High-Performance Architectures for High-Radix Switches

∼ PhD. Dissertation ∼

Gaspar Mora Porta

Valencia, December 2008

Advisors: José Flich Cardo and José Francisco Duato Marín

UNIVERSIDAD POLITECNICA DE VALENCIA

"*Arquitectura de los sistemas informáticos en red y sistemas empotrados*"
DISCA - Departamento de Informática de Sistemas y Computadores

Abstract

As the optimal radix for switches increases due to the benefits in lower latencies, overall reduction in cost and power consumption; the traditional switch architectures are no longer valid because of either low-performance or non-scalability with the number of ports.

This dissertation proposes a new switch architecture suitable for high-radix switches called *Partitioned Crossbar Input Queued* (PCIQ) that deals with one of the main constraints in high-radix switch design, the excessive memory requirements. Also, in general terms, PCIQ forms a new family of switch microarchitectures.

PCIQ relies on a smart partition of the crossbar into sub-crossbars, thus requiring less memory resources than other proposals for high-radix, yet obtaining high-performance and also increasing the arbiter efficiency. PCIQ uses two round-robin packet-based arbiters (one for each crossbar) that exhibit a linear cost and a logarithmic response time as the radix of the switch increases.

Here it is shown that PCIQ exhibits a cost (measured in terms of memory requirements, crossbar complexity and arbiter complexity) similar to or lower than basic organizations like CIOQ. However, it is able to achieve maximum switch efficiency for uniform traffic distribution, thus leveling costly organizations like BC.

The other big issue on high-radix switches is the HOL blocking problem, which reduces dramatically the switch performance. Traditional solutions for removing the HOL blocking problem were based on VOQ schemes, but having high number of ports on a high-radix switch prevents the use of any of them. In this dissertation, a new congestion management technique has been proposed. This solution is called RECN-IQ, is specific for IQ switches and differs from the original RECN idea (suitable only for CIOQ switches) in being highly efficient and simple to implement, reducing the memory requirements to the maximum. RECN-IQ introduced by first time a novel statistical approach for detecting congestion using just a single queue per input port.

By combining the PCIQ microarchitecture with RECN-IQ, a new switch architecture (called here PCIQ-enhanced) is derived and evaluated in this dissertation. The PCIQ switch architecture inherits the benefits of the Par-

titioned Crossbar microarchitecture in reducing the memory requirements for high-radix designs with the power of a congestion management technique that removes the HOL blocking dynamically, thus achieving maximum switch performance under all types of traffic.

We have seen that in modern interconnection networks it is mandatory the use of an effective congestion management technique in order to keep network performance at maximum level under congestion situations. Therefore, in this dissertation we describe the new congestion management technique (RECN-IQ) suitable for any type of IQ switches (which includes PCIQ). The idea behind RECN-IQ is, starting with a simple IQ switch with a single queue per input port, to add some extra queues dynamically allocated for congested packets. Congestion is detected as soon as HOL blocking begins to act, setting aside (in those extra queues known as SAQs) the congested packets in an efficient manner. Therefore, HOL blocking is completely eliminated (as proven by the simulation results). The hardware requirements for RECN-IQ, as we have seen, are reduced, making feasible its implementation on any IQ-based switch architecture like PCIQ.

In order to prove that fact, a feasible and realistic switch architecture implementing RECN-IQ has been proposed and described in detail. Moreover, we have detailed every functional unit and structure required to implement RECN-IQ on an Input-Queued switch architecture. This is the first time since the RECN proposal back in 2005 that a RECN-like congestion management technique has been implemented in such a detailed level.

Results proved that by using RECN-IQ switches, the network will benefit from low cost switches and high-efficiency under any type of traffic pattern or network circumstances. All this makes the network predictable and stable in performance, no more drops in throughput because of congestion.

Acknowledgements

Above all I would like to acknowledge my darling Raquel. We have walked together this life since we decided to share our lives many years ago, helping each other, growing, and sharing many experiences jointly. She was responsible of supporting me in pointing my career towards computers, so I could finally pursue my passion since I had my first computer (a C-64 by the way). She has been my muse all this years; inspiring me in putting passion, efficiency and love in this job. For that and many other moments in which she has been essential, this work should be honored to her as well. This work represents an academic milestone for us as a couple. Now it should be time for us to have her pursuing her PhD in Physics.

Of course, I am especially grateful to my two advisors, José Duato and José Flich for guiding and inspiring me during all my PhD studies. Knowing both of them has been one of the best things in my life.

José Duato opened to me this world of interconnection networks and computer architecture. The moment I met him I realized the amazing person I had in front of me. He is one of these persons that make history, with a mind so powerful and stimulating that can really change the world, all with humility and friendliness. He has been a role model to follow and I've been more than honored by his friendship and guidance these years; his ability to find the best approach and strategy for solving any problem still amazes me. Duato gave me the opportunity to join the Parallel Architectures Group (GAP) and I am absolutely indebted to him.

Likewise, in José Flich I also discovered a truly friend. He taught me important lessons of hard work, and enthusiasm for research. I am extremely happy that I could share a lot of moments with him these past years. I really thank him for mentoring me during my stay at the GAP; always ready for helping like a good close friend should be. Working with him is always a pleasure; he can always provide a smart overview on every subject we are working on. Also, we made such a good team, not only in the laboratory but in the sport courts as well, where we retired undefeated.

I would like to thank my family for understanding my dedication to this work that stole so many moments from them. Apart from their unconditional love and support there are many things I have to thank them. When I was a kid, my mother introduced me into the fine arts and creativity; helping me in developing my creativity and sense of esthetics in every work I am involved. From my father I learned the importance of the logic and having a structured way of thinking. He approached to me the wonderful world of mathematics and I am really thankful to him for that. Both of them helped me in providing the culture resources I needed to grow; either through enjoyable lessons and conversations, or with the appropriate books I always found at home. I have to thank my grand-parents from whom I learned the same educated values; they provided me with a stimulating environment as well. They showed to me the importance of language precision, respect and humbleness. Thanks to my brother for helping me in growing; he has been always a true support and still encourages me to enjoy life and challenge the status-quo. In the same way, I am really grateful to my family in-law for their support.

I have to acknowledge Manuel López Pellicer for introducing Duato to me. Manuel was key for starting this period in my life at the Parallel Architectures Group (GAP) as a PhD student. As this dissertation represents the end of that period, I have to thank all the people I had the luck to meet there. I specially thank Blas for being my friend there; given that we started at the same time in a similar position, our paths at the research group were kind of bounded from the beginning and I hope that our friendship helps in keeping us in touch. There are many especial moments I shared with him in conferences, soccer games, working together, etc.; but the epic time we had when we won our trophy as best table-football players of the School are always at the top of my mind, that story owes to be published someday. I am also lucky that I could meet people like Jose Miguel, the “senior-intern”, he provided unforgettable moments at the lab. I have to thank Crispín, Ricardo, Jordi and Paco as well; working with them has been delightful and full of really good times. I would like to thank Andres; he has become an important piece in my life. I thank him for being always ready to help, no matter how many times I asked him for. I would like to mention Carles, Rafa, Héctor, David, Noel and Samuel; what a pity not having more time to stay working with them because they are awesome.

Apart from my colleagues in the lab, there are other persons from the GAP I owe to mention. First I have to thank Pedro López, he has been inspirational during these four years and ready to give smart feedback on almost any subject. It has been such an honor to work with him. Likewise, I would like to thank Antonio Robles, Federico Silla, Mª Engracia Gómez, Vicente Santonja, and Elvira Baydal; they have contributed in several ways to this work. And thanks to the rest of people from the DISCA Department, all of them contributed in making me feel at home while working there.

But the GAP not only included people in Valencia. Since I started we have been working very closely with amazing people from Universidad Castilla-La Mancha, in Albacete, consequently I would like to thank all of them. But I have to give especial thanks to Pedro García for his sense of humor and meticulous work, it has been a pleasure to co-author papers with him; and I have to acknowledge Paco Alfaro for his readiness to help in any matters that come up.

Working with Duato brought me the opportunity to meet Olav Lysne, I would like to acknowledge him for his insightful comments regarding this work. It has been a pleasure to have the privilege of working with Olav and almost a dream come true to have co-authored some papers with him.

Another person I really thank is Ulrich Brüning, he opened to me the research group he leads in Mannheim, the Computer Architecture Group, where I stayed four months. That gave me the opportunity of working with extraordinary people Ill never forget like Heiner, Mondrian, David, and, especially, Holger, who has become a good friend since then and has helped so much in this work. I am really grateful to him for that and more. Of course I owe a mention to Marlis, who was responsible in making me feel like home, taking care of everything I required back these days.

Likewise, I would like to thank William Dally for inviting me to join his group (Concurrent VLSI Architecture) in Stanford University the summer of 2007. Working with one of the best teams in the world has been one of the greatest experiences in my life. Thanks to David, James and John for all the moments we shared there.

Realizing that my time with the GAP group was bounded to my PhD student status, an end was on the horizon. At first I couldnt imagine a more stimulating place for my mind than the GAP, but then two incredible persons come into my life with an amazing proposal. I want to give special thanks to Mani Azimi and Donglai Dai for giving me the opportunity to keep developing my skills in such an exciting and motivating environment like Intel Corporation. It started with an invitation for working in Santa Clara, CA as an intern last summer, but soon they gave me the chance of pursuing a career in the microprocessor industry with Intel, an awesome company. During my internship there I could met the amazing people who are now my colleagues and helped in creating such a good and challenging environment at least as great as the one I had at the GAP in Valencia. Thanks to Akhilesh, Roy, Partha, Naveen, Ani, Anahita, Ching- Tsun, Hari, Dongkook and SeungJoon for that.

My final thanks are to Antonio González and Shashi Kumar for accepting on being part of the tribunal evaluating my PhD defense. It is going to be a total honor for me to have them in the tribunal that day.

Let me reiterate my thanks to everyone who “jumped” into my life at some time, contributing to build what I am right now. Thanks to all of you.

To Raquel

Contents

Chapter 1

Introduction

Internet! Is that thing still around?
– Homer, "The Simpsons"

Digital systems can be found everywhere in modern society. Their applications range from control systems in industry, cars and appliances; to computers, cell-phones and communication systems. Three basic building blocks form the digital systems: logic, memory and communication.

The logic processes data by either transforming it (e.g. arithmetic operations) or making decisions. The memory takes care of storing that data for further uses. Finally, the communication system moves data from one element of the system to another.

In this dissertation, the attention is paid to the communication part of the digital systems. That part is indeed one of the most important and challenging nowadays. The reasons include that, whereas the logic and memory continue scaling down in size, the wire and pin density does not scale that fast. Moreover, most of the power consumed by a system is used to drive wires. Also, when designing a digital system, most of the clock cycle is spent on wire delay, not gate delay.

This increase in importance of interconnection networks is reflected on current trends in today's microprocessor manufacturers. The current trend is to boost computing performance by the use of many computation units (cores) within their chips, relying performance on the aggregate work capacity from all of these cores. This means that the design of the interconnection network that connects all those elements is critical so it not becomes the bottleneck of the system.

1.1 Interconnection Networks

A good definition of interconnection network can be found in [1]. It states that "*an interconnection network is a programmable system that transports*

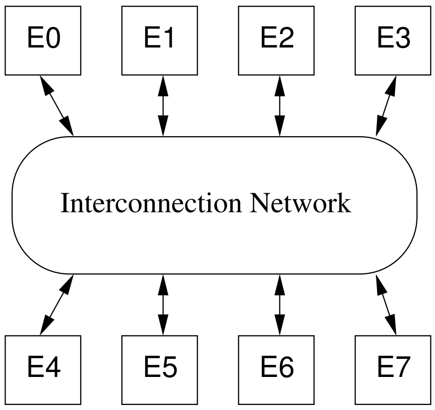


Figure 1.1: Functional view of an interconnection network.

Interconnect Family	Number of Supercomputers	Share
Gigabit Ethernet	284	56.8%
Infiniband [2]	121	24.2%
Proprietary	40	8.0%
SP Switch [3]	17	3.4%
Myrinet [4]	12	2.4%
NUMALink™	8	1.6%
Cray Interconnect™	8	1.6%
Quadrics [5]	5	1.0%
Crossbar	3	0.6%
Mixed	2	0.4%

Table 1.1: Different interconnect families used in the world's top 500 supercomputers as of June 2008 [6].

data between terminals". Therefore, the interconnection network is responsible of transport data between the subsystems of a digital system.

Figure 1.1 depicts the functional view of an interconnection network. Here, 8 elements (E0 - E7) are connected by an interconnection network using bidirectional channels (arrows). The network is programmable in the sense that at some point in time two elements (say E0 and E3) communicate thanks to a configuration of the network, and at a step later, another pair of elements (say E2 and E7) communicate thanks to a network reconfiguration.

There are many examples of the use of interconnection networks in digital systems. For instance, in computers, it is used for connecting the processor (or processors) with the memory and I/O at the system level. Even inside the processor, it connects several cores within the chip in modern multicore architectures. They are known as networks on chip (NoC).

Another typical example is the communication switches, where the interconnection network connects all inputs and outputs of the network routers. These include IP routers that form the backbone of Internet.

If we look at today's supercomputers, the corner stone of their performance is the interconnection network because they rely on having many computing devices working together. In Table 1.1 it is shown the share of the different interconnects used on the top 500 supercomputers as of June 2008 [6]. For instance, the BlueGene/L[7] supercomputer at the Lawrence Livermore National Laboratory (Lawrence, CA, USA) requires the interconnection of 212,992 computing cores (as of June 2008).

Also, in today's data centers, the interconnection network is fundamental in providing fast and reliable access to information spread on different cabinets of the system. A well-known example is Google and its powerful cluster which servers millions of search results 24 hours a day, 7 days a week.

Traditionally, the interconnection network was realized by using buses. But this isn't a solution anymore due to the increasing requirements in interconnection performance. Buses belong to shared-medium networks category because the transmission medium is shared by all communication devices [8]. Since only one device is allowed to use the network at a time, this type of interconnection network scales poorly with the number of connected devices.

To overcome the scalability issue for buses, *direct networks* are widely used. A direct network (or point-to-point network) consists of a "*set of nodes, each one being directly connected to a (usually small) subset of other nodes in the network*" [8] (See Figure 1.2). A node is any system or set of elements with communication requirements. It can be just a processor, a processor plus memory, graphics elements, memory controllers, I/O interfaces, etc.

The key element, from our perspective, on the node is the *router*. The router is the device which handles message communication among nodes. Each router has direct connections to its neighbors, either by bi-directional links or by two unidirectional links (one for each direction), these are known as *channels*. It is important to note that as the number of connected nodes increases, the aggregate bandwidth of the total system also increases.

There are several ways of connecting nodes by a direct network, defining the *topology* of the network. In Figure 1.3 some example topologies are shown. Choosing the correct topology when designing a network is fundamental. The following network properties can be defined from a graph representation of the network (vertexes are nodes and edges are channels):

- *Node degree*: Number of channels connecting that node to its neighbors.
- *Diameter*: The maximum distance between two nodes in the network.
- *Regularity*: A network is *regular* when all nodes have the same degree.
- *Symmetry*: A network is *symmetric* when it looks alike from every node.

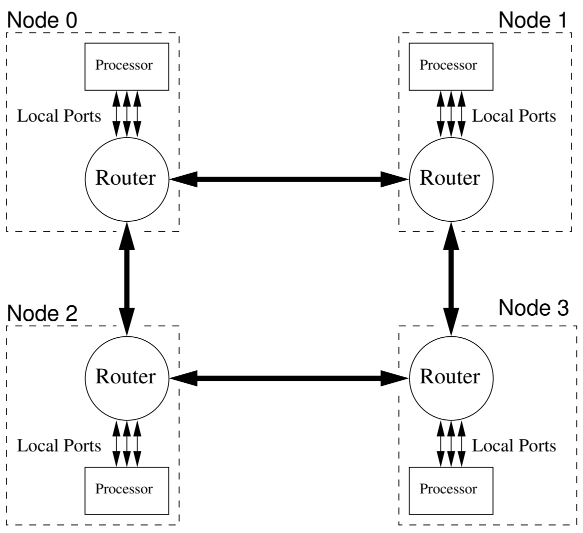


Figure 1.2: Typical elements of a direct network. Each node communicates with the router through local ports; and each router has direct connections to other routers.

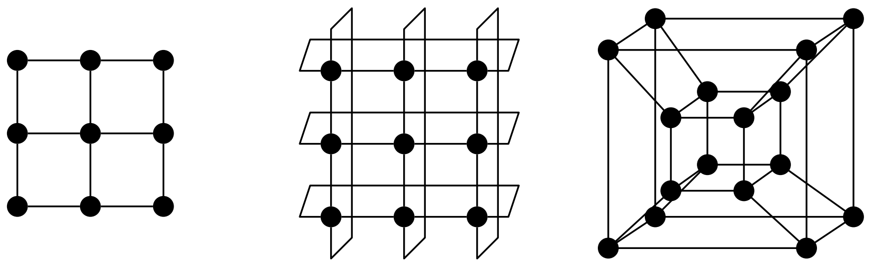
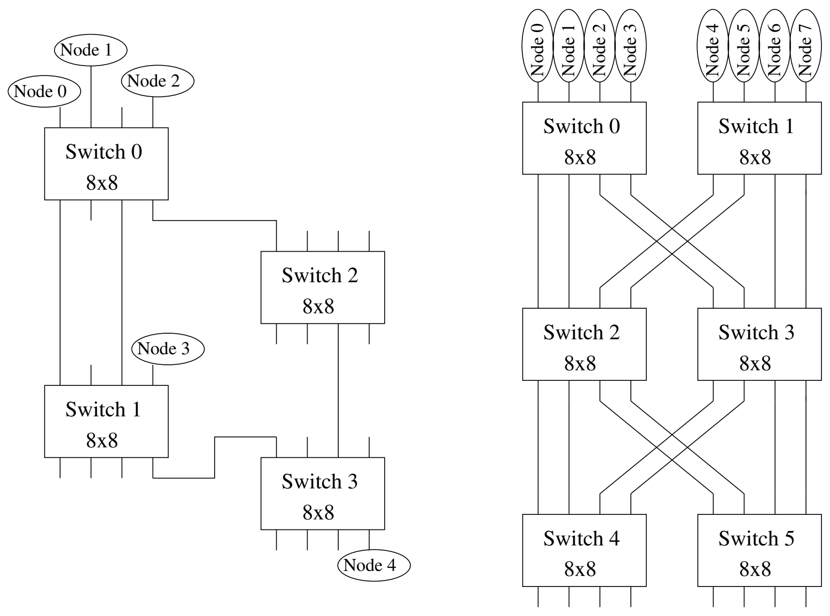


Figure 1.3: Different topologies for direct networks.

Another major class of interconnection networks can be derived from direct networks by taking the router out of the node. Now the router can be used independently as a communicating device and it is called *switch* in this context. This type of networks is known as *indirect networks* (or *switch-based networks*) and one difference with direct networks is that some switches do not have a computing device attached so they do not generate traffic by themselves. In Figure 1.4.(a), an example of an indirect network with irregular topology can be seen.

A *switch* is basically a *crossbar network*, i.e. it allows to connect any input to any free output simultaneously without contention (unless two or more inputs request the same output). Nodes are connected to switches through a *network adapter*.

The most common configuration for indirect networks is the *multi-stage*



(a) Irregular (b) Multi-Stage

Figure 1.4: Example of two indirect networks, also known as switch-based networks.

interconnection network or MIN. In a MIN, see Figure 1.4.(b), switches are grouped in stages. The communicating devices (or nodes) are connected to the first stage of switches whereas the next stages are connected following some connection pattern which provides full connectivity to the network. The connection patterns are based on permutations on the node identifier expressed in some base, for instance the perfect shuffle permutation corresponds to the following:

$$\sigma^k(x_{n-1}x_{n-2}\ldots x_1x_0) = x_{n-2}\ldots x_1x_0x_{n-1} \tag{1.1}$$

Both direct and indirect networks are mainly characterized by three factors: topology, *routing* and *switching*. We are already familiar with the topology concept. Let us have a look on the routing and switching aspects.

Communication among nodes is usually realized by sending packets with information. These packets must travel from source to destination, performing several hops throughout the network. For a given topology, the routing mechanism provides the route a packet must follow towards its destination. There are two major classes of routing mechanisms: deterministics and adaptives. In the former, the path followed by a packet is predetermined. In the latter, the packet may follow different routes depending on network circumstances. Because the number of resources within a network is finite, it can happen that a packet is waiting for a resource currently hold by another packet which is, in turn, waiting for the resources hold by the first packet; this is an example of a harmful situation known as *deadlock*. It is obvious that deadlocks must be avoided by all means. There is another type of blocked situation, known as *livelock*, when the packet can advance but can not reach ever its destination because the required resources by the packet are always busy, so the packet can only "orbit" the destination. Of course, any network design must also avoid *starvation* situations, where some packets are treated unfairly in the assignation of resources (giving priority to other packets), taking long time (even infinite time) to be served.

Switching refers to how packets are stored and forwarded between routers (or switches). This is related with flow control, which regulates the advance of packets dividing them on smaller information units, known as *flits* (a contraction from "flow control bits"). A flit is the minimum block of data that requires flow control synchronization, like request from sender and acknowledgment from receiver. Flow control is tightly coupled with buffer management because flits are usually buffered during their journey, and before sending anything from one place to another it must be ensured that there is enough free space, typically by means of *credits* or "*stop & go*" signals (also known as Xon/Xoff flow control). In Figure 1.5 there is an example of the credit-based flow control mechanism, whereas on Figure 1.6 it is shown the Xon/Xoff flow control mechanism.

Packets can be transmitted following an *store-and-forward* fashion, i.e. before start re-transmitting a packet, it must be fully stored at the current buffer. This penalizes the time a packet takes for travel along the network with the number of hops.

The switching technique known as *Virtual-Cut-Through* (VCT) starts re-transmission as soon as there are enough flits with routing information and, of course, the requested resources are available. Whenever a packet stops at some buffer, enough space for storing the entire packet must be guaranteed.

Another approach is to use *wormhole switching* (WH) [9], in which the buffers are usually very small so a full packet can not be stored on the same buffer, thus a packet spreads along several buffers. The benefit of having small buffers is shadowed by the fact that the network becomes more deadlock-prone (although techniques like virtual-channels can alleviate this), and its behavior during congestion[1] situations is tough, making very difficult for the network to recover from that point.

[1]Congestion refers to the situation when the network (or parts of it) is oversubscribed so many packets are contending for the same resources. It usually implies that packets spend long time waiting stopped at the buffers, thus occupying resources required for another packets to make progress.

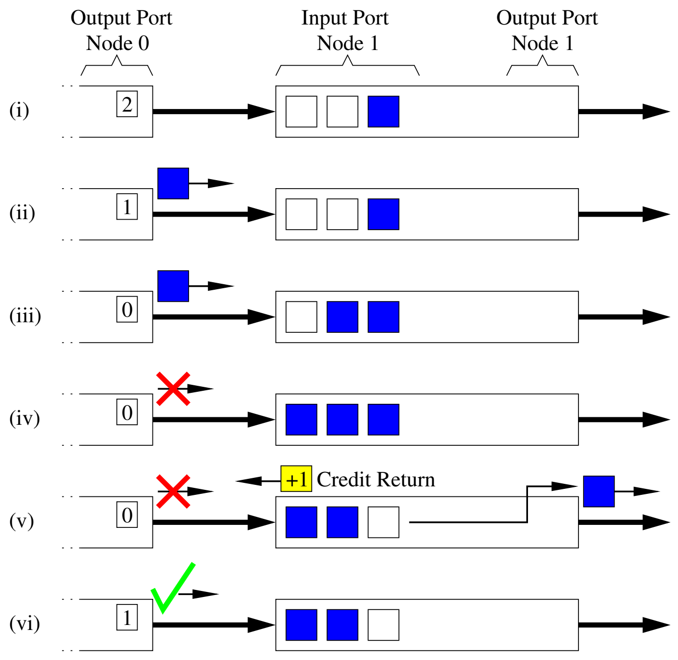


Figure 1.5: Example of a credit based flow control procedure. (i) There are two credits at the output port of node 0, meaning that there is space for two more packets at the input port of node 1. (ii) A packet is sent from node 0 so one credit is decremented. (iii) Another packet is sent, consuming all credits. (iv) Because there are no credits, no packet can be sent from node 0. (v) When a packet is sent from node 1, a slot becomes empty for new incoming packets, therefore a credit is returned to node. (vi) Upon reception of such credit at node 0, the counter is incremented to one, hence packet transmission can be resumed.

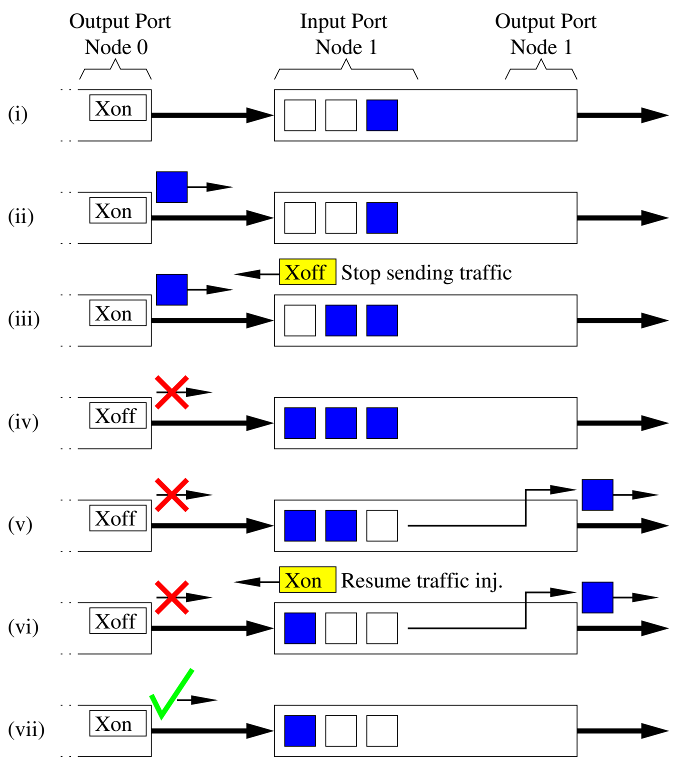
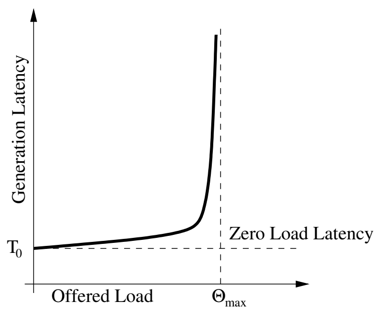
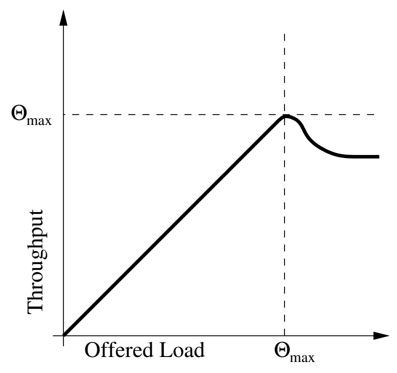


Figure 1.6: Example of an Xon/Xoff flow control procedure. (i) Output port at node 0 is on Xon state, that means that packets can be sent to node 1. (ii) Node 0 keeps sending packets as long as it is on Xon state. (iii) When the number of packets at node 1 reaches the Xoff threshold (2 packets in this case) an Xoff notification is sent to node 0 in order to stop transmission. (iv) Upon reception of such Xoff notification at node 0, transmission of new packets towards node 0 is stopped. (v) and (vi) When the number of packets at node 1 reaches the Xon threshold (1 packet in this case), because it is forwarding packets, an Xon notification is sent to node 0 indicating that there is space for new packets to be sent. (vii) Upon reception of such Xon notification at node 0, transmission of packets towards node 1 is resumed.



(a) Offered load versus generation latency



(b) Offered load versus throughput

Figure 1.7: Different curves showing network performance. Θ_{max} indicates the maximum throughput delivered by the network and T_0 the zero-load-latency.

1.2 Metrics for Measuring Network Performance

There are two main metrics for measuring the interconnection network performance. The first one is the *latency*, or the time it takes for a packet to travel from source to destination. If it is measured from the time the header is injected on the network to the moment the last flit of the packet arrives to destination, then it is known as *network latency.* If measuring from the time the packet is generated, then it is known as *generation latency.*

Of course, having low latencies is mandatory in high-performance interconnection networks. The latency is also an indicator of network saturation, it goes up when packets are waiting more than usual at any part of the network. It is evident that the latency is tightly coupled with the traffic conditions on the network, so when the network is empty and only one packet is sent (no contention with other packets for any resource), its latency is known as *zero-load-latency* and gives us the lower bound for latency in that particular network.

The second metric is the *throughput* delivered by the network. That is, the total bandwidth (in bits/s for instance) accepted by all destinations. For low injection bandwidths, the network is able to deliver all injected traffic but beyond some injection rate (saturation point) the network saturates and the maximum throughput is achieved.

While the latency is related on how fast packets are moving in the network, the throughput tells about how many packets at most can be delivered by the network per unit of time.

On Figure 1.7.(a) it is shown a typical curve of offered load versus latency. As can be seen the zero-load-latency (T_0) gives us the lower bound for latency while the maximum accepted load (Θ_{max}) establishes the asymptote where generation latency goes to infinite (packets are created at a faster rate than they are delivered). Figure 1.7.(b) depicts the complementary curve, that is, offered load versus throughput. There is a maximum load or throughput (Θ_{max}) beyond which the network is no longer able to deliver all injected traffic.

1.3 Switch Architecture

The general mission of a switch is to forward incoming packets at the input ports to the output ports. If the packet requires exclusive access to a resource already used by another packet, then that packet must wait in place, thus buffer space has to be provided by the switch in order to deal with such situations.

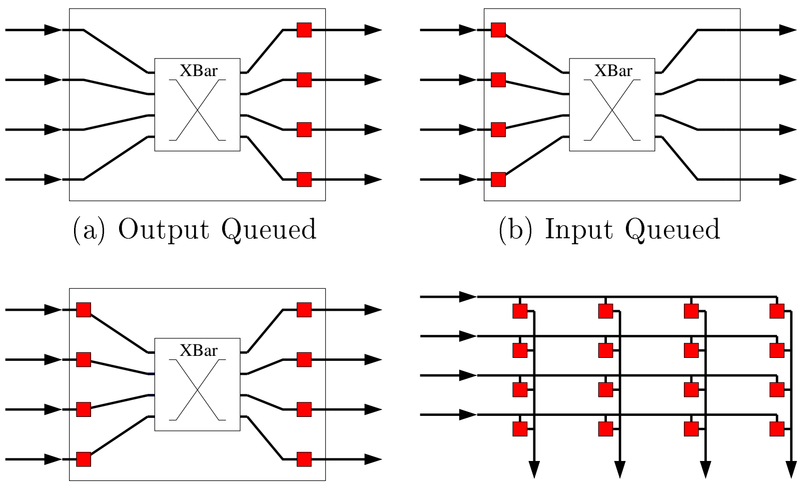
Several architectures have been proposed in the past years for the switches used in networks. Initially, a single shared memory was used for buffering packets, but when it no longer provided enough bandwidth to cope with aggregate port bandwidth, switches with multiple memories started being used. Those switches were originally deployed with memories only at the output links. Such switches are known as *Output Queued* switches (OQ) (Figure 1.8.(a)).

An OQ switch connecting N inputs with N outputs ($N \times N$ switch) must operate internally N times faster than the link (this is known as *speedup*), so the switch is able to manage the simultaneous arrival of up to N packets at the input ports, all of them requesting the same output (and therefore all of them put into that same output). Because of that speedup, packets may arrive to the output faster than they can abandon it (link transmission is slower), then buffer space must be provided at the output port.

The performance of an OQ switch is maximum, but nowadays the links are operating at such high frequencies, providing huge bandwidths, so implementing the speedup required in an OQ switch is unfeasible even for low number of ports.

Switches using only buffers at the input ports (known as *Input Queued* switches, IQ [10, 11]) (Figure 1.8.(b)) do not use speedup. The buffer requirement at the inputs comes from the necessity of buffering packets contending for the same output port at the same moment (only one can be granted, the rest must wait in place).

Buffers are typically implemented as FIFOs because of hardware constraints (they are required to be really fast), therefore if a packet at the head of a queue is blocked because it requires access to an output which is currently busy, then packets behind must also wait even if they request free outputs. This effect, known as Head-of-Line (HOL) blocking, affects dramatically the performance of a switch. IQ switches can only achieve a modest 58% of maximum switch efficiency [12].



(a) Output Queued (b) Input Queued

(c) CIOQ (d) Buffered Crossbar

Figure 1.8: Switch architectures.

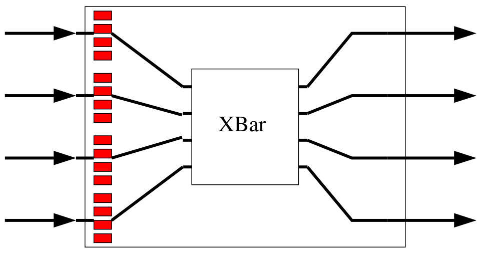


Figure 1.9: Virtual Output Queuing (VOQ) on an IQ switch, there are as many queues per input port as outputs has the switch.

One solution to eliminate the HOL blocking in IQ switches is the use of N FIFO queues (for an $N \times N$ IQ switch) at every input port and mapping the incoming packet on a queue associated to the requested output port. This technique is known as Virtual Output Queuing (VOQ) [13, 14] and must be used in conjunction to more complex logic in order to allow that each input port can request multiple outputs at the same time. It also increases the queue requirements quadratically with the number of ports (N^2 queues for an $N \times N$ switch). Therefore, as the number of ports increases, this solution becomes too expensive. In Figure 1.9 it is depicted a VOQ scheme on an IQ switch.

Another way to improve the efficiency of IQ switches is the use of internal speedup. In this case, as the internal bandwidth is higher than the aggregate link bandwidth, buffer space at the output ports are required. Such switches are known as *Combined Input Output Queued* switches (CIOQ) (Figure 1.8.(c)), and for these, the number of required memories is $2N$. Speedup can be implemented by using internal data paths with higher transmission

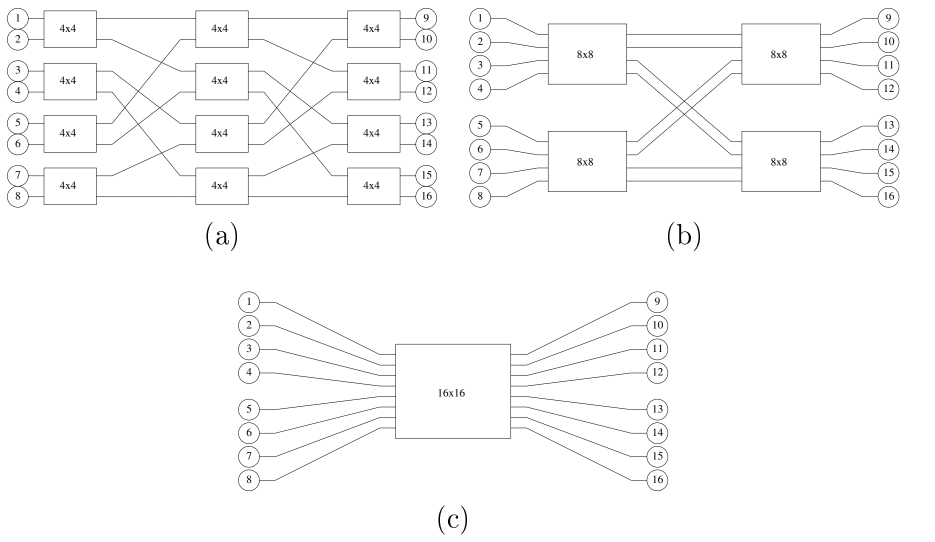


Figure 1.10: Switches with different radix can be used for connecting the same number of nodes. As the radix increases, the number of required switches and links is reduced. Sixteen nodes can be connected using (a) 4×4-switches, (b) 8×8-switches, or (c) a single 16×16-switch.

frequencies or wider transmission paths. Thus, as the external link bandwidth increases, sustaining the speedup value becomes difficult. A speedup of 4 (with VOQ at the inputs) is proven to be enough in providing the same high-efficiency of an OQ switch [15].

More recently, another switch organization has become popular. The *Buffered Crossbar* (BC) (Figure 1.8.(d)) switch organization (like the one proposed in [16]) uses a memory at every crossbar point. An input link is connected to N memories, each one connected to a different output link. The BC organization implements internal speedup, as many inputs can forward a packet to the same output at the same time. Additionally, such memory organization eliminates the HOL blocking (every packet is mapped to the memory associated with the requested output port). As a consequence, the BC organization requires low-cost arbiters per output port. However, the problem with such organization is that the number of memories increases quadratically with the number of ports (N^2), thus limiting scalability.

1.4 High-Radix Switches

The radix of a switch is simply the number of ports it serves. Choosing the appropriate radix for switches when designing an interconnection network is crucial because it directly affects performance, global cost, and power consumption of the system.

Figure 1.10 depicts three configurations using switches with different radices connecting the same number of nodes. It is trivial to see that as the radix increases, the number of required switches and links reduces.

As long as the cost of a network is roughly proportional to the number of channels (related with the number of pins and connectors), by increasing the radix of the switches used, the global cost goes down. In Figure 1.10, three systems with different radices have been plotted. As can be seen, the number of channels range from 48 for Figure 1.10.(a) (radix 4), to 24 for 1.10.(b) (radix 8). In the system in Figure 1.10.(c) just 16 channels are required (radix 16).

On the other hand, the power consumption is proportional to the number of switches in the network, hence, having less switches by using high-radix switches reduces the overall power dramatically.

But there are other benefits apart from the obvious reduction in power and cost when using high-radix switches. For instance, the latency is tightly coupled with the number of ports implemented on the switches.

There are two main contributions to the latency experienced by the packets traveling along the network. The first one is the serialization or the time expended on squeezing the packet on (usually) narrower channels than its size. The second main contribution is the time it takes to "process" a packet on each switch it has to traverse, thus it is proportional to the number of hops along a packet's route from source to destination.

As we increase the radix of the switches, the number of hops reduces, therefore the latency goes down at first. But if we keep increasing the radix, at the end, the benefits in reducing the number of hops is compensated by the negative effect of serialization, so there is an optimal radix for each technology.

In the process of designing an interconnection network, only after carefully analyzing the current technology and packaging constraints, the most appropriate topology can be selected. In the past, the packaging constraints suggested that networks made of low-radix switches exhibit lower packet latencies [17, 18] but this is not the case anymore.

In order to find the characteristics of the topology which minimizes latency, let us calculate the so-called zero-load latency. This is the latency for a packet that travels from source to destination without any contention at all, so there is any delay introduced because a resource is being used by another packet.

Therefore, under this situation of no contention at all, there are two contributions to the total latency, the header latency (T_h) plus the serialization latency (T_s). Hence, the zero-load latency can be written as:

$$T = T_h + T_s = t_r H + L/b \tag{1.2}$$

where H is the number of hops, t_r the time the packet takes to cross a router, L the length of the packet and b the link bandwidth.

Name	Year	B	t_r	N	L
J-Machine [19]	1991	3.84 Gb/s	62 ns	1024 nodes	128 b
Cray T3E [20, 21]	1996	64 Gb/s	40 ns	2048 nodes	128 b
SGI Altix [22]	2003	0.4 Tb/s	25 ns	1024 nodes	128 b
<projection> [23]	2010	20 Tb/s	5 ns	2048 nodes	256 b

Table 1.2: Technology parameters for different chip routers.

If we replace the number of hops by the average number of hops assuming uniform traffic on an N-node network made of k-radix switches,

$$H = 2\log_k N \tag{1.3}$$

and by applying the fact that the global bandwidth of a router chip is a constant (B) that have to be divided among all k ports (bidirectional ports, thus divided by $2k$) of the router, the link bandwidth b can be replaced by

$$b = \frac{B}{2k} \tag{1.4}$$

Then, the expression 1.2 becomes

$$T = t_r 2\log_k N + 2kL/B \tag{1.5}$$

Table 1.2 shows values of these parameters for interconnection networks of large supercomputers with single-word network accesses, as representative of different technologies in the past years.

With expression 1.5 and Table 1.2, the latency for different radices can be plotted. As can be seen on Figure 1.11, by increasing the radix, initially the latency goes down because the number of hops is reduced. Beyond the optimal radix (which gives the lowest latency) the latency goes high again as the serialization penalty grows.

Now, if we differentiate T with respect of k, and set dT/dk equal to zero in order to minimize latency, we obtain:

$$\frac{d}{dk}T = \frac{d}{dk}\left[2t_r \log_k N\right] + \frac{2L}{B} = 2t_r\frac{d}{dk}\left[\log_k N\right] + \frac{2L}{B} \tag{1.6}$$

$$0 = t_r\frac{d}{dk}\left[\log_k N\right] + \frac{L}{B} = t_r\frac{d}{dk}\left[\frac{\log N}{\log k}\right] + \frac{L}{B} = t_r \log N\left[\frac{-1}{k\log^2 k}\right] + \frac{L}{B} \tag{1.7}$$

$$\frac{L}{B} = t_r\left[\frac{\log N}{k\log^2 k}\right] \tag{1.8}$$

$$k\log^2 k = \frac{Bt_r \log N}{L} \tag{1.9}$$

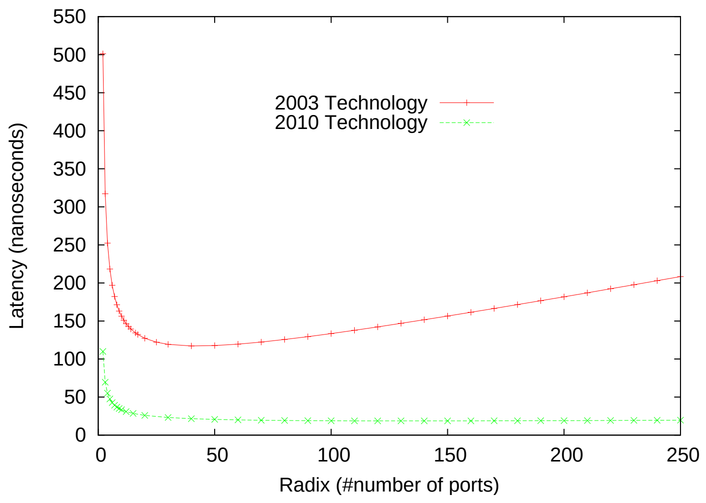


Figure 1.11: Latency versus radix for two different technologies, one corresponding to year 2003 (SGI Altix) and a projection for year 2010.

Name	Year	Aspect Ratio	Optimal Radix
J-Machine [19]	1991	13.84	5
Cray T3E [20, 21]	1996	163.74	18
SGI Altix [22]	2003	595.41	42
<projection> [23]	2010	3274.75	135

Table 1.3: Aspect ratio and optimal radix for different technologies.

Like in Kim's paper [23], we can define the aspect ratio A of the router as

$$A = \frac{Bt_r \log N}{L} \tag{1.10}$$

so it determines the optimal radix that minimizes latency for a given technology. In Table 1.3, both the aspect ratio for different technologies and the subsequent optimal radix are calculated. As can be seen, the optimal radix has been increasing over the years.

There are two main issues to deal with regarding the design of a high-radix switch. The first one is related with the difficult, for current VLSI technologies, of wiring within the chip in order to connect many inputs with many outputs when a centralized allocator[2] (or scheduler) is used. This

[2]An allocator is simply a device, usually made of several arbiters, that matches the inputs of the switch with the outputs. When each input can request more than one output at once, then it is used an allocator that produces (at least) an approach to a maximal (or even the maximum) number of connections each cycle. This is usually a complex logic structure.

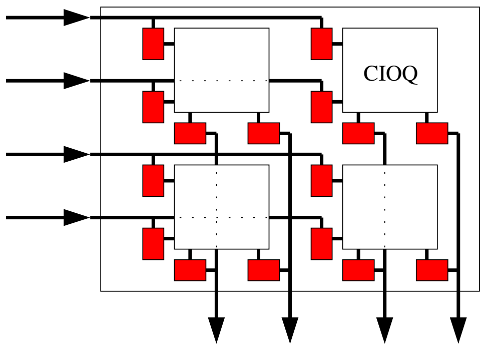


Figure 1.12: Hierarchical Crossbar schematic.

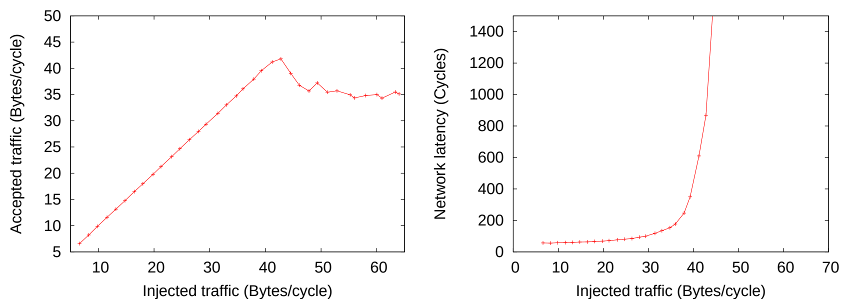
problem is overcome by using hierarchical arbiter schemes, as will be shown later in Section 2.2.4. Basically, the idea consists on grouping the input requests to the scheduler through smaller arbiters, submitting only one winning request to the next stage in this hierarchical scheme; in doing so, only few winning requests propagate to the next stages (placed along the switch), so the allocation is performed on a distributed manner.

The second problem is the HOL blocking which, as seen in Section 1.3, reduces the switch throughput to intolerable levels. As long as non-scalable techniques such as VOQ can not be applied in high-radix switches, new solutions must be provided. In this dissertation is presented a new technique for solving the HOL blocking problem, known as RECN-IQ.

High-radix technologies are present in new designs; as an example, a high-radix network is required in order to exploit the computing power of a system made of Merrimac stream processors [24]. Also new communication technologies like Proximity Communication (PxC) [25, 26] from Sun Microsystems are tied to high-radix architectural designs [27].

A switch architecture for high-radix switches was recently proposed in [23]. Basically, this solution relies on the use of a new crossbar organization. The internal data path is an intermediate solution between CIOQ and BC. This architecture is referred to as Hierarchical Crossbar (HC) and it is plotted in Figure 1.12. In particular, an $N \times N$ BC switch is substituted by $(N/p)^2$ smaller $p \times p$ sub-switches with the CIOQ memory organization. This organization requires $2p(N/p)^2$ memories. As p increases, the HC organization resembles the CIOQ thus becomes cheaper but less efficient. As p decreases, the HC organization resembles the BC organization, thus more efficient and more expensive.

Notice that the HC architecture implements internal speedup. Therefore, memories at the output links are needed. Additionally, the proposed switch architecture was intended for wormhole switching and flit-based arbiters which makes arbitration complex. As it will be seen later, a packet-based arbiter will be more efficient and simpler.



(a) Injected traffic versus accepted traffic

(b) Injected traffic versus packet latency

Figure 1.13: Performance plots for a perfect-shuffle MIN made of IQ switches connecting 64 nodes and random traffic. Saturation is reached at around 42 bytes/cycle. This is the peak throughput delivered by the network. Beyond the saturation point, the network throughput is lower and the latency very high.

1.5 Congestion Management

When the traffic injected on a network is below saturation, the network is able to deliver all injected packets. In Figure 1.13 there are two plots for a perfect-shuffle MIN made of IQ switches connecting 64 nodes and random traffic. On one hand it is plotted the injected traffic versus the accepted (delivered) traffic, and on the other the injected traffic versus packet latency is shown.

As can be seen on Figure 1.13, the good behavior of the network (i.e. when the network is able to deliver all injected packets) is broken when the saturation point is reached at 42 Bytes/cycle. Beyond this point, no matter how much traffic you generate, the limit for the network to deliver packets is reached. Moreover, the throughput is even lower than the peak in throughput achieved exactly at the saturation point. This region is known as the saturation region and this drop in performance must be avoided by all means.

Contention and Head-Of-Line (HOL) blocking are responsible of such performance drop after saturation. HOL blocking happens when a packet at the head of a FIFO queue blocks, preventing other packets at the same queue from advancing, even if they request free resources. This is indeed a problem for *lossless* networks in which packet discarding is not allowed (under normal operation), like in most of the modern high-speed networks. Moreover, the punctual congestion of even small parts of the network usually propagates and spreads all over the network very quickly.

The introduction of the wormhole switching technique [9] opened the possibility of implementing a switch in a single chip. That allowed such a

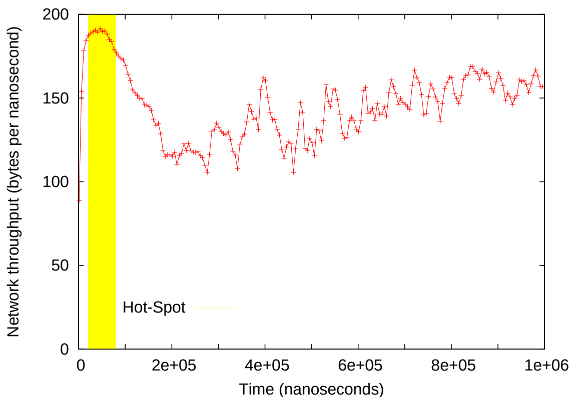


Figure 1.14: Punctual hot-spot injection over random traffic affects the throughput for a long period of time.

dramatic increase in link bandwidth, so interconnection networks could be overdimensioned at a low cost in the past. Therefore, they were working always far below saturation. But this is not the case today, when reducing the number of network components (links and switches) is a must in order to reduce power and cost (current VLSI technologies provide huge link speed increases, so interconnects are consuming an increasing part of the total system power [28]). This is especially true for high-radix networks, in which the optimal radix that minimizes latency according to the technology can lead to narrow channels forcing the network to work most of the time on the saturated region. It is worth to mention that the characteristic burstiness of traffic can lead to saturate temporarily at least local parts of the network, if not the whole.

Another example of how congestion affects performance can be seen on Figure 1.14. There, uniform traffic is being injected on a perfect-shuffle MIN of IQ switches connecting 256 nodes. At some point in time (yellow band on the plot), a *hot-spot* is injected. That means that for a small period of time ($6\mu s$), 10% of the traffic from each node is addressed to the same hot-spot (node number 2 in this very example). As can be seen, the injection of such small amount of hot-spot traffic for a short time produces that the network throughput is seriously harmed and takes long time to recover (more than $1000\mu s$).

It is important to mention that if the HOL blocking is completely removed, then the negative effects of congestion are eliminated as well. This can be seen on Figure 1.15, where a VOQ at the network level has been used. Because results in Figure 1.15 are for a MIN connecting 64 nodes (three stages made of 4×4 switches), this means that each input port has 64 different queues, one per each possible destination within the network. Figure 1.15 shows that by completely removing the HOL blocking, maximum

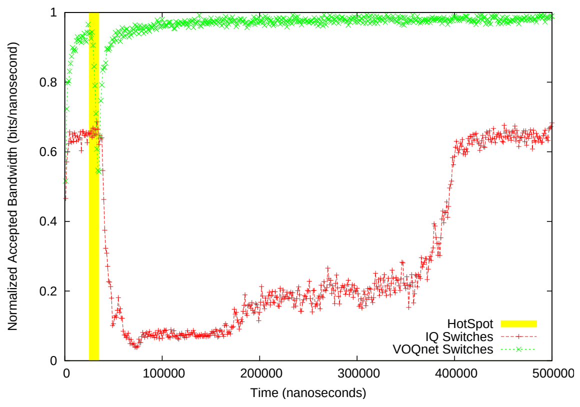


Figure 1.15: By using VOQ at the network level for completely removing HOL blocking, switch efficiency is maximum.

switch efficiency is achieved when a hot-spot traffic is applied; a hot-spot in which uniform traffic is injected at maximum rate, and for just $10\mu s$, 50% of the traffic injected by each node is addressed to the same hot-spot destination. On the other side, observe the great plunge on throughput produced by the HOL blocking on the IQ switches case (Figure 1.15).

The risk of congestion in interconnection networks is a well-known problem, and many strategies have been proposed to deal with it. The simplest of those strategies are the network overdimensioning and the packet dropping under congested situations. However, none of them are suitable for modern interconnection networks due, respectively, to the high cost and power consumption of current network components; and to the lossless character of these networks.

Other more elaborated techniques have been specifically proposed for avoiding or eliminating congestion. For instance, proactive strategies are based on reserving network resources for each data transmission, requiring a traffic scheduling based on network status [29]. However, this status information is not always available, and the resource reservation procedure introduces significant overhead. On the other hand, reactive congestion management is based on notifying congestion to the sources contributing to its formation, in order to cease or reduce the traffic injection from those sources [30]. Unfortunately, these solutions are not quite efficient due to the delay between congestion detection and notification.

Other strategies focus on eliminating the actual negative effect of congestion: The HOL blocking. In that sense, many HOL blocking elimination strategies have been proposed: Virtual Output Queues (VOQs) [31], Dynamically Allocated Multiqueues (DAMQs) [32], congestion buffers [33], etc. Most of these techniques rely on allocating different buffers for storing separately packets belonging to different flows.

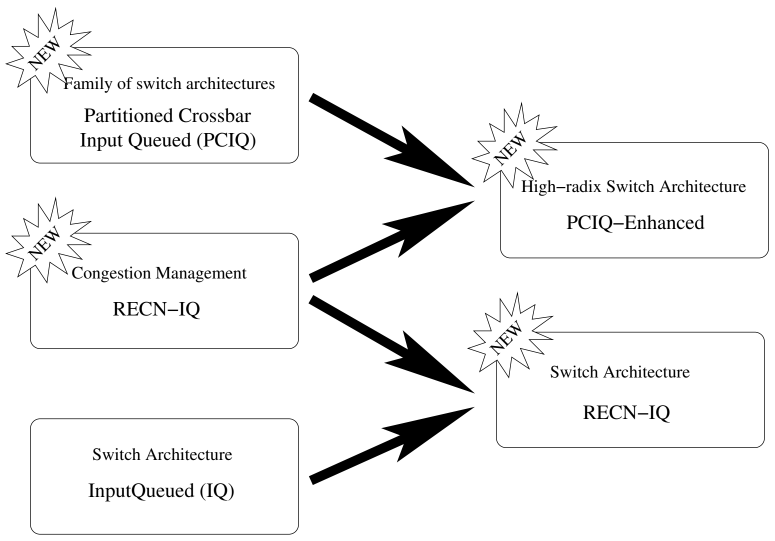


Figure 1.16: Schematic overview of the contributions of this dissertation.

In general, traditional HOL blocking elimination techniques are either feasible or effective, but not feasible and effective at the same time. For instance, the use of VOQs at network level requires as many queues at each port as end-points in the network, being so an effective but not scalable technique. A variation of VOQ uses as many queues at each port as output ports in a switch [13], so this technique (known also as VOQsw or VOQ at the switch level) is feasible, but it does not eliminate HOL blocking at the network level.

In contrast to these techniques, another solution known as "Regional Explicit Congestion Notification" (RECN) [34, 35] eliminates the HOL blocking in an efficient and scalable way. In order to achieve this, RECN identifies congested packets and stores them in special, dynamically-assigned set aside queues (SAQs). RECN was the first truly efficient and scalable HOL blocking elimination technique and it is explained in full detail on Section 3.2.

1.6 Contributions

In this dissertation we tackle the inherent problems in the design of high-radix switches; also, in any switch design one of the main constraints is the memory requirements. Therefore, a new family of switch microarchitectures known as *Partitioned Crossbar Input Queued* (PCIQ) is proposed. It relies on a smart partition of the crossbar into sub-crossbars, thus requiring less memory resources than other proposals for high-radix, yet obtaining high-performance.

The other big issue on high-radix switches is the HOL blocking problem, which reduces dramatically the switch performance. Traditional solutions for solving that problem were based on VOQ schemes, but having high number of ports on a high-radix switch prevents the use of any of them. In this dissertation, a new congestion management technique is proposed. This solution, known as RECN-IQ, is specific for IQ switches and differs from the original RECN idea in being highly efficient and simple to implement, reducing the extra memory requirements (even by removing the necessity of having queues at the outputs on the original RECN which was intended only for CIOQ switches). This memory reduction relies on the fact that RECN-IQ uses a novel statistical approach for detecting congestion instead of using expensive detection queues from the original RECN.

By combining the PCIQ microarchitecture with RECN-IQ (the new congestion management mechanism proposed here), a new switch architecture (called here PCIQ-enhanced) is derived and deeply evaluated in this dissertation. The PCIQ switch architecture inherits the benefits of the Partitioned Crossbar microarchitecture in reducing the memory requirements for high-radix designs with the power of a congestion management technique that removes the HOL blocking dynamically, thus achieving maximum switch performance under all types of traffic.

Because the RECN-IQ mechanism is so powerful in removing the HOL blocking and it can be applied to almost any IQ switch microarchitecture; we applied it to a basic IQ switch so we came up with a cheap, yet highly efficient, switch architecture known (just like the congestion management mechanism itself) as RECN-IQ switch architecture. We detail the microarchitecture and implementation details of such proposal and also evaluate its performance, showing the fact that by adding a little extra hardware for supporting RECN-IQ, a simple IQ switch can perform like (and even better than) more expensive solutions with several queues per input port or more complex allocators.

To sum up, the four main contributions on this dissertation (also depicted on Figure 1.16) are the following:

- Proposal of a new family of switch micro-architectures suitable for high-radix switches known as Partitioned Crossbar Input Queued (PCIQ).
- Proposal of a highly efficient congestion management technique known as RECN-IQ suitable for switch architectures with buffers only at the inputs.
- The combination of the previous two in order to come up with a high-performance switch architecture known as PCIQ with RECN-IQ or simply PCIQ-Enhanced.
- The congestion management technique allows us to derive another high-performance low-cost switch architecture known as RECN-IQ from a simple IQ switch architecture.

Chapter 2

The PCIQ Switch Architecture

Crazy theories one, regular theories a billion.
– Fry, "Futurama"

The interconnection network plays a key role in the overall performance achieved by high performance computing systems, also contributing an increasing fraction of its cost and power consumption, as we have seen in the previous chapter. Current trends in interconnection network technology suggest that high-radix switches will be the preferred option; so networks become smaller (in terms of switch count), and with the associated savings in packet latency, cost, and power consumption. Unfortunately, current switch architectures have scalability problems that prevent them from being effective when implemented with a high number of ports.

In this chapter, an efficient and cost-effective architecture for high-radix switches is proposed. The architecture, referred to as Partitioned Crossbar Input Queued (PCIQ), relies on three key components: a partitioned crossbar organization that allows the use of simple arbiters and crossbars, a packet-based arbiter, and a mechanism to eliminate the switch-level HOL blocking.

2.1 Introduction

Designing high-radix switches presents major challenges. The most important one is to keep a high switch efficiency with an affordable cost. The cost of a high-radix switch will largely depend on three key components: memory resources, arbiter logic and internal connection logic. Depending on the location of memories within the switch, different switch organizations (memory and crossbar capabilities and their interconnects) have been used. In some of them, the number of memories increases quadratically with the number of ports. Also, arbiters and crossbars must cope with more candidates and connections, and for that reason become expensive.

From another point of view, switch efficiency will be largely impacted by the head-of-line (HOL) blocking experienced within the switch. The HOL blocking problem appears when a packet at the head of a queue is blocked (because it is requesting an occupied resource), thus preventing packets in the same queue to make forward progress even if their requested resources are free. Dealing with HOL blocking requires more resources (queues and internal speedups), thus making the switch more expensive.

As seen on Chapter 1, traditional switch organizations are not suitable for implementing high-radix switches: IQ organization because of their low performance, and the rest because of their prohibitive cost in terms of memory resources (BC) and/or required internal speedups (OQ, CIOQ).

In the case of the HC architecture (Section 1.4), because it implements internal speedup, memories at the output links are needed. As mentioned before, HC was intended for wormhole switching and flit-based arbiters which makes arbitration complex. It is fair to say that by properly selecting parameter p, the HC organization may achieve a good trade-off among cost and efficiency. However, from our point of view, all the proposed organizations (including HC) still suffer from the same problem: they do not provide a cost-effective solution to the HOL blocking problem. Moreover, even the cheapest version of HC has a cost significantly higher than a CIOQ switch.

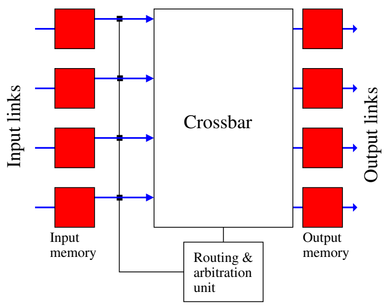
In this chapter, we propose a new switch architecture referred to as Partitioned Crossbar Input Queued (PCIQ). The goal of the PCIQ architecture is to achieve a high switch efficiency, a good scalability, and virtually eliminate HOL blocking while keeping cost as low as possible. These features make the proposed architecture suitable for implementing high-radix switches.

The PCIQ organization has been designed starting from a low-cost and simple switch organization (CIOQ with no speedup). This architecture has been improved by carefully selecting the different design choices: crossbar organization, memory organization, arbiters, switching mechanisms, and queue mapping policies, in order to enable the maximum switch efficiency with a low cost and allowing high switch radix.

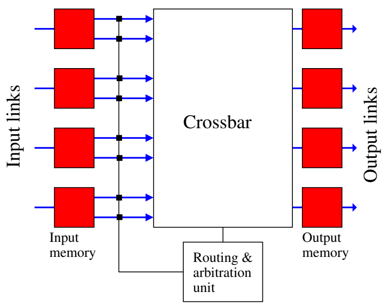
There are two main contributions in this chapter:

- The proposal of a new switch architecture with better cost/performance trade-off than any previously proposed architecture suitable for high-radix switches.

- A detailed performance evaluation showing that the proposed switch architecture achieves efficiency very close to 100% of the one achievable by an ideal architecture for different traffic patterns.



(a) CIOQ



(b) CIOQ with two read ports at the inputs

Figure 2.1: CIOQ is the starting point for deriving PCIQ.

2.2 Description of PCIQ

The PCIQ switch architecture will be deployed starting from the CIOQ switch organization with no internal speedup (see Figure 2.1.(a)). The efficiency of a CIOQ switch can be enhanced by increasing the read bandwidth of the input ports (Figure 2.1.(b)). This idea is not new by itself. Indeed, it was proposed long ago [36] in combination with a more complex crossbar and arbiter. In what follows, without loss of generality, we will describe our proposal assuming that the read bandwidth is doubled.

Basically, there are three ways to double the SRAM read bandwidth. The first one consists of increasing either clock frequency or word size. Both of these options imply a significantly higher cost or have undesirable side effects, so, we discard them. The second one consists of implementing two independent read ports (dual-ported SRAM). This solution may significantly increase silicon area, almost doubling in some cases. This is mostly due to the duplication of the internal wires from the cells to the ports. However, that increase can be significantly reduced by using a full-custom design [37] or a Hierarchical Multiport Architecture (HMA) [38]. As an example, in [39] a SRAM cache memory has been implemented with 4 ports with an area overhead of 25% (there is another example with low area overhead in [40]). The third way to double SRAM bandwidth is achieved by splitting the SRAM into two independent SRAMs. This solution doubles silicon area requirements but it also doubles SRAM size. However, it has two drawbacks to consider. First, it requires new logic selecting the SRAM where an incoming packet has to be stored. Second, it complicates flow control since separate credit counters are required for each memory. Therefore, a dual-port SRAM will be the preferred solution.

It should be noted that the three solutions to double SRAM read bandwidth, when used on PCIQ switch architecture, lead to the same architectural configuration: using no output speedup while using an input speedup

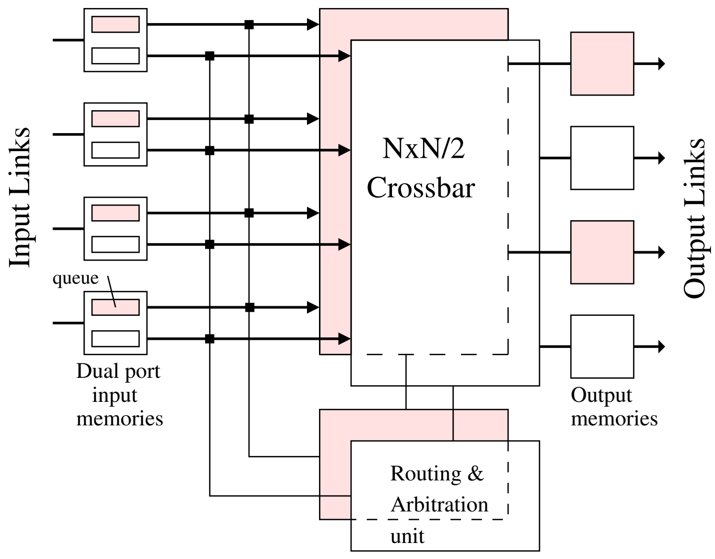


Figure 2.2: PC switch organization, the previous step towards PCIQ.

of 2. But they differ in their implementation cost.

In the switch organization proposed in [36], the number of input ports of the internal crossbar is doubled, making arbitration more complex. Contrary to this solution, the PCIQ switch organization increases the SRAM read bandwidth while keeping the arbiter and crossbar cost constant. To achieve this, the PCIQ switch architecture is based on three key mechanisms: a partitioned switch organization; a partitioned arbiter design; and the use, for the enhanced version of PCIQ, of a mechanism to remove internal HOL blocking (known as RECN-IQ and described on Chapter 4).

2.2.1 PC Crossbar Organization

In the PCIQ switch architecture, the original crossbar used in CIOQ with increased read bandwidth is split into two separate crossbars (referred to as *Partitioned Crossbar*; PC), as shown in Figure 2.2. Each read port of the input memory RAMs is attached to a different crossbar, thus providing concurrent access from each memory RAM to both crossbars. On the other hand, each crossbar only provides access to a subset of the output links, thus reducing cost while keeping connectivity. For instance, the first crossbar could connect to odd output links and the other crossbar to even ones. In this design, each crossbar has a size of $N \times N/2$, thus they are called asymmetrical (different number of inputs than outputs), in contrast with an $N \times N$ symmetrical crossbar.

Although this partition reduces flexibility a little bit, it allows significant simplifications in the internal crossbar, thus reducing cost and latency.

Effectively, instead of having a single $2N \times N$ crossbar, now we have two independent $N \times N/2$ crossbars. In this way, the total crossbar complexity is the same as for the basic CIOQ. Latency is also reduced with respect to using a $2N \times N$ crossbar because the crossbar latency grows logarithmically with the number of ports [41].

Notice that, in the PC organization (Figure 2.2), the queues implemented in each input memory are split into two sets, one for packets addressed to odd links and another one for packets addressed to even links.

2.2.2 Routing and Flow Control

When a packet arrives at a switch, it must be stored in the correct queue (odd/even). Since a single memory is used to implement all queues (odd and even ones) a mechanism is required to map the packet to the correct queue (odd or even). An effective solution is to map an incoming packet to a preallocated empty slot (not assigned to any queue). Upon reception of the packet's header, routing is performed while the rest of the packet is being received. Once the output port is known, the control pointers that implement the queues are updated accordingly and the packet is appended to the corresponding queue (odd or even).

On the other hand, both odd and even queues must implement flow control separately. An efficient way of doing this is by implementing credit-based flow control at the memory level combined with dynamic queue allocation and sizing, and Xon/Xoff flow control at the queue level [34] (refer to Figure 1.6 in Section 1.1 for an explanation of the credit based and Xon/Xoff flow control mechanisms). This way, we just need a single credit counter to track the availability of space in the corresponding input memory of the next switch, thus reducing cost. Additionally, dynamic queue sizing allows us to effectively implement several queues without having to implement the landing pad (i.e. the buffer area required to maximize link utilization, which is proportional to the round-trip time) in every queue. Finally, Xon/Xoff prevents a single queue from monopolizing the entire memory space while requiring only one bit per queue at the upstream switch to keep its status.

It should be noted that switch organizations like HC or the use of VOQs require the computation of the output port for the current switch at the upstream switch in order to flow control the different memories (HC) or queues (VOQs). Similarly, packets have to be prerouted at the upstream switch in our design so that packets are not transmitted to a queue that sent an Xoff. For interconnects based on source routing, like Myrinet [4] or Advanced Switching [42], this can be done simply by inspecting a few bits from the packet header.

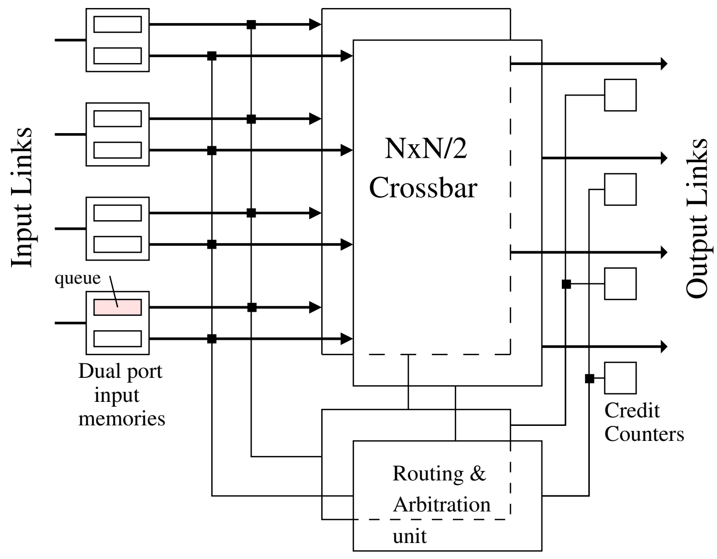


Figure 2.3: PCIQ switch organization.

2.2.3 PCIQ: Removing the Output Memories from PC

The PCIQ architecture does not require memories at the output links. Effectively, although two read ports are implemented in each input memory, they are used to forward packets to different sets of output links. Thus, output memories receive data at the same rate they should forward those data through the link. This fact allows us to remove the output memories (see Figure 2.3), thus sending packets through a crossbar and directly to the corresponding link. As a consequence, the extra area required for implementing the additional read ports in the input memories is compensated by the removal of output memories, thus exhibiting a cost similar to that of a conventional switch with memories both at the input and output links (CIOQ).

As can be seen, with such crossbar partition, PCIQ has an input speedup of 2 and no output speedup at all.

2.2.4 Arbiter

The second key component of the PCIQ switch architecture is the arbiter. Two identical arbiters are required, one per crossbar. Each arbiter matches candidates and output links for the corresponding crossbar. Thus, each arbiter is associated with one read port from each input memory. Each one is implemented as a hierarchical round-robin arbiter which is a common solution in commercial products.

The arbiter used in PCIQ is shown in Figure 2.4, this scheme allows to distribute the output arbiters across the router so wiring complexity is

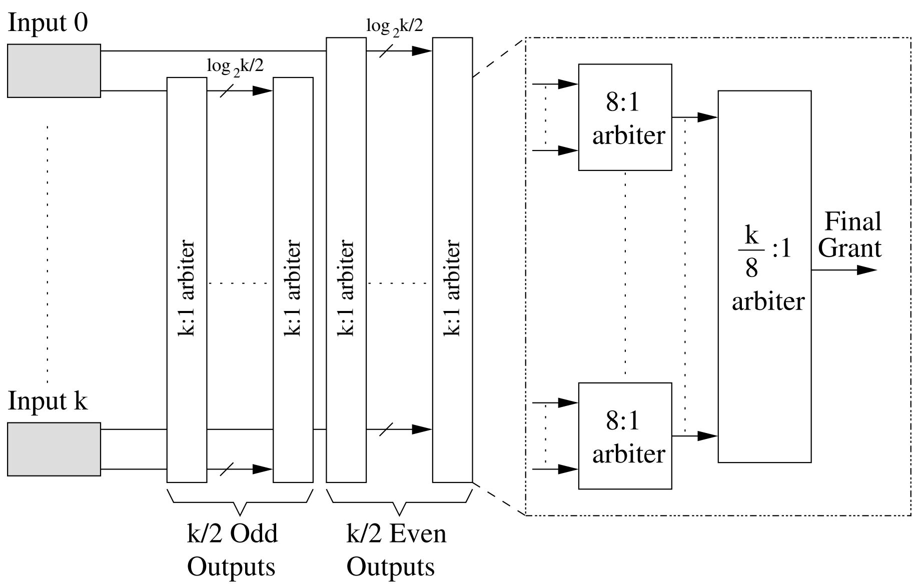


Figure 2.4: Hierarchical arbiters used in PCIQ. In this example, each output arbiter uses $8 : 1$ smaller sub-arbiters and a final $k/8 : 1$ sub-arbiter (k indicates the radix of the switch).

limited.

Another detail is that both arbiters (for odd and even links) work on a per-packet basis. That is, whenever an arbiter matches a request to a free output link, the entire packet will be forwarded. Therefore, arbiters are designed to work asynchronously, so that whenever a new packet reaches the head of the queue, or whenever an output link becomes free, the arbiter will be waken up.

Compared to a flit-based arbiter, there exist several benefits from using a packet-based arbiter in the PCIQ architecture. The first one is that the arbiter will be less frequently used (once per packet arrival or an end of crossbar connection versus once per flit), therefore consuming less power. Moreover, a flit-based arbiter must arbitrate among all the inputs and all the outputs every cycle. Unless a very simple scheme is used, the arbiter needs to be pipelined. But even more important, the efficiency of a packet-based arbiter increases with traffic load, while a flit arbiter have to match all the input-outputs pairs from scratch at every arbitration cycle.

When compared to an $N \times N$ arbiter, two $N \times N/2$ packet-based arbiters exhibit higher efficiency, as will be proved in Section 2.5. In an $N \times N$ arbiter, the ratio between the number of requests and the number of available resources is one regardless of the number of already established connections in the crossbar. However, in an $N \times N/2$ arbiter, every time a new connection is made, the ratio between the number of requests and the number of resources

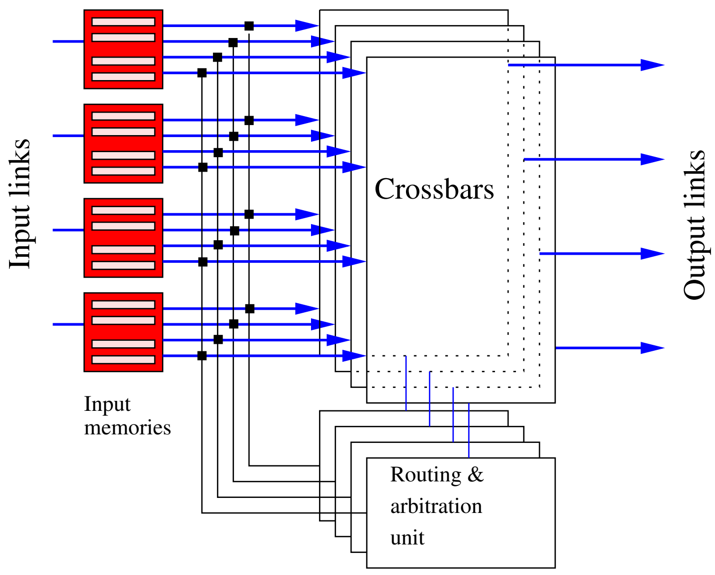


Figure 2.5: PCIQ switch organization with 4 subcrossbars.

improves, thus arbitration efficiency improves as well. This fact is detailed in Section 2.4.

Each arbitration level of the arbiter hierarchy could be executed in one clock cycle since simple round-robin arbiters are used. However, supposing that we initially designed the hierarchical arbiter divided into two levels, if the number of input queues per read port and/or the radix of the switch are high, the two levels may require several cycles. The number of required cycles grows logarithmically with the number of candidates. As an example, an $N \times 1$ round-robin arbiter model is proposed and analyzed in [43]. The arbiter is implemented with a hierarchical tree of smaller arbiters. In [43] it is shown that, as the switch radix increases, the proposed arbiter exhibits a logarithmic longest delay response. The arbiter proposed in [43] is valid for implementing all the arbiters required at both levels.

2.3 PCIQ as a Family of Switch Architectures

The partitioned switch organization can be further elaborated by partitioning the initial crossbar in more than two subcrossbars, for instance using K crossbars. As an example, an initial $N \times N$ crossbar can be substituted by four $N \times N/4$ subcrossbars where each one forwards packets to $N/4$ disjoint output links. This solution requires the use of memories with four read ports at the input, with the associated extra cost, but it also simplifies arbitration by splitting the $N \times N$ arbiter into four independent $N \times N/4$ arbiters. Figure 2.5 depicts such configuration of PCIQ with 4 subcrossbars.

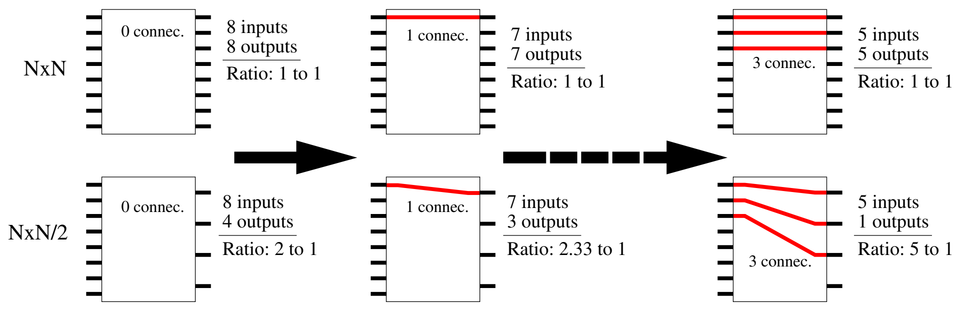


Figure 2.6: Example of scheduling efficiency improvement by using asymmetrical crossbars versus using symmetrical ones.

Therefore, the PCIQ architecture forms a new family of switch architectures just by choosing the appropriate number K of subcrossbars that fills the gap between the CIOQ and BC architectures, and thus, it should deliver a better trade-off between switch efficiency and cost depending on the selected K value for the number of crossbars in PCIQ.

2.4 Impact on Scheduling Efficiency When Asymmetrical Crossbars Are Used

Note that if we compare two schedulers, one symmetrical (i.e. $N \times N$) and the other asymmetrical (like the ones used in PCIQ, i.e. $N \times N/2$); then the asymmetrical scheduler will show higher efficiency. The reason behind this can be understood looking at Figure 2.6. A symmetrical scheduler will retain the same relationship between the number of input candidates to match and the possible (non-assigned) requested outputs no matter how many connections are already made; that relationship will be always 1:1 for an $N \times N$ scheduler.

On the other hand, as depicted on Figure 2.6, when an asymmetrical scheduler is used (as in PCIQ), each time a new connection is established (a pair of input-output is matched), then the possibility of establishing a new connection is higher because we will have higher number of candidates for less resources.

This intuitive view on the increase on scheduling efficiency for asymmetrical crossbar is what the theoretical model presented on Section 2.5 exposes.

2.5 Model for Asymmetric Crossbars

Because we are proposing the use of asymmetrical crossbars, in this section we develop a straightforward model that theoretically evaluates the PCIQ switch architecture. The model will only consider the structure of an asymmetric crossbar (the number of inputs differs from the number of outputs).

For the sake of understanding, the model is described in four steps shown in Figure 2.7. At step 1 (Figure 2.7.(a)), the model assumes that there are I inputs that compete for O outputs, and that all inputs have an equal probability to be requesting any output. The probability $H(I,O)$ that there is no input having a cell at the head of its queue destined to any given output is:

$$H(I,O) = \left(\frac{O-1}{O}\right)^I \tag{2.1}$$

It can relatively easily be verified that $1 - H(I,O)$ approaches $0,632$ if $I = O$ (symmetric crossbars) and O is large, which is in accordance to the results obtained in [12] (for crossbars that drop blocked packets).

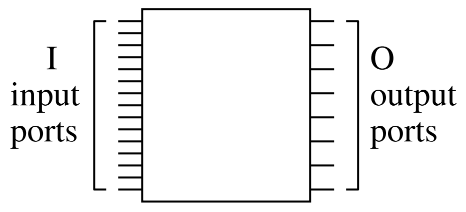
At step 2 (Figure 2.7.(b)), the model assumes σ to be the utilization of the outputs. Based on σ at any timestep t, the inputs request a subset σO of the outputs. Then, at step 3 (Figure 2.7.(c)), the number of inputs served at time t will be σO (which is identical to the number of outputs that will be served) and the number of inputs not served will be $I - \sigma O$.

The model now computes, at step 4 (Figure 2.7.(d)), the utilization of the crossbar at the next timestep $(t+1)$ from the configuration obtained at the previous timestep (t). In particular, the outputs not previously served $(I - \sigma O)$ will be requested now only by the inputs previously served (σO). Thus, they will have a probability of being served at timestep $t+1$ equal to $1 - H(\sigma O, O)$.

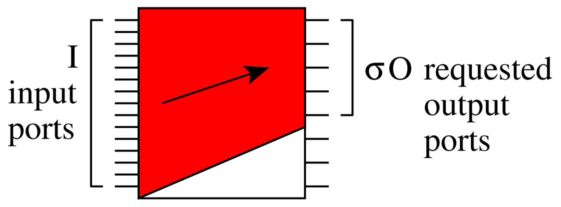
In the same way, the outputs previously served (σO) will now be requested by all the inputs (I). However, depending on the inputs, the probability will differ. The previously served outputs will have a probability of $H(\sigma O, O)$ of not being requested by the inputs that were previously served (σO), and a probability of $H(I - \sigma O, \sigma O)$ of not being requested by the input ports that were not previously served $(I - \sigma O)$ [1]. Therefore, the probability of an output previously served of being served at timestep $t+1$ is:

$$1 - H(\sigma O, O) \cdot H(I - \sigma O, \sigma O) \tag{2.2}$$

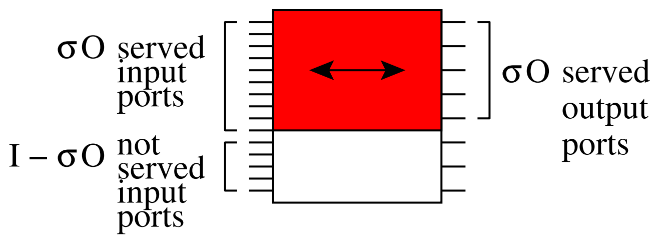
[1]At that point we choose to make the simplifying assumption that the inputs that were blocked at timestep t have an equal probability of requesting access to any of the outputs that were served, so the probability that there isn't any previously blocked input that will request any given previously served output is modeled to be $H(I - \sigma O, \sigma O)$. This is where we trade off accuracy. In fact the history of blockings that the inputs may have gone through will make the requests lump together their requests on fewer outputs. The reason is that the inputs that have been blocked twice in a row will compete only for the outputs that have been served twice in a row. Therefore the probability of a previously served output being requested by a previously blocked input is slightly overestimated in our model.



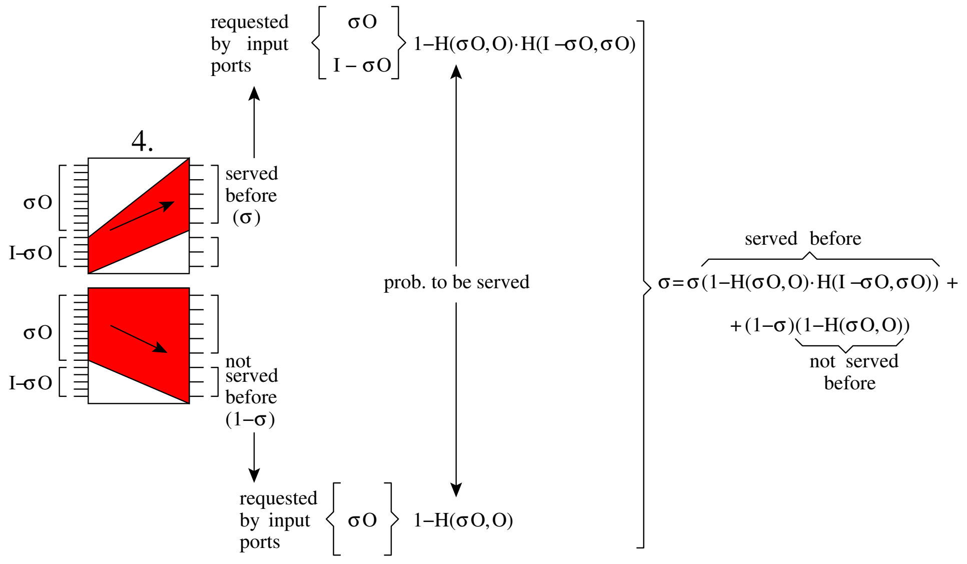
(a) Step 1



(b) Step 2



(c) Step 3



(d) Step 4

Figure 2.7: Steps to describe the model for asymmetric crossbars.

The final observation we need is the fact that each output will spend a fraction σ of its time being in a situation where it was served in the previous timestep, and a fraction $1 - \sigma$ of its time in a situation where it was not served. Simple calculation now gives us, as shown at the end of step 4, the utilization of a crossbar with I inputs and O outputs, that is:

$$U(I,O) = \sigma = (1-\sigma)(1-H(\sigma O,O))+\sigma(1-H(\sigma O,O)H(I-\sigma O,\sigma O)) \quad (2.3)$$

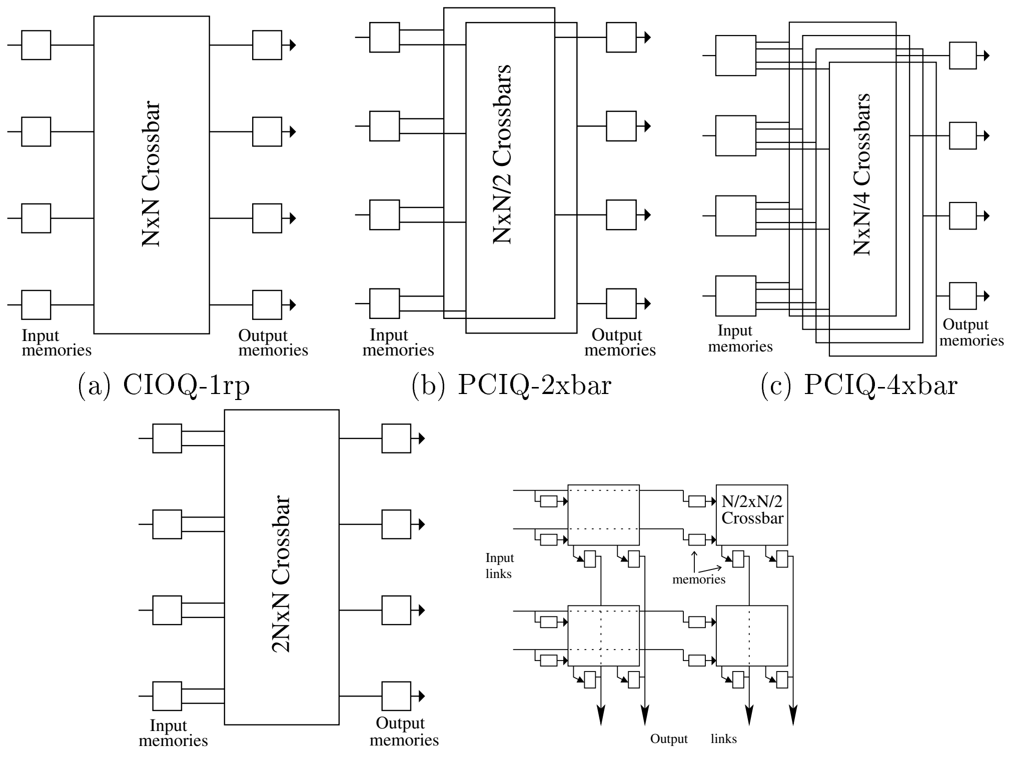
In order to solve this equation for various values of I and O we have used a simple numeric algorithm that approximates the right value through a binary search.

2.6 Evaluation with the Theoretical Model

In this Section we apply the previous model to evaluate different switch architectures. In particular, the model will allow us to evaluate architectures with different number of either asymmetric or symmetric crossbars connected to inputs and outputs in different ways. To do this we consider the following assumptions:

- The switch has infinite input queues.
- Each input follows a random and uniform address distribution.
- Each switch input port will be able to deliver one cell into each crossbar to which it is connected to at each cycle.
- Each switch output port will be able to receive one cell from each crossbar to which it is connected to at each cycle.
- Each cycle, a cell transfer takes place to a switch output link if an only if there is at least one switch input link that requests access to that output.
- When a switch input (or output) link is attached (through different read or write ports) to multiple crossbars, it is able to deliver (or accept) fast enough for this not to be a bottleneck.

The model is applied to each switch architecture showed at Figure 2.8 by estimating the number of cells destined for each output port that the structure of crossbars is able to accommodate. In particular, for the PCIQ architecture with K $N \times N/K$ crossbars, each switch output port is served by one crossbar, thus the expected throughput is $U(N, N/K)$. Similarly, the model can also be applied to CIOQ and HC architectures. The expected throughput of CIOQ is $U(N,N)$ and $2U(N/2,N/2)$ for HC (each output port is attached to two $N/2 \times N/2$ crossbars).



(a) CIOQ-1rp (b) PCIQ-2xbar (c) PCIQ-4xbar

(d) CIOQ-2rp (e) HC

Figure 2.8: Different switch organizations analyzed with the theoretical model. The organizations are the following: (a) CIOQ with a single read port per input; (b) PCIQ with two subcrossbars; (c) PCIQ with 4 subcrossbars; (d) CIOQ with two read ports per input; and (e) Hierarchical Crossbar with $p = 2$.

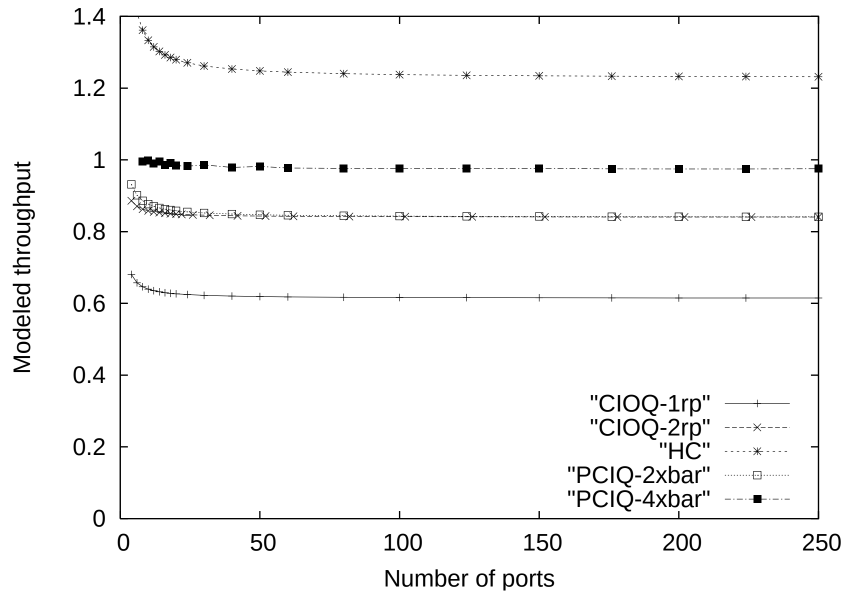


Figure 2.9: Throughput values given by our theoretical model.

Figure 2.9 shows the throughput (measured in cells/port/cycle) obtained by the model for each architecture of Figure 2.8 for different number of switch ports. All of the designs outperform the standard simple crossbar (CIOQ with one read port) completely. What is particularly interesting is that PCIQ with four crossbars (PCIQ-4xbars), reaches near maximum switch throughput (0.96), and thus, almost completely removes the drawback of standard input buffered switches with simple scheduling. PCIQ with two crossbars gives also a high throughput of 0.84. It achieves the same throughput than CIOQ with two read ports per memory (CIOQ-2rp in the Figure 2.9), however, without the need for a complex arbiter and crossbar. Another interesting result is that HC architecture achieves a throughput higher than one (is able to move 1.23 cells per cycle per port). This is due to the internal speedup implemented in HC. However, since switch bandwidth will be limited by link bandwidth, throughput values larger than one are not practical. It has to be noted that PCIQ architecture achieves maximum switch throughput (with four crossbars) without implementing internal speedup.

2.7 Evaluation of PCIQ Through Simulation

In this section we evaluate the performance of the PCIQ switch architecture. For this, a 24-port switch with different organizations is evaluated. The selected organizations are CIOQ, HC, and the proposed PCIQ with two and four subcrossbars. For HC, four 12×12 sub-switches are used ($p = 12$).

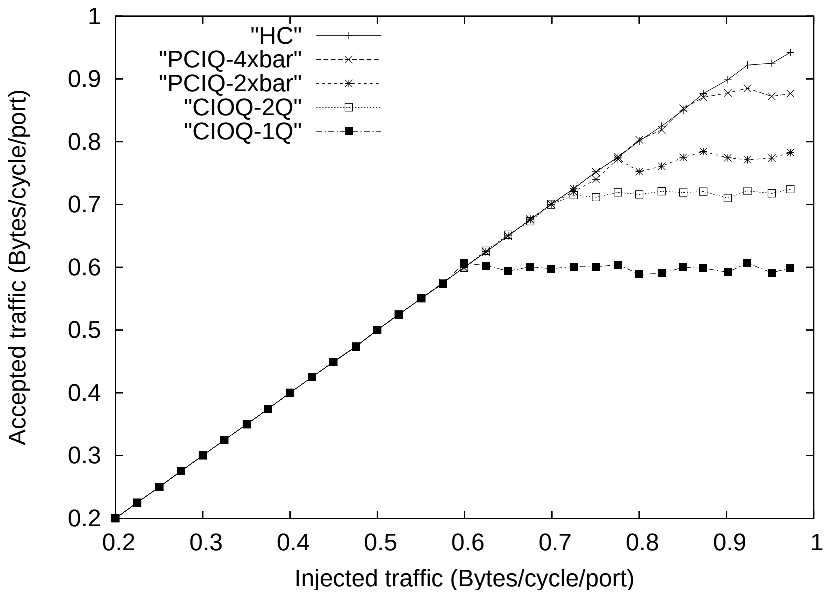


Figure 2.10: Switch efficiency of CIOQ, HC, and PCIQ. Uniform distribution of packet destinations.

As long as PCIQ switch architecture uses two separated queues to forward packets (one for even outputs and the other for odd outputs), it is fair to compare it with CIOQ with two queues (2Q).

Our simulator works at the clock cycle level. Each link of the switch is attached to an end node. Each end node injects traffic at the maximum link rate (one byte every cycle). The switch also forwards a byte from an input to an output in one cycle. Arbitration is assumed to take two cycles. Virtual cut-through switching is modeled in the simulator. Also, credit-based flow control for memories and Xon/Xoff flow control for queues are used for the three switch organizations. Therefore, the only difference between the evaluated switch architectures is the internal datapath and the associated arbiter partitioning.

First, uniform traffic with 256 bytes packet size is analyzed. As Figure 2.10 shows CIOQ with 1Q achieves the expected performance of 58% switch efficiency (as proven in [12]). This is due to the combination of the limited internal bandwidth (just one read port per input memory) and significant HOL blocking. For PCIQ with two subcrossbars, the switch efficiency is close to 80%. For the PCIQ with four subcrossbars, efficiency is higher (close to 90%).

It is observed that CIOQ with 2Q shows higher performance (72%) than the one queue case (58%). Having two separate and independent queues helps in reducing the impact of the HOL blocking within the switch. When

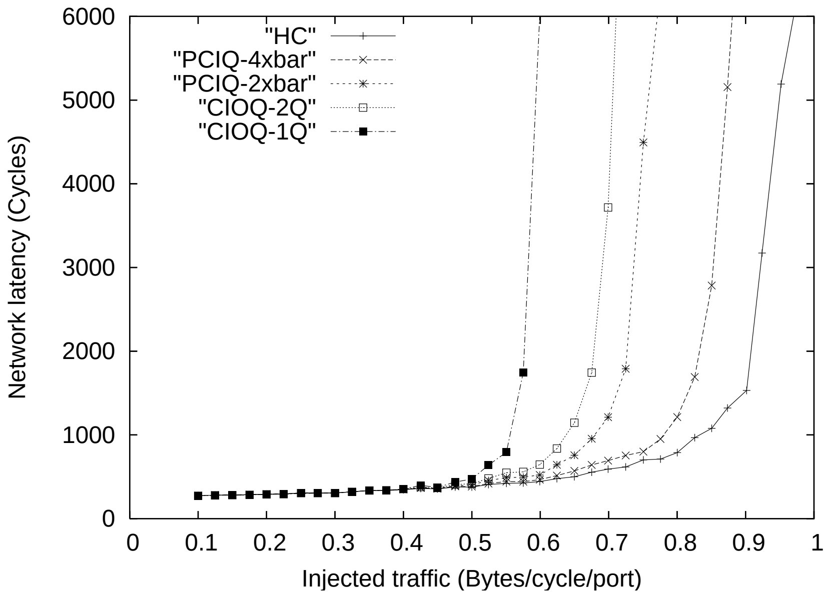


Figure 2.11: Network latency of CIOQ, HC, and PCIQ. Uniform distribution of packet destinations.

a packet is blocked at the head of one queue (causing HOL blocking to the rest of the packets in the queue), the remaining queue may provide (in subsequent arbitration cycles) good candidates.

On the other hand, HC efficiency is close to 100% (even for the largest p value). This remarkable result is achieved because HC implements by design internal speedup, that is, several writes to an output link at the same time are possible. Additionally, internal HOL blocking is also alleviated by the use of different memories at the input ports for different sets of output ports. As a consequence, for uniform traffic, the switch efficiency achieved by HC is limited only by the link bandwidth. However, the additional performance achieved by HC over PCIQ (approximately 10%) comes at the expense of a much higher implementation cost, as will be detailed in Section 2.10.

Results for network latency are also shown in Figure 2.11. In one switch network, the main contribution to latency is the HOL blocking problem experienced in the queues by the packets crossing the network.

Although the PCIQ switch architecture achieves very good levels of efficiency for uniform traffic, this scenario will not be the only case on a production system. Figure 2.12 shows switch throughput when hot-spot traffic is injected into the 24-port switch. In this traffic scenario, all the nodes inject 90% of their traffic to random destinations. The remaining 10% of traffic is injected to a hot-spot (node 6). All the nodes inject at the full injection rate. As the output link 6 will be overloaded, massive HOL blocking within

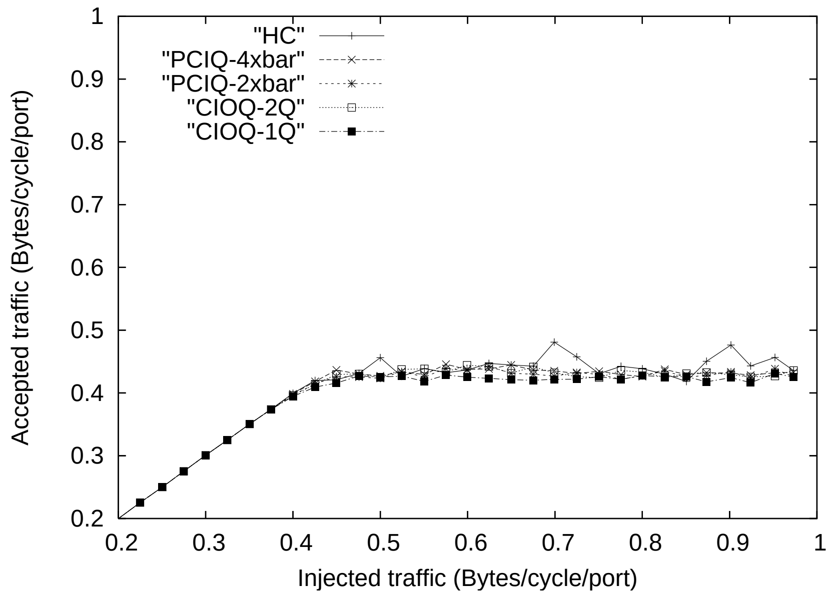


Figure 2.12: Switch efficiency of CIOQ, HC, and PCIQ. Hot-spot plus uniform distribution of packet destinations.

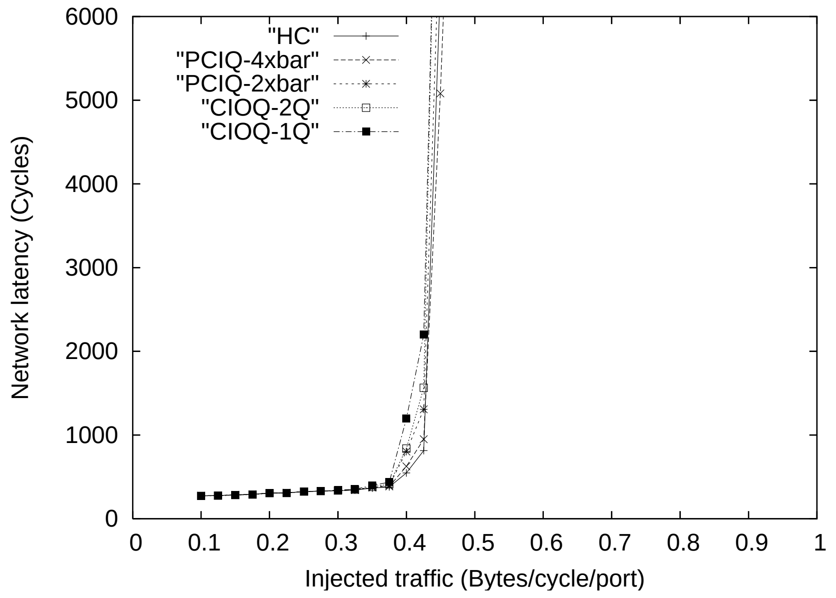


Figure 2.13: Network latency of CIOQ, HC, and PCIQ. Hot-spot plus uniform distribution of packet destinations.

the switch will be produced. Note that for this traffic pattern, switch efficiency is bounded by $90\% + 1/24 \times 10\% = 90.4\%$ regardless of the switch architecture.

As can be seen, all switch organizations suffer from HOL blocking. All of them achieve the same switch efficiency (around 45%). If the HOL blocking were addressed the switch would deliver roughly 90.4% of maximum switch efficiency.

From the results obtained in this section, two issues arise. First, the base PCIQ architecture with two subcrossbars requires new improvements in order to achieve maximum switch efficiency with uniform traffic. More precisely, the internal HOL blocking must be eliminated. Second, the switch should achieve maximum switch efficiency regardless of the injected traffic. Thus, hot-spot situations like the one analyzed must be handled correctly.

2.8 Enhancing PCIQ by Adding RECN-IQ

The HOL blocking experienced at some input queues prevents the lower-cost organizations (CIOQ and PCIQ with two subcrossbars) from achieving the maximum performance. In this section, we will incorporate a new mechanism in order to increase their efficiencies.

There are several ways to reduce HOL blocking. The easiest way is by using a number of queues at the input memories equal to the number of switch output ports and mapping packets to the queues depending on the requested output links. This is known as Virtual Output Queuing (VOQ) at the switch level. However, the queue requirements grow quadratically with the number of ports, thus becoming infeasible for high-radix switches.

Notice also that network-wide HOL blocking is not solved by using VOQ at the switch level. Such blocking occurs among packets that share along their paths one output link at the same switch. A packet blocks at the head of a queue because the requested next queue at the next switch is full (congested). As a consequence, other packets behind the blocked one, also block even if they would request a queue with space at the next switch.

Congestion trees and their dynamics [44] have proven to introduce massive HOL blocking, thus collapsing the network. One solution to the network-wide HOL blocking would be to have as many queues as final destinations on every input memory in all the switches (VOQ at the network level). Unfortunately, the cost of this solution is clearly prohibitive.

Another solution to increase switch efficiency is by adding internal speedup (as HC switches do). However, this significantly increases the switch cost and does not solve network-wide HOL blocking. Also, using four subcrossbars in the PCIQ switch organization reduces internal HOL blocking thus reporting higher efficiencies, however, notice that again network-wide HOL blocking is not solved.

A specific mechanism for congestion control (known as RECN) has been recently proposed [34]. With this mechanism, network-wide HOL blocking is removed by dynamically allocating queues to store packets going through the congested points in the network. Congestion spots within the network are detected, notified to the affected switches and Set Aside Queues (SAQs) are dynamically allocated. Packets traveling along the congested points are mapped to the SAQs and, therefore, they do not introduce HOL blocking.

To detect congestion, RECN implemented detection queues at the input links. There were as many detection queues as output ports in the switch, thus this comes with the same cost as VOQ at the switch level. However, we have developed in this dissertation a completely new version of RECN that no longer requires detection queues. Instead, the packet at the head of a congested queue is assumed to be responsible for congestion (new statistical detection mechanism), therefore only the extra queues for congested flows are required. This new congestion management technique, called RECN-IQ, is described in full detail in Chapter 3. With this improvement, the RECN-IQ mechanism handles the switch-level and network-wide HOL blocking in the same way (like the previous RECN does). Another big improvement achieved by RECN-IQ versus the original RECN is that the latter is intended only for CIOQ switches, whereas RECN-IQ is suitable for IQ switches.

The final PCIQ architecture will thus be enhanced with a third key component: RECN-IQ. With RECN-IQ, the HOL blocking experienced within a switch can be eliminated, thus the switch will achieve maximum efficiency. But the most important point is that the PCIQ architecture will be also able to avoid the HOL blocking within the network by eliminating the negative effects of congestion in a fast and efficient way. This has been demonstrated in [34, 44].

2.9 Evaluation of PCIQ with RECN-IQ

Figures 2.14 and 2.15 show results for the 24-port switch with the PCIQ architecture with two crossbars and two or four SAQs per read port. For comparison purposes, figures also show results for the base PCIQ architecture (without RECN-IQ) with four subcrossbars and HC. Uniform traffic with 256 bytes packet size is used.

As can be observed, PCIQ now achieves maximum switch efficiency. For all the cases, the switch efficiency of the improved PCIQ architecture is comparable to HC (or even better). This is due to the fact that the incorporated RECN-IQ mechanism is able to eliminate all the HOL blocking existing in the switch. Even when we use only 2 SAQs.

Figures 2.16 and 2.17 show the efficiency achieved for the hot-spot traffic. As can be seen, PCIQ (with RECN-IQ) is the only one that filters out the HOL blocking completely, thus achieving maximum switch efficiency. In

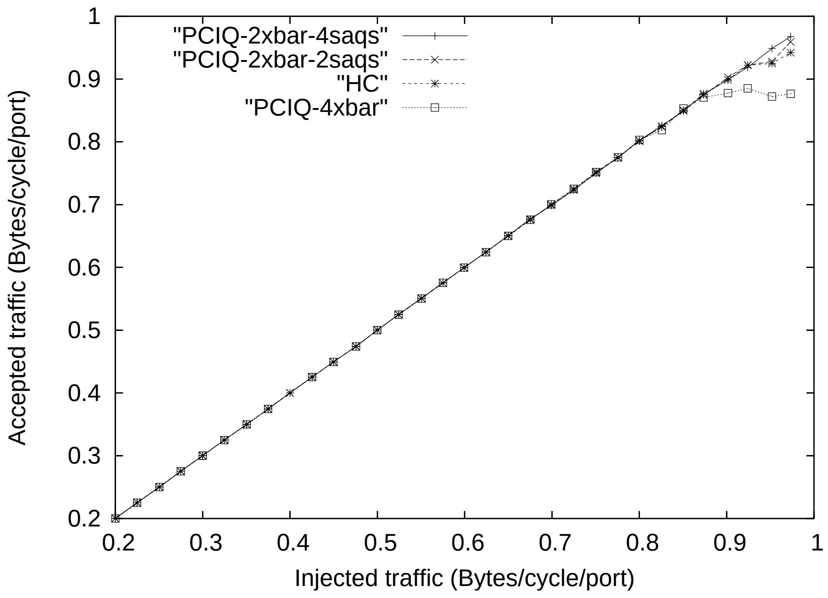


Figure 2.14: Switch efficiency of CIOQ, HC, and PCIQ. Uniform distribution of packet destinations.

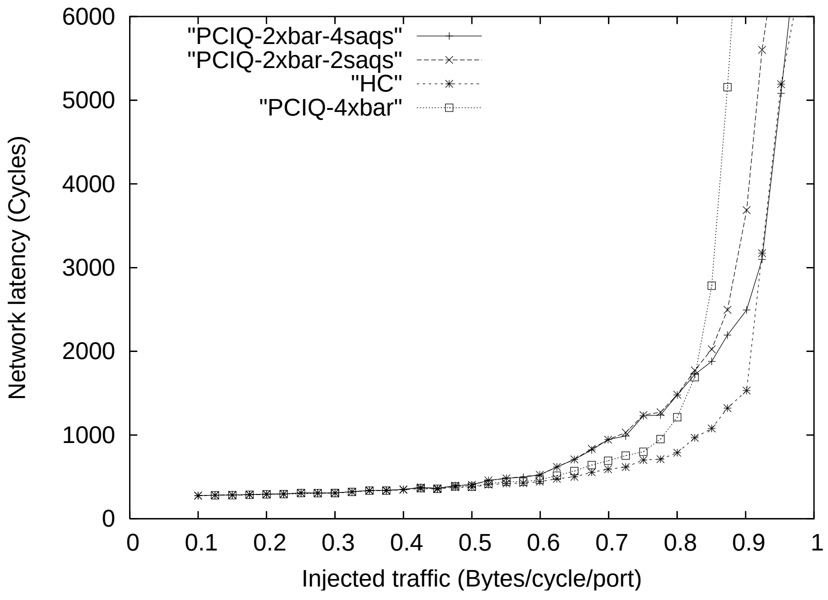


Figure 2.15: Network latency of CIOQ, HC, and PCIQ. Uniform distribution of packet destinations.

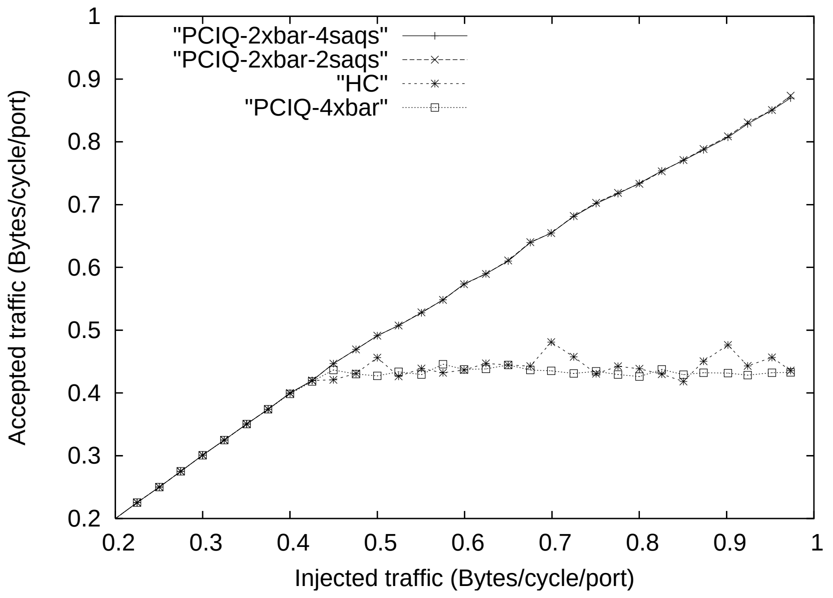


Figure 2.16: Switch efficiency of CIOQ, HC, and PCIQ. Hot-spot plus uniform distribution of packet destinations.

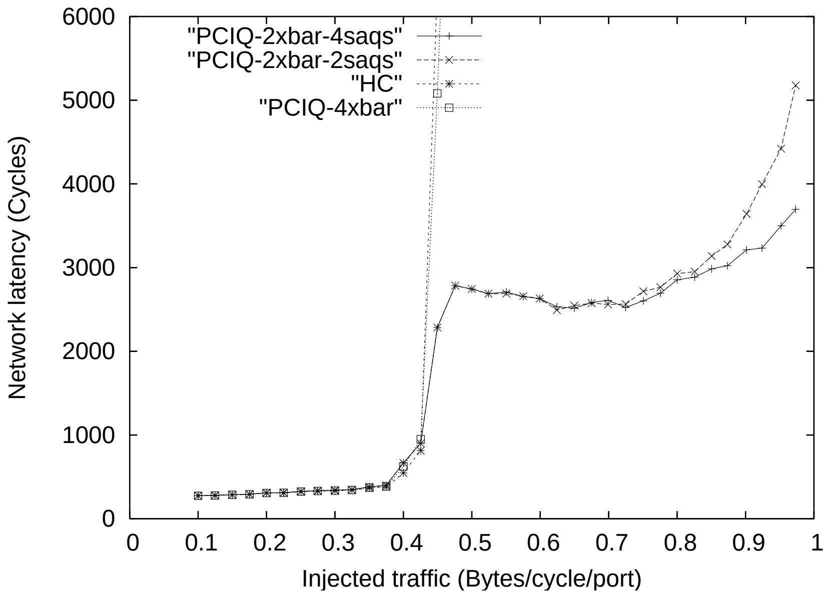


Figure 2.17: Network latency of CIOQ, HC, and PCIQ. Hot-spot plus uniform distribution of packet destinations.

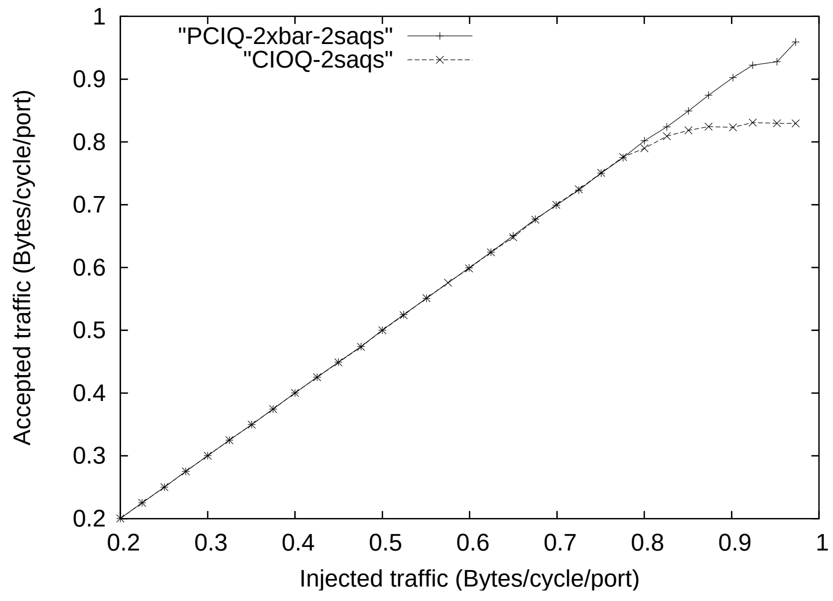


Figure 2.18: Comparing the switch efficiency of CIOQ and PCIQ, both with RECN-IQ. 256-bytes packets.

particular, by correctly handling HOL blocking, it achieves 90% of switch efficiency while CIOQ and HC organizations obtain results about 50% or even less.

When using RECN, the average packet latency is significantly increased once the hot-spot traffic congests (0.4 bytes per cycle per port). However, this is due to the latency experienced by congested packets ($2,500$ cycles on average) which is unavoidable.

RECN-IQ can also be added to any architecture. For instance, Figure 2.18 shows the switch efficiency for CIOQ architecture when RECN-IQ is used (with two SAQs) and a uniform traffic distribution. As we can observe CIOQ is not able to achieve maximum switch efficiency (even when using RECN-IQ). It obtains 80% of switch efficiency. Thus, CIOQ does not provide enough internal bandwidth. This motivates the need for a more elaborated switch organization. As shown, PCIQ reaches nearly 100% efficiency. As will be seen in Section 2.10, this comes at no extra overall cost.

2.9.1 Worst Case Analysis

Figure 2.19 shows the worst case analysis for the PCIQ architecture. The worst case is when the internal crossbars do not have enough packet candidates. This occurs when odd input links send packets only to odd output links and even input links send packets only to even output links. In this situation, each arbiter will only have $N/2$ packet candidates for $N/2$ poten-

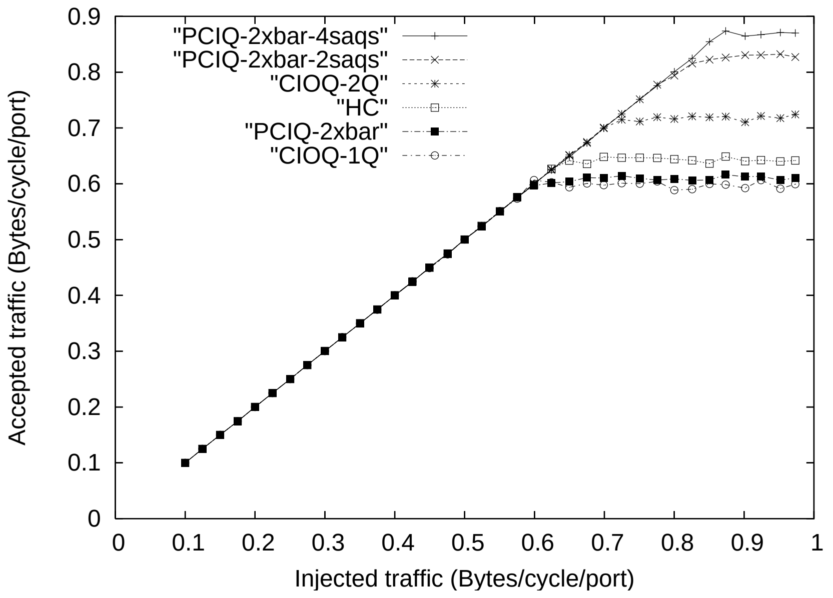


Figure 2.19: Switch efficiency for the worst case analysis. Odd input links sending to odd output links and even input links sending to even output links.

tial output links, thus resembling a CIOQ switch organization. As can be observed, the base PCIQ architecture (without RECN-IQ) achieves roughly 60% of switch efficiency, as CIOQ does. In [23] it is shown that in the worst case traffic for HC (first half of input links sending to the first half of output links and the second half of input links sending to the second half of output links) the maximum switch efficiency is bounded by 70%. However, notice that PCIQ (with RECN-IQ) is able to increase the switch efficiency to 90%. This is due, again, to the removal of the internal HOL blocking.

Up to now, we have presented evaluation results for one switch. For larger networks, the switch architecture has diminishing impact on overall performance as network size increases. For very large networks, congestion management becomes the dominant contributor to performance. It was already shown in [34] that RECN is able to eliminate HOL blocking even for large networks. As PCIQ incorporates RECN-IQ, it inherits its benefits. However, PCIQ is still a good switch architecture for very large networks because, as will be seen in the next section, it features a lower cost compared to other architectures.

2.9.2 Multi-stage Interconnection Network Analysis

We have also simulated a complete network using the different switch architectures studied here. A 16×16 multi-stage interconnection network,

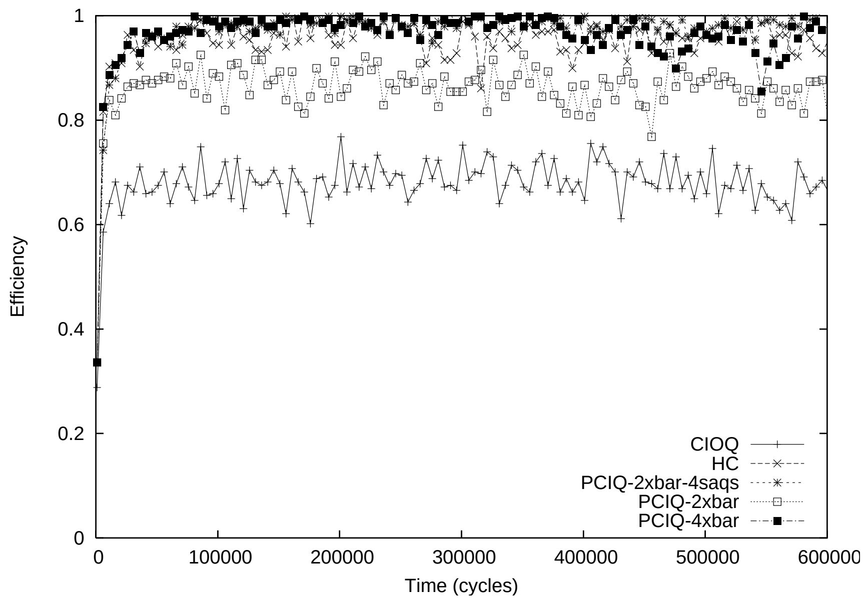


Figure 2.20: Efficiency over time for a 16 × 16 multi-stage interconnection network and uniform traffic.

connected by a perfect shuffle, was the configuration selected.

For uniform traffic, Figure 2.20 shows that the network built using PCIQ with four crossbars switches achieves maximum performance, the same happens for PCIQ with two crossbars and enhanced by RECN (using only 4 SAQs). HC achieves slightly lower network efficiency than PCIQ-4xbar, but still higher than the rest, as one can imagine from the results for a single switch (Figure 2.10).

On the other hand, when there is a hot-spot (the same type as described in Section 2.9) added to the uniform traffic, the network performance drops dramatically for all architectures but PCIQ enhanced with RECN-IQ. As shown in Figure 2.21, using a very limited number of SAQs we can achieve maximum network efficiency.

2.10 Cost Analysis

The cost of a switch is mainly influenced by three components: the memory resources, the arbiter, and the crossbar[2]. In this section we evaluate these components in the PCIQ architecture. For comparison purposes, we also analyze the cost of the CIOQ and HC switch organizations.

For the cost of the memory resources, Figure 2.22 shows the number of

[2]We do not include the link controllers since they are the same in all the architectures under comparison.

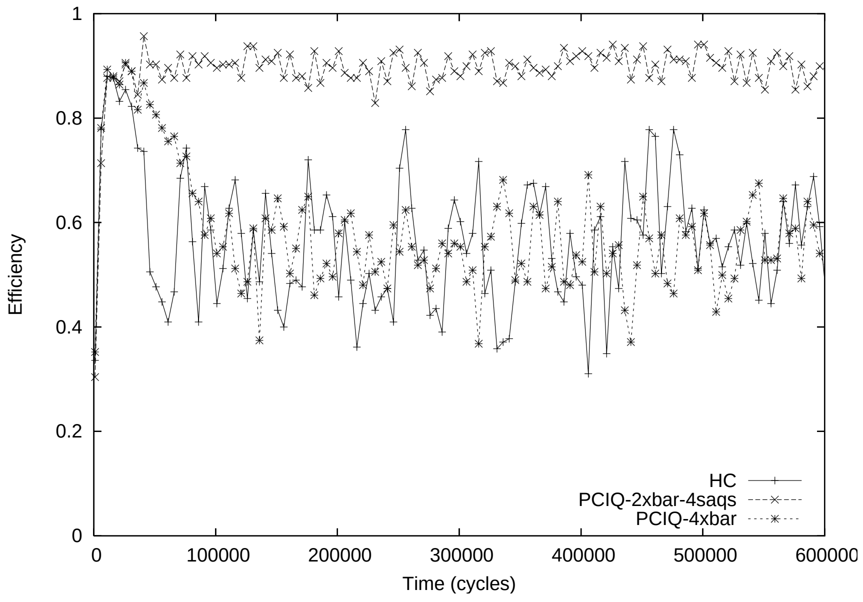


Figure 2.21: Efficiency over time for a 16×16 multi-stage interconnection network and a hot-spot traffic.

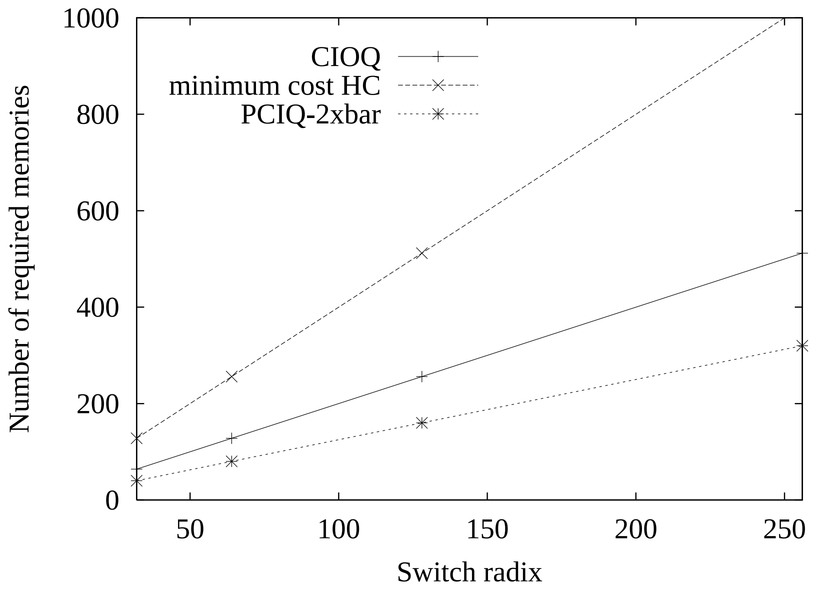


Figure 2.22: Memory cost for different switch architectures.

memories required by each architecture. This value provides a first-order estimation of the relative amount of silicon area required to implement the buffer memories. For the PCIQ architecture, a factor of 1.25 is added because of the additional silicon area required to implement the second read port and the control info for SAQs. As it can be observed, the PCIQ switch architecture achieves the minimum memory cost. As the radix (N) of the switch increases, the number of memories required for PCIQ increases linearly with the number of ports ($1.25N$). For the CIOQ architecture, the memory cost also increases linearly ($2N$), but exhibiting higher memory needs than PCIQ, due to the need for output memories. For the HC, memory cost varies depending on the selected p parameter (the memory cost is $2p(N/p)^2$). Keeping p constant and increasing the radix (N), the memory cost grows quadratically. Anyway, for any radix, the value of p that gives the lowest memory cost is $p = N/2$ (four subcrossbars within the switch). Considering always the value of p that minimizes memory cost, we can see that HC requires more queues ($4N$) than PCIQ. It may be argued that the extra cost of an additional memory read port depends on the VLSI technology and thus, the factor added to PCIQ could be higher. However, for different technologies it will not be higher than 2 (100% of additional cost per port). Thus, the reduction in memory area requirements from HC to PCIQ varies from a factor of 2 (for 100% additional cost per port) to a factor of 3.2 (for 25% additional cost per port).

The memory size requirements depend on the number of queues needed. PCIQ implements a single landing pad on each memory. Additionally, a pool of bytes is required to implement all the queues. Also, it needs just one queue and two SAQs to achieve maximum switch efficiency. On the other hand, the landing pad is also required by HC and CIOQ switch organizations (if they use dynamic memory allocation). However, the number of queues for CIOQ should be at least four (to achieve an acceptable performance level as previously seen). For HC, just one queue is required, but it does not solve HOL blocking. If HC and CIOQ were to address internal HOL blocking, they would require at least the same number of queues (when using RECN) as PCIQ, if not more (when using VOQs). In summary, memory size requirements for PCIQ are not higher than for other switch architectures.

An $N \times N$ crossbar can be implemented in several ways. A common alternative is by using small switches at the crosspoints. The number of crosspoints is N^2, thus a cost proportional to N^2 can be assumed for the crossbar. Assuming this cost we can now deduct the cost of the crossbars used in the different switch organizations. For the CIOQ organization its cost is clearly N^2, since it uses an $N \times N$ crossbar. For the HC organization, it uses $(N/p)^2$ subcrossbars of size $p \times p$. Since the cost of the subcrossbar is p^2, the total cost of all the crossbars is N^2. Finally, for the PCIQ organization and assuming q subcrossbars, it will have q subcrossbars of size $N \times N/q$. Thus the total cost will be also N^2. Therefore, all the switch organizations have the same cost. It should be noted that the organization used as a reference to elaborate PCIQ was the CIOQ with two read ports and a $2N \times N$ crossbar. In that organization, the crossbar cost ($2N^2$) is significantly higher.

The arbiter is assumed to be a two-level hierarchical arbitration scheme in all the switch architectures. For PCIQ, the first level consists of two round-robin arbiters per input port (one to handle requests for odd ports and the other one for requests for the even ports) that work in parallel. The complexity of such arbiters is $(Q + S) \times 1$ where Q is the number of queues per input port and S is the number of SAQs. In the second level, a request is issued from each arbiter to the arbiters located at the output links. Each output link has a round-robin arbiter with a complexity of $N \times 1$. Therefore, for a configuration with $Q = 4$, $S = 4$ and $N = 64$, arbiter implementation requires 128 arbiters with complexity 8×1 at the inputs and 64 arbiters with complexity 64×1. In [43], the proposed $N \times 1$ round-robin arbiter (implemented with a hierarchical tree of smaller arbiters) exhibits a linear cost in terms of silicon area. This arbiter is valid for implementing all the PCIQ arbiters.

Anyway, the cost of the arbiter can be deduced from the crossbars. Since an $N \times N$ crossbar allows N candidates to arbitrate for N resources, we can approximate the cost of the arbiter for an $N \times N$ crossbar as N^2 (candidates times resources). Following this model we can conclude that the cost of the arbiters used in PCIQ, HC, and CIOQ are similar.

To sum up, the overall cost of the PCIQ architecture is significantly lower than the cost of the CIOQ and HC organizations. Although the cost of the crossbars and arbiters is similar, the main benefit comes from the reduced cost for the memories. Taking into account that as the transmission frequency increases, round-trip time and buffer size requirements increase proportionally, we conclude that the overall cost of a switch is dominated by the memory cost. Therefore, the PCIQ architecture is the one that achieves the lowest cost.

2.11 Conclusions

In this chapter we have proposed a new switch architecture for high-radix switches that relies on three key components. First, a partitioned crossbar is used in order to increase the read bandwidth at the input memories without neither increasing the cost of the crossbar nor the arbiter. Second, two round-robin packet-based arbiters (one for each crossbar) are used. They exhibit a linear cost and a logarithmic response time (as the radix of the switch increases). Third, a congestion management technique (RECN-IQ) is incorporated to eliminate both the internal switch and the network-wide HOL blocking.

The proposed architecture, referred to as PCIQ, exhibits a cost (measured in memory requirements, crossbar complexity and arbiter complexity) similar to or lower than basic organizations like CIOQ. However, it is able to achieve maximum switch efficiency for uniform traffic distribution, thus leveling costly organizations like BC. Additionally, PCIQ is able to eliminate all the switch and network-wide HOL blocking, thus achieving maximum throughput in the presence of non-uniform traffic. These results come at a cost clearly inferior to the HC architecture.

In the next chapter, the RECN-IQ mechanism for eliminating the HOL blocking used on the enhancement of PCIQ is presented in full detail.

Chapter 3

The RECN-IQ Mechanism

Weaseling out of things is important to learn. It's what separates us from the animals ... except the weasel.
– Homer, "The Simpsons"

As we have seen before, network performance (which directly impacts on the overall system performance) may dramatically drop during congestion situations. Moreover, the current trend of reducing overall cost and power consumption by lessening the number of network components makes this problem even harder, thereby becoming mandatory the use of congestion management techniques.

Here, following the basic approach used by the Regional Explicit Congestion Notification (RECN) technique (eliminating the Head-of-Line blocking produced by congested packets turns congestion harmless), we completely redefine the RECN mechanism in order to achieve different goals. First, we adapt RECN to a switch organization with queues only at input ports. These switches are simple and cheap to produce. Second, we propose a new method for detecting congestion that does not require detection queues, thereby reducing memory requirements and switch cost.

These improvements lead to achieve a cost-effective switch organization that derives maximum performance even in the presence of congestion. So, we present in detail a realistic switch architecture, known as RECN-IQ, supporting the new congestion management mechanism. Results demonstrate that RECN-IQ achieves maximum network performance under all analyzed situations.

3.1 Introduction

Congestion occurs when several flows of packets simultaneously and persistently request the access to the same network resources (typically, a switch output port). In these cases, any packet not granted will block, and will remain stored[1] in a queue until its request is attended. This may cause the appearance of the phenomenon known as Head-Of-Line (HOL) blocking, that occurs when a packet at the head of a queue blocks, preventing the rest of packets in the same queue from advancing, even if they request available resources. When congested packets block and produce HOL blocking to non-congested ones (those packets belonging to flows that do not contribute to congestion), non-congested flows advance at the same speed as congested flows, thereby severely degrading network performance and eventually collapsing the network (the effect is rapidly spread over the entire network).

Although the negative consequences of congestion have been always evident, congestion has not been considered a critical problem until recently, due to several reasons. For instance, networks were traditionally overdimensioned (using more network components than strictly needed), and this leads to a very low link utilization, thereby reducing contention and congestion probability. Moreover, different queue organizations at switches reduce or eliminate the HOL blocking effect. This is the case of VOQ where a queue is used per each output port of the switch (VOQ at the switch level)[13] or per each possible destination end node (VOQ at the network level)[31]. Also, the traditional architecture for switches in communication networks used queues only at their output ports (Output Queuing, OQ switches), so HOL blocking (at switch level) was eliminated in this case.

But the high cost and power consumption of current network components discourages to overdimension the network. In fact, it is more appropriate to use a lower number of network components for interconnecting the terminals, thus reducing cost and power consumption as we have seen in Section 1.4. However, link utilization increases and subsequently congestion probability. On the other hand, the OQ scheme has become unfeasible because it requires the memories to operate at a much faster speed than the links, and link speed in current high-speed interconnects is on the order of Gbps. Thus, most current switches use either queues only at input ports (IQ switches), or queues at both input and output ports (CIOQ switches)[2]. However, IQ and CIOQ switches may be affected by HOL blocking. In fact, this problem may limit the throughput of the switch to about 58% of its peak value [12]. Although VOQ schemes may be used, they become very expensive as the memory is the component that drives the final cost of the switch. CIOQ overcomes this limitation by increasing the speedup within the crossbar, thus approaching to an OQ scheme and therefore with the same implementation problems explained before.

Taking all this into account and in order to keep network performance at maximum levels while using a reduced number of network components, the

[1]We assume lossless networks, where blocked packets are never discarded. Note that most current interconnects (Myrinet 2000, Quadrics, Infiniband, etc.) are lossless.

[2]Although other schemes, like BC (Buffered Crossbar)[16] switches, have been proposed, they are not so popular because of high cost.

use of an efficient congestion management technique is becoming mandatory in modern interconnects. Although many techniques have been proposed in this sense, none of them has been completely satisfactory until the proposal of Regional Explicit Congestion Notification (RECN) [34, 35]. RECN focuses on detecting and eliminating the HOL blocking produced by congested packets. In order to achieve this, RECN identifies congested packets and stores them in special, dynamically-assigned set aside queues (SAQs). RECN completely eliminates HOL blocking while requiring a small number of resources (queues) per port. In fact, RECN is the first truly efficient and scalable HOL blocking elimination technique.

However, RECN has been proposed and designed assuming CIOQ switches. This narrows the scope of RECN, whose application is limited to switches with this type of switch organization, while IQ switches are currently preferred since their implementation is usually simpler and cheaper. Note that the cost of a switch is mainly driven by the memory used, and most of the switch power and area is consumed by the memory. This fact suggests that switch designs with fewer and smaller memories are preferable, thus the IQ organization being more attractive than CIOQ; another benefit is that is can be applied to the PCIQ switch architecture presented in the previous chapter since PCIQ has no buffers at the outputs. Taking all this into account, one of the novelties we present in this chapter is a new RECN version whose main aim is to adapt to IQ switches, thereby allowing any switch model with this organization to get efficient and scalable congestion management. The new version will be referred to as RECN-IQ. In this sense, RECN will be compatible with both CIOQ and IQ architectures, on which most of the current switches are based.

Additionally, we have reduced the memory requirements of RECN at the input ports of a switch. Specifically, the previous RECN version required several detection queues at each port (one per output port in the switch) in order to detect congestion. In detail, whenever a detection queue fills over a given threshold, the output port associated to the detection queue is considered as a congested point. Although this method works accurately, it is expensive as it increases the silicon area required at input ports, even if memory is dynamically managed. In order to avoid all the aforementioned problems, this novel RECN-IQ mechanism includes a new method for detecting congestion at input ports that requires a single detection queue per input port, i.e., a reduction factor of N for an $N \times N$ switch.

Note that, since the new RECN proposal removes data memories at output ports and detection queues at input ports, the power and area required for switches will be significantly reduced, thereby allowing us to build cheaper networks as switches will have fewer and smaller memories. Of course, this will lead to a reduction in the cost and power consumption of the overall system.

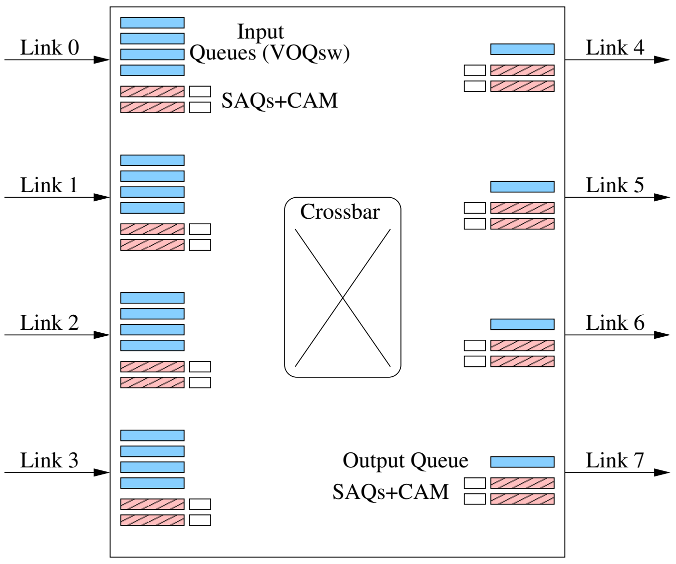


Figure 3.1: Queue distribution for a switch implementing the original RECN. At each input port, a set of VOQ is required plus some SAQs+CAM (two in this case). At the outputs it is required a single queue plus the some SAQs+CAMs (same number as there are at the input ports).

3.2 Previous RECN

RECN separates congested and non-congested flows by storing them into different queues, thus eliminating the HOL blocking introduced among them. Specifically, RECN adds a set of additional queues (Set Aside Queues, SAQs) at every input and output port of a switch. Figure 3.1 shows an schematic of the memory organization required in order to implement RECN on a CIOQ switch architecture.

RECN assumes that packets from non-congested flows can be mixed in the same queue without producing significant HOL blocking. Thus, standard queues will store non-congested packets and SAQs will be dynamically allocated for storing packets passing through a specific congested point. Every set of SAQs is controlled by means of a Content Addressable Memory (CAM) [45], each CAM line containing control information for managing an associated SAQ, including the information required for addressing a congested point.

In this sense, RECN (and also RECN-IQ) addresses network points by means of the routing information included in packet headers, assuming that source routing is used. For instance, Advanced Switching (AS) [42] packet headers include a turnpool made up of 31 bits, which contains all the turns (i.e. offset from the input port to the output port) for every switch in a route. See Figure 3.2 for an example of a route defined by turns. Therefore,

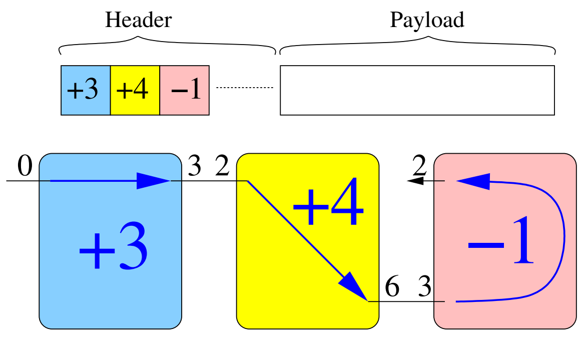


Figure 3.2: Source based routing with the header of the packets indicating the turns (offset between input port and input port at each hop along the packet's path).

in AS networks, CAM lines include turnpools addressing congested points, and these turnpools can be compared to the turnpool of any packet, in order to know whether the packet will cross the corresponding congested point. In this way, packets crossing a particular congested point can be easily detected.

In order to identify a point as congested, RECN implements different congestion detection mechanisms at input and output ports. At any output port (see Figure 3.1), whenever the standard queue fills over a given threshold, congestion is detected at this point, and a notification is sent to any input port of the switch sending packets to the congested output port. These notifications include the relative address required to reach the congested point from the input port receiving the notification (a turn in the turnpool). Upon reception of a notification, an input port must allocate a new SAQ and fill the corresponding CAM line with the received turnpool.

On the other hand, at input ports, the standard queue is divided into several small detection queues, one per output port (therefore following some kind of VOQsw scheme); this can be seen on Figure 3.1. Whenever a detection queue fills over a given threshold, congestion is detected at the corresponding output port and a new SAQ associated to this congested point is automatically allocated at the input port. Immediately, all the packets stored in the detection queue are transferred to the SAQ (the detection queue becomes empty and non-operative), and a notification containing the turnpool of the newly allocated SAQ is sent to the output port of the upstream switch, where a new SAQ should be subsequently allocated.

Once a SAQ is allocated at a given port, every incoming packet will be directly stored in the SAQ if it will pass through the associated congested point. Otherwise, the packet will be stored in the standard (or detection) queue. In this way, non-congested packets are always separated from the congested ones, thereby preventing the appearance of HOL blocking among them.

Furthermore, if any SAQ becomes congested (reaches a given threshold), a notification will be sent upstream, and the receiving input or output port will allocate a new SAQ. This procedure can be repeated until these notifications reach the sources. Therefore, a SAQ will be allocated at every point where otherwise HOL blocking could be introduced. RECN uses a SAQ-specific Xon/Xoff (Stop & Go) flow control in order to prevent SAQs from using all the memory space.

RECN also detects congestion vanishing at any point, in such a way that SAQs can be deallocated asynchronously. Specifically, the conditions for deallocating a SAQ are the following: the SAQ must be empty, and it must be in Xon state (must not be blocked by a downstream SAQ). Note that these conditions allow a distributed SAQ deallocation, in such a way that a SAQ can be deallocated independently of other SAQs. Since deallocated SAQs can be re-allocated for new congested points, this policy reduces the number of SAQs per port required for completely eliminating HOL blocking. For a detailed description of the RECN mechanism, refer to [34, 35].

3.3 The RECN-IQ Mechanism

In this section the new version of RECN for IQ switches is described. For this, we show the main modifications introduced in the previous RECN mechanism (suitable for CIOQ switches). In Chapter 4 we will describe a switch architecture that includes RECN-IQ, detailing all the required memory and logic components.

For the sake of an easier understanding, the following subsections describe in detail and separately each different functional aspect of RECN-IQ: memory management, congestion detection, allocation of queues, processing of packets, and flow control. Later, when describing the switch architecture including RECN-IQ, we will implement each aspect in a separate functional unit.

3.3.1 Memory Management and Requirements

The RECN-IQ mechanism has been designed assuming data memories only at the input ports of the switch. At each input port a data memory will be used for allocating a normal (Cold) queue and a set of SAQs. In order to manage the SAQs, an associated CAM (Content-Addressable Memory) will be required at each input port. Additionally, a CAM structure will be required also at output ports. Figure 3.3 depicts such memory requirements.

In the previous RECN mechanism, some detection queues were used at each input port. However, in the RECN-IQ mechanism only a single queue, the *Cold Queue* (CQ), will be used for storing all the incoming packets (both congested and non-congested). Therefore, the new RECN mechanism will require at each switch fewer memory resources, both at the output ports

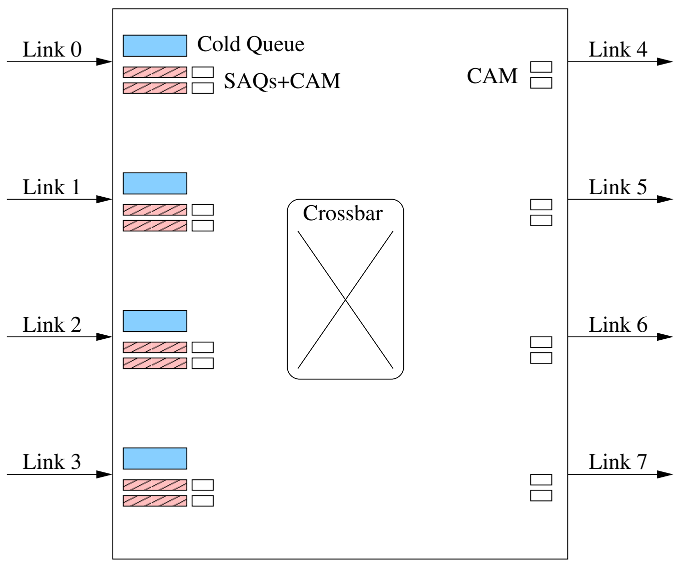


Figure 3.3: Queue distribution for a switch implementing the RECN-IQ congestion management. At each input port, a single queue (known as Cold Queue, CQ) plus some SAQs+CAM (two in this case) are required. There is no buffering at the outputs, only some CAMs for keeping track of the congestion situation.

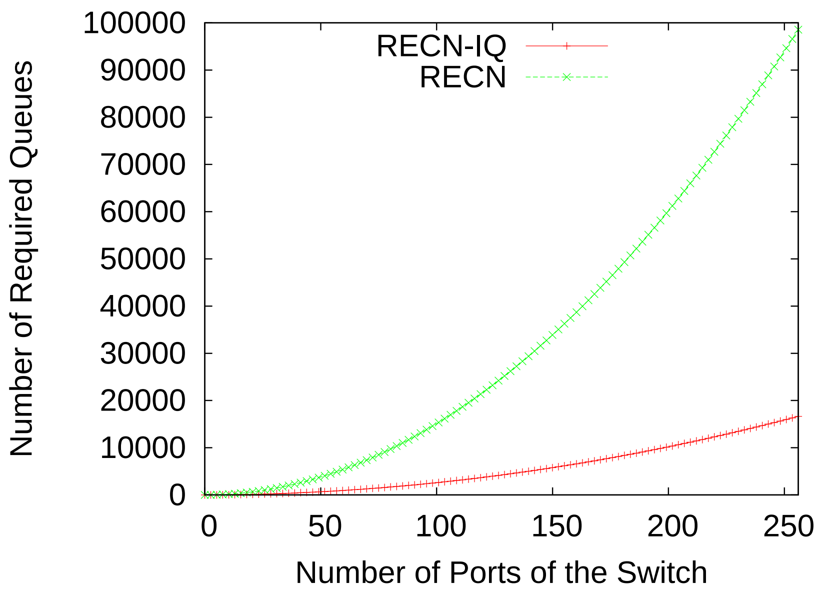


Figure 3.4: Relationship between the number of ports of the switch and the required queues for an architecture based on RECN and on RECN-IQ. It is assumed that the number of SAQs is equal to the number of ports divided by four.

(where only the CAM is required) and at the input ports (where detection queues are replaced by only one queue). Thus, the number of queues required for a RECN-IQ switch ($Q_{\text{RECN-IQ}}$) can be calculated through the following expression:

$$Q_{\text{RECN-IQ}} = N \times (1 + S) \tag{3.1}$$

where N is the number of ports of the switch and S is the number of SAQs implemented.

On the other hand, a switch implementing the previous RECN requires N detection queues plus S SAQs per input port, and one queue plus S SAQs per output port. Therefore, the number of queues required (Q_{RECN}) for a RECN switch is given by the following expression:

$$Q_{\text{RECN}} = N \times [(N + S) + (1 + S)] \tag{3.2}$$

Assuming that the number of SAQs required is always the number of ports divided by 4, the relationship between the radix of the switch and the queue requirements can be plotted (Figure 3.4). For instance, assuming a detection threshold of 2 slots (a slot will store one packet), an Xoff threshold (required for activating the "stop" function in the flow control between SAQs) of 2 slots, 4 SAQs, and a 16×16 switch; the RECN mechanism would require 800 memory slots, whereas just 160 slots is enough for the new RECN-IQ. A reduction factor of 5. It should be noted that these numbers have been obtained assuming short cables where Round-Trip-Time (RTT) is lower than a packet/slot.

3.3.2 Congestion Detection

The new RECN-IQ mechanism detects congestion only at input ports. In particular, whenever the number of packets in the *Cold Queue* (CQ) exceeds a given threshold (the *RECN-IQ* threshold), congestion is detected. Once a congested situation is detected, the congested point must be identified.

The RECN-IQ mechanism assumes that the origin of congestion is the output port requested by the first packet at the CQ queue. This assumption is based on the fact that, under a congested situation, it is very likely that the first packet in a congested queue is blocked because it requests a congested output port. Therefore, under these situations, the detection mechanism will hit the congested output port.

Indeed, in a non-congested situation, the rate at which packets arrive to a given input port will be roughly the same at which packets leave the input port, thus the queue's occupancy will be low. Thus, if the CQ increases in size is because the packet at the head is temporarily blocked.

On the other hand, it may happen that the packet at the head of the queue is not addressed to a congested output port. In that situation, the

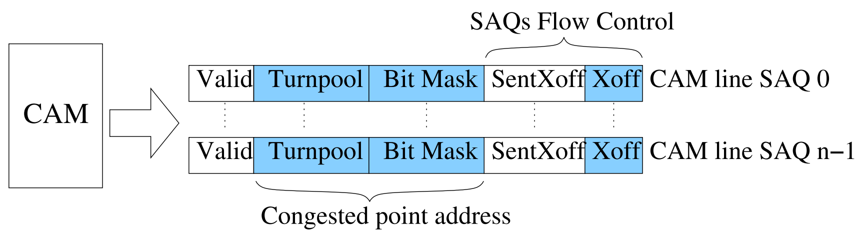


Figure 3.5: CAM structure for RECN-IQ.

detection mechanism will fail, but the Post-Processing unit and the deallocation policy of SAQs will minimize the impact of the false positives (as explained below, in Section 3.3.8).

3.3.3 SAQ Allocation and Deallocation

Once congestion is detected, the input port allocates a new SAQ for the congested point (output port requested by the packet at the head of the CQ).

The associated CAM line, whose structure can be seen in Figure 3.5, will include all the routing information (turnpool + bit mask) and the status (*Valid*, *Xoff*, and *SentXoff* bits) of the SAQ. An active *Valid* bit indicates that the SAQ is assigned to a congestion point; an active *Xoff* bit indicates that the SAQ is stopped by the flow control between SAQs; and an active *SentXoff* bit indicates that the SAQ has sent an Xoff signal to an upstream SAQ.

For instance, when a new SAQ is allocated because of congestion detection at the CQ, it will set the "*Valid*" bit, and will reset the "*Xoff*" and "*SentXoff*" bits. As in the previous RECN, if there exists an associated SAQ for the detected congested point, then no new SAQ is allocated. Thus, before allocating a new SAQ, a search in the CAM structure is performed.

Whenever a SAQ empties and is not blocked by the Xon/Xoff flow control (*Xoff* bit not set), then the SAQ is deallocated. So the RECN-IQ mechanism allows the deallocation of SAQs in a distributed way.

3.3.4 Packet Processing

Whenever a new packet arrives to the switch, it is stored in the CQ queue, regardless whether it is going to pass through a detected congested point or not. A mechanism (Post-Processing mechanism, PP) at each input port will decide later if the packet goes to any of the different SAQs allocated at this port. The PP mechanism cyclically inspects all the queues (CQ and SAQs), one at a time. At each queue, it processes the packet at the head of the queue. In particular, the routing information of that packet is compared to the routing information of each CAM line. Thus, it will be detected whether or not the packet is going through a congested point in the network. In the case of a match, the packet is moved to the corresponding SAQ. Otherwise, the packet is set as ready for the scheduler. In this way, congested packets will not be blocked at the head of the queue, thereby avoiding HOL blocking.

Notice that the routing information of a packet may match at the same time the routing information of several active CAM lines in an input port. Thus, different situations may arise depending on the queue where the packet is initially mapped into. The first situation occurs when the packet is initially in the CQ and there are two matches, in that case the shortest match is selected. Thus, in a first post-processing step, the packet is stored in the SAQ with the "less-specific" (shortest) associated turnpool. In a second situation, a packet stored in a SAQ matches (when it is post-processed) two CAM lines, one of them being less-specific than the one associated to the current SAQ. In that situation, the post-processing mechanism selects the less-specific match, but larger than the routing information for the current SAQ.

To sum up, packets move initially from CQ to one SAQ, then from that SAQ to another in increasing matching size, until the packet reaches the SAQ with the largest match. This is the penalty for preserving in-order delivery of packets which is important for some applications. Also, this prevents the burden of using reordering buffers at destination. Once a packet is post-processed and there is no match with any CAM line, then the packet is set as ready for the arbiter.

It is worth to mention that the arbitration must not distinguish between requests from the CQ and the ones from the SAQs, i.e. all queues from an input port must be treated equally by the scheduler, with no hard-priority. Therefore arbitration schemes like iSLIP [46], for instance, are the most appropriates for RECN-IQ.

3.3.5 Congestion Information Propagation

The detection of congestion propagates between switches in the following manner. Whenever the number of packets on a SAQ exceeds the Xoff RECN Threshold, then the input port sends backwards an Xoff signal (containing the routing information that points to the associated congested point) to the corresponding output port at the upstream switch. Upon reception, a new CAM line is allocated at the output port of the switch pointing to the congested point (if there is not already a SAQ allocated for the notified congested point). The *Valid* and *Xoff* bits are set.

Whenever a packet passes through the output port[3], the routing information is inspected and compared to the routing information stored in the active CAM lines at this output port. If there is a match and the matching

[3]Notice that queues are not implemented at output ports.

CAM line has its *Xoff* bit set, an internal Xoff notification is sent to the input port that sent the packet. Upon reception of that Xoff signal, the input port allocates a new SAQ+CAM line, with routing information pointing to the congested point (note that the routing information from the output port CAM line is updated at the input port CAM line to include an additional turn). The *Xoff* bit for that newly allocated SAQ is set. However, if a SAQ already existed for the congested point, then the *Xoff* bit of its associated CAM line is set.

3.3.6 Flow Control

The Cold Queues at the input ports forward packets on a credit flow control fashion at the memory level, i.e., whenever there is enough free space at the requested input memory, the Cold Queue can forward packets.

On the other hand, the flow control for the SAQs is different. A SAQ with its associated CAM line having the *Xoff* bit set cannot send packets. Whenever the number of packets on a SAQ goes below the RECN Xon threshold, an Xon signal is sent backwards to the connected output port at the upstream switch. Then, the corresponding CAM line at the output port resets the *Xoff* bit and broadcasts an internal Xon signal to all the input ports of the switch. The input ports with an allocated SAQ for the congested point reset the *Xoff* bit on the corresponding CAM line, thus allowing the packets stored in the SAQ to move forward.

3.3.7 Procedure Example of the RECN-IQ Mechanism

In order to better understand RECN-IQ, a graphical example showing the basic procedure is shown in figures 3.6 through 3.15.

Step 1 (Figure 3.6) shows how congestion is detected. Output link number 5 of switch 2 is oversubscribed, therefore packets begin to accumulate on the *Cold Queue* (CQ) of several input ports of switch 2. When the number of packets stored in the CQ at input link 0 of switch 2 exceeds the RECN-IQ threshold, congestion is detected at the output port requested by the packet at the head of that CQ. Because at that moment, the packet at the head of CQ at link 0 of switch 2 is requesting the output port number 5, that output port number 5 of switch 2 is considered the congested point.

Step 2 (Figure 3.7), a new SAQ+CAM line is allocated at input port 0 of switch 2, pointing to the detected congested point. From input's port 0 of switch 2 perspective, the congested point at link 5 is identified by the turn +5 at the current switch. Congested packets at the head of the CQ at input port 0 of switch 2 are moved into the SAQ by the Post-Processing mechanism, so avoiding the HOL blocking these packets may produce.

Step 3 (Figure 3.8), when the number of packets at the recently allocated SAQ exceeds the Xoff threshold, an Xoff signal is sent backwards to the

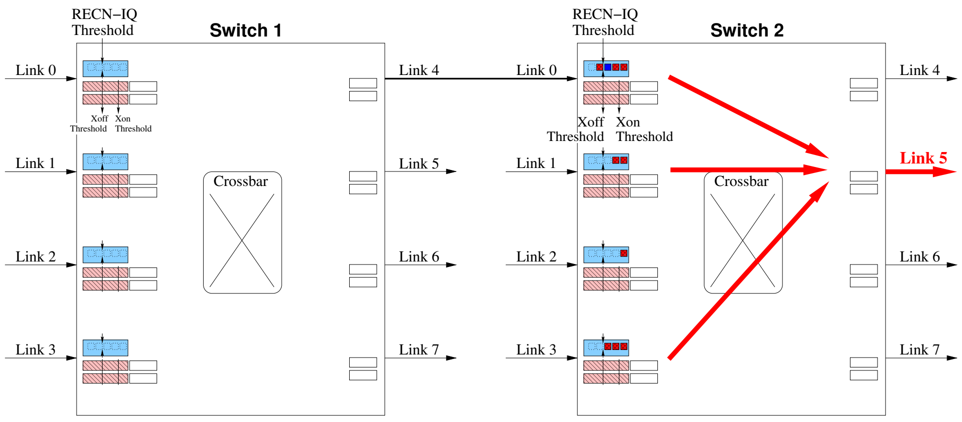


Figure 3.6: Step 1 of 10 of the RECN-IQ basic procedure example.

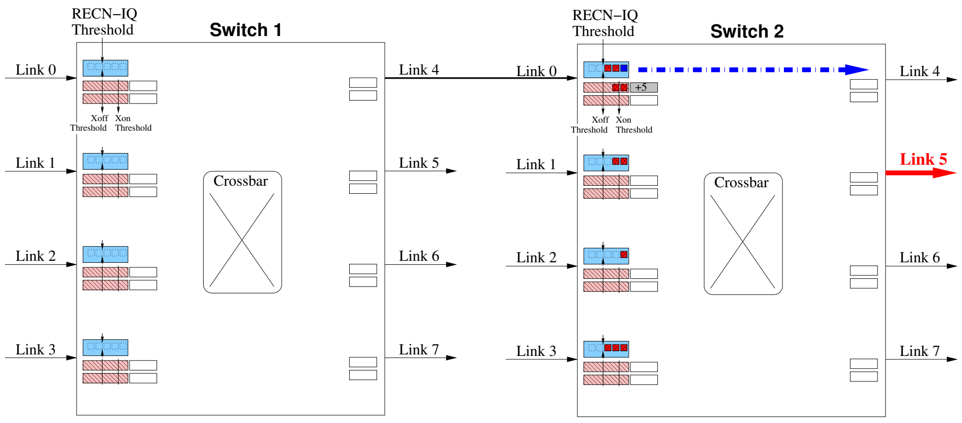


Figure 3.7: Step 2 of 10 of the RECN-IQ basic procedure example.

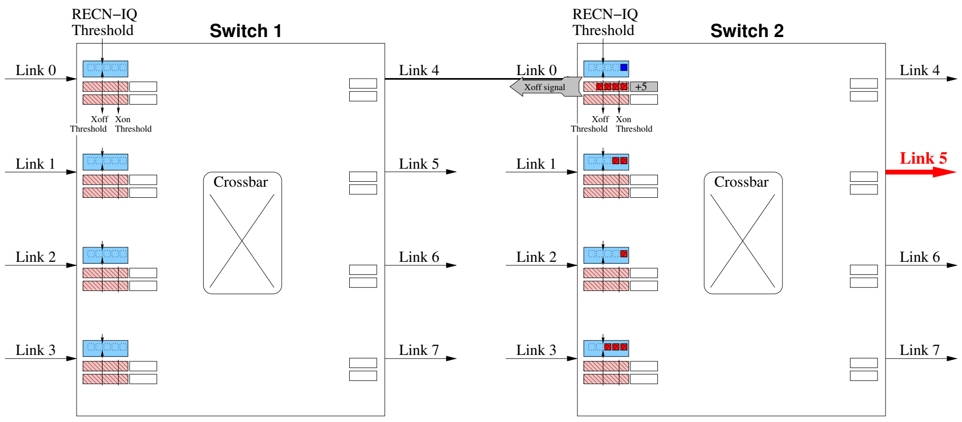


Figure 3.8: Step 3 of 10 of the RECN-IQ basic procedure example.

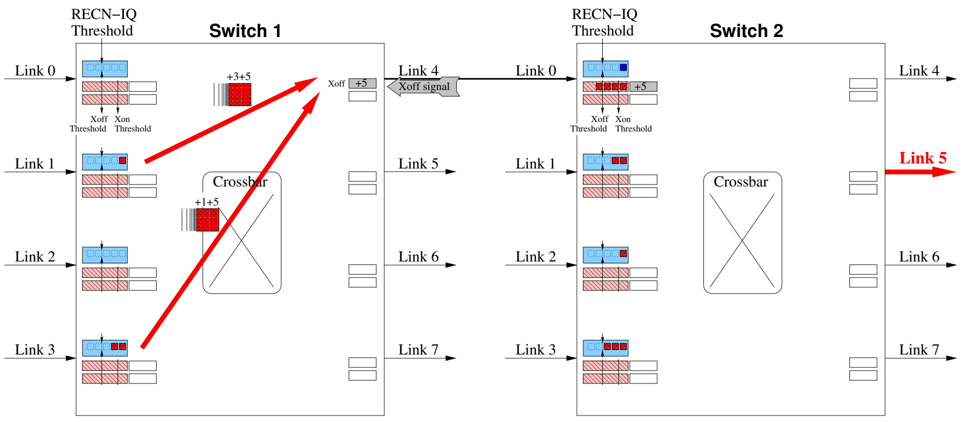


Figure 3.9: Step 4 of 10 of the RECN-IQ basic procedure example.

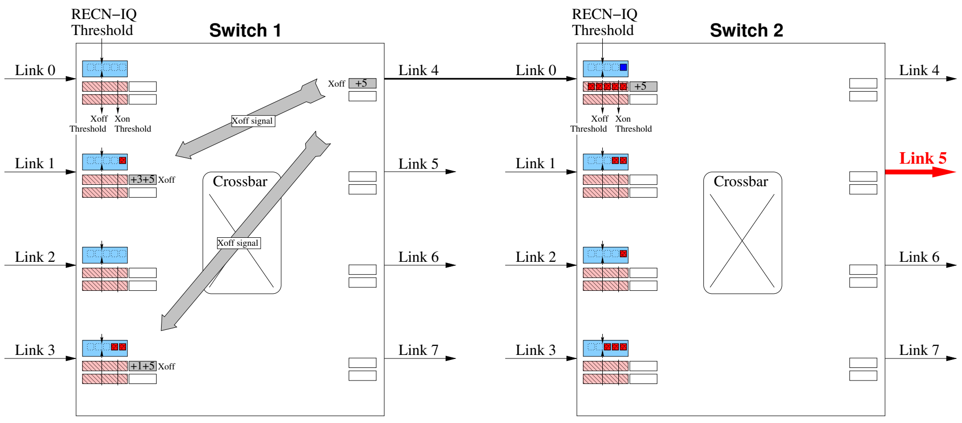


Figure 3.10: Step 5 of 10 of the RECN-IQ basic procedure example.

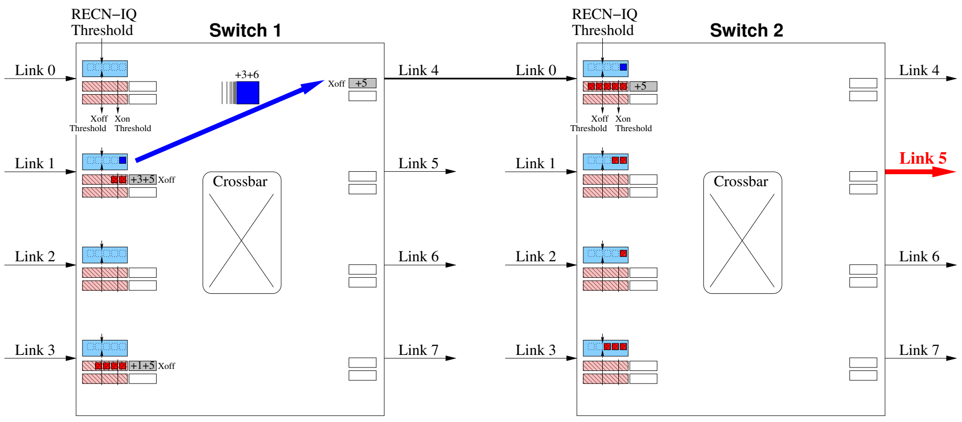


Figure 3.11: Step 6 of 10 of the RECN-IQ basic procedure example.

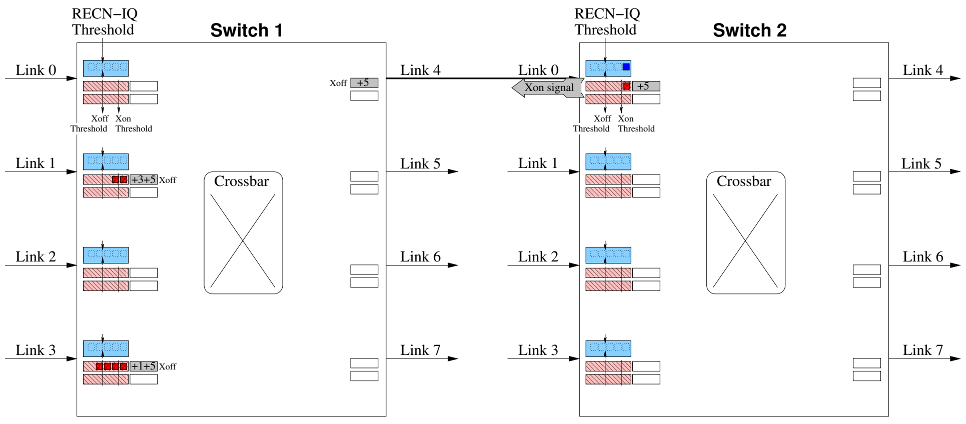


Figure 3.12: Step 7 of 10 of the RECN-IQ basic procedure example.

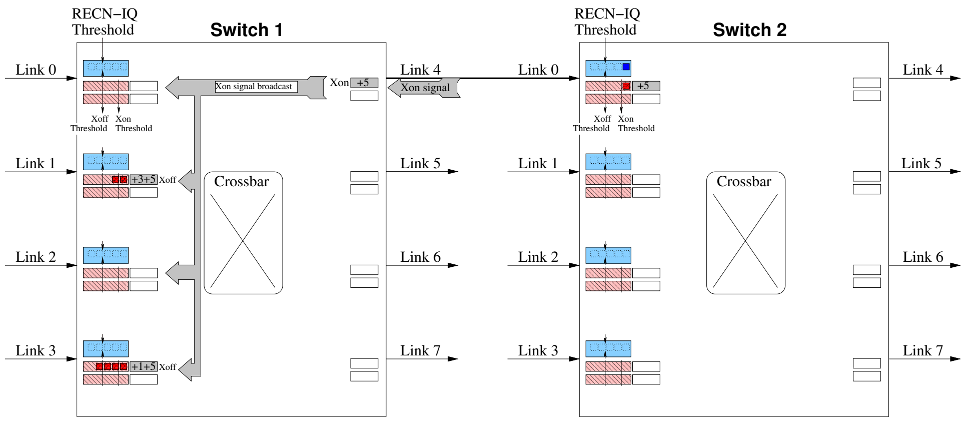


Figure 3.13: Step 8 of 10 of the RECN-IQ basic procedure example.

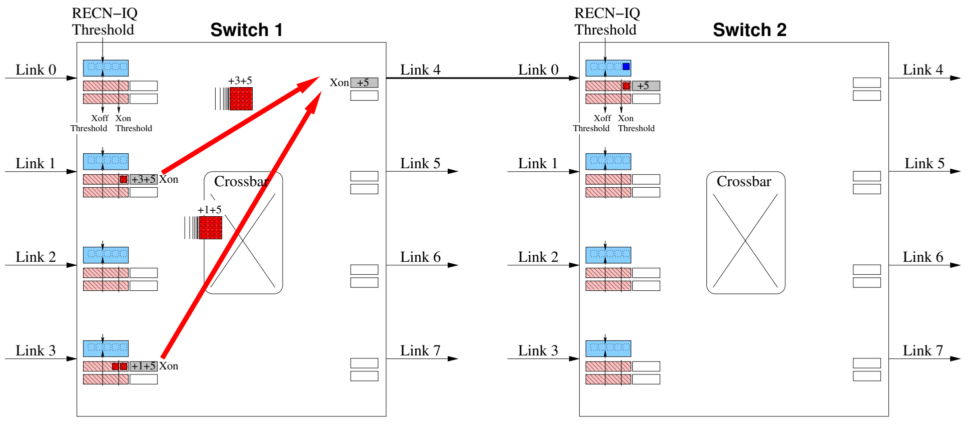


Figure 3.14: Step 9 of 10 of the RECN-IQ basic procedure example.

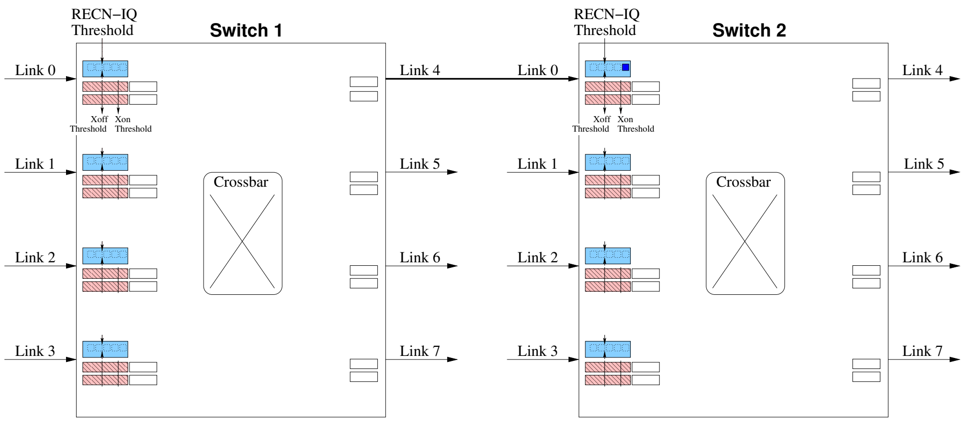


Figure 3.15: Step 10 of 10 of the RECN-IQ basic procedure example.

upstream output port (link 4 at switch 1).

Step 4 (Figure 3.9), upon reception of the Xoff signal at link 4 of switch 1, a new CAM line pointing to the congested point is activated. The contents of this CAM line are indicated by the Xoff signal and are the same as the ones at the downstream CAM line. When any input port of switch 1 sends a packet through output link 4, the turnpool of the packet is compared to the turnpool stored in the CAM line. In the case there is a match, the output port sends an Xoff signal to the input port sending that packet. In this example, input links 1 and 3 of switch 1 are sending congested packets through output link 4.

Step 5 (Figure 3.10), upon reception of the internal Xoff signal, each input port allocates a new SAQ+CAM line pointing to the detected congested point (at the switch downstream in this example). From input's port 1 of switch 1 perspective, the congested point is identified by the turns +3+5. On the other hand, from input's port 3 of switch 1 point of view, the congested point is +1+5.

Step 6 (Figure 3.11), a packet is moved into a SAQs only if there is a complete match between the remaining turns of that packet and the turn information at the CAM associated with the SAQ. Therefore, a packet going through output link 4 of switch 1 and then through link 6 of switch 2 (turn $+3+6$) is not considered congested, even if it shares the path with the identified congested route (identified by $+3+5$ here). Note that congested packets are being placed into the appropriate SAQ but this SAQ cannot send packets because it was created on an Xoff state. If at any time later, the SAQ is over the Xoff, the steps 3 to 5 are repeated until the congestion information (and the subsequent traffic separation) eventually reaches the sources.

Step 7 (Figure 3.12), at this point congestion at link 5 of switch 2 vanishes. The only SAQ sending traffic was the one at input port 0 of switch 2 (not on Xoff state). When the number of packets at that SAQ is below the Xon threshold, an Xon notification is sent to the output port of the switch upstream (link 4 of switch 1).

Step 8 (Figure 3.13), upon reception of the Xon signal at output link 4 of switch 1 the corresponding CAM switches its status from Xoff to Xon and broadcasts an internal Xon notification to all input ports in that switch.

Step 9 (Figure 3.14), upon reception of the Xon signal at the input ports of switch 1, if there is a match between the turn indicated by the Xon signal and an allocated SAQ, then the status of the matching CAM is switched from Xoff to Xon. At that moment, that SAQ is allowed to send congested packets. This scheme of Xon/Xoff flow control signals controls the movement of the congested packets at the SAQs.

Step 10 (Figure 3.15), whenever a SAQ is not on Xoff state and becomes empty, it can be de-allocated so it can be re-used for other upcoming congested flows.

3.3.8 False Positives when Detecting Congestion

There is still the question regarding what happens if, when congestion is detected, the packet at the header of the Cold Queue does not belong to a congested flow.

Imagine that this is indeed the case, then the RECN-IQ mechanism will allocate a SAQ for that non-congested point and thereafter will put the packet at the head of the CQ into that SAQ. Imagine now that the next packet at the CQ belongs to a congested flow. When it reaches the head of that queue, it is very likely that the packet will block because its requested output port is oversubscribed, thus producing HOL blocking. In that case the most probable is that the non-congested packet will gain the access to the crossbar (win the arbitration) versus the congested one at the CQ. Therefore, the SAQ will become empty at the next cycle so it can be de-allocated; whereas the blocked packet at the CQ is producing HOL blocking, keeping the number of packets in that queue over the threshold, triggering the congestion detection mechanism that will allocate another SAQ, this time pointing to the correct congested output port.

The conclusion is that the RECN-IQ mechanism overcomes the problem of miss-detections of congestion in a very straightforward way, with no significant penalty, due to the fact that the scheduler does not distinguish between requests from the CQ or the SAQs (they are all treated the same way, with no hard priority).

3.4 Conclusions

For modern interconnection networks, the use of an effective congestion management technique has become mandatory in order to keep network performance at maximum even in congestion situations. Although the formerly proposed RECN mechanism efficiently solves the problems related to congestion, its application is restricted to CIOQ switches, thereby not being suitable for the IQ switches.

In order to afford an effective congestion management technique to this type of switches, we have proposed in this chapter an adaptation of RECN to IQ switches. The resulting mechanism, referred to as RECN-IQ, also introduces a new way for detecting congestion at input ports that significantly reduces the data memory area required at each port. From the evaluation results presented in this chapter, we can deduce that RECN-IQ eliminates HOL blocking as well as RECN, while being an even more cost-effective technique.

In the next chapter the RECN-IQ congestion management mechanism is used to enhance a cheap and low-performance IQ switch architecture. The idea is that by adding some extra hardware (detailed in next chapter) for implementing RECN-IQ, the cost of the switch is kept low but with an impressive increase in performance.

Chapter 4

The RECN-IQ Switch Architecture

King Arthur: The swallow may fly south with the sun or the house martin or the plover may seek warmer climes in winter, yet these are not strangers to our land?
Soldier: Are you suggesting coconuts migrate?
– "Monty Python and the Holy Grail"

In this chapter, we propose an efficient and realistic switch architecture suitable for the new RECN mechanism, describing in detail the structure and behavior of each functional unit of this architecture. All the memory sizes, signals and components required for a real implementation are defined. Note that this is the first time a switch architecture implementing RECN is described at this level.

4.1 Description of the RECN-IQ Switch Architecture

In the previous chapter we have described the new RECN-IQ mechanism. Of course, this mechanism would influence many aspects (e.g., memory management, flow control issues, scheduler) of any switch supporting it. Therefore, we detail in this Section the entire switch organization required for implementing the RECN-IQ mechanism.

The switch architecture assumes a routing scheme similar to the one used in AS interconnects [42]. Thus, it is assumed that each packet includes in its packet header a turnpool and a turn pointer. The turnpool is made of 31 bits and contains a turn (offset from an input port to the requested output port) for each switch along the packet's path. The turn pointer, made of 5 bits, points to the set of bits of the turnpool encoding the turn for the current switch. Also, the switch architecture assumes VCT switching.

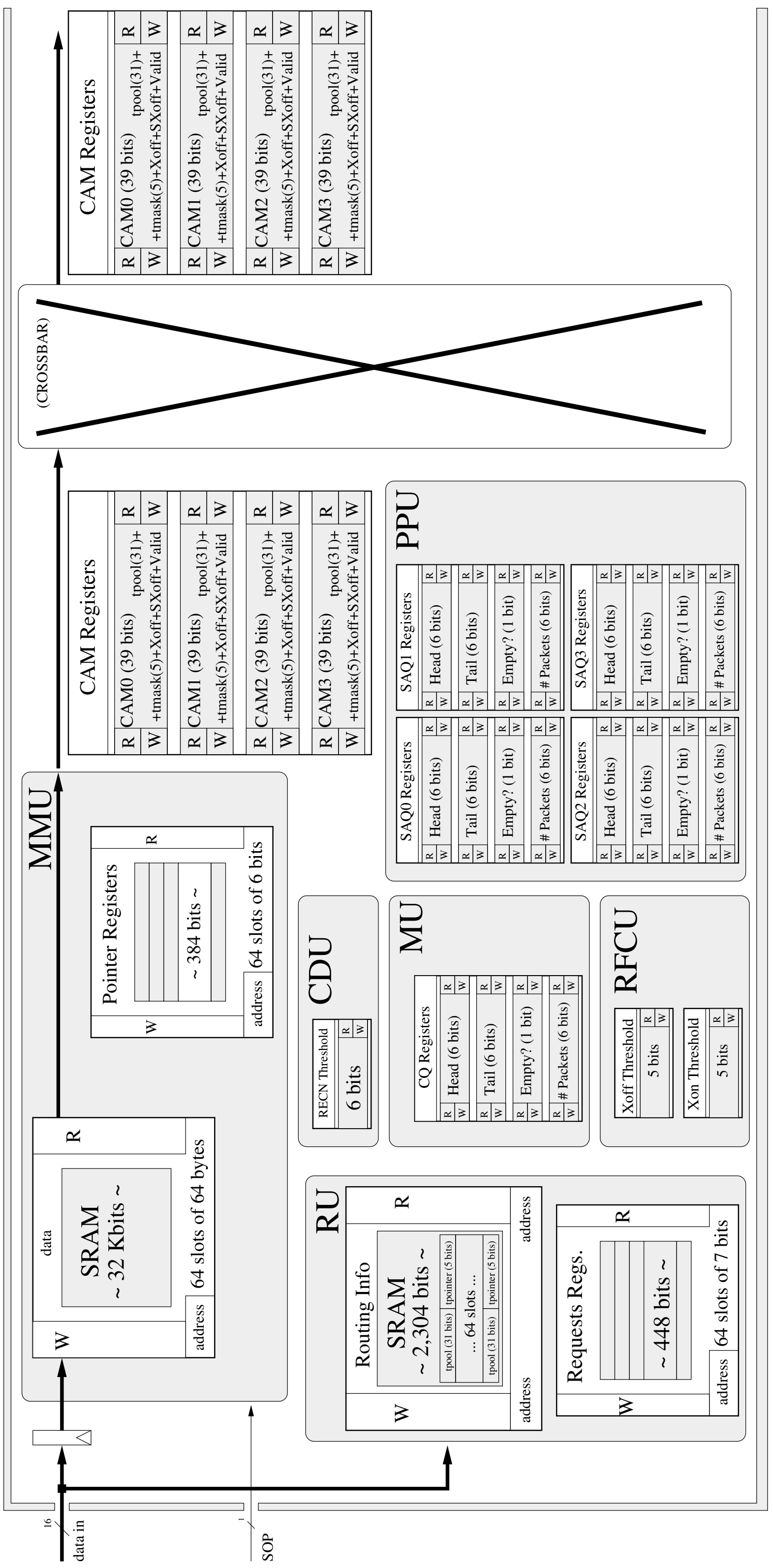


Figure 4.1: General overview of the proposed switch architecture.

Figure 4.1 shows a general overview of the switch architecture with RECN-IQ; depicting an input port, the crossbar and an output port. For the sake of an easier understanding, the switch has been divided into six functional units: Memory management (MMU), Mapping (MU), Post-Processing (PPU), Routing (RU), RECN-IQ Flow Control (RFCU), and Congestion Detection (CDU). In order to focus on the RECN-IQ mechanism and to avoid introducing graphical complexity, we do not include the arbiter in this Figure. Further in this Section we will discuss about the arbiter.

Also, the memory structures required by the mechanism and the switch architecture are shown in Figure 4.1. In addition to the SRAM memory that buffers incoming packets, an additional memory of 2Kb is required at each input port. This memory will store the routing headers of each packet. Also, two register files will be required: the Pointer Registers File (PRF) and the Requests Registers File (RRF). The PRF will store the pointers among different packets, in order to keep the logical structure of the queues. The RRF will store the requested output ports of the packets.

Additionally, besides some other small registers, a set of four registers is required for each implemented logic queue (either CQ or SAQ). These registers (*head*, *tail*, *empty* and *number of packets*) keep track of the queue structure and the queue occupancy level. Finally, a CAM structure per input port and another per output port are required by the RECN-IQ mechanism.

In the following sections we describe in detail each functional unit and all the required memory structures.

4.1.1 Memory Management Unit

In the proposed switch architecture packets are stored only at input ports. Thus, the Memory Management Unit (MMU) is located at the input side of the switch. The MMU is in charge of mapping all the incoming packets to the corresponding queue and to keep track of the allocated queues within the memory. Figure 4.2 shows the scheme for the MMU. This element works in a Dynamically Allocate Multi-Queue buffer (DAMQ) [14] fashion and it consists of a SRAM memory of 32 Kbits and an associated logic. The memory is divided into slots of 64 bytes (64 chunks of 64 bytes each). Slots are used atomically.

Whenever a new packet arrives (Start Of Packet signal, SOP), the writing logic selects a free slot in the memory for mapping the packet. In order to do this, the MMU incorporates two registers to keep track of the list of free slots, managed in a LIFO manner. Additionally, in order to keep track of the different queues implemented in the memory, the MMU incorporates a list of pointers, one for each possible slot. Thus, it incorporates 64 registers. Each register corresponds to a slot and indicates which is the next slot in the queue's order.

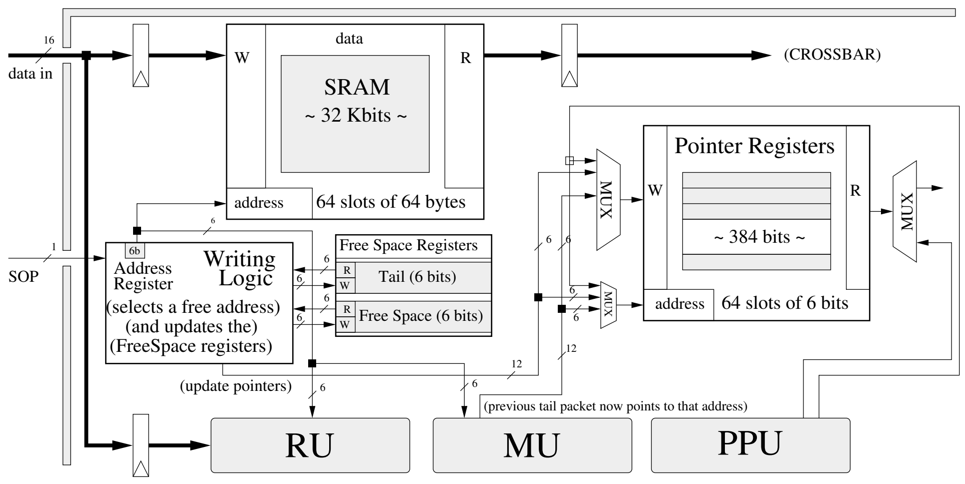


Figure 4.2: MMU (Memory Management Unit) detailed.

4.1.2 Mapping Unit

As already explained, in the RECN-IQ mechanism, the memory of each input port is logically divided into a single *Cold Queue* (CQ), and a set of SAQs for storing congested packets. In the proposed architecture, we assume four SAQs per port (this number of SAQs is enough for handling effectively the congestion situations and network configurations evaluated in Section 4.2). SAQs are dynamically allocated when required. Whenever a new packet arrives to an input port, it is stored always in the CQ queue. To keep track of the CQ, the Mapping Unit (MU) is used.

Figure 4.3 shows the scheme for the MU. It consists of four registers and an associated logic. The logic places the incoming packet in the CQ and updates the registers. The *head* and *tail* registers keep track of the structure of the CQ whereas the remaining two registers keep track of the queue's occupancy.

4.1.3 Routing Unit

At the same time packets are being stored into the CQ, the switch routes the packet in order to decide which output port the packet requests. For this, the Routing Unit (RU) shown in Figure 4.4 places the packet header in a separate SRAM memory of 2Kbits. This memory contains the header of each packet stored in the port, so it is divided into 64 slots. Each slot contains a turnpool and a turnpointer following the AS packet header format. Thus, each slot is 36 bits wide.

Additionally, the RU extracts from the header of the incoming packet the output port requested, and then stores that information into the *request register file* (RRF), made up of 64 registers 7 bits wide each. The RRF is

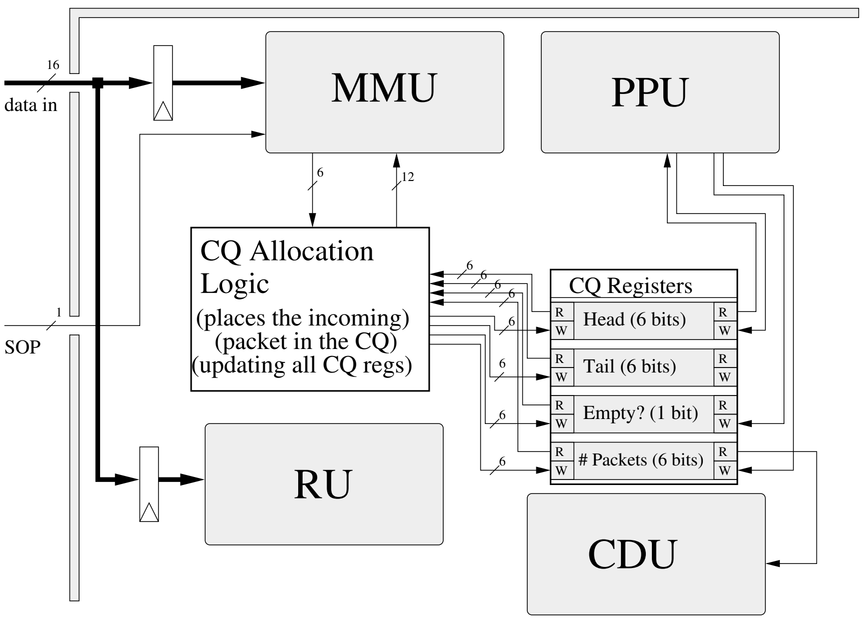


Figure 4.3: MU (Mapping Unit) detailed.

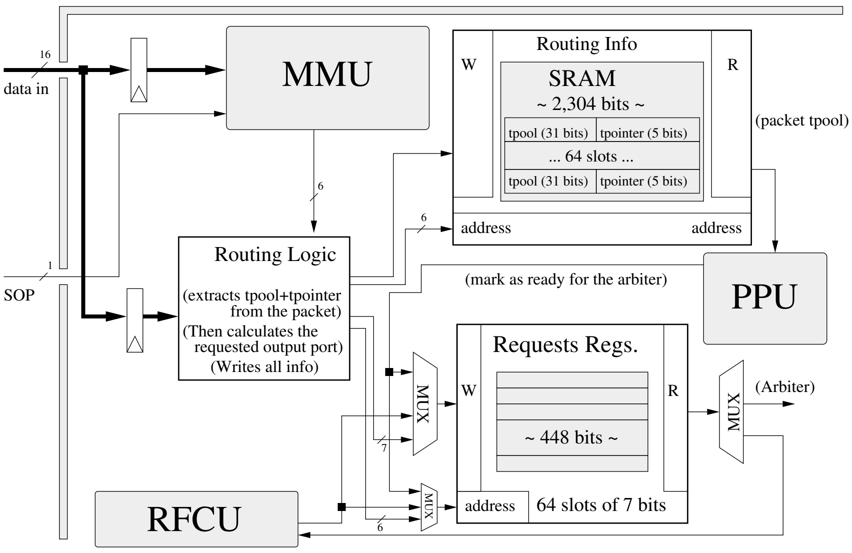


Figure 4.4: RU (Routing Unit) detailed.

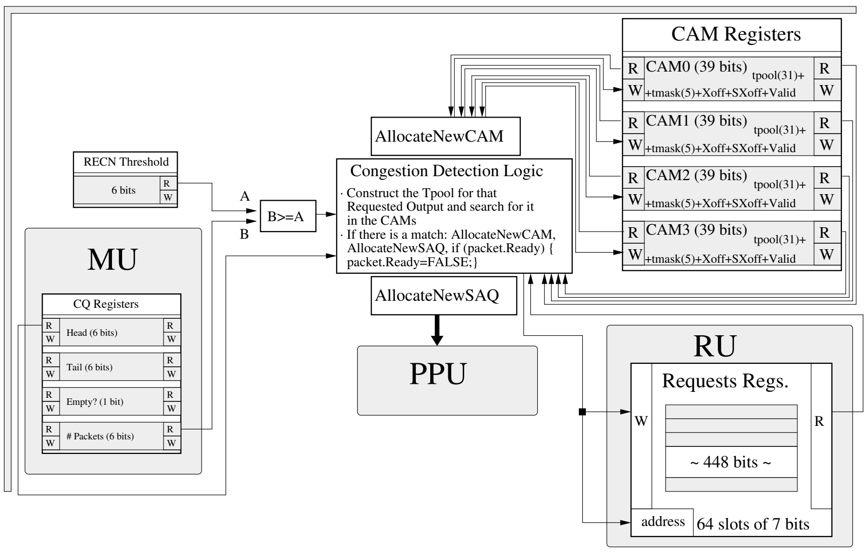

CAM Registers
R CAM0 (39 bits) tpool(31)+ R
W +tmask(5)+Xoff+SXoff+Valid W
R CAM1 (39 bits) tpool(31)+ R
W +tmask(5)+Xoff+SXoff+Valid W
R CAM2 (39 bits) tpool(31)+ R
W +tmask(5)+Xoff+SXoff+Valid W
R CAM3 (39 bits) tpool(31)+ R
W +tmask(5)+Xoff+SXoff+Valid W
AllocateNewCAM
RECN Threshold
6 bits
R
W
A
B
B>=A
Congestion Detection Logic
· Construct the Tpool for that Requested Output and search for it in the CAMs
· If there is a match: AllocateNewCAM, AllocateNewSAQ, if (packet.Ready) { packet.Ready=FALSE;}
AllocateNewSAQ
MU
CQ Registers
Head (6 bits)
Tail (6 bits)
Empty? (1 bit)
# Packets (6 bits)
PPU
RU
Requests Regs.
W
R
~ 448 bits ~
address
64 slots of 7 bits

Figure 4.5: CDU (Congestion Detection Unit) detailed.

inspected by the scheduler in order to decide when to forward the packet through the crossbar. However, one of these bits (the *ready-for-scheduler*, RFS bit) will be used in order to enable/disable the forwarding of the packet. Whenever a new packet arrives at the input port and is routed, the RFS bit is reset, thus the packet is disabled for being forwarded. Later, the PPU unit will set this bit, thus enabling the packet for the scheduler.

4.1.4 Congestion Detection Unit

The Congestion Detection Unit (CDU) shown in Figure 4.5 is in charge of detecting congestion, computing the output port congested, and, if required, allocating a new SAQ for the congested point. First, it compares the current queue's occupancy of the CQ against a given RECN threshold (measured in number of packets or slots). If the threshold is reached, then the congestion detection logic is triggered. This logic extracts from the *request registers file* (RRF) the output port requested by the packet at the header of the CQ. Then, it builds the turnpool and the bit mask addressing that output port, and compares this information against all the routing information stored in the CAM line of each active SAQ. In order to implement these CAM lines, the CDU incorporates a register for each possible SAQ. If there is no match, a new SAQ for the congested output port is allocated.

Notice that the logic associated to the CDU is enabled only when the detection threshold is reached. Thus, in the absence of congestion, most of the logic can be disabled.

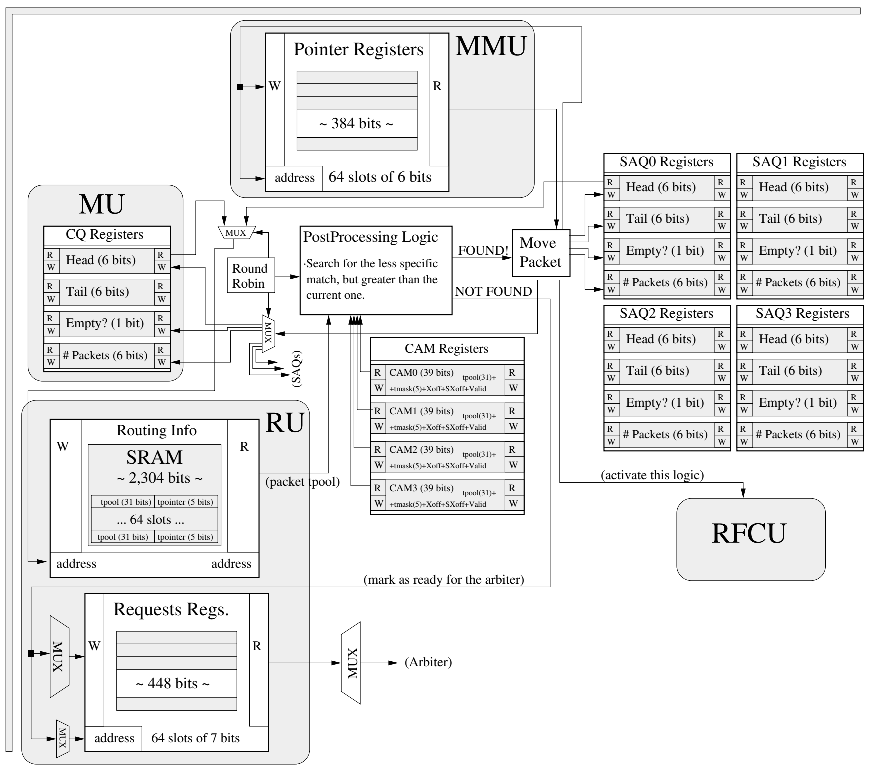


Figure 4.6: PPU (Post-Processing Unit) detailed.

4.1.5 Post-Processing Unit

The Post-Processing unit (PPU) shown in Figure 4.6 is in charge of separating the congested flows from the non-congested ones. Also, it separates every congested flow from each other.

In order to keep the logical queue structure of SAQs, the PPU requires some registers per SAQ (the same required for the CQ at the MU unit): *Head* and *tail* registers to keep queue's structure and two registers to keep queue's occupancy level.

The PPU works as a background task managing stored packets. Its main purpose is to classify packets according to the congested points already identified. To do this, the PPU continuously inspects the routing header (this information is located at the *routing info* memory at the RU) of each packet located at the head of any queue (CQ, SAQ0, SAQ1, SAQ2, and SAQ3). With this info, it checks if there is a match with any of the identified congested points. If so, the packet is moved to the SAQ associated to the congested point (it should be noted that the packet is not moved at all, only the pointers are updated).

Notice that the PPU may move packets from the CQ to a SAQ. In this case, the packet has been identified as passing through the congested point for which the SAQ has been allocated. The packet is simply moved to the tail of the SAQ. This is done by just adjusting the pointers of the CQ and the SAQ. Also, the *pointer registers file* (PRF) is updated accordingly.

However, note also that a packet stored at the head of a SAQ can be moved by the PPU unit. In this case, the unit moves the packet only if there is a match of the turnpool of the packet with the routing info associated to another SAQ allocated for a more specific congested point in the network (with a longer turnpool match).

Whenever a packet gets postprocessed (it is treated by the PPU unit), the RFS bit of the packet (located at the RRF file) is written. If the packet is moved (either from the CQ or from a SAQ) to a new queue, then, its RFS bit is reset. On the contrary, if the packet is not moved by the PPU unit, then its RFS bit is set, thus allowing the scheduler to forward the packet through the crossbar.

The PPU can be highly improved by disabling it for low network loads, thus avoiding the contribution of this hardware element to the latency. With this improvement, whenever the number of packets at the CQ is under the RECN-IQ threshold (no congestion) and there are no allocated SAQs at that input port, then the packets can be forwarded immediately. The moment a SAQs is allocated or the number of packets at the CQ reaches the RECN-IQ threshold, then the PP mechanism starts working the way previously described. In Section 4.2, some experimental results of this improvement are shown.

4.1.6 Flow Control Unit

The RECN-IQ mechanism implements Xon/Xoff (Stop & Go) flow control for the SAQs. As seen in Section 3.3, these flow control signals are used as notifications of congestion (propagation or vanishing) within the network as well. The RECN-IQ Flow Control unit (RFCU) is shown in Figure 4.7.

Each time a packet is moved into a SAQ by the Post-processing unit, the RFCU logic is activated in order to check whether the occupancy of the receiving SAQ goes over the Xoff threshold, and also to check whether the occupancy of the sending SAQ (in the case of packets being moved from a SAQ) goes below the Xon threshold. Therefore, the RFCU compares the number of packets of each SAQ against the Xoff (Xon) threshold. If the SAQ occupancy is over (below) the Xoff (Xon) threshold, and the SentXoff bit is unset (set), an Xoff (Xon) signal is sent backwards to the output port in the switch upstream. Then, the SentXoff bit for that SAQ is set (unset).

4.1.7 Global Flow Control and Scheduler

Although SAQs are individually flow controlled, the switch implements also a general (memory level) flow control mechanism. That is, each input port

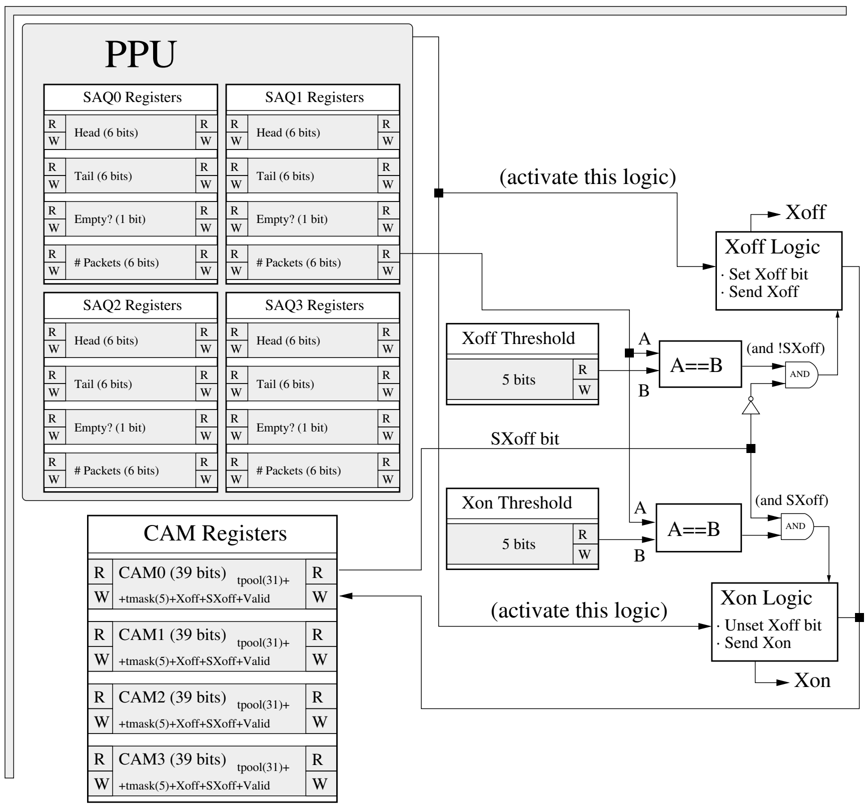


Figure 4.7: RFCU (RECN-IQ Flow Control Unit) detailed.

memory has a number of credits available for packets. The number of credits is the number of slots available in the memory, regardless of the receiving queue. Therefore, a packet will be transmitted over a link only if the receiving memory has an available slot (a credit) and the transmitting queue is not blocked (in the case the transmitting queue is a SAQ and its associated CAM has the "Xoff" bit set). Once a packet is transmitted the switch decrements the number of available credits at the downstream memory. Hence, at each output port the switch implements a counter of the number of available credits.

Therefore, the scheduler must take the credit counters at each output port into account when scheduling packets for transmission. Besides the credit counters, also, the scheduler will take into account the RFS bit of each packet and the Xoff bits of each CAM line associated to an active SAQ.

Regarding the arbiter, the switch architecture presented up to now is independent of the algorithm used to implement the arbiter. An arbiter based on the iSLIP scheduling algorithm has been selected so it does not represent the bottleneck of the switch architectures analyzed [46]. The iSLIP arbiter should perform enough iterations to converge to a maximal match. As long as RECN-IQ use extra SAQs that play the same role as the VOQ queues used in [46] and the number of SAQs should be always lower than the number of ports of the switch (less pairs to match), the number of iterations required would be even less than $\log_2 N$, thus the hardware requirements of RECN-IQ will be simpler.

4.2 Evaluation of RECN-IQ

In this section, the RECN-IQ mechanism is evaluated by means of simulation results. Specifically, we plot the network accepted traffic and network latency in a Burton Normal Form (BNF) graph [8]. Also, results of switch efficiency as a function of time are presented. These metrics have been measured for different values of the maximum number of SAQs available at input ports. Specifically, we have considered 2, 4 and 8 SAQs for simulating different RECN-IQ configurations and also no SAQs for simulating switches not using RECN-IQ. For comparison, a VOQ at the network level (VOQnet) has been also modeled. As long as the number of queues per input port cannot match all the possible destinations within the network, only the same number of queues for VOQ as the RECN-IQ mechanism uses for SAQs is used (packets are modulo-allocated at the VOQs at each switch according to their final destination).

Two different synthetic traffic patterns are used: uniform and hot-spot. And regarding network size and topology, the following two Multi-stage Interconnection Network (MIN) configurations have been analyzed, both for connecting 256 end nodes:

- Configuration 1: 256 end nodes, MIN made of 8x8 switches (256 switches in 4 stages, perfect shuffle as interconnection pattern).
- Configuration 2: 256 end nodes, MIN made of 32x32 switches (32 switches in 2 stages, perfect shuffle as interconnection pattern).

The organization of all switches in those configurations is the one described in Section 4.1, i.e., IQ switches in which RECN-IQ can be enabled/disabled. Other common parameters used in all the simulations are:

- Input Memory Size=4KB (64 packets)
- Packet Length=64 bytes
- Xon Threshold=2 packets
- Xoff Threshold=5 packets
- Congestion Detection Threshold=4 packets.

In the case of the VOQnet-like simulations, the same input memory size is used but it is divided evenly among all VOQ queues implemented on it. The bandwidth of all links is 1GByte/s and 4 nanoseconds is the inter-switch link delay.

4.2.1 Results for Uniform Traffic

Figures 4.8 and 4.9 depict the simulation results obtained for both MIN configurations. When the radix of the switches is low (8×8 at Figure 4.8), the maximum throughput achieved by a network made of basic IQ switches is about 63% due to the HOL blocking problem. By using 2 VOQnet-like extra queues per input port at each switch, the throughput jumps to 77%, but at the same cost in terms of extra queues, when RECN-IQ and 2 extra SAQs is used, the maximum throughput goes as high as 84%. To achieve such a performance, the VOQnet technology requires to divide the input memories into at least 4 extra queues. When RECN-IQ is implemented with 4 or 8 SAQs, then the maximum throughput is over 90%.

If we increase the radix of the switches (32×32 at Figure 4.9), then more extra queues are required in order to achieve high network throughput. In this case, having RECN-IQ with 4 SAQs is almost equivalent to having 8 extra VOQnet queues for uniform traffic (90% of the maximum throughput), thus with evident benefits in input port memory reduction. With RECN-IQ and 8 SAQs, a maximum throughput of 92% can be achieved.

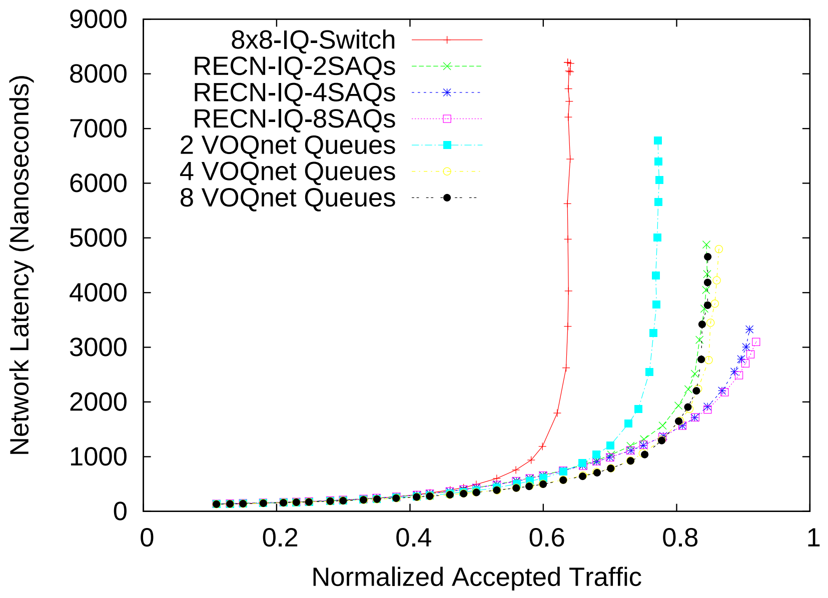


Figure 4.8: Accepted traffic versus network latency for an 8 × 8 network. Uniform distribution of packet destinations.

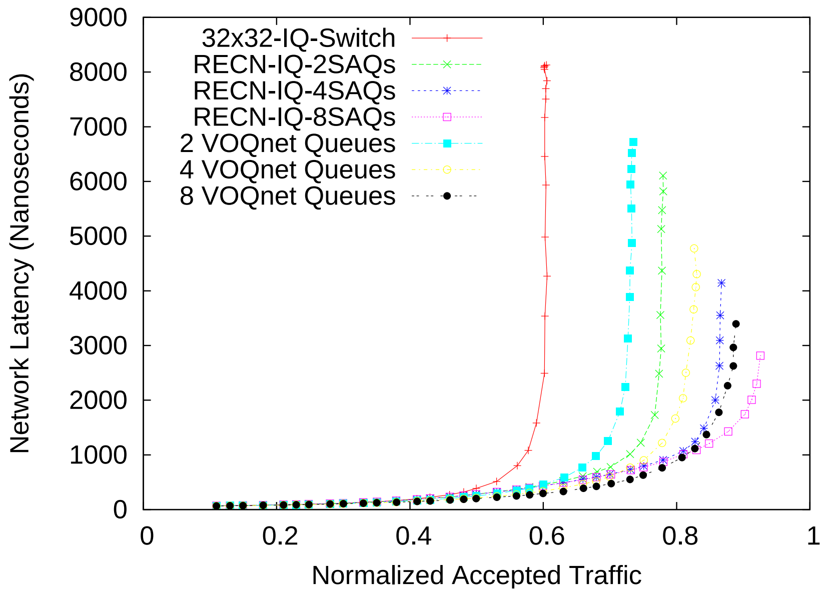


Figure 4.9: Accepted traffic versus network latency for an 32 × 32 network. Uniform distribution of packet destinations.

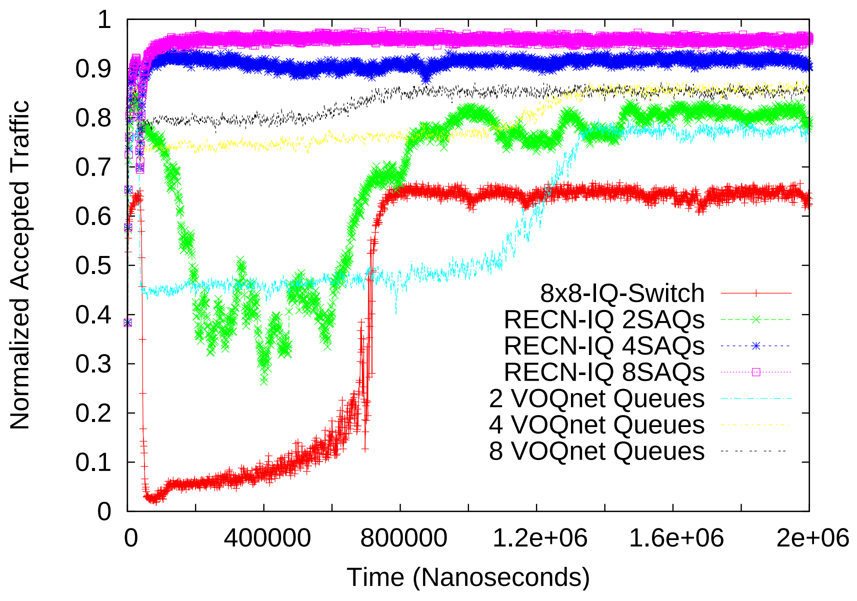


Figure 4.10: Switch efficiency versus time for Hot-Spot traffic for an 8×8 network.

4.2.2 Results for Hot-Spot Traffic

We have also conducted experiments using a hot-spot traffic pattern. Specifically, in this case all the end nodes inject 50% of their traffic to a single (hot-spot) end node (end node number 2 in this very case), whereas the rest of the traffic is randomly distributed among the rest of end nodes. Furthermore, the hot-spot traffic injection is only active during just 1 microsecond after 25 microseconds of simulation. As we will see, such a small hot-spot situation affects dramatically the network performance unless RECN-IQ is used.

Figures 4.10 and 4.11 show the switch efficiency versus time for MIN configurations 1 and 2 respectively. As can be seen in the case of 8×8 switches (Figure 4.10), after injecting the hot-spot traffic, the switch efficiency for the basic IQ switch architecture drops from the usual 63% for uniform traffic to about 3% due to the massive HOL blocking introduced throughout the network, and this situation lasts for more than 600 microseconds (for a hot-spot injection of just 1 microsecond).

On the other hand, when RECN-IQ with 4 or 8 SAQs is used, the switch efficiency is not affected by the introduction of the hot-spot situation. Immediately after the hot-spot injection ceases, the throughput of the network begins to recover as if nothing had happened. The solution of using a VOQnet-like mechanism, fails to keep the performance under this traffic situation, it requires long time to recover because complete paths are blocked

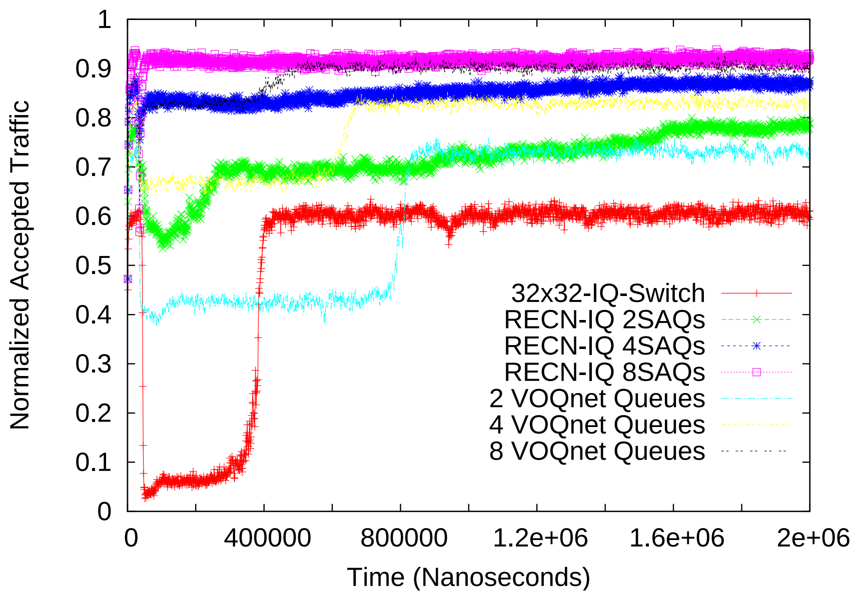


Figure 4.11: Switch efficiency versus time for Hot-Spot traffic for an 32×32 network.

by congested packets. This result was evident as long as a true congestion management technique is required to deal with that type of traffic, just like RECN-IQ does.

Similar results for a network made of 32×32 switches (Configuration 1) are shown in Figure 4.11. Unless RECN-IQ with at least 4 SAQs is used, the introduction of such a tiny (in terms of time) hot-spot degrades the throughput for long time, leaving the network into a situation of unacceptable performance.

4.2.3 Reducing the Network Latency of RECN-IQ for Low Network Loads

As explained in Section 4.1, the PPU unit can be disabled for low network loads, reducing the packet latency introduced by this unit. In this enhanced version of RECN-IQ, the PPU unit is disabled and packets are immediately forwarded by the input port whenever the number of packets at the *cold queue* (CQ) is below the RECN-IQ threshold and there are no SAQs allocated. Otherwise, the PPU unit works as previously described.

This solution is very useful specially when the PPU logic introduces high latencies compared with the rest of the switch logical elements. As example, we have carried out an experiment where we have chosen 10 nanoseconds for the PPU to process a packet. The rest of the parameters are exactly the same as described at the start of this Section. The network connects 256

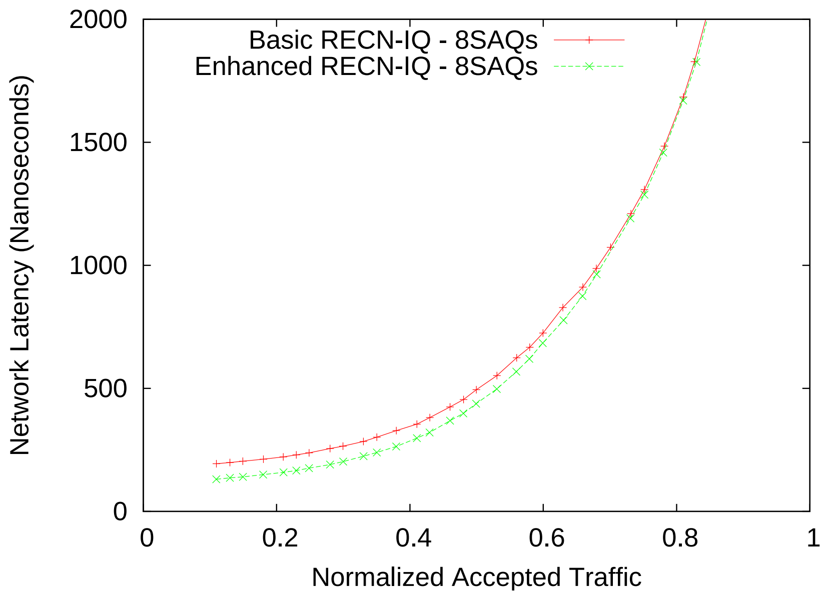


Figure 4.12: Enhanced RECN-IQ without PPU for low loads.

nodes through a MIN made of 8×8 switches and just 8 SAQs are used.

Figure 4.12 shows how the network latency is about 60 nanoseconds lower for this enhanced version of RECN-IQ compared to the basic RECN-IQ for low network loads (up to 50%). As the network load increases, this difference vanishes.

4.2.4 Impact of the Number of iSLIP Iterations on RECN-IQ Performance

We carried out an experiment for evaluating the impact of the number of iSLIP iterations on the RECN-IQ performance. It is well known that by increasing the number of iSLIP iterations, the number of input-output connections established increases up to a maximal (beyond which any extra iteration is useless). In the case of having N VOQs per input port (for an $N \times N$ switch), it is stated in [46] that the required number of iterations in order to achieve that maximal matching is about $\log_2 N$, that means 5 iterations on a 32×32 switch.

Figure 4.13 shows the results for a single switch of size 32×32, uniform traffic, and different number of iSLIP iterations and SAQs. The rest of parameters like timings and packet size are identical to the ones stated at the start of this section.

As can be seen in the experimental results, the improvement obtained when increasing the number of iterations is almost negligible. The maximal matching has been achieved with few iterations, even with just one. The

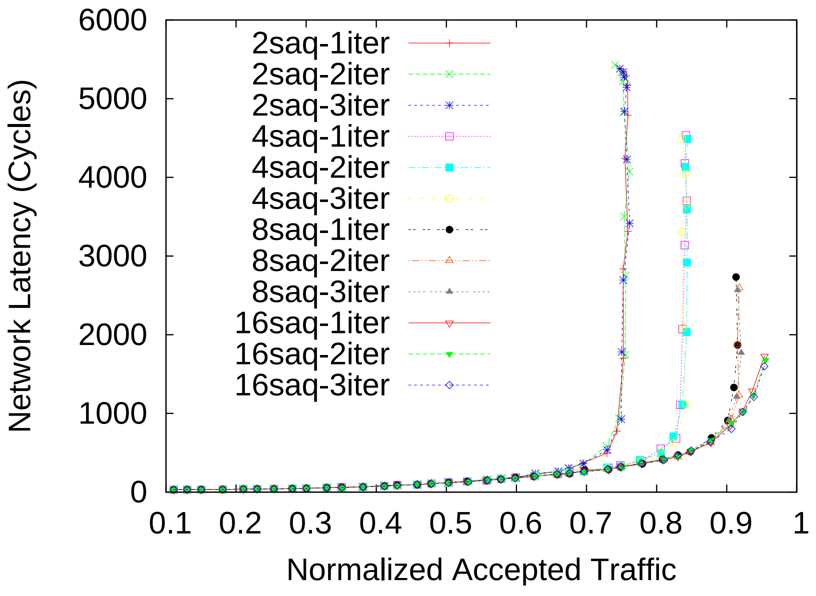


Figure 4.13: Impact of iSLIP iterations on RECN-IQ performance.

conclusion is that, because RECN-IQ does not use VOQ at the input ports but a reduced number of SAQs compared with the number of ports of the switch, the requirement of up to $\log_2 N$ iSLIP iterations in an $N \times N$ switch intuitively translates into a requirement of up to $\log_2 S$ iterations to achieve the maximal matching (S is the number of SAQs per input port). This means a simpler hardware implementation of RECN-IQ switch architecture.

4.3 Conclusions

A feasible, realistic switch architecture implementing RECN-IQ has been proposed and described in detail in this chapter. We covered every functional unit and structure required to implement RECN-IQ on an Input-Queued switch architecture, therefore turning realistic (by first time since its proposal in 2005 [34]) the use of a RECN-like technique.

Moreover, this architecture, RECN-IQ, could be the base for switches which would allow to build cheaper networks featuring congestion management. The benefits are huge and would make any network made of these kind of switches more predictable, ensuring performance at acceptable levels at any time and under any traffic circumstances.

Chapter 5

Conclusion and Future Work

Oh wait, you're serious. Let me laugh even harder.
– Bender, "Futurama",

As the optimal radix for switches increases due to the benefits in lower latencies, overall reduction in cost and power consumption; the traditional switch architectures are no longer valid because of either low-performance or non-scalability with the number of ports.

This dissertation proposes a new switch architecture suitable for high-radix switches called *Partitioned Crossbar Input Queued* (PCIQ) that deals with one of the main constraints in high-radix switch design, the excessive memory requirements. Also, in general terms, PCIQ forms a new family of switch microarchitectures as we have seen in Chapter 2.

PCIQ relies on a smart partition of the crossbar into sub-crossbars, thus requiring less memory resources than other proposals for high-radix, yet obtaining high-performance and also increasing the arbiter efficiency. PCIQ uses two round-robin packet-based arbiters (one for each crossbar) that exhibit a linear cost and a logarithmic response time as the radix of the switch increases.

In Chapter 2 it is shown that PCIQ exhibits a cost (measured in terms of memory requirements, crossbar complexity and arbiter complexity) similar to or lower than basic organizations like CIOQ. However, it is able to achieve maximum switch efficiency for uniform traffic distribution, thus leveling costly organizations like BC.

As we have seen, the other big issue on high-radix switches is the HOL blocking problem, which reduces dramatically the switch performance. Traditional solutions for removing the HOL blocking problem were based on VOQ schemes, but having high number of ports on a high-radix switch prevents the use of any of them. In this dissertation, a new congestion management technique has been proposed. This solution is called RECN-IQ, is specific for IQ switches and differs from the original RECN idea (suitable only for CIOQ switches) in being highly efficient and simple to implement, reducing the memory requirements to the maximum. RECN-IQ introduced by first time a novel statistical approach for detecting congestion using just a single queue per input port.

By combining the PCIQ microarchitecture with RECN-IQ, a new switch architecture (called here PCIQ-enhanced) is derived and evaluated in Chapter 2. The PCIQ switch architecture inherits the benefits of the Partitioned Crossbar microarchitecture in reducing the memory requirements for high-radix designs with the power of a congestion management technique that removes the HOL blocking dynamically, thus achieving maximum switch performance under all types of traffic.

We have seen that in modern interconnection networks it is mandatory the use of an effective congestion management technique in order to keep network performance at maximum level under congestion situations. Therefore, in Chapter 3 we describe the new congestion management technique (RECN-IQ) suitable for any type of IQ switches (which includes PCIQ). The idea behind RECN-IQ is, starting with a simple IQ switch with a single queue per input port, to add some extra queues dynamically allocated for congested packets. Congestion is detected as soon as HOL blocking begins to act, setting aside (in those extra queues known as SAQs) the congested packets in an efficient manner. Therefore, HOL blocking is completely eliminated (as proven by the simulation results shown in Chapter 4). The hardware requirements for RECN-IQ, as we have seen, are reduced, making feasible its implementation on any IQ-based switch architecture like PCIQ.

In order to prove that fact, a feasible and realistic switch architecture implementing RECN-IQ has been proposed and described in detail in Chapter 4, where we have detailed every functional unit and structure required to implement RECN-IQ on an Input-Queued switch architecture. This is the first time since the RECN proposal back in 2005 [34] that a RECN-like congestion management technique has been implemented in such a detailed level.

Results in Chapter 4 proved that by using RECN-IQ switches, the network will benefit from low cost switches and high-efficiency under any type of traffic pattern or network circumstances. All this makes the network predictable and stable in performance, no more drops in throughput because of congestion.

5.1 Future Directions

Regarding RECN-IQ as a congestion management technique, it can be used together with power consumption reduction techniques that rely on either disconnecting links or reducing their frequency, so the per-link bandwidth can be increased/reduced at will.

Because RECN-IQ is aware in a distributed manner of the network congestion, it can be used as an heuristic to increase/decrease the link bandwidth, thus saving power.

One big problem on saving energy by reducing link bandwidth is due to the characteristic burstiness of traffic. When a burst of traffic shows up, the link bandwidth can be in low power mode so the available bandwidth is not enough to deal with that traffic and the network enters in saturation with a huge drop in performance. When the power saving mechanism acts it is too late. By using RECN-IQ both for activating the links and as a congestion management technique, congestion never affects throughput. The transition between high load and low load will be done in a graceful way.

Appendix A

Contributions

While conducting the research necessary for this work I had the opportunity to publish some papers, having the chance of receiving very useful feedback from many reviewers. Here it is the list of all publications in which I have participated during my time as PhD student.

- G. Mora, P.J. Garcia, J. Flich and J. Duato. "*RECN-IQ: A Cost-Effective Input- Queued Switch Architecture with Congestion Management*". 2007 International Conference on Parallel Processing (ICPP-07). XiAn China, September 10-14, 2007. ISBN 0-7695-2933X. ISSN 0190-3918

- G. Mora, P. J. Garcia, J. Flich and J. Duato. "*The RECN-IQ Switch Architecture*". XVIII Jornadas de Paralelismo. Zaragoza (Spain). September 2007. ISBN 978-84-9732- 672-8. Pages 205-212

- G. Mora, J. Flich, J. Duato, E. Baydal, P. López and O. Lysne. "*Towards an Efficient Switch Architecture for High-Radix Switches*". ACM/IEEE Symposium on Architectures for Networking and Communications Systems. San Jose, CA, December 3-5, 2006

- G. Mora, J. Flich, J. Duato and O. Lysne. "*Partitioned Crossbar: Una organización de conmutador eficiente*". XVII Jornadas de Paralelismo. Albacete. September 2006. ISBN 84-690-0551-0. Pages 139-144.

- G. Mora, J. Flich and J. Duato. "*Report on new switch architectures and RECN with postprocessing, no tokens, and egress buffer-less queues*". September 2006. Research Report for Xyratex (http://www.xyratex.com)

- G. Mora, J. Flich, J. Duato, E. Baydal, P. López and O. Lysne. "*Towards an Efficient o Switch Architecture for High-Radix Switches*". Advanced Computer Architecture and Compilation for Embedded Systems. L'Aquila (Italy), July 26, 2006. Published by the HiPEAC Network of Excellence. ISBN 90 382 0981 9. Pages 257 - 260.

- J. Flich, G. Mora, J. Duato, E. Baydal, P. López and O. Lysne. "*PCIQ: An Efficient o and Cost-Effective Architecture for High-Radix Switches*". Advanced Networking and Communications Hardware Workshop. Boston, MA, June 18, 2006. Pages 3 - 12.

- G. Mora, J. Flich and J. Duato. "*Report on Post-Processing Approach over "No-Tokens RECN" Simulator*". June 2005. Research Report for Xyratex (http://www.xyratex.com).

The following list acknowledges the projects I participated and that funded all this research:

- "*Spanish CICYT*" (Spain) under Grant TIC2003- 08154-C06.

- "*Universidad Politécnica de Valencia*" (Spain) under Grant 20040937.

- "*Junta de Comunidades de Castilla-La Mancha*" (Spain) under Grant PBC-05-005-2.

- "*Spanish MEC*" (Spain) under Grant TIN2006-15516-C04.

- "*CONSOLIDER-INGENIO 2010*" (Spain) under Grant CSD2006-00046.

Appendix B

Summary of this PhD in Local Languages

B.1 Spanish

De cara a beneficiarse de una reducción en la latencia así como disminuir tanto el consumo como el coste, el número óptimo de puertos de un conmutador ha ido aumentando a lo largo del tiempo. Sin embargo, las arquitecturas tradicionales se han quedado atrás bien por bajo rendimiento o bien por problemas de escalabilidad con el número de puertos.

En esta Tesis se propone una nueva arquitectura de conmutador válida para conmutadores de elevado grado llamada *Partitioned Crossbar Input Queued* (PCIQ). Esta arquitectura resuelve una de los mayores problemas en el diseño de arquitecturas de elevado grado, es decir, el excesivo requerimiento de memoria. Además, ya en términos generales, PCIQ forma una nueva familia de arquitecturas de conmutador.

PCIQ se basa en un particionado inteligente del *crossbar*, dividiéndolo en *sub-crossbars*, por lo que se requieren menos recursos de memoria que las otras propuestas para conmutadores de elevado grado y que consigue una mayor eficiencia debido en parte a un incremento en la eficiencia de los árbitros empleados en el diseño. En este sentido, PCIQ emplea dos árbitros con prioridad rotativa (uno para cada *sub-crossbar*) que presentan un coste lineal y una respuesta en el tiempo logarítmica conforme aumenta el número de puertos del conmutador.

En este trabajo, se muestra que PCIQ tiene un coste (medido en términos de requerimientos de memoria, complejidad del crossbar y complejidad en el arbitraje) similar o incluso menor que organizaciones básicas como CIOQ. Sin embargo, con este reducido coste, PCIQ es capaz de conseguir máxima eficiencia para distribuciones de tráfico uniforme.

Otro gran problema a resolver en los conmutadores de elevado grado es el bloqueo por paquete al principio de cola (o HOL en inglés), bloqueo que reduce dramáticamente el rendimiento del conmutador. Las soluciones tradicionales para eliminar el bloqueo por HOL están basadas en *Virtual Output Queueing* (VOQ), pero dado que en nuestro caso el número de puertos es elevado, estas soluciones están totalmente descartadas. En esta Tesis se propone una técnica de control de la congestión que elimina el bloqueo por HOL llamada RECN-IQ. RECN-IQ está diseñada para conmutadores con memorias sólo a la entrada y es una técnica altamente eficiente y sencilla de implementar, con unos requerimientos extras de memoria mínimos. Además introduce una técnica novedosa basada en estadística para detectar congestión empleando tan sólo una cola por puerto de entrada en el conmutador.

La combinación de la arquitectura PCIQ con RECN-IQ para eliminar el bloqueo por HOL es descrita y evaluada en esta Tesis. Esta combinación se conoce como PCIQ-Enhanced y une los beneficios del particionado del *crossbar* en cuanto a la reducción de memoria con la potencia de una técnica de control de la congestión que elimina el bloqueo por HOL, obteniéndose máximas prestaciones bajo cualquier tipo de tráfico.

Esta nueva técnica de control de la congestión, RECN-IQ, se describe en profundidad en esta Tesis. La idea detrás de RECN-IQ es que, partiendo de un conmutador sencillo con memoria sólo a la entrada, se añaden unas pocas colas extra que son manejadas dinámicamente para colocar los paquetes congestionados. La congestión se detecta tan pronto como el bloqueo por HOL comienza a aparecer, de modo que es inmediatamente separada del flujo normal y puesta en esas colas extra llamadas SAQs.

Para probar que el mecanismo RECN-IQ elimina completamente el bloqueo por HOL y que es viable su implementación en cualquier arquitectura de conmutador con colas sólo a la entrada, en esta Tesis se propone su aplicación a un conmutador simple de este tipo. Para ello se describe todo elemento funcional y estructura lógica (y de memoria) requerido para la implementación de RECN-IQ.

Los resultados prueban que, mediante el uso de RECN-IQ sobre una arquitectura de conmutador básica, la red se beneficia de conmutadores de bajo coste y alta eficiencia bajo cualquier tipo de tráfico o circunstancias de la red. Esto otorga a la red un comportamiento previsible y estable en rendimiento, sin caídas en productividad debidas a la congestión.

B.2 Catalan

De cara a beneficiar-se d'una reducció en la latència així com disminuir tant el consum com el cost, el nombre òptim de ports d'un commutador ha anat augmentant al llarg del temps. No obstant això, les arquitectures tradicionals s'han quedat arrere bé per baix rendiment o bé per problemes d'escalabilitat amb el nombre de ports.

En esta Tesi es proposa una nova arquitectura de commutador vàlida per a commutadors d'elevat grau anomenada Partitioned Crossbar Input Queued (PCIQ). Esta arquitectura resol un dels majors problemes en el disseny d'arquitectures d'elevat grau, és a dir, l'excessiu requeriment de memòria. A més, ja en termes generals, PCIQ forma una nova família d'arquitectures de commutador.

PCIQ es basa en un particionat intel·ligent del crossbar, dividint-ho en sub-crossbars, per la qual cosa es requereixen menys recursos de memòria que les altres propostes per a commutadors d'elevat grau i que aconseguix una major eficiència gràcies en part a un increment en l'eficiència dels àrbitres emprats en el disseny. En aquest sentit, PCIQ utilitza dos àrbitres amb prioritat Rotativa (un per a cada sub-crossbar) que presenten un cost lineal i una resposta en el temps logarítmica conforme augmenta el nombre de ports del commutador.

En aquest treball, es mostra que PCIQ té un cost (mesurat en termes de requeriments de memòria, complexitat del crossbar i complexitat en l'arbitratge) semblant o inclús menor que organitzacions bàsiques com CIOQ. No obstant això, amb aquest reduït cost, PCIQ és capaç d'aconseguir màxima eficiència per a distribucions de tràfic uniforme.

Un altre gran problema a resoldre en els commutadors d'elevat grau és el bloqueig per paquet al principi de cua (o HOL en anglés), bloqueig que reduïx dramàticament el rendiment del commutador. Les solucions tradicionals per a eliminar el bloqueig per HOL estan basades en Virtual Output Queueing (VOQ), però atés que en el nostre cas el nombre de ports és elevat, aquestes solucions estan totalment descartades. En esta Tesi es proposa una tècnica de control de la congestió que elimina el bloqueig per HOL anomenada RECN-IQ. RECN-IQ està dissenyada per a commutadors amb memòries només a l'entrada i és una tècnica altament eficient i senzilla d'implementar, amb uns requeriments extres de memòria mínims. A més introduïx una tècnica nova basada en estadística per a detectar congestió emprant tan sols una cua per port d'entrada en el commutador.

La combinació de l'arquitectura PCIQ amb RECN-IQ per a eliminar el bloqueig per HOL és descrita i avaluada en esta Tesi. Esta combinació es coneix com PCIQ-Enhanced i uneix els beneficis del particionat del crossbar quant a la reducció de memòria amb la potència d'una tècnica de control de la congestió que elimina el bloqueig per HOL, obtenint-se màximes prestacions sota qualsevol tipus de tràfic.

Esta nova tècnica de control de la congestió, RECN-IQ, es descriu en profunditat en esta Tesi. La idea darrere de RECN-IQ és que, partint d'un commutador senzill amb memòria només a l'entrada, s'afegeixen unes poques cues extra que són manejades dinàmicament per a col·locar els paquets congestionats. La congestió es detecta tan prompte com el bloqueig per HOL comença a aparéixer, de manera que és immediatament separada del flux normal i posada en eixes coles extra anomenades SAQs.

Per a provar que el mecanisme RECN-IQ elimina completament el bloqueig per HOL i que és viable la seua implementació en qualsevol arquitectura de commutador amb cues només a l'entrada, en esta Tesi es proposa la seua aplicació a un commutador simple d'este tipus. Per a això es descriu tot element funcional i estructura lògica (i de memòria) requerit per a la implementació de RECN-IQ.

Els resultats proven que, per mitjà de l'ús de RECN-IQ sobre una arquitectura de commutador bàsica, la xarxa es beneficia de commutadors de baix cost i alta eficiència sota qualsevol tipus de tràfic o circumstàncies de la xarxa. Açò atorga a la xarxa un comportament previsible i estable en rendiment, sense caigudes en productivitat degudes a la congestió.

Bibliography

[1] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks.* Morgan Kaufmann Publishers, 2004.

[2] G. Pfister, "An introduction to the infiniband architecture (http://www.infinibandta.com)," *IEEE Press*, 2001.

[3] A. Farazdel, G. Archondo-Callao, and F. V. E. Hocks, T. Sakachi, "Understanding and using the sp switch," *http://www.redbooks.ibm.com/redbooks/pdfs/sg245161.pdf*, April 1999.

[4] "Myrinet, 2000 series networking." Available at http://www.cspi.com/multicomputer/products/2000_series_networking/2000_networking.htm.

[5] "Quadrics qsnet." Available at http://doc.quadrics.com.

[6] "Top500 supercomputer sites." http://www.top500.org.

[7] "Ibm bg/l team: An overview of bluegene/l supercomputer," in *ACM Supercomputing Conference*, Nov. 2002.

[8] J. Duato, S. Yalamanchili, and L. M. Ni, *Interconnection Networks: An Engineering Approach.* Morgan Kaufmann Publishers, 2003.

[9] W. J. Dally and C. L. Seitz, "The torus routing chip," *Journal of Distributed Computing*, vol. 1, pp. 187–196, October 1986.

[10] M. Karol and M. Hluchyj, "Queuing in high-performance packet-switching," *IEEE J. Select. Areas. Commun*, vol. 6, pp. 1587–1597, Dec. 1998.

[11] N. McKeown, M. Izzard, A. Mekkittikul, W. Ellersick, and M. Horowitz, "The tiny tera: A packet switch core," *IEEE Micro*, vol. 17, pp. 27–33, Jan./Feb. 1997.

[12] M. J. Karol, M. G. Hluchyj, and S. P. Morgan, "Input versus output queueing on a space-division packet switch," *IEEE Transactions on Communications*, vol. COM-35, no. 12, pp. 1347–1356, 1987.

[13] T. Anderson, S. Owicki, J. Saxe, and C. Thacker, "High-speed switch scheduling for local-area networks," *ACM Transactions on Computer Systems*, vol. 11, pp. 319–352, Nov 1993.

[14] Y. Tamir and G. L. Frazier, "High-performance multi-queue buffers for vlsi communications switches," *SIGARCH Comput. Archit. News*, vol. 16, no. 2, pp. 343–354, 1988.

[15] B. Prabhakar and N. McKeown, "On the speedup required for combined input and output queued switching," *Stanford University Technical Report, STAN-CSL-TR-97-738*, Nov 1997.

[16] R. Rojas-Cessa, E. Oki, and H. Chao, "Cixob-k: Combined input-crosspoint-output buffered packet switch," *Proceedings of the IEEE Global Telecomunications Conference*, 2001.

[17] A. Agarwal, "Limits on interconnection network performance," *IEEE Transactions on Parallel Distributed Systems*, vol. 4, no. 2, pp. 398–412, 1991.

[18] W. J. Dally, "Performance analysis of k-ary n-cube interconnection networks," *IEEE Transactions on Computers*, vol. 39, no. 6, pp. 775–785, 1990.

[19] M. Noakes, D. Wallach, and W. Dally, "The j-machine multicomputer: An architectural evaluation," *Proceedings of the 20th International Symposium on Computer Architecture*, pp. 224–235, May 1993.

[20] S. Scott, "Synchronization and communication in the t3e multiprocessor," *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 26–36, October 1996.

[21] S. Scott and G. Thorson, "The cray t3e network: Adaptive routing in a high performance 3d torus," *Proceedings of Hot Interconnects Symposium IV*, August 1996.

[22] "Sgi altix 3000." http://www.sgi.com/products/servers/altix/.

[23] J. Kim, W. J. Dally, B. Towles, and A. K. Gupta, "Microarchitecture of a high-radix router," *32nd Annual International Symposium on Computer Architecture (ISCA '05)*, pp. 420–431, 2005.

[24] W. J. Dally, P. Hanrahan, M. Erez, T. J. Knight, F. Labonte, J. H. Ahn, N. Jayasena, U. J. Kapasi, A. Das, J. Gummaraju, and I. Buck, "Merrimac: Supercomputing with streams," *Proc. of Supercomputing, SC'03*, Nov 2003.

[25] R. Drost, R. Hopkins, and I. Sutherland, "Proximity communication," *IEEE Custom Integrated Circuits Conference*, pp. 469–472, Sep 2003.

[26] R. Drost, R. Hopkins, and I. Sutherland, "Electronic alignment for proximity communication," *IEEE Solid-State Circuits Conference*, pp. 144–158, Feb 2004.

[27] W. Olesinski, H. Eberle, and N. Gura, "Obig: the architecture of an output buffered switch with input groups for large switches," *IEEE GLOBECOM*, Nov 2007.

[28] L. Shang, L. S. Peh, and N. K. Jha, "Dynamic voltage scaling with links for power optimization of interconnection networks," *Proceedings of the International Symposium on High-Performance Computer Architecture*, pp. 91–102, February 2003.

[29] M. Wang, H. Siegel, M. Nichols, and S. Abraham, "Using a multipath network for reducing the effects of hot spots," *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, pp. 252–268, March 1995.

[30] M. Thottethodi, A. Lebeck, and S. Mukherjee, "Self-tuned congestion control for multiprocessor networks," *Proc. Int. Symp. High-Performance Computer Architecture*, Feb 2001.

[31] W. J. Dally, P. Carvey, and L. Dennison, "The avici terabit switch/router," *Proc. Hot Interconnects 6*, Aug 1998.

[32] Y. Tamir and G. Frazier, "Dynamically-allocated multi-queue buffers for vlsi communication switches," *IEEE Trans. on Computers*, vol. 41, June 1992.

[33] A. Smai and L. Thorelli, "Global reactive congestion control in multicomputer networks," *Proc. 5th Int. Conf. on High Performance Computing*, 1998.

[34] J. Duato, I. Johnson, J. Flich, F. Naven, P. J. Garcia, and T. Nachiondo, "A new scalable and cost-effective congestion management strategy for lossless multistage interconnection networks," *Proc. 11th International Symposium on High-Performance Computer Architecture (HPCA05)*, pp. 108–119, Feb 2005.

[35] P. J. Garcia, J. Flich, J. Duato, I. Johnson, F. Quiles, and F. Naven, "Efficient, scalable congestion management for interconnection networks," *IEEE Micro*, vol. 26, pp. 52–66, Sep 2006.

[36] W. J. Dally, "Virtual-channel flow control," *Proceedings of the 17th annual International Symposium on Computer Architecture*, pp. 60–68, 1990.

[37] G. Kornaros, C. Kozyrakis, P. Vatsolaki, and M. Katevenis, “Pipelined multi-queue management in a vlsi atm switch chip with credit-based flow-control,” *17th Conference on Advanced Research in VLSI (ARVLSI '97)*, 1997.

[38] H. J. Mattausch, “Hierarchical n-port memory architecture based on 1-port memory cells,” *Solid-State Circuits Conference, 1997. ESSCIRC '97. Proceedings of the 23rd European*, pp. 348–351, 1997.

[39] K. Johguchi, Z. Zhu, K. Aoyama, Y. Mukuda, H. J. Mattausch, T. Koide, and T. Hironaka, “Unified data/instruction cache with distributed crossbar, hidden precharge pipeline and dynamic cmos logic,” *Fourth Hiroshima International Workshop on Nanoelectronics for Tera-Bit Information Processing*, 2005.

[40] Z. Zhu, K. Johguchi, H. Mattausch, T. Koide, T. Hirakawa, and T. Hironaka, “A novel hierarchical multi-port cache,” *Solid-State Circuits Conference, 2003. ESSCIRC '03. Proceedings of the 29th European*, pp. 405 – 408, 2003.

[41] A. A. Chien, “A cost and speed model for k-ary n-cube wormhole routers,” *Proceedings of Hot Interconnects*, August 1993.

[42] “Advanced switching core architecture specification.” Available at http://www.asi-sig.org/specifications for ASI SIG.

[43] E. S. Shin, V. J. M. III, and G. F. Riley, “Round-robin arbiter design and generation,” *Proceedings of the 15th International Symposium on System Synthesis*, 2002.

[44] P. J. Garcia, J. Flich, J. Duato, I. Johnson, F. Quiles, and F. Naven, “Dynamic evolution of congestion trees: Analysis and impact on switch architecture,” *Proceedings of the 2005 International Conference on High Performance Embedded Architectures and Compilers*, Nov 2005.

[45] K. Pagiamtzis and A. Sheikholeslami, “Content-addressable memory (cam) circuits and architectures: A tutorial and survey,” *IEEE J. Solid-State Circuits*, vol. 41, pp. 712–727, Mar 2006.

[46] N. McKeown, “The islip scheduling algorithm for input-queued switches,” *IEEE/ACM Trans. Networking*, vol. 7, no. 2, pp. 188–201, 1999.