2011 International Conference on Parallel Processing

Combining Congested-Flow Isolation and Injection Throttling in HPC Interconnection Networks

Jesus Escudero-Sahuquillo*, Ernst Gunnar Gran†, Pedro Javier Garcia*, Jose Flich‡,
Tor Skeie†, Olav Lysne†, Francisco Jose Quiles* and Jose Duato‡

*Dept. of Computing Systems, University of Castilla-La Mancha, Spain.
Email:{jesus.escudero, pedrojavier.garcia, francisco.quiles}@uclm.es
†Simula Research Laboratory, Norway. Email:{ernstgr, tskeie, olav.lysne}@simula.no
‡Dept. of Computer Engineering, Technical University of Valencia, Spain. Email:{jflich,jduato}@gap.upv.es

***Abstract*—Existing congestion control mechanisms in interconnects can be divided into two general approaches. One is to throttle traffic injection at the sources that contribute to congestion, and the other is to isolate the congested traffic in specially designated resources. These two approaches have different, but non-overlapping weaknesses. In this paper we present in detail a method that combines injection throttling and congested-flow isolation. Through simulation studies we first demonstrate the respective flaws of the injection throttling and of flow isolation. Thereafter we show that our combined method extracts the best of both approaches in the sense that it gives fast reaction to congestion, it is scalable and it has good fairness properties with respect to the congested flows.**

***Keywords*-Interconnection Networks; Congestion Management; HoL-blocking;**

I. Introduction

In interconnection networks [1], traffic congestion may degrade the network and overall system performance if no countermeasures are taken [2]–[4]. Congestion is simply a result of high load of traffic fed into a network link, exceeding the link capacity at that point. Hot spot traffic patterns, network burstiness, re-routing around faulty regions, and conducting link frequency/voltage scaling (lowering the link speed in order to save power), can all lead to congestion. If all these factors are known in advance, the network administrator may alleviate the consequences by effective load balancing of the traffic, but typically this is not the case. Furthermore, in cases where multiple nodes send more data to a single destination than the node can handle, no dynamic re-routing can be done to avoid network congestion. It becomes even more severe when a parallel computer is running multiple different jobs as an on-demand service (e.g. cloud computing), where the resulting traffic pattern becomes unpredictable.

Congestion control (CC) as a countermeasure for relieving the effects of congestion has been widely studied. In particular, this problem is well understood and solved by dropping packets in traditional lossy networks such as local area networks (LANs) and wide area networks (WANs). In these environments packet loss and high latency are indications of network congestion. Herein it is mainly TCP that implements end-to-end CC, either by a traditional window control mechanism [5] for detecting dropped packets or through changes in latency [6], [7]. Very often those networks are also over-provisioned in order to avoid congestion.

In high performance computing (HPC) data centers low latency is crucial, and packet dropping and retransmission are not allowed under regular circumstances due to loss of performance. Lossless behavior is achieved with credit based link-level flow control, which prevents a node or a switch from transmitting packets if the downstream node or switch lacks buffer space to receive them.

Typically, when congestion occurs in a lossless network, a congestion tree starts to build up due to the backpressure effect of the link-level flow control. The switch where the congestion starts will be the root of a congestion tree that grows towards the source nodes contributing to the congestion. This effect is known as congestion spreading. The tree grows because buffers fill up through the switches in the network as the switches run out of credits (not necessarily in the root). As the congestion tree grows, it introduces head-of-line (HoL) blocking [8] that slows down packet forwarding. HoL-blocking appears when the packet of the head of a queue is blocked and prevents packets behind it from advancing. This effect also affects flows which are not contributing to the congestion, severely degrading the entire network performance. The HoL- blocked flows become victims of congestion [8].

CC for link-level flow controlled networks cannot be based on a traditional window control mechanism as deployed in TCP, though it effectively limits the amount of buffer space that a flow can occupy in the network [9]. The reason for this is the relatively small bandwidth-delay product in this environment, where even a small window size may saturate the network [8]. A rate control (throttling) based CC mechanism is more appropriate for link-level flow controlled networks, since it increases the range of control compared to a window-based system. The mechanism relies on the switches to detect congestion, and inform the sources that contribute to the congestion they must reduce their corresponding injection rates. By reducing the injection rate, the sources remove the congestion tree and by that the HoL-blocking as well.

A problem associated with throttling-based CC is that it takes time from a switch detects congestion until the sources contributing to congestion are notified about congestion. During that time HoL-blocking degrades network performance. This slow reaction has inspired Duato et. al. to take a completely different approach for CC [10], [11]. Instead of removing the congestion tree itself, this approach strives to relieve the unfortunate side effects the congestion tree has on flows not contributing to the congestion. That is, the HoL-blocking is removed by using special set-aside-queues for contributors to congestion, effectively making it possible for

0190-3918/11 $26.00 © 2011 IEEE
DOI 10.1109/ICPP.2011.80

victim flows to bypass the congested flows without actually removing the congestion tree. Such an approach has the advantage of being able to react immediately and locally at each switch, at the cost of the extra buffers needed for the set aside queues and the added complexity in the switch to manage them. Isolating the congested flows in this way does, however, not address the real cause of the problem, sources injecting too much traffic into the network. The congestion trees themselves are left untouched. This poses a scalability challenge as the number of congestion trees could exceed the resources available for the set aside queues in the switches. If this happens, congestion trees will grow inside the queues supposed to be used only for victim flows, and by this reintroduce HoL-blocking and lead to performance degradation in the network.

Summing up, an injection throttling mechanism is able to remove the congestion tree, and by that the introduced HoL-blocking, but the mechanism has a challenge when it comes to reaction time. It operates behind schedule. A mechanism based on congested-flow isolation, on the other hand, reacts immediately but faces scalability issues as the number of congestion trees increases. In addition, there is another less obvious difference between the two CC mechanisms. A throttling mechanism has been shown to have the potential of improving fairness in the network by solving the well known parking lot problem [12]. This problem arises when several flows are stored in a queue at a switch while another flow addressed to the same output port is the sole user of another different queue. The flows sharing the same queue are granted access to the requested output port with less frequency, than the flow being the sole user of the queue. The throttling mechanism solves the parking lot problem by decreasing the injection rate on a per flow basis at the sources. On the other hand, a congested-flow isolation mechanism can actually have a negative effect on fairness depending on the arbitration in the switches.

In this paper we present *Combined Congested-Flow Isolation and Throttling* (CCFIT), a novel mechanism which combines the ideas of congested-flow isolation with a throttling mechanism. Using simulations, we show that CCFIT is able to remove HoL-blocking immediately, assure scalability by removing the congestion trees, improve fairness in the network, and last but not least, CCFIT achieves an allover higher throughput than injection throttling and congested-flow isolation do as standalone concepts.

The remainder of the paper is organized as follows: In Section II we discuss previous work related to CC in interconnection networks, to place our proposed CCFIT mechanism into the proper context. Section III contains a thorough description of the CCFIT mechanism, while in Section IV we evaluate the mechanism using results from simulation studies. In Section V we conclude the paper.

II. Related Work

Congestion control (CC) based on injection throttling is a popular approach to congestion handling. As aforementioned, the basic idea is to detect congestion in the network at the switches, then to notify the contributing sources about the congestion, and finally for the contributors to stop or cease traffic injection. This closed-loop feedback control philosophy is the basic approach of several proposals which, on the other hand, differ in several aspects. For instance, notifications could be sent to all the sources [13] or just to the sources contributing to congestion [14]. Other proposed mechanisms [15] notify congestion just to the local endpoints attached to the switch where congestion is detected. Furthermore, the switches can mark the packets contributing to congestion in order to notify the destinations about the situation, which subsequently notify the sources (the forward explicit notification approach), or the switches can themselves generate notification packets that are sent directly to the source nodes (the backward explicit notification approach). The InfiniBand (IB) network [16] applies the former approach, while the emerging Data Center Bridging standard [17] is implementing the latter.

There is also a body of work that propose different strategies for congestion notification and marking, e.g. a congested packet can be marked both in the input and output switch buffer, as well as being tagged with information about the severity of the congestion. Moreover, there are some different approaches for designing sources response function, i.e. the actions taken to reduce the injection rate, later followed by an increase in the rate when congestion is resolved [9], [18]–[20].

The injection throttling part of the CCFIT mechanism is inspired by the injection throttling mechanism specified for InfiniBand (IB), one of the most successful interconnect technologies. The IB Architecture Specification [16] defines two bits in the packet header for congestion notification. Specifically, if a packet is considered to contribute to congestion at a switch port, the *Forward Explicit Congestion Notification* (FECN) bit in the packet header is set. The FECN bit is then carried through the network to the destination node by the packet contributing to congestion. Upon reception of a "FECN-marked" packet, a destination will return back to the source a packet whose header will have the *Backward Explicit Congestion Notification* (BECN) bit set. Any source receiving a BECN packet will reduce its injection rate of the corresponding congested traffic flow, thus alleviating congestion.

The performance of IB CC depends on several configurable parameters. For instance, a *threshold* parameter mapped to a buffer fill ratio at a switch port[1] determines when the port is considered to be congested. If the buffer fill ratio is above the *threshold* the corresponding switch port is moved into the *Congestion State*, given that the port is also considered to be at the root of the congestion tree, that is, the port has available credits to forward packets. However, in order not to generate too many BECNs, not all the packets crossing a port in the Congestion State are "FECN-marked": Only those whose size is greater than the value of the *Packet_Size* parameter, and among them again, only a fraction corresponding to the *Marking_Rate*

[1]For simplicity, the concept of *Virtual Lanes* (VL) has been left out of this explanation of the IB CC.

parameter, are finally marked.

Similarly, the exact reaction of a source node upon the reception of a BECN also depends on a set of CC parameters. Specifically, the injection rate of a congested flow is reduced by introducing a *injection rate delay* (IRD) between consecutive packets of that flow. Source nodes store a list of possible IRD values in a *Congestion Control Table* (CCT), each congested flow holding an index (CCTI) into this table. CCT values are typically arranged in such a way that the higher the index, the greater the IRD. Upon reception of a BECN the index of a flow is increased by a value stated by the *CCTI_Increase* parameter. The CCTI is decremented again by one when a timer (whose value is stated by the *CCTI_Timer* parameter) expires. In this way, flows contributing to congestion will be throttled while congestion is present, and being released when congestion vanishes. More details about the IB CC mechanism can be found in [8].

While the IB CC mechanism, like injection throttling techniques in general, has the potential to remove the congestion tree and even improve fairness [8], the mechanism has a major drawback. The delay between congestion detection and reaction at the sources results in a CC mechanism operating behind schedule, where oscillating sources are adjusting their injection rates based on "old" information.

An alternative approach to the injection throttling mechanism is to remove the HoL-blocking without removing the congestion tree, e.g. using a congested-flow isolation technique. If HoL-blocking produced by congested flows to non-congested ones is eliminated, congestion turns harmless [4].

Many techniques have been proposed to reduce or eliminate HoL-blocking, most of them relying on having different queues at each switch port, in order to separately store packets belonging to different flows. For instance, a well-known HoL-blocking elimination technique is *Virtual Output Queues* (VOQs), either at switch level (VOQsw) [21] or at network level (VOQnet) [22]. The latter requires at each port as many queues as destinations in the network. Then, at each port, all the packets addressed to a specific destination are exclusively stored in the queue assigned to that destination, and they never share that queue with packets addressed to other destinations, thus completely removing HoL-blocking. Note, however, that VOQnet does not scale with network size. VOQsw uses as many queues at each port as output ports in the switch, so that each incoming packet is stored in the queue assigned to its output port. VOQsw scales with network size, and eliminates HoL-blocking in a switch if it is directly caused by packets contending for output ports in the same switch. Unfortunately, in switches affected by congestion spreading from other switches, VOQsw can not guarantee that congested packets do not share queues with non-congested ones, thus VOQsw just partially eliminates HoL-blocking. Other similar techniques that also reduce HoL-blocking, but do not completely eliminate it, are *Dynamically Allocated Multi-queues* (DAMQs) [23], *Destination-Based Buffer Management* (DBBM) [24], *Dynamic Switch Buffer Management* (DSBM) [25] and *Output-Based Queue-Assignment* (OBQA) [26].

All the mentioned HoL-blocking elimination techniques do not explicitly identify congested flows, but they rely on separating packets from different flows as much as possible with the available queues at each port. Their effectiveness then greatly depend on the number of queues per port. By contrast, other techniques explicitly detect and keep track of congested flows in order to isolate them in special, dynamically-assigned queues, while the non-congested flows may share queues without suffering significant HoL-blocking. In this way, the number of queues required to efficiently eliminate HoL-blocking is reduced. This is the main strategy followed by *Regional Explicit Congestion Notification* (RECN) [10], [27], *Regional Explicit Congestion Notification-Input Queued* (RECN-IQ) [28] and *Flow-Based Implicit Congestion Notification* (FBICM) [11].

In order to identify congested flows, these techniques implement some mechanism to locate congested points (i.e. to detect congestion), then identifying congested packets as those whose route crosses a congested point. In general, like the IB mechanism, these techniques detect congestion by locally monitoring queue occupancy at each switch port, and comparing it with a *Detection Threshold*. Once congestion is detected in a port, a special queue is immediately allocated to store congested packets crossing that port. Additionally, these techniques also require a set of queues, at each port, devoted to store congested packets[2], and some control memory to manage them, mainly to store the location of the congested point each special queue is assigned to. This control memory is implemented by means of a Content-Addressable Memory (CAM) present at each port. In the case of RECN and RECN-IQ, designed for source-based routing networks, each CAM line stores (among other information) the explicit path towards the root of the congestion tree assigned to a specific SAQ, while in the case of FBICM, designed for deterministic distributed-based routing, stores a set of destinations. If the occupancy of any SAQ (or CFQ) in a port reaches a specific threshold, the congestion information stored in its corresponding CAM line is propagated (by control packets) to the switch connected to that port, which in turn will allocate a new SAQ (or CFQ) for this specific congestion tree. In this way, special queues are allocated all along the way of congested flows to separate them from non-congested ones, thereby eliminating HoL-blocking. SAQs (or CFQs) are dynamically deallocated when the corresponding congestion tree vanishes, and later reallocated if a new congestion tree appears.

As mentioned in the introduction, although these solutions are quite effective, they also present some flaws, probably the most important one being the limited number of special queues per port, which may not be enough to handle all the possible congestion trees simultaneously present at a port. Note, however, that it is unlikely that many congestion trees are present in many ports at the same time, thus in most cases only a small fraction of ports would run out of SAQs. Nevertheless, any congestion tree may partially spoil

[2]These queues are called Set-Aside-Queues, SAQs, in RECN and RECN-IQ, and Congested-Flow-Queues, CFQs, in FBICM.

network performance if not suitably managed in a port.

To conclude, note that two of the most popular strategies for CC in high-performance interconnection networks, injection throttling and HoL-blocking elimination based on congested-flow isolation in dynamically-allocated queues, present different drawbacks. The initial idea behind CCFIT was that a combination of both approaches would alleviate the respective flaws. On one hand, congested-flow isolation eliminates HoL-blocking even if sources are not yet aware of the appearance of congestion. On the other hand, the throttling of congested flows would reduce the probability of having many of them simultaneously present in any port, i.e. also reducing the probability of running out of special queues in any port. In the following sections we describe and evaluate CCFIT, our new proposal to combine these approaches to CC. In addition, we also show that CCFIT in certain situations greatly improves fairness, both compared to a CC mechanism based on congested-flow isolation alone, as well as compared to a network configuration running without CC.

III. CCFIT Description

In this section, we describe the CCFIT mechanism, detailing the switch and end-node architecture, as well as their specific operation. After that, we analyze the CCFIT parameters.

A. Switch Architecture

The injection throttling part of the CCFIT mechanism is heavily influenced by the CC throttling mechanism specified for IB, evaluated in [8], [12]. This throttling mechanism is then combined with FBICM to achieve congested-flow isolation in switches using distributed routing. That is, the CCFIT switches are responsible for both detecting congestion and notifying the contributing sources, as well as isolating congested-flow packets, eliminating the HoL-blocking: When detecting congestion, a CCFIT switch moves the corresponding output port into the congestion state, mark packets using this output port by setting the FECN bits, and allocates a CFQ for packets belonging to the corresponding congestion point.

Regarding switch architecture, CCFIT does not limit the number of switch ports. Specifically, it has been developed for Input Queued (IQ) switches, where memories are only present at input ports. The current and popular IQ-switches are simpler and cheaper than the CIOQ ones, offering high bandwidth and low latency if Virtual Output Queues (VOQ) are used [29].

Regarding switch routing logic, CCFIT has been designed for networks using distributed deterministic routing (InfiniBand being a prominent example), thus routing logic can be implemented using any distributed deterministic routing hardware solution, like tabled-based routing, look-ahead routing, etc. The unique routing information packets need to include in their header is the destination they are addressed to, instead of a explicit, complete route as source routing does. Actually, this routing information and the stored congestion information allow CCFIT to detect if a packet is congested or not.

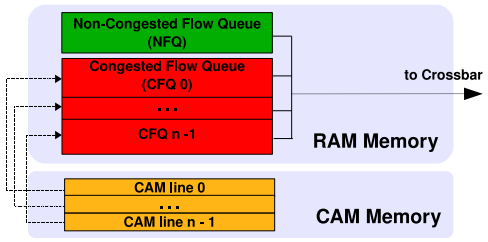


Figure 1. CCFIT Input Port Organization Diagram.

In order to deal with the HoL-blocking effect, CCFIT uses the input port organization shown in Fig. 1. Specifically, RAM is organized in queues, dynamically managed: CCFIT assumes two types of queues per input port: a normal flow queue (NFQ), where non-congested packets are stored, and a small number of congested flow queues (CFQs), where congested packets are isolated, thereby, not delaying the advance of non-congested ones. Moreover, CCFIT uses a post-processing mechanism (see Section III-C), similar to the FBICM one, in order to move congested packets from the NFQ to the CFQs. As we describe latter, the post-processing mechanism is in charge of moving an output port into the congestion state.

Like FBICM, CCFIT uses content addressable memories (CAMs) [30], in order to both keep track of the congestion information and store the CFQ status. Notice that each CAM line is associated with one CFQ. Although switch output ports have neither NFQs nor CFQs, CCFIT requires a CAM per output port, in order to propagate congestion information from a given input port CAMs to upstream input port CAMs. In that sense, CCFIT follows the same approach used in FBICM for congestion information propagation and resource deallocation.

Regarding switch scheduling, CCFIT uses iSlip [31], a Round-Robin (RR) algorithm achieving a fair arbitration inside the switch (as demonstrated in [12]). Specifically, all the switch input ports are served in a round robin fashion. An input port currently accessing an output port, will not get access to the same output port again until all other input ports requesting the same output port have been granted access. In particular, when a congestion tree arises inside the switch this scheduling policy allows a complete fairness between all the input ports which have allocated CFQs, even if several flows are sharing the same CFQ. Introducing injection throttling, however, may break the fairness property of RR if care is not taken. As shown in [12] it is important to use two thresholds ("high" and "low") for congestion detection to maintain fairness in RR-based switches. CCFIT follows this approach, though comparing the thresholds against the fill ratio of the CFQs rather than the VOQs. This is further described in Section III-C.

Although RR scheduling combined with the use of two thresholds allows fairness between input ports, it does not achieve fairness between traffic flows if some flows are exclusive users of their CFQs while other flows are sharing a CFQ (the parking lot problem). This problem is, however, solved by introducing the injection throttling [12]. All in

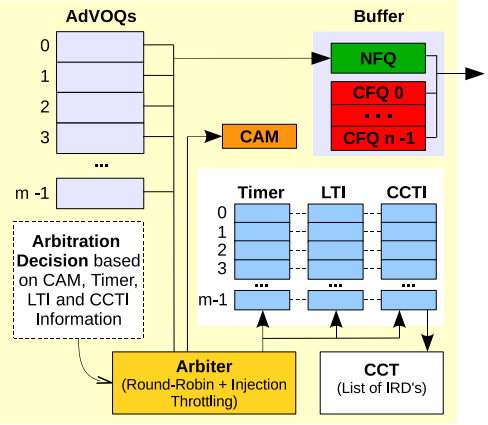


Figure 2. CCFIT Input Adapter Architecture.

all, CCFIT is able to achieve fairness at two levels: among different flows arriving at a hot spot switch from different input port CFQs, and by solving the parking lot problem. The fairness of CCFIT is further addressed in the evaluation Section IV-C.

Summing up, by using the architecture outlined above, the CCFIT switches are, as we describe in Section III-C, able to detect congestion, move output ports into the congestion state, marking packets contributing to congestion, isolating congested flows in the CFQs, and finally, release the required congestion information resources when congestion vanishes. The next section describes the end-nodes architecture.

B. End-nodes Architecture

An end-node receiving a packet with the FECN bit active should, as soon as possible, notify to the packet source about the congestion in order to throttle the injection. Similar to the IB CC mechanism, CCFIT returns a congestion notification packet (CNP) with the BECN bit active. The BECN packet has priority in the switches for being transmitted, and it only uses NFQs. Fig. 2 shows a diagram of the end-node architecture in charge of generating traffic, receiving BECNs and throttling the injection. For the sake of simplicity, we have omitted end-node structures in charge of receiving data packets and generating BECNs. In the following, we will refer this part of the end-node as the Input Adapter (IA).

Basically, CCFIT IAs have a fixed number of admittance queues (AdVOQs) equal to the network end-nodes, each AdVOQ$_i$ storing packets addressed to destination i, thus avoiding the HoL-blocking that may arise while generating traffic. Like the switch input ports, IAs have an output buffer organized in queues: one NFQ storing non-congested packets, and a small number of CFQs storing congested packets. Moreover, IA has a CAM with the same behavior as the ones located at switches. The CCFIT post-processing mechanism moves congested packets to the corresponding CFQ, so the HoL-blocking elimination is assured. Furthermore, CCFIT IAs include specific structures for the injection throttling, following an IB approach (see Section II). Specifically, the *Congestion Control Table* (CCT) stores a list of Injection Rate Delays (IRDs) which can be applied to any AdVOQ$_i$ in order to reduce its injection rate. The *CCT indexes array* (CCTI) stores a CCT index for each AdVOQ$_i$ and, by that, the IRD for a given AdVOQ$_i$ can be found at $CCT[CCTI[i]]$. Each CCTI index is increased when a BECN is received at the IA, thus increasing the value of the IRD applied to the AdVOQ$_i$. Notice that, during heavy congestion situations the IA will receive a lot of BECNs which will increase the CCTI index, and therefore the IRD value. In this way, the IA reduces the injection for the congested destinations.

For a given AdVOQ$_i$, the *Timer* array is used to decrease in one unit the CCTI[i] when the timer expires. In this way the IRD is reduced for that AdVOQ$_i$. A *Last Time of Injection* (LTI) array is in charge of storing the last time an AdVOQ$_i$ injected a packet in the network. This value is used by the arbitration together with the IRD in order to calculate if the next packet of the corresponding AdVOQ$_i$ could be sent or not. That is, the IA arbiter selects a packet from a specific AdVOQ$_i$ by applying a RR policy, making the "arbitration decision" based on the CAM, Timer, LTI and CCTI structures.

Finally, the most important effect of using CCFIT is achieved by the injection throttling since the CAM lines and CFQs, which were allocated when congestion appeared, are released quickly. As we show in the CCFIT evaluation (section IV) the network throughput is increased in comparison with the RECN-like and injection throttling techniques. In order to clarify the overall behavior of CCFIT at the IAs, an operation example is described in Section III-D.

C. Switch Behavior

Fig. 3 shows an example of CCFIT switches behavior. The switch stores an incoming packet (Event #1) in the corresponding NFQ. After that, CCFIT may detect congestion (Event #2) based on the NFQ occupancy level. When the NFQ fill ratio exceeds the congestion detection threshold, a congestion situation is detected and a CFQ is allocated in the input port, together with a CAM line containing the information related to the new congestion point.[3] Therefore, the CAM line information is used to detect if an incoming packet is addressed to the same destination. [4]

Like FBICM, CCFIT moves congested packets from the NFQ to the corresponding CFQ by means of the packet post-processing mechanism (Event #3). Basically, when a packet reaches the head of the NFQ, CCFIT looks up in the input port active CAM lines if the packet destination is stored in one of them. In the case of a match, the packet is moved to the CFQ the CAM line is referred to, otherwise, the packet crosses to the requested output port. Note, the post-processing mechanism leaves in the head of the NFQ only non-congested packets, thus avoiding the HoL-blocking problem between congested and non-congested packets. Moreover, this mechanism decides which input port queue (NFQ or CFQs) can request the output

[3]Notice that CCFIT, like FBICM, only requires to store in the CAM the destination the congested packet is addressed to.

[4]More information about FBICM CAMs can be found in [11], [32].

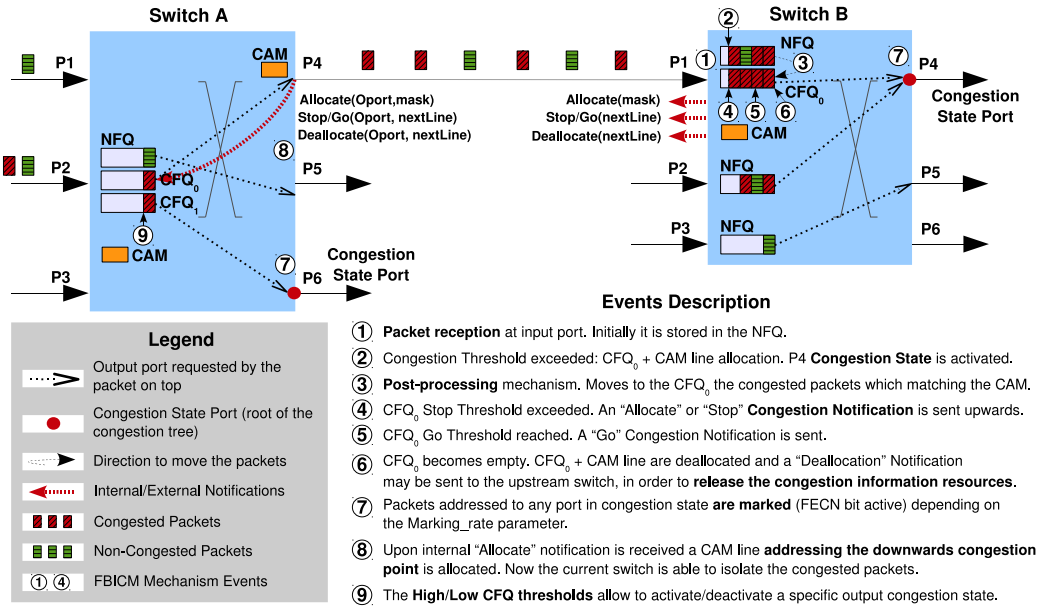


Figure 3. Example of the CCFIT Operation at Switches.

port. In that sense, it will create the crossing-requests that the arbiter will use for crossing packets to their requested output port. As we have described, the switch uses the iSlip scheduling algorithm.

CCFIT follows the FBICM scheme for propagating the congestion information. A CFQ Stop/Go flow control (Events #4 and #5) is used between every two switches containing allocated CFQs belonging to the same congestion tree. In this way, CCFIT separates congested and non-congested flows in different CFQs along any path followed by congested packets, thus isolating them and minimizing the HoL-blocking effect.

The dynamic and distributed resource deallocation process begins when a CFQ satisfies some conditions: it is empty and its associated CAM line is in Go status. When a CFQ is deallocated (Event #6), and it is part of one congestion tree branch (a previous "allocation" notification was sent to the upstream switch), a "deallocation" message is sent to the upstream switch to notify about the new situation. A similar process takes place between deallocated output port CAM lines and its linked input port CFQs+CAM lines.

As previously described, an important feature of the post-processing mechanism is that it is in charge of deciding if some output port should be moved into the congestion state. For each CFQ allocated in the root of the congestion tree (it is 1-hop away from the congested point), CCFIT looks at the CFQ occupancy level and, if that CFQ occupancy level exceeds the "High" threshold, the post-processing mechanism moves the output port, pointed to by the CFQ, into the congestion state. However, it may occur that the output port was already in the congestion state. When this happens CCFIT keeps track of the number of CFQs which have an occupancy level above the "High" threshold. Packets crossing an output port in the congestion state will be marked (Event #7) as congested (FECN bit set). If the CFQ occupancy level decreases below the "Low" threshold the output port counter is decreased, until it reaches the value 0, then moving the output port out of the congestion state. From this moment, no more packets are marked at the output port. Notice that, a CFQ which is placed 2-hops away from the congestion state (e.g. in Fig.3 CFQ_0 of P2 at Switch A) does not move its referred output into the congestion state, thus packets are not marked.

As it has been described in Section II, a FECN bit will be set or not depending on *Packet_Size* and *Marking_Rate* parameters. The influence of the *Packet_Size* and *Marking_Rate* parameters in the accuracy of the injection throttling mechanism is described in Section III-E.

D. Input Adapter Behavior

Fig. 4 shows an example of CCFIT IAs behavior. Basically, the IA is connected to switch A, which has just detected a congestion situation, which in turn, has been triggered by an incoming packet (Events #1 and #2). The post-processing mechanism moves all the congested packets to the CFQ (Event #3), and the CFQ flow control (Event #4) propagates the congestion information to the IA.

As the packets crossing through port P3 at Switch A are marked, the IA will receive BECN notifications indicating the injection throttling must start. When a BECN is received at the IA (Event #6), CCFIT obtains the $AdVOQ_i$ (i being the destination generating the BECN) which is sending packets to the destination this BECN belongs to. At this moment, CCFIT increases the CCTI[i] by one, thus increasing the IRD_i for the $AdVOQ_i$. Moreover, the Timer[i] (see Fig. 2) is initialized with the *CCTI_Timer* value. When the timer expires (Event #7) the CCTI[i] is decreased by one and the IRD_i is reduced, thus the IA increases the injection rate.

The IA arbiter makes the decision of which packet must be moved from the $AdVOQ_i$ to the NFQ (Event #8) based on a RR policy among all the AdVOQs. For each $AdVOQ_i$

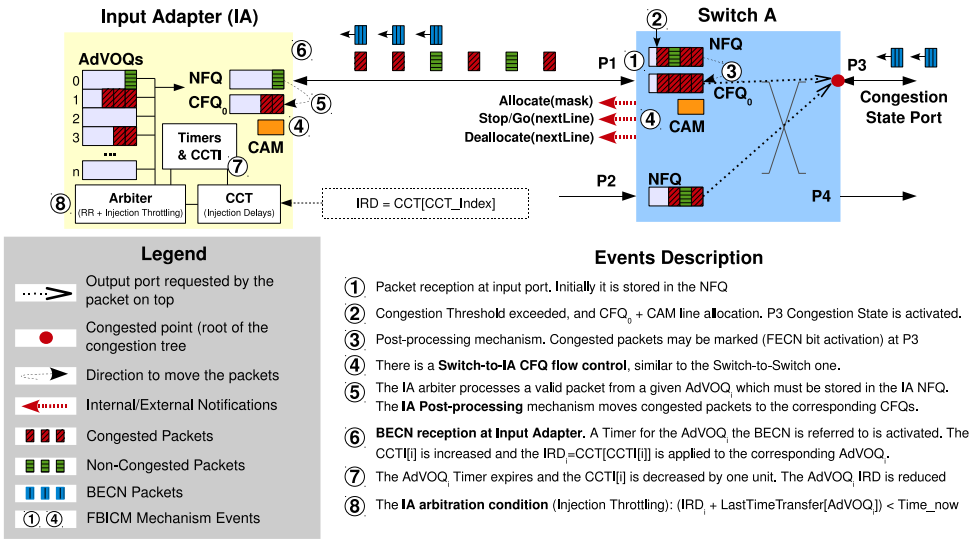


Figure 4. Example of the CCFIT operation at IAs.

the arbiter checks if the IRD_i value applied to that AdVOQ is greater than the current time, thus allowing the injection.

In this way CCFIT reduces the injection for congested destinations during congested situations, thus reducing the congestion trees and releasing the required resources for storing congestion information (mainly CFQs and CAMs) in a fast way. The injection rate is increased when congestion vanishes. As the evaluation shows (see Section IV), CCFIT significantly improves the overall network throughput achieved by RECN-like approaches, such as FBICM, and injection throttling ones.

E. Parameter Tuning Discussion

As it has been mentioned throughout the paper, CCFIT requires several parameters to be configured in the switches and IAs. These parameters are *Congestion detection threshold*, *CFQ stop/go thresholds*, *CFQ High/Low Thresholds*, *CCTI_Timer*, *Marking_Rate* and *Packet_Size*. We have made the same experiences as in [8] for the *CCTI_Timer*, *Marking_Rate* and *Packet_Size* parameters, thus a further description has been omitted. The remainder of the parameters need to be established taking into account the following ideas: The CFQ "High/Low" thresholds, as it is described in [12], should have a distance of at least one packet MTU. As one CFQ moves the corresponding output port into the congestion state when its occupancy level exceeds the "High" threshold, the "Stop" flow control threshold should be greater than the "High" one, in order to not block congested packets being transmitted from upward switch CFQs. On its side, the difference between "Stop" and "Go" thresholds needs to be sufficient for neither blocking too much upward congested flows, or allowing too much forwarding of congested traffic. Finally, the *detection threshold* value should allow to detect congestion not too early and not too late.

IV. Evaluation

In this Section CCFIT is evaluated in terms of network performance and fairness. It is important to note that the CCFIT main contribution is the significant good performance in comparison to FBICM, which is achieved especially in congestion situations where the latter has not a sufficient, available number of CFQs for storing congested packets. First of all, we describe the simulation tool, modeled traffic patterns and network configurations used in the experiments. Next, we analyze CCFIT performance results and fairness.

A. Simulation Model

The simulation tool used in our experiments is an event-driven simulator written in the programing language $C++$, which models interconnection networks at the cycle level, end-nodes and links. In our experiments, we model different network configurations which are shown in table I.

Table I
Evaluated Interconnection Network Configurations

	Config. #1	Config. #2	Config. #3
# Nodes	7	8	64
Topology	Ad-hoc (Fig. 5)	2-ary 3-tree (Fig. 6)	4-ary 3-tree
# Switches	2	12	48
Crossbar BW	5 GBytes/s	2.5 GBytes/s	
Switching	Virtual Cut-Through		
Scheduling	iSlip algorithm [31]		
Packet MTU	2048 Bytes		
Memory Size	64 KBytes		
Link Bandwidth	2.5, 5 GBytes/s	2.5 GBytes/s	
Flow Control	Credit-based		
Routing Algorithm	Deterministic	Deterministic (referred as DET [33])	
Routing Logic	Table-Based		

Config. #1 (Fig.5) and Config. #2 (Fig.6) are used to study throughput and fairness when several traffic flows create congestion. Config. #2 is furthermore used for studying congestion when uniform (random) traffic is generated. As congestion appears/disappears in a fast fashion, congested-flows need to be isolated immediately, while the fairness is maintained. A third configuration, Config. #3, is used for

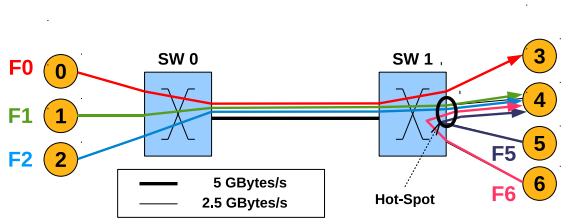


Figure 5. Configuration #1 Diagram.

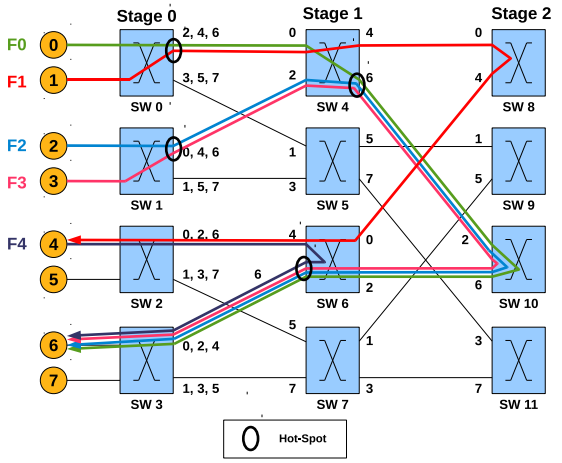


Figure 6. Configuration #2 Diagram.

testing CCFIT's reaction against heavy congestion situations, where several congestion trees overflows the number of available CFQs. In addition, the modeled traffic patterns for the above configurations are:

- **Case #1** (Config. #1). Five flows (F0, F1, F2, F5 and F6, see Fig. 5) are injected in the network. The injection rate is 100% of the link bandwidth (2.5 GBytes/s). Specifically, F0 (the victim flow) is active during the whole simulation period, while the other flows are activated in a sequential way. F1 is active during the time interval [$2ms$, $10ms$], F2 between [$4ms$, $10ms$], F5 between [$6ms$, $10ms$], and finally F5 is activated in the interval [$6ms$, $10ms$]. This traffic pattern generates a congestion point in the link connecting switch 1 with end-node 4.
- **Case #2** (Config. #2). Like the Case #1, five flows (F0-F4) are sequentially injected in the network at 100% of the link speed. In this case the flow F1 remains active during the whole simulation period. F0 is activated in the interval [$2ms$, $10ms$], F4 between [$4ms$, $10ms$], F2 between [$6ms$, $10ms$], and F3 in the interval [$6ms$, $10ms$]. This traffic pattern creates several congestion points in the network which divide the link bandwidth among all the flows contributing to congestion.
- **Case #3** (Config. #2). Here the situation is the same as for the Case #2, but three uniform traffic flows (sending packets to random destinations) are now active during the simulation (from nodes 5, 6 and 7). All the flows are injected at 100% of the link bandwidth. In particular this traffic pattern may add short lived congestion situations which quickly appear and disappear. Such congestion situations require a fast CC mechanism.
- **Case #4**. (Configuration #3) 75% of the sources inject uniform traffic at 100% rate of the link bandwidth. Suddenly, the remaining 25% of the sources generate congested traffic during the time interval [$1ms$,$2ms$]. These sources stop injecting traffic after this time period. This configuration tests if CCFIT copes with heavy congestion situations where more congestion trees than the number of CFQs are present. We have introduced 1, 4 and 6 congestion trees in the network during the congested traffic period of time.

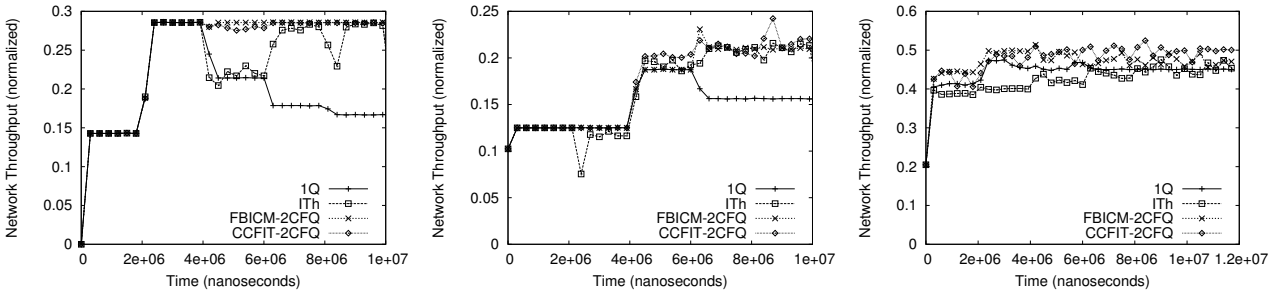
We have modeled an IQ-switch architecture, thus RAM memories are only added at each switch input port, with different sizes depending on the CC technique. Specifically, the simulator models the following CC techniques:

- Single Queue (1Q). This is the simplest case, with only one queue at each input port storing all the incoming packets. Hence, there is no HoL-blocking reduction policy at all. Note that this scheme is used for evaluating the performance of the DET routing algorithm "alone".
- FBICM. We use 2 CFQs per input port. Moreover, there are CAMs both at input and output ports.
- Injection Throttling (ITh). We have fixed the *CCTI_Timer* to 8000 ns, and the *Marking_Rate* to 85% of packets. We assume 8 Virtual Output Queues (VOQs) per input memory. High/Low thresholds are respectively set to 4 and 2 packets (2 MTUs as it is discussed in [12]).
- CCFIT. We assume 2 CFQs per input port. Like ITh technique, the same values for the *CCTI_Timer* and *Marking_Rate* are assumed. Only uses 2 CFQs per queue are defined for congested packets and entering output ports in the congestion state as a difference with ITh which has been configured with 8 VOQs. Moreover, the "Stop" threshold is established to 10 packets MTUs while the "Go" one is set to 4 MTUs.
- VOQnet. This scheme (theoretically the most effective one) requires greater memory sizes per input port, because each memory must be divided into as many queues as network end-nodes, and each queue requires a minimum size. Considering flow control restrictions, packet size, and link bandwidth and delay, we fix minimum queue size to 4 KB, which implies port memories of 256 KB for configuration #3 networks. This scheme is actually almost unfeasible, but it is used to show the theoretical maximum efficiency in HoL-blocking elimination.

Finally, although the simulator offers many metrics, we base our evaluation on two metrics: Flow Bandwidth, which shows the throughput achieved by each traffic flow, and network throughput, which shows network efficiency when normalized. In the following subsections we analyze, by means of these metrics, the obtained network performance.

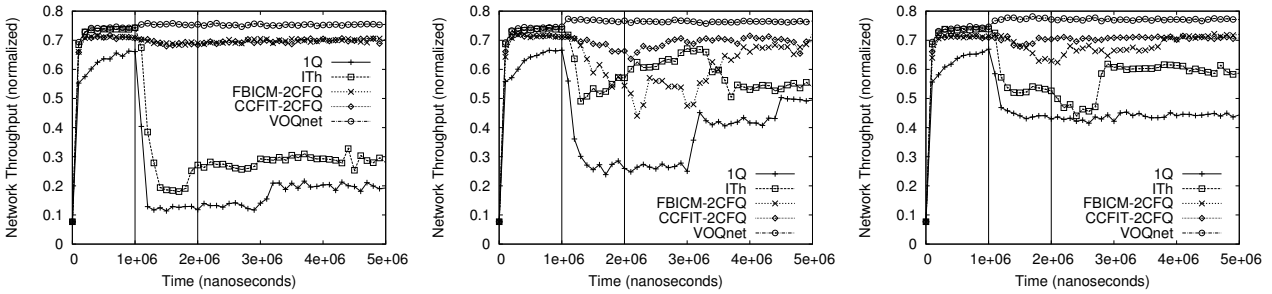
B. Throughput Results

Fig. 7 and 8 show the overall network throughput as a function of time for the network configuration #1, #2



(a) Configuration #1. Traffic Case #1 (b) Configuration #2. Traffic Case #2 (c) Configuration #2. Traffic Case #3

Figure 7. Throughput versus Time. Flow-Based Traffic Configuration



(a) 1 Congestion Tree. (b) 4 Congestion Trees. (c) 6 Congestion Trees.

Figure 8. Throughput versus Time (Configuration #3, Traffic Case #4).

and #3, using different traffic patterns. Considering all the plots, we clearly see how CCFIT outperforms the other CC techniques, even FBICM in some of the configurations. Specifically, in the plots shown in Fig. 7, the three CC techniques all show similar results, while 1Q struggles as soon as congestion is introduced in the network. In these scenarios the injected traffic is small, leaving the congestion less severe, making it rather easy for the different techniques to cope with the situation. In Fig. 7a ITh experience a drop in performance in the $[4ms, 6ms]$ time frame due to congestion detection at the left switch (Fig. 5). In configuration #2 case #3, Fig. 7c, we observe the tendency of ITh operating too slow as it takes time for this throughput to reach the level of the others.

In Fig. 8, we clearly see how CCFIT benefit from both doing congested-flow isolation and injection throttling as we increase the number of congestion trees from 1 to 4 and 6 in the network. In Fig. 8a, having one congestion tree in the network, CCFIT achieves excellent performance, at the level of FBICM. In this case FBICM, using 2 CFQs, have sufficient resources to isolate packets belonging to the single congestion tree. Notice that VOQnet, which achieves the maximum performance, has 64 queues per input port and uses 256 KB of memory, while the other CC techniques use less queues and uses less memory. The ITh scheme is not able to cope well with the situation. This could partly be caused by unfortunate CC parameter values for ITh, but then again finding optimal CC parameters for throttling is a challenging task [8], and is probably the main reason for CC not being in widespread use in IB networks today. Notice, however that the CCFIT is not as sensitive to the parameters (see in Fig. 8a), ITh reacts slow, even though the switches are VOQsw based, while CCFIT only has 1 NFQ and 2 CFQs per input port.

Fig. 8b shows the network throughput when 4 congestion trees are present in the network. Now FBICM struggles as it has not enough resources to isolate all the congested flows in a switch (FBICM uses only 2 CFQs per port): HoL-blocking is happening in the NFQs. CCFIT, on the other hand, shows a significant throughput improvement with respect to FBICM. The CCFIT injection throttling is able to release the resources used for isolating congestion flows (CFQs and CAMs) before the congestion-flow isolation part of the mechanism runs out of resources. Resources are released and made available to handle new congestion situations before the new situations arise. If we look at the ITh technique alone, it is in this scenario able to better cope with the congestion even though it still shows sign of instability and oscillation; the "saw-shape" effect. The 1Q scheme, which does not implement any HoL-blocking elimination mechanism at all, again achieves the worst results.

Finally, when 6 congestion trees appear in the network (Fig. 8c) we obtain similar results. This traffic pattern represents a situation where congested traffic is better balanced in the network. Again, CCFIT outperforms FBICM, while ITh needs more time to adjust the injection rate.

In conclusion, CCFIT significantly outperforms the other CC techniques, even FBICM, especially as the number of congestion trees grows, as the throttling part of CCFIT is able to release the resources used for congested flow isolation, preventing the congested-flow isolation part of the mechanism to run out of resources.

C. Fairness Study

Fig. 9 shows the throughput as a function of time for each individual traffic flow of config. #1 using traffic pattern case #1 (Fig. 5). In Fig. 9a, when no CC mechanism is present, not only does the throughput of the victim flow, F0, suffer from HoL-blocking, but the contributors to congestion also suffers from the parking lot problem as some contributors (the ones being the sole users of their input buffers) got more than their fair share of the link between switch 1 and end node 4. Enabling ITh, we not only improve the performance of the victim flow, but at the same time the parking lot problem is solved as the throttling happens at a per flow basis. A flow exploiting the parking lot problem will in return get more packets marked with a FECN (and then receive more BECNs), and then slow down. The improved fairness is clearly visible in Fig. 9b. Enabling FBICM, on the other hand, improves the throughput of the victim, even compared to ITh, but the parking lot problem prevails. The flows being the sole users of their respective CFQs still get more than their fair share of the bottleneck link towards end node 4. Actually, the unfairness has increased when using FBICM, as priority has been given to the victim flow addressed to the end-node 3.

Similar fairness study results, for network configuration #2, traffic case #2, are shown in Fig. 10. As can be seen in Fig. 6, there are now 4 congestion points, the parking lot problem possibly being present at two of them (switch 4 and switch 6). Again, both HoL-blocking and the parking plot problem leads to poor throughput and unfairness in the 1Q scenario (Fig. 10a), while the introduction of ITh improves performance in both aspects (Fig. 10b). FBICM improves the throughput even further, but again the unfairness in the network is dominant (Fig. 10c). Finally, Fig. 10d shows that both the best throughput and the highest degree of fairness is achieved by the CCFIT mechanism. It reacts fast due to the congested-flow isolation and improves fairness by reducing the injection rate on a per flow basis.

Summing up, besides efficiently eliminating HoL-blocking, CCFIT improves fairness in the network by solving the parking lot problem.

V. Conclusions

The link-level flow control of interconnection networks makes congestion spread from the oversubscribed link to the rest of the network. This has the adverse effect that flows that do not contribute to congestion will suffer from it. For this reason mechanisms that handle congestion are important.

The two classes of congestion control mechanisms previously described, attack the problem in different ways. Injection throttling approaches try to remove congestion by reducing the amount of traffic that is injected. On the other hand, congested-flow isolation methods lets congestion prevail, but confines the traffic that goes through the oversubscribed link to specially designated resources. This ensures that flows that do not contribute to congestion do not fall victim to it. These two approaches have different strengths and weaknesses.

In this paper we have presented three insights. Firstly, we have demonstrated that the weakness of injection throttling mechanisms is the reaction time of congestion events Secondly, we have shown that the weakness of congested-flow isolation is that it has limited scalability with respect to the number of congested points, and that it displays poor fairness between congested flows. Finally, and most importantly, we describe in detail CCFIT a mechanism that combines congested-flow isolation with injection throttling.

Our simulation results demonstrate that CCFIT extracts the best features of its two predecessors. In particular, injection throttling works in a way that limits the number of congested points that are alive in the network, and thus removes the scalability problem of congested-flow isolation. Furthermore, congested-flow isolation provides a quick and local response to congestion that removes the problems created by the slow reaction time of injection throttling. Finally, the good fairness properties of injection throttling are preserved in the combined method.

Acknowledgments

This work is jointly supported by the MEC, MICINN and European Commission under projects Consolider Ingenio-2010-CSD2006-00046 and TIN2009-14475-C04, and by the JCCM under projects PCC08-0078 (PhD. grant A08/048) and POII10-0289-3724.

References

[1] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks An Engineering Approach*, revised edition ed. Morgan Kaufmann, 2003.

[2] G. F. Pfister and V. A. Norton, ""Hot Spot" contention and combining in multistage interconnection networks," *IEEE Trans. Computers*, vol. 34, no. 10, pp. 943–948, 1985.

[3] W. Dally, "Virtual-channel flow control," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, no. 2, pp. 194–205, 1992.

[4] P. J. García, J. Flich, J. Duato, I. Johnson, F. J. Quiles, and F. Naven, "Dynamic evolution of congestion trees: Analysis and impact on switch architecture," *Proc. 1st HiPEAC Conf.*, pp. 266–285, November 2005.

[5] V. Jacobson, "Congestion avoidance and control," in *SIGCOMM*. ACM, 1988, pp. 314–329.

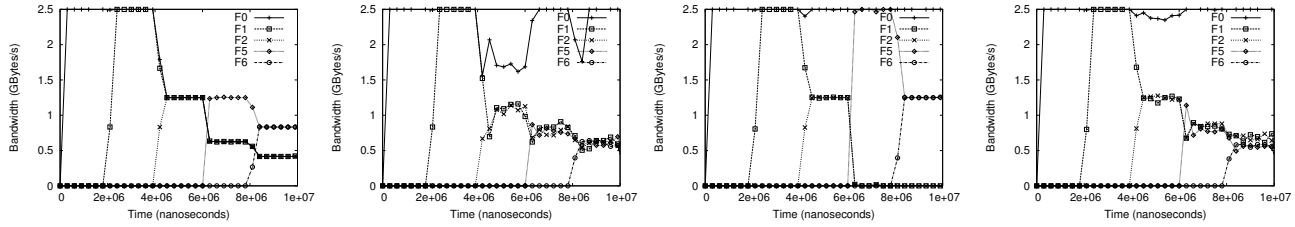
[6] L. S. Brakmo and L. L. Peterson, "TCP vegas: End to end congestion avoidance on a global internet," *IEEE Journal on selected Areas in communications*, vol. 13, pp. 1465–1480, 1995.

[7] C. Parsa and J. Garcia-Luna-Aceves, "Improving TCP congestion control over internets with heterogeneous transmission media," in *7th International Conferance on Network Protocols (ICNP99)*. IEEE Computer Society, 1999, pp. 213–221.

[8] E. Gran, M. Eimot, S.-A. Reinemo, T. Skeie, O. Lysne, L. Huse, and G. Shainer, "First experiences with congestion control in InfiniBand hardware," in *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, 2010, pp. 1–12.

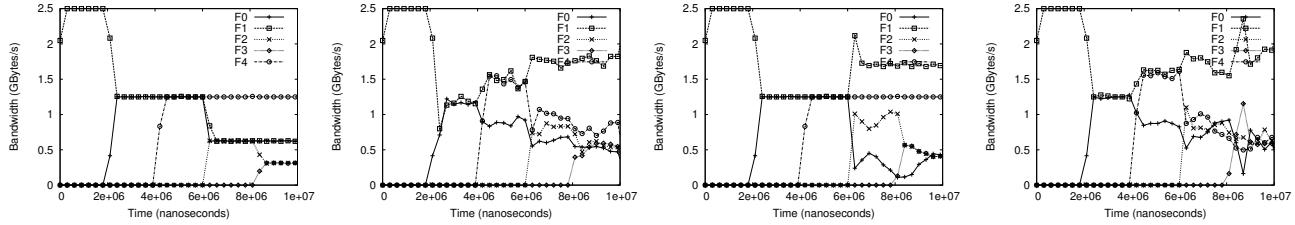
[9] J. R. Santos, Y. Turner, and G. J. Janakiraman, "End-to-end congestion control for InfiniBand," in *INFOCOM*, 2003.

[10] J. Duato, I. Johnson, J. Flich, F. Naven, P. J. García, and T. Nachiondo, "A new scalable and cost-effective congestion management strategy for lossless multistage interconnection networks," in *Proceedings of the 11th Symposium on High Performance Computer Architecture (HPCA)*, 2005.



(a) 1Q. (b) ITh. (c) FBICM 2-CFQ. (d) CCFIT 2-CFQ.

Figure 9. Flow Bandwidth versus Time (Configuration #1, Traffic Case #1).



(a) 1Q. (b) ITh. (c) FBICM 2-CFQ. (d) CCFIT 2-CFQ.

Figure 10. Flow Bandwidth versus Time (Configuration #2, Traffic Case #2).

[11] J. Escudero-Sahuquillo, P. J. García, F. J. Quiles, J. Flich, and J. Duato, "FBICM: Efficient congestion management for high-performance networks using distributed deterministic routing," in *LNCS Series - 15th Conference on High Performance Computing - (HiPC 2008), Bangalore, India*, December.

[12] E. G. Gran, E. Zahavi, S.-A. Reinemo, T. Skeie, G. Shainer, and O. Lysne, "On the relation between congestion control, switch arbitration and fairness," in *International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2011)*.

[13] M. Thottetodi, A. Lebeck, and S. Mukherjee, "Self-tuned congestion control for multiprocessor networks," in *Proc. of 7th. HPCA*, February 2001.

[14] J. Kim, Z. Liu, and A. Chien, "Compressionless routing: a framework for adaptive and fault-tolerant routing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 8, no. 3, pp. 229 –244, Mar. 1997.

[15] E. Baydal and P. López, "A robust mecahnism for congestion control: Inc," in *Euro-Par*, 2003, pp. 958–968.

[16] *InfiniBand architecture specification. Release 1.2.1*, InfiniBand Trade Association, Nov. 2007.

[17] *IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks - Amendment: 10: Congestion Notification.*, IEEE 802.1Qau-2010 ed., IEEE 802 LAN/MAN Standards Committee, 2010. [Online]. Available: http://www.ieee802.org/1

[18] J. R. Santos, Y. Turner, and G. J. Janakiraman, "Evaluation of congestion detection mechanisms for InfiniBand switches," in *IEEE GLOBECOM – High-Speed Networks Symposium*, 2002.

[19] J.-L. Ferrer, E. Baydal, A. Robles, P. López, and J. Duato, "Congestion management in MINs through marked and validated packets," in *PDP*, 2007, pp. 254–261.

[20] ——, "On the influence of the packet marking and injection control schemes in congestion management for MINs," in *Euro-Par*, 2008, pp. 930–939.

[21] T. Anderson, S. Owicki, J. Saxe, and C. Thacker, "High-speed switch scheduling for local-area networks," *ACM Transactions on Computer Systems*, vol. 11, no. 4, pp. 319–352, November 1993.

[22] W. Dally, P. Carvey, and L. Dennison, "Architecture of the Avici terabit switch/router," in *Proc. of 6th Hot Interconnects*, 1998, pp. 41–50.

[23] Y. Tamir and G. Frazier, "Dynamically-allocated multi-queue buffers for vlsi communication switches," *IEEE Transactions on Computers*, vol. 41, no. 6, June 1992.

[24] T. Nachiondo, J. Flich, and J. Duato, "Buffer management strategies to reduce hol blocking," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, pp. 739–753, 2010.

[25] W. Olesinski, H. Eberle, and N. Gura, "Scalable alternatives to virtual output queueing," in *Proc. IEEE International Conference on Communications*, 2009.

[26] J. Escudero-Sahuquillo, P. J. García, F. J. Quiles, and J. Duato, "An efficient strategy for reducing head-of-line blocking in fat-trees," in *LNCS Series. Parallel Processing, 16th International Euro-Par Conference, Ischia, Italy*, september 2010, pp. 413–427.

[27] P. J. García, J. Flich, J. Duato, I. Johnson, F. J. Quiles, and F. Naven, "Efficient, scalable congestion management for interconnection networks," *IEEE Micro*, vol. 26, no. 5, pp. 52–66, September 2006.

[28] G. Mora, P. J. García, J. Flich, and J. Duato, "RECN-IQ: A cost-effective input-queued switch architecture with congestion management," in *Proc. ICPP*, 2007.

[29] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch," in *IEEE TRANSACTIONS ON COMMUNICATIONS*, 1996, pp. 296–302.

[30] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: A tutorial and survey," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, March 2006.

[31] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM Transactions on Networking*, vol. 7, no. 2, pp. 188–201, Apr. 1999.

[32] J. Escudero-Sahuquillo, P. J. García, F. J. Quiles, J. Flich, and J. Duato, "Cost-effective congestion management for interconnection networks using distributed deterministic routing," in *Proceedings of the 16th International Conference on Parallel and Distributed Systems (ICPADS 2010), Shanghai, China*, december 2010.

[33] C. Gomez, F. Gilabert, M. Gomez, P. Lopez, and J. Duato, "Deterministic versus adaptive routing in fat-trees," in *Workshop on Communication Architecture on Clusters, as a part of IPDPS'07*, March 2007, p. 235.