this interval, whereas buffer A can drain T segments, because no backpressure is exerted from the output adapter; therefore buffer A will never overflow. Next, consider the switch-output buffers B1 and B2 of the 4th stage that steer traffic to buffer A. As the load from each

[8]When buffers fit just one or two such maximum-size grants, input-output pairings need be exact; otherwise throughput will be lost.


(a) No speedup: throughputs.


(b) No speedup: delays of packets to non-hotspots.


(c) Speedup 1.2: throughputs.

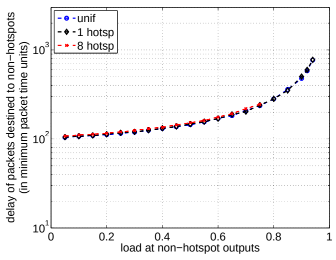

(d) Speedup 1.2: delays of packets to non-hotspots.

Figure 6: Implicit rate regulation, with each grant referring to up to 128 segments ($\max_s = \min_p =$ 40 Bytes). Eight (8) hotspots, each oversubscribed by a factor of 2.5×. Bursty (36) $\min_p$ packet arrivals; 64×64, three-stage Clos network, made of 8×8 switches. Each fabric buffer can fit 4 KBytes. For additional information on the simulation parameters, please refer to Sec. 4.

flow is split evenly, the load towards each fabric-output will also be split equally across all available paths. Therefore, each of the buffers B1 and B2 may be targeted by half of the segments that go to A, i.e., up to $\frac{T+b}{2}$ segments each. Of course, these buffers may also need to handle an equal number of segments that go to output 6. In total, the number of granted segments that target buffer B1 (or B2) in interval T never exceeds $T + b$; thus also buffer B1 will never overflow. Going one stage upstream, each of the buffers C1 and C2 will carry an equal portion of the segments that go to B1 thanks to load balancing, and an equal portion of the segments heading to the other output in the same switch with B1 again thanks to load balancing. Thus, each buffer will be targeted by at most $b + T$ segments, and never overflow. We finished with the buffers in the routing part of the network.

For the buffers in the distribution network of Fig. 7, we start from the first stage. Buffer E1 carries a portion of the load from two fabric-inputs. In interval T, each of these inputs may inject at most T segments. As the load from each flow is split evenly across all available paths, the load from each of these inputs will also be split equally to buffers E1 and E2. Thus, in interval T, buffer E1 will need to handle at most T segments and never overflow. Going one stage upstream, buffer D1 will carry half of the load from buffers E1 and E3; the other half will be distributed on buffer D2. Thus, buffer D1 will also need to handle at most T segments and thus never overflow.

The actual proof for Theorem 1 is based on a fluid traffic model in which the load from each flow is perfectly dis-

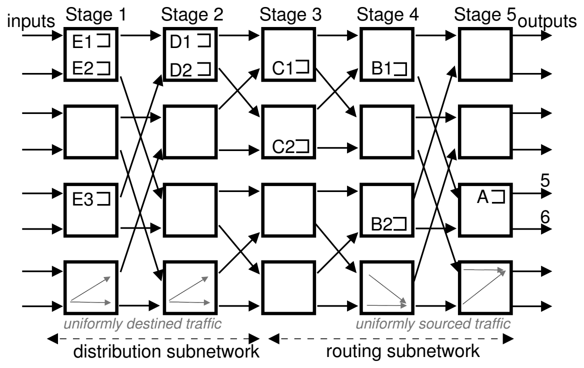
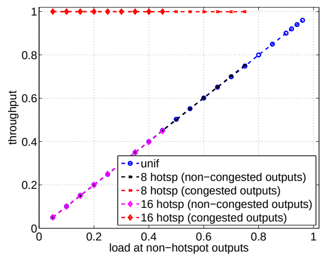


Figure 7: An 8×8, five-stage Clos network, made of 2×2 switching elements with output buffers. The bottom switch in each stage shows traffic being uniformly split in the distibution network and then uniformly merged in the routing network.

tributed among all available paths. For the real system, we guarantee that output adapters never exert backpressure on fabric-outputs, and that all injected segments will 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-segment basis may not be perfect in the short term. Our results show that these buffers will exert backpressure only sporadically, owing to random short term conflicts rather than persisting contention.

Finally note that by reserving space for fabric-output buffers, we can increase the rate of admissions, L_G, even above the rate of the fabric, L_F because the congestion control no longer relies on output schedules.


(a) No speedup: throughput.

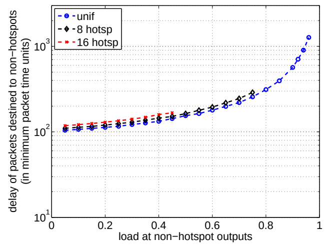

(b) No speedup: delay of packets to non-hotspots.

Figure 8: Reserve fabric-buffers, with each grant referring to up to 128 segments ($\max_s = \min_p = 40$ Bytes); 8 hotspots, each oversubscribed by a factor of 2.5×. Bursty (36) $\min_p$ packet arrivals; 64×64, three-stage Clos network, made of 8×8 switches. Each fabric buffer can fit 32 KBytes. For additional information on the simulation parameters, please refer to Sec. 4.

In Fig. 8, we repeat the same experiment as in Fig. 6, but with request and grants sent out-of-band. Because the 4 KByte fabric-output buffers do not even suffice for one maximum-size grant, we now dimension fabric-output buffers to 36 KBytes. (Similar results are obtained if we decrease the maximum grant size accordingly.) As shown in the figure, by reserving fabric-buffer space, we get excellent delays and throughputs for any number of hotspots, without requiring any speedup.

3. SYSTEM DESCRIPTION

In this section, we describe the system operation as modeled in our simulations. We start our description with the switching elements that we use and then describe the mechanisms at the input and output adapters.

3.1 Switching Elements

We assume switching elements (switches) that (a) only route segments from their arriving (local) input to the (local) output specified in the segment headers; and (b) implement simple hop-by-hop flow control, enabling lossless operation when switches are connected with each other. In particular, we use the combined-input-output-queueing switches shown in Fig. 9. Each of them has M input/output ports, with one buffers queues each. The sizes of the input and the output queues are denoted b_{in} and b, respectively, and are measured in segments. Observe that there is just one first-in-first-out (FIFO) queue at each input. The internal crossbar, which reads segments from input and writes them to output buffers, operates at a speed $L_S = 2 \cdot L_F$.

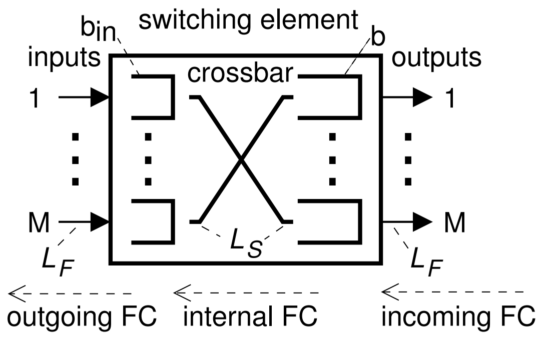


Figure 9: Switching elements.

In [12] [13], we considered buffered crossbar switches. We have also examined them in the present architecture. Our results suggest that buffered crossbars, although they have full internal speedup, implement more elaborate local flow control (similar to having virtual queues per local output at the inputs) and are more costly in terms of buffers, behave similarly to input-output queued switches when employed in a non-blocking multi-stage network.

3.2 Request message generation at adapters

Figure 10 depicts the organization of a network adapter. When a new network packet is received by the input adapter (ingress side), classification and routing are performed first, then the packet is stored in the VOQ for the targeted output. There are N VOQs at each input adapter. In addition, the adapter maintains a state of counter arrays, with N entries each, i.e. one per destination: *received grant* counters, g, maintain the number of received grants per output; *pending request* counters, pr, maintain the number of pending requests per output; *pending unsolicited segments* counters, ps, maintain the number of unsolicited and not

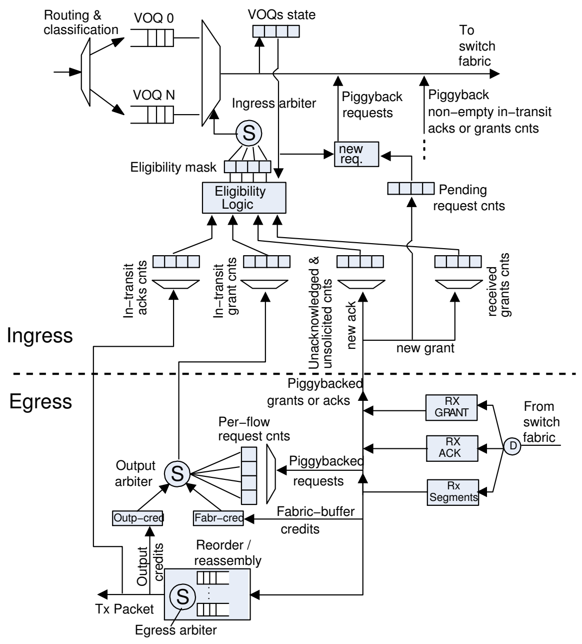


Figure 10: Organization of network adapters.

yet acknowledged segments per output[9]; and *transit grant* counters, tg, and *transit ACKs* counters, ta, with per-output entries maintain the number of grants and ACKs, respectively, that the egress (output) side of the present adapter has generated. Note that the input adapter has to convey the values of the transit counters to their final destination (adapter). It can do that either by piggybacking or by sending a standalone grant/ACK message of size s_{st} (20 Bytes in our simulations).

On every new segment that it injects, the input can piggyback requests for new segments in the corresponding VOQ in a dedicated request field of the segment's header. The requested size from each VOQ relates to reassembly buffer management and is described in Sec. 3.4. For the moment, note that a VOQ cannot have more than G requests pending, where $G \geq \lceil \frac{\max_p}{\max_s} \rceil$ (the maximum-packet size in segments).

As mentioned in Sec. 2.1, to start the communication with some output, e.g. when a packet arrives at an empty VOQ, the input adapter is allowed to inject an unsolicited segment, which will convey requests for the additional segments in the packet (if any) or in the VOQ. After injecting the segment, the adapter increases the corresponding ps counter. This counter is decremented when the input receives an ACK message confirming that the corresponding segment releases the buffer space it occupies in the output adapter, i.e., after the packet that it belongs to has been reassembled and starts being forwarded onto the output line.

The eligibility logic computes whether there is something to send to each output and stores the result in an N-bit wide eligibility bitmask. In summary: 1) VOQ for output i is eligible when $g[i] > 0$, in which case the HOL segment from the VOQ can be injected. Otherwise, 2) when $g[i] == 0$, a non-empty VOQ can inject its HOL segment if $r[i] == 0$ and

[9]Note that unsolicited segments are never dropped because buffer space has been preallocated for them in output adapters.

$ps[i] == 0$; this segment will be unsolicited. When the VOQ for output i is empty or not granted and not eligible for unsolicited injection, the input may inject a special ACK/grant segment to output i if ($tg[i] == 1$ or $ta[i] == 1$). In all other cases, output i is ineligible for service. According to these rules, as long as the input can inject a (granted or unsolicited) VOQ segment, it will not inject special ACK/grant segments towards the same destination: VOQ segments go first, because they can convey the current values of the corresponding transit ACK/grant counters. All injections from the adapter are subject to hop-by-hop backpressure; the adapter holds a credit counter which maintains the available segment slots in the input queue of the downstream switch, and resets all eligibility flags when no such local credits are present.

A round-robin scheduler scans the eligibility mask, starting from a "next-to-serve" position, and selects the first nonzero entry. A (VOQ or special) segment is then fetched for the destination selected and injected into the switching fabric. The route of the segment is pointed by a per-flow distribution counter. The adapter increases the distribution counter by one, so that the next segment of the selected flow will be routed through the next available path. The header of the injected segment contains a route field, a request-size field, a grant-size field, an ACK-flag, and an unsolicited flag.

3.3 Grant message generation

As shown in Fig. 10, every output adapter maintains the pending segment requests from each input, i, using a counter $r[i]$. The number of requests in a counter corresponds to the number of non yet granted segments in the corresponding VOQ at the input adapter. Remember that each flow may have at most G pending requests, thus $r[i] \leq G$.

In addition, the output adapter maintains two credit counters: the output-buffer credit counter , oc, initialized at B, which monitors how many segment slots are available in the reorder/reassembly buffer in the adapter, and the fabric-buffer credit counter, fc, which is initialized at b.

To avoid deadlocks when the reassembly buffers is smaller than $N \cdot \max_p$, the output arbiter either grants all segments of a VOQ packet or none of them. To establish this condition, we require that the output grant input i only if $oc \geq r[i]$ and $fc \geq r[i]$.

Fairness: However, with this allocation strategy, fairness can be violated: inputs presenting a few requests to some output may starve other flows that have many requests when either of the credit counters has dropped to a small value. To restore fairness, the output arbiter issues a new grant only as long as $oc \geq G$ and $fc \geq G$, thus guaranteeing that either all or none of the inputs are eligible. Furthermore, to maintain fair (round-robin on a per-byte level) allocation of output bandwidth, the scheduling discipline used to select among the eligible inputs at the output has to be aware of the actual sizes ($r[i]$) it serves each time. Candidate disciplines are weighted or deficit round-robin. In our simulations, we use a hardware-efficient version of WRR, which is based on virtual intervals [?].

After selecting an input, the arbiter decrements oc and fc by gs, and issues a grant that is equal to the allocated space gs. The output adapter then stays idle for $\frac{gs}{L_G}$, where L_G is the peak output grant rate. The grant message issued contains the input ID and is routed to the ingress path of the adapter, where it updates the corresponding transit grant counter. From there, the grant is eventually routed to the waiting adapter.

Grant messages (piggybacked or standalone) are load-balanced, and are subject to hop-by-hop backpressure, thus the per-flow grants can be delivered to the ingress linecards out-of-order. Out-of-order delivery is not an issue here, and the per-flow grant messages can safely be merged in the received grant counters g: we can use grants that were actually issued for the "next-in-line" packet in a VOQ to inject segments that belong to the HOL packet because we know that all segments of both packets have been granted by the output.

The fabric-buffer credit counter, fc, is incremented by one when a granted segment arrives at the output adapter. When a reassembled packet starts departing from the output adapter the output-buffer credit counter is incremented by the number of granted segments in that packet. If the packet contained any unsolicited segments, an ACK message is generated, and routed to the waiting input adapter, as is done for grant messages.

3.4 Fitting the reassembly buffer into on-chip memories

A traditionally managed reassembly buffer needs to have space for N maximum-size packets; for $N = 10$ K, and $\max_p = 9$ KBytes, the required buffer space is ≈ 720 Mbits. Now we discuss how the input and output adapter coordinate so as to prevent deadlock in a reassembly buffer that can fit only a *few*
pmax packets[10]. As mentioned, this can be guaranteed as long as the output either grants all the segments of a HOL (VOQ) packet or none of them. In this way, the input adapter may safely use any output grant, knowing that space has been reserved at the reorder/reassembly buffer for the entire packet that this segment belongs to.

To establish this condition, we require that each new request issued from a VOQ be for the entire payload of a number of next-in-line packets in that VOQ. If requesting grants for the next-in-line packet in some VOQ increases the number of pending requests beyond G, the input will not request any grant for that or additional packets in the same VOQ. The output arbiter, in turn, only grants the complete requested size from each input. It follows that when an input starts injecting segments from some packet, p, space has been reserved for the entire payload of p.

Dependence on the RTT: As mentioned above, $G \geq \lceil \frac{\max_p}{\max_s} \rceil$, and the reorder buffer space needs to have a size greater than G for the scheme to work without deadlocks. Consider that if $G = \lceil \frac{\max_p}{\max_s} \rceil$ and a train of back-to-back $\max_p$ and $\min_p$ packets arrives at some input, the output that all of these packets are targeting may have idling periods. We can circumvent this if $G = \lceil \frac{\max_p}{\max_s} \rceil$ *plus* an RTT's worth of segments.

4. SIMULATIONS

This section evaluates the proposed schemes using computer simulations. The simulation code models the operation of the system at byte-level granularity following the description in Sec. 3. We use a maximum and minimum segment sizes of $\max_s = 256$ B and $\min_s = \min_p = 40$ B, respectively. Standalone ACK/grant messages are 20B

[10]Output adapters additionally buffers for the unsolicited segments, i.e., for up to 1 segments from each input.

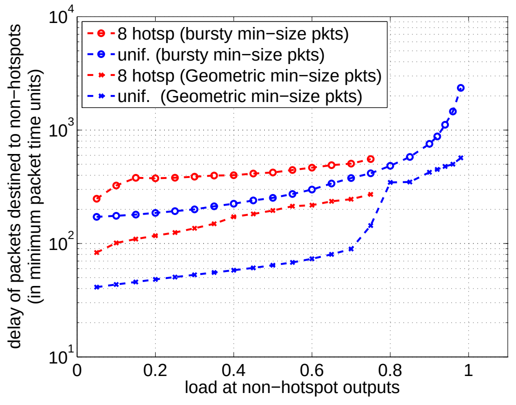


Figure 11: Reserve fabric-buffers, $L_F = 1.1 \times L$: delay of packets destined to non-hotspots.

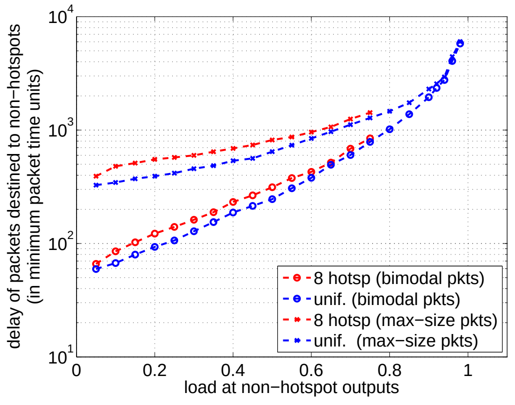


Figure 12: Reserve fabric-buffers, $L_F = 1.1 \times L$: delay of packets destined to non-hotspots.

long. Packet sizes range from 40 to 9000 B, i.e., up to the size of Ethernet Jumbo frames, on a byte granularity. Each adapter contains a reorder/reassembly packet memory of 4 Mbits, corresponding to 2048 $\max_s$. The input and output queue sizes of switching elements are 8 and 60 KB, corresponding to 240 and 32 $\max_s$ respectively.

We will report delays normalized with respect to the duration of minimum-sized packet, $t_{\min} = \min_p / L$. The end-to-end scheduling RTT and the switch-to-switch RTT are 55 and 13 $t_{\min}$, respectively. The packet delay is measured (in $t_{\min}$ units) from the time that the last bit of the packet arrived at the input adapter, until the first bit of the packet is forwarded on the external line. This delay includes one transfer time of the packet's payload (at L_F rate), incurred at the reassembly unit. The minumum delay (without contention) of a small ($\leq$ 256 B) unsolicited packet is 29 $t_{\min}$, plus its transfer time (i.e. < 7 $t_{\min}$). The delay of a similar request-grant packet is 83 $t_{\min}$. The delay of a 9 K Jumbo frame is 280 $t_{\min}$ or approximately equal to $\frac{3}{2}\times$ scheduling-RTT + $\frac{\max_p}{\min_p}$. The maximum number of pending requests or grants per flow is set to $G = 50$ segments. Thus each fabric-output buffer can fit about 5 maximum-size grants and the output adapter buffer about 40.

In experiments with hotspots, each hotspot fabric-output is oversubscribed by a factor of 2.5. All hotspots receive traffic uniformly from all inputs. Other outputs receive a varying load (again from all inputs). We present results for a 64×64 three-stage Clos network. In the reserve fabric-buffers method, output arbiters grant segments at a rate L_G, which is *ten* times higher than L; for implicit rate regulation, $L_G = L$, even when internal speedup is used.

In the first experiment, we use a speedup factor of 1.1× in the switching fabric to accommodate the request-grant overhead, and we examine the performance of the reserve fabric buffers method when all (8) outputs of a last-stage switch are oversubscribed. In Fig. 11, we present the delay of packets destined to non-hotspot outputs. Packets here have constant size of $\min_p$ and arrive in bursts of an average length of 36 $\min_p$ or are smoothed according to Poisson (smooth, geometric) arrivals. For comparison, we also present the performance for uniform traffic when no output is congested. Note that the packet delay for smooth, uniform traffic at low loads is just above the minimum possible of 29 $t_{\min}$: most packets avoid the request-grant latency. As

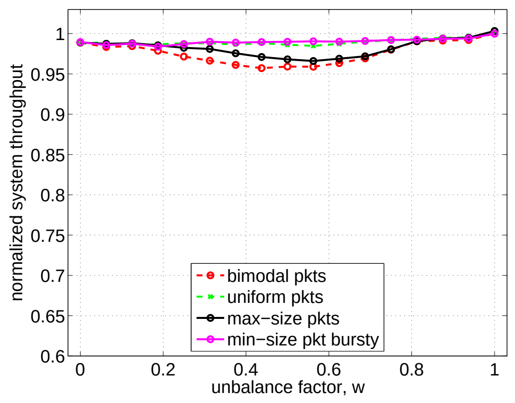
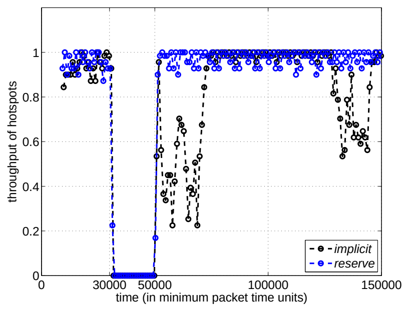


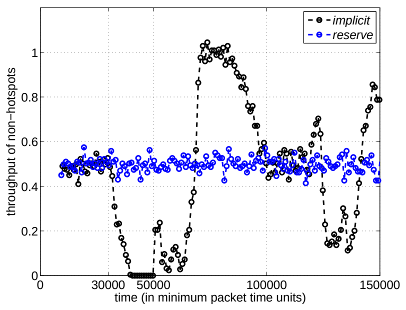
Figure 13: $N = 64$. Throughput under non-uniform traffic for reserve fabric-buffers, $L_F = 1.1 \times L$.

can be seen, in the presence of hotspots, this capability is lost (e.g., because of delays in ACKs), even when arrivals are smooth, and thats why we see a worse delay at low loads. For the bursty plots, the discrepancies are slightly larger and are due to increased delays in the request-grant paths. (Consider the ACKs and grants that are destined to the hotspot adapters, which in addition act as sources of non-hotspot traffic.) In Fig. 12, we repeat the same experiment for Bimodal (95% $\min_p$ and 5% $\max_p$), and $\max_p$ (Jumbo frames) packets. Here the discrepancies caused by the presence of hotspots are less discernible, because other delays dominate.

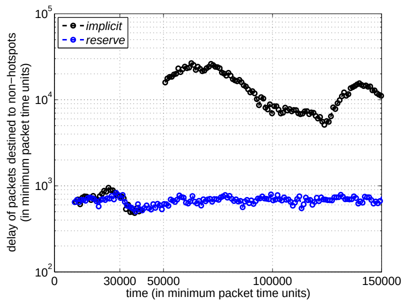
In Figures 13 and 14, we present the throughput performance for non-uniform traffic, for N= 64 and N= 256, respectively. In this traffic model, traffic ranges from uniform ($w = 0$) to one-to-one ($w = 1$); at intermediate w values, traffic is heavily unbalanced: each input (or output) has one "heavy" output (or input) communication pair, and many "light" ones. We have separate plots for different packet size distributions. The worst throughput (of about 90%) occurs for minimum-size packets, with smooth-geometric arrivals at intermediate w values. Normally smooth arrivals yield the best throughput results. But here, the grants of light flows are frequently smaller than G (and their segments in many cased unfilled), thus the grant bandwidth overhead increases.



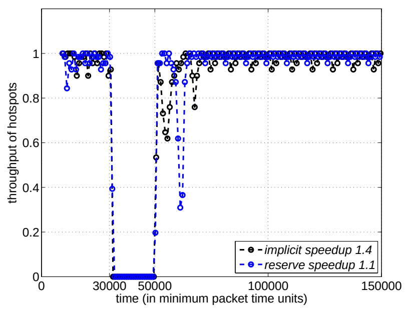
(a) Throughput of hotspots in time.



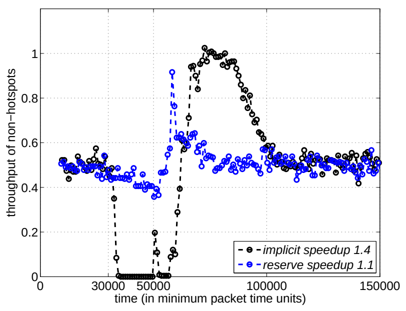
(b) Throughput of non-hotspots in time.



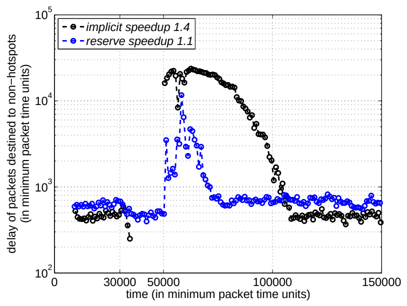
(c) Delay of packets to non-hotspots in time.



(d) Throughput of hotspots in time.



(e) Throughput of non-hotspots in time.



(f) Delay of packets to non-hotspots in time.

Figure 15: Worst-case scenario: The departures from the fabric-output buffers, upstream to hotspot outputs, are blocked in the time interval [30000, 50000]. Panels (a), (b), and (c) belong to a run in which the request and grants are sent out-of-band and no internal speedup is used. Panels (d), (e), and (f) belong to a run in which request and grants are sent in-band, and the internal speedup is 1.1× for reserve fabric-output buffers and 1.4× for implicit rate regulation.

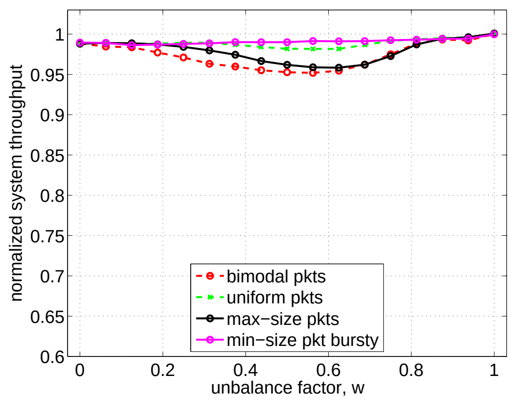


Figure 14: $N = 256$. Throughput under non-uniform traffic for reserve fabric-buffers, $L_F = 1.1{\times}L$.

We note here that, for the same parameters, implicit rate regulation works smoothly for a speedup between 1.2× and 1.4×; nevertheless, with a speedup of 1.1×, it yields worse results than reserve fabric-buffers do. We conclude that with larger buffers and some internal speedup, the regenerating saturation trees discussed in Sec. 2.3 are less frequent: speedup reduces instantaneous backlogs, whereas larger buffers do not fill up as rapidly. Nevertheless, no one can guarantee that the buffers will not fill up. (We have actually seen this happening for 32 KB fabric-output buffers under a heavy workload of max$_p$ packets.) When we reserve space in fabric-output buffers, these may only fill up because of control messages. (Note that 2K concurrent control messages are required in order to fill up a buffer of 60 KB.) Fabric-output buffers may also fill up when more than 200 inputs concurrently inject an unsolicited 256 B segment to the same output.

4.1 Host devices suddenly stop accepting data

To test how these system will behave under such unfavorable circumstances, we intentionally blocked the departures from fabric-output buffers in front of congested outputs (i.e., hotspots), for 10,000 time slots. In Fig. 15 we plot the delay and throughput behavior as a function of time in a scenario with eight hotspots and max$_p$ packets: nothing departs from fabric-output buffers in the simulation time interval [30000-50000]. In Fig. 15(a,b,c), ACK, grants, and requests are sent out-of-band (thus incurring no overhead), unsolicited injections are prohibited, and the systems have no speedup. As can be seen in Fig. 15(b,c), the reserve fabric-buffers method is almost transparent to the event, as the delay and throughput of non-hotspot flows are only marginally affected. For implicit rate regulation without speedup, on the other hand, the throughput first drops to close zero (while hotspots are blocked) and then takes a long time to recover, and its delay remains high for much longer[11].

[11]Note that in the plots for implicit rate regulation some sam-

In Fig. 15(d,e,f), control messages are sent in-band, as in a real system, and unsolicited injections are allowed. Implicit rate regulation is equipped with a speedup of 1.4×, and reserve fabric-buffers with a speedup of 1.1×. As can be seen, the delays in reserve fabric-buffers are not affected within the idling period, but increase just after its end. In the simulations, we have actually seen that, in contrast to implicit regulation, buffers do not fill up. What happens is that the hotspot adapters receive a burst of grant messages after the end of the idling period (especially from other hotspots) and coordinate to directed injections that have to go through the same first-stage switch: the increased delays we see occur in first-stage switches because of their limited internal bandwidth. Despite its larger speedup, implicit rate regulation yields significantly worse delays and throughput.

5. CONCLUSIONS AND FUTURE WORK

We proposed efficient and scalable request-grant congestion management for highly utilized, converged data-center networks. Although the overhead of request-grant messages is modest, and allocating special network resources (e.g. a special scheduling network) for these messages does not cost that much (compared to the benefits that come in return), here we examined an alternative approach: all congestion management functions are delegated to the edge of the network (adapters), without any need for support from within the network itself (switches). We showed how to minimize the request-grant bandwidth overhead and the size of the reassembly buffers at output adapters.

For this distributed setup, we analyzed two candidate congestion management schemes. The first one, implicit rate regulation, reserves time-slots. For this scheme we described pathological scenarios, and concluded that it works correctly when internal speedup is employed. The other scheme reserves slots in fabric-output buffers, instead of time, and was found to increase robustness, trading off additional onchip memories vs. reduced speedup. Despite their differnces, both schemes can be combined conveniently with reorder/reassembly buffer management, but also with end-to-end reliable-delivery schemes.

Reserving space in fabric-output buffers has some prominent benefits. As we have shown here, if a receiving host / device stops accepting new data from the network for some time, then a typical network will collapse. But by making sure that all packets that are destined for this idling output can "escape" into the corresponding fabric-output buffer, we avoid most unwanted interferences. This is critical for the data center environments, wherein a host device or virtual machine may suddenly refuse accepting more data. Reserving fabric-output buffers is also very beneficial for "incast congestion" circumstances encountered in storage networks [25].

For future work, we plan to examine how these schemes perform in network topologies that may exhibit internal blocking. Along another direction, we plan to perform an elaborate study where we will compare these proactive schemes with reactive congestion management techniques. In order to be complete, such a study should take into account the communication delay overhead under both low and high utilization, the delay in recovering congestion, the delay in recovering flow rates, as well as the control message overhead.

pling points are missing in the time interval [30000-50000]. This happens because no packet gets reassembled (because of extensive blocking, throughput has dropped to zero), and no samples are taken.

6. REFERENCES

[1] C. Minkenberg, R. P. Luijten, F. Abel, W. E. Denzel, and M. Gusat, "Current issues in packet switch design," *Computer Communication Review*, vol. 33, no. 1, pp. 119–124, 2003.

[2] P. Sindhu, P. Lacroute, M. Tucker, J. Weisbloom, and D. Winters, US Patent US 7,102,999 B2, Sept., 2006.

[3] "A day in the life of a fibre channel frame: Cisco mds 9000 family switch architecture," 2006.

[4] A. Bianco, P. Giaccone, E. M. Giraudo, F. Neri, and E. Schiattarella, "Performance analysis of storage area network switches," in *Proc. IEEE HPSR*, Hong Kong, May 2005.

[5] N. Chrysos, "Request-grant scheduling for congestion elimination in multi-stage networks," Ph.D. dissertation, Univ. of Crete, Greece, Dec. 2006.

[6] O. Iny, US Patent US 7,619,970 B2, Nov., 2009.

[7] C. Clos, "A study of non-blocking switching networks," *Bell System Tech. J.*, vol. 32, pp. 406–424, 1953.

[8] V. E. Benes, *Mathematical Theory of Connecting Networks and Telephone Traffic.* New York City, NY: Academic Press, 1965.

[9] F. Petrini and M. Vanneschi, "k-ary n-trees: High performance networks for massively parallel architectures," in *Proc. 11th Int'l Parallel Processing Symposium (IPPS '97)*, Geneva, Switzerland, Apr. 1997, pp. 87–93.

[10] G. Pfister and V. Norton, "Hot spot contention and combining in multistage interconnection networks," *IEEE Trans. Computers*, vol. C-34, no. 10, pp. 933–938, Oct. 1985.

[11] P. Pappu, J. Turner, and K. Wong, "Work-conserving distributed schedulers for terabit routers," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 257–268, 2004.

[12] N. Chrysos and M. Katevenis, "Scheduling in non-blocking buffered three-stage switching fabrics," in *Proc. IEEE INFOCOM.* Citeseer, 2006.

[13] N. Chrysos, "Congestion management for non-blocking clos networks," in *Proceedings of the ACM/IEEE Symposium on Architecture for Networking and Communications Systems (ANCS)*, Florida, Dec. 2007.

[14] Y. Tamir and G. Frazier, "High-performance multiqueue buffers for VLSI communication switches," in *Computer Architecture, 1988. Conference Proceedings. 15th Annual International Symposium on.* IEEE, 2002, pp. 343–354.

[15] M. Katevenis, "Fast switching and fair control of congested flow in broadband networks," *IEEE Journal of Selected Areas in Communication*, vol. SAC-5, no. 8, pp. 1315–1326, Oct. 1987.

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

[17] R. Pan, "QCN pseudo code version 2.2," Nov. 13 2008. [Online]. Available:

http://www.ieee802.org/1/files/public/docs2008/au-pan-QCN-pseudo-code-ver2-2.pdf

[18] “IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks - Amendment: 10: Congestion Notification,” http://www.ieee802.org/1/pages/802.1au.html, 802.1Qau.

[19] J. Duato, I. Johnson, J. Flich, F. Naven, P. J. García, and T. N. Frinós, “A new scalable and cost-effective congestion management strategy for lossless multistage interconnection networks,” in *HPCA*. IEEE Computer Society, 2005.

[20] H. Chao, J. Park, S. Artan, S. Jiang, and G. Zhang, “TrueWay: a highly scalable multi-plane multi-stage buffered packet switch,” in *High Performance Switching and Routing, 2005. HPSR. 2005 Workshop on.* IEEE, 2005, pp. 246–253.

[21] S. Iyer and N. W. McKeown, “Analysis of the parallel packet switch architecture,” *IEEE/ACM Transactions on Networking*, vol. 11, no. 2, pp. 314–324, Apr. 2003.

[22] M. Katevenis and G. Passas, “Variable-size multipacket segments in buffered crossbar (CICQ) architectures,” in *Communications, 2005. ICC 2005. 2005 IEEE International Conference on*, vol. 2. IEEE, 2005, pp. 999–1004.

[23] I. Iliadis and C. Minkenberg, “Performance of a speculative transmission scheme for arbitration latency reduction,” *IEEE/ACM Trans. Computers*, vol. 16, no. 1, pp. 182–195, Feb. 2008.

[24] N. Chrysos and M. Katevenis, “Preventing buffer-credit accumulations in switches with small, shared output queues,” in *High Performance Switching and Routing, 2006 Workshop on.* IEEE, 2006, p. 8.

[25] A. Phanishayee, E. Krevat, V. Vasudevan, D. G. Andersen, G. R. Ganger, G. A. Gibson, and S. Seshan, “Measurement and analysis of TCP throughput collapse in cluster-based storage systems,” in *FAST*, M. Baker and E. Riedel, Eds. USENIX, 2008, pp. 175–188.