Alejandro Martínez Vicente

EFFICIENT QoS SUPPORT FOR HIGT-PERFORMANCE INTERCONNECTS

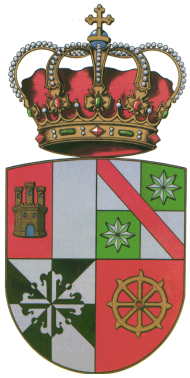
I.S.B.N. Ediciones de la UCLM
978-84-8427-640-1

Ediciones de la Universidad
de Castilla-La Mancha

Cuenca, 2008

UNIVERSIDAD DE CASTILLA-LA MANCHA

Departamento de Sistemas Informáticos



Efficient QoS Support for High-Performance Interconnects

Tesis Doctoral
presentada al Departamento de Sistemas Informáticos
de la Universidad de Castilla-La Mancha
para la obtención del título de
Doctor en Informática

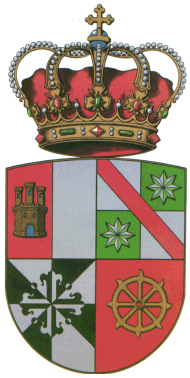
Presentada por:
Alejandro Martínez Vicente

Dirigida por:
Dr. D. Francisco José Alfaro Cortés
Dr. D. José Luis Sánchez García

Albacete, Junio de 2007

UNIVERSITY OF CASTILLA-LA MANCHA

Computing Systems Department



Efficient QoS Support for High-Performance Interconnects

A dissertation for the degree of Doctor of Philosophy in Computer Science to be presented with due permission of the Computing Systems Department, for public examination and debate.

Author:
Alejandro Martínez Vicente

Advisors:
Dr. Francisco José Alfaro Cortés
Dr. José Luis Sánchez García

Albacete, June 2007

A mis padres y mi hermana

Agradecimientos

Esta tesis no habría sido posible sin el apoyo y la ayuda de mucha gente que me rodea. En esta página quiero ofrecerles mi homenaje sincero.

En primer lugar, quiero agradecer su labor a mis directores, José Luis y Paco. No sólo no habría sido posible sin ellos el haber completado este trabajo, sino que también han sido, durante estos años, apoyo en los momentos de adversidad, ánimo cuando las cosas no salían, y amigos sinceros siempre.

También quiero agradecer a José Duato su apoyo. De él partieron las ideas iniciales y durante estos años siempre ha estado disponible para ofrecer buenos consejos e ideas. Sin duda, sin él tampoco habría sido posible esta tesis.

Tengo que agradecer también el apoyo prestado por mis compañeros del Departamento de Sistemas Informáticos y especialmente quiero mencionar a mis compañeros del grupo de Redes y Arquitecturas de Altas Prestaciones (RAAP). Gracias a su excelente trabajo, he disfrutado del apoyo material, financiero y humano necesarios para el desarrollo de mi trabajo.

También tengo que dar las gracias a todos los becarios del RAAP y del Instituto de Investigación en Informática de Albacete (I3A). Juntos hemos compartido excelentes momentos y hemos establecido una relación más allá del trabajo. Tampoco puedo dejar de mencionar a Pedro García, que además de hacer sustanciales aportaciones a esta tesis, ha sido un gran compañero y amigo.

Finalmente, quiero agradecer el amor, apoyo y comprensión de mi familia. En el día a día mis padres Ángel y Paqui y mi hermana Andrea han hecho todo esto posible. También quiero agradecer a mis abuelos, tíos y primos, con los que he compartido momentos maravillosos y que, independientemente de como iban las cosas en el ámbito profesional, me han dado su apoyo y cariño incondicionales.

Resumen

Las redes de interconexión son un componente clave en un gran número de sistemas. En la actualidad, son necesarias interconexiones de baja latencia y gran ancho de banda para la ejecución de aplicaciones en sistemas tales como supercomputadores, *clusters* de PCs y otros.

Hay muchos aspectos necesarios para obtener redes de interconexión de altas prestaciones satisfactorias. Entre ellos, la calidad de servicio (QoS) es la responsable de asegurar que se alcanza un cierto rendimiento. Dependiendo de las aplicaciones concretas, pueden ser necesarias garantías de lantecia, de productividad o de otro tipo.

Las soluciones tradicionales para ofrecer QoS en redes de interconexión de altas prestaciones normalmente se basan en arquitecturas complejas. Por ejemplo, los canales virtuales son un mecanismo que se propone con frecuencia para aislar diferentes clases de tráfico. Sin embargo, es poco habitual el ver en implementaciones finales tantos canales virtuales como aparecen en propuestas y especificaciones.

El principal objetivo de esta tesis es investigar si podemos ofrecer mecanismos eficientes de QoS. Nuestro propósito es alcanzar un soporte completo de QoS con el mínimo de recursos. Para ello, se identifican redundancias en los mecanismos propuestos de QoS y son eliminadas sin afectar al rendimiento.

Esta tesis consta de tres partes. En la primera comenzamos con las propuestas tradicionales de QoS a nivel de clase de tráfico. Nosotros proponemos un marco más eficiente de QoS que sólo necesita dos canales virtuales en los conmutadores. Esto es posible porque se aprovecha la planificación realizada en los nodos terminales.

En la segunda parte, proponemos cómo adaptar los mecanismos de QoS basados en *deadlines* para redes de interconexión de altas prestaciones. Utilizando un enfoque similar al de la parte anterior, obtenemos resultados notables con conmutadores de arquitectura simple.

Por último, también investigamos la interacción de los mecanismos de QoS con el control de congestión. Proponemos una solución integrada que se basa en la sinergia entre ambos aspectos de las redes de interconexión de altas prestaciones.

Summary

Interconnection networks are a key component of a variety of systems. Today, low-latency and contention-free interconnection networks are demanded for the execution of many applications in systems like supercomputers, clusters of PCs, and others.

There are many issues involved in successful high-performance interconnects. Among them, quality of service (QoS) is responsible of guaranteeing that a certain performance is achieved. Depending on the specific applications, guarantees on latency, throughput, and other indices may be critical.

Traditional solutions to provide QoS in high-performance interconnects usually rely on complex architectures. For instance, virtual channels are a mechanism that is often proposed to isolate several traffic classes. However, it is unusual to see in a final implementations as many virtual channels as in proposals.

The main objective of this thesis is to investigate if we can offer efficient mechanisms to provide QoS. Our purpose is to achieve a full QoS support with the minimum of resources. We do that by identifying redundancies in current proposals for QoS support and eliminating them.

This thesis consists of three parts. In the first one we take as a starting point the traditional proposals of QoS at the traffic class level. We propose a more efficient framework for QoS provision using just two virtual channels at the switches. This is possible because we take advantage of scheduling performed at end-nodes.

In the next part, we propose how to adapt deadline-based QoS algorithms to high-performance interconnects. Using an approach similar to that of the previous part, we obtain remarkable results with simple switch architectures.

Finally, we investigate in the interaction of QoS mechanisms and congestion management techniques. We propose an integrated solution that benefits from the synergy between both aspects of high-performance networking.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

IN this chapter we start this thesis by discussing its motivation and objectives. Firstly, we discuss the framework in which we are working. We identify the problem we aim to solve and offer the motivation that drives this work. Finally, we settle the objectives we want to accomplish and introduce the organization of the following chapters.

1.1. Environment

In the information era, networks are responsible of communicating the ubiquitous computing devices. There are a plethora of implementations of this general idea, each implementation aiming to satisfy the requirements of different applications.

An interconnection network is the subsystem that satisfies the communication necessity of a larger system. For instance, there are interconnection networks in multicomputers, in multimedia broadcasting, in telephony, etc.

Point-to-point interconnection networks have replaced buses in an ever growing range of applications that includes on-chip interconnect, switches and routers, and I/O systems. Point-to-point networks provide the scalability and performance demanded by many emerging and consolidated applications. In this way, the field of *high-performance* networks or interconnects is still a very active research area.

The technology of high-performance networks has been a subject of constant research during the last two decades. In this way, new proposals are constantly

appearing and many *ad hoc* solutions have been implemented. It has not been until recently that standards have emerged to uniformize the development of interconnection networks.

Interconnection networks are currently being used for many different applications, ranging from internal buses in very large-scale integration (VLSI) circuits to wide area computer networks. Among others, these applications include backplane buses and system area networks; telephone switches; internal networks for routers; processor/memory interconnects for vector supercomputers; interconnection networks for multicomputers and distributed shared-memory multiprocessors; clusters of workstations and personal computers; local area networks; metropolitan area networks; wide area computer networks; and networks for industrial applications. Additionally, the number of applications requiring interconnection networks is continuously growing.

Parallel to the evolution of interconnection networks, there is also an evolution of applications. In a feedback cycle, new network capabilities allow new applications and the use of new applications drives the research for better networks.

However, the emerging applications are different from the old ones. Traditionally, the applications have only required that the network did its best to communicate applications. This is called *best-effort*. On the other hand, the new applications require *guarantees* on the performance that the network will offer. For instance, if we are to set up a telephony system over the network, we must guarantee that the users will have satisfactory communication. This is only possible if the interconnection network can also guarantee that the application will achieve a certain performance.

These performance guarantees to applications and network users are known as *quality of service* (QoS). The most typical example of QoS-requiring applications are multimedia applications. These are characterized by high bandwidth requirements and low response time. However, we will see in the next chapter that there are many application-specific QoS requirements.

In addition to multimedia applications, QoS provision is useful in interconnection networks for other applications. For instance, it is usual that in a multicomputer there are several types of applications and, therefore, several types of traffic. We can identify:

- Parallel applications traffic. This is the "mission" traffic, the traffic that comes from the applications that provide the functionality of the system. Both for shared-memory and message-passing architectures, the communication between processes must be as fast as possible, to reduce network overhead.
- I/O traffic. When the applications need to access storage media, they generate a considerable amount of bandwidth because they usually need

large amounts of data. In this case, delay is not as important as the *throughput*, or the amount of information transferred each time unit.

- Management traffic. In addition to "mission" applications, the system needs to execute management applications to work properly. For instance, we need backup copies, accounting, operating system updates, etc. This traffic is not specially QoS-demanding, but it is desirable that it does not disturb the normal operation of the system.

An expensive although common solution is to provide three separate networks, one for each kind of traffic. In this way, QoS-provision is achieved by implementing each network with worst-case requirements, which is a complete waste of resources.

A QoS-provision-based solution would be to just implement a network which is shared by the three kinds of traffic. In this way, resources are reused and the interconnection becomes cost-effective. Of course, in that case it is essential to schedule the available resources in a correct way such that the different traffic classes achieve their requirements.

1.2. High-Performance Networks with QoS Support

In the last years, there have been many proposals of high-performance networks with QoS support. In the next chapter we will review some of them in depth, but we can anticipate that most of these proposals rely on *virtual channels* (VCs).

The idea behind VCs is to build several virtual networks over a single physical network and achieve efficiency by statistically multiplexing the traffic of these networks. This has been a very successful idea, and there have been many proposals on how to assign applications to VCs and how to multiplex traffic from different VCs. Moreover, this idea has made its way into the few specifications of high-performance interconnect standards.

Most proposals incorporate 16 or even more VCs, as we will see in the next chapter. The problem with VCs is that they are expensive to implement if we put too many of them. Therefore, there are few implementations with more than a few VCs, and hence the main motivation of our work, as we see in the following.

1.3. Motivation

The main motivation of this thesis is the lack of efficient QoS-provision proposals for high-speed interconnects. Most of the proposals in the literature fall into one of the following two categories:

- Effective, but too complex to be implemented or to scale properly.
- Scalable and affordable, but with a reduced QoS provision, often for just two or three broad categories.

The motivation of our work is to offer an architecture both effective and affordable for QoS provision in high-speed interconnects. As we will see later, our proposal is based on the elimination of redundancies in QoS support. Therefore, we are not aiming at a better performance, but at obtaining similar performance while dramatically reducing implementation cost.

1.4. Objectives

Given the motivation of our work, we can outline a series of smaller objectives, which gradually converge towards our main motivation. These objectives are:

1. Study of previous work. This involves three areas of research: high-performance networks in general, QoS provision in general, and QoS provision in high-performance networks. We must also pay attention to specific proposals and industry standards.
2. Develop a simulation tool to model high-performance networks. This tool has to be flexible enough to test different architectures and proposals of QoS support. Moreover, it also has to be accurate enough to provide meaningful results. Besides, a great variety of performance metrics are desirable, to measure the goodness of different proposals.
3. Identify the sources of redundancy in QoS-provision proposals. The idea is to be able to achieve the same or similar results with simpler network elements.
4. Propose an efficient network architecture to provide QoS in high-speed interconnects. This proposal has to consider a limited set of traffic classes and has to offer bandwidth and delay guarantees to them.
5. Evaluate this proposal from the implementation point of view, considering how much silicon area, power consumption, and delay are introduced when using this proposal.
6. Evaluate this proposal from the performance point of view. In this case, we study by simulation the traditional QoS indices such as latency and throughput.
7. Propose an efficient QoS-provision at the individual flow level, instead of just considering a few traffic classes. This has also to be evaluated to study its performance.

8. Propose an integrated framework of QoS-provision and congestion management. Although congestion management is not the topic of this thesis, it is desirable an integrated solution. In this way, both mechanisms can be implemented at the same time with an affordable cost.

These points will be covered along this thesis. Moreover, in the last chapter we will revisit them to see how they were accomplished.

1.5. Organization of the Thesis

This thesis is organized in six chapters, which are briefly introduced here:

- Chapter 1. This chapter introduces the framework and the motivation of the thesis.
- Chapter 2. The related work is introduced here. We offer a general view of high-speed interconnects. We also review QoS provision in networks. Finally, we study specific proposals of high-performance interconnects with QoS provision.
- Chapter 3. In this chapter we introduce our proposal of efficient QoS provision at the traffic class level. This chapter includes a more detailed motivation, the new proposal, and the evaluation.
- Chapter 4. The proposal of efficient per-flow QoS provision is presented here. We also follow the structure of motivation, proposal, and evaluation.
- Chapter 5. Our last proposal is presented here. In this case, we integrate traffic-class QoS provision with efficient congestion management. The evaluation of this integrated architecture is also included in this chapter.
- Chapter 6. This last chapter summarizes and concludes the thesis. We will discuss which are the main contributions of our work, which publications have followed, and which are the directions of future work.

In addition to these, there is a detailed bibliography at the end of the thesis.

CHAPTER 2

High-Performance Interconnects

INTERCONNECTION networks are a key component in a variety of systems. These include systems-on-chip, system area networks, high-speed local area networks for clusters, and interconnects for the most powerful parallel machines. In all these cases, the requirements of high bandwidth and very low latency differentiate this kind of networks from classical local area networks.

The objective of this chapter is to introduce the basic concepts regarding interconnection networks that will be necessary to fully understand our proposals in the following chapters. Since this is a brief review of a broad topic, the reader is encouraged to use bibliographic references for futher information. On the other hand, two general books that are particularly suitable for introducing interconnection networks are [Duato et al. 02] and [Dally and Towles 03].

Firstly, we will review the special characteristics of interconnection networks, specially focusing on switch design and architecture. The next part of the chapter focuses on QoS, both as a general characteristic of networks and specifically for interconnection networks. Finally, we will review several high-performance switches and interconnects with QoS support.

2.1. High-Performance Networks

Networks are responsible for the communication between the components of many systems. The problem has been extensively studied and there are a

plethora of proposals. The networks can be classified according to their size, from tiny on-chip networks, system area, local area, metropolitan area, and wide area networks. Moreover, the physical medium used to provide the interconnect can also vary, from copper wires, optical fiber, and wireless radio-based networks.

High-performance networks are a subset of networks that are characterized by the application requirements, rather than physical characteristics [Duato et al. 02]. In this case, the applications demanding high-performance networks require a very high *bandwidth* and a very short response time or *delay*. When these requirements are strong enough, the traditional techniques used for network design are not enough to offer satisfactory results. Since commodity components cannot be used, a new set of techniques and new architecture designs are needed to build *high-performance networks* or *high-performance interconnects*.

There are many systems where a high-performance interconnect is necessary. Here, we give a list of examples, but the number of applications requiring interconnection networks is continuously growing.

- Internal communication in very large-scale integration (VLSI) circuits.
- System and storage area networks.
- Internal networks for telephone switches and internet protocol (IP) routers.
- Processor-to-processor and processor-to-memory interconnects for supercomputers.
- Interconnection networks for multicomputers and distributed shared-memory multiprocessors.
- Clusters of workstations and personal computers.

As we see, interconnection networks are a key component in a variety of systems. Specially in supercomputers, the network is usually the bottleneck, rather than the processors. For this reason, the theoretical maximum performance from parallel applications is limited by the communication subsystem [Duato et al. 02]. This illustrates the importance of efficient high-performance interconnects.

2.1.1. Classification

Interconnection networks can be classified according to network topology [Duato et al. 02]. In this way, we would have:

- Shared-medium networks. The transmission medium is shared by all communicating devices.

- Direct networks. Point-to-point links directly connect each communicating device to a (usually small) subset of other communicating devices in the network.
- Indirect networks. The devices are connected by means of one or more switches, instead of being directly connected.
- Hybrid approaches. In this case, an intermeditate solution is implemented.

These network classes and the corresponding subclasses will be described in the following.

2.1.1.1. Shared medium networks

In shared medium networks, there is a transmission medium shared by all communicating devices. In such shared-medium networks, only one device is allowed to use the network at a time. On the other hand, all the devices can receive the information transmitted simultaneously.

The most common shared medium is a bus. Wireless communications also belong to this category, but usually do not provide a performance good enough for high-performance applications. Optical buses, on the other hand, can greatly increase bandwidth, but suffer of the same problems than classical buses.

The arbitration strategy is an important issue in shared medium networks. When several devices want to communicate, the bus arbiter has to choose one of them to become the bus master. There are many alternatives to implement this arbiter, but nearly all of them must make use of specialized hardware.

The main advantage of shared medium networks, besides their simplicity, is their ability to support atomic broadcast. This property is important to efficiently support many applications requiring one-to-all or one-to-many communication services, such as barrier synchronization and snoopy cache coherence protocols.

However, due to limited network bandwidth, a single shared medium can only support a limited number of devices before the medium becomes a bottleneck. Therefore, shared medium networks *scale* badly. This means that the interconnection cannot be efficiently expanded to cope with increasing numbers of communicating devices.

2.1.1.2. Direct networks

As we have seen, the main problem with shared-medium based networks is the scalability. The direct network or point-to-point network is a popular network architecture that scales well to a large number of devices. A direct network consists of a set of nodes, each one being directly connected to a (usually small)

subset of other nodes in the network. Each node contains one of the devices that are communicating and, in addition to it, contains a *switch*. Switches handle message communication among nodes, since each switch is connected to some other switches, belonging to neighbor devices. Usually, two neighboring nodes are connected by a pair of unidirectional channels in opposite directions. A bidirectional channel may also be used to connect two neighboring nodes.

Each switch supports some number of input and output *channels* or *links*. Internal channels or ports connect the local device. External channels are used for communication between switches. Usually, each node has a fixed number of input and output channels, and every input channel is paired with a corresponding output channel. Through the connections among these channels, there are many ways to interconnect these nodes. Obviously, every node in the network should be able to reach every other node.

Direct networks are characterized by their *topology*, which is the way in which the switches are connected by channels. Topologies are modelled as a graph, where the vertexes of the graph represent the nodes and the edges of the graph represent the communication channels. This is a very simple model that allows to study a lot of network properties in an abstract way.

Depending on graph properties, a topology is said to be superior than other. For instance, an all-to-all connection of all nodes is an excellent topology from the performance point of view, since the maximum distance between any two nodes is 1. However, this topology has the disadvantage of requiring a switch with N links (including the internal one) at each node for a network with N nodes. Therefore, the cost is prohibitive for networks of moderate to large size.

Popular network topologies include:

1. Meshes. In a n-dimensional mesh, most nodes have $2n$ external channels. However, nodes at the borders of the structure have less channels and, therefore, this topology is not regular.
2. Torus. A n-torus is almost like a n-dimensional mesh, but every node has exactly $2n$ external channels. In this case, there are no borders in the topology, since each row is connected to form a ring.
3. K-ary n-cubes. These are n-dimensional cubes with k nodes along each edge. Every node has n neighbors if $k = 2$ and $2n$ neighbors if $k > 2$.
4. Trees. This topology has a root node connected to a certain number of descendant nodes. Each of these nodes is in turn connected to a disjoint set (possibly empty) of descendants. A node with no descendants is a leaf node. A characteristic property of trees is that every node but the root has a single parent node. Therefore, trees contain no cycles.

There are many other topologies. The designer of the system must carefully select a topology which properties are the best suitable for the application that will communicate, but also with an efficient implementation.

Direct networks are very popular for high-speed interconnects, specially in multicomputers. There are many real-life examples of this network design, interested readers can consult [Duato et al. 02].

2.1.1.3. Indirect networks

Indirect networks are another major class of interconnection networks. Instead of providing a direct connection among some nodes, the communication between any two nodes has to be carried through some external switches. That means that nodes no longer have switches, but *network adapters* or *network interfaces*.

The interconnection of the switches defines various network topologies, just like in direct networks. However, the main advantage of indirect networks is that several nodes can share the same switch, thus reducing component count.

In addition to regular topologies, like those for direct networks, in indirect networks there is support for irregular topologies. This is a typical case in clusters, which can be built just by adding new switches and computers to the existing system.

Multistage interconnection networks (MINs) are also a popular topology for indirect networks. In this case, the devices are connected through a number of switch stages. The number of stages and the connection patterns between stages determine the routing capability of the networks.

There are many variations of MIN topology, depending on connection pattern and whether they are blocking or not. In non-blocking MINs, any two free devices can be connected without affecting any existing connections. This is very important in telephony and circuit switching networks, as we will see later.

Classical MINs are unidirectional and there are input devices and output devices. In this case, links are also unidirectional and switches differentiate input and output ports. However, since in most interconnection technologies links are bidirectional (or pairs of two unidirectional links are bundled together), bidirectional MINs are usually used. These are also known as folded MINs.

In bidirectional MINs, connections have a forward path, a turn-around point, and a backward path. The advantage of this topology is that there are no cycles and routing is easy.

2.1.2. Switching techniques

In the previous section we saw a classification of high-performance networks. This thesis will focus in direct and indirect networks, since they are the most efficient solutions.

A *switching technique* is the technique used to transfer information through the network. At the application level, *user messages* or just messages are generated. These are pushed to the network level through *network interfaces*. In these devices, messages are converted into *packets*. A message can generate one or more packets. In the latter case, packets must be reassembled at the receiver's network interface, to forward the original message to the application.

Taking into account the previous information, the switching technique deals on how to transfer packets from one end-node to another, passing through one or more switches. In the case of direct networks, each device is a switch by itself, while in indirect networks switches are separated devices.

The term *router* has a different meaning depending on the context. Since it may be confusing, we will rather use switch. However, it is usual in bibliography to use "router" for "switch" and "switch" for the "switch fabric" (usually a crossbar).

In this section we briefly review switching techniques. A more comprehensive description of these techniques can be found in [Duato et al. 02].

For the purposes of comparison, for each switching technique we will consider the computation of the base latency of an L-bit message in the absence of any traffic. The *phit* (physical unit) size and *flit* (flow control unit) size are assumed to be equivalent and equal to the physical data channel width of W bits. The routing header is assumed to be 1 flit; thus the message size is L + W bits. A switch can make a routing decision in t_r seconds. The physical channel between two switches operates at B Hz; that is, the physical channel bandwidth is B×W bits per second. We assume that channel wires are short enough to complete a transmission in one clock cycle. Therefore, the propagation delay across this channel is denoted by $t_w = \frac{1}{B}$. Once a path has been set up through the switch, the switching delay is denoted by t_s. The switch internal data paths are assumed to be matched to the channel width of W bits. Thus, in t_s seconds a W-bit flit can be transferred from the input of the switch to the output. The source and destination are assumed to be D links apart.

We will see in the following the four switching techniques more often used in high-performance networking: circuit switching, packet switching, virtual cut-through, and wormhole. Other less usual techniques can be found at [Duato et al. 02].

2.1.2.1. Circuit switching

In circuit switching, a physical path from the source to the destination is reserved prior to the transmission of the data. In this case, messages are not *packetized*, i.e. we transfer messages directly. The path is established by injecting a special message, the *routing probe*, into the network. This message contains the destination address and some additional control information. It progresses

toward the destination reserving physical links as it is transmitted through intermediate switches. When the control message reaches the destination, a complete path has been set up and an acknowledgment is transmitted back to the source. In this way, a circuit is established.

The message contents may now be transmitted at the full bandwidth of the hardware path. The circuit may be released by the destination or by the last few bits of the message. The circuit may also be kept for a longer period, as in telephony networks.

Circuit switching is generally advantageous when messages are infrequent and long; that is, the message transmission time is long compared to the path setup time. Once the circuit has been established, there is no significant delay, other than propagation delay, introduced on messages.

The base latency of a circuit-switched message is determined by the time to set up a path and the subsequent time the path is busy transmitting data. While the routing probe is buffered at each switch, data bits are not. There are no intervening data buffers in the circuit, which operates effectively as a single wire from source to destination.

The base latency of circuit switching can be expressed as:

$$\begin{aligned} t_{circuit} &= t_{setup} + t_{data} \\ t_{setup} &= D[t_r + 2(t_s + t_w)] \\ t_{data} &= \frac{1}{B}\left\lceil \frac{L}{W} \right\rceil \end{aligned}$$

The main disadvantage of circuit switching is that the physical path is reserved for the duration of the message and may block other messages. For example, consider the case where the probe is blocked waiting for a physical link to become free. All of the links reserved by the probe up to that point remain reserved, cannot be used by other circuits, and may be blocking other circuits, preventing them from being set up.

2.1.2.2. Packet switching

In circuit switching, the complete message is transmitted after the circuit has been set up. Alternatively, the message can be partitioned into packets. When packets have a fixed-length, they are usually called *cells*. The first few bytes of a packet contain routing and control information and are referred to as the packet header. Each packet is individually routed from source to destination. This technique is referred to as packet switching. A packet is completely buffered at each intermediate node before it is forwarded to the next node. This is the reason why this switching technique is also referred to as store-and-forward switching.

The header information is extracted by the intermediate switches and used to determine the output link over which the packet is to be forwarded. The latency

experienced by a packet is proportional to the distance between the source and destination nodes.

Packet switching is advantageous when messages are short and frequent. Unlike circuit switching, where a segment of a reserved path may be idle for a significant period of time, a communication link is fully utilized when there are data to be transmitted. Many packets belonging to a message can be in the network simultaneously even if the first packet has not yet arrived at the destination.

However, splitting a message into packets produces some overhead. In addition to the time required at source and destination nodes, every packet must be routed at each intermediate node.

Another disadvantage of packet switching is that the storage requirements at the switches can become extensive if packets can become large and many packets must be buffered at a node. This can happen when networks are large and switches have a significant radix (number of ports).

The base latency of a packet-switched message can be computed as follows:

$$t_{packet} = D\left(t_r + (t_s + t_w) + \left\lceil \frac{L+W}{W} \right\rceil\right)$$

The actual result may vary depending on the actual architecture of switches, but the important point to note is that the latency is directly proportional to the distance between the source and destination nodes.

2.1.2.3. Virtual cut-through switching

Packet switching is based on the assumption that a packet must be received in its entirety before any routing decision can be made and the packet forwarded to the destination. This is not generally true and, rather than waiting for the entire packet to be received, the packet header can be examined as soon as it is received. The switch can start forwarding the header and following data bytes as soon as routing decisions have been made and the output buffer is free. In fact, the packet does not even have to be buffered at the output and can cut through to the input of the next switch before the complete packet has been received at the current switch. This switching technique is referred to as virtual cut-through switching. In the absence of blocking, the latency experienced by the header at each node is the routing latency and propagation delay through the switch and along the physical channels. The packet is effectively pipelined through successive switches. If the header is blocked on a busy output channel, the complete packet is buffered at the node. Thus, at high network loads, virtual cut-through switching behaves like packet switching.

The base latency of a packet that successfully cuts through each intermediate switch can be computed as follows:

$$t_{vct} = D(t_r + t_s + t_w) + max(t_s, t_w) \left\lceil \frac{L}{W} \right\rceil$$

Cut-through routing is assumed to occur at the flit level with the routing information contained in the first flit. This model assumes that there is no time penalty for cutting through a switch if the output buffer and output channel are free. Depending on the speed of operation of the switches, this may not be realistic. Note that only the header experiences routing delay, as well as the switching delay and wire delay at each switch. This is because the transmission is pipelined and the switch is buffered at the input and output. Once the header flit reaches the destination, the cycle time of this packet pipeline is determined by the maximum of the switch delay and wire delay between switches.

Note that the unit of flow control is a packet. Therefore, even though the packet may cut through the switch, sufficient buffer space must be allocated for a complete packet in case the header is blocked.

2.1.2.4. Wormhole switching

The need to buffer complete packets within a switch can make it difficult to construct small, compact, and fast switches. In wormhole switching, message packets are also pipelined through the network. However, the buffer requirements within the switches are substantially reduced over the requirements for virtual cut-through switching. A message packet is broken up into flits. The flit is the unit of flow control, and input and output buffers at a switch are typically large enough to store a few flits.

The packet is pipelined through the network at the flit level and is typically too large to be completely buffered within a switch. Thus, at any instant in time a blocked packet occupies buffers in several switches.

The primary difference between wormhole switching and virtual cut-through switching is that, in the former, the unit of flow control is a single flit and, as a consequence, small buffers can be used. Just a few flits need to be buffered at a switch.

In the absence of blocking, the message packet is pipelined through the network. However, the blocking characteristics are very different from that of virtual cut-through. If the required output channel is busy, the packet is blocked "in place". The small buffer sizes at each node (smaller than packet size) cause the packet to occupy buffers in multiple switches, similarly blocking other packets. In effect, dependencies between buffers span multiple switches. This property complicates the issue of deadlock freedom. However, the small buffer requirements and packet pipelining enable the construction of switches that are small, compact, and fast.

Note that routing information is associated only with the header phits (flits) and not with the data flits. As a result, each incoming data flit of a packet is simply forwarded along the same output channel as the preceding data flit. This means that the transmission of distinct packets cannot be interleaved or multiplexed over a physical channel. The packet must cross the channel in its entirety before the channel can be used by another packet. This is why deadlock is easier when using wormhole switching.

The base latency of a wormhole-switched packet can be computed as follows:

$$t_{wormhole} = D(t_r + t_s + t_w) + max(t_s, t_w) \left\lceil \frac{L}{W} \right\rceil$$

Note that in the absence of contention, virtual cut-through and wormhole switching have the same latency.

2.1.3. Flow control in buffered switching fabrics

When using packet switching, it may happen that instantaneous rate demanded of a link is higher than its capacity. Buffers are provided to attenuate this problem, but if demand persists, buffers may be overflowed. A flow control mechanism is needed to manage this situation, even if the solution provided consists only in dropping excess of packets.

2.1.3.1. Lossy versus lossless flow control

Flow control mechanisms can be classified by whether or not the receiving node is allowed to drop packets [Dally and Towles 03]. The characteristics of lossy flow control are:

- Packets are dropped when there is not enough space to store them in a buffer.
- The information that is lost must be retransmitted by sources. A source knows that a packet was dropped either because it receives an NACK or because a configured amount of time passes without receiving an ACK.
- Lossy flow control is very simple to implement. Moreover, deadlock situations cannot happen and congestion never propagates backwards.
- However, since packets may be dropped, some bandwidth is wasted. This leads to the concept of *goodput*, which is the fraction from network throughput that is actually useful.
- Another problem of lossy networks is that, due to packet drops and retransmissions, delay of packets may get intolerably high for some applications.

The other type of techniques are lossless. In this case, the receiver sends some feedback to the sending node. Sometimes, these are called *backpreassure* techniques.

- Since packets are never dropped, no retransmission are needed and bandwidth is not wasted. However, there is some *control overhead*, which is the bandwidth consumed by flow control messages. However, this is usually negligible.
- Lossless flow control introduces new challenges, like head-of-line blocking and congestion. We will talk about these later.
- The hardware is more complex since the switches must implement the flow control logic.
- Packets may be delayed or blocked when they may overflow a buffer in the next hop. The designers must be careful to avoid *deadlock* situations, where there is a cycle of dependencies between packets and no-one can advance.

In high-performance networks, lossless flow control is generally preferred. The reason is that retransmissions and the delays they involve are not tolerable by the applications which use the network.

2.1.3.2. Stop and Go

In stop and go, the receiver buffer, of size B, has two marks, k_{STOP} and k_{GO}, such as $0 < k_{GO} < k_{STOP} < B$, as can be seen in Figure 2.1. The state of the buffer is characterized by the amount of information contained, f. Initially, $f = 0$ and it may grow as packets are stored in the buffer. Likewise, f decreases as packets are forwarded to the next stage and, thus, are removed from the buffer.

The objective of the flow control mechanism is to avoid that $f > B$ happens. In order to achieve this, two control symbols are used. The buffer generates a STOP control symbol when f increases to k_{STOP}, and generates a GO control symbol when f decreases to k_{GO}. The k_{STOP} and k_{GO} parts of the buffer provide the slack necessary for the delay between sender and receiver. The margin between both marks provides hysteresis, i.e. a working area where no signals are generated.

The parameter k_{STOP} is the slack available for stopping the flow on the channel when the data sink becomes blocked, or when the data sink is operating at a lower bandwidth than the channel. In the worst case in which the data sink becomes blocked, k_{STOP} must be large enough to stop the sender before the buffer overflows. The amount of information in transit on the round-trip

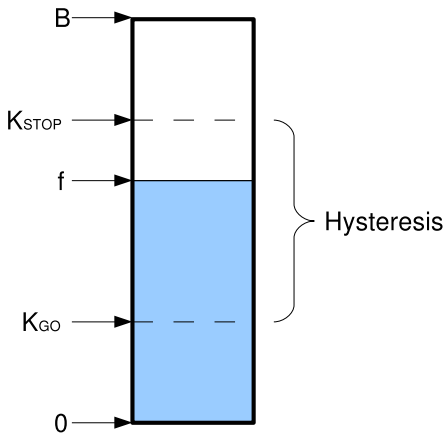


Figure 2.1: Elements of stop and go flow control.

path depends on link bandwidth and length. k_{STOP} must be at least this value plus additional buffer positions required due to the latency in generating and interpreting the STOP control symbol.

Similarly, k_{GO} is the slack available for maintaining the flow into the data sink after f decreases to k_{GO}, and the GO control symbol is sent. Unlike the k_{STOP} part of the buffer, which prevents data loss, the k_{GO} part of the buffer is required only for performance reasons. The value of k_{GO} must be larger than a round-trip time worth of data, plus the latency in generating and interpreting the GO control symbol. In this way, the sink will constantly have packets to be transmitted if the sender keeps generating them.

The hysteresis parameter, $k_{STOP} - k_{GO}$, is important only for reducing the number of STOP and GO symbols that must be sent on the opposite-going channel in cases in which the data sink takes packet data in short bursts. Under the most adversarial conditions, two control symbols are generated every $k_{STOP} - k_{GO}$ bytes of data transmitted.

The main advantage of stop and go flow control is its simplicity to be implemented. The receiving buffer must take into account just two thresholds and send the STOP and GO control tokens, usually by means of special control messages. On the other hand, the sending device also needs simple logic to handle the flow control protocol.

The biggest drawback of stop and go is that the optimum value of k_{STOP} and k_{GO} is difficult to calculate. It depends on link bandwidth, link length, and delay to produce and decode the control messages. For this reason, a compromise value is often used, with enough slack for the worst case.

Another disadvantage of stop and go is that the minimum buffer size to produce optimum performance is 4 RTTs[1] worth of data: the minimum value for k_{STOP} plus the minimum for k_{GO}. Moreover, since conservative values are usually taken for both parameters, the buffers must be even larger to work properly. In the next section we will see a different technique that can work efficiently with smaller buffers.

2.1.3.3. Credit-Based Flow Control

In credit-based based flow control, the receiver buffer is divided in a set of slots. In the most simple implementations, each slot is equivalent to a packet. However, when variable packet size is used, the slot represents a fixed amount of information, for instance 64 bytes. This is known as the flow control unit or *flit*.

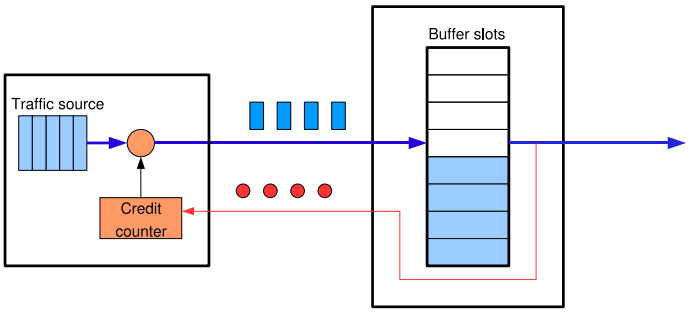


Figure 2.2: Elements of credit-based flow control.

When the system is initialized, the receiver informs the sender about the number of flits in its buffer. The sender stores this value in a register, the *credit counter*. The operation of the system is as follows: every time the sender transmits a packet through the link, it decrements the *credit counter* with the number of flits of the packet. If a packet is larger than the amount of flits available in the *credit counter*, it is not transmitted. In this way, the receiver's buffer cannot be overflowed.

When the receiver is able to transmit messages to the next stage and, therefore, slots become available in the buffer, corresponding credits are sent upstream. In this way, the *credit counter* of the sender is incremented and more packets are allowed into the link. The elements of the systems are illustrated in Figure 2.2.

[1] RTT: round-trip time. In this case, the time taken by the shortest message to go from the sender to the receiver, plus the time taken by the flow control feedback to return from the receiver to the sender.

The optimum buffer size at the receiver is $RTT \times R$, where R is the peak rate of the link. Assuming than sender has always packets available and that the receiver generates credits at R rate, the minimum necessary credits are those for in flight data over the link, which are $RTT \times R$. However, this assumes a very fine grain of flits. In practice, minimum buffer size is $RTT \times R + 1\ flit$. Moreover, if we are using packet switching or virtual cut-through switching and credits are only generated when a packet is fully transmitted, then minimum buffer size is:

$$B_{MIN} = MTU \times \left\lceil \frac{RTT \times R + MTU}{MTU} \right\rceil$$

where MTU is the maximum transfer unit. Note that this usually translates into just 2 $MTUs$ which is a noticeably improvement over stop and go flow control buffer requirements.

Taking into account the previous information, the main advantage of credit-based flow control is that buffer management is much more efficient than in stop and go flow control. For this reason, this is a widely adopted technique in recent high-performance networks [Inf 00, Adv 05].

The main drawback of credit-based flow control is the complexity of its implementation. Although this technique is not as complex as being unaffordable in high-performance implementations, when compared with stop and go or with packet dropping, credit-based flow control is more complex.

2.1.3.4. Congestion and traffic shaping

Network performance heavily depends on input load. Not only the amount of information to transfer, but also the distribution of packet destinations and the distribution of packet arrivals in time. In this sense, when many packets have to be transferred from the same origin towards the same destination in a short period of time, we talk about *bursty* traffic.

Bursty traffic may dramatically degrade network performance. The reason is that the destination is not able to cope with all the incoming traffic and buffers get full of packets that cannot make further progress. This prevents other packets that are addressed towards destinations with available bandwidth to advance, since they share buffers with *congested* packets.

In order to cope with this problem, there has been several proposals:

- Proactive techniques. These techniques completely avoid congestion but only allowing traffic into the network when it is sure that it will not cause any congestion. To implement this, there is some kind of *connection admission control* that allows or not new connections.

- Reactive techniques. In this case, all traffic is allowed into the network. However, when congestion is detected, some special measures are taken towards reducing its effects. These techniques usually do not scale very well.

- Traffic shaping. This is similar to proactive techniques, but in a relaxed way. The idea is to prevent certain packet patters to be injected into the network. The classical example is token buckets, which limits both the maximum burst length and injection rate of a flow of packets.

- Other techniques concentrate in preventing the situation where congested packets can utilize the totality of buffers. Congestion is not eliminated, but its effects are greatly diminished.

In summary, the use of flow control introduces the problem of congestion, which does not happen in lossy networks. Most interconnection networks propose to use one or more of the previous congestion management techniques.

2.1.4. Technology of high-performance networks

In this section we will discuss how are the different elements of a high-performance network implemented with current technology. Nowadays, high-performance networks have three key elements: network interfaces, switches, and links. We will start briefly reviewing the main components of a high-performance switch. In the case of network interfaces at end-nodes, the components are similar to that of the switches.

In Figure 2.3 we show a generic high-speed switch, aimed for a single-chip implementation. Its main components are:

- Link controller (LC). Flow control across the physical channel between adjacent network elements is implemented by this unit. The link controllers on either side of a channel coordinate the transfer of flow control units. Sufficient buffering must be provided on the receiving side to account for delays in propagation of data and flow control signals. When a flow control event signaling a full buffer is transmitted to the sending controller, there must still be sufficient buffering at the receiver to store all of the phits in transit, as well as all of the phits that will be injected during the time it takes for the flow control signal to propagate back to the sender. If virtual channels are present, the controller is also responsible for decoding the destination channel of the received phit.

- Buffers. These are FIFO buffers for storing messages in transit. In the figure, a buffer is associated with both the input physical channels and output physical channels. The buffer size must be a multiple of flow control

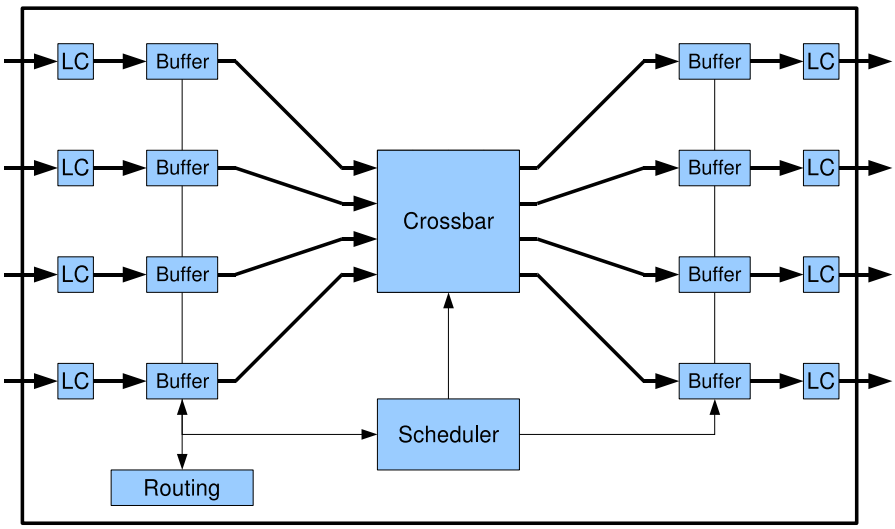


Figure 2.3: Generic switch architecture.

unit size, otherwise some space would be wasted. In alternative designs, buffers may be associated only with inputs (input buffering) or outputs (output buffering).

- Routing unit. This logic implements the routing function. For adaptive routing protocols, the message headers are processed to compute the set of candidate output channels and generate requests for these channels. For oblivious routing protocols, routing is a very simple operation.

- Crossbar scheduler. This unit configures the crossbar every scheduling cycle, selecting the output link for incoming messages. Output channel status is combined with input channel requests. Conflicts for the same output must be arbitrated (in logarithmic time). If the requested buffer(s) is (are) busy, the incoming message remains in the input buffer until a requested output becomes free. Fast scheduling algorithms are crucial to maintain a low flow control latency through the switch.

- Crossbar. This component is responsible for connecting switch input buffers to switch output buffers in high-speed switches. In the past, other alternatives were used, like buses, but nowadays, crossbars are very popular.

These basic blocks are found in most high-performance switch designs. However, the organization of the switch may vary, as we will see in the next section. Now, we will review the technological facts related with these components.

2.1.4.1. Links

We have seen that modern high-performance networks use point-to-point links to transfer information. These links are either optical fibers, or metallic (usually copper) cables. We will review the two types of links, parallel, which was common in the past and now is used in short distances; and serial, which is the most usual kind of link nowadays.

Parallel transmission links use multiple conductors to carry an entire word –rather than a single bit– in each unit of time. In addition to the n data lines, a clock signal must be sent in parallel. Moreover, in order to delineate packet boundaries, framing information is also needed. This can be encoded in the data lines if all 2^n combinations of the n bits are not used for data. Otherwise, when all bit combinations are acceptable values inside the network traffic, or when convenience or robustness dictates it, out-of-band signaling is used and one or more separate framing wires are included with the link.

In modern high-performance switching systems, parallel links are used extensively inside ASIC chips, and in few cases between nearby chips, usually on the same printed circuit board. The primary limitation of high-speed parallel links is timing skew among the n cables making up the link. As clock frequency increases synchronization among the bits of a word is lost, and the link becomes unusable.

Serial transmission links carry all of their information through a single line. The main reason is to avoid the inevitable timing skew when clock frequency is very high. Since there is no clock signal alongside with data, the receiver is forced to extract the clock from the single serial signal received over the single line.

A serial line receiver has three functions: clock recovery, serial-to-parallel conversion, and framing recovery. Let us review them briefly:

- Clock Recovery. This is achieved by using a local oscillator whose frequency is approximately equal to the known signaling rate. The objective is to keep adjusting the oscillator so as to match the incoming singal clock.

  For clock recovery to be successful, the incoming serial signal must change between 0 and 1 at least once every given number of clock periods (bit intervals). In other words, the maximum length of trains of consecutive 1's or 0's must be bounded.

  In order to cope with that, proper line coding schemes are used, which introduce a certain proportion of redundancy or overhead information to ensure that signal transitions will occur often enough. A popular line code, used in Fibre Channel and Gigabit Ethernet, is the 8B/10B encoding, which encodes each 8-bit byte using a 10-bit codeword [ANSI 93, Seifert 98].

- Framing recovery. This problem consists in how to discover the boundaries of bytes and packets in a long series of bits, in the lack of additional information. If these boundaries are found once, then, in theory, the system would be able to maintain such synchronization forever: byte boundaries are found by counting bits (eight-by-eight); cell or frame boundaries are found by counting bytes (modulo the fixed, known cell or frame size); and for variable-size packets, their boundaries are found by looking at the packet size field in the header of each packet to determine where this packet ends, hence where the next packet begins, thus where that next packet's header and size field are, etc.

  However, a practical system must operate robustly in the presence of noise and bit errors, hence in the presence of occasional losses of bit and/or frame synchronization. Thus, the line coding scheme must be properly designed in order to enable the receiver to recover frame synchronization after occasional loss of it.

  This can be achieved by using special synchronization control characters regularly. For instance, taking the 8B/10B encoding, there are 1024 possible characters. Out of them, 256 codewords are selected to represent the 256 different data byte values. However, after eliminating the illegal values, which contain too few 0-1 or 1-0 transitions, i.e. which contain sequences of 1's or 0's that are too long, there are still enough valid characters that can be used as control characters.

  A codeword is chosen, usually called SYNC, that is appropriate as a synchronization control character; SYNC characters are to be transmitted periodically in order for the receiver to use them when it needs to recover byte and frame synchronization. When synchronization is lost (as detected by too high of an error rate), the receiver examines all groups of 10 bits, starting at every arbitrary bit position, looking for a SYNC character; when found, the SYNC character indicates byte boundaries, and it may also indicate cell or frame or packet boundaries.

- Serial-to-parallel conversion. The received data must be converted to parallel form and decoded from the line code to plain, unencoded values. Serial to parallel conversion is usually done in order to reduce clock frequency, because serial link transmission rate is usually quite higher than ASIC clock frequencies. Moreover, the usual form of on-chip processing is parallel rather than serial datapaths.

We have briefly reviewed the technology that allows data to go from one device to another. Now, we will look into the components found inside switches and network interfaces.

2.1.4.2. Buffers

Buffers are an important component of network devices, specially when packet switching is implemented. The amount of information to be stored is too large to use normal registers and, therefore, RAM must be used.

The RAM used to implement the buffers can be inside or outside the chip containing the switch. The main advantage of external RAM is that it opens up the possibility of using very large buffers. However, the delay introduced by such external chips is an overhead that is regarded as intolerable for high-speed interconnects.

On chip SRAM is the most popular alternative since it can provide a very high bandwidth. This is achieved by using wide datapaths and accessing several SRAM modules in parallel.

The characteristics of SRAM modules in terms of area, power consumption, and clock frequency depend on the actual integration technology and on the manufacturer design of the SRAM blocks. For a typical single-chip switch, total memory devoted to buffers is usually in the order of hundreds of kilobytes.

Implementing a single FIFO queue in a RAM is a simple task. The idea is to build a circular array, where two pointers are used: one for reading at the queue head and another for writing at the queue tail.

If several queues are to be implemented over the same RAM, it can be statically partitioned. In this case, we would have parts of the RAM devoted exclusively to one queue. Each partition would implement a circular array. In this way, we would need two pointers for each queue.

On the other hand, it is usually desirable to have multiple queues that can grow dynamically and take advantage from all the RAM implemented. In order to build a structure like this, the RAM is divided in blocks of fixed size. These blocks store data and a pointer to another block. Therefore, several linked lists can be built. In addition to regular queues, there is a list that links all the empty blocks.

The operation of this system would require the management of all the dynamic lists. When a packet arrives at the system, one or more blocks from the empty list must be allocated to the corresponding queue, in order to store the packet. Likewise, when a packet leaves the RAM, the blocks that are freed are linked to the empty list.

The operation of these dynamic lists is more complex than just storing words in a circular array. Moreover, since the memory has to be splitted in fixed size blocks, some space may be wasted if incoming packets are not in block size increments. Besides, the pointers to link blocks also take additional space in the RAM. Note that larger blocks reduce the overhead of pointers, but also increase the space that can be wasted when packets are not using all the space they have assigned.

Despite their disadvantages, dynamic queues are necessary to build the data structures required by many switch designs. Moreover, they are much more advantageous for high-speed switches than external RAM.

2.1.4.3. Crossbars

The main task of a switch is to forward information from the input ports to the appropriate output port. The alternatives to achieve this task include shared medium (buses), central memory, time division switching over a fast link, and crossbars.

A crossbar is a device that has N inputs and M outputs, which allows up to $min(N, M)$ one-to-one interconnections without contention. Usually, N equals M in switches. The cost of the crossbar is $O(N \times M)$ and, therefore, it is usually only feasible for values like 8×8, 16×16, or 32×32.

When crossbars are used for switching, the problem consists in matching the requests from input ports with requested output ports in the most efficient way. This task corresponds to the crossbar *scheduler*.

Packets may be transferred across the crossbar in variable or fixed-length units. When using fixed-length cells, the crossbar is configured in fixed-length increments, or time-slots. At the end of each time slot, the scheduler examines the packets waiting to be transferred across the crossbar switch and selects the configuration that will connect the appropriate inputs and outputs. This scheduling decision can be made in a way that is fair among connections, never starves input or output, and maintains highly efficient use of the crossbar switch. Efficiency is important, because the crossbar is a central resource shared among all the input ports. Wasting its bandwidth would degrade the performance of the switch.

If variable-length packets are used, they may finish traversing the crossbar switch at any time. This is a problem because the scheduler must constantly track the busy and idle status of the outputs and, as soon as an output becomes idle, quickly choose the next input allowed to connect to it next. However, at this moment, most of the switch outputs are probably busy. If the scheduler immediately connects the input to an idle output, this could lead to an starvation situation, where some connections are never established because there is always traffic available to keep current crossbar configuration. On the other hand, the scheduler can make the input wait, but this would waste bandwidth. This is why switches that handle variable-length packets have lower throughput, and are able to use only about half the system's aggregate bandwidth [Iliadis 92].

Although the scheduler may operate on fixed-size units, the scheduler must also consider the switching technique implemented in the network. For instance, when using virtual cut-through or wormhole, once the scheduler selects the first portion of a packet to cross the crossbar, the rest of the parts of the packet must

follow immediately. Otherwise, the output link would not be able to forward the first flits if it is not sure that the other flits will be ready when needed.

In recent years, there has been a significant amount of research work on scheduling algorithms for crossbar switches that use input queuing. The desireable properties of a good scheduler are [McKeown 99]:

- High throughput. Ideally, the algorithm will sustain an offered load up to 100 percent on each input and output.
- Starvation free. The algorithm should not allow any input queue to be unserved indefinitely.
- Fairness. When there is a bottleneck at one of the output ports, flows competing for the same bandwidth are entitled to the same amount of resources.
- Fast. The scheduling algorithm must not become the performance bottleneck. It should select a crossbar configuration as quickly as possible.
- Simple to implement. To be fast in practice, the algorithm must be implemented inside the switch chip.

Since the design of efficient crossbar schedulers for input-queued switches is a complex task, there is a trend on providing internal speed-up to the crossbars. That means that the crossbar point-to-point connections are faster than the links connected to the switch. Typical values for this speed-up are 1.5 or 2.0, meaning that the crossbar is 50% to 100% faster than the links. In this way, the crossbar, despite scheduler inefficiencies, is not the bottleneck of the system.

When there is speed-up in the crossbar, buffers at the outputs of the switch are mandatory. The reason is that, since the crossbar is faster than the output links, some memory is needed to store the excess of information transferred each scheduling cycle. However, this also implies that some kind of internal flow control is needed to avoid overflowing the output buffers, leading to more complex architectures.

Another design decision to take when implementing crossbars is if they will be multiplexed or not. When there are several virtual channels implemented, we can choose whether to provide separate crossbar ports for each VC or not. This is another way of providing crossbar speed-up, but it may be difficult to implement if there are many virtual channels.

2.1.5. Switch organization

In order to do its switching function, the most efficient switches implement a crossbar. However, we also saw that the switches also have to implement some

buffer space when using packet switching or any of its variants. In this case, there are several options regarding where to put the buffers.

Each switch organization has its advantages, but also some are easier to implement than others. We will see in the following the most usual switch organizations.

2.1.5.1. Central buffer

In central buffer organization, there is only one central buffer in the switch, which is accessed by all input and output ports. In this case, if L is channel rate and N is the number of ports of the switch, the access rate to the central buffer is $(N + N) \times L$.

In order to achieve such a large bandwidth, the memory wide must be very large. For instance, if the switch must support 4 links of 8 Gigabit/s bandwidth, the aggregated bandwidth is 32 Gigabit/s. If memory access frequency is 500 MHz, then word wide must be 64 bits, since

$$\frac{32 \text{ Gigabit/s}}{500 \text{ MHz}} = 64 \text{ bits}$$

As link speed and number of ports grow, so does the memory word wide. In order to implement this, the total available memory must be organized in many separate memory blocks.

Although it seems that there is no crossbar in this architecture, in fact there is one. In order to connect all the input and output ports with the memory modules, a crossbar organization is needed.

This switch organization is very advantageous for buffer utilization, since all the traffic flows can take advantage of the shared buffer. Moreover, all the output ports have access to packets, without the problems of input-buffered switches.

However, the requirements of memory bandwidth make this switch organization poorly scalable. The memory may grow so wide that packets do not extend across all memory banks. Moreover, even if packets are larger than access word, it is likely that they do not come in word increments and that a lot of bandwidth is wasted with these odd ends.

Finally, flow control is more complex in this architecture, since any flow can use any portion of the buffer. We can fix this by partitioning the buffer in space dedicated to each input channel, but, by doing so, we would lose some of the advantages of the central buffer switch architecture.

2.1.5.2. Output queuing

The output buffer organization consists in a separate buffer for each output port. In this way, incoming packets are stored immediately in their corresponding output port.

In order to achieve this, upon packet reception, they are decoded and the output port is calculated. Afterwards, packets are immediately sent through a crossbar to the output port, where they are stored in a buffer.

The aggregate bandwith of each output port buffer is $(N+1) \times L$. Moreover, the crossbar must work at rate of $N \times L$, since, in the worst case, all input ports may require to inject packets for the same output port.

The flow control in output queuing is complex. The reason is that the buffer space at any output port is shared by all input channels. We can solve this by partitioning the buffers in space reserved for each input, but this is disadvantageous since we lose flexibility in buffer assignment.

The buffers and crossbar required rate make this architecture poorly scalable. For this reason it is not usually proposed for high-speed switches.

2.1.5.3. Input queuing

In input queuing switch organization, there is a buffer at each input port. When packets are received, they are stored in the input port where they arrive. Independently, the crossbar is scheduled matching packets ready at input buffers with free output channels. When a packet is chosen for transmission, it passes through the crossbar and is immediately forwarded through the output link. In this way, there are no buffers at the output ports.

The rate requirements of input buffers are just $(1+1) \times L$, which makes this architecture an excellent choice for scalable switches.

However, input queuing has an important disadvantage that did not have the previous switch architectures: the head-of-line (HOL) blocking [Karol et al. 87]. It happens when a packet at the head of a queue blocks, because it is requesting an output port which is currently busy with another packet. This packet may prevent other packets in the same queue from advancing, even if they request available output links. According to synthetic traffic studies [Hluchyj and Karol 88], the maximum throughput of input-queued switches is below 60%.

There are two commonly accepted solutions for this problem. The first one is called virtual output queues (VOQ) [Dally et al. 98], although it is also known as advanced input queuing. This solution consists in organizing the input buffers in such a way that there are as many queues as output ports. These are dynamical queues and do not require additional bandwidth in buffers. Since packets requesting different output ports are stored in different queues, the HOL blocking is completely eliminated.

The second solution for performance issues in input-queued switches consists in providing some *speed-up* for the switch. This solution is discussed in the next section.

2.1.5.4. Combined input and output queuing

When an input-buffered switch has a crossbar that operates faster than the link rate, the output ports need to implement some buffer to store the additional packets. In this architecture, the memory access rate needed, both at input and output buffers, is $(S + 1) \times L$, where L is the external line rate and S is the speed-up factor (1 means no speed-up).

This architecture can also implement the VOQs at the input ports and provide even better performance. In this way, an scalable solution exists for high-speed switches.

The combined input and output queuing switch architecture is a widely accepted solution in high-performance switches. For this reason, we will assume in the rest of this thesis that this is the architecture implemented in our switch models.

2.1.5.5. Combined input-crosspoint queuing

The type of crossbar most commonly implemented in switches is bufferless, i.e. buffers are either at the inputs, the outputs, at both places, or at a central location. However, there is an old design that is recently getting much attention where there are small buffers at each crosspoint. This "buffered crossbar" or combined input-crosspoint queuing (CICQ) architecture has significant advantages over the previous, traditional bufferless configuration:

- The scheduling task is dramatically simplified; there are no scheduler inefficiencies to be compensated by speedup.
- The crossbar can operate directly on variable-size packets, hence there is no need for segmentation and reassembly circuits; the need for mutually synchronized line cards (at the cell-time level) is also eliminated.
- Internal speedup is not needed, because there is no packet segmentation and no scheduler inefficiencies; hence, the external line rate can be as high as the crossbar line rate.
- The egress path of the switch needs no buffer memory –at least no large, off-chip memory– because packet reassembly is not needed, and because, in the lack of internal speedup, there is no output queue build up; this eliminates a major cost component.

The rate of crosspoint buffers is $(1 + 1) \times L$, but the switch needs $N \times N$ such small buffers. This has two drawbacks: the first is that buffer is very fractionated and, therefore, at a certain point most of the buffers will be likely empty, while space would be necessary at others. The second disadvantage is that this architecture scales poorly when compared with the bufferless crossbar architectures. However, this problem will be attenuated as CMOS technology improves.

2.1.6. Input-queued switch scheduling

In input queued switches, the problem of matching input requests with available output ports is a complex task. When we discussed the crossbars, we hinted that a *scheduler* is responsible of planning the crossbar configuration. We also suggested that the most efficient schedulers work on fixed data amounts or cells.

In this section we discuss the most popular algorithms for input-queued switch scheduling. We also assume the use of VOQ, since this is the case in most of the recent switch designs.

In a switch with N ports, each input has N FIFO virtual output queues, one for each output. When a cell with destination output j arrives at input i, the switch stores it in the VOQ, denoted Q_{ij}.

We can model the scheduling problem as a matching problem in a bipartite graph with N input nodes and N output nodes. The edge between input i and output j is present if Q_{ij} is nonempty; we give it weight w_{ij}, which equals the length of Q_{ij}. Given the transfer constraints in the crossbar, i.e. at most one input connects to one output, a matching for this bipartite graph is a valid schedule.

The maximum-weight matching (MWM) algorithm delivers a throughput of up to 100 percent [McKeown et al. 96, Dai and Prabhakar 00] and provides low delays by keeping queue sizes small. However, it is too complex to implement because it requires $O(N^3)$ iterations in the worst case. Therefore, an efficient design of the overall system (scheduler and switching fabric) requires the best possible compromise between ease of implementation and goodness of throughput and delay performance. Moreover, it has been shown that maximal matching is not always desirable, since it may lead to unfairness under admissible traffic and to starvation under inadmissible traffic [McKeown 95].

In the following, we introduce the most common alternatives for this task.

2.1.6.1. Parallel Iterative Matching

Parallel iterative matching (PIM) [Anderson et al. 92] is a three phase scheduling algorithm. Each of these phases operates independently at each of the output ports:

1. Request. Each unmatched input sends a request to every output for which it has a queued cell.

2. Grant. If an unmatched output receives any requests, it chooses one randomly to grant. The granted input is notified.

3. Accept. If the input receives at least one grant, it chooses one by randomly selecting one of them, thus completing the connection.

The above three steps can be iterated until a maximal matching is found. Since the algorithm does not require coordination among the output ports, it is fast. Other advantages of PIM are that it converges in $O(log(N))$ iterations, and it does not suffer from starvation problem.

Its main drawback is that making a random selection among the members of a time-varying set is difficult and expensive to implement in hardware.

2.1.6.2. iSLIP

The iSLIP algorithm [McKeown 95,McKeown 99] is an improvement of PIM. It replaces the random selection of candidates and grants in PIM by a round-robin selection. The round-robin pointers are updated not after every grant, but only if that grant is accepted by the input in the next step (step 3) of the algorithm. This is the most important contribution of iSLIP, since updating the pointers with every grant leads to a very poor performance [McKeown 95].

The main advantage of iSLIP is the simplicity of its hardware implementation resulting from deterministic round-robin arbiters and a simple request-grant-accept algorithm. Simulations in [McKeown 95] show that iSLIP converges in less than about $log_2(N)$ iterations. iSLIP outperforms PIM in simplicity, speed, throughput, and average latency.

2.2. Quality of Service

The importance of network QoS is widely accepted by both the research community and the manufacturers. However, the problem is that existing networks are not so well prepared for the new demands. Implementing QoS is still a very active research topic, with multiple possible solutions competing against each other. Depending on the network architecture, different techniques have to be taken into consideration. Many research efforts are today performed around the main aspects related to QoS in different environments.

As mentioned earlier, the increasing use of the Internet and the appearance of new applications have been the dominant contributions to the need of QoS. For this reason, it is not surprising that most of the studies are focused on delivering

QoS on the Internet [Ferguson and Huston 98, Xiao and Ni 99]. Many of the services available through the Internet are provided by applications running on clusters. Therefore, the researchers are also proposing mechanisms for providing QoS on these platforms, as we will show later.

More recently, with the advent of different types of wireless technologies, wireless devices are becoming increasingly popular for providing the users with Internet access. It is possible to transmit data with them but also voice, or executing multimedia applications for which QoS support is essential. The QoS mechanisms proposed for wired networks are not directly applicable to wireless networks, and therefore, specific approaches have been proposed [Chalmers and Sloman 99, Bull et al. 99].

Therefore, QoS is a very interesting topic in network design, in all of its specific contexts. Our proposal is focused in cluster interconnects and, thus, we focus on the work which has more relationship with the proposal in this thesis. During the last decade several cluster switch designs with QoS support have been proposed. Next, we review some of the most important proposals.

2.2.1. Traffic requirements of applications

Multimedia applications have grown as important drivers for the need of high-performance networks. This kind of applications can take advantage of the new capabilities of interconnection networks. However, these new applications have additional requirements which are different to the requirements of traditional applications. Multimedia applications integrate several *media*, like video, audio, static images, graphics, text, etc. This multimedia traffic introduces new requirements the network must satisfy.

The QoS requirements of present applications can be grouped in four indices. These parameters were not taken into account in the design of most best-effort networks [Giroux and Ganti 99, Black 00], because they were not as important in the past as nowadays. Let us see which are these four commonly considered indices:

- Bandwidth. The provision of bandwidth to an application means that the network has enough capacity to support application throughput requirements. This means that the network is able to transfer all the information generated by the application without introducing meaningful congestion in the source. This requirement is fundamental for almost any multimedia application to work properly.

- Latency. The latency or delay represents the amount of time taken by a user message to reach its destination. There are applications, like voice or video transmission, which have very restrictive latency requirements, since messages that do not arrive in time are discarded as useless. There-

fore, messages that are delivered late translate into wasted throughput and worse QoS for the applications. The limits of 10 and 100 ms for audio and video, respectively, are commonly accepted (see, for instance, annex G of IEEE standard 802.1D-2004 [IEEE 04]).

- Jitter. The variation of the latency of two consecutive messages received by an application is called *jitter* [Elhanany et al. 05]. In multimedia applications, the receiver expects to receive information in regular intervals. Messages that arrive too ahead in time must be buffered in the destination until the application is ready to consume them. Therefore, if too many messages arrive too early it may happen that the application buffer is overflowed and some packets are discarded. This can happen, for instance, when transmitting video frames for a video-on-demand application. The commonly accepted limits for jitter are the same as for latency, that is, 10 and 100 ms for audio and video, respectively.

- Information loss. Another requirement for multimedia applications is the loss of information. This is important because if some messages are lost, then it would affect the quality perceived by the user. In traditional applications, the loss of data is also a serious problem, but in this case it can be solved with retransmissions. However, the retransmission of data introduces a waste of throughput and additional delays that are usually intolerable in interconnection networks. For this reason, most high-performance networks are lossless.

In order to define which are the actual requirements of a multimedia application, in terms of the indices we have presented before, it is usual to study the quality perceived by the users [Halsall 01]. That means that the bandwidth, latency, jitter, and data loss requirements are chosen in order to obtain a performance in the application level that allows the users to perceive the multimedia application without degradations.

2.2.2. Traffic classes

In the previous section, we talked about QoS from the point of view of the applications. These applications need that the network satisfies some QoS indices. From the network perspective, it is very difficult to consider the requirements of every possible application. Therefore, a set of *traffic classes* are defined to group applications.

The number and characteristics of traffic classes must be diverse enough to satisfy all the users, but, at the same time, it is interesting from the network perspective to be as narrow as possible, to reduce complexity.

Once we have settled for a certain group of traffic classes, it is the responsibility of users to choose which one is the most appropriate for their applications.

Each traffic class is defined by a set of specific QoS requirements. There are several proposals for classifying traffic. A classical one is the introduced by ATM [ATM 95], based on five different traffic classes:

- CBR (*Constant Bit Rate*). This traffic class includes the connections that require fixed bandwidth and low delay. The assigned bandwidth will be always available during the lifetime of the connection, thus it can be used to emulate circuit switching. Some applications that can use this traffic class are telephony, video conference, raw audio and video transmission, and, in general, communications where bounds are needed on delay and bandwidth.

- rt-VBR (*real time-Variable Bit Rate*). In this case, this traffic class is aimed for applications also with delay requirement, but with a variable injection rate, often in bursts of packets. The two characteristics of this traffic are average and peak rate. Instead of using peak rate for bandwidth reservation, the network can make statistic multiplexing in order to save some bandwidth. Examples of applications belonging to rt-VBR are compressed audio and video transmission.

- nrt-VBR (*non real time-Variable Bit Rate*). The applications that also generate bursty traffic, but do not require a short delay, belong to this category. For instance, some kinds of video and audio bradcasts can fit in this category.

- ABR (*Available Bit Rate*). This traffic class was proposed for regular data traffic, like file transfer or e-mail, which does not require service guarantees. Although there are no guarantees on maximum delay or minimum bandwidth, it is desireable that switches provide the best performance that is possible. For this reason, this traffic is also known as *best-effort traffic*.

- UBR (*Unespecified Bit Rate*). This traffic class was proposed for applications that use any excess of network capacity, after all the other traffic classes have been served. In this way, there are no requirements on bandwidth or delay and packets can be safely dropped. Applications using this class can be like ABR applications, but with even less priority.

Other authors have proposed alternatives to this classification. For instance, in [Kim et al. 98] there are only three of the previous classes, since there is no distinction between both VBR classes and UBR traffic is not considered. Another example is [Pelissier 00], which proposes four traffic classes:

DBTS (*Dedicated Bandwidth Time Sensitive*). This kind of traffic requires a guaranteed minimum of bandwidth and also a maximum delay. It would

be similar to ATM's CBR and rt-VBR traffic classes. Interactive applications like videoconference and Voice over IP (VoIP) would belong to this category.

DB (*Dedicated Bandwidth*). This traffic class demands a minimum bandwidth, but it is not too sensible to delay. Therefore, it is similar to ATM's nrt-VBR.

BE (*Best-Effort*). This traffic is usually bursty. In this case, there are no strict requirements of bandwidth or latency. This category is similar to ATM's ABR. The majority of traffic generated by conventional applications belongs to this category. This includes FTP, e-mail, web browsing, etc.

CH (*Challenged*). This traffic class receives a degraded performance in order to avoid that it disturbs BE traffic. In this sense, it is similar to ATM's UBR. An example of traffic of this category would be a backup copy, which would take place in moments when it would not disturb the normal user operation.

There are many other classifications, but we will see just another one. In recent IEEE standards, like for instance IEEE standard 802.1D-2004 [IEEE 04], there are seven traffic classes, briefly described in Table 2.1.

Type	**SC**	**Description**
Control	Network control (NC)	Traffic to maintain and support the network infrastructure characterized by a "must get there" requirement.
QoS	Voice (VO)	Traffic with a limit of 10 ms for latency and jitter.
QoS	Video (VI)	Traffic with a limit of 100 ms for latency and jitter.
QoS	Controlled load (CL)	Traffic subject to some form of admission control based on bandwidth.
Best-effort	Excellent-effort (EE)	The best-effort type services that an information services organization would deliver to its most important customers.
Best-effort	Best-effort (BE)	LAN traffic as we know it today.
Best-effort	Background (BK)	Bulk transfers and other activities that should not impact the use of the network by other applications.

Table 2.1: Service classes suggested by the standard IEEE 802.1D-2004.

2.2.3. QoS models

There are two main perspectives on how to provide QoS to the applications. The difference is on the strictness of the guarantees. When the network is able to provide a hard bound on delay and to guarantee a minimum bandwidth, we talk about *hard QoS guarantees*. On the other hand, when the network will differentiate among the traffic classes, but there is no compromise on specific performance, then we talk about *soft QoS guarantees*.

The two outstanding examples of both models are integrated services (IntServ) [Braden et al. 94] for hard QoS, and differentiated services (DiffServ) [Blake et al. 98, Bernet 98] for soft QoS.

IntServ is an architecture that specifies the elements to guarantee QoS in IP networks. The idea of IntServ is that every router in the system implements IntServ, and every application that requires some kind of guarantees has to make an individual reservation. "Flow Specs" [Partridge 92] describe what the reservation is for, while RSVP [Braden et al. 97] is the underlying mechanism to signal it across the network.

The main problems of IntServ QoS architecture are [Xiao and Ni 99]:

- The amount of information that each intermediate router has to handle and store grows proportional to the number of established connections. This is a very large amount of information, since routers may handle millions of connections.
- The requirements of IntServ routers are very high. All of them must implement the RSVP protocol, a connection admission control, QoS arbiting, etc.
- In order of IntServ to work, all the routers must be able to provide QoS. In this way, a gradual adoption of IntServ would not be possible.

Therefore, IntServ is an architecture that does not scale well. On the other hand DiffServ [Blake et al. 98] is a QoS architecture that specifies a simple, scalable, and coarse-grained mechanism for classifying, managing network traffic and providing QoS guarantees on modern IP networks.

In DiffServ, IP packets must be labeled with a traffic class tag. This tag, known as *DiffServ Code Point* (DSCP), is used at every router to choose where to store packets, how to schedule, when to drop packets, etc.

The only drawback of DiffServ is that there are no strict guarantees of performance for any traffic class. For many applications, this may be alright, but there are some applications that require bounds on latency and throughput to work properly.

2.2.4. Output link scheduling for QoS

A considerable number of output scheduling algorithms has been proposed [Jha and Hassan 02, El-Gendy et al. 03]. These algorithms seek to allocate link bandwidth among the various classes of traffic in *a priori* or fair manner, and provide statistical, aggregate or per-flow guarantees on network parameters such as delay, jitter, and packet loss.

Most scheduling algorithms can be divided into two classes [Zhang 95]: a *work-conserving* scheduler is not idle when there is a packet to transmit in any of its queues and a *non-work-conserving* scheduler may choose to remain idle even if there is a packet waiting to be served. The latter may be useful in reducing burstiness of the traffic or in providing a strict guarantee for a particular class of traffic.

In the following, we offer a brief review of output link schedulers that are interesting for the purposes of this thesis. More details will be given in the next chapters when the algorithms are used.

2.2.4.1. First in first out scheduling

First in first out (FIFO) scheduling provides the simplest scheduling mechanism – packets are served in the order they are received. The delay and packet-loss properties are directly proportional to the buffer size available at the queue. However, no guarantees can be provided to individual flows, and moreover, the FIFO scheduling works best when all flows behave in the same way. Therefore, FIFO scheduling is not suitable for providing service differentiation and QoS guarantees [Jha and Hassan 02].

2.2.4.2. Priority scheduling

A static priority scheduler is based on multiple FIFO queues where each queue is assigned a priority parameter. The queues are served in the order of their priority. There is also a preemptive version of priority scheduling in which a packet from a lower-priority queue already in transmission may be delayed or dropped if a high-priority packet arrives at the interface. However, most routers deploy non-preemptive scheduling, and a small delay is added to any high-priority packet awaiting service while a packet is being transmitted over the outgoing link.

Priority scheduling [Jha and Hassan 02] is able to give predictable performance for the case when all flows entering the scheduler (irrespective of the priority queues) have the same packet size and input rate.

Priority queuing can be implemented easily since it requires maintenance of only a small number of states per queue.

In priority scheduling a packet must wait for all other higher-priority packets to be forwarded. This can cause starvation to lower-priority classes.

2.2.4.3. Generalized processor sharing and variants

To get over the starvation problem in priority scheduling, generalized processor sharing (GPS) scheduling assigns a logical queue for each flow, and the scheduler serves an infinitesimal amount of data from each queue within a given quantum or finite time interval [Parekh and Gallager 93].

GPS is ideal in achieving fair allocation of bandwidth; however, the scheme is not implementable due to the infinitesimal data requirement. Instead, variations such as round robin (RR) [Hahne and Gallager 86] and weighted-round robin (WRR) [Katevenis et al. 91] schemes are often used as simple implementations for GPS.

In RR scheduling [Hahne and Gallager 86], each queue outputs a packet (instead of an infinitesimal amount of data) in a round-robin fashion within each specified time-frame or cycle. This is a fair scheduling method since even the high-throughput or bursty flows output only one packet per cycle. However, this only works if the packet sizes of all flows are equal. As pointed out in [Jha and Hassan 02], for flows with different packet sizes, the fairness in bandwidth allocation is no longer provided. Flows with larger packet sizes will consume more share of bandwidth than flows with smaller packets.

WRR [Katevenis et al. 91] was developed to address this issue. It outputs n packets instead of a single packet from a queue in each cycle where n is a weight[2] of the queue. Therefore, flows with smaller packet sizes can be served more often and get a fair share. However, WRR is not efficient in handling variable-sized packets in a single flow since it works best when the mean packet size of a queue is known a priori.

The deficit round robin (DRR) scheme [Shreedhar and Varghese 96] addresses this problem by keeping track of queues that are not able to transmit a packet in the previous round because of a large packet size, and by adding the deficit (the remainder from the previous quantum) to the quantum of the next round.

2.2.4.4. Weighted fair queueing

A weighted fair queuing (WFQ) scheduler [Demers et al. 90] prevents the starvation problem for the lower priority queues in priority scheduling while trying to approximate the GPS scheduling [Keshav 97]. WFQ calculates the finish time for each packet as if it was served by GPS and then uses this time stamp to order the service of packets.

[2] The weight of a traffic class is its importance relative to the weights of the rest of the traffic classes.

Because WFQ allows a fair share of bandwidth among all the queues, it is one of the most popular scheduling algorithms implemented in commercial routers. It is suitable for traffic with variable-sized packets such as the Internet. One of the disadvantages of WFQ is the need to maintain per-flow queuing information since an appropriate weight for each flow must be generated. Several variations of WFQ have been proposed in order to reduce the complexity of per-flow queuing, such as self-clocked fair queuing (SCFQ) [Rexford et al. 96] and start-time fair queuing (STFQ) [Goyal et al. 96].

2.3. High-Performance Switches with QoS Support

In the last years there have been many proposals of switches with QoS support for clusters, local area networks (LANs), and interconnection networks. In this section, we review the main designs of switches, routers, and interconnection standards that include QoS support. All of them incorporate VCs to provide differentiated service to a number of traffic classes, usually 16. However, there are examples of designs with very few VCs (Avici TSR, ATLAS I) and with hundreds of VCs (MMR). We are revising some of them in the following sections.

2.3.1. Multimedia Router

The Multimedia Router (MMR) [Duato et al. 99] is a router architecture, specifically oriented towards multimedia traffic in local area networks. The MMR takes ideas and techniques already employed in switches for conventional traffic in clusters and LAN/SAN, but it combines them in a novelty way in order to offer adequate support to the different traffic classes which are present in new clusters and LANs. These traffic classes are multimedia flows, including constant bit-rate and variable bit-rate, short control messages and conventional best-effort traffic.

The key characteristics of MMR are shown in Figure 2.4 and are briefly commented in the following:

- **Switch Organization.** The internal switch fabric is implemented as a crossbar. Although some routers use a fully demultiplexed crossbar [Dally et al. 94], with as many ports as VCs, this organization becomes prohibitive when the number of VCs is large. Therefore, in MMR multiplexed crossbar is used.

- **Switching Technique.** In order to be able to provide the necessary QoS to the multimedia flows, a connection oriented scheme must be adopted for their transmission. In this way, resources are allocated to the flows in the

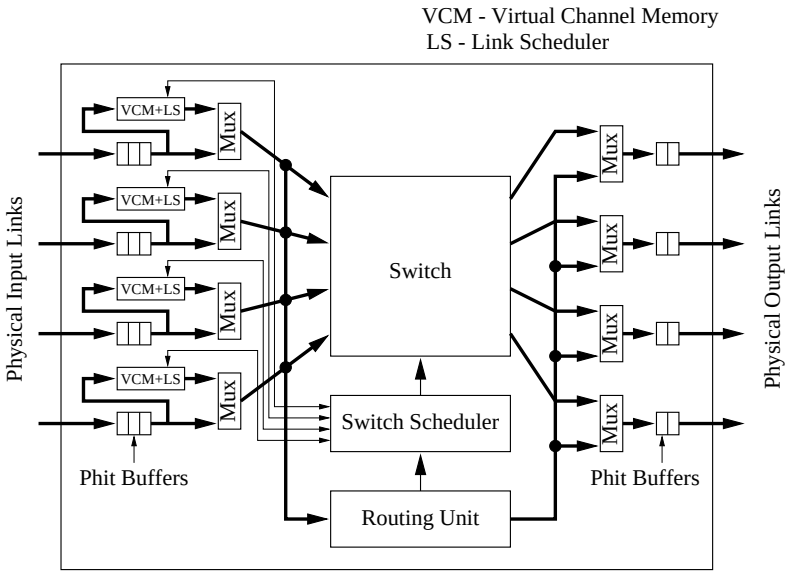


Figure 2.4: MultiMedia Router organization.

connection establishment phase, and, when data are effectively transmitted, delays caused by busy resources will be bounded. On the other hand, conventional best-effort traffic will coexist with multimedia flows. In this case, cut-through switching techniques are the most suitable ones.

Hence, as there is no single switching technique suitable for both kinds of traffic, a hybrid approach is implemented, where the most suitable switching technique is used for each kind of traffic: a connection oriented scheme for the multimedia flows, and a connectionless scheme for best-effort messages.

- **Buffer Organization.** MMR buffers are placed *at the input ports*, because they are better suited to the current trends in link speed (i.e., no speed-up is needed with respect to the line rate). Besides, this is the organization typically used together with the selected switching techniques, and flow control is easier to perform. In order to avoid HOL-blocking [Karol et al. 87], per-connection buffering will be provided as a large set of VCs. This will also help to provide QoS guarantees to multimedia flows.

  Because of the use of a multiplexed crossbar, buffers are not required at the output side. As switch output ports are directly connected to output links, flits are directly transmitted through the switch and the corresponding output link. However, a few phit buffers must be used to pipeline information through the switch and the link.

When a connection is set up, a VC is reserved in every link belonging to the path between source and destination. In this way, every connection can be considered independently from each other, and performing per-flow scheduling is possible. Thus, in order to be able to support a large number of concurrent connections, needed in a multimedia environment, the buffers associated to each link must be organized as a large set of VCs.

Summing up, MMR is a hybrid router. It uses pipelined circuit switching for multimedia traffic and virtual cut-through for best-effort traffic. Pipelined circuit switching is connection-oriented and, therefore, needs a VC for each connection. This is the main drawback of this proposal, since the number of VCs that can be implemented is limited by silicon area devoted for buffers and ports. Consequently, it may happen that the MMR has not enough VCs for all the potential multimedia connections.

2.3.2. MediaWorm

MediaWorm is a wormhole router/switch architecture proposed by Das and others [Yum et al. 02]. The main idea is to reuse the extensive work done in wormhole switching by communications industry and academia, by proposing the minimum necessary changes to adapt a wormhole switch to QoS traffic.

In their work, the authors propose two modifications over the design of a standard wormhole switch. Firstly, they assume that VC are divided into two broad categories, best-effort and real time traffic. Although this is a static partition, a second version of the switch is able to handle dynamic partitions. Although MediaWorm does not assume any number of VCs, in their performance evaluation, the authors propose the use of 16 VCs.

The second proposal they made for wormhole switches consists in replacing the switch's scheduler. Instead of using a round robin or First-In-First-Out (FIFO) scheduler, they propose a priority-based algorithm. More specifically, they modify the connection-oriented algorithm Virtual Clock [Zhang 91] to obtain the Fine Grained Virtual Clock (FGVC). In this way, the switch provides soft QoS guarantees to traffic flows without the need of explicit connection establishment. As opposed to original Virtual Clock, the FGVC offers bandwidth reservation at the message level.

The authors study the capabilities of MediaWorm to support QoS-demanding traffic while there is also traditional best-effort traffic. In this study, they arrive to the following conclusions:

- The FGVC scheduler can provide a good performance for QoS-demanding traffic when compared to a traditional scheduling (round robin/FIFO).
- The performance of QoS-demanding traffic is not affected by the presence of best-effort traffic. As VBR QoS traffic load grows, the adverse effects

are suffered by best-effort traffic. On the other hand, the MediaWorm is able to provide a packet delivery with reduced jitter for VBR QoS traffic up to a global link utilization of between 70% and 80%.

Summing up, MediaWorm is a proposal to update the design of conventional wormhole switches, in such a way that they can better support traffic with QoS necessities. In [Yum et al. 01], the authors propose an improvement to the initial design, which consists in a resource preemption mechanism. This mechanism is used when a high-priority packet arrives at the switch and its output link is currently being used by a best-effort packet, this last packet interrupts its transmission and the output link is assigned to the high-priority packet.

2.3.3. Avici Terabit Switch/Router

The Avici Terabit Switch/Router (TSR) was proposed by Dally and others in 1998 [Dally et al. 98, Dally 99]. Its main innovation was the use of a direct network with a 3-D torus topology for the interconnection of router's line cards.

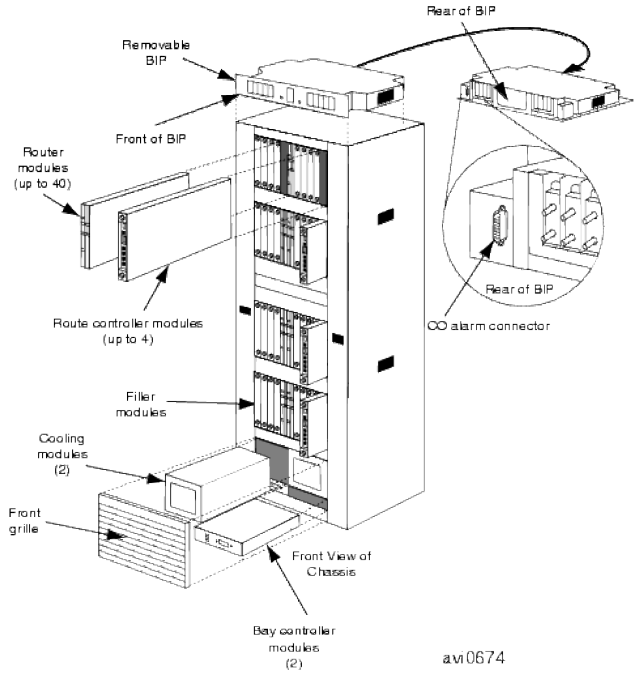


Figure 2.5: Modular architecture of Avici Terabit Switch/Router.

In an IP router, a line card is the part of the router responsible of managing one or more input and output links. These line cards are connected with a switch fabric, which, in this case, is a high-performance direct network.

According to the authors, the main advantages of this router design and organization are:

- Affordable scalability. The 3-D torus can grow up to hundreds of nodes (560 in first generation TSR) with a lineal cost in the number of nodes and with a global bandwidth that grows with each new node.
- Incremental growth. The network of TSR can be expanded by adding new line cards one-by-one. Since all the switch fabric is contained in the line cards (remember that this is a direct network, each line card includes a switch) the customer only needs to buy as many line-cards as he needs.
- Load balancing. The switch fabric with 3-D torus topology of TSR offers a great amount of independent routes between any two nodes. The routing algorithm of the TSR distributes packets along all these paths in order to balance the load in link nodes.
- Fault tolerance. The routes diversity also provides the TSR network with fault tolerance. The network automatically reconfigures around failures in links or nodes, restricting the number of routes to avoid failing devices. With a minimum of two disjoint routes between any two nodes in the network, the TSR is able to support a failure in any component and most combinations of two component failures. All the components can be hot-plugged and, therefore, the network can be repaired, expanded, or upgraded without interrupting service.
- Non-blocking. The TSR uses VOQ at the network level, which means that there is a separate VC for every destination in the network. In this way, two packets that are addressed to two different network outputs are never stored in the same buffer. Therefore, these packets cannot block each other. However, these packets do share the same link bandwidth. In order to assure that there is always available bandwidth, there is a fair arbiting through the network and there is a previous reservation of bandwidth.
- Low and bounded latency for CBR traffic. TSR provides a separate set of VCs for traffic with guaranteed bandwidth. Since this traffic never competes with best-effort traffic for buffers or bandwidth, the TSR is able to transmit CBR traffic up to the guaranteed bandwidth with a bounded delay of 33 μs, regardless of best-effort traffic.

Despite its advantages, the TSR also has some drawbacks. The use of VOQ at the network level would require lots of resources and make the architecture

not very scalable. In order to solve this, the design uses wormhole and only provides space for a flit of each packet in the buffers. Moreover, since most of the time most of the VCs are empty, the authors propose to support a reduced number of queues and just write down which VC each queue corresponds to.

2.3.4. ATLAS I

ATLAS I (ATm multi-LAne backpressure Switch One) [Katevenis et al. 97, Kornaros et al. 98] is a single-chip gigabit ATM switch with optional credit-based flow control. It was designed as a general-purpose building block for high-speed communication in wide, local, and system area networking. It supports a mixture of services from real-time, guaranteed QoS to best-effort, bursty, and flooding traffic, in a range of applications from telecom to multimedia and multiprocessor NOW.

As stated, the switch is implemented in a single chip. This 6-million-transistor 0.35-micron CMOS chip offers: 10 Gbit/s outgoing throughput, sub-microsecond cut-through latency, 256-cell shared buffer containing multiple logical output queues, priorities, multicasting, VP/VC translation, advanced flow control architecture, and load monitoring.

The main characteristics of ATLAS I are:

- The total outgoing throughput is 10 Gigabit/s. It supports flexible configurations:
  - 16×16 configuration, at 622 Mbps/link (per direction), or
  - 8×8 configuration, at 1.24 Gbps/link, or
  - 4×4 configuration, at 2.5 Gbps/link, or
  - 2×2 configuration, at 5.0 Gbps/link, or combinations of the above (e.g. 12 ports at 622 Mbps and 1 port at 2.5 Gbps, etc.)
- Centralized on-chip buffer architecture which supports 256 ATM cells.
- Three priority levels (service classes).
- Each link supports up to 4096 flow groups, the VP/VC translation table is on-chip.
- Credit-based flow control. Although the ATM Forum opted for rate (lossy) rather than credit based flow control, the designers saw the opportunity of using their switch in more environments, rather than just wide area networks. In this way, the credits-flow control makes the switch suitable for high-speed interconnects.

Flow control in ATLAS I works as follows. The central buffer holds space for 256 cells. On the other hand, flow control does not work at the level of individual flows. Since at a given moment, millions of flows could be using the switch, the designers opted to bundle the flows up into flow groups. As stated, the switch supports up to 4096 flow groups identifiers per switch ports. Note that packets belonging to the same flow group would never be able to take over each other.

On the other hand, in order to futher simplify the switch architecture, only one cell per traffic flow may be present at the switch buffer. The restriction of at most one cell per flow group is acceptable in switching fabrics, where round-trip times are on the order of one cell time, but it may be a severe restriction in local area networks environments.

Finally, each input channel has devoted a portion of the 256-cell central buffer. For instance, if the switch will use 16 links, it could be configured 16 cells for each channel. In this way, conditions for a cell to be transmitted into the switch are i) there is no other cell of the same flow group in the stored in the switch; and ii) there are less than 16 cells stored in the switch that came from this link.

ATLAS I implements three levels of priority, each level having its own queues. The shared buffer maintains 54 logical output queues: 3 times 16 outputs plus 1 management port, and 3 multicast queues. These queues are implemented as linked lists of cell buffer pointers.

2.3.5. InfiniBand

The InfiniBand Architecture (IBA) specification [Inf 00] describes a system area network (SAN) for connecting multiple independent processor platforms (i.e. host processor nodes), I/O platforms and I/O devices. The IBA SAN is a communication and management infrastructure supporting both I/O and interprocessor communications for one or more computer systems. The architecture is independent of the host operating system and processor platform.

After a very serious crisis, IBA is ramping up again but its current use focuses almost exclusively on clusters for high-performance computing. However, IBA may expand its market in the future, being implemented in clusters for different application areas that may require QoS support.

IBA is designed around a switch-based interconnect technology with high-speed point-to-point links. An IBA network is divided into subnets interconnected by routers, each subnet consisting of one or more switches, processing nodes and I/O devices. IBA supports any topology defined by the user, including irregular ones, in order to provide flexibility and incremental expansion capability.

IBA links are bidirectional full-duplex point-to-point communication channels, and may be either copper cable, optical fiber or printed circuit on a back-

plane. The signaling rate on the links is 2.5 GHz in the 1.0 release, the later releases possibly being faster. The physical links may be used in parallel to achieve greater bandwidth. Currently, IBA defines three link bit rates: 2.5 Gbps, 10 Gbps, and 30 Gbps (referred to as 1x, 4x, and 12x, respectively).

IBA's basic unit of communication is a message. Messages are segmented into packets for transmission on links and through switches. The packet size is such that after headers are considered, the Maximum Transfer Unit (MTU) of data may be 256 bytes, 1KB, 2KB, or 4KB. Each packet, even those for unreliable datagrams, contains two separate CRCs, one covering data that cannot change, and another covering data that changes in switches or routers which means that the packet must be recomputed during the trip.

IBA switches route messages from their source to their destination based on forwarding tables that are programmed during initialization and network modification. The forwarding table can be linear, specifying an output port for each possible destination address up to a switch-specific limit, indexed by that address; or random, initialized by storing {destination, output port} pairs. The number of ports of a switch is vendor-specific, but is limited to 256 ports. Switches can be cascaded to form large networks. Switches may also optionally support multicast routing.

Routing between different subnets (across routers) is done on the basis of a Global Identifier (GID) 128 bits long, modeled over IPv6 addresses. On the other hand, the addressing used by switches is with Local Identifiers (LID) which allow 48K endnodes on a single subnet, the remaining 16K LID addresses being reserved for multicast.

IBA management is defined in terms of managers and agents. While managers are active entities, agents are passive entities that respond to messages from managers. Every subnet must contain a single master subnet manager, residing on an endnode or a switch that discovers and initializes the network.

The IBA transport mechanisms provide both connection-oriented and datagram services, and these services could be reliable (acknowledged) or unreliable. Although QoS could be provided for any transport mechanisms used, applications should use reliable connections in order to be able to carry out resource allocation and provide them with a QoS guarantee.

Basically, IBA has three mechanisms to support QoS: service levels, virtual lanes, and virtual lane arbitration. IBA defines a maximum of 16 service levels (SLs), but it does not specify what characteristics the traffic of each service level should have. Therefore, it depends on the implementation or the administrator how to distribute the different existing traffic types among the SLs.

IBA provides fields for marking packets with a class of service. By allowing the traffic to be segregated by categories, we will be able to distinguish between packets from different SLs and to give them a different treatment based on their needs.

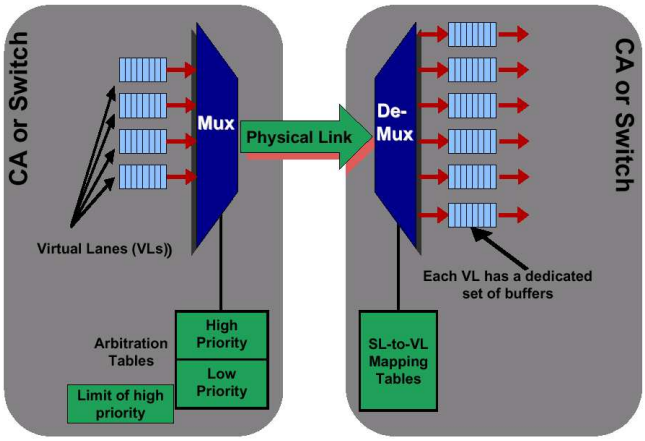


Figure 2.6: Virtual lanes in a physical link.

IBA ports support virtual lanes, providing a mechanism for creating multiple virtual links within a single physical link. A virtual lane is an independent set of receiving and transmitting buffers associated with a port (Figure 2.6). Each virtual lane must be an independent resource for flow control purposes.

IBA ports have to support a minimum of two and a maximum of 16 virtual lanes ($VL_0 \ldots VL_{15}$). All ports support VL_{15}, which is reserved exclusively for subnet management, and must always have priority over data traffic in the other virtual lane. Since systems can be constructed with switches supporting different numbers of virtual lane, the number of virtual lane used by a port is configured by the subnet manager. Also, packets are marked with a service level (SL), and a relation between SL and virtual lane is established at the input of each link with the *SLtoVLMappingTable*.

When more than two virtual lane are implemented, an arbitration mechanism is used to allow an output port to select which virtual lane to transmit from. This arbitration is only for data virtual lanes, because VL_{15}, which transports control traffic, always has priority over any other virtual lane. The priorities of the data lanes are defined by the VLArbitrationTable.

The structure of the VLArbitrationTable is shown in Figure 2.7. The VLArbitrationTable has two tables, one for scheduling packets from high-priority virtual lanes and another one for low-priority virtual lanes. However, IBA does not specify what is high and low priority. The arbitration tables implement weighted round-robin arbitration within each priority level. Up to 64 table entries are cycled through, each one specifying a virtual lane and a weight, which is the number of units of 64 bytes to be transmitted from that virtual lane. This

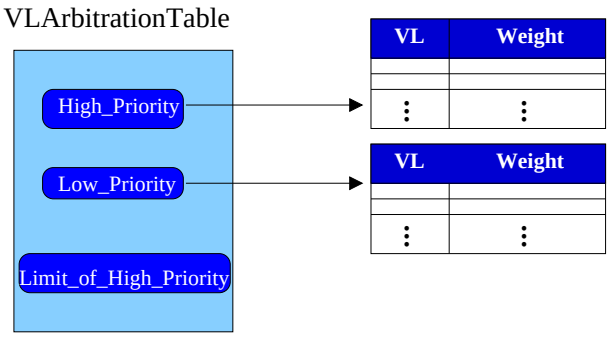


Figure 2.7: VLArbitrationTable structure.

weight must be in the range of 0 to 255, and is always rounded up as a whole packet.

A LimitOfHighPriority value specifies the maximum number of high-priority packets that can be sent before a low-priority packet is sent. More specifically, the virtual lanes of the High_Priority table can transmit a maximum of *LimitOfHighPriority*×4096 bytes before a packet from the Low_Priority table can be transmitted. If no high-priority packets are ready for transmission at a given time, low-priority packets can also be transmitted.

2.3.6. PCI Express Advanced Switching

AS is a multipoint, peer-to-peer switched-fabric architecture designed to provide, in an open standard, the functionality of the proprietary interconnects that have been at the core of storage, communications, and embedded computing systems. The AS Interconnect Special Interest Group [ASI] published v1.1 of the AS specification [Adv 05].

AS architecture is built upon the data link and physical layers established by the PCI Express architecture [PCI 03] to achieve widespread interoperability and cost-effective reuse of technology. The physical layer consists in a dual-simplex channel that is implemented as a transmit pair and a receive pair. A data clock is embedded using the 8b/10b encoding scheme. The initial frequency is 2.5 Gb/s, but the bandwidth of a link may be linearly scaled by adding signal pairs to form multiple lanes. In AS, the maximum packet size is 2176 bytes. A credit-based flow control protocol ensures that packets are only transmitted when the buffer at the other end is able to receive those packets. Finally, Virtual Cut-Through switching is used in AS.

The AS supports unicast and multicast traffic. For unicast traffic the AS transaction layer provides source-based path routing versus the memory-mapped routing of PCI Express. By eliminating the hierarchical structure of memory-

mapped routing, flexible topologies can be constructed such as star, dual-star, full mesh, or multi-stage networks.

AS encapsulates data packets and attaches to them a header that routes the packets through the fabric, regardless of the original packet format. This header contains a protocol interface field that is used at the packet destination to determine the packet's format. Thus, nearly any transport, network, or link layer protocol can be routed through an AS network.

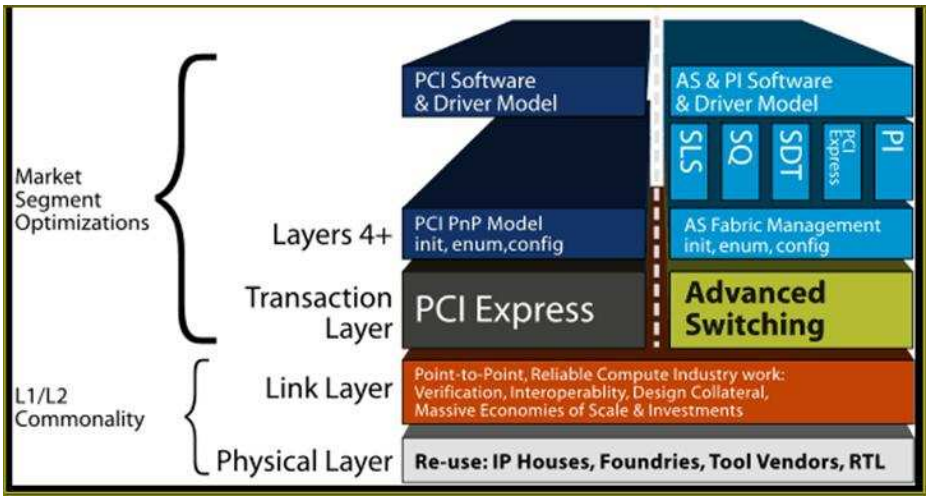


Figure 2.8: AS switch with input and output buffers.

2.3.6.1. AS architecture

AS provides mechanisms that permit QoS to be supported. Specifically, an AS fabric permits us to employ traffic classes (TCs), VCs, egress link scheduling, and an admission control mechanism to provide a different treatment to the traffic.

VCs provide a means of supporting multiple independent logical data flows over a given common physical channel. AS supports up to 20 VCs of three different types: Up to 8 bypassable unicast VCs, up to 8 ordered-only unicast VCs, and up to 4 multicast VCs. The bypassable VC with the highest number in each network element (usually VC 7) is called the Fabric Management Channel.

The AS packet header contains a field with a TC identifier. This field permits us to specify one of eight possible TCs. Moreover, the AS packet header specifies the type of VC that the packet employs. The packet's TC identifier and the VC type are transmitted unmodified from source to destination through an AS

fabric. At each hop within an AS fabric, the TC identifier is used to apply VC selection of the appropriate type. Each VC type (bypassable or ordered) is governed by a distinct TC/VC mapping.

AS defines two egress link schedulers to resolve between the up to 20 VCs competing for bandwidth on the egress link: The VC arbitration table scheduler and the minimum bandwidth egress link scheduler. A given implementation may choose either of them or may implement its own proprietary mechanism.

The VC arbitration table scheduler, or just table scheduler, provides an implementation of the Weighted Round Robin (WRR) algorithm, proposed by Katevenis et al. [Katevenis et al. 91]. The VC arbitration table is a register array with fixed-size entries of 8 bits. Each table entry, which contains a VC identifier value, corresponds to a slot of a WRR arbitration period. When arbitration is needed, the table is cycled through sequentially and a packet is transmitted from the VC indicated in the current table entry. If the current entry points to an empty VC, that entry is skipped. The number of entries of the AS VC arbitration table may be 32, 64, 128, 256, 512, or 1024.

The minimum bandwidth egress link scheduler, or just MinBW scheduler, is intended for a more precise allocation of bandwidth regardless of packet size. This scheduler consists of two parts. The first is a mechanism to provide the Fabric Management Channel (FMC) with an absolute priority over the other VCs. However, FMC has its bandwidth limited by a token bucket. The second part of the MinBW egress link scheduler is a mechanism to distribute bandwidth amongst the rest of the VCs according to a specification of relative weights.

AS does not specify an algorithm or implementation for the MinBW scheduler, but only its behavior. However, according to the specification, several well-known scheduling algorithms exhibit the desired properties of the MinBW scheduler. Examples include variants of Weighted Fair Queuing (WFQ) [Demers et al. 90] such as Self-Clocked WFQ [Golestani 94], and variants of WRR such as Deficit WRR [Shreedhar and Varghese 96]. The properties that the MinBW scheduler must exhibit are [Adv 03]:

- Work conserving: If there are packets ready to be injected through an output link, this link must not be kept idle.
- Minimun bandwidth guarantee: Each VC has to have an associated guaranteed minimum bandwidth.
- Bandwidth metering: The bandwidth allocation to the different VCs must take into account the size of packets. That is, bandwidth is measured in terms of information, not in terms of sent packets.
- Fair redistribution: If there is some extra bandwidth, not allocated to any VC, it must be distributed among VC proportionally to the minimum guaranteed bandwidth for each VC.

- Memoryless: If a VC is assigned some bandwidth, but there are no ready packets to use it, this bandwidth is not reserved and cannot be recovered by this VC later.

A connection admission control implemented in the fabric management software may regulate the access to the AS fabric. It would allow new packet flows entry to the fabric only when sufficient resources were available. Fabric management software may track resource availability by monitoring AS fabric congestion and tracking active packet flows and their bandwidth. This is very useful when traffic flows are predominately connection-oriented and carefully rate-limited.

2.4. Conclusions

In this chapter we have reviewed the state of the art in high performance interconnects. We have briefly reviewed the different aspects of this kind of networks that will be useful to understand our proposals in the next chapters.

We have also reviewed the concept of Quality of Service (QoS). There are many applications today that demand from the network something more sophisticated than a best-effort service. The idea is that, for these applications to work properly, the network has to satisfy several requirements in terms of some QoS indices.

After that, we have reviewed the mechanisms proposed for QoS support in high-speed switches. There are two points where arbitration and, therefore, QoS must be taken into account. First, there is crossbar scheduling, which selects which packets at the input ports progress to output ports. Second, there is output port arbitration, which selects packets to be injected through the link.

Finally, in this chapter we have also reviewed recent proposals for high-speed interconnects with QoS support. In addition to academic proposals, we have also reviewed the well known standards InfiniBand and PCI AS.

CHAPTER 3

Efficient Traffic Class-Level QoS Support

THE use of a limited number of traffic classes (TCs) for QoS provision is a popular framework in high-performance networks. This is like the *DiffServ* [Blake et al. 98] proposal for Internet, which consists in assigning a virtual channel (VC) for each TC. This approach can be found in the recent InfiniBand and PCI Advanced Switching (AS) standards, and in other proposals as we have seen in the previous chapter.

The cost of implementing many VCs is high since the switch ports and the scheduler are much more complex. Therefore, there are few implementations that include more than two or four VCs.

In this chapter we review the use of VCs for QoS provision and conclude that they are an inefficient solution. We propose a new technique to provide QoS with only two VCs based on the elimination of redundancies in traffic scheduling.

Later, we evaluate a switch architecture using this proposal in terms of hardware requirements. Finally, we also evaluate the performance of our proposal through simulation.

3.1. Virtual Channels for QoS Provision

This section is about VCs and their use for QoS provision. We review their advantages and disadvantages, and their implementation cost. Finally, we explain why VCs are not efficient when handling bursty traffic.

Virtual channels were originally introduced to solve the problem of deadlock in wormhole-switched networks, as we saw in the previous section. In the context of QoS provision for high-speed interconnects, a VC is usually assigned to each TC.

In this way, interference from one TC over others is avoided. Separate VCs also open up the possibility of per-TC QoS provision. In this section we review the advantages and the disadvantages of VCs.

3.1.1. Advantages of virtual channels

Virtual channels are popular in many proposals for QoS support in high-performance switches. In the following, we review which are the problems that they solve.

3.1.1.1. Head-of-line blocking

Head-of-line (HOL) blocking is a well known problem in input-buffered switches [Karol et al. 87]. It happens when a packet at the head of a queue blocks, preventing other packets in the same queue from advancing, even if they request available resources (typically, an output link).

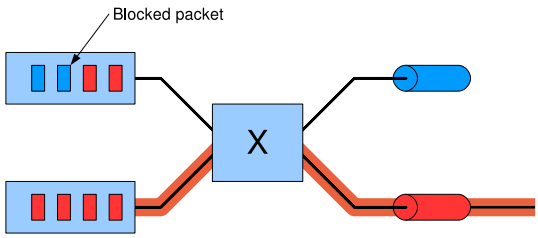


Figure 3.1: Head-of-line blocking.

HOL blocking is illustrated in Figure 3.1[1]. We can see in the figure that the blocked packet requires the topmost channel, which is available. However, the packet cannot advance because it is waiting behind another packet, which is requiring a busy channel.

[1] In the figure, small rectangles are packets, larger boxes are buffers, the square in the middle is the crossbar, and the cylinders are links. Colours of packets indicate the desired output link.

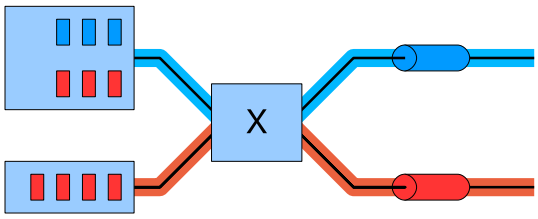


Figure 3.2: Head-of-line blocking solved with VCs.

A similar situation can happen with packets of different TCs. If all packets share the same buffer, a packet from one TC can block packets from other TCs if the first packet is blocked by the scheduler.

We can put traffic flows from different TCs in separate VCs and avoid the HOL blocking. The buffer structure would consist of a queue per VC and the packets at the head of all these queues would be candidates for the scheduler.

In Figure 3.2 we can see the resulting architecture. In this case, both channels can be fully utilized, since packets are stored in separate queues. Likewise, if resources are VCs instead of links, the solution would be similar.

3.1.1.2. Buffer hogging

Buffer hogging is a problem similar to HOL blocking. It happens when all the available buffer space is used up by blocked packets. In this way, packets in a upstream buffer would not be able to advance, even if they would request available resources downstream.

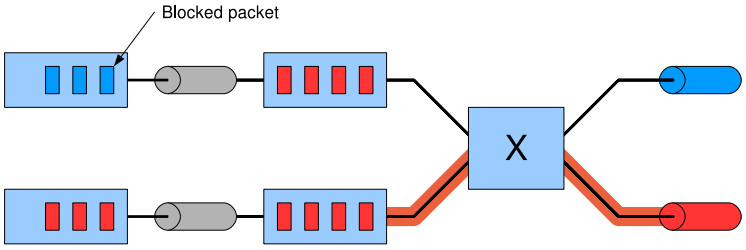


Figure 3.3: Buffer hogging.

Figure 3.3 shows the buffer hogging problem. The blocked packet cannot advance to the next buffer because it is full of packets. Note that the blocked packet requires the topmost link, which is available, while the packets in the next buffer are waiting a busy resource. For this reason, buffer hogging degrades performance.

The problem of buffer hogging can arise related with QoS support mechanisms. It may happen that the blocked packets belong to a TC that is being delayed by the scheduler. In this case, packets from other TCs waiting in a previous stage would not be able to advance.

Buffer hogging is solved by using VCs, where each TC is assigned a different VC. Note that it is also necessary to separate flow control at a VC level. This means that each VC has its own credit counter and that no VC can occupy all the buffer space. In this way, buffer hogging is avoided.

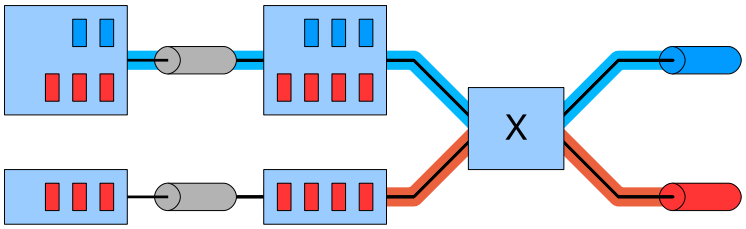


Figure 3.4: Buffer hogging solved with VCs.

In Figure 3.4 we show the implementation of this solution. Each VC holds packets demanding different resources. Moreover, the space that a VC can take is limited and, therefore, no TC can cope the buffer.

3.1.1.3. Traffic scheduling

We have seen in the previous chapter that, in order to provide QoS, the scheduler has to be aware of the TC of packets. If all the packets are stored in the same queue, it is difficult to provide a differentiated service or to provide QoS guarantees to individual TCs.

A popular solution, implemented for example in InfiniBand and AS, takes advantage of the assignation of a different VC to each TC, allowing the scheduler at the switches to merge traffic from the different VCs according to a QoS algorithm.

3.1.2. Disadvantages of virtual channels

Although VCs can help to solve the aforementioned problems, they are not enough by themselves. In order to provide good performance, the network requires more elements than only VCs.

3.1.2.1. Head-of-line blocking

As stated, the HOL blocking is avoided between different TCs by using VCs. However, they do not solve HOL blocking between packets of the same TC. This problem can dramatically degrade performance of input-buffered switches and, therefore, a solution is needed.

Packets that belong to the same TC are always stored in the same VC. These packets may be requiring different output ports. However, packets are stored in arrival order in the same queue. Therefore, if the first packet of the queue is waiting for a busy output port, other packets in the same VC, but demanding a different and available output port, would be unnecessarily blocked.

Virtual Output Queues (VOQs) [Dally et al. 98] is considered the most effective way to avoid HOL blocking. In this case, there are at each switch port as many queues as endpoints in the network, and any incoming packet is stored in the queue assigned to its destination. The aim of this policy is to prevent flows addressed to congested destinations from sharing queues with those flows addressed to non-congested destinations, thereby avoiding HOL blocking.

Note that in order to implement VOQ and, at the same time, provide several VCs for several TCs, the number of queues is multiplied. For instance, in a network with 64 end-nodes and 16 TCs, a total of $64 \times 16 = 1024$ queues per switch port are needed. Therefore, such an architecture would be very difficult to implement.

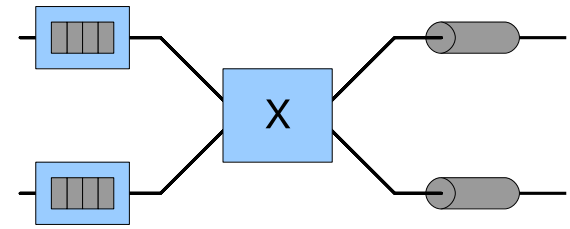
A variation of VOQ uses as many queues at each port as output ports in a switch [Anderson et al. 93], reducing so queue requirements. However, still many queues are needed. For instance, if switches have 16 ports and, again, there are 16 TCs, a total of $16 \times 16 = 256$ queues per switch port are needed. Although this number of queues could be implemented with some technologies, the scheduling process would be much more difficult, and its implementation could be unfeasible for high-speed interconnects.

Figure 3.5 illustrates the different switch organizations we have discussed in this section. We can see that each improvement increases the number of queues. This may lead to organizations that are difficult to implement in practice.

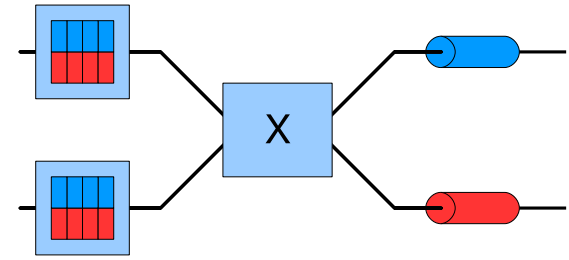
3.1.2.2. Buffer hogging

This problem is solved among TCs by using VCs and separate flow control domains. However, packets of the same TC can still cause buffer hogging to happen. For instance, a long burst of packets towards the same destination could prevent packets of the same TC but addressed to a different end-node to advance.

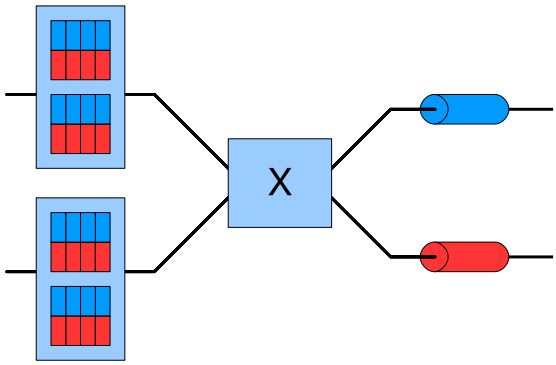
A solution to avoid this problem is again Virtual Output Queues at the network level. However, as stated, this is hard to implement, specially combined



(a) Simple architecture



(b) VOQ



(c) VOQ + 2 VCs

Figure 3.5: Different switch organizations with VOQ and VCs.

with VCs. On the other hand, bursts of packets can be avoided by using an admission control and token buckets, which are usually available in the latest technologies.

3.1.2.3. Traffic scheduling

In order to provide QoS guarantees, it is not enough to implement VCs at the switches. For instance, a great deal of packet delay is spent in the end-nodes, waiting to be injected in the network. If the end-nodes do not implement also VCs and QoS-aware scheduling, the performance would suffer.

Moreover, VCs are not enough to provide strict guarantees of bandwidth or jitter. VCs avoid interference from other TCs, but nothing prevents packets of the same TC to oversubscribe a link. If the end-nodes issue more packets that a link or end-node is able to handle, the delays of these packets will grow dramatically and throughput will not be the ideal.

In order to provide bandwidth and latency guarantees, a connection admission control is mandatory. In this way, before starting a flow of packets, the network only accepts flows for which it can guarantee performance.

3.1.3. Implementation of virtual channels

Virtual channels are a very well studied mechanism. In standards for high-speed interconnects, like InfiniBand or AS, the behavior of VCs is very similar. We will review it in this section and indicate how they would be implemented.

3.1.3.1. Logical view of virtual channels

From a QoS point of view, a VC is a dedicated buffer for a TC. That means that each TC has its own buffer at any switch port. Moreover, these buffers have individual flow control and their size does not depend on the occupancy of other VCs.

The standards usually give freedom to designers on how to handle the buffers dedicated to VCs. For instance, it is usually possible to dynamically implement VOQ at the switch level. In this way, a VC would compromise several queues, but the buffer would be shared among them. In this way, there is no limit on the size of any queue and one of them could grow and take all the available space.

Regarding scheduling, standards give indications and constraints that have to be accomplished, but the designers can implement whatever they want within these limits. For instance, AS gives a list of properties that the scheduler has to exhibit, but the designer can choose any algorithm that presents these properties.

3.1.3.2. Physical view of virtual channels

In the previous chapter, we showed how to implement dynamic queues in a buffer. There is a single SDRAM memory per port and the queues are managed with pointers. In this way, each VC would be a different queue.

Since it is usual to implement VOQ at switch level, each VC consists of a group of queues. The flow control is handled with a separate credit counter for each VC. Any of the dynamic queues of a single VC can take all the space assigned to the VC, but it will never grow in the space of a different VC.

Note that packets from all queues and VCs are stored in the same physical memory. Therefore, the implementation of VOQ does not require additional memory. However, this technique requires additional pointers and management logic to handle several dynamic queues.

3.1.3.3. Implementation cost

When implementing VCs, although the physical memory is the same, there are several overheads. Let us look at them in detail.

- The use of VCs increases the cut-through latency of the switch. This means that the delay of a packet that crosses the switch is higher when there are VCs. The reason is that it takes time deciding in which queue the packet should be stored. Also, the management of dynamic queues is more complex than the management of a simple FIFO queue. Moreover, the scheduler of the switch and the arbiter at output links have to decide between many more candidates, which increases the delay. On the other hand, this delay can be masked with pipelining techniques, but this increases the switch complexity.

- The switch requires more silicon area to be implemented. The reason is that all the components are now more complex. The input ports need additional logic to decide in which queue packets must be stored. Also, for each dynamic queue, additional pointers and counters are needed. Moreover, each VC requires a separate credit counter. The switch scheduler is much more complex because there are many more candidates to schedule. Finally, at output ports the buffers are also more complex and the arbiter has to consider more packets.

Therefore, the higher cut-through latency and the extra silicon area are two important drawbacks that arise due to the implementation of a complete set of VCs in each switch port.

3.1.4. Buffer efficiency

A switch is a system where information is transferred from inputs to outputs. We call *throughput* to the amount of information per time unit that is transferred. Given a large time-frame, outgoing throughput matches incoming throughput. However, instantaneous throughput may differ a lot from average throughput.

Buffer memory is needed in order to smooth out the fluctuations in instantaneous throughput.

A burst is a persistant difference between the rate at which packets come to the switch and the rate of departure. In other words, a burst is a period of time while instantaneous throughput is higher than average throughput. The buffers of the switch must be large enough to accommodate the bursts in order to offer good performance.

If buffers are not large enough to accommodate bursts, then packets must be delayed and the switch will not be operating at full capacity, and thus, performance will suffer. In the following, we study how the implementation of VCs affects to the efficiency of buffers to cope with bursty traffic.

3.1.4.1. Performance of virtual channels with bursty traffic

The VCs must have independent flow control in order to work properly. However, this also means that the buffers are statically partitioned among VCs: one of them cannot use space reserved for the others.

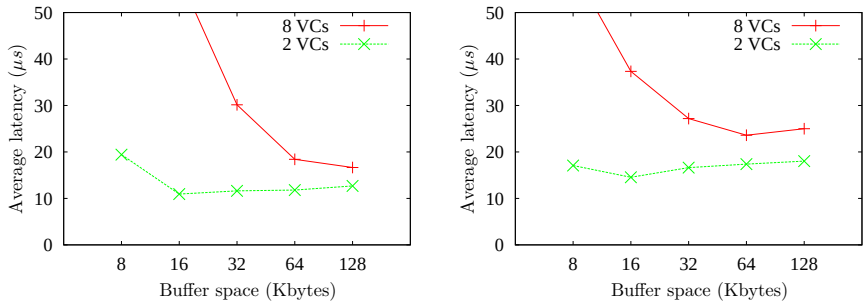
On the other hand, the switch accommodates packets from a number of flows. The length of bursts depends on the peak rate of these flows. If we separate these flows in TCs, the peak rate of each group will not be significantly smaller than the peak rate of the aggregate, provided that all flows are similar.

For instance, let us assume that our switch has to transfer 1000 video sequences through one of its inputs. We have enough confidence that at a certain point, a maximum of 100 sequences will be transmitting at peak rate. Now, if we separate the sequences in four TCs, and we put each group in a separate buffer, we cannot assume that at maximum only 25 flows of each class will be transmitting at peak rate. In fact, the worst case will be somewhere between 25 and 100.

Therefore, even though the buffer for each TC is 1/4 the original, the peak rate of each TC is more than 1/4 the original peak rate. For this reason, the partitioned buffers are less efficient. Or, in other words, to be able to accommodate the same burst length, partitioned buffers must be larger than a combined buffer.

This theoretical result can be confirmed through simulation. At Figure 3.6 we can see a little experiment to illustrate this. We inject 8 service levels of bursty traffic into a network. We evaluate two alternatives, one with eight VCs (one per TC) and another with only two VCs. The total buffer space per port in both alternatives is the same, only the management changes[2]. In the plot, we see the average latency of the four top-most priority service levels. We can see

[2] Packet size is tuned so there is always one packet size plus a round trip time (RTT) of buffer per VC. On the other hand, latency results are at message level, which in all the cases involves several packets.



(a) Synthetic (self-similar) (b) Multimedia (MPEG-4)

Figure 3.6: Average latency of QoS traffic with different buffer sizes per port (Input load = 100% link capacity; 64 end-nodes MIN; 16 port switches).

that the two-VC design requires only 16 Kbytes of buffer per port to achieve the best performance, while the eight-VC alternative needs as much as 128 Kbytes per port to achieve the same performance.

We can conclude that the use of VCs introduces a waste of buffer space due to the static management of the available space. When a burst of packets is injected, traffic from a congested TC cannot use buffer space assigned to a different TC, even if it is available.

The problem presented in this section has no easy solution. A first idea would be to allow packets from one VC to use memory assigned to another VC. However, this could lead to buffer hogging. One could devise a complex scheme of flow control, where a VC can take space from another, but only up to a certain limit. However, the specification of VCs in recent high-speed interconnect standards, like InfiniBand or AS, is very clear and there is not a solution compatible with these standards, as long as VCs are implemented.

3.1.5. Summary

Virtual channels are a popular solution for QoS provision and are proposed in many switch designs with QoS support. The VCs are used to avoid HOL blocking and buffer hogging. Moreover, they are also proposed to separate different TCs and enable QoS-aware scheduling.

However, we have seen that VCs rely on a number of other mechanisms and techniques that also have to be implemented. A connection admission control, QoS aware end-nodes, and VOQ at least at switch level are needed to offer the desired performance.

We have also seen that the implementation of VCs is expensive. A switch design with many VCs requires complex buffers and scheduler. Also, the cut-through delay of the switch is increased. For this reason, there are very few high-speed interconnect implementations of switches with more than two or four VCs.

Finally, we have seen that VCs introduces a static partition of buffers. This means a waste of buffer space and, therefore, it also means a worse performance under bursty traffic.

3.2. Efficient QoS with Two Virtual Channels

Most proposals to provide QoS in clusters incorporate 16 or even more VCs, devoting a different VC to each TC. This increases the switch complexity and required silicon area. Moreover, it seems that, when the technology enables it, the trend is to increase the number of ports instead of increasing the number of VCs per port [Minkenberg et al. 03].

In most of the recent switch designs, the buffers are the most silicon area consuming part (see [Simos 04] for a detailed design). The buffers at the ports are usually implemented with a memory space organized in logical queues. These queues consist of linked lists of packets, with pointers to manage them. Therefore, the complexity and cost of the switch heavily depend on the number of queues at the ports. For instance, the crossbar scheduler has to consider 8 times the number of queues if 8 VCs are implemented (greatly increasing the area and power consumed by this scheduler). Then, a reduction in the number of VCs (and in the required buffer space) necessary to support QoS can be very helpful in the switch design and implementation.

In this section, we show that it is enough to use only two VCs at each switch port for the provision of QoS. One of these VCs would be used for QoS packets and the other for best-effort packets. We also explore a switch design that takes advantage of this reduction and we evaluate it with realistic traffic models.

Although using just two VCs is not a new idea, the novelty of our proposal lies in the fact that the global behavior of the network is very similar as if it had many more VCs. This is easily achieved by taking advantage at the switches of the scheduling made at end-nodes.

Note that our proposal does not intend to achieve better performance than other designs with many more VCs and able to devote a different VC to each TC. However, our proposal intends to achieve almost the same performance than other implementations with many more VCs, but using just two VCs.

Simulation results show that our proposal provides a very similar performance compared with a traditional architecture with many more VCs both for the QoS traffic and the best-effort traffic. Moreover, comparing our proposal

with a traditional architecture with only 2 VCs, our proposal provides a significant improvement in performance for the QoS traffic, while for the best-effort traffic the traditional design is unable to provide the slightest differentiation among packets of the same VC.

3.2.1. Motivation

In modern interconnection technologies, like InfiniBand or AS, the obvious strategy to provide QoS support consists in providing each TC with a separate VC. Separate VCs allow the switches to schedule the traffic in such a way that packets with more priority can overtake packets with less priority. In this way, head-of-line (HOL) blocking between packets of different TCs is eliminated. Moreover, buffer hogging is also avoided, since each TC has its own separate credit counter.

However, we have seen in the previous section that VCs are expensive to implement. The switch architecture is more complex because the buffers have to manage several dynamic queues. Moreover, flow control is also more complex, because several domains have to be considered. Besides, the switch scheduler has to take into account more packets, leading to a more complex design and to longer delays. For these reasons, there are very few implementations with all the VCs proposed by specifications.

We have also seen that VCs aim to solve several problems in high-performance networks. HOL blocking can be avoided by separating the different TCs in independent VCs. However, in order to fully eliminate this problem, a separate VC per destination and TC would be necessary, and this is totally unfeasible. For a similar reason, the buffer hogging problem cannot be completely eliminated.

VCs are also proposed to provide QoS guarantees. For instance, recent standards propose to implement WRR [Katevenis et al. 91] at the VC level. However, we have seen that, in order to provide traffic with the required performance, there are other mechanisms necessary. More specifically, a connection admission control (CAC) must be implemented to provide bandwidth guarantees. In addition to this, the end-nodes also must implement VCs and a QoS-based scheduling. Otherwise, implementing VCs only at the switches would not be enough.

Finally, we have also shown in the previous section that VCs are not an efficient solution. In order to work properly, the VCs must provide a static partition of available buffer space. In this way, it may happen that packets from a TC are blocked waiting for credits in a buffer that may have plenty of available space, but for a different TC.

Taking into account all the previous information regarding VCs, we think that they are not a satisfactory solution for the QoS provision problem in high-performance networks, in the terms of a VC per TC that are usually assumed. In this way, a QoS provision based on a VC per TC would be inefficient. Hence,

the lack of implementations seen in commercial products for interconnection networks.

The main motivation of our work is to provide a similar QoS with less resources. Therefore, we do not aim at achieving a higher performance but, instead, at drastically reducing buffer requirements while achieving similar performance and behavior of systems with many more VCs. In this way, a complete QoS support can be implemented at an affordable cost.

3.2.2. Observation of traffic scheduling

In order to provide the applications with the QoS they demand, the end-nodes must be QoS-aware. To achieve this, it is necessary to provide separate queues for each TC and a scheduler which implements the QoS policy.

Therefore, the end-nodes have to examine all the packets generated by applications and decide which ones to inject first, based on some QoS criteria. This implies that the stream of packets leaving the end-node and entering the network contains implicit information regarding the relative priority of these packets. Let us examine this in detail.

3.2.2.1. Bandwidth-based scheduling

Bandwidth-based scheduling consists in providing bandwidth guarantees to TCs. In this way, each TC is associated to a VC. Besides, the end-nodes and the switches implement a scheduling policy that takes into account these bandwidth requirements.

Popular algorithms for bandwidth-based scheduling are WRR [Katevenis et al. 91], DRR [Shreedhar and Varghese 96], and WFQ [Demers et al. 90] variants. In all the cases, it is necessary that every network component is aware of the reserved bandwidth and the presence of a CAC mechanism. This last element ensures that bandwidth requirements are feasible and starvation will not happen to QoS-demanding TCs.

When a given end-node schedules packets for transmission based on relative bandwidth requirements, it produces a mix of packets from different flows, which are multiplexed over the link connecting to the corresponding switch input port. Therefore, if the switch is receiving packet streams from several interfaces, two conditions are met:

- The proportion of packets from each flow in the incoming streams is roughly proportional to the bandwidth requirements for each flow (provided that the sample is large enough).
- The outputs of the switch have enough bandwidth to cope with all the incoming packets. This is ensured by the CAC mechanism.

In these circumstances, separating at the switch the incoming packets into several VCs, one per TC, would be redundant. If all the packets are put in a FIFO queue per input port, the percentage of packets from each flow stored at the queue still corresponds to their relative bandwidth requirements. Thus, there is no need to alter packet ordering and, therefore, there is no need to demultiplex incoming packets at a given switch input port into several VCs. If these input ports are served with a round-robin algorithm, no TC would suffer starvation, provided that each flow has bandwidth reservation, and streams of packets leaving the output ports would also have the correct proportions of packets.

We have seen that a switch that receives streams of packets in the correct proportions and that does not have any of its output links oversubscribed, can produce at the outputs streams of packets also in the correct proportions, while using a single VC. This situation would also happen for the next switch in the flow path due to recursivity. Therefore, the whole network can offer bandwidth guarantees without additional VCs at the switches.

3.2.2.2. Delay-based scheduling

Delay-based scheduling is based on the concept of *deadline*. The deadline is the desirable time for a packet to reach destination. Therefore, each packet is assigned a deadline tag used for scheduling. There are many proposals on how to assign these deadlines, but once they are set, the behavior of network components is the same. At every buffer in the network, both in end-nodes and switches, packets must be kept in deadline order and when resources are available, an Earliest Deadline First (EDF) policy is applied.

In ideal conditions, packets leave the end-nodes in deadline order. This means that if a packet leaves earlier than another, it has also a earlier deadline. In this case, the incoming streams of packets arriving at a switch are also ordered. Therefore, separating again the packets in a VC per TC would be redundant. With packets already in order, it is possible to just consider the first packet at each queue, in the confidence that packets coming afterwards have higher deadlines.

The behavior of the switch would be analogous to a sorting algorithm: if the switch has in its input ordered chains of packets and has to produce at the output an ordered sequence, it only needs to look at the first packet of each input.

3.2.2.3. Conclusions

After this study of the behavior of the traffic scheduling, the main conclusions from our observation are:

- There is a significant amount of redundancy in the scheduling decisions performed at end-nodes and switch fabric. Some of the scheduling decisions made at end-nodes to multiplex flows through the link connecting to the switch fabric can be reused at the switch fabric while keeping the same overall behavior (including QoS guarantees, fairness, and performance).
- A single FIFO queue has to be implemented in the switch ports. This can be done reusing scheduling decisions at the end-nodes when scheduling packets at the switch and provided that there is an admission control.

These observations are used in the next section in order to draft our proposal.

3.2.3. Our proposal

Based on the previous observation, we present a proposal to provide QoS with a minimum of resources. This proposal does not aim at achieving better performance, but at dramatically reducing implementation costs by eliminating redundancies in traffic scheduling, while performance is kept.

In this section we assume a QoS framework based on a discrete amount of *traffic classes* (usually 16, like in InfiniBand or AS). We will see details on the emulation of per-flow QoS in the next chapter.

3.2.3.1. Details on the proposal

Let us see how our proposal works. At the end-points, there are schedulers that take into account the QoS requirements of TCs. Therefore, packets leaving the interfaces are ordered by the interface's scheduler. Thus, if packet i leaves earlier than packet $i+1$, it is because at that moment it was the best decision (with this scheduling strategy) to satisfy the QoS requirements of both packets, even if packet $i+1$ was not at the interface when packet i left. Therefore, we can assume that the order in which packets leave the interfaces is correct.

For the purposes of the switches, it is enough to assume that in all the cases packet i has more priority than packet $i+1$. In this case, the switch is receiving at its input ports ordered streams of packets. Now, the task of the switch is similar to that of a sorting algorithm. The switch inspects the first packet at each stream and chooses the one with the highest priority, building at its output another ordered flow of packets.

Note that what "priority" means will depend on the actual scheduling at the network interfaces. For instance, if absolute priority between TCs is applied, then the scheduler at the switches has to consider the original priority of the packets at the head of the queues, instead of just whether they are regulated traffic or not. If, for instance, the switch has four ports, the scheduler examines the first packet of the four buffers and chooses the packet with the highest

priority. This is not very complex because very efficient priority encoder circuits have been proposed [Huang et al. 02]. Note that this cannot lead to starvation on the regulated traffic because the CAC mechanism assures that there is enough bandwidth for all the regulated flows.

Thereby, by using this scheduler, the switches achieve some reutilization of the scheduling decisions made at network interfaces. This is because the order of the incoming messages is respected, but at the same time, the switches merge the flows to produce a correct order at the output ports. Note that a different scheduler, like round-robin or iSLIP, would not merge the packets in the best way because it would not take into account packets priority. Therefore, the latency of the packets with the highest priority would be affected and results would not be optimal.

If the network has to provide service to TCs that do not demand bandwidth guarantees, like *best-effort* TCs, these TCs should use a different VC. Note that our proposal uses 2 VCs in order to support both regulated and unregulated traffic. If all the traffic in the network could be regulated, we would use just one VC.

In this way, the switch organization would consist of two VCs at each switch port (inputs and outputs). In order to avoid any influence on the regulated traffic, we give the QoS VC absolute priority over best-effort VC. Using just two VCs at the switches and provided that there is regulation in the traffic, we will obtain very similar performance as if we were employing many more VCs.

3.2.3.2. Advantages of our proposal

The most obvious advantage of our proposal is that we achieve with only two VCs a performance similar to proposals involving many more VCs. The immediate advantages of this feature are that the buffers and the switch scheduler are more simple and efficient. In this way, the cost of the switches is drastically reduced and QoS can be implemented at an affordable cost.

Another advantage is that buffers are not statically partitioned in many flow control domains, but only two. This means that the assignation of buffers is more flexible. We saw in the previous sections that implementing many VCs prevents us from fully using the buffer space, due to the static partition that VCs introduce. For this reason, when using our proposal with only two VCs, performance is better under bursty traffic.

3.2.3.3. Limitations of our proposal

Let us examine possible limitations of our proposal. When using bandwidth-based QoS, it may happen that end-nodes inject more traffic from a TC than the amount reserved. The reason for this is the *burstiness* of traffic: even if

the aggregate of average bandwidth requirements is correct, traffic sources do not usually inject packets in *constant bit rate*. This could disrupt the expected proportions of packets at the outputs of the switches and affect to bandwidth results of all TCs. This problem can be solved in several ways:

- Using a CAC strategy that takes into account peak bandwidth. In this way, instead of just taking into account the average bandwidth, the CAC mechanism also considers peak rate and maximum burst length of the flows.
- Providing large buffers at the switches, to be able to absorb the packet bursts. Sometimes this is not possible, due to the limitations of on-chip memory.
- Using a traffic shaping mechanism, like *token buckets*. The *token buckets* can be configured alongside with the CAC mechanism and offer very good performance.

A combination of the previous strategies offers the best performance. Note that a traditional switch design with a VC per TC could better isolate misbehaving TCs. However, the performance would still be affected by packet bursts. This means that the techniques discussed above would also be necessary in that case.

If using priority or deadline-based scheduling, the main limitation of our proposal is that packets may not always come ordered from the interfaces. It may happen that when no more high-priority packets are available, a low-priority packet is transmitted, especially if the end-nodes are *work-conserving*. If this packet has to wait at a switch input queue, and other packets with higher priority are transmitted from the network interface, they would be stored in the same VC as the low-priority packet (both packets are QoS-requiring), and be placed after it in the queue. Thus, the arbiter would penalize the high-priority packets, because they would have to wait until the low-priority packet is transmitted. But this situation, which we call *order error*, has a small impact on performance because there is bandwidth reservation for QoS packets. This means that all the QoS packets will flow with short delay.

In the performance evaluation section, we will see that the order errors have a low impact on the performance. However, they make latency and jitter more variable. Although the average value will remain similar, peak values will be slightly increased.

In general, switches using our proposal are not able to reschedule traffic as freely as they would be with a technique where a different VC for each TC were implemented. This problem is attenuated by the connection admission, because connections are only allowed if we can satisfy their bandwidth and latency requirements all along their path. That means that connections are established as

if all the VCs were implemented at the switches and there were also the same schedulers as in the switches with all the VCs. In this way, we ensure that the required QoS load is feasible. We will not obtain exactly the same performance, but it will be very similar.

On the other hand, the best-effort TCs only receive coarse-grain QoS, since they are not regulated. However, the interfaces are still able to assign the available bandwidth to the highest priority best-effort TCs and, therefore, some differentiation is achieved among them. If stricter guarantees were needed by a particular best-effort flow, it should be classified as QoS traffic. Therefore, although best-effort traffic can obtain a better performance using more VCs, the results do not justify the higher expenses.

Note that this proposal does not aim at achieving a higher performance but, instead, at drastically reducing buffer requirements while achieving similar performance and behavior of systems with many more VCs. In this way, a complete QoS support can be implemented at an affordable cost.

3.2.4. Summary

The traditional strategy to provide QoS support in interconnection technologies like InfiniBand or AS consists in providing each TC with a separate VC. Only devoting a VC per TC at the switches is not enough to provide adequate QoS and other techniques and mechanisms are necessary. More specifically, a CAC mechanism is needed to provide bandwidth guarantees; end-nodes also have to implement VCs; and head-of-line blocking and buffer hogging must be dealt with, at least using virtual output queuing (VOQ) at the switch level.

We have observed that, once the previous conditions are met, traffic travels seamlessly through the network; congestion, if any, only happens temporarily. Therefore, regulated traffic flows with short latencies through the fabric. In this case, to devote a different VC to each TC might be redundant.

Our proposal consists in reducing the number of VCs at each switch port needed to provide flows with QoS. Instead of having a VC per TC, we propose to use only two VCs at switches: one for QoS packets and another for best-effort packets. In order for this strategy to work, we guarantee that there is no link oversubscribed for QoS traffic by using a CAC strategy.

If we can offer the same QoS with just two VCs, this opens up the possibility of using the remaining VCs for other concerns, like adaptive routing or fault tolerance. Furthermore, it is also possible to reduce the number of VCs supported at the switches, thereby simplifying the design, or increasing the number of ports. We will examine the implications of our proposal on the switch design in the next section.

3.3. Switch Architecture

In this section, we describe thoroughly the architecture of a switch using our proposal. We will study a switch model similar to those proposed for InfiniBand or AS. In the next chapters we will study two variations of this design, aimed for different problems.

In the following, we study a 16 port, single-chip, virtual cut-through switch intended for clusters/SANs and for a 8 Gb/s line rate. We assume QoS support for distinguishing two traffic categories: QoS-requiring and best-effort traffic. Credit-based flow control is used to avoid buffer overflow at the neighbor switches and network interfaces. For the rest of the design constraints, like packet size, routing, etc., we take AS [Adv 05] as a reference model.

3.3.1. Switch organization

The block diagram in Figure 3.7 shows the switch organization. We consider a *combined input output queued* (CIOQ) switch because it offers line rate scalability and good performance [Chuang et al. 99]. Moreover, it can be efficiently implemented in a single chip. This is necessary in order to offer the low cut-through latencies demanded by current applications. Moreover, this also allows to provide some internal speed-up, without the need of faster external links.

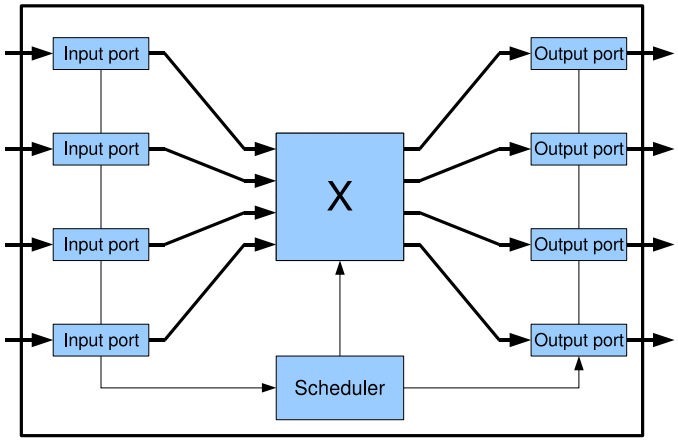
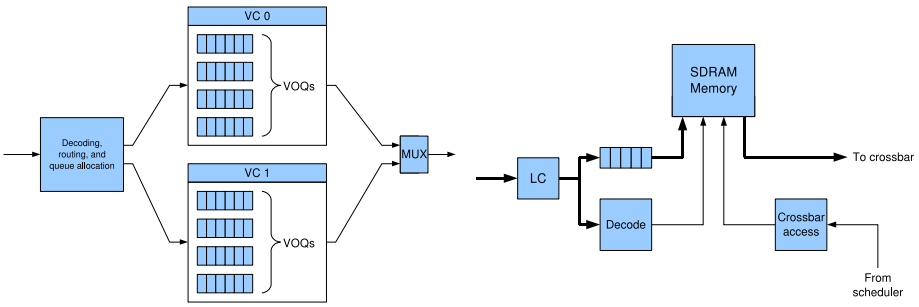


Figure 3.7: Switch organization.

In the CIOQ architecture, output conflicts (several packets requesting the same output) are resolved by buffering the packets at the switch input ports. Packets are transferred to the switch outputs through a crossbar whose configuration is synchronously updated by a central scheduler. To cope with the

inefficiencies of the scheduler and packet segmentation overheads[3], the crossbar core operates faster than the external lines (internal speed-up). Thus, output buffers are needed, resulting in the CIOQ architecture. In this architecture, the memory access rate needed (including input and output accesses) is $(S+1) \times L$, where L is the external line rate and S is the speed-up factor (1 means no speed-up).

By contrast, the required memory access rate is $(N+1) \times L$ for *output queuing* architecture and $(N+N) \times L$ for *shared memory* switches, where N is the number of switch ports. These access rates make these architectures less adequate for our design (usually, $N >> S$) with the performance and technology we are aiming at. On the other hand, the *buffered crossbar* architecture has good performance and the memory access rate matches the line rate. However, with the technologies we are considering, it would be expensive to implement the required buffer space at the crosspoints. Detailed information on these alternative architectures can be found in [Duato et al. 02, Dally and Towles 03].



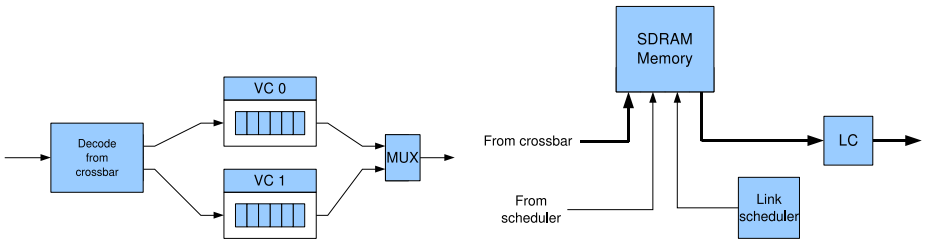
(a) Input port logical view (b) Input port physical view

Figure 3.8: Input port architecture.

The organization that we propose for a switch input port can be seen in Figure 3.8. There are only two VCs: VC 0 is intended for QoS traffic, while VC 1 is intended for best-effort traffic. Each VC is further dynamically divided into 16 queues, which correspond to each switch output port. These are logical queues which share the same physical memory and implement virtual output queuing (VOQ) at the switch level [Anderson et al. 93]. In contrast, the memory space used for VC 0 and VC 1 is statically partitioned to avoid buffer hogging.

The output ports of the switch (Figure 3.9) are simpler: there are only three queues, one per VC plus one for the outgoing credits (not shown in the figure). These queues, although sharing the same memory, are implemented in a static partition of the memory. Since this design is intended for a network

[3] Crossbars inherently operate on fixed size cells and thus external packets are traditionally converted to such internal cells.



(a) Output port logical view (b) Output port physical view

Figure 3.9: Output port architecture.

with multiple switches, some head-of-line blocking [Karol et al. 87] may appear on the VC 1 (best-effort traffic), although it would only affect the non-regulated TCs. Note that congestion in VC 0 (QoS-requiring traffic) is avoided by the CAC mechanism.

In the evaluation section, we will use weighted round-robin for the end-nodes. In this case, the best way to schedule our switches is the following. There is a strict precedence of VC 0 (QoS traffic) over VC 1 (best-effort traffic). Among the queues inside each VC, a simple FIFO algorithm is applied. The scheduling algorithm is very similar to iSLIP [McKeown 99]. However, iSLIP was proposed as a *cell-mode* scheduler: external packets are split into fixed size internal cells which are scheduled ignoring which cell belongs to which packet. In that case, packet reassembly is required at the switch output, and cut-through cannot be used. Since we want to provide virtual cut-through switching, our scheduling decisions are made for whole packets (*packet-mode* scheduling [Marsan et al. 02]). In this way, once a packet is selected by the scheduler, the crossbar connection is kept until all cells of the packet have been delivered to the output. This allows the output port to start transmitting the packet on the line as soon as the first cell of the packet arrives at the switch output.

3.3.2. Design evaluation

In the following, we study the silicon area, the power consumption, and the expected cut-through latency of the switch architecture proposed in the previous section. We consider 0.18 μm and 0.13 μm technologies, because they are popular in interconnection components and plenty of information is available.

3.3.2.1. Silicon area

In order to find out the area requirements of this design, we consider the individual components of the switch core. These are the buffers, the crossbar, and the scheduler. Table 3.1 shows area estimates for each module.

Module	Technology 0.18 μm	Technology 0.13 μm
Buffers ($32 \times 16 Kbytes$)	64 mm^2	32 mm^2
Crossbar and datapath	10 mm^2	5 mm^2
Scheduler	5 mm^2	3 mm^2
Total	79 mm^2	40 mm^2

Table 3.1: Area consumption by components.

The internal clock of the system is 250 MHz and the datapath is 64 bits wide. This provides a speed of 16 Gbit/s, which is twice the speed of the external links. That means there is an internal speed-up of 2.0.

The number of buffers at the switch comes from 16 ports per input and output. We chose 16 Kbytes per port (which are shared between the two VCs, 8 Kbytes each) as a compromise between silicon area and performance. The memory area estimates are based on datasheets of typical ASIC technologies available to European universities. The crossbar and datapath estimates come from the actual numbers of the switch design in [Simos 04]. Finally, we base our estimates for the scheduler area on the data provided by McKeown in [McKeown 99].

Note that usually there is not a full utilization of the available area in an ASIC design [Weste and Harris 05] and, therefore, the final chip would be larger. In order to find out more accurate estimates, all the design flow should be performed. However, these area estimates are very helpful to compare the alternative architectures.

3.3.2.2. Power consumption

In order to figure out the power consumed by this design, we follow a similar methodology to that of the previous section: we will analyze the power consumption of each individual component. Note that power consumption heavily depends on the activity of the different components and, therefore, on the load of the system. In the following, we consider worst-case power consumption.

In Table 3.2, we see the estimates for the different elements (since these are estimates, we round the results to one decimal). Serializer-Deserializer circuits (also known as transceivers) are the most power consuming part of the switch [Minkenberg et al. 03]. According to [Younis et al. 01], the transceivers consume

Module	Technology 0.18 μm	Technology 0.13 μm
Transceivers	9.0 Watt	6.4 Watt
Buffers ($32 \times 250MHz$)	4.0 Watt	2.6 Watt
Crossbar and datapath	3.0 Watt	1.8 Watt
Scheduler	0.9 Watt	0.5 Watt
Total	16.9 Watt	11.3 Watt

Table 3.2: Peak power consumption by components.

175 mW per 2.5 Gb/s of full duplex bandwidth, in 0.18 μm technology. The figure drops to 125 mW in 0.13 μm technology. As our switch design provides an aggregate throughput of 128 Gb/s, we obtain the results shown at Table 3.2.

We also consider power consumed by the memory at the ports. The worst case would happen when all the memory ports are accessed simultaneously. The buffers that we assume (250 MHz, 64 bits wide, two access ports) typically consume 0.5 mW per MHz in 0.18 μm and 0.32 mW per MHz in 0.13 μm.

For the crossbar and datapath, and for the scheduler, we obtain our numbers from the theoretical study at [Wang et al. 02], using the appropriate parameters for the equations at [Wang 04]. Moreover, we have compared these figures with those at [Simos 04], confirming that the results are quite reasonable.

3.3.2.3. Cut-through latency

Finally, we calculate the expected delay of the header of a packet crossing our switch. We assume a pipelined design of the switch, as is usually the case in current high performance switches. The stages of our design to process a message are:

- Header decode/Routing/VC allocation. Routing is very quick since we use source routing (as in AS). We can assume that this is partially performed in conjunction with the next stage.
- Block allocation. Since we are using dynamic queues, blocks at the memory of the input ports must be allocated for the incoming data. This allocation involves the management of linked lists of these blocks. For a detailed description of these algorithms, consult [Duato et al. 02, Dally and Towles 03].
- Writing and scheduling. Scheduling of a block can take place in parallel with its storage at the input buffer.
- Crossbar traversal. This operation consists in transferring the blocks through the crossbar to the output buffers.

- Output scheduling. Since there are two queues at output ports, some scheduling is needed. Note that there is no need for dynamic queues: VC 0 and VC 1 statically share the memory space at the outputs. In addition to these VCs, there is a third queue for the outgoing credits, since we assume that there are no special lines for flow control.

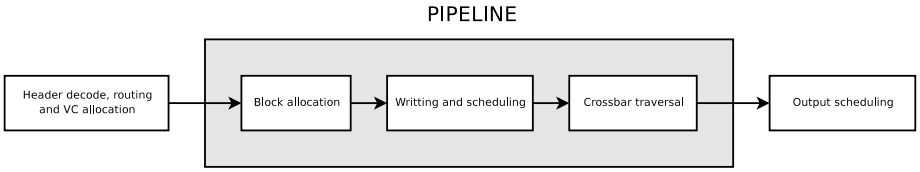


Figure 3.10: Packet processing at the switch.

We can see in Figure 3.10 the stages of processing a message. The first and the last operations are performed very quickly, while the operations in the shaded box have to process whole blocks of 64 bytes. These three block-size operations form the pipeline stages. Taken the former into account, our switch model differs from other canonical models [Duato and López 94, Peh and Dally 01] in the output buffering and the corresponding scheduling.

The latency of the first operation would be 1 cycle, which translates to 4 ns. The time to process a block of 64 bytes with a clock cycle of 4 ns and with a datapath of 64 bits would be 32 ns. The latency of the three stages operating over these blocks is 3×32 ns $= 96$ ns. Finally, the output scheduling could also be performed in a single cycle, since very efficient priority encoder circuits have been proposed [Huang et al. 02]. Therefore, the total latency would be $4 + 96 + 4 = 104$ ns. In addition to this, the latency of the transceiver should be added. It would depend on the specific circuit used, but at [Xil 06] there is a very detailed timing analysis of one that would take around 40 ns; a complete implementation process would be necessary for more accurate delays.

Finally, we have confirmed these results using the theoretical analysis at [Peh and Dally 01], which is itself based on the theory of logical effort [Sutherland et al. 99]. This theory provides a simple and broadly applicable method for estimating the delay of high-speed integrated circuits. The results we obtain in this way are in the same order of magnitude of the 104 ns value.

3.3.3. Summary

In conclusion, our design is feasible with the current technology. Note that if we were to implement a full number of VCs, like it is proposed in the specifications of AS or InfiniBand, then much more buffer space per port would be

needed: each VC should have an amount of memory proportional to round trip time (RTT). With the trend of increasing RTT and line rate [Minkenberg et al. 03], this amount would be large and, therefore, the number of ports of the switch would have to be reduced to keep within reasonable limits of area, power, and latency. Note that, as technology improves, this problem persists: Less ports can be implemented in the chip if a lot of VCs/buffer are needed at each port.

After this hardware characteristics study, we proceed to examine, through simulation, the performance of this switch architecture in the following section.

3.4. Simulation Conditions

In this section, we will explain the simulated network architecture. We will also give details on the parameters of the network and the load used for the evaluation.

3.4.1. Simulated architecture

In sections 3.5, 3.6, and 3.7, we evaluate our proposal varying different parameters. In most of the cases, we will use several switch architectures, which we discuss in the following. Also, in most of the cases we consider 8 different TCs, and hence there are 8 VCs at end-nodes.

The switch architectures considered for the performance evaluation are the following:

- *New 2 VCs*. This architecture is based on our proposal, which uses just 2 VCs at each switch port (the network interfaces still use 8 VCs). There are two variants of this proposal, depending on the number of ports of the switches:
  - *New 2 VCs-P*. This architecture has 32 Kbytes/port and 16 ports/switch.
  - *New 2 VCs-B*. This architecture has 64 Kbytes/port and 8 ports/switch.

  In both cases, the total buffer memory per switch is the same.
- *Traditional 8 VCs*. We have also performed tests with switches using 8 VCs (as many VCs as TCs). However, we have also included the same two variants as in the case of using our proposal. Therefore, we have:
  - *Traditional 8 VCs-P*. This architecture has 32 Kbytes/port and 16 ports/switch.
  - *Traditional 8 VCs-B*. This architecture has 64 Kbytes/port and 8 ports/switch.

These switches have the same memory per port than their 2-VC counterparts. Therefore, the four switch models would use the same total memory and roughly the same silicon area. Note that, actually, *Traditional 8 VCs* cases are more complex and would incur higher delays, which would bias the comparison towards our proposal, but this is not taken into account in our performance evaluation for the sake of clarity.

- *Traditional 2 VCs.* In some tests we will also show performance of a *Traditional 2 VCs* architecture. In this case, both the switches and the end-nodes implement only 2 VCs. The performance of this proposal will be poor due to the lack of VCs at the end-nodes, but it is useful for comparison purposes.

Using these switch architectures we cover a broad spectrum of both traditional solutions and variations over our basic proposal.

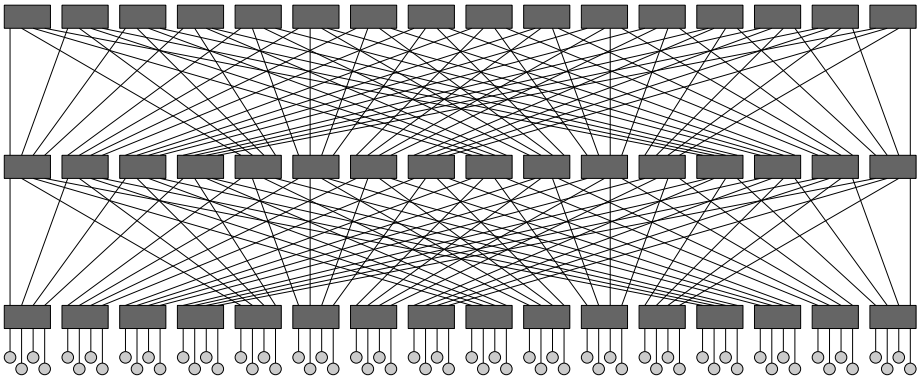


Figure 3.11: Bidirectional multi-stage interconnection with 64 end-nodes and 8-port switches.

The network used to test the proposals is, in most cases, a butterfly multi-stage interconnection network (MIN) with 64 end-points (Figure 3.11). The actual topology is a folded (bidirectional) perfect-shuffle. We have chosen a MIN because it is a usual topology for clusters. However, our proposals are valid for any network topology, including both direct networks and MINs. In fact, we have also performed tests using meshes.

The network interfaces implement the full number of VCs in all the cases (including the *New 2 VC* cases), except when using the *Traditional 2 VCs* architecture. There is a queue per end-point and VC combination and, finally, the length of those host-interface queues is infinite, to simulate the large buffers at these devices.

Remember that our proposal is based on eliminating redundancies by taking advantage of the scheduling performed at network interfaces. Therefore, we have to configure the interfaces properly to achieve the desired results. We will use three types of schedulers in the tests:

- Absolute priorities. In this case, some TCs have absolute priority over others, but the CAC can avoid that starvation happens.
- Table-based Weighted Round Robin. This is a proposal contained in both InfiniBand and AS standards and, therefore, is very interesting to study.
- Weighted Fair Queuing. This algorithm has been regarded as very appropriate for QoS, but is also considered too complex to be implemented. Nevertheless, it is also interesting to test if our proposal can emulate it.

The CAC we have implemented for QoS-requiring traffic is a simple one, based on average bandwidth. Each connection is assigned a path where enough bandwidth is assured. The CAC guarantees that less than 70% of bandwidth is used by QoS traffic at any link. The other 30% of available bandwidth could be used by non-regulated traffic. We also use a load-balancing mechanism when a QoS connection is stablished, which consists in assigning the least occupied route among the possible paths.

3.4.2. Traffic model

Table 3.3 presents the characteristics of the traffic injected in the network in most cases. We have considered the TCs defined by the IEEE standard 802.1D-2004 [IEEE 04] at the Annex G, which are generally accepted for interconnection networks. However, we have added an eighth TC, *Preferential Best-effort*, with a priority between *Excellent-effort* and *Best-effort*. In this way, the workload is composed of 8 different TCs: four QoS TCs and four best-effort TCs.

Name	% BW	Packet size	Notes
Network Control	1	[64,512] bytes	self-similar
Audio	16.333	128 bytes	CBR 64 KB/s connections
Video	16.333	[64,2048] bytes	750 KB/s MPEG-4 traces
Controlled Load	16.333	[64,2048] bytes	CBR 1 MB/s connections
Excellent-effort	12.5	[64,2048] bytes	self-similar
Preferential Best-effort	12.5	[64,2048] bytes	self-similar
Best-effort	12.5	[64,2048] bytes	self-similar
Background	12.5	[64,2048] bytes	self-similar

Table 3.3: Traffic injected per host.

The proportion of each category has been chosen to provide meaningful results. Our intention is to lead the network to a situation where the different TCs

have to compete for limited resources. We also want to have diversity between the sources, combining different packet sizes and different traffic distributions, that is, constant bit rate (CBR) flows combined with variable bit rate (VBR) flows. It is possible that this mix of traffic is not actually present in a real-life cluster, but it serves perfectly to show the advantages of the different architectures we are testing.

Real-life packet destinations are not uniformly distributed; rather, traffic tends to be focused on preferred or popular destinations. This being so, a flexible destination distribution model based on Zipf's law has been proposed [Breslau et al. 99]. Therefore, the destination pattern we have used is based on Zipf's law [Zipf 65], as recommended in [Elhanany et al. 05]. Zipf's law states the frequency of occurrence of certain events, as a function of the rank. In this way, the probability that an arriving packet is heading toward a destination with rank i is given in the next expression, where i is the rank of packet destination, k is the Zipf order and N is the number of addresses.

$$Zipf(i) = \frac{i^{-k}}{\sum_{j=1}^{N} j^{-k}}$$

In this way, the traffic is not uniformly distributed, but, instead, for each TC and input port a ranking is established among all the possible destinations. Therefore, there will be destinations with a higher chance of being elected by a group of flows, where this probability is obtained with the aforementioned Zipf's law. The global effect is a potential full utilization of the network, but with a reduced performance compared with a uniform distribution. If the Zipf order k is fixed to 0, then the destination distribution would be uniform. On the other hand, experimental research [Breslau et al. 99] has found that the value of $k = 1$ is the closest to real traffic.

In our tests, the packets are generated according to different distributions, as can be seen in Table 3.3. *Audio*, *Video*, and *Controlled Load* traffic are composed of point-to-point connections of the given bandwidth. Note that *Audio* traffic models both the audio part of the video transmissions and plain audio connections.

The self-similar traffic is composed of bursts of packets heading to the same destination. In that case, the packets' sizes are governed by a Pareto distribution, as recommended in [Jain 91]. In this way, many small size packets are generated, with an occasional large size packet. The periods between bursts are modelled with a Poisson distribution. With this distribution, if the burst size is long (60 packets, approximately 10 Kbytes), there is a lot of temporal and spatial locality and should show worst-case behavior because at a given moment, many packets

are grouped going to the same destination. The length of the bursts will be noted as the B parameter in the figures.

3.4.3. Simulation results

In the next sections, the performance of our proposals is shown. We have considered the next QoS indices for this performance evaluation:

- Throughput. This is the amount of information transferred each time unit. We measure it in percentage of network capacity.
- Latency. This is the delay of a packet since it is created until it arrives at destination. This value can be split in network latency and interface latency.
  - The network latency is the delay since the packet is sent to the first switch until it arrives at destination.
  - The interface latency is the delay since the packet is created until it is injected in the network.
  - Some application messages are larger than network maximum transfer unit (MTU). For instance, a video frame from a video sequence is much larger than typical MTU. In this case, the application messages generate several packets. When this happens, we can show latency of individual packets and latency of the global message, when the last part is received.
  - For some applications, it is useful to show, in addition to average latency, maximum values. However, maximum values may vary a lot and, thus, are not very useful. For that reason, we use a quantile, usually the 99th percentile.
- Jitter. The jitter measures the variation of latency. However, there is not a consensus on how to actually measure jitter. We use the absolute difference between the delays of two consecutive packets belonging to the same connection [Elhanany et al. 05]. A connection may be a TCP connection, the transmission of a video sequence, etc. Note that jitter is only meaningful for connected traffic and, therefore, it is only measured for *Audio* and *Video* traffic.
  - Average jitter results are not as useful as maximum results. The reason is that jitter is used to dimension reception buffers, and we would want to prepare buffers for the worst case. However, as with latency results, maximum jitter is a very unstable value and, therefore, we use the 99th percentile.

- Dropped packets. In general, since we are using credit-based flow control, packets are not dropped. However, for some applications, if a packet has to wait a long time before being injected, it may be worthless to inject it. For instance, video packets that wait more than 100 ms in the interface should not be injected since they will not be useful for the receiver. In this case, those packets are dropped.
- For latency and jitter, we also show the cumulative distribution function (CDF), which represents the probability of a packet achieving a latency or jitter equal to or lower than a certain value.

In addition to these QoS results, we will show other results that are useful to illustrate the differences between the architectures we are evaluating.

3.4.4. Summary

We will evaluate in the following the proposal that has been discussed in previous sections. For that purpose, we have developed a simulator which accurately emulates the behavior of a high-speed interconnection.

In most of the cases, we will use an indirect network topology, more specifically a folded MIN. However, we will also test direct networks.

In order to evaluate our proposal, we will use the performance of several traditional designs. These architectures will be similar to those proposed in standards like InfiniBand and AS.

We will use synthetic traffic in most of the cases. For that purpose, we will use complex models like Zipf's law destination distribution and self-similar arrival models. Moreover, we will also evaluate the transmission of MPEG-4 video sequences.

3.5. Strict Priorities

In the following sections we will evaluate several switch architectures. All of them have in common the use of strict priorities schedulers at the end-nodes' network interfaces. Moreover, traditional designs also apply strict priorities in their scheduling.

In general, strict priorities can introduce starvation: if the service level with the highest priority takes all the bandwidth, there is no guarantee for the rest of the service levels that they will get any bandwidth. However, this could be easily solved by using a connection admission control (CAC). In this way, we guarantee a minimum bandwidth assignation to all the QoS requiring traffic classes. We also limit the traffic injection of service levels with the highest priority.

In order to emulate strict priority switches, we use the following strategy. Let us suppose that several packets arrive at a switch from a network interface. Taking into account that the interface implements a priority-based arbiter, the first packet should be the one with the highest priority. So, instead of separating the packets among several VCs according to their traffic classes, we put them all in the same queue in the arrival order. Later, when the switch must decide which packets should be transmitted, it will seek in the input queues. In that case, it is only necessary to look at the first packet in each queue, because its position at the front of the queue indicates that it had a higher priority when it left the network interface. The scheduler we propose considers the original priority of the packets at the head of the queues, instead of just whether they are regulated traffic or not. This is not very complex because very efficient priority encoder circuits have been proposed [Huang et al. 02].

Obviously, the network interface can only arbitrate among the packets it holds at a given moment. Therefore, when no more high-priority packets are available, a low-priority QoS packet can be transmitted. If this packet has to wait at a switch input queue, and other packets with higher priority are transmitted from the network interface, they would be stored in the same VC as the low-priority packet, and be placed after it in the queue. Thus, the arbiter would penalize the high-priority packets, because they would have to wait until the low-priority packet is transmitted. But this *order error* situation has a small impact on performance because there is bandwidth reservation for QoS packets. This means that all the QoS packets will flow with short delay.

3.5.1. Initial scenario

We will start this evaluation section with a simple test. We will use synthetic traffic based on a self-similar pattern to compare three alternative switch designs. In the three cases we use 16 port switches, but we have:

- Traditional 2VC: This switch has two VCs, and there are only two categories of traffic in the network, including end-nodes.
- Traditional 8VC: In this case, we have as many VCs as service levels, which are eight in this test.
- New 2VC: This is a switch using our proposal in this chapter, and so it has two VCs, but the end-nodes have eight.

As we have mentioned, we will inject traffic from eight different service levels, each one with increasing priority, since we are testing strict priorities schedulers.

Our objectives in this first test are to confirm that our proposal works in a basic scenario, and to discard the traditional two VCs architectures as valid approach for QoS provision.

At the right, we have the results in Figure 3.12. In the first group of results (from left to right, and top to bottom, service levels 0 to 3), which shows the average latency of the first four service levels (QoS group), we can see that for all the service levels the best performance corresponds to the *Traditional 8VC* case. This is not surprising, since it is the most complex architecture. However, we can see that our *New 2VC* proposal has similar results, even though it only implements two VCs in the switches. This is more remarkable since the *Traditional 2VC* case has much worse performance. It only has better latency results for service level 3 (second row, right), which has the lowest priority of the QoS group.

The next results are the throughput performance of the next four service levels (from left to right and top to bottom, service levels 4 to 7). An interesting result is the throughput yielded by the *Traditional 2VC* architecture. As can be seen, it is the same, no mater of which service level. This is very inappropriate, since a differentiation would be desireable. The reason for this behavior is simple: the network only considers two broad traffic classes. For this reason, traditional two VC architectures are not valid when we need more differentiation.

On the other hand, we have the latency results of the *Traditional 8VC* and *New 2VC* cases. In this case, both are able to provide differentiation in throughput. Note that our *New 2VC* proposal offers slightly better throughput results, we will investigate the reason for this in the following scenarios.

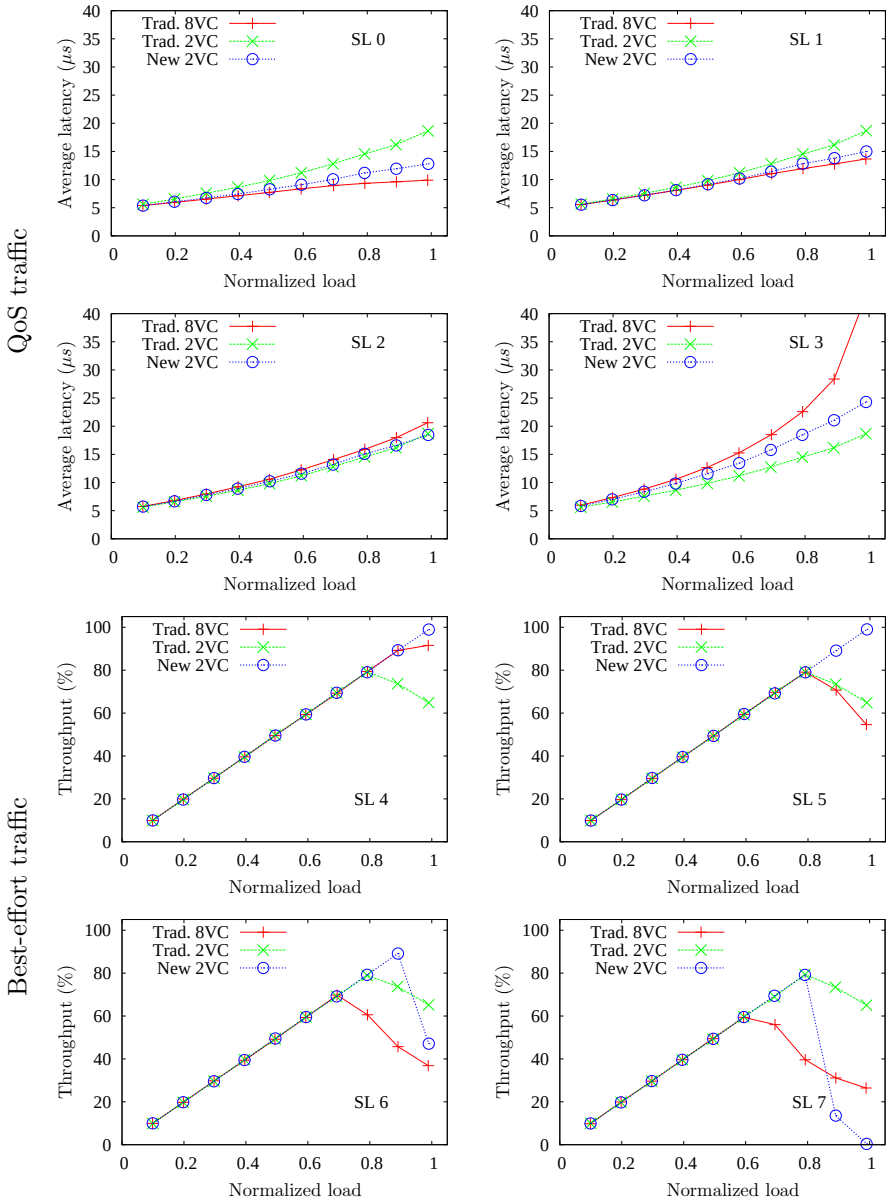


Figure 3.12: Results of initial scenario with strict priorities.

3.5.2. Buffer organization scenario

In Section 3.4.1 we presented the four architectures we are going to use in the rest of this chapter for performance evaluation. These are:

- *New 2 VCs-P*. This architecture has 32 Kbytes per port and 16 ports per switch.
- *New 2 VCs-B*. This architecture has 64 Kbytes per port and 8 ports per switch.
- *Traditional 8 VCs-P*. This architecture has 32 Kbytes per port and 16 ports per switch.
- *Traditional 8 VCs-B*. This architecture has 64 Kbytes per port and 8 ports per switch.

The first two architectures are based on our proposals, while the next two are traditional models as assumed in InfiniBand or PCI AS specification.

After the results we obtained in the previous section, we are not going to show more results of traditional 2 VC architectures. The reason is that they are not able provide QoS to more than two categories and including these results would only result into more difficult to read figures.

In this scenario we evaluate these four cases with the same synthetic traffic we used in the previous section. In addition to observe average latency, we also show maximum latency values.

Figure 3.13 shows the results. With this synthetic load, there are no big differences in performance for the four architectures. Moreover, latency results are in the order of a few μs for the most delay sensitive service levels, which is usually an acceptable results.

Maximum latency results are increased up to 50% when using our proposals. However, this only happens for the traffic class with the highest priority when load is very high. The reason is that our switches are not perfectly imitating strict priority switches and some order errors are introduced.

On the other hand, we have remarkably reduced the number of VCs. Besides, this is not the only advantage of our proposal, as we will see in the following sections.

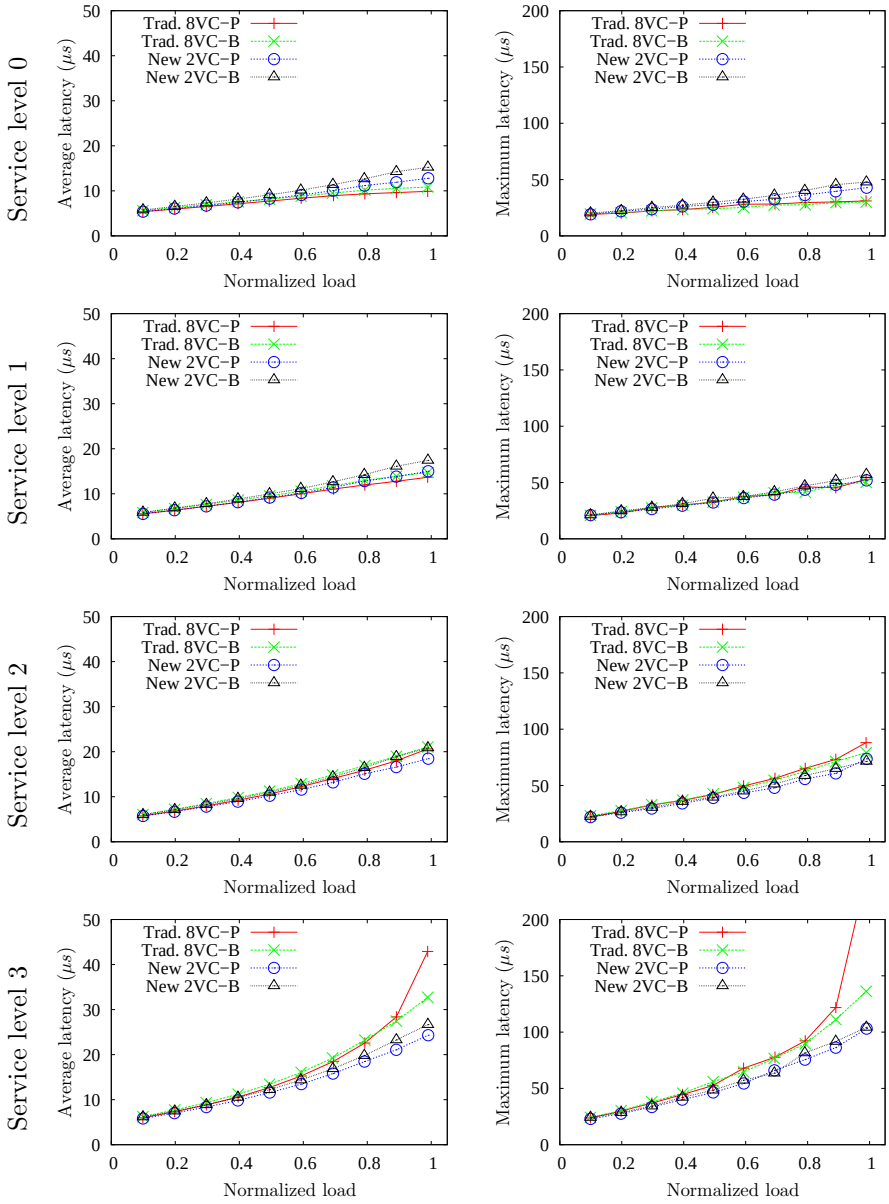


Figure 3.13: Results of buffer organization scenario with strict priorities.

3.5.3. Trace scenario

In the previous tests, we have used synthetic traffic in order to test the performance of the different architectures. In this case, we are going to use traffic traces from a real machine. This traffic comes from I/O system from *Hewlett-Packard Labs*.

Since the traces are a bit old, a compression factor of $\times 40$ is applied in order to obtain a traffic load more appropriate for nowadays technology.

We have distributed trace messages into 8 traffic classes, each one with increasing priority. The first four are assumed "QoS traffic" and the rest are "best-effort traffic".

In Figure 3.14, in the next page, we can see the performance of the different traffic classes for the same four switch architectures we evaluated in the previous section.

Regarding the "QoS traffic" results, we can see that in all the cases, the best performance corresponds to service level 0 (which has the highest priority), while the worst performance corresponds to service level 3. Moreover, general performance is very similar in the four architectures.

If we look at the injection peaks at $150\mu s$, $250\mu s$, and $370\mu s$, we can see that our proposals, *New 2VC-B* and *New 2VC-P*, offer lower latency values than their traditional counterparts. This phenomenon is due to the more flexible buffer assignation, as we explained in Section 3.1.4. We will examine this in more depth in the following tests.

The performance of "best-effort traffic" is similar in all cases. However, again we see better results for our proposals.

In this scenario we have seen that even with realistic traffic our proposals are able to provide traffic differentiation. Moreover, the latency results are better than those obtained with more complex architectures.

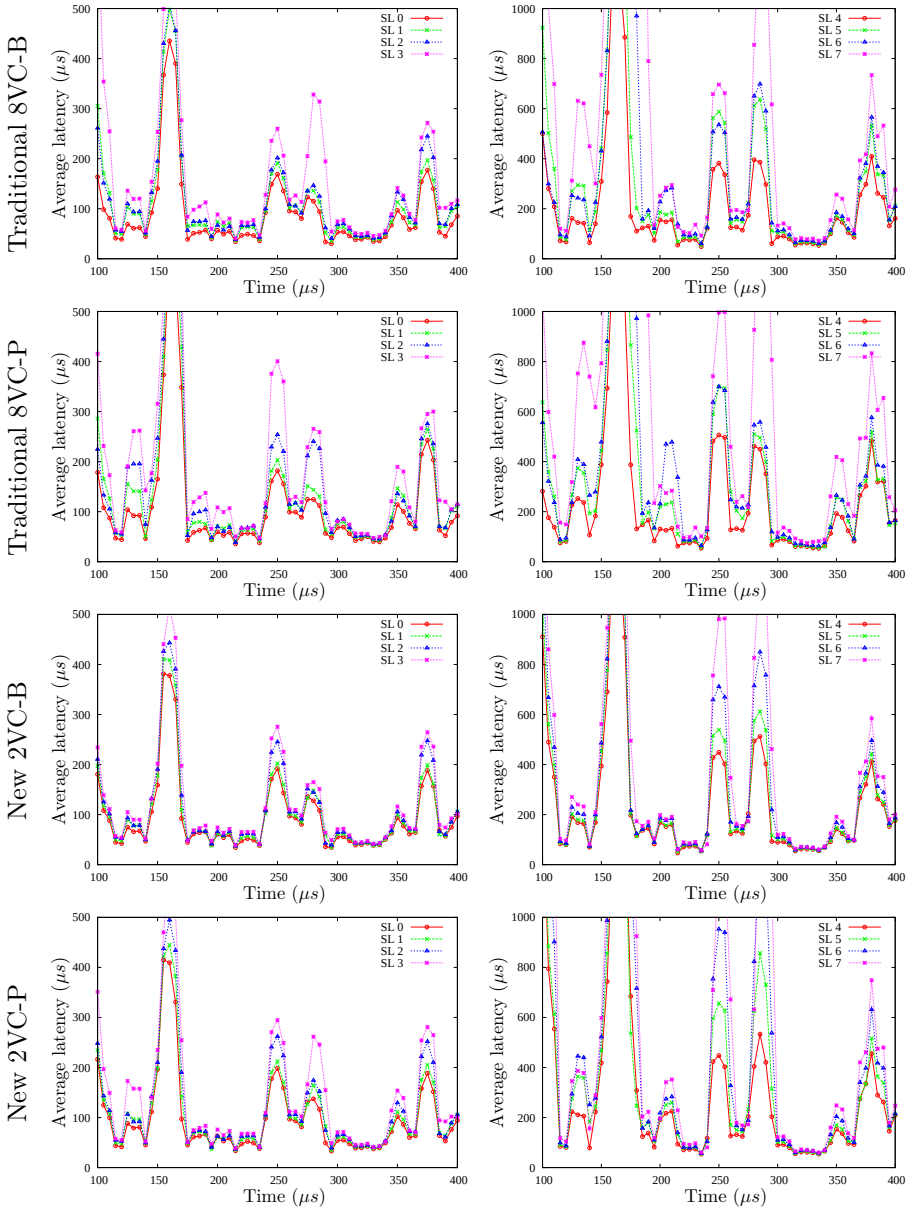


Figure 3.14: Results of trace scenario with strict priorities.

3.5.4. Realistic traffic scenario

In order to evaluate the goodness of the different switch architectures we are considering, we have so far used synthetic traffic and traffic from traces of an storage system. However, in Section 3.4.2 we introduced a more realistic traffic model, which considers several kinds of traffic. These include traces from MPEG-4 video sequences and realistic models of audio, control, and best-effort traffic.

In addition to the different input load, we are also interested in examining latency distribution results. Average latency results may sometimes not be enough to characterize performance. More specifically, we will study the cumulative distribution function (CDF) of latency of packets for an input load of 100%.

Likewise, for multimedia traffic the *jitter* results are also very interesting, because they affect the quality of the signal perceived by final users. In this case, we will also see the CDF of jitter for an input load of 100%.

In Figure 3.15 we can see the performance of the four switch architectures we have used in the last scenarios. Regarding *Network Control* traffic, we can see that our proposals increase average latency. In the CDF distribution of latency we can see that also maximum values are increased.

Regarding *Audio* traffic, average latency results are similar. We also see jitter results, which are also worse when using our proposals with reduced VCs. However, note that results are reasonable in all the cases and very few applications would be affected by a jitter in the order of $\mu seconds$.

The *Video* traffic results are very similar in the four architectures. In this case, our *New 2VC-P* architecture is producing the best results. Similarly, the performance of *Controlled Load* traffic does not differ too much when using the four switch architectures studied.

Of the three traffic class-level schedulers, the strict priorities scheduler is the most difficult to emulate by our proposals. This is because the switches with all the VCs can freely reschedule high-priority packets over low priority ones, while in our case we are bound to the initial sorting produced by end-nodes. However, we see in the figure at the right that this has a very limited impact. Moreover, in the next scenario we will see that our proposals have more advantages over the traditional ones than just the VC reduction.

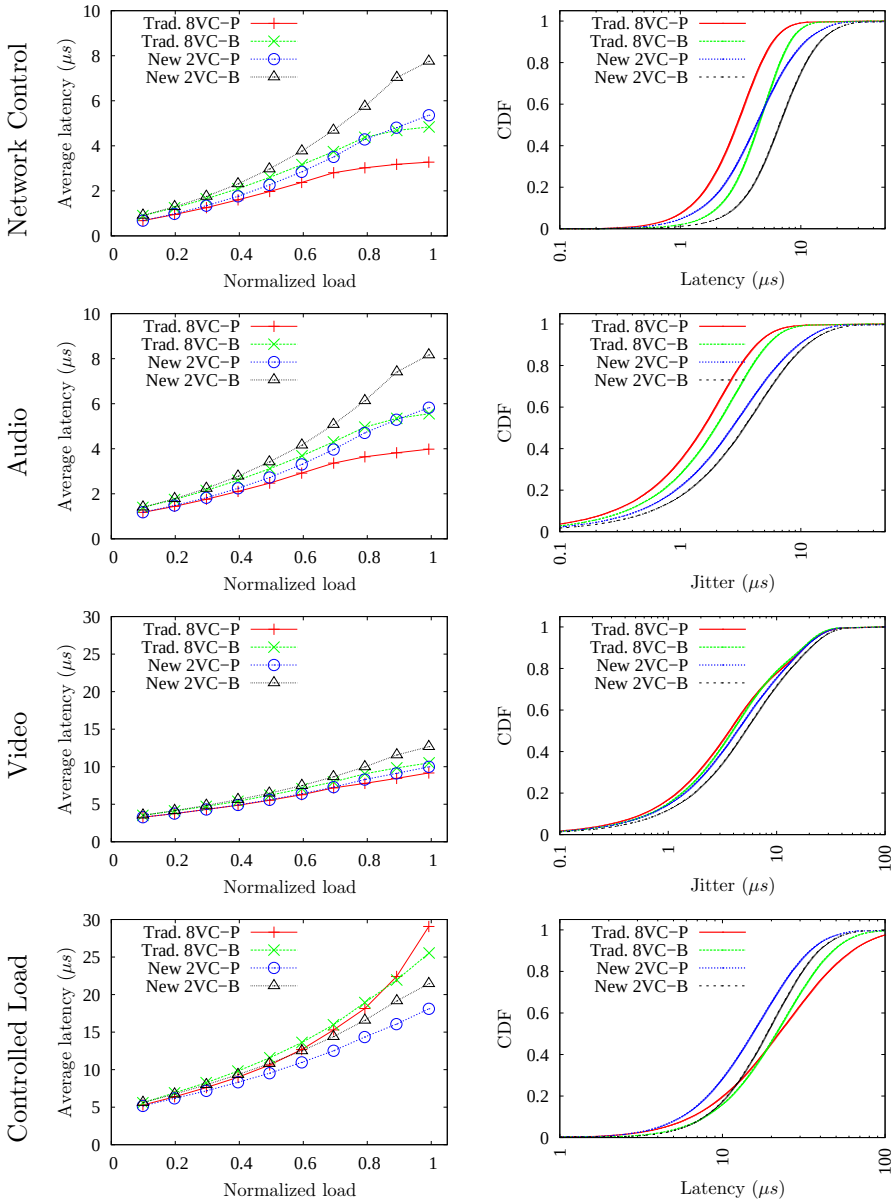


Figure 3.15: Realistic traffic scenario with strict priorities.

3.5.5. QoS traffic acceptance

The objective of this scenario is to vary the amount of QoS traffic injected. We want to see up to which level of QoS traffic proportion the performance is still acceptable. In this section, we do not use the limitation of 70% maximum load of the links when establishing QoS traffic connections. Instead, we vary the proportion of QoS traffic, from 10% to 90% of the total available network load. We fill in the remaining bandwidth with best effort-traffic. Therefore, input links are injecting at 100% of their capacity. We can see that the different traffic classes saturate at different points when using the four architectures. In this way, QoS requirements are satisfied only up to a certain level of QoS traffic load.

The results can be seen in the Figure 3.16, in the next page. We can see at Table 3.4 which is the maximum QoS load at which the different architectures yield acceptable results[4]. For instance, performance of *Audio* traffic with the *New 2VC-B* architecture is only acceptable up to a QoS load of 80%. If more QoS traffic were injected, maximum latency would be to high.

The last row of the table contains the minimum of the column, which means the maximum QoS load where all the QoS requirements can be satisfied. We can see that both architectures using our proposal can accept up to 80% of QoS traffic, whereas the *Traditional 8VC-P* and *Traditional 8VC-B* cases can only accept 70% and 60%, respectively. This is because of the buffer management efficency issue we have discussed before.

Since the scheduling in this case is strict priorities, in all the cases we see that the traffic class which suffers is always the *Controlled Load* traffic class, which has less priority than all the other QoS traffic classes.

Taking into account these results, our proposal is able to cope with more QoS traffic while keeping QoS guarantees. This is due to a more flexible buffer management. We also conclude that the *New 2VC-P* architecture is better than the *New 2VC-B* because performance is similar but the first greatly reduces component count.

Traffic Class	**Tr. 8VC-P**	**Tr. 8VC-B**	**New 2VC-P**	**New 2VC-B**
Network Control	90%	90%	90%	80%
Audio	90%	90%	90%	80%
Video	90%	90%	90%	80%
Controlled Load	60%	70%	80%	80%
All QoS	**60%**	**70%**	**80%**	**80%**

Table 3.4: Maximum QoS load with acceptable performance.

[4] We consider acceptable a 100% throughput and latency values of few μs.

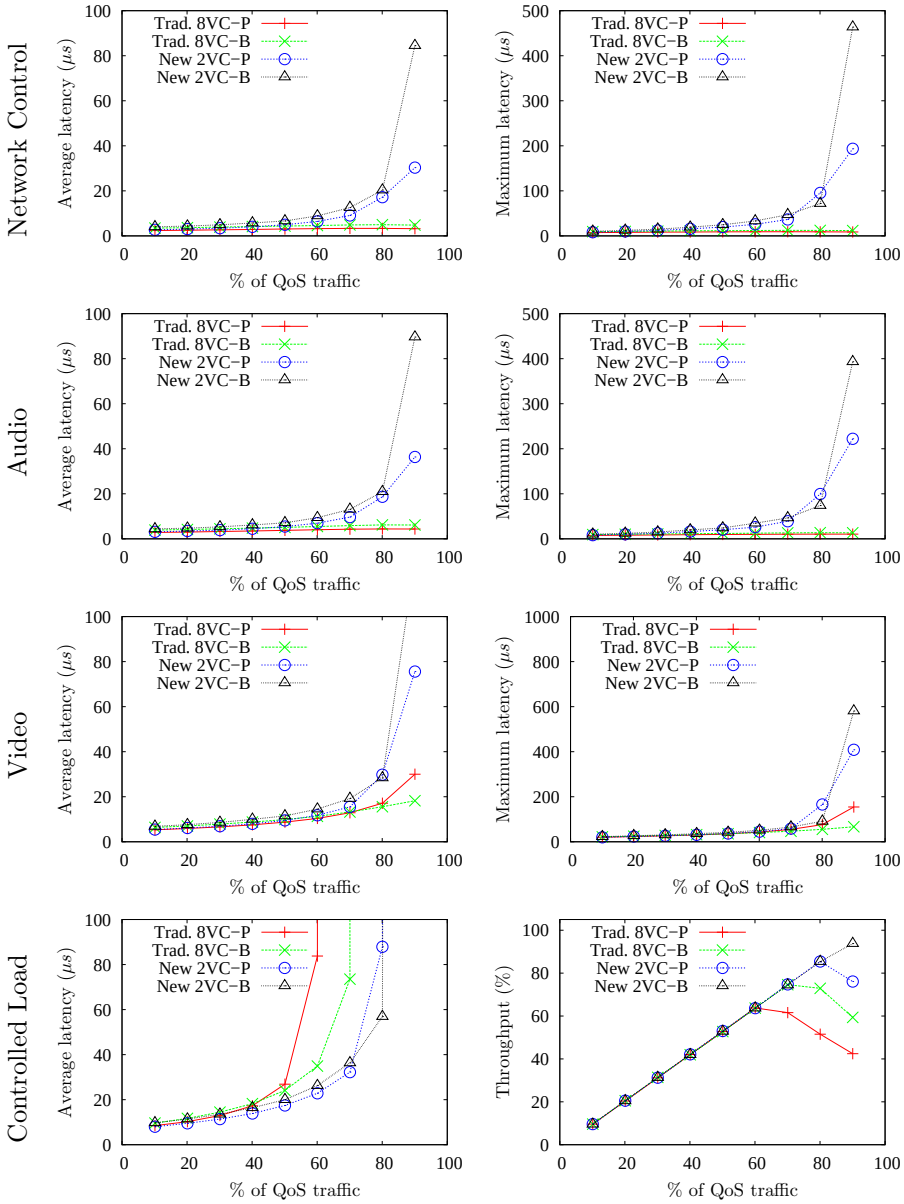


Figure 3.16: QoS traffic acceptance scenario with strict priorities.

3.5.6. Summary

In this section we have evaluated different switch architectures for traffic class-level QoS, while the scheduling was strict priority-based. We have seen that our proposals, despite using just two VCs, succeed in offering similar performance as the more complex traditional architectures, which have eight VCs, one per traffic class.

There are several differences between our proposals and the more complex ones. Firstly, with our switch architectures, latency of the service levels with the highest priorities is increased. This is because the 2 VC switches are vulnerable to order error situations, which cannot happen in the more complex switches.

However, a key advantage of our proposed switches is that buffer space can be assigned with more flexibility. While in the traditional case memory devoted to one VC cannot be used by another, in our case the buffer are splitted into just two parts, instead of eight. In this way, performance is better when there are bursty patterns of traffic.

Summing up, even though strict priorities is not the most convenient scheduler for our new proposals, we have shown that after considering all the trade-offs, our switches are more advantageous than the traditional ones.

3.6. Table-Based Scheduling

In recent standards as InfiniBand and PCI AS, there is a recommendation of using a table-based scheduling. More specifically, they propose to implement a table-based weighted round robin algorithm. The arbitration table is a register array with fixed-size entries. Each table entry, which contains a VC identifier value, corresponds to a slot of a WRR arbitration period. When arbitration is needed, the table is cycled through sequentially and a packet is transmitted from the VC indicated in the current table entry. If the current entry points to an empty VC, that entry is skipped.

How to configure such a table is out of the scope of this thesis. However, the topic has been discussed in [Alfaro et al. 04]. Briefly, in order to provide traffic of a given VC with a minimum bandwidth, the number of table entries assigned to that VC must be proportional to the desired bandwidth. In order to provide maximum delay requirements to a VC, the maximum separation between two consecutive table entries devoted to that VC must be fixed to an appropriate value. This allows us to control the maximum latency to cross each network element and, therefore, the global delay.

In Table 3.5 we show the configuration used in end-nodes and in traditional switches.

In order to emulate table-based WRR in our proposed switches, we proceed as we introduced at Section 3.2.2.1. Since the end-nodes are producing flows of packets with the correct proportions of each service level, and the CAC guarantees that the QoS traffic classes will have enough bandwidth, we can use just two VCs, as we explained earlier.

Since table-based scheduling is a popular alternative for high-performance networks, we will perform some additional tests in this case. More specifically, we will perform a scalability test varying network size from 64 to 512 end-nodes. Moreover, we will also see a scenario with 3-D meshes instead of MINs.

TC	Name	Table entries	Max. separation
0	Network Control	16	4
1	Audio	16	4
2	Video	12	5
3	Controlled Load	10	Unspecified
4	Excellent-effort	4	Unspecified
5	Preferential Best-effort	3	Unspecified
6	Best-effort	2	Unspecified
7	Background	1	Unspecified

Table 3.5: Table scheduler configuration.

3.6.1. Initial traffic scenario

We will start the evaluation of our proposals with table-based WRR schedulers with a simple scenario. We will use synthetic self-similar traffic and the four switch architectures we discussed in the previous section. These are:

- *New 2 VCs-P*. This architecture has 32 Kbytes per port and 16 ports per switch.
- *New 2 VCs-B*. This architecture has 64 Kbytes per port and 8 ports per switch.
- *Traditional 8 VCs-P*. This architecture has 32 Kbytes per port and 16 ports per switch.
- *Traditional 8 VCs-B*. This architecture has 64 Kbytes per port and 8 ports per switch.

The results can be found at the right, in Figure 3.17. Just like with strict priorities schedulers, results are also acceptable in all the cases, with latencies in the order of μs. However, in this case, our proposal achieves better performance both in average and maximum latency.

The reasons for this are two. In the first place, we have shown in Section 3.2 how to effectively imitate bandwidth based scheduling. This is achieved with greater success than with strict priority scheduling. Moreover, as we mentioned in the previous section, our switches handle more efficiently buffer space. In this case, this characteristic gives them advantage over the traditional ones. Therefore, we have this slight reduction in delay.

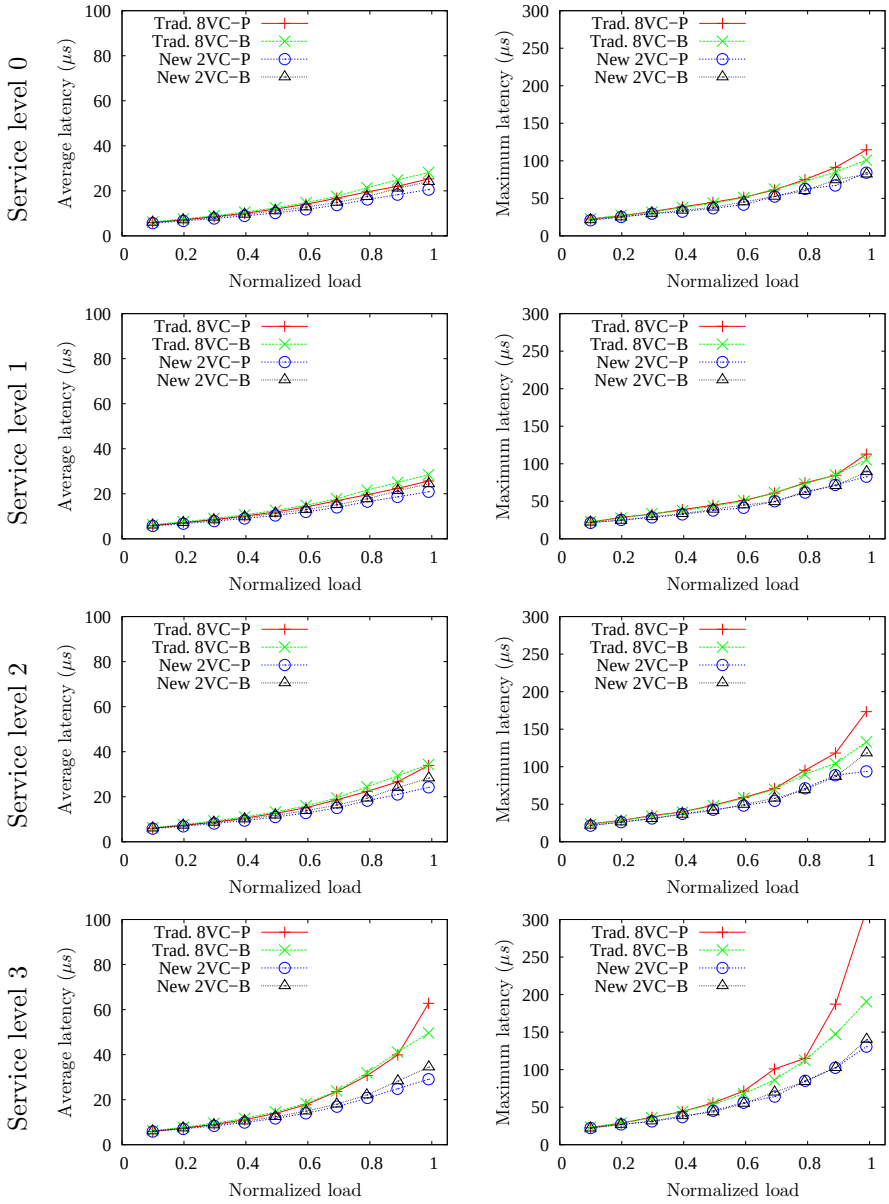


Figure 3.17: Results of buffer organization scenario with table-based WRR.

3.6.2. Trace scenario

Just like with the strict priorities scheduler, we will evaluate the table-based WRR switches with traffic from traces of a real machine. In this way, we have the characteristics of real traffic instead of synthetic Poisson arrivals. We will also apply a compression factor of $\times 40$ in order to obtain a traffic load more appropriate for nowadays technology.

The messages in the trace are evenly distributed into 8 traffic classes, each one with different weights in the table:

SL	# entries	Max. sep.
0	16	4
1	16	4
2	12	5
3	10	Unspec.

SL	# entries	Max. sep.
4	4	Unspec.
5	3	Unspec.
6	2	Unspec.
7	1	Unspec.

Table 3.6: Table arbiters configuration.

In Figure 3.18, in the opposite page, we can see the performance of the different traffic classes for the same four switch architectures we have evaluated in the previous section.

Regarding the first four service levels, the performance is almost the same in all the case, no matter which switch architecture is used. Note that all of them have enough table entries to cope with injected bandwidth.

The points where injection reaches a peak are at $150\mu s$, $250\mu s$, and $370\mu s$. There we can see that our proposals, *New 2VC-B* and *New 2VC-P*, offer lower latency values than their traditional counterparts, once again due to better buffer management.

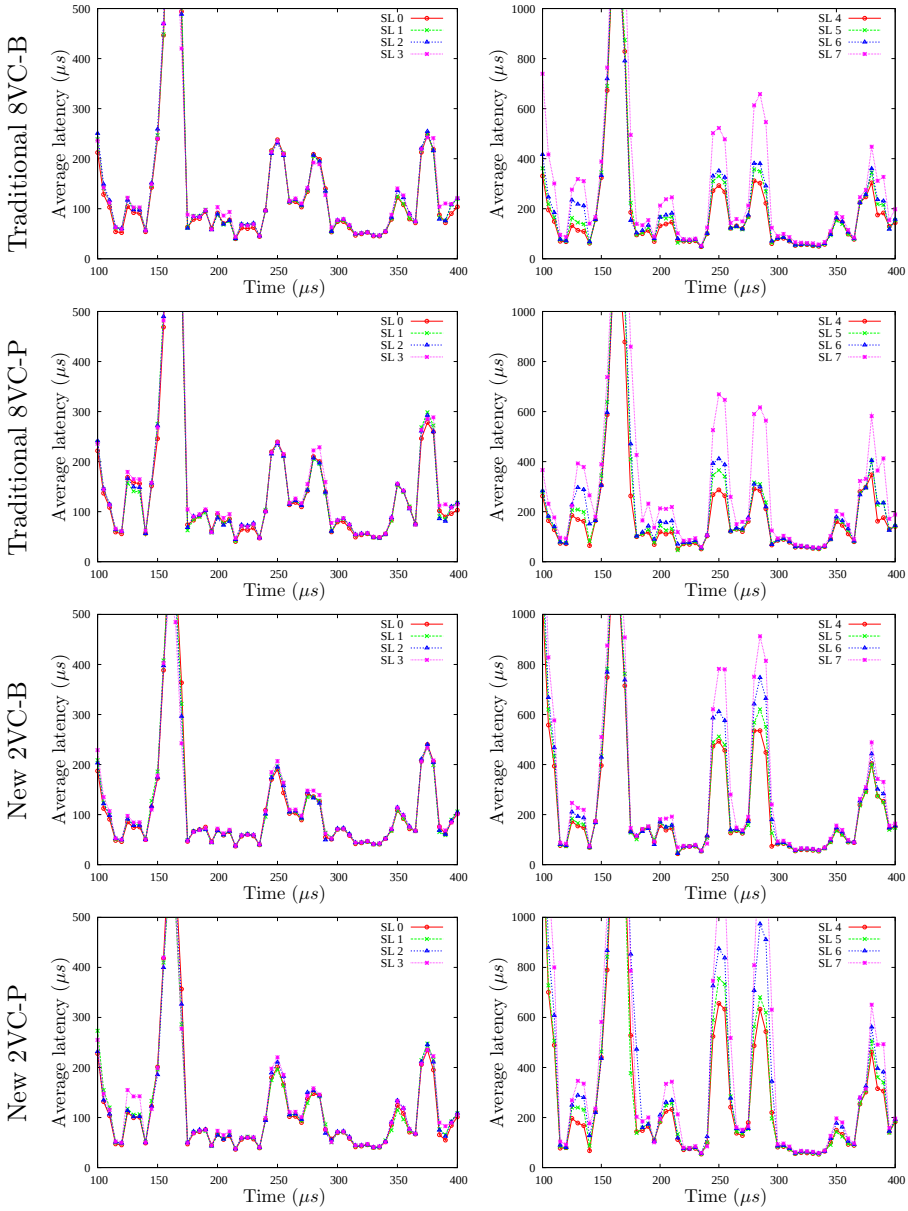


Figure 3.18: Results of trace scenario with table-based WRR.

3.6.3. Realistic traffic scenario

In this scenario we use the more advanced or realistic traffic load. The configuration of the table schedulers in this case is as was described at the beginning of the section.

We will have four service levels of QoS traffic: *Network Control*, *Audio*, *Video*, and *Controlled Load*. Moreover, we have four levels of best-effort service levels. However, we will focus only on performance of the QoS service levels.

In this scenario we study the cumulative distribution function (CDF) of latency and jitter for an input load of 100%. In this way, in addition to average values, we can see all the spectrum of performance of packets.

The Figure 3.19 shows the performance of the QoS service levels. Regarding *Network Control* traffic, we can see that our proposed switch models obtain almost the same average latency as their traditional counterparts (with the same number of ports per switch). There is a slight increase of average latency that is explained in the CDF plot. We can see that most of the packets obtain the same latency with 2 and 8 VCs, but some of them have larger latency results. That means that maximum latency results are increased when using our architectures.

These differences are smaller when we consider *Audio* traffic and they are almost nonexistent with *Video* traffic. Finally, regarding *Controlled Load* traffic, performance is slightly better when using our proposals.

As we introduced at the beginning of the chapter, bandwidth-based algorithms can be effectively emulated with just two VCs. This has been confirmed experimentally. Moreover, just like with strict priorities schedulers, we will see in the following the advantages of the flexibility in buffer assignation.

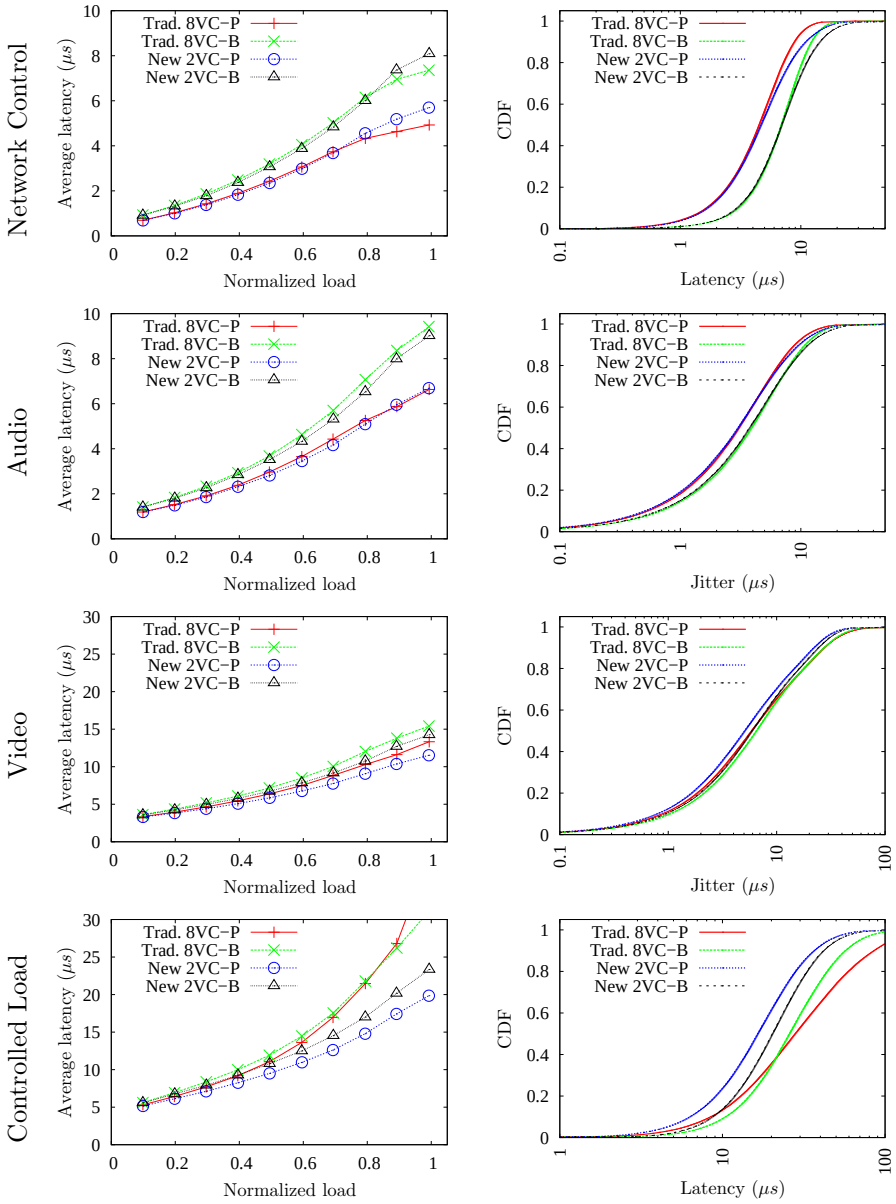


Figure 3.19: Realistic traffic scenario with table-based WRR.

3.6.4. QoS traffic acceptance

In the following experiments, we vary the proportion of QoS traffic, from 10% to 90% of the total available network load. We fill in the remaining bandwidth with best effort-traffic. Therefore, input links are injecting at 100% of their capacity.

In this way, just like we did for the strict priorities architectures, we will see up to which level of QoS traffic proportion the performance is still acceptable. Since in this case there is table-based WRR instead of strict priorities, we will see that the different traffic classes saturate at different points, depending on their bandwidth assignation and on the traffic characteristics. The results can be see in the Figure 3.20, in the next page. The summary of performance is in Table 3.7.

As can be seen in the last row of the table, which shows the maximum QoS load where all the QoS requirements can be satisfied, we can see that both architectures using our proposal can accept up to 80% of QoS traffic. However, the *Traditional 8VC-P* and *Traditional 8VC-B* cases can only accept 50% and 60%, respectively.

The reason why performance is worse with table-based WRR traditional switches compared with strict priorities is that in the WRR case there is a minimum bandwidth guaranteed for best-effort traffic classes, while in the other case QoS service levels were allowed to take as much bandwidth as they needed.

Once again, our proposals show that they are very advantageous over traditional ones. Not only we reduce the number of VCs, but we also improve performance.

Traffic Class	**Tr. 8VC-P**	**Tr. 8VC-B**	**New 2VC-P**	**New 2VC-B**
Network Control	90%	90%	90%	80%
Audio	90%	90%	80%	80%
Video	60%	70%	80%	80%
Controlled Load	50%	60%	80%	80%
All QoS	**50%**	**60%**	**80%**	**80%**

Table 3.7: Maximum QoS load with acceptable performance.

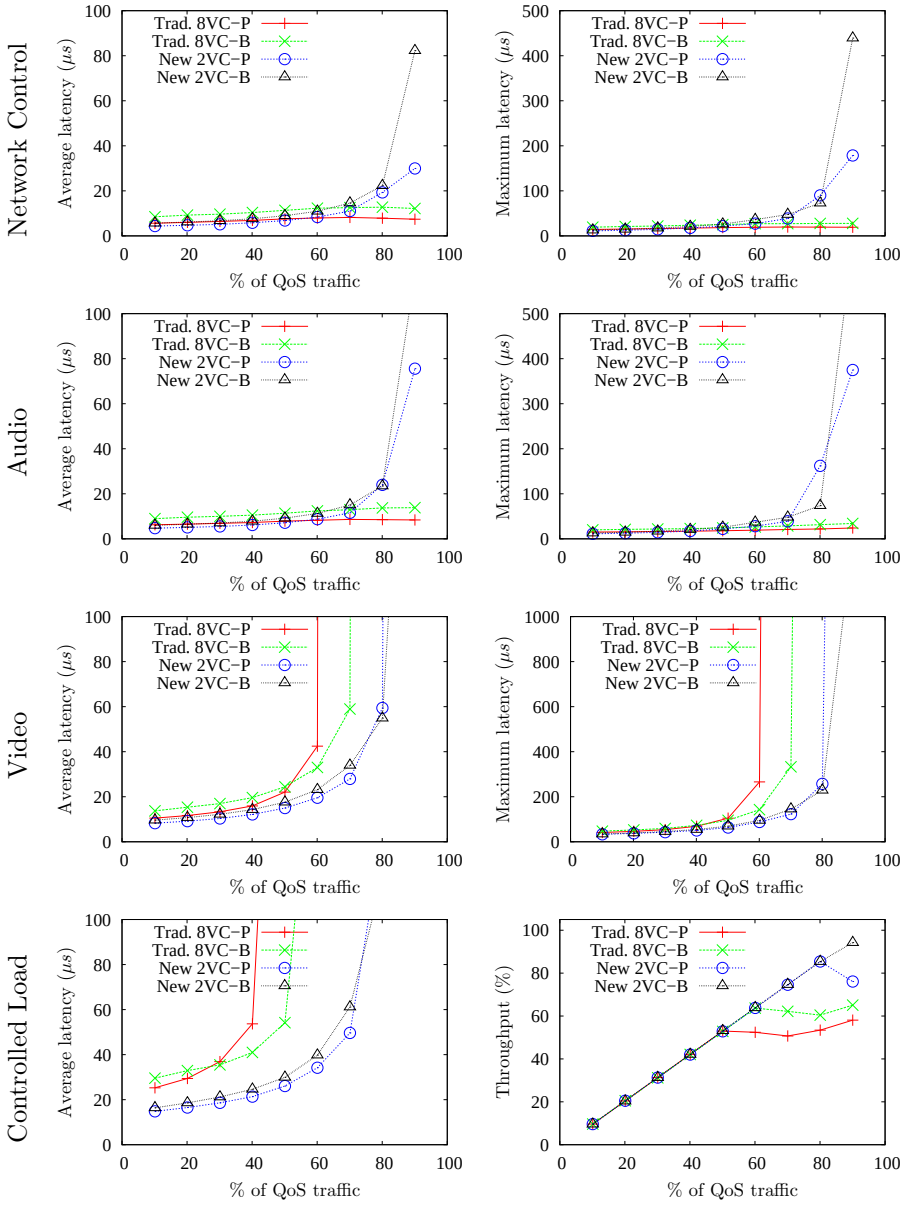


Figure 3.20: QoS traffic acceptance scenario with table-based WRR.

3.6.5. Realistic traffic with mesh scenario

We have repeated the realistic traffic scenario (Section 3.6.3), but n this case the network topology is a 64 end-nodes three dimensional mesh (Figure 3.21). The routing algorithm is the well known dimension-order routing, which is deadlock-free [Dally and Towles 03].

There are only two competing architectures: *Traditional 8VC* and *New 2VC*. In this case, all switches have seven ports (except switches at the edges of the mesh), and total buffer memory is the same in the two alternative designs.

The Figure 3.22 shows the performance of the QoS service levels. In general, latency results are larger than those obtained with the MIN topology. The reason is that, on average, a packet has to cross more switches in the mesh topology than in the MIN topology. Besides, dimension-order routing does not provides the best load balancing possible in a mesh topology. However, it is the most widely used routing algorithm used in this type of networks.

Regarding the performance of the two alternative designs we are evaluating, we can see that our proposal offers the best latency results. As we have discussed before, the reason is in the buffers.

On the other hand, we repeated this test with strict priority and WFQ schedulers and results were very similar both for *Traditional 8VC* and *New 2VC* cases. We are not going to show these results in the thesis in order to show only the most interesting plots.

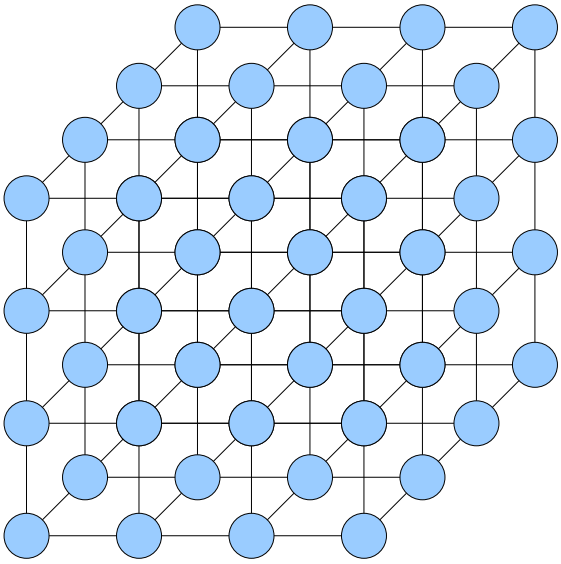


Figure 3.21: 64 end-nodes three dimensional mesh.

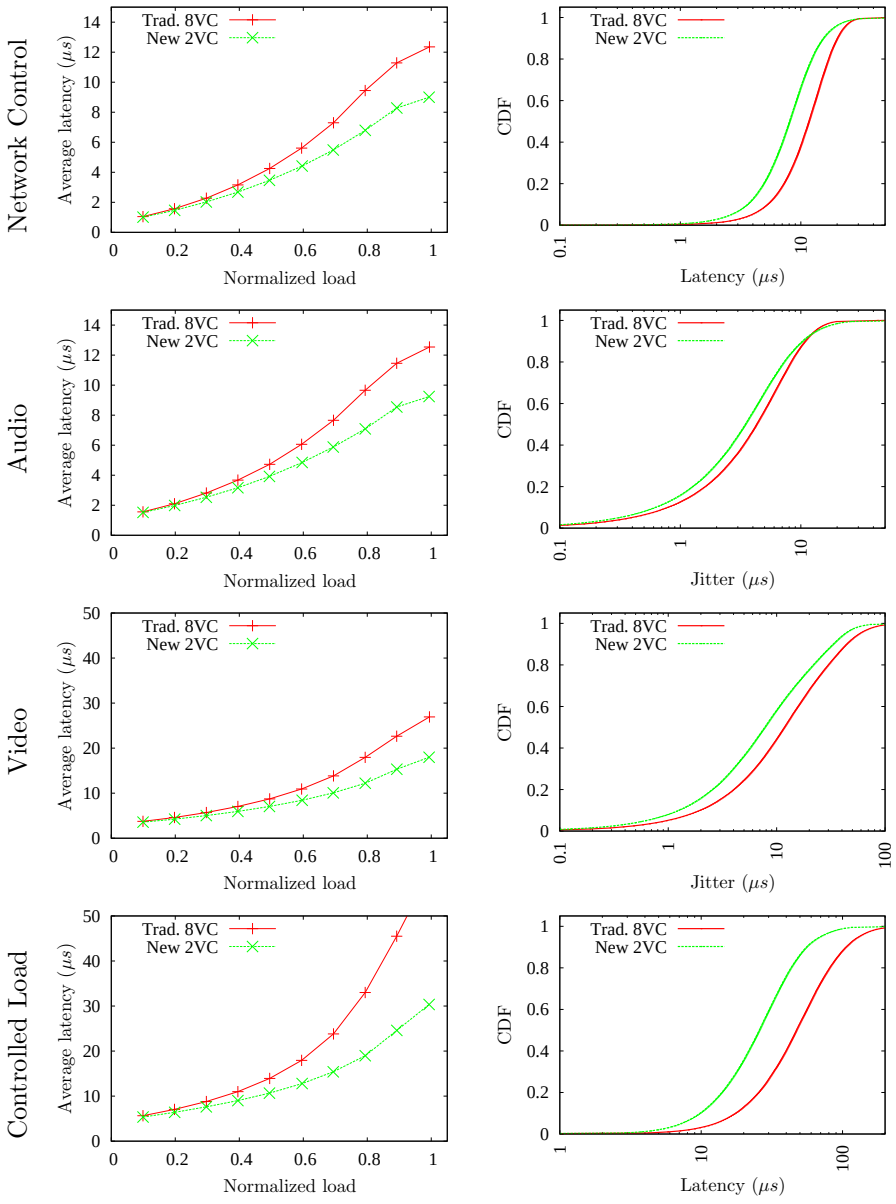


Figure 3.22: Realistic traffic scenario with table-based WRR and mesh topology.

3.6.6. Scalability scenario

In this scenario we perform a scalability study. We observe the results of varying network size from 64 to 512 end-nodes. We will use synthetic Poisson traffic. Conclusions with other types of input traffic are similar and we have not included them for clarity reasons.

In Figure 3.24, we show the scalability of the four cases studied attending to different parameters. Global throughput is very similar in the four cases, although our *New 2VC-B* architecture offers better results due to buffer design. Regarding latency of service level 0 traffic, the one with the highest bandwidth assignation, it scales well both in terms of average and maximum latency for the four architectures.

Figure 3.23 summarizes the trade-offs of using 16 port proposals against 8 port proposals. As can be seen, there is a very noticeable reduction in switch and link counts and, therefore, in the associated power consumption of the interconnection network. This is because 16 port architectures lead to more compact topologies.

Note that our *New 2VC-P* proposal offers the best results in terms of latency, component count, and power consumption. Also, this architecture is able to handle more QoS traffic than the others, as we saw in the previous scenario. For these reasons, we believe that it is the most advantageous of all the architectures we have studied.

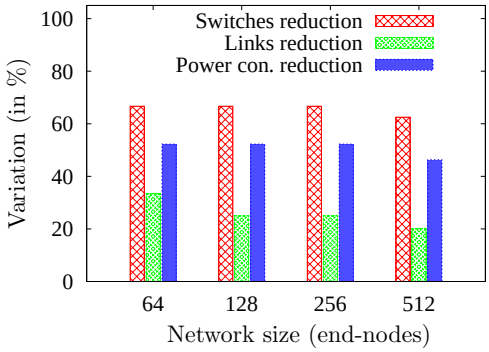
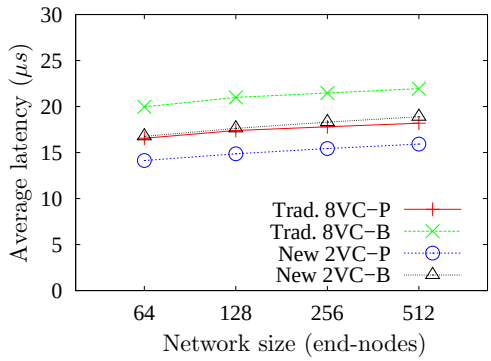
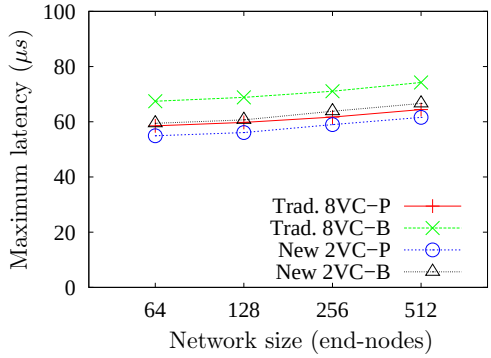


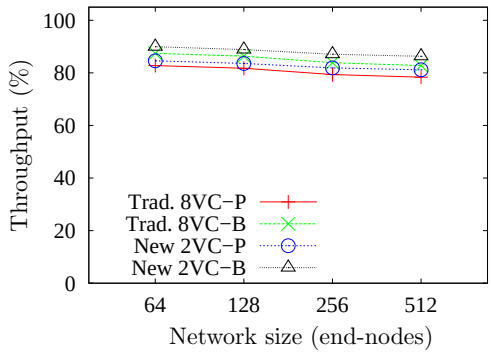
Figure 3.23: Advantages of 16 ports proposals, compared with 8 ports proposals.



(a) Average latency of service level 0



(b) Maximum latency of service level 0



(c) Global throughput

Figure 3.24: Results for scalability scenario.

3.6.7. Summary

We have tested table-based WRR switches in this section. The tests were similar to those performed for strict priorities, but the results were different. In this case, we have shown that our proposals are very good at doing bandwidth-based scheduling and, therefore achieved similar results to the performance of traditional switches.

However, we have also shown that once again, our switches achieve better performance under heavy QoS load. This is because the management of buffers, which is more flexible in our case. This added to the VC reduction, makes our proposal a very attractive alternative for switch design.

3.7. Weighted Fair Queuing Scheduling

In order to offer a complete performance evaluation, we have considered a third option for the schedulers, the Weighted Fair Queuing (WFQ) algorithm. There are several reasons for choosing it. In the first place, it is regarded a good algorithm for QoS provision that can offer both latency and bandwidth results.

Secondly, the PCI AS specification defines three valid schedulers for the switches, and one of them, the minimum bandwidth egress scheduler, can be implemented with the WFQ algorithm. For this reason, we believe that it is interesting to compare its performance with that of our proposals.

In Table 3.8 we have the configuration used by schedulers in traditional switches and in the end-nodes. The bandwidth assignation is the same that was used in the previous section.

TC	Name	WFQ Weight
0	Network Control	0.2500
1	Audio	0.2500
2	Video	0.1875
3	Controlled Load	0.15625
4	Excellent-effort	0.0625
5	Preferential Best-effort	0.046875
6	Best-effort	0.03125
7	Background	0.015625

Table 3.8: WFQ scheduler configuration.

3.7.1. Initial traffic scenario

We will start the last part of the evaluation section with the same scenario we used in the previous two. We will use synthetic self-similar traffic for the evaluation of the following four architectures:

- *New 2 VCs-P*. This architecture has 32 Kbytes per port and 16 ports per switch.
- *New 2 VCs-B*. This architecture has 64 Kbytes per port and 8 ports per switch.
- *Traditional 8 VCs-P*. This architecture has 32 Kbytes per port and 16 ports per switch.
- *Traditional 8 VCs-B*. This architecture has 64 Kbytes per port and 8 ports per switch.

The results are shown in Figure 3.25, in the next page. We can see that in this case the traditional architectures offer better performance in terms of latency for the service levels 0 and 1. The reason is that these service levels have large weights assigned in the WFQ scheduler and these packets are immediately forwarded in the switches.

On the other hand switches using our proposals cannot do this, similarly to what happened with the strict priorities scheduler. However, note that the WFQ algorithm is not trivial to implement. In a real system, the delays and additional silicon area introduced by the increased complexity could make the advantages of WFQ worthless.

Finally, the results with our proposal are still valid for high-performance interconnects, since they are similar to those obtained with more traditional switches, using table-based WRR scheduling.

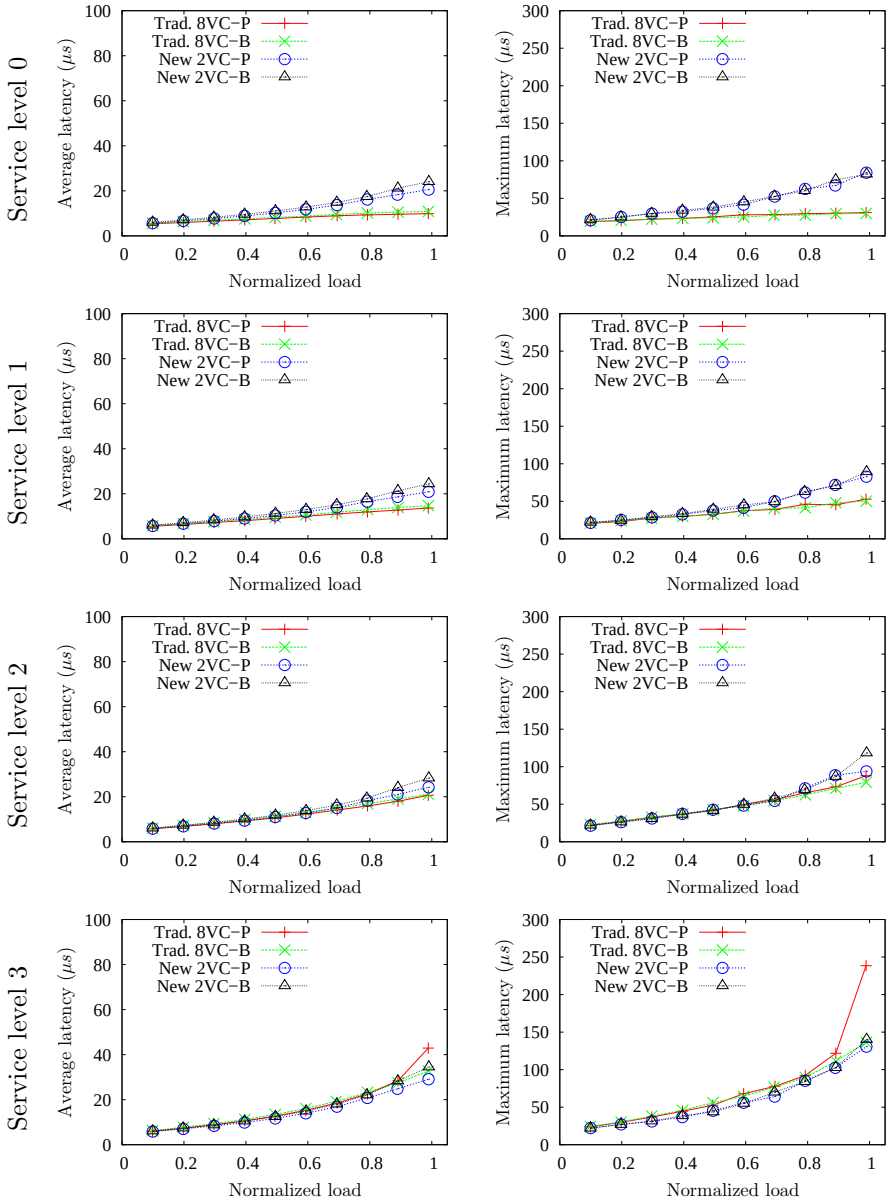


Figure 3.25: Results of buffer organization scenario with WFQ scheduling.

3.7.2. Trace scenario

In this section, we repeat the trace traffic scenario, but this time using the WFQ schedulers. The trace used are also compressed with a $\times 40$ factor.

The messages in the trace are evenly distributed into 8 traffic classes, each one with different weights in the schedulers:

Service level	Weight	Service level	Weight
0	0.2500	4	0.0625
1	0.2500	5	0.046875
2	0.1875	6	0.03125
3	0.15625	7	0.015625

Table 3.9: Weights assignation.

In Figure 3.26, in the opposite page, we can see the performance of the different traffic classes for the same four switch architectures we evaluated in the previous section. Results for the first four service levels are very similar regardless of which switch architecture is used.

The points where injection reaches a peak are at $150\mu s$, $250\mu s$, and $370\mu s$. There we can see that our proposals, *New 2VC-B* and *New 2VC-P*, offer lower latency values than their traditional counterparts, once again due to better buffer management.

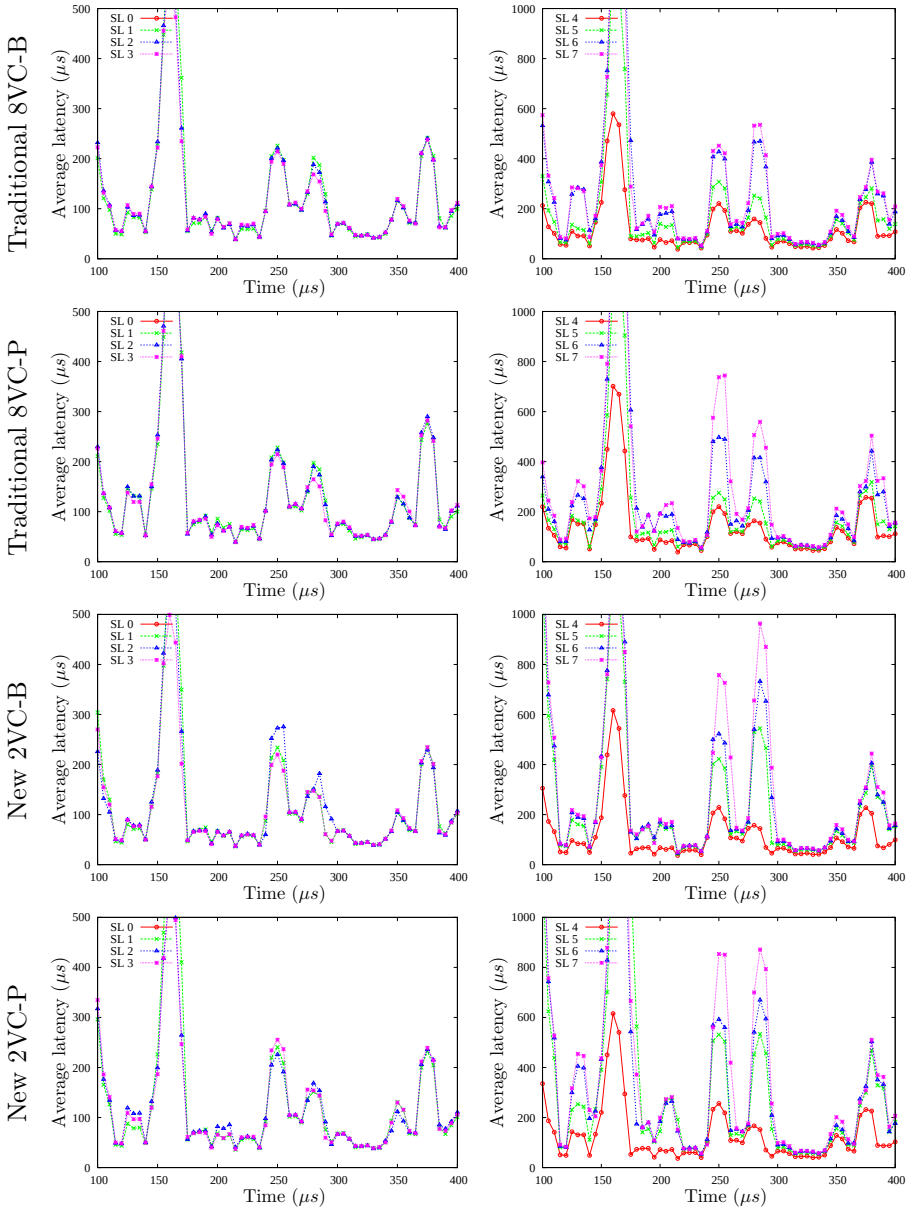


Figure 3.26: Results of trace scenario with WFQ.

3.7.3. Realistic traffic scenario

The use of synthetic traffic did not offer a good approximation to real system performance. On the other hand, the trace-based traffic we used does not contain traffic from multimedia applications. For these reasons, once again we evaluate the system using a more realistic load.

There are eight service levels, four service levels of QoS traffic: *Network Control*, *Audio*, *Video*, and *Controlled Load*; and also, we have four levels of best-effort service levels. However, we will focus only on performance of the QoS service levels.

In this scenario we also study the cumulative distribution function (CDF) of latency and jitter for an input load of 100%. In this way, in addition to average values, we can see all the spectrum of performance of packets.

The performance plots can be be seen in Figure 3.27. We can see that traditional switches improve their performance of *Network Control* and *Audio* traffic, when compared with the results with table-based scheduling. See the results at Figure 3.19 in page 101.

The reason for this improvement is that in WFQ, *Network Control* and *Audio* packets can be immediately forwarded, while in the case of table-based scheduling, they had to wait for the table to be cycled through until an entry of the appropriate service level appeared.

Regarding our proposals, since they cannot forward these packets immediately, they increase average latency by 50% when load is very high. Moreover, maximum latency results are also worse. However, the results are acceptable in all the cases, since they stay in the order of μs.

Regarding the rest of the traffic classes, differences between the performance of the switch architectures studied are very little.

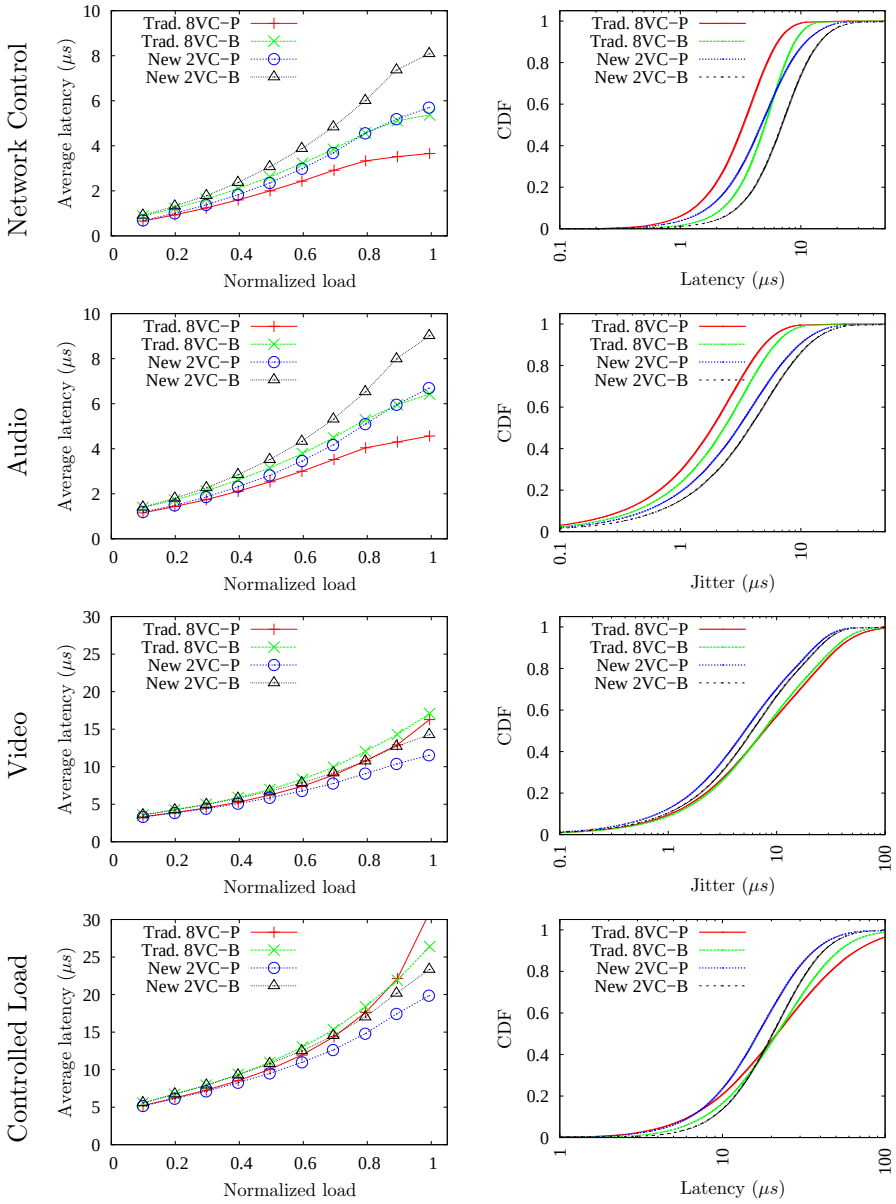


Figure 3.27: Realistic traffic scenario with WFQ scheduling.

3.7.4. QoS traffic acceptance

In this scenario we want to confirm, with the third scheduler we are using, if our proposals still offer better performance with a large proportion of QoS packets. For this purpose, we vary the proportion of QoS traffic, from 10% to 90% of the total available network load. We fill in the remaining bandwidth with best-effort traffic. Therefore, input links are injecting at 100% of their capacity.

Once again, we will see that the different traffic classes saturate at different points when using the four architectures. In this way, QoS requirements are satisfied only up to a certain level of QoS traffic load.

Results are shown in Figure 3.28. Results are similar to those obtained with table-based schedulers. However, the traditional switches offer slightly better performance, specially when looking at *Controlled Load* traffic. This is due to the advantages of WFQ scheduling over table-based WRR.

There is a summary of results in Table 3.10. Although the traditional switches have better performance than in the previous section, our proposed switches still offer better results. This has shown, with another scheduler at switches and end-nodes, that our proposal is superior due to the flexibility in buffer management.

Traffic Class	**Tr. 8VC-P**	**Tr. 8VC-B**	**New 2VC-P**	**New 2VC-B**
Network Control	90%	90%	90%	80%
Audio	90%	90%	80%	80%
Video	60%	70%	80%	80%
Controlled Load	60%	70%	80%	80%
All QoS	**60%**	**70%**	**80%**	**80%**

Table 3.10: Maximum QoS load with acceptable performance.

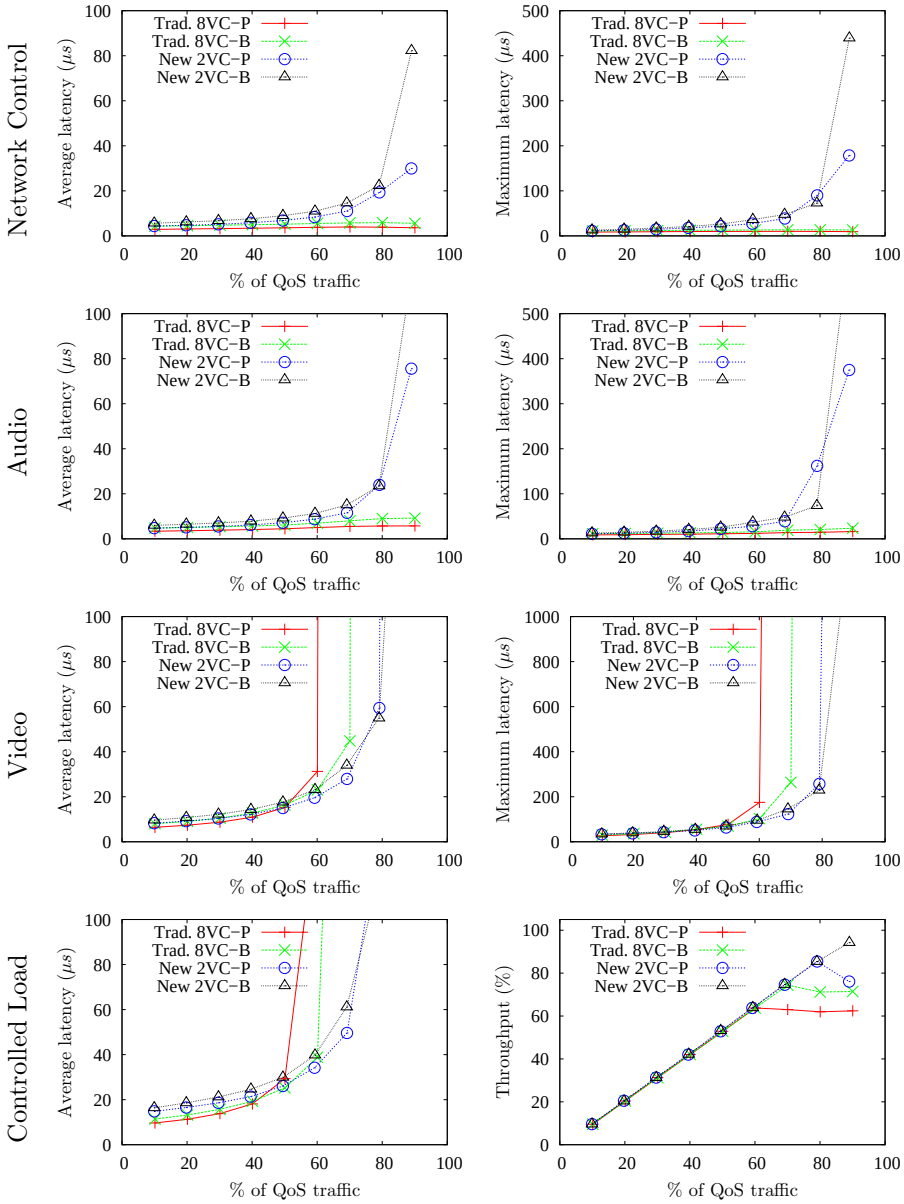


Figure 3.28: QoS traffic acceptance scenario with WFQ scheduling.

3.7.5. Summary

With the last of the three schedulers considered, we conclude our performance evaluation. When using the WFQ schedulers we have confirmed the main results that were obtained in the previous sections. These conclusions are the following:

- Our proposed switches, even though they only have two VCs, can still offer traffic differentiation. This is almost as good as the one obtained with table-based schedulers, and not far from the performance obtained with strict priorities and WFQ schedulers.
- The average and maximum latency of service levels with a high priority or bandwidth assignation is increased when using our proposal. This is because more complex architectures can forward immediately this kind of packets, while in our case they have to wait in the same shared buffer as other service levels.
- When there is a lot of bursty traffic, for instance when the proportion of video traffic is high, our proposal is clearly superior to traditional designs. The reason is that in a traditional design the buffer is splitted in many small parts (VCs), which can only be used for their corresponding service levels. In our case, many service levels share a larger memory area, which allows for a better utilization of buffers. This leads to the better performance we have seen in the previous sections.

The most representative scheduler of current high-performance interconnect technologies is the table-based WRR. This is proposed in recent interconnect standards as InfiniBand and PCI AS, for instance. Against this kind of switches, we believe that our proposal is clearly superior.

When compared with other proposals, like strict priorities and WFQ, there is the trade-off of the increased latency. However, we still think that our proposal is more advantageous because of the reduced complexity and increased throughput performance.

3.8. Conclusions

Traffic class-level QoS support is a popular proposal for high-performance interconnects. It is usually implemented by means of a VC for each TC. In this chapter we have shown that this is an inefficient solution.

In this chapter we have also presented our proposal consisting in making the network elements cooperate, building together ordered flows of packets. Consequently, the switches try to respect the order in which packets arrive at the switch ports, which is probably correct. This allows a drastic reduction in the number of VCs required for QoS purposes at each switch port.

This study has shown that it is possible to achieve a more than acceptable QoS performance with only two VCs. We reuse at the switches some of the scheduling decisions made at the network interfaces. This opens up the possibility of using the remaining VCs for other concerns, like adaptive routing or fault tolerance. Furthermore, it is also possible to reduce the number of VCs supported at the switches, thereby simplifying the design, or increasing the number of ports.

We have proposed a switch design which benefits from that proposal. We examine its feasibility as a single-chip switch and the hardware constraints that it would have. We also have compared, through simulation, the performance of this design with that of more traditional architectures. We have found that we can provide performance very similar to a more complex architecture, but reducing the component count of the network.

CHAPTER 4

Efficient Flow-Level QoS Support

IN the previous chapters, we have seen that QoS is becoming an attractive feature for high-performance networks and parallel machines. The reason is that in those environments there are different traffic types, each one having its own requirements which must be satisfied.

In Chapter 3, we have seen how to provide QoS at the traffic-class level in an efficient way. However, there have been proposals for a different kind of QoS support, which are the flow-level QoS algorithms.

Deadline-based algorithms, which are a sub-set of flow-level algorithms, can provide powerful QoS provision. However, the cost associated with keeping ordered lists of packets makes these algorithms impractical for high-performance networks.

In this chapter, we explore how to adapt efficiently the Earliest Deadline First family of algorithms to the high-speed network environments. The results show an excellent performance using just two virtual channels, FIFO queues, and a cost feasible with today's technology.

4.1. Flow-Level QoS

We have motivated that the QoS provision is an important feature of high-performance interconnects. It allows to support a great variety of applications

with a single network, without harmful influences between the different types of traffic. Moreover, it allows to provide performance guarantees, which are demanded by some applications, like multimedia applications.

The two main types of QoS support are per-traffic-class and per-flow support. The first approach requires the classification of the traffic in traffic classes and the assignment of one VC per traffic class. The network switches offer a traffic differentiation based on these traffic classes by applying different scheduling algorithms at the VC level. In the previous chapter we have seen how to deal efficiently with this QoS provision, by using just two VCs.

On the other hand, there is also per-flow QoS support, which offers a fine-grained provision of QoS. It usually requires a flow identifier to be associated with each packet and per-flow information to be kept at each switch of the network. This second approach is much more powerful, but it is also so complex that it has never been implemented in a high-performance environment, perhaps with the exception of ATM [ATM 95].

In this chapter, we discuss how to obtain most of the benefits of the per-flow QoS approach within the constraints of high-performance switches. More specifically, we will propose a novel strategy to emulate the Earliest Deadline First (EDF) family of algorithms by using just a pair of FIFO queues.

4.1.1. Weighted fair queuing

The Generalized Processor Sharing (GPS) is an output port arbitration algorithm that offers excellent properties regarding QoS provision. GPS is a general form of the head-of-line Processor Sharing (PS) service discipline [Kleinrock 76]. With PS, there is a separate FIFO queue for each session sharing the same link. During any time interval, when there are exactly N non-empty queues the server services the N packets at the head of the queues simultaneously, each at a rate of $1/N$ of the link speed. While a PS server services all non-empty queues at the same rate, GPS allows different sessions to have different service shares (weights) and services the non-empty queues in proportion to the weights of their corresponding sessions (Figure 4.1).

PS is an ideal discipline where the server can service N sessions simultaneously. In real systems, the server has to transmit one packet at a time. Demers et al. [Demers et al. 90] proposed a packet approximation algorithm of PS called Fair Queuing. They show that Fair Queuing provides fair allocation of bandwidth and offers protection from misbehaving sources.

Parekh and Gallager [Parekh and Gallager 93] demonstrated that, by employing GPS servers at switches, end-to-end delay bound can be guaranteed to a session provided its traffic is constrained at the source by a leaky bucket. They also proposed a packet approximation algorithm for GPS which they called Packet-

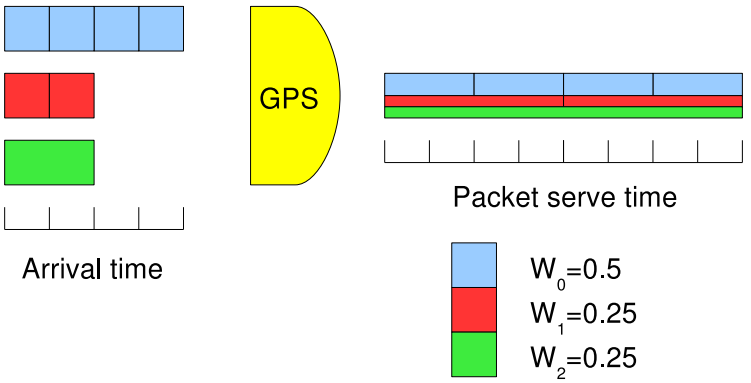


Figure 4.1: Generalized Processor Sharing.

by-Packet Generalized Processor Sharing or PGPS. It turns out that PGPS is identical to the weighted version of Fair Queuing or WFQ [Demers et al. 90].

The WFQ was found to be insufficiently accurate when approximating GPS. For this reason, Worst-case Fair Weighted Fair Queuing or WF^{2}Q [Bennett and Zhang 96] was proposed.

Nevertheless, WFQ has remained a popular class of service disciplines to provide tight end-to-end rate and delay guarantees while being fair in how it handles traffic. The downside is that the implementation of this class of scheduling disciplines is complex since they need to approximate the operation of GPS. A number of variants has been introduced (see [Guerin and Peris 99] for a brief review) that simplifies implementation but also reduces the fairness or the quality of guarantees provided.

4.1.2. Earliest deadline first

With Earliest Deadline First (EDF) algorithm packets are tagged with deadlines, which are used for packet scheduling. The deadline of a packet represents the ideal delivery time of that packet, in order to satisfy the bandwidth and latency requirements of its traffic flow.

The scheduling in this case is very simple: the packet with the earliest deadline is always chosen (Figure 4.2). An advantage of this discipline over the WFQ variants is that EDF can separate the rate and delay guarantees provided; the WFQ policies can only provide a rate guarantee and, indirectly through it, a delay guarantee: a flow that needs very low delays would have to reserve a high rate.

In order to provide end-to-end guarantees with EDF schedulers, it is necessary to provide additional shaping at each node. This means that a mechanism,

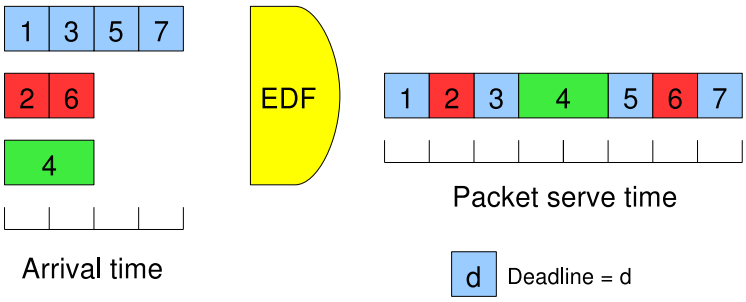


Figure 4.2: Earliest Deadline First.

like leaky buckets, must limit the amount of consecutive packets injected by one flow. With this addition it has been shown that the EDF discipline is the most efficient when guaranteeing end-to-end delays [Georgiadis et al. 94].

Nevertheless, both the simplified WFQ disciplines as well as EDF algorithm are too difficult to implement since they require separate queues for each flow. Large packet switches may have to handle 100,000s of flows, making fast hardware implementations of these policies impractical.

4.1.3. Summary

We have reviewed in this section the advantages of per-flow QoS. They can provide bounds on latency and they can also guarantee bandwidth of individual flows.

We have also seen that there are two main families of algorithms in this class: WFQ variants and EDF algorithms. Both are valid to provide adequate performance, but both classes are also very difficult to implement, due to the limitations of the switches to handle per-flow information.

4.2. Efficient Flow-Level QoS in High-Performance Interconnects

As stated before, there are no available implementations of per-flow QoS support in high-speed interconnects. The reason is that keeping information regarding so many flows is impractical.

In this section we propose a switch architecture that implements deadline-based scheduling in a very efficient way.

4.2.1. Motivation

The generally accepted solution for QoS provision in high-speed interconnects is traffic-class level QoS. This is based on aggregating traffic flows from several applications into a reduced group of service levels. These service levels are usually assigned to different VCs and some kind of QoS-aware scheduling is used.

The other family of QoS algorithms, per-flow QoS, is not considered for high-speed interconnects. We have seen in the previous section that these algorithms are very powerful, but difficult to implement. If an efficient solution existed that still was able to provide this powerful per-flow scheduling, it would be very intersting for cluster and interconnect designers. In the next section we present our proposal to solve this problem, which we believe accomplishes these objectives.

4.2.2. Efficient architecture for per-flow QoS support

In a typical cluster environment we have a set of hosts connected through a high-speed interconnect like InfiniBand or AS. The hosts have enough resources to maintain per-flow information for the traffic flows they originate. On the other hand, the switches that make up the interconnect have drastically less resources and usually cannot maintain more than a few different VCs. Unavoidably, the interconnect can only support aggregate QoS despite the hosts ability to keep track of per-flow information. In this work we explore if it is possible to maintain reasonable traffic differentiation through a very simple interconnect that only uses 2 VCs at the switch ports.

As discussed above, it is essential to keep the cost of interconnect switches as low as possible, because even a few VCs may push the cost of the switches to unacceptable levels. The hosts implement a per-flow EDF type of scheduling discipline and all the traffic is merged into the two VCs used by the interconnect. The interconnect switches perform a very simple sorting operation that can be implemented without the queue management complexity of the EDF scheduling. The effectiveness of the sorting operation depends on the fact that packets arrive from the hosts already sorted in priority.

We focus on the following traffic classes:

- High-priority low bandwidth control traffic. This traffic has to be delivered as fast as possible.
- High-priority real time traffic that has to be delivered so that it does not violate its deadlines and, thus, requires a certain amount of bandwidth availability.
- Best-effort low-priority traffic.

Note that the proposals in this chapter are aimed basically for the same kind of environment, that is, high-performance interconnects, that the proposals in the previous chapter. This means that the traffic considered before is the same that is considered here. However, we organize it in the aforementioned three traffic classes for clarity reasons and to facilitate the understanding of the following sections.

We also want to provide further differentiation for different types of best-effort traffic. We do not aim at providing specific QoS guarantees such as delay of rate bounds, but we focus on providing sufficient traffic differentiation beyond the limitations of the 2 VCs in the interconnect. We will show how, thanks to the scheduling done at the hosts, we can effectively differentiate among multiple traffic classes.

We want to adapt efficiently the Earliest Deadline First family of algorithms to a high-speed network. More specifically, we will use a variation of the Virtual Clock [Zhang 91] algorithm. In our architecture, each packet will carry one tag, the deadline, which is the cycle in which it is supposed to be delivered to the final destination host. In order to compute this, the sender host is responsible to keep some information about the flows with origin in that host.

A flow would be a single connection, like a TCP connection, or traffic from a single application. Each flow would have the following parameters: source, destination, a fixed route, and the information necessary to compute deadlines, which usually would be average bandwidth, but may vary depending on the type of flow, as we will see later.

In addition to deadline, packets have another tag while they are at the sender host: the eligible cycle. This indicates the earliest cycle in which a packet is allowed to get into the network and it is not used in the switches, therefore, it is not transmitted in the packet header.

A cornerstone of our proposal is to avoid any book-keeping of the flows at the switches. For scheduling, only the information in the header of packets is used: the deadline and the routing information.

We use an admission control similar to what is proposed for InfiniBand or PCI AS. Bandwidth reservation is performed at a centralized point and no record is kept in the switches. This makes the use of fixed routing mandatory, so that packets use the route they have reserved.

On the other hand, we have to provide a connectionless or unregulated service like UDP or ATM's UBR for best-effort traffic. In this case, we still propose to use fixed routing to avoid out-of-order delivery, which may happen with adaptive routing. Although we use fixed routing, the admission control can ensure load balancing when assigning paths, as opposed to deterministic routing, where there is only a single path between a given pair of hosts.

For unregulated traffic, a generic flow record is kept in the end-hosts, with the necessary parameters. In this case, there is no bandwidth reservation and

there is no guarantee of delivery. However, if we want to support several classes of best-effort traffic, we can configure several aggregated flows, each one with a different bandwidth to compute deadlines.

Summing up, we propose to inject packets from hosts using an unmodified EDF algorithm. However, at the switches we use just two VCs, implemented as FIFO queues, and apply a scheduling based on the deadlines packets include in their header.

4.2.2.1. Calculus of deadline

Taking into account the flow parameters, the packets are stamped in the end-hosts with the deadline tag. In addition, an eligible time tag is also used while the packet remains in the interface. For most flows, deadline of packet P_i is

$$D(P_i) = maximum(D(P_{i-1}),\ T_{now}) + \frac{L(P_i)}{BW_{avg}}$$

where $L(P_i)$ is the length of the packet P_i, T_{now} is the host's clock when the packet arrives from the application level, and BW_{avg} is the reserved average bandwidth for this flow. This computation does not consider the number of hops that a packet needs to reach its destination. However, this is fine in high-performance networks, where base latency is very short.

Some specialized types of traffic require a different method to compute bandwidth. Control traffic needs a latency as short as possible but takes almost no bandwidth. For this type of traffic, we would use no connection admission and BW_{avg} would be the link bandwidth. In this way, control traffic gets the maximum priority.

Multimedia traffic usually consists in bursts of packets followed by silence periods. Let us assume that we want to transmit a MPEG video sequence with average bandwidth of 400 Kbyte/s. Moreover, we know that the video sequence consists in one video frame each 40 milliseconds and the frame size can be between 1 and 120 Kbytes. For this kind of traffic, an average bandwidth assignation is not enough because during peak-rate periods it would introduce intolerable delays. We could use the maximum bandwidth (based on maximum frame size) to generate deadlines, but in that case two problems arise: first, if the frame to be transmitted is short, we are introducing unnecessary bursts of packets. Secondly, the latency of each frame will vary a lot, since it will depend on the size of frames.

We propose to use the following strategy: the user fixes a desired latency per frame, for instance 10 milliseconds. Upon reception of a new frame, we compute the number of network level packets it will generate. For instance, if frame size is 80 Kbytes and the maximum transfer unit (MTU) is 2 Kbytes, it will generate

40 packets. In that case, for each packet P_i, deadline is

$$D(P_i) = maximum(D(P_{i-1}),\ T_{now}) + \frac{10\ msecs}{Parts(F_i)}$$

where $Parts(F_i)$ is the number of packets generated by the frame to which P_i belongs. In this way, every frame will have a latency close to 10 milliseconds, independently of frame size, and a smooth distribution of packets. This is good both for avoiding unnecessary bursts in the network and for preventing a lot of variability in frame latency.

Another central element of our proposal is that the deadline of the packets is not recomputed at the switches. The main reason is that the ideal implementation of a high-speed switch is a single chip to minimize delays. That means that silicon area is limited and there is no space for recording information regarding all the flows traversing the switch. Moreover, recomputing the deadline would introduce additional delay and would not introduce any significant benefit in a high-performance network.

The use of eligible time in the end-nodes is optional, since some traffic classes do not tolerate being smoothed. When it is used, typically for multimedia traffic, we propose to compute eligible time of a packet as its deadline minus a fixed value, the *eligibility factor*. We have found in our tests that 20 microseconds works well. In this way, when a traffic flow is smoothed, packets leave the end-node at most 20 microseconds before their deadline, not earlier (they can leave later due to competition for the link). This strategy, together with the aforementioned method to compute deadlines for multimedia traffic, produces almost constant latency for multimedia frames, which in turn reduces jitter, and also produces low burstiness, since packets are more evenly distributed.

4.2.2.2. Packet scheduling

Ideally, using deadline-based QoS, each switch would schedule packets implementing an EDF algorithm. However, searching for the packet with the minimum deadline through all the buffers is not practical. An alternative is to implement a heap buffer, which always keeps the packet with the lowest deadline at the top of the queue. A design for this is discussed in [Ioannou and Katevenis 01]. However, the associated cost is not practical for high-speed switches with high radix (number of ports).

On the other hand, we have observed that, when traffic is regulated (no over-subscription of the links), the switches can just take into account the first packet at each input buffer in arrival order. The idea is that traffic coming from the interfaces has already been scheduled and this traffic is coming in ascending order of deadline. This being so, it is possible to just consider the first packet at each queue, in the confidence that packets coming afterwards have higher deadlines.

The behavior of the switch would be analogous to a sorting algorithm: if the switch has as input ordered chains of packets and has to produce at the output an ordered sequence, it only needs to look at the first packet of each input.

The main limitation of this algorithm is that packets may not always come ordered from the interfaces. It may happen that when no more low-deadline packets are available, a high-deadline packet is transmitted, especially if eligible time is not being used. If the high-deadline packet has to wait in a switch input queue, and other packets with lower deadline are transmitted from the network interface, they would be stored after the high-deadline packet in the same queue. Thus, the arbiter would penalize the low-deadline packets, because they would have to wait until the high-deadline packet is transmitted. We call this an *order error* situation.

The order error situation, alongside with the eligibility time feature, are illustrated with an example at Figure 4.3. Let us assume that at a given host, there are two applications injecting traffic: a multimedia application and a control application. At a certain point there are several multimedia packets waiting for injection and no control packet at the host. In $T = 4$, the first multimedia packet can proceed because actual time, 4, plus the eligibility factor, 3 in this example, is less or equal than the packet's deadline, 7. In the same way, in $T = 9$, next multimedia packet can proceed. However, in $T = 11$ a control packet is generated and immediately forwarded. In that case, there is an order error because the deadline of the control packet is smaller than the deadline of the preceding multimedia packet in the switch buffer. Also note that, the maximum magnitude of order errors is the eligibility factor, 3 in this example. In our simulations, we use 20 microseconds obtaining good performance.

Order errors will violate our assumptions and degrade the service offered to the low-deadline packets. We will analyze this problem and how to attenuate it, later. Note that there is no chance for starvation since traffic is regulated and there is enough bandwidth guaranteed for QoS-requiring flows.

On the other hand, there is also unregulated (best-effort) traffic that could interfere with the regulated traffic. This is the reason why we propose to use two different VCs: one for regulated traffic and the other for non-policed traffic. The regulated traffic has absolute priority over the best-effort traffic. Therefore, we can guarantee that regulated traffic will not be delayed by congestion and still accept best-effort traffic to make use of the remaining bandwidth.

There are also two VCs at the end-hosts, but packets are always in deadline order. In the regulated traffic VC, we propose two queues (Figure 4.4), one feeding the other. In the first queue, packets are stored in ascending eligible time. As soon as the first packet in the queue is eligible, it goes to another queue where packets are sorted according to ascending deadlines. Packets are injected from this queue as soon as the link is available and there are enough credits. On the other hand, packets in the best-effort VC are also sorted by

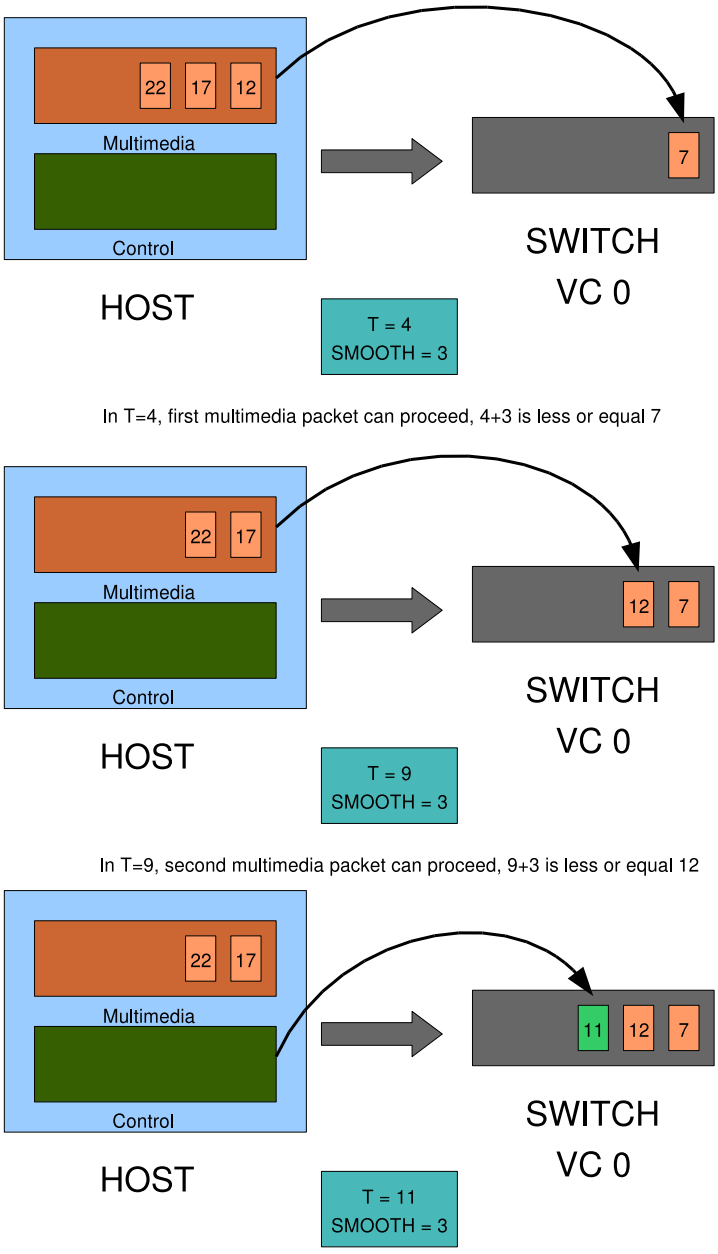


Figure 4.3: Example of order error.

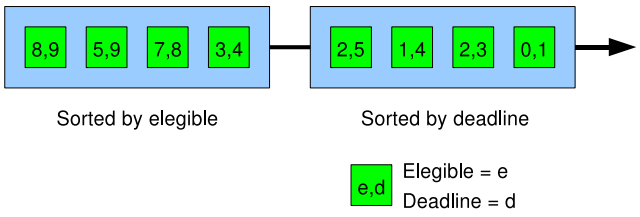


Figure 4.4: QoS packet queues at end-nodes.

deadline. They are injected only when the link is available, there are credits, and the regulated traffic VC has no packets ready to inject (there might be packets waiting for eligible time).

4.2.2.3. Clock synchronization

In a first glance, precise clock synchronization between the end-hosts would be needed for the viability of our proposal. However, we can avoid synchronization with a simple strategy. The deadlines we are computing consist in a base time value, linked to local clock, plus some additional time to reach the destination. By subtracting local clocks, we would have the time to reach the final destination (TTD). This value has the advantage of not needing any synchronization of clocks. The host indicates that a packet has to reach its destination before n milliseconds instead of the absolute time to do the same.

The problem is that this value would change every clock cycle, which is undesirable. However, we can reconstruct a packet's deadline by adding the local clock again. Therefore, our strategy would be the following: Packets receive a deadline in the hosts and are stored as usual. When a packet is about to leave a host or switch, the TTD is computed:

$$TTD_i = D_i - T_{local}$$

where T_{local} is the local clock at the host or switch the packet is leaving. When the packet arrives at the next switch, the deadline is reconstructed adding to the TTD of the packet the new local clock. This deadline is used for scheduling locally and, when the packet is chosen to be transmitted, a new TTD is computed and put in the packet header.

Regarding eligibility time, when this feature is active, no smoothed packet can leave the interface with a TTD higher than the eligibility factor. For instance, in the simulations, multimedia packets have a maximum TTD of 20 microseconds. This also bounds the number of bits necessary to store the TTD at the packet header.

The only drawback of this proposal is that the CRC of the header would be needed to be recomputed at each hop. This is because a packet's TTD will be

changing with each hop. However, other fields of the header also change with each hop, for instance the pointer to the next hop in PCI AS's source routing, and so, recomputing of the CRC of the header is usually necessary anyway.

4.2.3. Improved architecture

We have observed through simulation that the performance achieved by the previous proposal is similar to having full ordered queues. However, the latency of the most demanding flows may be increased as much as 25% due to order errors. To attenuate this effect, we propose an improvement to this proposal.

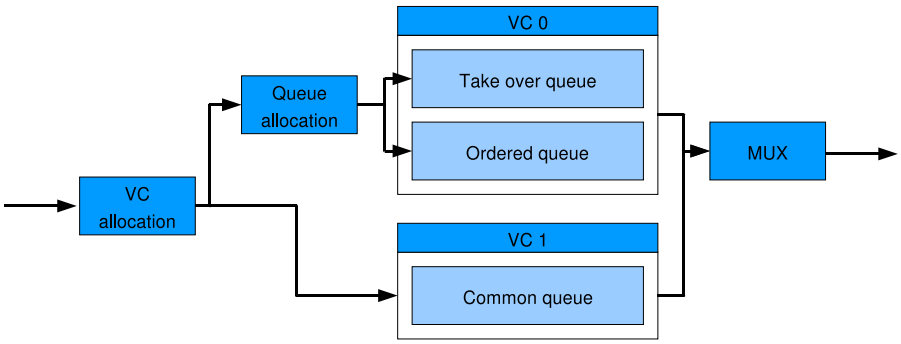


Figure 4.5: New buffer structure for the switch ports.

The key idea consists in splitting in the switches the regulated traffic VC into two FIFO queues (Figure 4.5). These queues dynamically share the space assigned to the VC and, potentially, both queues can grow up to all the available space. One of these queues is the *ordered queue* and the other is the *take-over queue*. When a packet arrives, its deadline is compared with the deadline of the packet in the last position of the *ordered queue*. If the new packet has a higher deadline, it is put in the *ordered queue*. If the deadline is smaller, the packet goes to the *take-over queue*.

The dequeuing algorithm is very simple: the packet chosen to be transmitted is always the one with the smallest deadline of both queue heads. In this way, we give low-deadline packets a chance to advance over packets with a high deadline. With this algorithm the amount of order errors is not completely eliminated, but greatly diminished, as we will see in the performance evaluation.

This improvement to the buffer ports does not introduce out-of-order delivery. The demonstration can be found in the next section. Note that *out-of-order* delivery to the destination means that packets from a particular flow do not arrive in the same order in which they were injected from the source end-node. As opposed, when we talk about *order errors*, it means that packets from different

flows are put in a queue out-of-order of deadline tags. Therefore, out-of-order delivery has to be avoided because it would require re-order buffers at the end-nodes, which are expensive. On the other hand, *order error* simply means that sometimes the scheduler will not choose the packet with the earliest deadline. In the latter case, some packets would be delayed, but no special hardware would be needed.

4.2.3.1. In-order delivery

In this section we are going to prove that the improved structure for the QoS VC of the switches does not introduce out-of-order delivery. Note that this is an important issue: in many high-speed standards out-of-order delivery is explicitly forbidden (for instance, in PCI AS). Firstly we indicate the notation in Table 4.1, introduce several initial hypotheses, and present the enqueuing and dequeuing algorithms. Finally, we prove some theorems.

U	Upper queue (Take over queue)
L	Lower queue (Ordered queue)
U_i	Packet at position i in the U queue. ($i = 1$, packet at the front of the queue)
L_i	Packet at position i in the L queue. ($i = 1$, packet at the front of the queue)
m_U	Number of packets in the U queue at a given moment
m_L	Number of packets in the L queue at a given moment
$D(p)$	Deadline function. $D(L_i)$ is the deadline of the ith packet of the L queue
n_F	Number of packet flows
$F^1, ..., F^{n_F}$	Packet flows
np_j	Number of packets of the jth flow
$F^j_{np_j}, ..., F^j_1$	Individual packets of the flow F^j in the generation order and, thus, in arrival order. F^j_1 is the packet at the first position
$Dep(p)$	Time at which packet p leaves the system
$I(p)$	Arrival time of the packet p at system

Table 4.1: Notation.

Initial hypotheses

Two conditions are accomplished by every flow:

$$D(F^j_k) < D(F^j_{k+1}) \quad 1 \leq j \leq n_F \ , \ 1 \leq k < np_j, \ np_j \geq 2 \tag{4.1}$$

$$I(F^j_k) < I(F^j_{k+1}) \quad 1 \leq j \leq n_F \ , \ 1 \leq k < np_j, \ np_j \geq 2 \tag{4.2}$$

Intuitively, the previous expressions say that packets from a flow arrive ordered at the system and they have increasing deadlines.

Now, we will formally define the enqueuing and dequeuing algorithms:

***Definition* 1.** *(Enqueuing algorithm) Enqueuing of an incoming packet p works as follows:*

- *If both queues are empty, store p in the L queue.*
- *If there are m_L packets in the L queue*
  - *If $D(p) \geq D(L_{m_L})$ store p in the L queue.*
  - *Else, store p in the U queue.*

Note that incoming packets always have space in the system due to the credit flow control. Also note that the two queues can dynamically take all the memory allowed for the VC and, therefore, it is not possible for a queue to become full while there is space in the other queue.

***Definition* 2.** *(Dequeuing algorithm) The algorithm for removing packets works as follows:*

- *If both queues are empty, there is no packet to choose.*
- *If there are packets only in the L queue, L_1 is chosen.*
- *If there are packets in both queues, the packet with the smallest deadline between $D(L_1)$ and $D(U_1)$ is chosen.*
- *A situation where there are only packets in the U queue is not possible (Lemma 1).*

With respect to the flow control mechanism, the following possibility could arise: if two packets are available and $D(U_1) < D(L_1)$ but L_1 is smaller (in bytes) than $D(U_1)$ and there are only credits for L_1, the latter would be forwarded out of order in case both belonged to the same flow. This would corrupt the dequeuing discipline, but we prevent this by imposing the condition that only the packet with the smallest deadline of the potential two available is checked for credits and, thus, for transmission.

***Definition* 3.** *(Out-of-order delivery) Out-of-order delivery occurs if packets from an individual flow leave the system in a different order from arrival order. Therefore, there is out-of-order delivery iff*

$$\exists\ j,k\ /\ Dep(F_k^j) > Dep(F_{k+1}^j) \qquad 1 \leq j \leq n_F\ , \quad 1 \leq k < np_j$$

In the following, we are going to prove several theorems, some of them more or less intuitive, which will permit us to prove that, given the previous enqueuing and dequeuing algorithms, out-of-order delivery is not possible in our proposed queue system.

***Theorem* 1.** *Packets in the L queue are in deadline order.*

$$D(L_i) \leq D(L_{i+1}) \qquad 1 \leq i < m_L,\ m_L \geq 2$$

Proof: By reductio ad absurdum: if packets in the L queue are not in deadline order

$$\exists\ i\ /\ D(L_{i+1}) < D(L_i) \qquad 1 \leq i < m_L$$

but that would contradict the enqueuing algorithm, which only stores a packet in the L queue when its deadline is higher than or equal to that of the last packet in the queue. Since packets can only leave the queue in a FIFO discipline, this order is preserved by the dequeuing algorithm. □

***Theorem* 2.** *The packet with the highest deadline in the two queues is always the last packet in the L queue.*

$$D(L_i) \leq D(L_{m_L}) \qquad 1 \leq i \leq m_L$$

$$D(U_j) < D(L_{m_L}) \qquad 1 \leq j \leq m_U$$

Proof: The first part of the theorem follows from Theorem 1.

On the other hand, packets in the U queue always have a smaller deadline than the last element of the L queue. Following the enqueuing algorithm, in the event of a new packet arriving with a larger deadline than the maximum, it would be stored in the last position of the L queue and would become the new maximum deadline.

□

***Corollary* 1.** *L_{m_L} is always the last element to leave the system.*

Proof:

No packet L_i, $i \neq m_L$, in the L queue can leave earlier due to the FIFO discipline. On the other hand, since all the packets in the U queue have a smaller deadline than L_{m_L} (Theorem 2), they cannot leave earlier than it with the dequeuing algorithm given, which always chooses the packet with the minimum deadline.

□

***Lemma* 1.** *A situation where there are only packets in the U queue is not possible.*

Proof: A situation where the L queue is empty and the U queue has packets cannot be obtained from having both queues empty since the enqueuing algorithm indicates that if the two queues are empty and a packet arrives, the packet is stored in the L queue.

Hence, an empty L queue and U queue with packets could only arise from a situation in which both queues have packets and dequeuing takes place in both. However, from Theorem 2, the packet with the highest deadline is in the L queue and thus all the packets in the two queues will leave before the former and U queue will become empty before L queue. □

***Theorem* 3.** *There is no out-of-order delivery. Formally,*

$$Dep(F_k^j) < Dep(F_{k+1}^j) \quad 1 \leq j \leq n_F \ , \quad 1 \leq k < np_j,\ np_j \geq 2$$

Proof: Since arrival of packets is ordered, conflicts can only arise if F_{k+1}^j manages to overcome F_k^j while it is still waiting at the system. That means that we have to study the cases where both packets are stored in the queues. Let's analyze the different possible cases:

- When they arrive, both F_k^j and F_{k+1}^j go to the same queue, either L or U. In this case, they leave in arrival order because both U and L queues are FIFO queues. Since they arrived ordered (Eq. 4.2), they leave ordered.
- Upon arrival, F_k^j goes to the L queue and later F_{k+1}^j goes to the U queue. This may happen if $D(F_k^j)$ is the maximum deadline at the arrival time, but before of the arrival of F_{k+1}^j at least one packet p arrives with $D(p) > D(F_{k+1}^j)$.

  ¿From Theorem 1 we know that the L queue is ordered and from dequeuing algorithm when F_{k+1}^j is ready to leave, it means that its deadline is smaller than that of any packet in the L queue. Since $D(F_k^j) < D(F_{k+1}^j)$ (Equation 4.1), it is sure that packet F_k^j already left and the order is preserved.

- When they arrive, F_k^j goes to the U queue and later F_{k+1}^j goes to the L queue. It may happen if $D(F_k^j)$ is smaller than the maximum but $D(F_{k+1}^j)$ is larger.

  L_{m_L} is the last packet in the L queue when F_k^j is stored in the U queue. From Theorem 2, L_{m_L} has a higher deadline than any packet in the U queue at that moment, including F_k^j. Therefore, it will leave later than all those packets: to leave earlier it would need to be compared with a packet from the U queue with a higher deadline, but this is not possible. In consequence, it is true that:

  $$Dep(F_k^j) < Dep(L_{m_L})$$

  Since F_{k+1}^j is positioned in the L queue behind L_{m_L} (maybe with other packets between), it cannot leave earlier (FIFO queuing) and, therefore, it has to leave after F_k^j:

  $$Dep(L_{m_L}) < Dep(F_{k+1}^j)$$

  $$Dep(F_k^j) < Dep(L_{m_L}) < Dep(F_{k+1}^j) \Rightarrow Dep(F_k^j) < Dep(F_{k+1}^j)$$

  □

4.2.4. Summary

In this section we have proposed an efficient architecture to provide per-flow QoS in high-speed interconnects. It is based on taking advantage of the implicit information in the order at which packets are injected by end-nodes.

We have also shown an improved design, aimed at reducing the impact of order errors. As we will see in the following, we obtain a performance almost as good as an ideal architecture with random access buffers.

4.3. Performance Evaluation

In this section we will evaluate the performance of our proposal for per-flow QoS support in high-speed interconnects. The methodology will be very similar to that of the previous chapter and, therefore, we will use an interconnection simulator.

The network used to test the proposals is again a butterfly multi-stage interconnection network (MIN) with 128 end-points. The actual topology is a folded (bidirectional) perfect-shuffle. We have chosen a MIN because it is a common topology for clusters. Although not included here, we have also tested direct

networks (torus and meshes) obtaining similar performance. The switches use a combined input and output buffer architecture, with a crossbar to connect the buffers. We use virtual output queuing at the switch level, which is the usual solution to avoid head-of-line blocking.

We will evaluate five different architectures:

- *Traditional 2 VCs.* A traditional switch architecture with some QoS support. This is based on the PCI AS specification and provides two VCs to distinguish between two broad traffic categories. It also has CAC, but no traffic smoothing (it is work-conserving).
- *Traditional 4 VCs.* A more advanced switch architecture, also based in PCI AS specification. In this case, there is a VC per-traffic class (four in our study). Also, we have implemented token buckets for multimedia and best-effort traffic, in order to alleviate the problems with bursty traffic.
- *Ideal.* An ideal switch architecture based on our EDF algorithm. This implements two VCs (regulated and unregulated traffic), but each one is a heap queue which always keeps at the top the packet with the highest deadline. Order errors would not happen in this case but the implementation of this architecture would be unfeasible due to the kind of buffers. However, it is a good reference to compare with.
- *Simple 2 VCs.* A simple switch architecture based on our first proposal (presented in Section 4.2.2). This emulates the previous ideal architecture, but, since order errors are possible, performance will be degraded.
- *Advanced 2 VCs.* The improved version of the previous architecture. This implements the take-over queue proposed in Section 4.2.3.

In all the cases, the switches implement 16 ports and 8 Kbytes of buffer per VC. In our tests, the link bandwidth is 8 Gb/s. The remaining parameter values are picked from the AS specification. In general, all the parameters used in the simulations are quite typical for high-speed interconnects.

The CAC we have implemented for QoS-requiring traffic is a simple one, based on average bandwidth requirements. Each connection is assigned a path where enough bandwidth is assured. The CAC guarantees that less than 70% of bandwidth is used by QoS traffic at any link. The other 30% of available bandwidth will be used by unregulated traffic. We also use a load-balancing mechanism when a QoS connection is stablished, which consists in assigning the least occupied route among the possible paths.

Table 4.2 presents the characteristics of the traffic injected in the network. We follow the recommendations of The Network Processing Forum Switch Fabric Benchmark Specifications [Elhanany et al. 05]. We have considered a mix of

Name	% BW	Application frame	Notes
Control	25	[128 bytes, 2 Kbytes]	Small control messages
Multimedia	25	[1 Kbyte, 120 Kbytes]	3 Mbyte/s MPEG-4 traces
Best-effort	25	[128 bytes, 100 Kbytes]	Self-similar internet-like traffic
Background	25	[128 bytes, 100 Kbytes]	Self-similar low-priority traffic

Table 4.2: Traffic injected per host.

QoS-requiring traffic flows and best-effort flows. In this way, the workload is composed of 2 different traffic classes: two QoS traffic classes and two best-effort traffic classes. Note that there are many individual flows of the four categories, each one with the characteristics shown in the table. We have also tested our proposals using the traffic load presented in the previous chapter.

Control traffic models traffic from applications that demand a latency as short as possible. Since it has a connection-less nature, it is not subject to the CAC or the load-balancing. *Multimedia* traffic consists in actual MPEG video sequences, transmitted through the network. Best-effort traffic classes, *Best-effort* and *Background*, are not subject to CAC or smoothing. For this reason, they can saturate network links and impact performance. These traffic classes are modelled with self-similar traffic, which is composed of bursts of packets heading to the same destination. The packet size is governed by a Pareto distribution, as recommended in [Jain 91].

4.3.1. Simulation results

The next sections contain the scenarios we have tested for this performance evaluation. The main advantage of deadline-based scheduling is that a fine-grained QoS can be offered. In this way, we will see that video sequences obtain constant latency, due to the use of these deadlines. Moreover, control traffic achieves very short latencies, while best-effort traffic obtains high-throughput, mostly because we have eliminated the bursts of video packets.

We will study the following scenarios:

- Video scenario. We study if our proposals work properly with several different video traces.
- Mixed traffic scenario. In this case, we combine video and control traffic.
- Realistic traffic scenario. We repeat this scenario that we proposed in the previous chapter and compare the performance of the new switches.
- QoS traffic acceptance. We test the scalability of our proposals by varying the amount of QoS traffic injected.

Let us start with the first of them.

4.3.2. Video scenario

In this scenario, we show how our proposal for efficient deadline-based scheduling works with video traffic. For this purpose, we use four different kinds of video sequences, with different properties, that are injected into the network.

The sequences were chosen because they are popular in the evaluation of video performance [Vid]. They are the following:

Name	CIF/QCIF	Bandwidth
Highway	QCIF	0.492 Mb/s
Paris	CIF	3.42 Mb/s
Mobile	CIF	9.71 Mb/s
Funny	CIF	13.00 Mb/s

Table 4.3: Video sequences for performance evaluation.

In the deadline-based architectures, we use the QoS VC for the four groups of video sequences, and we use 50% of injection capacity in this way. The rest is used by best-effort traffic.

On the other hand, we also use, for comparison purposes, a 8 VC architecture, with a VC for each of the service levels (four video plus four best-effort).

As we can see at Figure 4.7, performance is exactly the same for any video sequence and load for the deadline-based architectures. By using the deadline tags and the eligibility time, we are able to obtain a latency of 10 milliseconds for every video frame (individual packets have much shorter delays).

On the other hand, the traditional architecture offers variable performance, which depends on the traffic load and on the characteristics of the video sequence being transmitted. Moreover, due to the burstiness of video traffic, performance of best-effort traffic is much worse in this case, as can be seen in Figure 4.6.

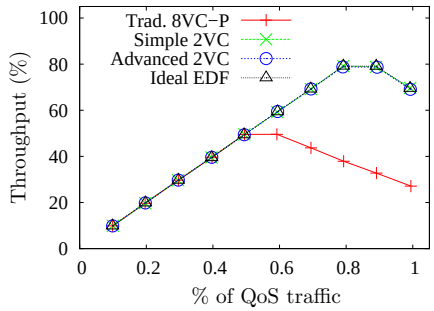


Figure 4.6: Throughput of best-effort traffic classes.

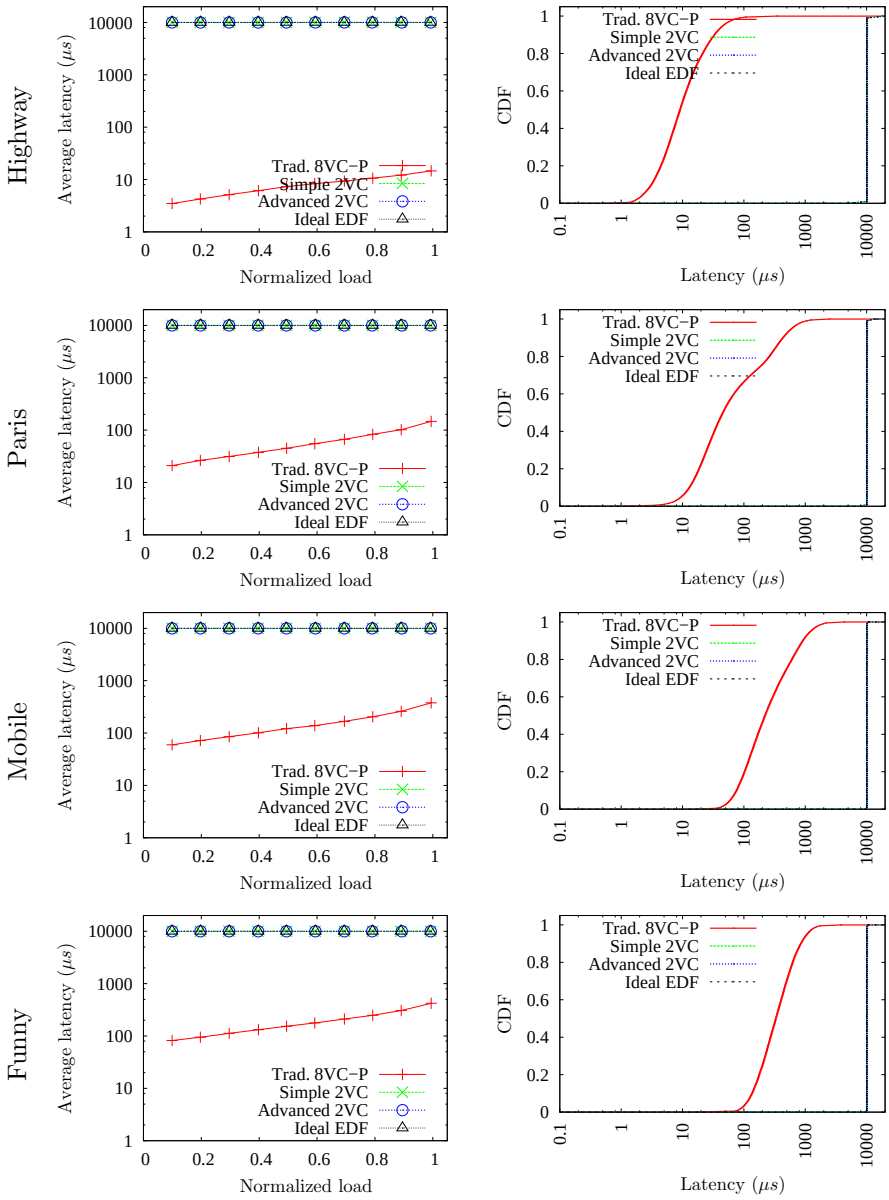


Figure 4.7: Results of video scenario with deadline-based scheduling.

4.3.3. Mixed traffic scenario

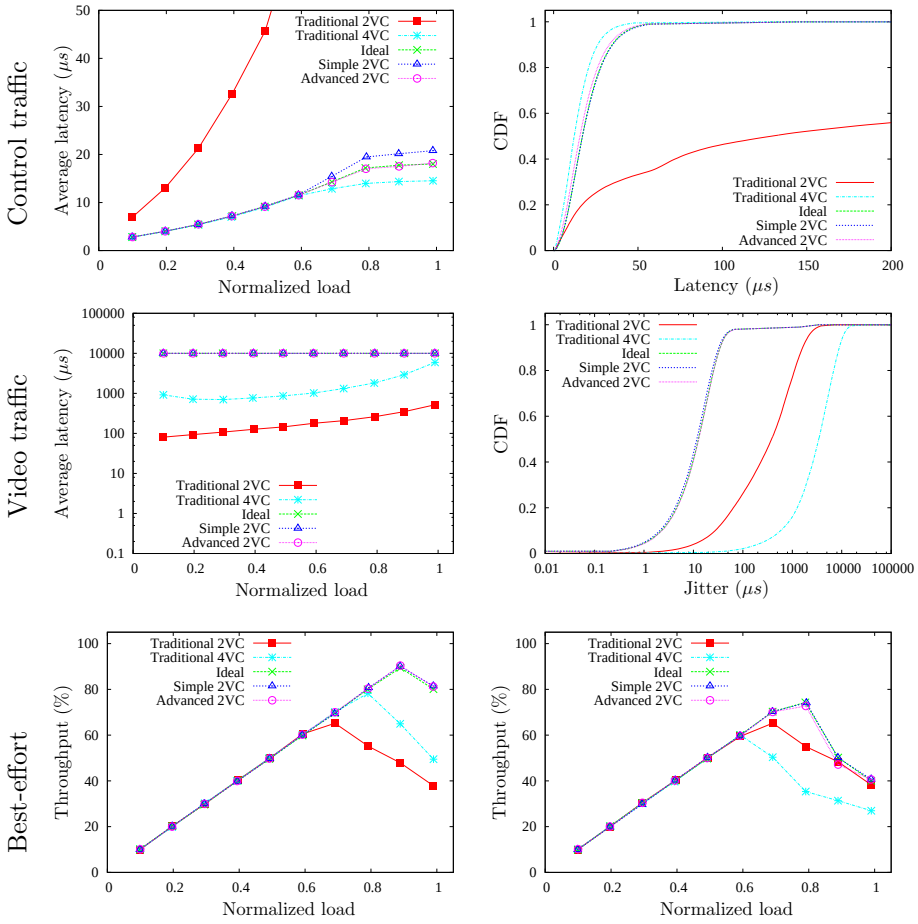
In this scenario we mix video traffic with control traffic in the same VC. In this way, we want to see if this control traffic is not affected by the video packets.

In Figure 4.8, we can see the performance of the most delay-sensitive traffic class we are considering, *Control* traffic. The best results correspond to the *Traditional 4 VCs* case. This is because, in this case, *Control* has its own VC and maximum priority. On the other hand, EDF architectures also have good performance. We can see at the bottom of the Figure that our simplest proposal, *Simple 2 VCs*, increases maximum latency up to 15% compared with the *Ideal* case. On the other hand, the *Advanced 2 VCs* proposal has almost the same behavior as the *Ideal* case. Finally, the *Traditional 2 VCs* case has very bad performance.

Regarding the performance of *Multimedia* traffic, when using the method we propose to compute deadlines, the average latency of video frames is almost exactly 10 milliseconds (the value configured as desirable latency). Note that latency results refer to full transfers and not to individual packets (i.e. latency is for each frame of the video sequence). Looking at the CDF of latency, we notice that there is little variation in latency for EDF-based architectures (the probability of a latency of 10 milliseconds is more than 99%). On the other hand, latency can vary considerably when using traditional architectures, which would introduce a lot of jitter, as can be seen in the figure. Note that the *Traditional 4 VCs* case increases latency compared with the *Traditional 2 VCs* case due to the use of token buckets.

Finally, the figure also shows the performance in terms of throughput of the two best-effort classes we have considered. For the *Traditional 2 VCs* case both classes look the same (they share VC 1) and, thus, receive the same performance. On the other hand, the EDF-based architectures can label each packet with deadlines according to the reserved bandwidth of each flow. In this way, not only can we differentiate multiple classes within a single VC, but we can guarantee minimum bandwidth if we are careful assigning weights to the different best-effort flows. Finally, the *Traditional 4 VCs* case is able to offer differentiation, but the achieved throughput is not as good as with the EDF architectures. The reason is that, although using token buckets, they have to be configured to allow peak rate of video transfers and, therefore, they alleviate but not completely eliminate the burstiness of *Video* traffic.

We can conclude that EDF-based architectures are clearly superior to a traditional two classes QoS. Note that the cost of these architectures is similar, except the *Ideal* architecture. Using our proposals, the only difference is a slight increase of latency for *Control* traffic compared with the *Traditional 4 VCs* architecture.



(a) Best-effort (b) Background

Figure 4.8: Results of mixed traffic scenario.

4.3.4. Realistic traffic scenario

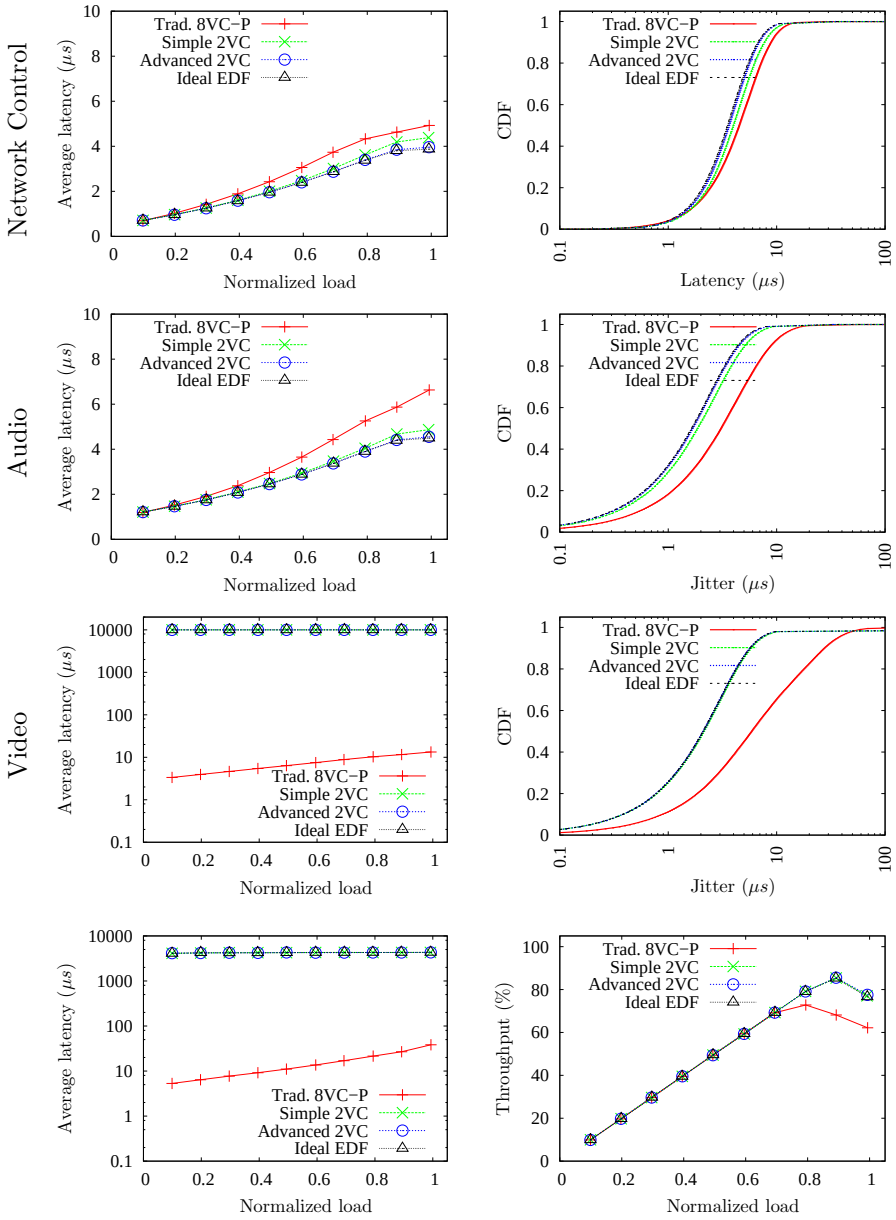
In this scenario we use the same traffic pattern we used in the previous chapter with the VC-based QoS. We compare the performance of our deadline-based QoS switches with the performance of a traditional 8 VC switch as specified by InfiniBand or PCI AS.

We will have four service levels of QoS traffic: *Network Control*, *Audio*, *Video*, and *Controlled Load*. Also, we have four levels of best-effort service levels. However, we will focus only on performance of the QoS service levels.

The Figure 4.9 shows the performance of the QoS service levels. Regarding *Network Control* traffic, we can see that the deadline-based QoS switches offer the best performance. They can reduce the delay of these packets since deadline-based scheduling is more appropriate than table-based WRR for latency requirements. Similar results are obtained for *Audio* traffic.

Regarding *Video* traffic, we obtain a constant latency of 10 milliseconds (the value configured as desirable latency). On the other hand, latency varies when using traditional switches. For that reason, we obtain better jitter results.

In the last row of the figure we have *Controlled Load* latency results. These are similar to video results. We also have a throughput plot of the four best-effort traffic classes. We can see that our proposals improve the obtained throughput, compared with the traditional switch model.



(a) Controlled Load (b) Best-effort

Figure 4.9: Realistic traffic scenario with table-based WRR.

4.3.5. QoS traffic acceptance

In this section we evaluate which amount of QoS traffic can be accepted by each architecture before QoS requirements are not satisfied. Moreover, we can see the behavior of the three EDF variants we are considering under stressing QoS load.

In this scenario we vary the proportion of QoS traffic, from 10% to 90% of the total available network load. We fill in the remaining bandwidth with best-effort traffic. Therefore, input links are injecting at 100% of their capacity. We can see that the different traffic classes saturate at different points when using the five architectures. In this way, QoS requirements are satisfied only up to a certain QoS load.

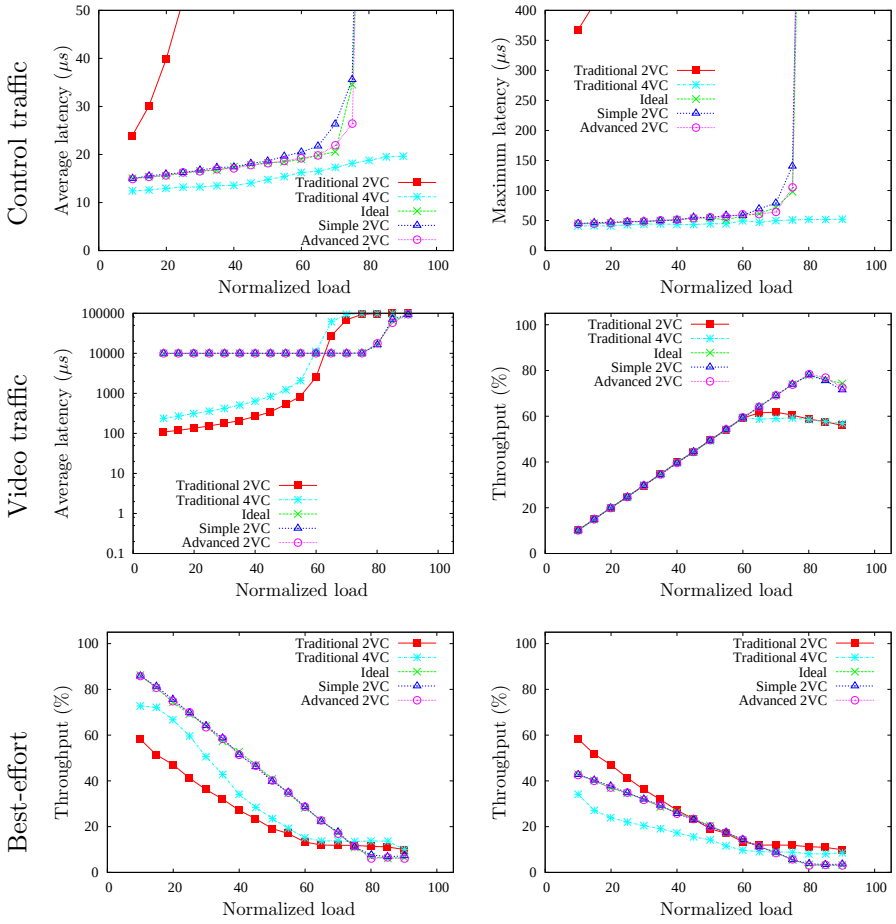
In Figure 4.10 we can see *Control* latency results. For the *Traditional 2 VCs* case results are very bad for almost any QoS load. On the other hand, the *Traditional 4 VCs* case offers good performance at any amount of QoS traffic, because *Control* traffic has its own VC and maximum priority. Finally, the three EDF architectures, including our two proposals, have good performance up to a QoS load of 75%. Note that the *Advanced 2 VCs* case reduces the latency of this traffic class up to a 20% compared with the *Simple 2 VCs* case.

Regarding *Video* traffic, we also see at Figure 4.10 that latency is 10 milliseconds for every video frame using the three EDF architectures up to a load of 75%. On the other hand, both traditional architectures start losing[1] *Video* packets at a QoS load of 60%.

Finally, we can observe the throughput of the two best-effort traffic classes we have considered. In this case, the EDF architectures offer the best performance for *Best-effort*. For *Background* traffic, results are similar in all the cases. When using the EDF algorithms, more best-effort traffic can be injected in the gaps left by QoS traffic.

Summing up, EDF architectures provide very good performance for all traffic classes up to a 75% of QoS traffic (100% network load). On the other hand, traditional architectures are only able to handle up to a 60% of QoS load with similar results. Our proposals emulate very well the behavior of an ideal architecture, even with a lot of QoS traffic.

[1] Packets are discarded at the end-nodes after waiting 100 milliseconds for injection.



(a) Best-effort (b) Background

Figure 4.10: Results of traffic acceptance scenario.

4.3.6. Summary

In this section we have tested our proposal for efficient deadline-based scheduling. In the results, we have shown the superiority of this kind of algorithms. Moreover, we have also shown that our proposals, which are easy to implement, offer excellent performance.

Our first attempt, the *Simple 2 VCs* architecture works very well with video traffic, but increases the latency of control packets when mixing them with video traffic. This is fixed with the *Advanced 2 VCs* architecture, which using a take-over queue greatly diminishes the problem.

4.4. Conclusions

In this chapter we have explored how to adapt efficiently the Earliest Deadline First family of algorithms to the high-speed network environments. As far as we know, no similar attempt has been made since some adaptations of ATM, due to the cost of using ordered buffers. On the other hand, our proposal is an architecture which, using FIFO buffers, offers almost the same performance, even for the most delay-sensitive traffic.

We also have compared our proposals with typical VC-based QoS, as can be found in recent standards such as InfiniBand or PCI AS. We have seen that, for similar cost in terms of the silicon area, our proposals offer a much better performance. Achieving similar performance in a traditional multi-VC architecture would require a larger number of VCs that would significantly increase the cost of the switches and would limit the applicability to high-speed interconnects.

CHAPTER 5

Integrated QoS Support and Congestion Management

CONGESTION is a well-known problem in lossless high-performance interconnects. Congestion dramatically degrades network performance because it leads to the appearance of head-of-line (HOL) blocking.

In this chapter, we start by reviewing congestion and the proposed solutions for this problem. As we will see, the most effective traditional proposals require too many resources for being implemented in large networks.

If we wanted to implement QoS support in addition to congestion management, the total implementation cost would be unacceptable. For this reason, we focus on a novel proposal for congestion management that offers excellent results. Based on it, we propose an integrated switch architecture that offers at the same time congestion management and QoS provision very efficiently. This is possible because we combine our proposal for QoS provision with this recent, scalable, and efficient proposal for congestion management. As we will see, we have evaluated the new switch architecture and results are excellent.

5.1. Congestion in High-Performance Interconnects

The interconnection network is becoming the most expensive part of many current computing and communication systems. In fact, current network components are quite expensive compared to processors. Furthermore, power con-

sumption in current network technologies has increased due to the growing link speed.

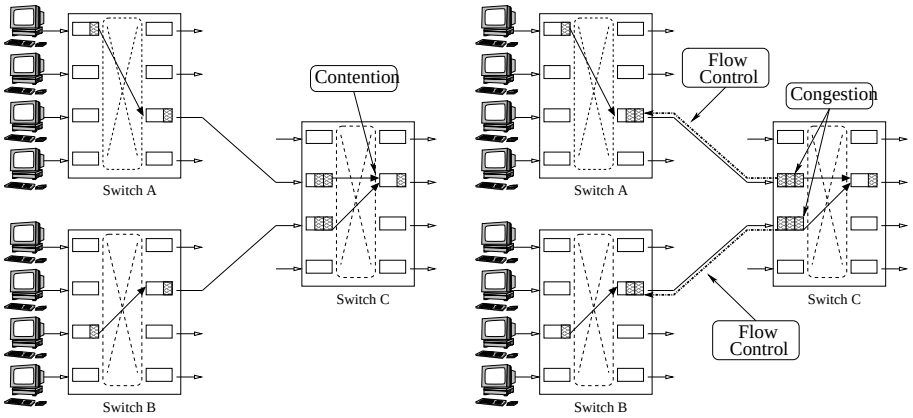
Therefore, both the cost and consumption constraints may limit network size, making desirable a reduction of network components while keeping constant the number of end-nodes in the network. However, this solution may lead to a higher link utilization that may produce serious problems. Traditionally, network designers overdimensioned the network in order to avoid these problems, but, as was explained, this solution is nowadays too expensive and power consuming. Therefore, more clever approximations are needed.

The origin of congestion is contention, which happens when several flows of packets simultaneously request the access to some network resources (typically, a switch output port). If the internal speedup of switches is not enough for attending several requests at the same time, the access to the requested output port will be granted to only one packet, while the rest must wait. Figure 5.1 (a) shows a contention situation caused by two incoming flows requesting the same output port at Switch C. In this example, it is assumed that switch speedup is 1. On the other hand, note that the requested output port may be connected to an end-node or to another switch.

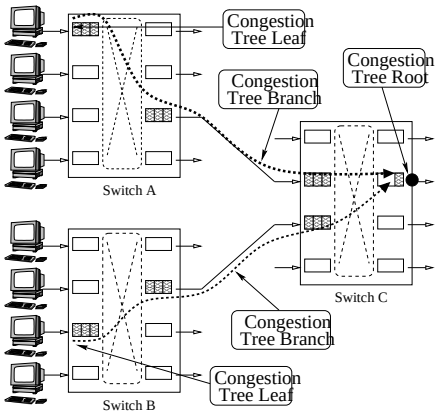
When contention persists over time, congestion appears. In these situations, the buffers containing the blocked packets will be filled and the flow control, in lossless networks, will prevent other switches from sending packets to the congested ports. Although flow control is essential to avoid discarding packets, it will rapidly propagate congestion to other switches, as packets stored at some of their ports will be also blocked. Figure 5.1 (b) shows a congestion situation whose origin is the contention situation shown in Figure 5.1 (a). In this example, it is assumed that the network is a lossless one (so, packets are not discarded when buffers are full), and also that the sources of the contending flows continuously inject packets into the network. These flows contributing to congestion are usually referred to as "hot flows".

In this way, congestion may progressively spread through the network, even reaching the end-nodes injecting hot flow packets. All the network resources affected by the spreading of congestion are commonly known as a "congestion tree". In a congestion tree, the "root" is the point where the hot flows causing congestion finally meet, the "branches" are series of consecutive congested points along any path followed by hot flows, and the "leaves" are the ending points of each branch. Figure 5.1 (c) shows the final congestion tree that is formed from the congestion situation shown in the previous examples.

However, the congestion tree shown in Figure 5.1 (c) is a very simple example, since congestion trees may exhibit very complex dynamics. In fact, congestion trees may actually evolve in very different ways, depending on traffic patterns and switch architecture. For example, a congestion tree may grow from leaves to root and vice versa. Also, several congestion trees may grow independently



(a) Contention situation (b) Congestion situation



(c) Congestion tree

Figure 5.1: Contention and congestion in switches of an interconnection network, reprinted from [García et al. 06], with permission from the authors.

and later merge, and it is even possible that some trees completely overlap while being independent.

Congestion degrades dramatically network performance because it leads to blocked packets that prevent the advance of other packets stored in the same queue, even if they are requesting free resources further ahead. This phenomenon is known as HOL blocking, and this may cause that data flows not contributing to congestion advance at the same speed as congested flows, thereby degrading network performance. It has been reported that this problem may limit switch

throughput to 58% [Hluchyj and Karol 88]. In order to address this problem, implementing a congestion management technique is mandatory.

Although congestion in interconnection networks is a well-known phenomenon, proposals for managing congestion in modern high-speed networks have not been satisfactory until recently. Traditional, simple solutions are not suitable for modern interconnects. For instance, network overdimensioning is not currently feasible due to cost and power consumption constraints. On the other hand, more elaborated techniques that have been specifically proposed for solving the problems related to congestion have not been really efficient until very recently.

5.1.1. Solutions for congestion management

The simplest of the strategies that deal with congestion are the network overdimensioning and the dropping of packets in congestion situations. However, none of them are suitable for modern interconnection networks due respectively to the current network components cost and consumption, and to the lossless character of these networks.

Other more elaborated proposals try to avoid or eliminate congestion. These are known as proactive or reactive congestion management techniques. Proactive strategies are based on reserving network resources for each data transmission, requiring a traffic scheduling based on network status [Wang et al. 95, Yew et al. 87]. 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 [Vogels et al. 00, Thottetodi et al. 01, Kim et al. 98]. Unfortunately, these solutions are not quite efficient due to the delay between congestion detection and notification, which leads to slow reactions of these strategies.

Other strategies focus on eliminating or reducing the main negative effect of congestion: HOL blocking. In fact, an effective HOL blocking elimination would turn congestion harmless. Many HOL blocking elimination or reduction strategies have been proposed: Virtual Output Queues (VOQs) [Dally et al. 98], Dynamically Allocated Multiqueues (DAMQs) [Tamir and Frazier 92], congestion buffers [Smai and Thorelli 98], etc. Most of these techniques rely on allocating different buffers for packets with different destinations. By doing this, packets from different flows will be stored in different queues, avoiding, therefore, HOL blocking.

In general, traditional HOL blocking elimination/reduction techniques are scalable or efficient, but not scalable and efficient 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 [Anderson et al. 93]. So, this technique is scalable, but it does not eliminate completely HOL blocking (just the HOL blocking at switch level).

Recently, a new HOL blocking elimination technique has been proposed [Duato et al. 05]. This technique eliminates the HOL blocking in a scalable and efficient way, as we will see in the next section.

5.1.2. RECN (Regional Explicit Congestion Notification)

RECN (Regional Explicit Congestion Notification) [Duato et al. 05, García et al. 06] is a new congestion management strategy that focuses on eliminating the main negative effect of congestion: The HOL blocking. In order to achieve it, RECN detects congestion and dynamically allocates separate buffers for each congested flow, assuming that packets from non-congested flows can be mixed in the same buffer without producing significant HOL blocking. Therefore, maximum performance is achieved even in the presence of congestion trees.

Currently, the routing mechanism used is the only restriction for applying RECN on any network technology. This is because RECN requires the use of some source deterministic routing that makes possible to address a particular network point from any other point in the network. In fact, RECN has been designed for PCI Express Advanced Switching (AS) [Adv 05], a technology that uses source routing. AS packet headers include a turnpool made up of 31 bits, which contains all the turns (offset from the input port to the output port) for every switch in a route. Thus, a switch, by inspecting the appropriate turnpool bits, can know in advance if a packet that is coming through one of its input ports will pass through a particular network point.

In order to separate congested and non-congested flows, RECN requires additional resources both at input and output switch ports. In particular, besides the standard queue, at each port RECN uses a set of additional queues, called set-aside queues (SAQs). A port dynamically allocates a SAQ to store packets belonging to a specific congestion tree once the port detects, or is notified of, the tree's existence. When congestion clears, the port deallocates the corresponding SAQ, making it available to store packets belonging to other congestion trees. Each switch port stores packets belonging to noncongested flows in its standard queue. Thus, the normal queue and the set of SAQs share the data RAM at each port.

RECN uses a content addressable memory (CAM) to manage the SAQs. Each CAM line contains the control information associated with a specific SAQ. This information consists of a valid bit (V), indicating whether the SAQ is active; a turnpool and a bit mask, identifying the root of the congestion tree for which this SAQ is allocated; a blocked bit (B) for the Xon/Xoff flow control (where Xon and Xoff are the thresholds for resuming and stopping the incoming flow);

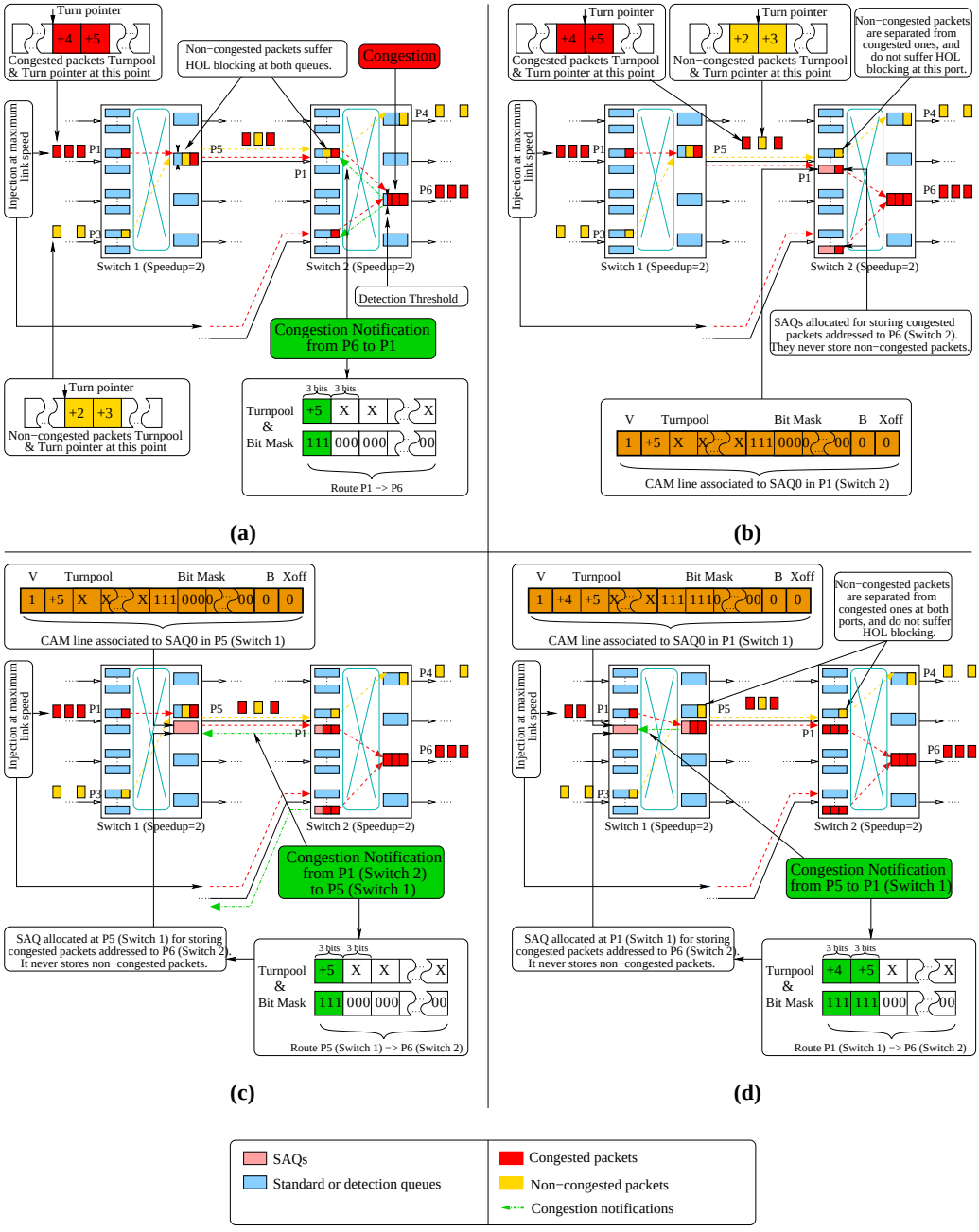


Figure 5.2: RECN congestion detection and allocation of SAQs, reprinted from [García et al. 06], with permission from the authors.

and a ready bit (R), which indicates whether the SAQ may send packets to the link controller. Only 65 bits of control information are required per SAQ: 62 bits for the turnpool and bit mask, and the 3 control bits.

Whenever an input or output standard queue receives a packet and fills over a given threshold, congestion is detected and a RECN notification is sent to the sender port indicating this situation (Figure 5.2 (a)). RECN notifications include the turnpool required to reach the congested point from the notified port. When a port receives a notification, it allocates a new SAQ and fills the corresponding CAM line with the turnpool received. From this moment, every packet received in this port will be stored in the newly allocated SAQ if it will cross the congested point associated to this SAQ (this can be deduced from the packet turnpool). As non-congested packets are stored in a different queue (the standard one), the HOL blocking that congested packets may cause is eliminated (Figure 5.2 (b)).

Furthermore, if any SAQ becomes congested, another notification will be sent upstream, and the receiving port should allocate a new SAQ (Figure 5.2 (c) and (d)). This procedure can be repeated until the notifications reach the congestion sources. Therefore, there will be SAQs for storing congested packets at every point where otherwise these packets could produce HOL blocking (so, at every branch of a congestion tree).

RECN also detects congestion vanishment at any point, in such a way that the SAQs associated to this point can be deallocated. Of course, deallocated SAQs can be re-allocated for newly detected and notified congested points. This allows RECN to eliminate HOL blocking while using a reduced number of SAQs. Further details about RECN can be found in [García et al. 06].

5.1.3. Congestion management and QoS

A high-performance interconnect has to offer many features in order to be effective. The core of this thesis focuses on the QoS provision, which is very important if there is a variety of applications sharing the network, and when it is necessary to offer guarantees on performance.

Another important feature of the network is congestion control. As we have discussed in the previous sections, a congestion control technique is necessary in order to achieve the best performance when the network works near the saturation point. This is a desirable situation, since it means that we are obtaining the best performance with the minimum resources, reducing cost and power consumption.

When the network designers want to combine different techniques in their network architecture, it may happen that resources are multiplied and the resulting design is completely unfeasible. For instance, if we want to support 16 VCs for different traffic classes and, at the same time, we want to provide VOQ

at the switch level, with 16 port switches, the resulting switch design requires 256 queues per switch port, which is not usually possible to implement.

Therefore, the motivation of this chapter is to obtain an integrated solution for the problems of congestion management and QoS support. We talk about integration because we want to use the same set of resources for both purposes. The proposal that we will present in the next section uses the same queues for congestion management and QoS support, thus being a very efficient solution.

5.1.4. Summary

In this section we have introduced the problem of congestion in high-performance interconnects. When the network is working near saturation point, HOL-blocking may appear. Traditional solutions are either effective but not scalable, or they scalable well but are not effective enough.

RECN is a novel proposal to solve this problem. It is based on set aside queues which are used to store congested flows. In this way, traffic flows which are not introducing congestion are not affected, and maximum performance is achieved.

Our purpose in this chapter is to integrate this proposal with our proposal for QoS provision presented in Chapter 3. In this case, as we will see in the following, we will not use any VCs.

5.2. Integrated QoS Support and Congestion Management

As we have already mentioned, our proposal tries to exploit the resources RECN uses for eliminating HOL blocking (as explained above), in such a way that QoS could also be provided without increasing queue requirements. The following subsections explain both the minimum changes it is necessary to introduce in the RECN architecture and the criteria that allow to use the modified architecture for providing both QoS and congestion management.

5.2.1. Proposed architecture

The organization of the switch that we propose consists in a combination of input and output buffering, which is a usual design for this kind of switches. Note that all the switch components are intended to be implemented in a single chip. This is necessary in order to offer the low cut-through latencies demanded by current applications.

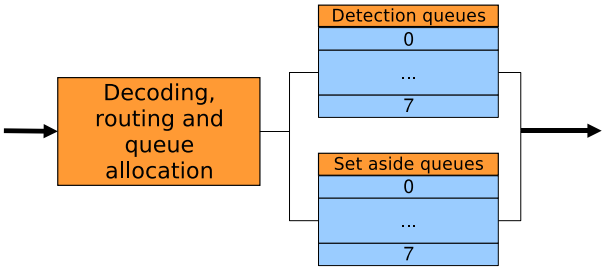


Figure 5.3: Logical input port organization.

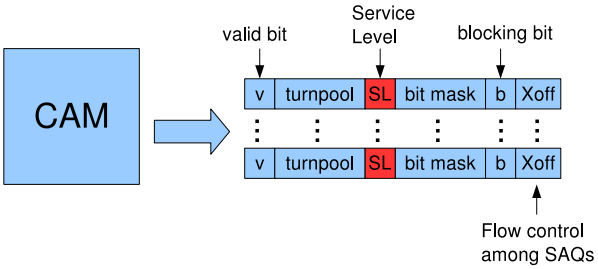


Figure 5.4: CAM organization.

The logical organization of an input port can be seen in Figure 5.3. This is the standard scheme for a RECN input port, so there are as many detection queues as output ports (8 in this case[1]) for storing non-congested packets, and a group of SAQs for storing congested packets. There are 8 of these SAQs since it has been shown that 8 or less SAQs are enough for eliminating HOL blocking almost completely [García et al. 06]. The use of these queues will be discussed later. A CAM is required at each input or output port in order to manage the set of SAQs. The organization of the CAM can be seen in Figure 5.4. Each CAM line contains all the fields defined by RECN, plus a new field for storing the service level the corresponding SAQ is assigned to.

Figure 5.5 shows the logical organization of output ports. In this case detection queues are not necessary, and a unique standard queue is used for storing non-congested packets. Following also the RECN scheme for output ports, a set of SAQs (8) is also used at each output port. In addition to this RECN queue structure, our proposal introduces at the output ports a set of bandwidth counters, one per service level. These counters must dynamically compute the difference between the reserved bandwidth and the current bandwidth consumption for each service level, and they will be used for two purposes: firstly, to per-

[1] For the sake of simplicity, we assume 8 port switches and 4 service levels. Anyway, architectures with a different number of ports or service levels could be easily deducted.

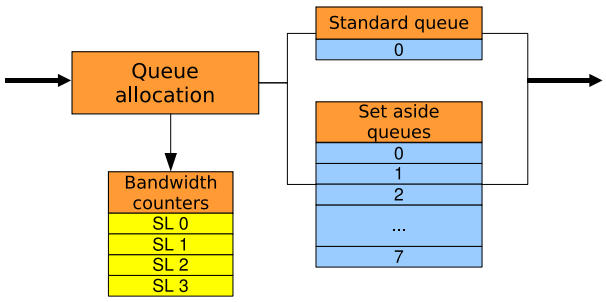


Figure 5.5: Logical output port organization.

form deficit round-robin scheduling at the switches; and secondly, for congestion detection (both issues will be detailed in the following subsections).

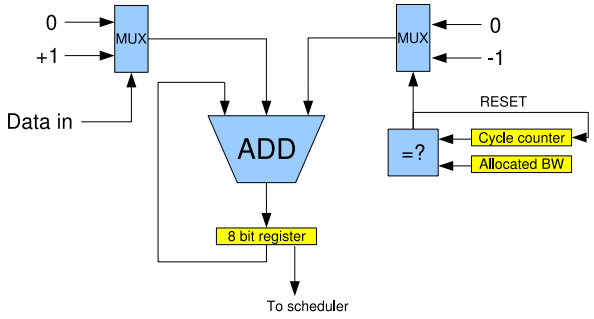


Figure 5.6: Bandwidth counter.

The bandwidth counters' structure can be seen in Figure 5.6. Basically, their behavior is the following: each time a block of 64 bytes from a packet[2] is scheduled to cross towards the output port, the bandwidth counter corresponding to the service level of the packet is increased by 1. We have assumed a 8-bit register for implementing the bandwidth counter, so its range of values is from -8 kbytes to 8 kbytes[3]. On the other hand, the same register must be automatically decremented at a rate that matches the bandwidth we guarantee to the corresponding service level. For instance, if 128 Mbytes/s have been guaranteed to the service level and the internal clock is 100 MHz, the bandwidth counter must be decreased by one every 50 cycles. This decrease is implemented by configuring the "Allocated BW" register with the appropriate number of cycles. When the cycle counter matches the configured cycles, the bandwidth counter register is decreased by one and the cycle counter is reset. All these operations

[2] The unit used in our counter is 64 bytes because in PCI AS each credit is 64 bytes and, thus, packets come in 64 bytes increments.
[3] These values are adequate to monitor instantaneous traffic behavior.

effectively allow to measure the difference between reserved and consumed bandwidth, while they are simple and do not introduce significant delay or require much silicon area.

5.2.1.1. Switch scheduler

The switch scheduler implements a deficit round-robin (DRR) algorithm based on the bandwidth counters' information. This algorithm consists in giving priority to flows that are consuming less bandwidth than the amount reserved for them. Following this scheduling, for each packet ready to cross the crossbar, the scheduler checks the value of the bandwidth counter corresponding to the output port and the service level of the packet. Afterwards, packets are served in the order of these registers, first packets with the smallest values and later packets with higher values.

It is possible to use a simplified version of this scheduler, in such a way that a threshold value is set, and the scheduler performs two rounds of scheduling: first, a round for packets with counter values lower than the threshold, and later another round with the unassigned outputs and the rest of the packets. In our tests, we used a threeshold of 4 kbytes, which worked fine.

This scheduler needs another feature compared to a typical one. Specifically, our scheduler requires to know whether a packet is going to be stored in an output port SAQ. This is known by comparing the turnpool of the packet with the information at the CAM. In the case of matching, packets can only be selected if the SAQ is not filled over a certain threshold. By applying this rule, buffer hogging is prevented.

Moreover, the scheduler does not treat specially packets coming from SAQs. However, if such packets are contributing to congestion, the bandwidth counter will have a high value and they will be penalized by the DRR algorithm. On the other hand, if the packets are no longer contributing to congestion, the bandwidth counter will have a small value and they will achieve a high priority. This contributes to empty unnecessary SAQs as soon as possible, allowing so the deallocation of these SAQs (SAQs must be empty for being dellocated).

5.2.1.2. Congestion management

RECN detects congestion both at input or output ports of switches, always by measuring the number of packets stored in the queues. Our new proposal also detects congestion at input and output ports, but in this case these detections do not depend only on queue occupancy. Specifically, in the new proposal, input detections happen when a detection queue in an input port fills over (in terms of stored information) a certain threshold and the value of the bandwidth counter associated to the packet service level at the requested output port is over another threshold. The actions to take after a congestion detection at the input port

are: first, a SAQ with turnpool equal to the output port and the appropriate service level is allocated in the input port, and, in addition to this, another SAQ, with empty turnpool but associated to the service level is allocated at the corresponding output port. These SAQs with empty turnpool (not considered in original RECN) will store any packet belonging to the associated service level. This is necessary in order to avoid that packets from a single service level completely fill an output buffer.

On the other hand, the conditions for a detection of congestion at an output port are similar. If a packet arrives from an input port, and causes the occupancy of the standard queue at this output port to be over a certain threshold; and the bandwidth counter of the service level of this packet is over another threshold; then the detection of congestion takes place. The actions after an output detection are the same as in an input detection: allocation of two SAQs, one at the input port and another with empty turnpool (but associated to the service level) at the output port.

If congestion persists and SAQs start to fill over a certain threeshold (known as the propagation level), then information about the corresponding turnpool and service level is propagated backwards the congestion flow, in order to allocate new SAQs for storing packets belonging to the flow wherever these packets are. In the case of propagation from a SAQ at an output port, SAQs are allocated at the input ports that cause the overflow of the output port SAQ. Of course, these input SAQs will have an associated turnpool with one more hop than the output SAQ. SAQs with empty turnpools (only allocated at output ports) may also produce the allocation of SAQs at the input ports, that in this case will have a one-hop turnpool.

If SAQs at input ports fill over the propagation level, a control packet is sent to the preceding switch. This packet includes the turnpool and service level associated to the filled input SAQ, in order to also have an allocated SAQ associated to this information at the receiving output port.

In this way, there will be SAQs at any point where they are necessary in order to store congested packets, thereby eliminating HOL blocking. SAQs can be deallocated when they are not necessary for eliminating HOL blocking at the point where they are allocated. The conditions for SAQ deallocation are exactly the same as in RECN [García et al. 06].

5.2.1.3. QoS Provision

In order to provide QoS guarantees, each traffic class is assigned a percentage of link bandwidth. For instance, if there are four traffic classes, each one could be assigned 25%. The total assigned bandwidth must not exceed the bandwidth of any link. At the end-points, we assume a traditional DRR implementation with a VC per traffic class, which is something feasible in these devices.

Provided that end-points implement a QoS policy, and as long as there is no contention, we have observed that packets pass through the switches in the same proportions as they are injected into the network. The reason is that the switches do not introduce any significant delay when links are not oversubscribed.

However, since there is no admission control, it may happen that any link of the network becomes oversubscribed. In this situation, congestion appears because at this point, one or more traffic classes introduce more traffic than their assignation.

A traditional congestion management technique would penalize all traffic regardless of its traffic class. From this point of view, all packets are equally contributing to congestion. However, with the bandwidth counters we have proposed and the switch scheduler we have presented before, only traffic classes injecting more than their allowance are treated by the congestion management technique. In this case, this is done by putting "guilty" packets in SAQs. Note that any traffic class can inject additional traffic if there is unused bandwidth. Therefore, problems only arise as a consequence of oversubscribed links.

Therefore, QoS guarantees are achieved in the sense that if traffic from a class is injected up to its allowed bandwidth, it will achieve maximum throughput and experiment short delay. The scheme proposed for QoS provision uses the same resources provided for congestion management. Therefore, we can offer a satisfactory solution for both problems, as we will confirm in the next section.

5.2.2. Summary

In this section we have presented an integrated solution for QoS support and congestion management. We use two novel, efficient, and cost-effective techniques both for provision of QoS and for congestion management, that are combined in a single switch architecture.

This switch architecture is very scalable because it uses a reduced set of resources, independently of network size. In the following section we will evaluate this design by simulation to confirm that it is also an efficient solution.

5.3. Performance Evaluation

In this section we will evaluate the performance of the proposal presented in this chapter. We will use the same simulator developed to evaluate the proposals of the previous chapters. In this sense, the configuration will be similar, but we will briefly review it here, remarking the differences with evaluation in previous chapters.

In most cases, we have supposed a workload of 4 service levels. Each one has been assigned a guaranteed bandwidth, but, for the sake of simplicity, in

many cases we assume that each traffic class has been guaranteed 25% of the link bandwidth.

We have considered a multi-stage interconnection network (MIN) with 64 end-points. We have chosen a MIN because it is a usual topology for computer clusters. However, our proposal is valid for any other network topology, including direct networks. Other assumptions, like link bandwidth or packet size, are based on the AS specifications [Adv 05]. Another assumption is the use of source routing, since it is needed by RECN.

We have run simulations for several architectures. In all cases, switches have 16 ports. We have considered the following architectures:

- *VOQ-sw 1 VC.* A simple switch design with VOQ at switch level, but without VCs. In this case, the queues per port are 16.
- *VOQ-sw 4 VC.* An architecture with a VC per traffic class and each VC is further divided in VOQs at the switch level. The total number of queues per switch port is 64.
- *VOQ-net 4 VC.* A switch design with classic VOQ at network level and also a VC per traffic class. This is a complex architecture, since it requires 256 queues per switch port.
- *RECN+QoS.* The switch architecture we have proposed in this chapter. Since it integrates QoS and congestion management with the same resources, the number of queues per port are 24: 16 detection queues plus 8 SAQs.

Total buffer size per port is 32 KBytes in all cases except when using VOQ at network level. In this case, queues would be so small using this amount of memory that performance is unacceptable. In this case, each port implements 8 MBytes of buffer, which is completely unfeasible with current technology. Nevertheless, it is a good reference for ideal performance.

5.3.1. Simulation results

In the following, we will see several tests. In them, we will show the advantages of the different architectures from two different points of view: congestion management and QoS support. However, validating RECN as a congestion management technique is out of the scope of this thesis. Consequently, we will just comfirm that our proposed integrated architecture still offers the expected performance regarding congestion.

Regarding QoS results, we have stated at the beginning of this chapter that the aim of this architecture is not to offer strict QoS guarantees, but a compromise from the network to offer good performance if traffic injection is inside

the configured bounds. Moreover, since all traffic shares the same VC, some additional delays are unavoidable when the congestion detection takes place. For instance, to detect that a traffic class is injecting too much, a buffer must be fill up to a certain level, which implies that packets using this buffer will be delayed until the SAQs start storing congested packets. Therefore, latency results will not be as good as if there were a dedicated VC for regulated traffic. Nevertheless, this architecture has the advantage of not requiring connection admission control (CAC) at all.

The scenarios we will study in the following are:

- Hot-spot scenario. We test the different alternatives with a static congestion traffic pattern.
- Incremental static scenario. This is similar to the previous scenario, but congestion is persistent in this case, and the load is incremental.
- Multimedia traffic scenario. In this case we use dynamic congestion. To achieve this, we transmit video sequences, which are very bursty.
- Multimedia traffic scenario with priorities. This scenario is similar to the previous one, but we vary the bandwidth assignation of each service level, to see how we provide QoS in the different switch architectures.
- Trace scenario. In the last scenario, we use traffic from traces of a real system.

Let us start with the first of them.

5.3.2. Hot-spot scenario

The objective of this scenario is to show that our proposal is able to identify and isolate congested traffic, just as the original RECN technique. In this way, traffic from service levels that are injecting less traffic than their reserved bandwidth should not be affected by this congestion. Moreover, congested traffic should get at least as much bandwidth as reserved.

For this scenario we have considered a multi-stage interconnection network (MIN) with 64 end-points. In this network, there is uniform traffic belonging to four service levels. However, during a small period of time, there is a sudden burst of traffic towards a hot-spot coming from a single service level. Without loss of generality, we will assume that this hot-spot is the node 5 and the service level 1. In this way, in addition to the uniform traffic, some of the interfaces inject traffic of service level 1 towards end-node 5.

In all the cases, congestion starts at 1000 μs and stops at 1300 μs. In this way, congestion is a sudden burst of packets towards a single destination. Moreover, schedulers are configured to assign the same bandwidth to all service levels (25%). We have considered four variations of this scenario, summarized in the Table 5.1.

	Uniform			Congested		
Number	Sources	Destination	Rate	Sources	Destination	Rate
1	87.5%	Random	50%	12.5%	5	100%
2	87.5%	Random	80%	12.5%	5	100%
3	75%	Random	50%	25%	5	100%
4	75%	Random	80%	25%	5	100%

Table 5.1: Video sequences for performance evaluation.

In Figure 5.7 we can see throughput and latency results for the four hot-spot scenarios. The best performance corresponds to ideal VOQ at the network level. However, the *RECN+QoS* proposal also achieves excellent performance in all cases. Note that latency is lightly increased for a short period while the congested traffic is allocated in SAQs. Afterwards, the latency of the uncongested service levels is not affected.

The *VOQ-sw 4VC* architecture offered lackluster performance. Even though the uncongested traffic classes are not affected, the global throughput results are not very good due to the poor management of congestion. On the other hand, performace is totally unacceptable when using the *VOQ-sw 1 VC* case.

From this test, we can conclude that our proposal is able to guarantee bandwidth of several traffic classes, even if there is another traffic class generating a hot-spot.

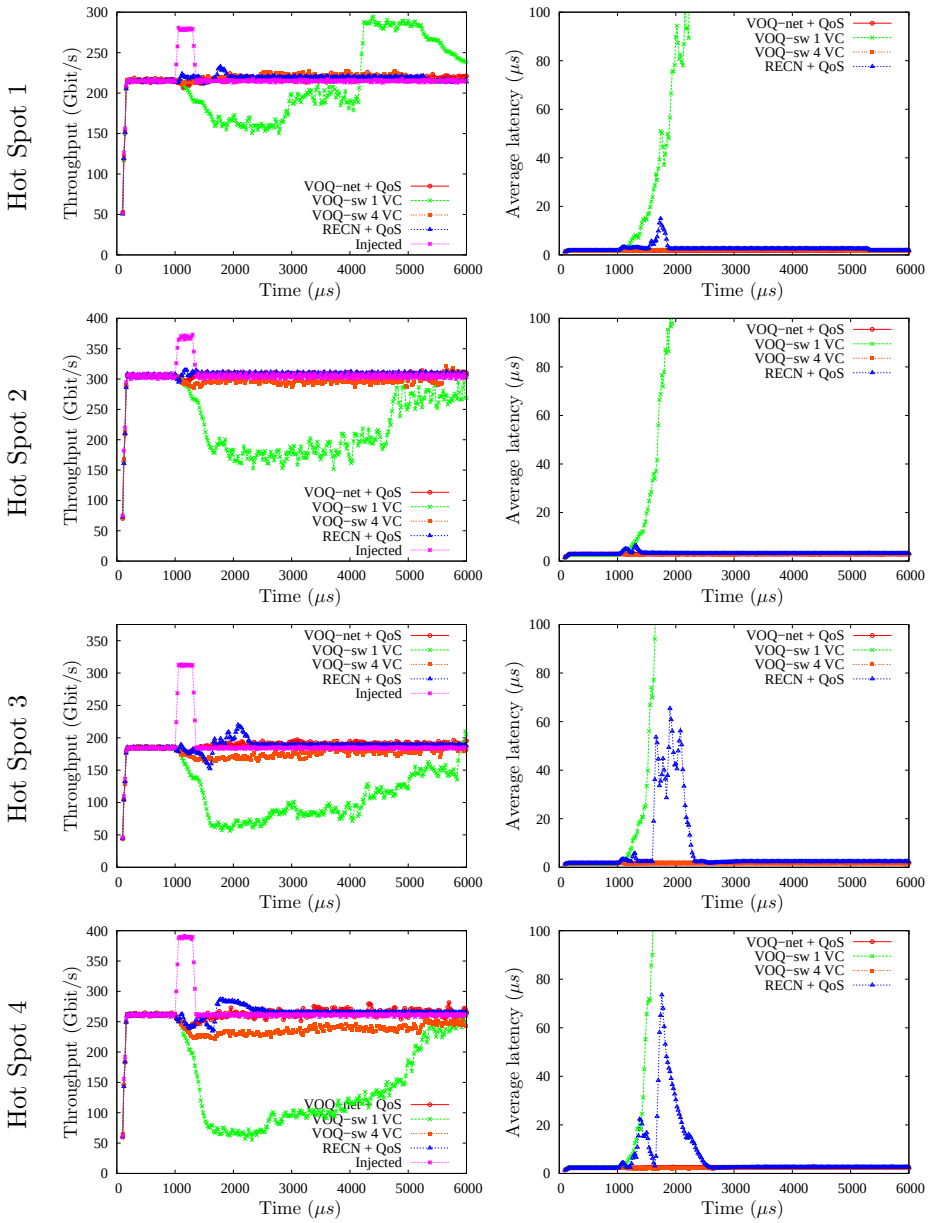


Figure 5.7: Results for hot-spot scenarios.

5.3.3. Incremental static scenario

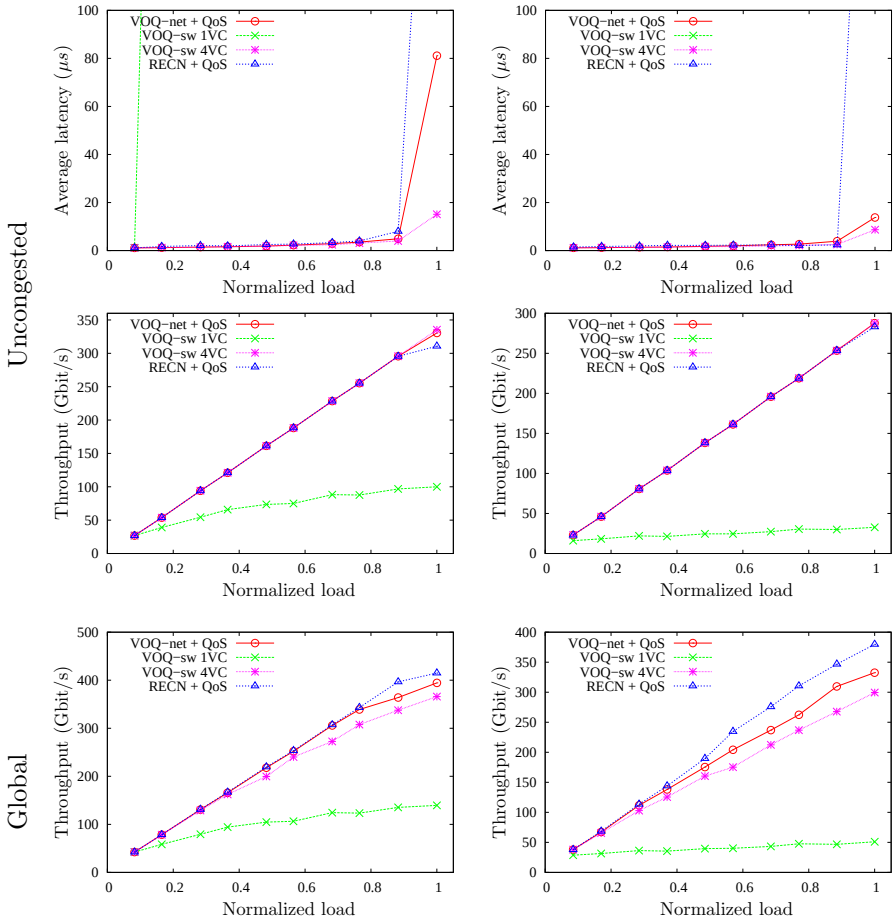
The purpose of this scenario is to show that the proposed architecture works even when congestion is not an instantaneous event, but a persistent condition in the network. In this way, we have two traffic scenarios, one with 12,5% of congested traffic and the other with 25% of congested traffic. In this way, these scenarios are similar to that of the previous test, but in this case, congested traffic is injected all along the simulation.

In the figures that we show for this scenario, the x axis shows traffic load, measured as the fraction of time that input links are busy and normalized to the $[0, 1]$ interval.

In the first column of Figure 5.8 we can see the results of the first scenario, with 12,5% of congested sources. Up to a load of 90%, performance of our proposal is the same as that of the much more complex architectures, with dedicated VCs for each traffic class. Once again, the basic switch architecture with VOQ at the switch level and one VC offers the worst performance.

The *VOQ-net 4VC* offers better global throughput than the *VOQ-sw 4VC* case, due to the more complex architecture. However, our proposal offers even better throughput. The reason is that our architecture can inject more traffic from the congested service level.

The results at the right column, corresponding to a 25% of congested sources, deserve similar comments to those of the first scenario. Therefore, we can conclude the our proposal can also identify congested traffic classes even when congestion is persistent.



(a) Scenario 1 (b) Scenario 2

Figure 5.8: Results of progressive static congestion test.

5.3.4. Multimedia traffic scenario

After studying the performance of the different architectures under static congestion, we look at results when congestion is a dynamic condition in the network. In real life, congestion rarely appears regarding a single hot-spot, with the rest of the traffic behaving properly. Rather, hot-spots appear and disappear quickly as bursts of packets are injected into the network. An excellent example of this is multimedia traffic like video sequences.

Video sequences are composed of a set of video frames that are generated at regular intervals. Compression algorithms produce frame patterns in which some frames are smaller than others. More specifically, there are intra-coded frames, which are basically normal pictures compressed with an algorithm like JPEG; besides, there are inter-coded frames, which only encode the differences with some neighbor frames. Therefore, frame size presents a lot of variability [Moving Picture Experts Group 94].

In this way, every 40 *ms* a long burst of packets is produced. Moreover, if many of these sources are multiplexed, they can quickly deplete buffers in the network and lead to a very poor performance.

In this scenario, all traffic will come from video traces of MPEG-4 sequences. Each sequence has an average throughput of 750 Kbytes/s, but bursts can be as large as 300 Kbytes. There are four classes of sequences, each with a guaranteed 25% of the throughput.

We can see at Figure 5.9 the performance of the different switch architectures we are considering. In the throughput results, we can see that our proposal offers a performance identical to the ideal architecture.

Regarding the VOQ switch architectures, the performance is very bad, with a peak throughput near 65%. Note that this is even the case of the 4 VC architecture, which has many more queues than our proposal.

In the figure we also show latency results, both in linear and logarithmic scale at the y axis. In this case, we can see that the best results are for our *RECN+QoS* architecture, even better than the *VOQ-net* case. The reason is that our *RECN+QoS* proposal is able to cope better with congestion at any point in the network, while VOQs are designed to handle congested end-nodes. As a consequence, large bursts of packets (like big video frames) progress faster, and hence our proposal obtains the best frame-level latency results.

We can conclude that our proposal is prepared to handle very difficult traffic, like multimedia traffic.

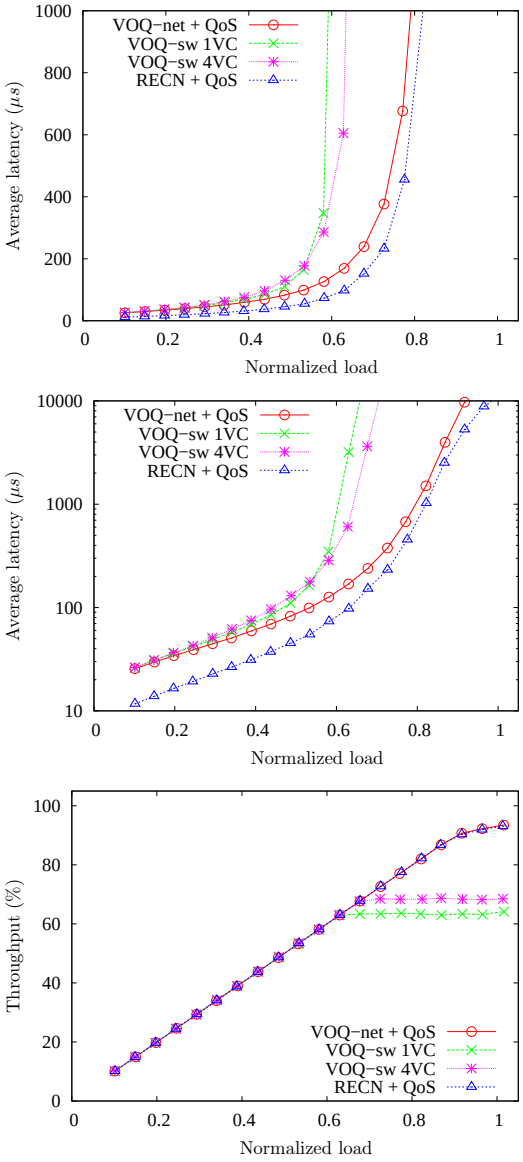


Figure 5.9: Results for multimedia traffic scenario.

5.3.5. Multimedia traffic scenario with priorities

Up to this point, we have shown that our proposal handles congestion properly. Moreover, it is able to isolate one traffic class from the others, in such a way that it can get a guaranteed minimum throughput at every link. In this scenario we vary weights of the traffic classes. This is done by configuring the weights in our switches and by assigning different number of table entries in traditional switches. As can be seen in the Table 5.2, we will use four different bandwidth assignations for the four traffic classes.

Traffic Class	Bandwidth	Percentage	Table Entries
Class 1	3 Gbit/s	37.5%	24
Class 2	2.5 Gbit/s	31.25%	20
Class 3	1.5 Gbit/s	18.75%	12
Class 4	1 Gbit/s	12.5%	8
Total	8 Gbit/s	100.0%	64

Table 5.2: Scheduler configuration.

In this test we will inject also multimedia traffic, just like in the previous test. In this way, in addition to varying priority, we also have a very bursty traffic to be handled by the switches.

Despite the bandwidth assignations, exactly the same amount of traffic is injected from every class (25%). In this way, we should see that the classes 4 and 3 are the first to get congested, while class 1 is not affected.

The performance results of the different architectures are shown in Figure 5.10. In the first column we have the latency results and in the second column we have the throughput results. Once again, our proposal offers performance identical to ideal architecture regarding throughput. We can also appreciate the differentiation of the four traffic classes. We can see that classes 1 and 2 obtain 100% throughput while classes 3 and 4 get congested earlier. Note that all these traffic classes share the unique VC.

Regarding the VOQ switch traffic classes, they obtain very poor performance, similar to that of the previous test. Note that even the *VOQ-sw 1VC* obtains some differentiation between the four traffic classes. This is because we are considering ideal network interfaces in all the cases, with as many queues as traffic classes times network destinations.

Finally, the latency results are as expected from throughput results. Note that, once again, the best latency performance corresponds to our proposed architecture.

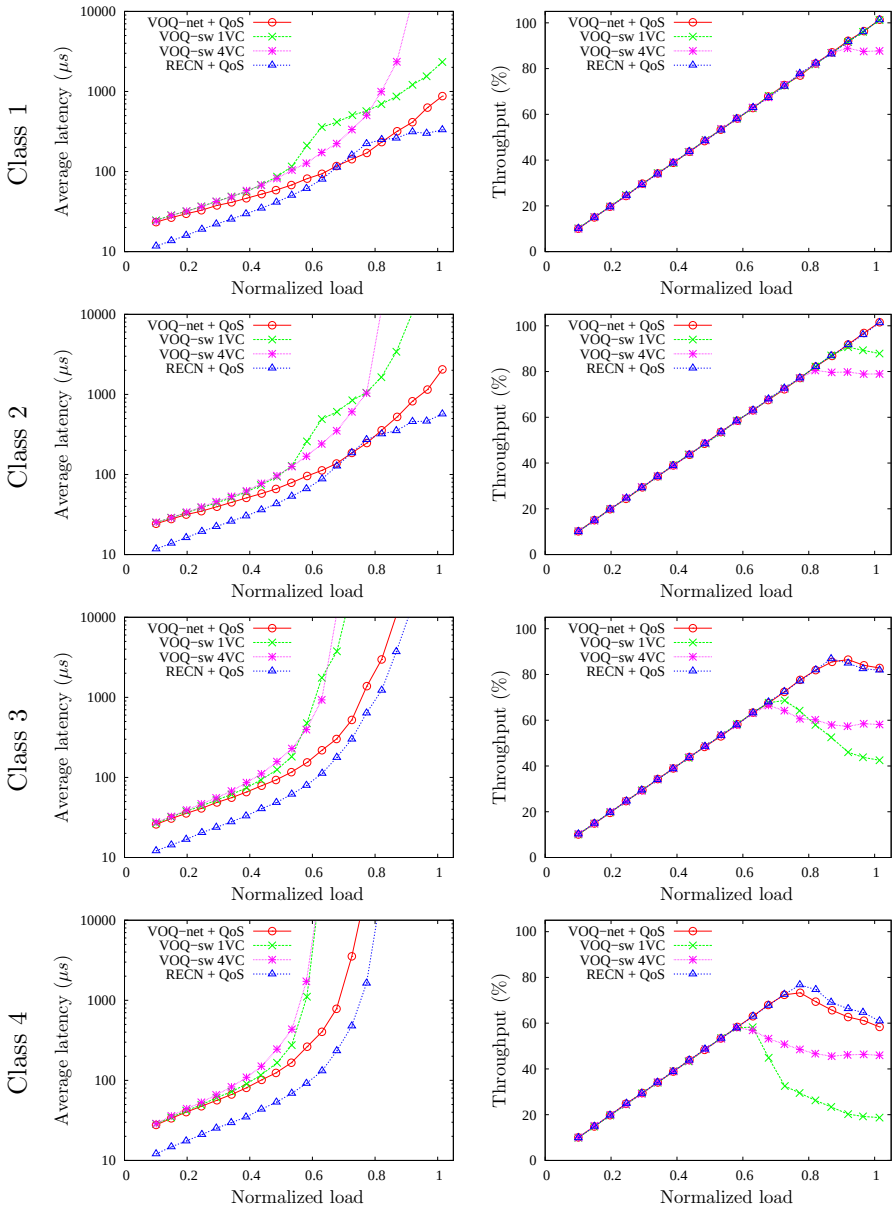


Figure 5.10: Results of multimedia traffic with different weights.

5.3.6. Trace scenario

The purpose of this scenario is to test the performance of our proposal using traffic from traces taken in a real network. This traffic is the same used in the previous chapter. Briefly, this traffic comes from I/O system from *Hewlett-Packard Labs*.

Since the traces are a bit old, a compression factor is applied in order to obtain a traffic load more appropriate for nowadays technology. More specifically, we have used $\times 32$ and $\times 40$ compression factors.

In order to simulate several service levels, we have distributed randomly the messages into four traffic classes. These traffic classes have a 25% bandwidth reservation each one.

The performance results can be found in Figure 5.11. In the first row, we see performance under $\times 32$ compression factor. We observe that both *VOQ-net* and *RECN-QoS* architectures offer the best performance in terms of latency and troughput results.

When we test the performance under a heavier load, this time a compression factor of $\times 40$, we can find similar results. In this way, we confirm that our proposal also works with traffic taken from real systems.

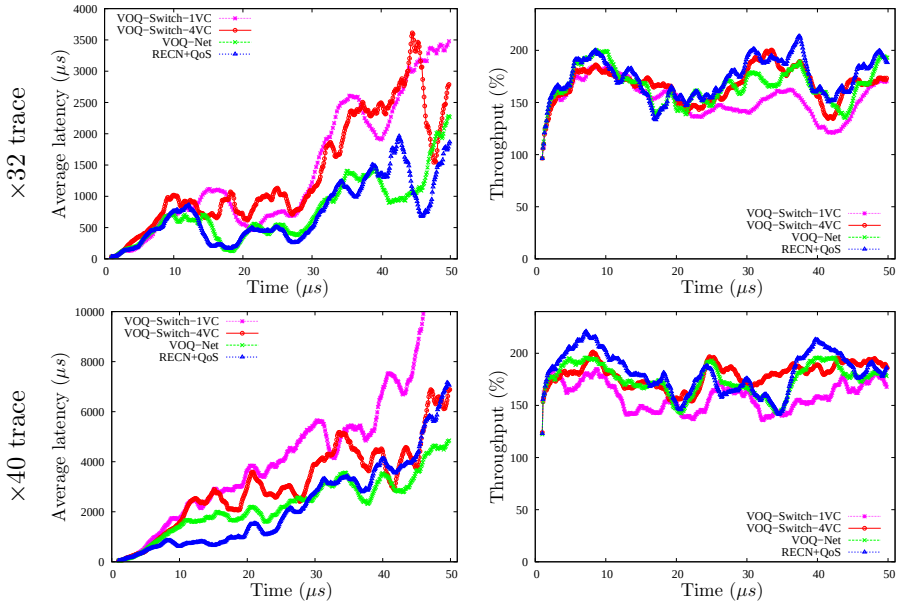


Figure 5.11: Results of trace scenario.

5.3.7. Summary

In this section we have tested our proposal for integrated QoS and congestion management. We have used a variety of scenarios, with different types of load, including static and dynamic hot-spots, multimedia traffic, and traces from a real system.

We have obtained excellent results in both QoS provision and congestion management. Regarding QoS, we have seen that our proposal can efficiently isolate the traffic of each service level, in such a way that each one can obtain its guaranteed throughput, regardless of the behavior of the other service levels. Moreover, latency results are better when using our proposal than when using a much more complex VOQ-net architecture.

Regarding congestion management, the performance was excellent when we injected static hot-spots. Moreover, this behavior continued when we introduced dynamic hot-spots and trace traffic.

This outstanding performance was achieved with a cost-effective architecture, which integrates the QoS and congestion management mechanisms in an efficient way.

5.4. Conclusions

Due to cost and power consumption constraints, current high-speed interconnection networks cannot be overdimensioned. Therefore, some solutions are needed in order to handle the problems related to high link utilization. In particular, both QoS support and congestion management techniques have become essential for achieving good network performance. However, most of the techniques proposed for both issues require too many resources for being implemented.

In this chapter we have proposed a new switch architecture able to face the challenges of congestion management and, at the same time, QoS provision, while being more cost-effective than other proposals, since it uses the same resources for both purposes.

According to the results presented in this chapter, we can conclude that our proposal can provide an adequate QoS while properly dealing with congestion. In our proposal we use advanced techniques for the buffer management, which allow a good performance under heavy and unbalanced load, while still providing appropriate QoS levels. Since all this is achieved with a reduced number of resources, this architecture would also reduce network cost.

CHAPTER 6

Conclusions and Future Work

IN this chapter we summarize and conclude this thesis. Firstly, we will point out the main contributions of our work, which are the most interesting conclusions. Afterwards, we show a list of the publications that have been derived from our work. Finally, we discuss which are the tasks that could be developed as future work.

6.1. Conclusions and Contributions

At the beginning of this work, we set a number of objectives. These were summarized in Section 1.4 in page 4 and now we review them and show in which degree they have been accomplished.

1. Study of previous work. During the development of this thesis, a deep understanding of the operation of high-performance interconnects and quality of service has been achieved. A summary of this study can be found in Chapter 2.

2. Develop a simulation tool to model high-performance networks. A tool that fits the requirements of this thesis was developed alongside with the researcher Raúl Martínez. This tool is a development of previous simulation tools used in the research group during many years.

3. Identify the sources of redundancy in QoS-provision proposals. We have shown in Chapter 3 that assigning a virtual channel per traffic class is

an important source of redundancy in the quality of service mechanism of interconnection networks.

4. Propose an efficient network architecture to provide QoS in high-speed interconnects. This was achieved also in Chapter 3. We proposed a number of techniques to provide full QoS support with just two VCs, for several types of scheduling.

5. Evaluate this proposal from the implementation point of view, considering how much silicon area, power consumption, and delay are introduced when using this proposal. This was achieved with the estimates provided in Section 3.3.

6. Evaluate this proposal from the performance point of view. We did a thorough evaluation of our proposals in sections 3.5, 3.6, and 3.7. We took into account three types of schedulers, several types of traffic, including traces from real systems and video sequences, and different switch architectures.

7. Propose an efficient QoS-provision at the individual flow level, instead of just considering a few traffic classes. In Chapter 4 we accomplished this objective. We proposed a new technique for efficiently implementing deadline-based algorithms in high-performance networks, we evaluate it, and we even proposed some improvements that made its performance very close to that of ideal systems.

8. Propose an integrated framework of QoS-provision and congestion management. The last part of the thesis, in Chapter 5, addressed this objective. We also performed an exhaustive performance evaluation and showed promising results.

Therefore, we consider that all the objectives initially proposed are satisfactory accomplished.

6.2. Publications

The different proposals, developments, and results compiled in this thesis have yield to several articles that have been published in journals or presented or published in proceedings of international conferences. In the following, we show all these publications and give a brief description of their main contributions.

We have organized the publications in the same three categories as this thesis. Inside each category, we use a chronological ordering.

6.2.1. Efficient traffic class-level QoS support

Providing Full QoS Support in Clusters Using Only Two VCs at the Switches

- Alejandro Martínez, Francisco J. Alfaro, José L. Sánchez, and José Duato.
- Publication: Lectures Notes in Computer Science
- Book: Proceedings of 12th IEEE International Conference on High Performance Computing - HiPC
- Publisher: Springer-Verlang
- I.S.B.N.: 3-540-30936-5 I.S.S.N.: 0302-9743
- Volume: 3769, Pages: initial: 158 final: 169
- December 2005
- Impact: 0.402 (JCR 2005) Acceptance rate: 50/271 = 18.5%

In this paper we first propose how to use 2 VC switches to efficiently provide QoS at the traffic class level. This was done with strict priority schedulers in the end-nodes.

Scalable Low-cost QoS Support for Single-chip Switches

- Alejandro Martínez, Francisco J. Alfaro, José L. Sánchez, and José Duato.
- Publication: International conference
- Conference: 12th International Conference on Parallel and Distributed Systems (ICPADS)
- Publisher: IEEE Computer Society Press
- I.S.B.N.: 978-0-76952612-6
- July 2006
- Acceptance rate: 64/185 = 34.6%

In this paper we do a scalability test of the first proposal. In this way, we conclude that it is suitable for network sizes ranging from 32 to 512 nodes.

Full QoS Support with 2 VCs for Single-chip Switches

- Alejandro Martínez, Francisco J. Alfaro, José L. Sánchez, and José Duato.
- Publication: International conference

- Conference: 5th IEEE International Symposium on Network Computing and Applications (IEEE NCA06)
- Publisher: IEEE Computer Society Press
- I.S.B.N.: 978-0-7695-2640-9
- July 2006
- Acceptance rate: 35%

In this paper we offer initial estimates of the constraints of a single-chip implementation of the switches we proposed. We also compare these results with the implementation of traditional switches.

Providing Full QoS with 2 VCs in High-speed Switches

- Alejandro Martínez, Francisco J. Alfaro, José L. Sánchez, and José Duato.
- Publication: International conference
- Conference: 21st International Conference on Information Networking (ICOIN 2007)
- Publisher: Proceedings of ICOIN 2007 (LNCS approval pending)
- January 2007
- Acceptance rate: 91/305 = 29.8%

This paper expands the previous one with more details on delay, power consumption, and silicon area of the switch implementation. Also, the performance evaluation takes this information into account, leading to more realistic simulation.

Efficient Switches with QoS Support for Clusters

- Alejandro Martínez, Francisco J. Alfaro, José L. Sánchez, and José Duato.
- Publication: International conference
- Conference: Workshop on Communication Architecture for Clusters (CAC 2007)
- Publisher: Proceedings of IPDPS 2007
- I.S.B.N.: 978-1-4244-0909-9
- March 2007
- Acceptance rate: 10/31 = 32.25%

In this paper we discuss how to efficiently emulate the bandwidth-based schedulers. Moreover, we show the increased efficiency of our proposals due to a better buffer management.

A Low-Cost Strategy to Provide Full QoS Support in Advanced Switching Networks

- Alejandro Martínez, Raúl Martínez, Francisco J. Alfaro, and José L. Sánchez.
- Publication: International journal
- Journal: Journal of Systems Architecture
- Volume: 53, Number: 7, Pages: initial: 339 final: 464
- Editorial: Elsevier Science
- I.S.S.N.: 1383-7621
- July 2007
- Impact: 0.402 (JCR 2005)

In this journal article, we take the proposal made in HiPC 2005 and adapt it for PCI AS. In this way, we discuss how would we build a switch that benefits from our proposal but still is compliant with the specifications.

A New Cost-effective Technique for QoS Support in Clusters

- Alejandro Martínez, Francisco J. Alfaro, José L. Sánchez, Francisco J. Quiles, and José Duato.
- Publication: International journal
- Journal: IEEE Transactions on Parallel and Distributed Systems (TPDS)
- To appear in second half of 2007
- Impact: 1.462 (JCR 2005)

This paper summarizes all our work in the topic of efficient QoS provision at the traffic class level.

6.2.2. Efficient flow-level QoS support

QoS Support for Video Transmission in High-speed Interconnects

- Alejandro Martínez, Georgios Apostolopoulos, Francisco J. Alfaro, José L. Sánchez, and José Duato.

- Publication: Lectures Notes in Computer Science
- Book: Proceedings of High Performance Computing and Communications Conference - HPCC
- Publisher: Springer-Verlang
- I.S.B.N.: 3-540-39368-4 I.S.S.N.: 0302-9743
- Volume: 4208, Pages: initial: 631 final: 641
- October 2006
- Impact: 0.402 (JCR 2005) Acceptance rate: 95/328 = 29.0%

This paper discusses an initial design for efficient deadline-based QoS, focused on the transmission of video sequences.

Deadline-based QoS Algorithms for High-performance Networks

- Alejandro Martínez, Francisco J. Alfaro, José L. Sánchez, and José Duato.
- Publication: International conference
- Conference: 21st IEEE International Parallel & Distributed Processing Symposium
- Publisher: Proceedings of IPDPS 2007
- I.S.B.N.: 978-1-4244-0909-9
- March 2007
- Acceptance rate: 109/419 = 26.0%

In this paper we offer a general and efficient solution for deadline-based QoS in high-performance switches. In addition to that, we also include an improvement over the initial idea, which consists in using a take-over queue.

Efficient Deadline-based QoS Algorithms for High-performance Networks

- Alejandro Martínez, Georgios Apostolopoulos, Francisco J. Alfaro, José L. Sánchez, and José Duato.
- Publication: International journal
- Journal: IEEE Transactions on Computers
- Under review

This paper summarizes our research in efficient deadline-based QoS for high-performance networks.

6.2.3. Integrated QoS support and congestion management

A Cost-Effective Interconnection Network Architecture with QoS and Congestion Management Support

- Alejandro Martínez, Pedro J. García, Francisco J. Alfaro, José L. Sánchez, José Flich, Francisco J. Quiles, and José Duato.
- Publication: Lectures Notes in Computer Science
- Book: Proceedings of EuroPar 2006 Conference
- Publisher: Springer-Verlang
- I.S.B.N.: 3-540-37783-2 I.S.S.N.: 0302-9743
- Volume: 4128, Pages: initial: 884 final: 895
- August 2006
- Impact: 0.402 (JCR 2005) Acceptance rate: 109/328 = 33.2%

This paper shows our initial attempt to integrate QoS provision and congestion management. In this paper we propose 2 VC switches that solve efficiently both problems.

Integrated QoS Provision and Congestion Management for Interconnection Networks

- Alejandro Martínez, Pedro J. García, Francisco J. Alfaro, José L. Sánchez, José Flich, Francisco J. Quiles, and José Duato.
- Publication: Lectures Notes in Computer Science
- Book: Proceedings of EuroPar 2007 Conference
- Publisher: Springer-Verlang
- To be presented in August 2007
- Impact: 0.402 (JCR 2005) Acceptance rate: 89/333 = 26.7%

In this paper we propose an enhanced proposal of integrated QoS support and congestion management. This only need one VC and has been thoroughly discussed in this thesis.

6.3. Future Work

The work that we have presented in this thesis can be expanded in several ways. In the following, we present the research lines that could be followed in the future:

- **Connection admission control.** A key component in many of the proposals presented in this thesis, specially in chapters 3 and 4, is the connection admission control (CAC). We have assumed a simple strategy based on average bandwidth and route utilization.

  With the CAC we have implemented for QoS-requiring traffic, each connection is assigned a path where enough bandwidth is assured. The CAC guarantees that less than 70% of bandwidth is used by QoS traffic at any link. The other 30% of available bandwidth will be used by non-regulated traffic. We also use a load-balancing mechanism when a QoS connection is stablished, which consists in assigning the least occupied route among the possible paths.

  There are more sophisticated alternatives for this simple scheme. Moreover, we could also research a new proposal specially aimed for our new switch architectures, that would take into account their peculiarities.

- **Traffic adaptation.** In the performance evaluation sections, we have assumed generally accepted traffic models for input traffic. However, we could focus into more specific environments to study which traffic classes are there and how to map them into the QoS mechanisms.

  - In symmetrical multi-processors (SMPs) there is a shared memory architecture. In this environment we have, in addition to memory-processor and I/O, cache coherency messages and synchronization messages. In this way, there are several types of communication that have different requirements and effects on final performance.
  - In multimedia communications we have several kinds of traffic, which have been modeled in this thesis. However, we have not gone into the details of specific video and audio standards and how to map them into the service levels.

- **Traffic model of parallel applications.** In the performance evaluation we have used a traffic model that is generally accepted for interconnection network evaluation. In this model, each packet or message is independent of others. However, although this model is very convenient for performance evaluation, in real life there are dependencies between packets.

  In general, a parallel application generates a limited number of messages before stopping until it receives the answers. In this way, instead of having infinite queues of messages, the number of messages in flight is limited by the number of communicating applications.

  Even more important, the performance metric when dependencies of packets are taken into account is not latency of throughput, but the delay introduced in applications by the communications.

In order to simulate this kind of traffic, advanced tools are needed, like simulators driven by execution of real applications. We have taken the first steps to integrate the SIMICS/GEMS simulator with a network simulator in order to do this kind of evaluations.

- **Real implementation.** We have given estimates of the performance and constraints of a real implementation of a switch using the proposals in Chapter 3. However, a complete design would be necessary to find out the actual figures.

  An interesting research line would be to write the Verilog or VHDL code for the switch. Then we could offer a fair comparison of the different architectures depending on the number of VCs, ports, buffer sizes, scheduling algorithms, etc.

- **Improvement of multimedia traffic.** When we have modelled multimedia traffic we have used traces of video sequences and synthetic sources of audio traffic. However, it could be very interesting to study how the delays introduced by the network affect the final quality of the signal received by the user.

  In addition to this, there are many proposals on how to efficiently map video sequences into network packets. In these proposals, there are some packets that are more important than others and differentiated QoS could be applied to them.

  Finally, when video sequences have to be broadcasted to many users, there are special algorithms that are used to distribute n sequences in such a way that a minimum of bandwidth is used and, at the same time, the receivers have the maximum flexibility to choose which sequence to see at any moment in time. These proposals take advantage of multicast traffic, which has not been implemented in this thesis.

- **On-chip networks.** Interconnection networks are necessary to produce scalable systems on chip. This kind of networks presents a number of challenges like synchronization, high error rates, low power consumption, short delays, etc. Moreover, QoS is also important to guarantee performance when bandwidth is limited.

  Depending on the kind of system built, we can expect different kinds of traffic. Moreover, some of these traffic classes will be more important than others and will have different QoS necessities. However, simple designs are necessary since silicon and power devoted to communications should be limited. Hence, we believe that our efficient proposals are very interesting in this environment.

  On the other hand, most proposals for on-chip networks use wormhole switching. Perhaps the most important challenge to adapt our proposal to on-chip networks is the use of this switching technique.

- **Different switch architectures.** In Chapter 2 we introduced several switch organizations. In all this thesis we have used a combined input and output buffering organization, which is very well known and widely used. However, we would like to see how to adapt our proposals to different switch architectures.

  More specifically, the buffer crossbar architecture is attracting a lot of attention recently. With the improvements in VLSI technology, to have a matrix of many little buffers is becoming feasible and it could lead to very efficient switches. It could be very interesting to study how to provide QoS in such a switch, where buffer size is very constrained.

Bibliography

[Adv 03] Advanced Switching Interconnect Special Interest Group. *Advanced Switching Core Architecture Specification. Revision 1.0*, Dec. 2003.

[Adv 05] Advanced Switching Interconnect Special Interest Group. *Advanced Switching Core Architecture Specification. Revision 1.1*, March 2005.

[Alfaro et al. 04] F.J. Alfaro, J.L. Sánchez, J. Duato: QoS in InfiniBand subnetworks. *IEEE Transactions on Parallel Distributed Systems*, Vol. 15, No. 9, pp. 810–823, Sept. 2004.

[Anderson et al. 92] T.E. Anderson, S.S. Owicki, J.B. Saxe, C.P. Thacker: High speed switch scheduling for local area networks. In *ASPLOS-V: Proceedings of the fifth international conference on Architectural support for programming languages and operating systems*, pp. 98–110. ACM Press, 1992.

[Anderson et al. 93] T. Anderson, S. Owicki, J. Saxe, C. Thacker: High-speed switch scheduling for local-area networks. *ACM Transactions on Computer Systems*, Vol. 11, No. 4, pp. 319–352, November 1993.

[ANSI 93] ANSI. Fibre channel physical and signaling interface, Rev. 4.2. Technical report, X3T9.3 Task Group, Oct. 1993.

[ASI] ASI SIG. *Advanced Switching Interconnect Special Interest Group.* `http://www.asi-sig.org`.

[ATM 95] ATM Forum. *ATM Forum traffic management specification. Version 4.0*, May 1995.

[Bennett and Zhang 96] J.C.R. Bennett, H. Zhang: WF^2Q: Worst-case fair weighted fair queueing. In *INFOCOM*, pp. 120–128, 1996.

[Bernet 98] Y. Bernet. A Framework for Differentiated Services. Internet draft 2275, Internet Engineering Task Force, May 1998.

[Black 00] U. Black: *QoS in Wide Area Networks.* Prentice Hall Series in Advanced Communications Technologies. Prentice Hall, 2000.

[Blake et al. 98] S. Blake, D. Back, M. Carlson, E. Davies, Z. Wang, W. Weiss. An Architecture for Differentiated Services. Internet Request for Comment RFC 2475, Internet Engineering Task Force, Dec. 1998.

[Braden et al. 94] R. Braden, D. Clark, S. Shenker. Integrated Services in the Internet Architecture: an Overview. Internet Request for Comment RFC 1633, Internet Engineering Task Force, June 1994.

[Braden et al. 97] R. Braden, L. Zhang, S. Berson, S. Herzog, S. Jamin. Resource ReSerVation Protocol (RVP) – version 1 functional specification. Internet Request for Comment RFC 2205, Internet Engineering Task Force, Sept. 1997.

[Breslau et al. 99] L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker: Web caching and zipf-like distributions: Evidence and implications. In *INFOCOM (1)*, pp. 126–134, 1999.

[Bull et al. 99] D. Bull, N. Conagarajah, A. Nix: *Insights into Mobile Multimedia Communications*. Academic Press, 1999.

[Chalmers and Sloman 99] D. Chalmers, M. Sloman: A survey of quality of service in mobile computing environments. *IEEE Communications Surveys and Tutorials*, Vol. 2, No. 2, 1999.

[Chuang et al. 99] S.T. Chuang, A. Goel, N.W. McKeown, B. Prabhakar: Matching output queueing with a combined input output queued switch. In *INFOCOM (3)*, pp. 1169–1178, 1999.

[Dai and Prabhakar 00] J.G. Dai, B. Prabhakar: The throughput of data switches with and without speedup. In *INFOCOM*, pp. 556–564, 2000.

[Dally 99] W.J. Dally. Scalable switching fabrics for internet routers. White paper, Avici Systems Inc., 1999. `http://www.avici.com/technology/whitepapers/TSRfabricWhitePaper.pdf`.

[Dally and Towles 03] W.J. Dally, B. Towles: *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., 2003.

[Dally et al. 94] W.J. Dally, L. Dennison, D. Harris, K. Kan, T. Xanthopoulos: Architecture and implementation of the Reliable Router. In *Hot Interconnects II*, 1994.

[Dally et al. 98] W.J. Dally, P. Carvey, L. Dennison: Architecture of the Avici terabit switch/router. In *Proceedings of the 6th Symposium on Hot Interconnects*, pp. 41–50, 1998.

[Demers et al. 90] A. Demers, S. Keshav, S. Shenker: Analysis and simulation of a fair queueing algorithm. *Journal of Internetworking Research and Experience*, pp. 3–26, Oct. 1990.

[Duato and López 94] J. Duato, P. López: Performance evaluation of adaptive routing algorithms for k-ary-n-cubes. In *PCRCW '94: Proceedings of the First International Workshop on Parallel Computer Routing and Communication*, pp. 45–59, London, UK, 1994. Springer-Verlag.

[Duato et al. 99] J. Duato, S. Yalamanchili, M.B. Caminero, D. Love, F.J. Quiles: MMR: A high-performance multimedia router. Architecture and design trade-offs. In *Proceedings of the 11th Symposium on High Performance Computer Architecture (HPCA)*, Jan. 1999.

[Duato et al. 02] J. Duato, S. Yalamanchili, L. Ni: *Interconnection networks. An engineering approach.* Morgan Kaufmann Publishers Inc., 2002.

[Duato et al. 05] J. Duato, I. Johnson, J. Flich, F. Naven, P.J. García, T. Nachiondo: A new scalable and cost-effective congestion management strategy for lossless multistage interconnection networks. In *Proceedings of the 11th Symposium on High Performance Computer Architecture (HPCA)*, 2005.

[El-Gendy et al. 03] M.A. El-Gendy, A. Bose, K.G. Shin: Evolution of the internet QoS and support for soft real-time applications. *Proceedings of the IEEE*, Vol. 91, No. 7, pp. 1086–1104, July 2003.

[Elhanany et al. 05] I. Elhanany, D. Chiou, V. Tabatabaee, R. Noro, A. Poursepanj: The network processing forum switch fabric benchmark specifications: An overview. *IEEE Network*, pp. 5–9, March 2005.

[Ferguson and Huston 98] P. Ferguson, G. Huston: *Quality of Service: delivering QoS on the Internet and in corporate networks.* John Wiley & Sons, Inc., 1998.

[García et al. 06] P.J. García, J. Flich, J. Duato, I. Johnson, F.J. Quiles, F. Naven: Efficient, scalable congestion management for interconnection networks. *IEEE Micro*, Vol. 26, 2006, No. 5, pp. 52–66, September 2006.

[Georgiadis et al. 94] L. Georgiadis, R. Guerin, A.K. Parekh: Optimal multiplexing on a single link: Delay and buffer requirements. In *INFOCOM (2)*, pp. 524–532, 1994.

[Giroux and Ganti 99] N. Giroux, S. Ganti: *Quality of Service in ATM Networks.* Prentice Hall, 1999.

[Golestani 94] S.J. Golestani: A self-clocked fair queueing scheme for broadband applications. In *In Proceedings of IEEE INFOCOM*, pp. 636–646, 1994.

[Goyal et al. 96] P. Goyal, H.M. Vin, H. Chen: Start-time fair queueing: a scheduling algorithm for integrated services packet switching networks. In *SIGCOMM '96: Conference proceedings on Applications, technologies, architectures, and protocols for computer communications*, pp. 157–168, New York, NY, USA, 1996. ACM Press.

[Guerin and Peris 99] R. Guerin, V. Peris: Quality-of-service in packet networks: basic mechanisms and directions. *Comput. Networks*, Vol. 31, No. 3, pp. 169–189, 1999.

[Hahne and Gallager 86] E.L. Hahne, R.G. Gallager: Round robin scheduling for fair flow control in data communication networks. In *ICC*, pp. 103–107, 1986.

[Halsall 01] F. Halsall: *Multimedia Communications: Applications, Networks, Protocols and Standard*. Addison-Wesley, 2001.

[Hluchyj and Karol 88] M.G. Hluchyj, M.J. Karol: Queueing in high-performance packet switching. *IEEE Journal on Sel. Areas in Commun.*, Vol. 6, No. 9, pp. 1587–1597, Dec. 1988.

[Huang et al. 02] C. Huang, J. Wang, Y. Huang: Design of high-performance CMOS priority encoders and incrementer/decrementers using multilevel lookahead and multilevel folding techniques. *IEEE Journal of Solid-State Circuits*, Vol. 1, No. 37, pp. 63–76, Jan. 2002.

[IEEE 04] IEEE. 802.1D-2004: Standard for local and metropolitan area networks. `http://grouper.ieee.org/groups/802/1/`, 2004.

[Iliadis 92] I. Iliadis: Performance of a packet switch with input and output queueing under unbalanced traffic. In *IEEE INFOCOM '92: Proceedings of the eleventh annual joint conference of the IEEE computer and communications societies on One world through communications (Vol. 2)*, pp. 743–752, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press.

[Inf 00] InfiniBand Trade Association. *InfiniBand architecture specification volume 1. Release 1.0*, Oct. 2000.

[Ioannou and Katevenis 01] A. Ioannou, M. Katevenis: Pipelined heap (priority queue) management for advanced scheduling in high speed networks. In *Proceedings of the IEEE International Conference on Communications (ICC'2001)*, 2001.

[Jain 91] R. Jain: *The art of computer system performance analysis: techniques for experimental design, measurement, simulation and modeling*. John Wiley and Sons, Inc., 1991.

[Jha and Hassan 02] S.K. Jha, M. Hassan: *Engineering Internet QoS*. Artech House, Inc., Norwood, MA, USA, 2002.

[Karol et al. 87] M.J. Karol, M.G. Hluchyj, S.P. Morgan: Input versus output queueing on a space-division packet switch. *IEEE Trans. on Commun.*, Vol. COM-35, pp. 1347–1356, 1987.

[Katevenis et al. 91] M. Katevenis, S. Sidiropoulos, C. Courcoubetis: Weighted round-robin cell multiplexing in a general-purpose ATM switch. *IEEE J. Select. Areas Commun.*, pp. 1265–1279, Oct. 1991.

[Katevenis et al. 97] M. Katevenis, P. Vatsolaki, D. Serpanos, E. Markatos: ATLAS I: A single-chip ATM switch for NOWs. In *In Proceedings of the Workshop on Communication and Architectural Support for Network-based Parallel Computing (CANPC 97),*, San Antonio, Texas, USA, 1997.

[Keshav 97] S. Keshav: *An engineering approach to computer networking: ATM networks, the Internet, and the telephone network*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[Kim et al. 98] J. Kim, Z. Liu, A. Chien: Compressionless routing: a framework for adaptive and fault-tolerant routing. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 8, No. 3, pp. 229–244, March 1998.

[Kleinrock 76] L. Kleinrock: *Queueing Systems, Volume 1: Computer Applications*. John Wiley & Sons, 1976.

[Kornaros et al. 98] G. Kornaros, D. Pnevmatikatos, P. Vatsolaki, G. Kalokerinos, C. Xanthaki, D. Mavroidis, D. Serpanos, M. Katevenis: Implementation of ATLAS I: a single-chip ATM switch with backpressure. In *Proceedings of the 6th Symposium on Hot Interconnects*, 1998.

[Marsan et al. 02] M.A. Marsan, A. Bianco, P. Giaccone, E. Leonardi, F. Neri: Packet-mode scheduling in input-queued cell-based switches. *IEEE/ACM Trans. Netw.*, Vol. 10, No. 5, pp. 666–678, 2002.

[McKeown 95] N.W. McKeown. *Scheduling algorithms for input-queued cell switches*. Ph.D. thesis, Berkeley, CA, USA, 1995.

[McKeown 99] N.W. McKeown: The iSLIP scheduling algorithm for input-queued switches. *IEEE/ACM Transactions on Networking*, Vol. 7, pp. 188–201, 1999.

[McKeown et al. 96] N.W. McKeown, V. Anantharam, J.C. Walrand: Achieving 100% throughput in an input-queued switch. In *INFOCOM*, pp. 296–302, 1996.

[Minkenberg et al. 03] C. Minkenberg, F. Abel, M. Gusat, R.P. Luijten, W. Denzel: Current issues in packet switch design. In *ACM SIGCOMM Computer Communication Review*, Vol. 33, pp. 119–124, Jan. 2003.

[Moving Picture Experts Group 94] Moving Picture Experts Group. Generic coding of moving pictures and associated audio. Rec. H.262. Draft Intl. Standard ISO/IEC 13818-2, 1994.

[Parekh and Gallager 93] A.K. Parekh, R.G. Gallager: A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Trans. Netw.*, Vol. 1, No. 3, pp. 344–357, 1993.

[Partridge 92] C. Partridge. A proposed flow specification. Internet Request for Comment RFC 1363, Internet Engineering Task Force, Sept. 1992.

[PCI 03] PCI Special Interest Group. *PCI Express Base Architecture Specification. Revision 1.0a*, April 2003.

[Peh and Dally 01] L.S. Peh, W.J. Dally: A delay model and speculative architecture for pipelined routers. In *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, 255. IEEE Computer Society, 2001.

[Pelissier 00] J. Pelissier: Providing Quality of Service over Infiniband Architecture Fabrics. In *Proceedings of the 8th Symposium on Hot Interconnects*, Aug. 2000.

[Rexford et al. 96] J. Rexford, A.G. Greenberg, F. Bonomi: Hardware-efficient fair queueing architectures for high-speed networks. In *INFOCOM*, pp. 638–646, 1996.

[Seifert 98] R. Seifert: *Gigabit Ethernet: Technology and Applications for High-Speed LANs*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998.

[Shreedhar and Varghese 96] M. Shreedhar, G. Varghese: Efficient fair queueing using deficit round robin. *IEEE/ACM Transactions on Networking*, Vol. 4, No. 3, pp. 375–385, June 1996.

[Simos 04] D. Simos. Design of a 32x32 variable-packet-size buffered crossbar switch chip. Technical Report FORTH-ICS/TR-339, Inst. of Computer Science, FORTH, July 2004.

[Smai and Thorelli 98] A. Smai, L. Thorelli: Global reactive congestion control in multicomputer networks. In *Proc. 5th Int. Conference on High Performance Computing*, 1998.

[Sutherland et al. 99] I. Sutherland, B. Sproull, D. Harris: *Logical effort: designing fast CMOS circuits*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[Tamir and Frazier 92] Y. Tamir, G. Frazier: Dynamically-allocated multi-queue buffers for vlsi communication switches. *IEEE Transactions on Computers*, Vol. 41, No. 6, June 1992.

[Thottetodi et al. 01] M. Thottetodi, A. Lebeck, S. Mukherjee: Self-tuned congestion control for multiprocessor networks. In *Proc. of 7th. Int. Symp. on High Performance Computer Architecture*, February 2001.

[Vid] Video Traces Research Group. *YUV Video Sequences*. `http://trace.eas.asu.edu/yuv/index.html`.

[Vogels et al. 00] W. Vogels, D. Follett, J. Hsieh, D. Lifka, D. Stern: Tree-saturation control in the AC3 velocity cluster interconnect. In *Proc. 8th Conference on Hot Interconnects*, August 2000.

[Wang 04] H.S. Wang. A detailed architectural-level power model for router buffers, crossbars and arbiters. Technical report, Department of Electrical Engineering, Princeton University, 2004.

[Wang et al. 95] M. Wang, H.J. Siegel, M.A. Nichols, S. Abraham: Using a multipath network for reducing the effects of hot spots. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 6, No. 3, pp. 252–268, March 1995.

[Wang et al. 02] H.S. Wang, L.S. Peh, S. Malik: A power model for routers: Modeling Alpha 21364 and InfiniBand routers. In *HOTI '02: Proceedings of the 10th Symposium on High Performance Interconnects HOT Interconnects (HotI'02)*, 21, Washington, DC, USA, 2002. IEEE Computer Society.

[Weste and Harris 05] N.H.E. Weste, D. Harris: *CMOS VLSI Design: A Circuits and Systems Perspective*. Addison-Wesley, 2005.

[Xiao and Ni 99] X. Xiao, L. Ni: Internet QoS: A Big Picture. *IEEE Network Magazine*, pp. 8–18, March 1999.

[Xil 06] Virtex-4 RocketIO multi-gigabit transceiver. User Guide UG076 (v2.0), Xilinx, Inc., 2006.

[Yew et al. 87] P. Yew, N. Tzeng, D.H. Lawrie: Distributing hot-spot addressing in large-scale multiprocessors. *IEEE Transactions on Computers*, Vol. 36, No. 4, pp. 388–395, April 1987.

[Younis et al. 01] A. Younis, C. Boecker, K. Hossain, F. Abughazaleh, B. Das, Y. Chen, M. Robinson, S. Irwin, B. Grung: A low jitter, low power, cmos 1.25-3.125gbps transceiver. In *Proceedings of the 27th European Solid-State Circuits Conference (ESSCIRC 2001)*, 2001.

[Yum et al. 01] K.H. Yum, E.J. Kim, C.R. Das: QoS provisioning in clusters: An investigation of router and NIC design. In *Proceedings of the 28th Annual International Symposium on Computer Architecture (ISCA)*. IEEE Computer Society, July 2001.

[Yum et al. 02] K.H. Yum, E.J. Kim, C.R. Das, A.S. Vaidya: MediaWorm: A QoS capable router architecture for clusters. *IEEE Transactions on Parallel Distributed Systems*, Vol. 13, No. 12, pp. 1261–1274, Dec. 2002.

[Zhang 91] L. Zhang: VirtualClock: A new traffic control algorithm for packet switched networks. *ACM Transactions on Computer Systems*, Vol. 9, 2, pp. 101–124, 1991.

[Zhang 95] H. Zhang: Service disciplines for guaranteed performance service in packet-switching networks. *Proceedings of IEEE*, Vol. 83, No. 10, pp. 1374–1396, Oct. 1995.

[Zipf 65] G.K. Zipf: *The Psycho-Biology of Languages.* Houghton-Miffin, MIT, 1965.