Raúl Martínez Moráis

PROVIDING QUALITY OF SERVICES IN SYSTEMS BASED ON ADVANCES SWITCHING

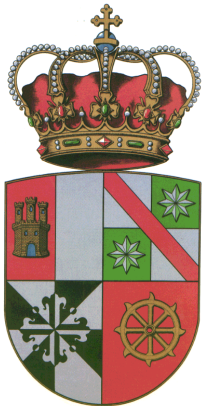
I.S.B.N. Ediciones de la UCLM
978-84-8427-641-8

Ediciones de la Universidad
de Castilla-La Mancha

Cuenca, 2008

UNIVERSIDAD DE CASTILLA-LA MANCHA

DEPARTAMENTO DE SISTEMAS INFORMÁTICOS



Providing Quality of Service in Systems Based on Advanced Switching

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:
Raúl Martínez Moráis

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

Albacete, Julio de 2007

Agradecimientos

Esta tesis es fruto de un trabajo que no hubiera sido posible sin el apoyo y ayuda de muchas personas. Quiero aquí dedicarles mi más sincero agradecimiento.

En primer lugar quiero darle las gracias a mis tutores, Paco y José Luis, por su guía, apoyo, paciencia y amistad durante estos años. Sin duda este trabajo no hubiera sido posible sin ellos.

Quiero agradecer también a la gente del Departamento de Sistemas Informáticos, especialmente al grupo de Redes y Arquitecturas de Altas Prestaciones (RAAP), su apoyo material y financiero, pero sobre todo humano.

No puedo olvidarme de mis compañeros becarios del RAAP y del Instituto de Investigación en Informática de Albacete (I3A), especialmente de Alejandro, por su compañerismo, apoyo y ayuda durante estos años.

Por último, quiero dar mi más cariñoso agradecimiento a mis padres, Emilio y Cristina. Esta tesis es también fruto de su esfuerzo, sacrificio y amor incondicional. Y también a mi familia y amigos, especialmente a Alfonso, por todo el apoyo y cariño que me han dado, a pesar de todo el tiempo que la tesis me ha impedido dedicarles. Gracias por compartir conmigo los buenos y malos momentos.

A todos gracias, sin vosotros no hubiera sido posible llegar hasta aquí.

Resumen

Advanced Switching (AS) es una tecnología de red basada en PCI Express. PCI Express es la nueva generación PCI, la cual está ya reemplazando el extensivamente usado bus PCI. AS es una extrapolación de PCI Express que toma prestadas sus dos capas arquitecónicas de más bajo nivel e incluye una capa de transacciones optimizada para permitir nuevas capacidades como la comunicación *peer-to-peer*. Mientras que PCI Express ya ha empezado a reformar una nueva generación de ordenadores personales y servidores tradicionales, una red de interconexión común con la industria de las comunicaciones parece lógico y necesario. Así pues, AS estaba pensado para proliferar en los entornos de multiprocesadores, sistemas *peer-to-peer* en las comunicaciones, almacenamiento, redes de interconexión, servidores y plataformas empotradas.

Por otro lado, la calidad de servicio (Quality of Service, QoS) se está convirtiendo en una característica importante para las redes de altas prestaciones. Proporcionar QoS en entornos de computación y comunicaciones es actualmente el centro de muchos esfuerzos de investigación por parte de la industria y en el ámbito académico. AS incorpora mecanismos que pueden ser usados para proporcionar QoS. En concreto, AS permite utilizar Canales Virtuales (Virtual Channels, VCs), arbitraje en los puertos de salida y un mecanismo de control de admisión. Además, AS proporciona un control de flujo a nivel de enlace y VC. Estos mecanismos nos permiten agregar el tráfico con características similares en un mismo VC y proporcionar a cada VC un tratamiento diferenciado en base a sus requisitos.

El objetivo principal de la tesis ha sido el estudio de los diferentes mecanismos de AS con el fin de proponer un marco general para proporcionar QoS a las aplicaciones sobre esta tecnología de red. En este sentido, el foco principal del trabajo, dada su importancia para proporcionar QoS, ha sido sido el estudio de los mecanismos de planificación de AS. Nuestro objetivo ha sido implementarlos de una manera eficiente, teniendo en cuenta tanto sus prestaciones como su complejidad. Para conseguir estos objetivos, hemos propuesto varias posibles implementaciones del planificador de mínimo ancho de banda de AS. Hemos propuesto modificar el planificador basado en tabla de AS con el objetivo de solucionar los problemas de éste para proporcionar requisitos de QoS con tamaños de paquete variable. Hemos también propuesto cómo configurar el planificador basado en tabla resultante para desacoplar las asignaciones de ancho de banda y latencia. Además, hemos llevado a cabo un diseño *hardware* de los diferentes planificadores para obtener estimaciones sobre el tiempo de arbitraje y el área de silicio que requieren. Además, hemos desarrollado nuestro propio simulador para evaluar las prestaciones de nuestras propuestas.

Summary

Advanced Switching (AS) is a network technology based on PCI Express. PCI Express is the next PCI generation, which is already replacing the extensively used PCI bus. AS is an extrapolation of PCI ExpressSummary that borrows its lower two architectural layers and includes an optimized transaction layer to enable new capabilities like peer-to-peer communication. Whereas PCI Express has already begun to reshape a new generation of PCs and traditional servers, a common interconnect with the communications industry seems logical and necessary. In this way, AS was intended to proliferate in multiprocessor, peer-to-peer systems in the communications, storage, networking, servers, and embedded platform environments.

On the other hand, Quality of Service (QoS) is becoming an important feature for high-performance networks. The provision of QoS in computing and communication environments is currently the focus of much discussion and research in industry and academia. AS provides mechanisms that can be used to support QoS. Specifically, an AS fabric permits us to employ Virtual Channels (VCs), egress link scheduling, and an admission control mechanism. Moreover, AS performs a link-level flow control in a per VC basis. These mechanisms allow us to aggregate traffic with similar characteristics in the same VC and to provide each VC with a different treatment according to its requirements.

The main objective of this thesis has been to study the different AS mechanisms in order to propose a general framework for providing QoS to the applications over this network technology. In this line, the main focus of this work, due to its importance for the QoS provision, is the study of the AS scheduling mechanisms. Our goal has been to implement them in an efficient way, taking into account both their performance and their complexity. In order to achieve these objectives, we have proposed several possible implementations for the AS minimum bandwidth egress link scheduler taking into account the link-level flow control. We have proposed to modify the AS table-based scheduler in order to solve its problems to provide QoS requirements with variable packet sizes. We have also proposed how to configure the resulting table-based scheduler to decouple the bandwidth and latency assignations. Moreover, we have performed a hardware design of the different schedulers in order to obtain estimates on the arbitration time and the silicon area that they require. We have also developed our own network simulator in order to evaluate the performance of our proposals.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter we introduce this thesis. Firstly, we discuss about the importance of the provision of Quality of Service (QoS) over high-performance networks, including a brief introduction to Advanced Switching (AS) and some of the mechanisms that this interconnection technology incorporates in its specification to provide QoS. Secondly, we motivate the importance of studying how to provide QoS over AS. Finally, we settle the objectives we want to accomplish and introduce the organization of the following chapters.

1.1 QoS in High-Performance Networks

The evolution of interconnection network technology has been constant along the previous decades. The speed and capacity of various components in a communication system, such as links, switches, memory, and processors, have increased dramatically. Moreover, network topologies have become more flexible, and the efficiency of switching, routing and flow control techniques have been improved.

The advent of high-speed networking has introduced opportunities for new applications. Current packet networks are required to carry not only traffic of applications, such as e-mail or file transfer, which does not require pre-specified service guarantees, but also traffic of other applications that requires different performance guarantees, like real-time video or telecommunications [MP01]. The best-effort service model, though suitable for the first type of applications, is not so for applications of the other type [Par05]. Even in the same application, different kinds of traffic (e.g. I/O requests, coherence control messages, synchronization and communication messages, etc.) can be considered, and it would be very interesting that they were treated according to their priority [CMR06].

This is the reason because the provision of QoS in computing and communication environments has been the focus of much discussion and research in academia during the last decades. This interest in academia has been renewed by the growing interest on this topic in industry during the last years. A sign of this growing interest in industry is the inclusion of mechanisms intended for providing QoS in some of the last network standards like Gigabit Ethernet [Sei98], InfiniBand [Inf00], or Advanced Switching (AS) [Adv03]. An interesting survey with the QoS capabilities of these network technologies can be found in [RSS06].

A key component for networks with QoS support is the output (or egress link) scheduling algorithm (also called service discipline)[DKS89], [GM92], [Zha95]. In a packet-switching network, packets from different flows will interact with each other at each switch. Without proper control, these interactions may adversely affect the network performance experienced by clients. The scheduling algorithm, which selects the next packet to be transmitted and decides when it should be transmitted, determines how packets from different flows interact with each other. Therefore, the scheduling algorithm plays an important role in providing the traffic differentiation that is necessary to provide QoS.

Apart from providing a good performance in terms of, for example, good end-to-end delay (also called latency) and fair bandwidth allocation, an ideal scheduling algorithm implemented in a high-performance network with QoS support should satisfy other property: To have a low computational and implementation complexity. This is because in order to achieve a good performance, the time required to select the next packet to be transmitted must be smaller than the average packet transmission time. This means that the scheduler computation time must be very small, if we consider the high speed of high-performance networks. Moreover, a low complexity is required in order to be able to implement the scheduler in a small silicon area (note that high-performance switches are usually implemented in a single chip).

During the last decades a vast amount of scheduling disciplines has been proposed in the literature for different purposes. In general, these algorithms have been proposed for lossy networks, like Internet or ATM, where packets are thrown away in the presence of congestion.

1.2 Advanced Switching

Advanced Switching Interconnect, or just Advanced Switching (AS) [Adv05], is an open-standard fabric-interconnect technology based on PCI Express [PCI03]. PCI Express is

already replacing the extensively used PCI bus. The PCI bus has served industry well for the last 10 years and is currently extensively used. However, the processors and I/O devices of today and tomorrow demand much higher I/O bandwidth than PCI 2.2 or PCI-X can deliver. The reason for this limited bandwidth is the parallel bus implementation. PCI Express eliminates the legacy shared bus-based architecture of PCI and introduces an improved and dedicated point-to-point interconnect. The primary strength behind PCI Express is in its support for legacy PCI while addressing its inadequacies.

AS is an extrapolation of PCI Express, borrowing its lower two architectural layers from PCI Express, and including an optimized transaction layer to enable essential communication capabilities like peer-to-peer communication. The need for AS essentially comes because computing and communication platforms begin to converge by exhibiting increasing overlap in terms of the functions they serve. Whereas PCI Express has already begun to reshape a new generation of PCs and traditional servers, a common interconnect with the communication industry seems logical and necessary, in order to keep development cost down, performance up and to reduce time-to-market. In this way, AS was intended to proliferate in multiprocessor, peer-to-peer systems in the communications, storage, networking, servers and embedded platform environments. Together, PCI Express and AS were thought to have the potential for building the next generation of interconnects [MK03].

AS provides some mechanisms, which correctly used permit us to provide QoS. Specifically, an AS fabric permits us to employ Virtual Channels (VCs), egress link scheduling, and an admission control mechanism. Moreover, AS performs a link-level flow control in a per VC basis. This means that both the scheduling and the flow control are made at a VC level. These mechanisms allow us to aggregate traffic with similar characteristics in the same VC and to provide each VC with a different treatment according to its traffic requirements. AS defines two egress link schedulers: The VC arbitration table scheduler and the Minimum Bandwidth egress link scheduler (MinBW). The main problem of the AS table scheduler is, as we will show, that it does not work properly with variable packet sizes. Regarding the MinBW scheduler, AS does not specify an algorithm or implementation for it, but some characteristics that it must respect.

1.3 Motivation and Objectives

As stated before, AS was intended for extending the capabilities of PCI Express, which is expected to become the next *de facto* local I/O interconnect. It was born with the

support of companies like Agere, Alcatel, Huawei, Intel, Siemens, Vitesse, and Xilinx, which founded the AS Interconnect Special Interest Group (ASI SIG), which was formed to develop the AS standard.

When this thesis began, AS in conjunction with PCI Express was believed to have the potential to provide an evolutionary yet revolutionary approach for building the next generation of interconnects [MK03]. On the other hand, QoS is an old topic that must be revisited in order to be adapted to the new high-performance interconnection technologies, which are incorporating QoS mechanisms in their specifications. The necessity to study, propose ways to configure, and improve the mechanisms integrated in the AS specification seemed a promising field of research.

Therefore, the main objective of this work has been to study the different mechanisms that AS provide in order to propose a general framework for providing QoS to the applications over this network technology. In this line, the main focus of this work, due to its importance for the QoS provision, is the study of the AS scheduling mechanisms. Our goal has been to implement them in an efficient way, taking into account both their performance and their complexity.

However, the AS interconnection technology is not finally going to meet its expectations. In fact, in February of 2007 the ASI SIG disbanded and transferred its specification and documentation to the PCI Industrial Computer Manufacturers Group (PICMG), which is a consortium of over 450 companies who collaboratively develop open specifications for high-performance telecommunications and industrial computing applications.

Nevertheless, the research performed in the provision of QoS over AS is still quite valuable. Most of the proposals that we present in this thesis can be directly employed in other high-performance technologies or at least can be easily adapted to them. Moreover, some of the ideas behind the AS interconnection technology are probably going to appear in those interconnection technologies intended to fill the gap that AS was intented to cover.

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

1. Studying the previous work. This involves two main research areas: High-performance networks in general and QoS provision in high-performance networks, including the study of the scheduling algorithms proposed until now.

2. Studying the AS specification. A deep study of the specification, especially of the mechanisms intended to provide QoS requirements, is required.

3. Developing a simulation tool to model high-performance networks. This tool must be adjusted to the AS specification but, it has to be flexible enough to test different 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.

4. Proposing possible implementations for the MinBW scheduler. As stated before, AS does not specify an algorithm or implementation for this scheduler, but some characteristics that it must respect. These characteristics and also the possible interaction with other mechanisms, like the link level flow control, must be studied.

5. Solving the problem of the AS table scheduler with variable packet sizes. The main limitation of the AS table scheduler is its problem to handle in an appropriate way variable packet sizes. This problem must be solved in order to be able to provide QoS based on bandwidth requirements with this scheduler.

6. Decoupling the bounding between the bandwidth and latency assignments of the table scheduler. Table-based schedulers can be configured to provide QoS based on latency requirements. However, this entails that those flows that require a low latency are assigned a high bandwidth, which can be a waste of resources. In order to be able to distribute the resources in an efficient way this bounding between latency and bandwidth must be decoupled.

7. Studying the hardware complexity of the different schedulers. A hardware design of the different schedulers must be done in order to obtain estimates of the computational and implementation complexity. Specifically, the objective is to obtain estimates of silicon area and arbitration time required by the schedulers.

8. Proposing a general framework for providing QoS over AS. This point includes how to configure the schedulers and the admission control mechanism in order to provide QoS based on bandwidth and latency requirements.

9. Evaluating our proposals from the performance point of view. In this case, we study the traditional QoS indices such as latency, jitter, and throughput.

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

1.4 Organization of the Thesis

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

- Chapter 1: This chapter introduces this thesis. Specifically, it presents the motivation and objectives of this work.
- Chapter 2: This chapter presents an architectural overview of high-performance networks. We review different network architectures that we can found. We also review the main components and possible organizations of the switches, which are one of the key network elements. Finally, we compare lossy networks with lossless networks, due to the fact that high-performance networks are usually lossless.
- Chapter 3: This chapter presents a brief state of the art on the provision of QoS in high-performance networks. Due to the importance of the packet scheduling algorithm to provide QoS, we dedicate a specific section to this topic.
- Chapter 4: In this chapter, we review the AS technology. Specifically, we focus on those AS traffic management mechanisms that can be used to provide QoS.
- Chapter 5: In this chapter we discuss about the implementation of the AS MinBW scheduler. We present three new fair queuing scheduling algorithms that fulfill all the properties that this scheduler must have and, therefore, can be implemented in this technology.
- Chapter 6: In this chapter we present the Deficit Table scheduling mechanism and its decoupling configuration methodology. Moreover, we show several possibilities in order to adapt the existing AS table scheduler into the DTable scheduler without modifying too much the AS specification.
- Chapter 7: In this chapter we present the hardware design employed to obtain estimates on silicon area and arbitration time required by the minimum bandwidth and table schedulers. Moreover, we analyze and compare the effect of several design parameters over the complexity of the different schedulers.
- Chapter 8: In this chapter we present a general framework to provide QoS over AS that uses some of the AS mechanisms reviewed in Chapter 4. Specifically, we present a traffic classification based on bandwidth and latency requirements, employ an admission control (AC) mechanism to ensure QoS provision, and show how to configure the minimum bandwidth and table-based AS egress link schedulers.

- Chapter 9: In this chapter we evaluate the performance of our proposals by simulation.
- Chapter 10: This chapter finishes the thesis. We summarize the work done and 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 Networks

Networks are responsible for the communication between the components of many systems. Therefore, They have been extensively studied and there are a plethora of proposals. High-performance networks are a subset of networks that are characterized by the application requirements, rather than physical characteristics [DYL02]. In this case, the applications demanding high-performance networks require a very high *bandwidth* and a very short response time or *delay*.

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. For example: 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, and 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 communications subsystem [DYL02]. This illustrates the importance of efficient high-performance interconnects.

In this chapter, we will review different network architectures that we can found. We will also review the main components and possible organizations of the switches, which are one of the key network elements. Finally, we will compare lossy networks with lossless networks, due to the fact that high-performance networks are usually lossless.

2.1 Network classification

Interconnection networks can be classified according to network topology [DYL02]. In this way, we would have: Shared medium networks, direct networks, and indirect networks. Hybrid approaches are also possible. This network classes will be described in the next sections.

2.1.1 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 listen simultaneously. The most common shared medium is a bus.

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.

2.1.2 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. Each node, in addition to the device contains a *switch*. Switches handle 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.

Direct networks are characterized by their *topology*, which is the way in which the switches are connected by channels. Popular network topologies include: Meshes, torus, K-ary n-cubes, trees.

Direct networks are very popular for high-speed interconnects, specially in multicomputers. There are many real-life examples of this network design and interested readers can consult [DYL02].

2.1.3 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 the connection pattern. In most interconnection technologies links are bidirectional (or pairs of two unidirectional links are bundled together) and thus, bidirectional MINs are used. In bidirectional MINs, connections have a forward path, a turnaround point, and a backward path. The advantage of this is that there are no cycles and routing is easy.

2.2 Switching techniques

A *switching technique* is the technique used to transfer information through the network. At the application level, the application generates *user messages* or just messages. These messages 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 with 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.

We will see in the following the four switching techniques more oftenly used in high-performance networking: Circuit switching, packet switching, virtual cut-through, and wormhole. A more comprehensive description of these techniques and other less usual techniques can be found at [DYL02].

2.2.1 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. This is realized by injecting a special message, which is called probe, into the network. This probe 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 probe 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.

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.

2.2.2 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. The first few bytes of a

packet contain routing and control information and are referred to as the packet header. The header information is extracted by the intermediate switches and used to determine the output link over which the packet is to be forwarded. This means that each packet is individually routed from source to destination. This technique is referred to as packet switching.

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 is 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).

Store-and-forward vs. Virtual cut-through

Packet switching can be implemented in two possible ways. In the first case, which is referred to as **store-and-forward** switching, a packet is completely buffered at each intermediate node before it is forwarded to the next node. Store-and-forward 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.

However, this is not generally necessary 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. With this switching technique 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 store-and-forward.

2.2.3 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 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.

2.3 Switch architecture

One of the key elements of a high-performance network are the switches. The main components of a generic high-performance switch are:

- Buffers. These are FIFO buffers for storing messages in transit. The buffer size must be an integer number of flow control units, otherwise some space would be wasted. Depending on the design, buffers may be associated only with inputs (input buffering), outputs (output buffering), or both.

- 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. 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.

- 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.

  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.

These basic blocks are found in most high-performance switch designs. However, the organization of the switch may vary. In order to do their 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. We will see in the following the most usual switch organizations.

2.3.1 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 the channel rate and N is the number of ports of the switch, the access rate to the central buffer is $(N + N) \times L$.

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. Moreover, 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 central buffer switch architecture.

2.3.2 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 output ports buffer aggregate bandwidth 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.

2.3.3 Input queuing

In an 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 [KHM87]. 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 [HK88], 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) [DCD98], 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 dynamic 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.

2.3.4 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.

2.3.5 Combined input-crosspoint queuing

The kind 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; QoS support is easily implementable; 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 fractioned 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.

2.4 Lossy versus lossless networks

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 the demand persists, buffers may be overflowed. There are two ways of handling this problem. The first one consists in dropping packets when buffers get full. The second one consists in implementing mechanisms that avoid transmitting packets if there is not enough space at

the other end to store those packets. The first possibility makes a network lossy, the second lossless.

2.4.1 Lossy networks

Buffer management algorithms are used in lossy networks to decide which packets to discard in congestion situations and when. The simplest buffer management algorithm is *Tail Drop*, which simply discards a packet if the queue is full at the arrival time of the packet. Another commonly algorithm used is Random Early Detection (RED) [FJ93], which can drop packets with certain probability even if the queues are not yet full. The characteristics of lossy networks are:

- The information that is lost must be retransmitted by the sources. A source knows that a packet was dropped either because it receives a NACK or because a configured time passes without receiving an ACK.
- 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, the delay of packets may get intolerably high for some applications.

A typical example of this kind of networks is ATM [For95] and traditionally Ethernet. However, this last technology included in its gigabit version [Sei98] an *stop and go* link level flow control mechanism that makes this technology lossless [RS05]. Nevertheless, this mechanism is not usually employed and thus, Ethernet is in general still a lossless network.

2.4.2 Lossless networks

Lossless networks employ link-level flow control mechanism to avoid dropping packets when congestion arises. Sometimes, these mechanism are also called *backpreassure* techniques. The characteristics of lossless networks are:

- Since packets are never dropped, no retransmissions 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.
- The problem of Head-Of-Line (HOL) blocking appears. This phenomenon happens when a packet ahead of a FIFO queue blocks because of the flow control, preventing the rest of packets in the same queue from advancing.
- When congestion persists over time, the buffers containing the blocked packets will be filled and the flow control will prevent other switches from sending packets to the congested ports. Therefore, the congestion will be rapidly propagated to other switches, even reaching the injecting end nodes. This has been called *tree saturation* [PN85] or, in other contexts, *congestion spreading* or *congestion tree*. Congestion trees may dramatically affect the performance of the network. The reason is that they affect not only flows that are directly contributing to the congestion, but other flows that share the same buffers due to the HOL blocking effect.
- 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 none of them can advance.

In high-performance networks, lossless flow control is generally preferred. This is the case in for example Myrinet [BCF95], Quadrics [BAP03], InfiniBand [Inf00], and Advanced Switching (AS) [Adv03]. The reason is that retransmissions and the delays they involve are not tolerable by the applications which use the network. There are two main flow control mechanisms for lossless networks: *Stop and go* and *credit based flow control*.

Stop and Go

In stop and go, the receiver buffer, of size B, has two marks, k_{STOP} and k_{GO}, such as $0 < k_{GO} < k_{STOP} < B$. 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

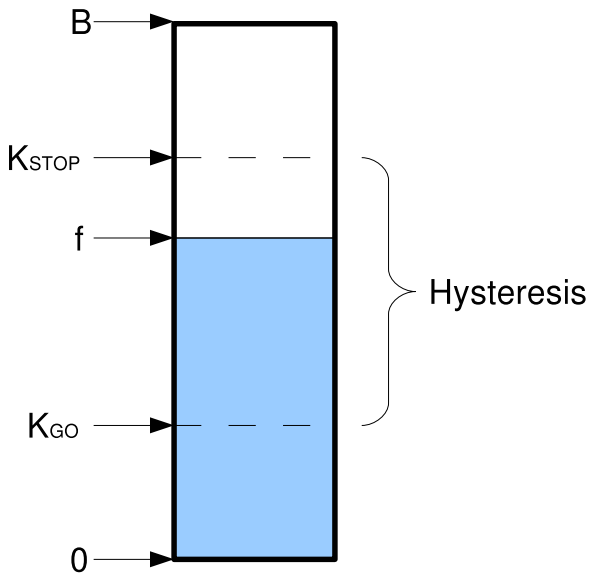


Figure 2.1: Elements of stop and go flow control.

between sender and receiver. The margin between both marks provides hysteresis, i.e. a working area where no signals are generated. The elements of the systems are illustrated in Figure 2.1.

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 oftenly used, with enough slack for the worst case.

Credit-Based Flow Control

In credit-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 sizes are used, the slot represents a fixed amount of information, for instance 64 bytes. This is known as the flow control unit or *flit*.

When the system is initialized, the receiver informs the sender with the number of flits in its buffer. The sender stores this value in a register, the *credits counter*. The

operation of the system is as follows: Every time the sender transmits a packet through the link, it decrements the *credits counter* with the number of flits of the packet. If a packet is larger than the amount of flits available in the *credits counter*, it is not transmitted. In this way, the receiver's buffer cannot be overflowed.

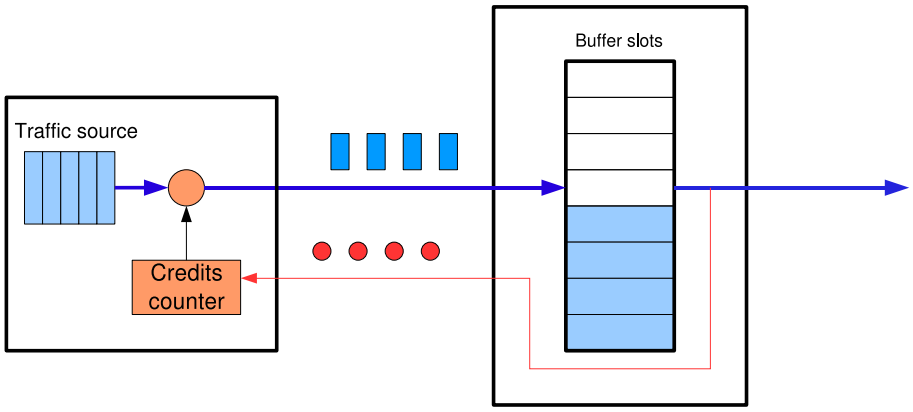


Figure 2.2: Elements of credit-based flow control.

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 *credits counter* of the sender is incremented and more packets are allowed into the link. The elements of the system are illustrated in Figure 2.2.

Chapter 3

QoS in High-Performance Networks

The importance of network QoS is widely accepted by both the research community and the manufacturers. However, the problem is that existing network devices 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.

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 [FH98, XN99]. 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 [CS99, BCN99].

Therefore, QoS is a very interesting topic in network design in many contexts. The work presented in this thesis is about providing QoS over AS, which is a packet-switched high-performance network. Therefore, in this section we will focus on the provision of QoS in high-performance networks.

3.1 Application traffic requirements

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.

From a networking perspective, 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 [GG99, Bla00], 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 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. Therefore, messages that are delivered late translate into wasted throughput and worse QoS for the applications. The analysis of the delay components over the source-to-destination path shows that up to 100-150 ms can be spared for compression, packetization, jitter compensation, propagation delay, etc. [GGK99], leaving no more than few tens of milliseconds for queuing delay within the many switches on the path. 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 [IEE04]).

- Jitter. The variation of the latency of two consecutive messages received by an application is called *jitter* [ECT05]. Figure 3.1 illustrates a typical probability density function for latency and shows that jitter is bound between the minimum and maximum latencies [Wan01]. 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

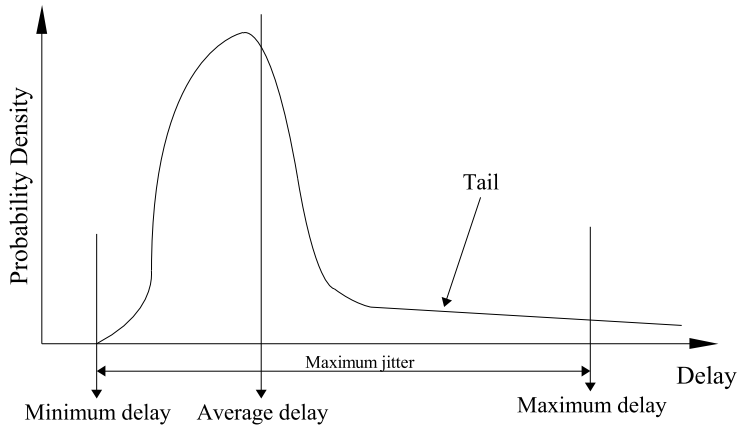


Figure 3.1: Packet latency probability density.

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 rate 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 high-performance networks. For this reason, most high-performance networks implement mechanisms to avoid dropping packets in the face of congestion (as we saw in Section 2.4).

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 [Hal01]. 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.

The degree of tolerance or sensitivity to each of these parameters varies widely from one application to another. For example, multimedia applications are usually sensitive to latency and jitter, but many of them can tolerate packet losses to some extent. However, the severity of the effect of losses on the quality of these applications is also influenced by parameters such as the compression and encoding techniques used, the loss pattern,

the transmission packet size, and the error recovery technique implemented [WZ98]. For a further discussion about different applications and their requirements, see [EGBS03].

The QoS parameters an application may wish to specify can be expressed in a quantitative manner or a qualitative manner. While the former specifies numbers and "quantities" in order to express the QoS requirements, the latter usually specifies relative levels such as *better than* or *low loss*. Quantitative requirements can be expressed in a *deterministic* or *statistical* way such as percentile or average values.

3.2 Traffic classes

In the previous section, we introduced 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 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 introduced by ATM [For95], 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, oftenly 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 broadcasts 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 desirable that switches provide the best performance that is possible. For this reason, this traffic is also known as *best-effort traffic*.

- UBR (*Unspecified 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 [KLC98] 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 the proposed by Pelissier [Pel00], 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 other types of traffic.

There are many other classifications, but we will see just another one. In recent IEEE standards, like for instance IEEE standard 802.1D-2004 [IEE04], there are seven traffic classes, briefly described in Table 3.1.

Table 3.1: Traffic types suggested by the standard IEEE 802.1D-2004.

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

This IEEE classification mixes quantitative and qualitative requirements for the different types of traffic. In defines a control traffic type with a generic low latency requirement and two traffic types, voice and video, with explicit bandwidth, latency, and jitter requirements. This two traffic types could be compared with the DBTS traffic type proposed by Pelissier. It also considers a traffic type with only bandwidth requirements, as the Pelissier DB traffic type. Finally, this classification propose to differentiate among three types of best-effort traffic with qualitative requirements.

3.3 Per flow versus per class QoS provision

There are many choices related to the provision of QoS. One of the most important choices is whether resources are allocated for individual flows or for traffic aggregates. Depending on this decision we will have a different QoS model. The two outstanding examples of both models are integrated services (IntServ) [BCS94], which handle individual flows; and differentiated services (DiffServ) [BBC98, Ber98], which handle flow aggregates.

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.

Besides the resource reservation procedure, in IntServ also classification and forwarding actions, such as scheduling, are made on a per-flow basis. "Flow Specs" [Par92] describes what the reservation is for, while RSVP [BZB97] is the underlying mechanism to signal it across the network.

The main problems of IntServ QoS architecture are [XN99]:

- 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 RSVP protocol, connection admission control, QoS arbiting, etc.
- In order to 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 [BBC98] is a QoS architecture that specifies a simple, scalable, and coarse-grained mechanism for classifying and 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.

In DiffServ, the traffic is divided into a limited number of forwarding classes meaning that the resources are allocated to traffic aggregates instead of individual flows. The forwarding class of a packet is encoded in the IP packet header. In DiffServ, no resource reservations are made. Instead, assurances are based on prioritization, provisioning and possibly admission control.

The DiffServ architecture addresses the problem of scalability by keeping the state information at the network edges. The task of the edge nodes is to perform packet classification and traffic conditioning: Packets are first classified based for example on their source or destination address or application type (port number) and marked with an appropriate DiffServ Code Point (DSCP) value; the conditioner then measures how well the traffic of the flow matches its traffic profile. All packets that are in-profile are sent to the network, while the out-profile packets may be remarked, shaped or dropped. The core nodes, on the other hand, merely forward packets according to the DSCP in the packet

header: Forwarding actions, such as scheduling, are performed on per-class rather than per-flow basis.

Contrary to IntServ, DiffServ does not define any end-to-end service. Instead, in DiffServ each forwarding class represents a per hop behavior (PHB) that defines how a packet is treated in a single node. The PHB can be implemented by using buffer management and scheduling. Even though the PHBs only define forwarding treatments, not end-to-end services, the end-to-end services can be constructed from the PHBs by combining them with admission control.

3.4 Traffic management mechanisms

In order to provide applications with their QoS requirements, different control mechanisms must be implemented in the operations and management of the network. During the last decade various QoS control mechanisms have been proposed for different purposes. In order to provide QoS guarantees congestion must be avoided in the queues employed by QoS flows. Therefore, a certain degree of overlap exists with congestion control techniques.

One useful way to classify QoS mechanisms is based on the time scale at which they operate. Starting from the shortest time scale, the levels of QoS control mechanisms can be divided into packet level, round-trip-time level, session level and long-term level [FBT01]. Figure 3.2 shows a classification of some QoS mechanisms following this approach.

Mechanisms operating at the packet level time scale ($\backsim 1 - 100\mu s$) include traffic classifiers, policers, markers, shapers, packet schedulers, buffer management, link-level flow control mechanisms, etc. The next fastest time scale, the round-trip-time scale ($\backsim 1 - 100ms$), is the time scale where feedback-based flow and congestion control operate. The session time scale (from seconds to minutes or longer) refers to the time that user sessions usually last and thus this is the time scale of admission control and QoS routing.

Finally, the long-term time scale ranges from minutes to even months. Mechanisms operating at this level are for instance traffic engineering and capacity planning. The rest of this section gives a brief description of the most important QoS control mechanisms mentioned above.

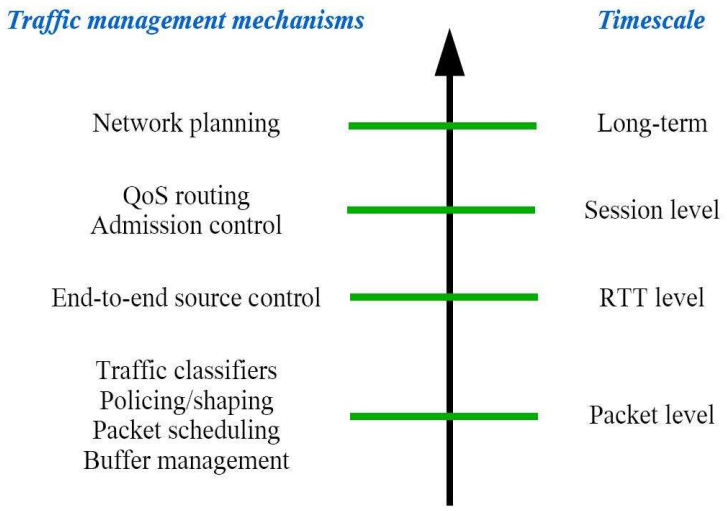


Figure 3.2: Time scale hierarchy of QoS control mechanism.

3.4.1 Traffic classification

Classification is a process that recognizes which connection or class the incoming packets belong to. As a result of the classification process, the total incoming traffic stream is divided into logically separate substreams that can be treated in different ways.

3.4.2 Policing/Shaping

Traffic policing is typically deployed at the edge of a network and/or close to the source. Upon arrival of a packet, a policing algorithm first determines if the packet is in compliance with the service-level agreement negotiated between the source of the traffic and the network. If not, the traffic may be remarked, shaped or even dropped. Shaping mechanisms conformate the traffic into a given controlled pattern. It is used to smooth traffic and reduce its variation over time.

The token buckets (and leaky buckets) are the most common mechanisms used for policing/shaping traffic at a network node. A token bucket has a bucket of depth b and generates tokens at the rate of τ. Each arriving packet consumes a token (or a number of tokens directly proportional to the packet size, depending on the implementation) before it can be transmitted into the network.

3.4.3 Packet scheduling

Packet scheduling is one of the most important QoS mechanisms because it enforces resource allocation by deciding when packets are transmitted. These decisions have a great impact on performance parameters such as throughput, delay, or jitter. The objective of packet scheduling is to share the common resources so that some predefined policy will be met. We have dedicated Section 3.5 to this important topic.

3.4.4 HOL blocking elimination techniques

As stated in Section 2.4.2 in page 19 lossless networks have the problem of HOL blocking. This phenomenon happens when a packet ahead of a FIFO queue blocks because of the flow control, preventing the rest of packets in the same queue from advancing. Several techniques have been proposed for eliminating HOL blocking and, in general, the most effective ones are based on storing packets belonging to different flows in separate queues at each network port. The most relevant techniques implementing this basic idea are reviewed in the following sections.

Virtual Output Queues at network level (VOQnet)

This technique requires, at each switch port, as many queues as end nodes in the network, and every incoming packet will be stored in the queue assigned to its destination. This technique is in general very effective in HOL blocking elimination, since flows addressed to different destinations will be always stored in different queues. However, VOQnet requires a lot of resources and does not scale with network size.

Virtual Output Queues at switch level (VOQsw)

In this case, there will be, at each incoming switch port, as many queues as output ports in the switch, and every incoming packet will be stored in the queue assigned to the output port requested by the packet. Therefore, the number of queues at each port depends on the number of switch ports, but not on the number of network endpoints. However, this technique only solves the HOL blocking problem at the switch level.

Regional Explicit Congestion Notification (RECN)

.

Regional Explicit Congestion Notification (RECN) [GQF06] is a novel congestion control technique that claims to be efficient and scalable. The RECN mechanism detects flows that produce congestion and it separates them in independent queues. This is a dynamic process that efficiently manages problematic flows by allocating new queues only when needed.

3.4.5 QoS routing

Traditional routing is based on finding the shortest path between the source and the destination. However, this shortest path approach may not be optimal, since usually it causes traffic to centralize in some parts of the network. Therefore, it could be better for a flow to be routed along a path that may not be the shortest but that is less heavily loaded. The idea in QoS routing is to find a path for a flow or for a traffic aggregate in the network under multiple constraints, such as delay and bandwidth.

QoS routing algorithms can be greedy in the sense that they try to optimize the performance of one flow or aggregate without taking into account network wide effects [Wan01]. However, in wider sense one objective of QoS routing is also to achieve high resource utilization in the network.

3.4.6 Admission control

Connection admission control or just Admission Control (AC) is a set of actions taken by the network to decide whether a new connection is accepted or rejected. It is a preventive load control mechanism that protects the QoS requirements for all the connections including the newly admitted one [LZ01]. Many studies dedicated to AC can be found in the literature because of the AC crucial role with respect to QoS guarantees and network resources management. A lot of solutions centralized or distributed, static or dynamic, more or less adaptive have been proposed in papers and different research projects.

The AC approaches can be classified based on the main method used to take decisions about the admission or rejection of the new connections. The work [GTP04] identifies several basic solutions for AC: Based on a priori traffic knowledge or descriptors, based on

measurements upon the actual resources utilization, and based on probe packets sent into the network to test its current capabilities. Combined methods can be also used.

A priori-based admission control

The a priori-based AC [LZ99] is based on the assumption that it has perfect knowledge of the traffic characteristics of the new connection and the number and traffic characteristics of each connection that is traversing each link. The AC also knows the total network resource capabilities in terms of available bandwidth and in some cases also available buffer space. This information will enable the AC to compute the total amount of resources required. Hence, it will only accept a new connection if there are enough resources to provide the QoS requirements to the new connection and to the already established connections. The implementation of this approach is simpler than for other methods because it does not involve the monitoring system of the network.

In [KS99] a classification depending on the test needed to accept/reject a new connection of several a priori-based AC schemes is performed:

- Tests based on average and peak rate combinatorics.
- Tests based on additive effective bandwidths.
- Tests based on engineering the *loss curve*.
- Tests based on maximum variance approaches.
- Tests based on refinements of effective bandwidths using large deviations theory.

The actual performance of the a priori-based AC schemes depends essentially on the accuracy of traffic descriptors and the degree of conformance of the real traffic flows with respect to the descriptors. Note that, since no traffic measurement is taken into consideration, the performance of this admission control scheme can be very low if the provided traffic descriptors do not depict the actual behavior of the sources, for instance that could happen in case of non-conformant non-policed sources, or the appropriate traffic descriptors are not known a priori.

Measurement-based admission control

The measurement-based AC [GT99] does not take its decision based on the user issued information on traffic descriptors, but on information delivered by the network monitoring

system. This subsystem makes real-time measurements, thus trying to "learn" the traffic characteristics. Therefore, the total demand is not calculated by AC based on traffic models and the number of active connection instances but it uses the real traffic load value which has been measured. This method has the advantage that the user-specified traffic descriptors can be very simple, which can be easily policed (e.g. peak rate only). The over-provisioning is less probable than in the first method. Also, by measuring the aggregated flows, the statistical values computed are more accurate than estimating the statistical characteristics for individual flows. The main problems of the method are related to the accuracy of measurements (estimation errors), system dynamics and memory related issues [GT99].

Probe-based admission control

In the probe-based AC, the end host/application sends probe packets through the network to test the desired path [BKS00]. Using some predefined metric the host decides if the flow can be admitted. The route followed by the probes should be the same for real packets. The probe-based schemes deduce the network ability to sustain the offered load directly, without relying on pre-allocated network capacity information. Because these methods rely on potentially imprecise end-to-end measurements to guide their AC decisions, endpoint AC is primarily intended for soft real-time services, similar to Intserv Controlled Load or Diffserv qualitative services, in which the aggregate load is kept at reasonable levels but no hard guarantees are given to individual flows.

They introduce latency in response times, and have inherent problems caused by probes stealing bandwidth from established flows and denial of service when simultaneous attempts congest the network and none is accepted although resources are available [BKS00]. Moreover, probe-based algorithms are limited by a traffic awareness that is restricted to the traversal route while fluctuating traffic patterns, especially within a busy network, provide limited temporal information describing the network load. The collection and calculation of statistical data can be both costly to gather and process [RSJS03].

3.4.7 Network planning

Network planning is a longer term process that includes deciding what elements and mechanisms will be used in the network and how the network should be dimensioned and provisioned. For example, when planning a radio network, questions such as how many base stations are required, need to be addressed.

3.5 Scheduling algorithms

Service discipline, also called packet scheduling, is an important mechanism to provide QoS guarantees in computer networks, such as end-to-end delay bounds and fair bandwidth allocation [DKS89], [GM92], [Zha95]. During the last decades a vast amount of scheduling disciplines have been proposed in the literature for different purposes. This section outlines some desirable properties of scheduling disciplines and presents possible ways to classify scheduling disciplines.

In order to be able to design new scheduling disciplines and to compare the existing ones with each other, it is important to define the desirable properties of a scheduling discipline. It is obvious that many of these properties are tightly related to the QoS guarantees made for the end user. However, there are also some general desirable properties:

Good End-to-End Delay As stated before, the end-to-end delay (also called latency) is defined as the sum of the transmission delay, the propagation delay, and the queuing delay experienced at each network node. The last component is by far the most significant. In some applications if a packet experiences a latency higher than a certain value, the value of the packet information may be greatly diminished or even worthless. Moreover, a larger delay bound implies increased burstiness of the session at the output of the scheduler, thus increasing the buffering needed at the switches to avoid packet losses [SV98]. Thus, a good scheduling algorithm should guarantee acceptable queuing delay.

Flexibility The scheduling discipline should be able to accommodate applications with varying traffic characteristics and performance requirements rather than just optimize the performance from a certain application's point of view [Zha95]. In future networks several applications with diverse requirements will have to be supported making necessary for the scheduling discipline to be flexible.

Protection Real network environment is not static. As a consequence, the scheduling discipline should be able to protect the well behaving users from different sources of variability, such as best-effort traffic, bad behaving users and network load fluctuations [Zha95]. Bad behaving users refer, for example, to users who send more packets than their traffic profile allows. Network load fluctuations, on the other hand, are caused by traffic bursts at a router. These bursts may accumulate even if the users meet their traffic constraints at the entrance of the network. Ideally, the scheduling discipline should be able to satisfy the performance requirements of well behaving users even in the presence of these factors.

Simplicity Performance characteristics are not the only parameters that must be taken into account when deciding which is the best scheduler in networks with QoS support. Other important property, specially in high-performance networks, is simplicity [Siv00]. This is because in order to achieve a good performance, the processing overheads must be some orders of magnitude smaller than the average packet transmission time. This means that the time needed to decide the next packet to be transmitted must be very small, if we consider the high speed of high-performance networks. Moreover, a low complexity is required in order to be able to implement the scheduler in a small silicon area (note that high-performance switches are usually implemented in a single chip).

Scheduling disciplines can be categorized in many ways. Traditionally they have been divided into work-conserving and non-work-conserving disciplines [Zha95]. Another possible classification is based on their internal structure, according to which there are two main architectures: Sorted-priority and frame-based [Sti96]. Other differentiation can be made based on if they are intended to provide bandwidth or latency requirements. Figure 3.3 summarizes these possible categorizations.

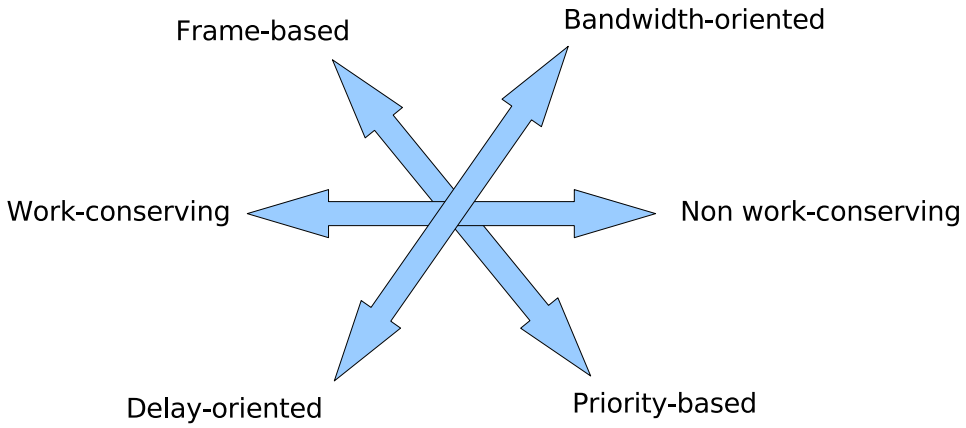


Figure 3.3: Scheduler classification.

A work-conserving scheduling discipline serves packets as long as there is a nonempty queue in the system. The server is idle only when there are no packets to be sent. A non-work-conserving server, on the other hand, may remain idle even if there are packets waiting in the system. Non-work-conserving servers obviously have larger average delays than work-conserving servers. They also result in lower utilization of network resources

[LY99]. However, non-work-conserving scheduling disciplines also have some important advantages. For instance, the server may postpone the transmission of a packet if it expects a more important packet to arrive soon [Sti96].

Moreover, non-work-conserving scheduling disciplines may also be used to control the delay jitter (maximum difference between the inter arrival times of consecutive packets) of real-time applications: The server delays the packets so that their inter arrival times remain roughly constant. Another possible application of non-work-conserving scheduling disciplines is shaping [LY99].

Sorted-priority scheduling disciplines use a global variable, often called virtual time (to distinguish it from real time), associated with the server. The purpose of this variable is to keep track of the progress of the server and it is usually updated at packet arrival and departure instants. For each packet in the system, a time stamp is computed as a function of this variable. Packets are then sorted based on these time stamps and served in this order. The complexity of a sorted-priority algorithm is determined by the complexity of calculating the time stamp, updating the priority list and selecting the highest priority packet for transmission. The complexity of time stamp calculation is dependent on the specific scheduling discipline. For example, in Weighted Fair Queuing (WFQ) [DKS89] the updating of virtual time is considerably more complex than in the Self-Clocked Fair Queuing (SCFQ) [Gol94].

Frame-based scheduling disciplines use a frame of fixed or variable length which is divided among different connections/classes based on the reservations of the connections/resources allocated for the class. The more resources are allocated for a connection/class, the larger part of the frame it receives. The frame is split among the connections/classes in a similar way in each service round.

In the next section we will review two kind of schedulers of special interest in this thesis: Fair scheduling algorithms, which are work-conserving bandwidth-oriented algorithms, and table-based schedulers, which are included in the frame-based category.

3.5.1 Fair queuing algorithms

Fair queuing algorithms allocate bandwidth to the different flows in proportion to a specified set of weights. The perfect fair queuing scheduling algorithm is the General Processor Sharing (GPS) scheduler [DKS89], [PG93]. However, GPS is an ideal fluid-based algorithm that cannot be actually implemented and thus, several packet-based approximations have been proposed, which try to emulate the GPS system as accurately and simply as possible.

General Processor Sharing (GPS)

GPS is said to be an ideal algorithm since it is based on a fluid model (as shown in Figure 3.4) and thus, assumes that traffic is infinitely divisible and that different flows can be served simultaneously in a weighted fashion.

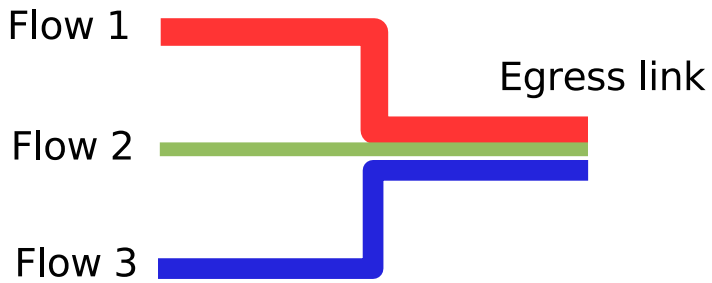


Figure 3.4: GPS fluid model.

In a link of rate R served by a GPS scheduler, each session i is assigned a weight value ϕ_i that reflects the amount of resources that should be allocated for the flow. If there are N flows served, then for any two backlogged[1] flows i and j,

$$\frac{r_i(\gamma,t)}{r_j(\gamma,t)} = \frac{\phi_i}{\phi_j},$$

where $r_i(\gamma,t)$ denotes the amount of traffic served for flow i in an interval (γ,t). Therefore, in any interval (γ,t) flow i receives service with a rate:

$$r_i \geq \frac{\phi_i}{\sum_{j=1}^{N}\phi_j} \times R$$

This corresponds to the situation where all the flows are backlogged during the interval. However, if some flows are not backlogged, the excess of bandwidth will be distributed among the backlogged flows in proportion to their weights. Then, each backlogged flow i at every moment t is served simultaneously at rate:

$$r_i(t) = \frac{\phi_i}{\sum_{j\epsilon B(t)}\phi_j} \times R,$$

where $B(t)$ is the set of sessions that are currently backlogged at time t.

[1]A flow is considered backlogged when there are packets from that flow ready to be transmitted in the scheduler queues.

GPS is considered to be an attractive scheduling discipline because it has many desirable properties. First, it provides fairness for the flows by servicing each flow with a rate equal to or greater than the flows's guaranteed rate. Second, if the incoming traffic is leaky-bucket constrained [Tur86], it has been proved that strict bounds for worst-case network queuing delay exist [PG93]. Third, the classes can be treated in different ways by varying the weights. For instance, if there are two classes with weights $\phi_1 = 1$ and $\phi_2 = 0$, GPS reduces to strict priority scheduling. On the other hand, if all classes are assigned equal weights, GPS behaves as a uniform processor sharing system.

However, despite these advantages, GPS is not a realistic service discipline since in a packet network, service is performed packet-by-packet, rather than "bit-by-bit" and thus, it cannot be implemented in practice. Different packet-by-packet approximations of GPS have been proposed, which try to emulate the GPS system as accurately and simply as possible while still treating packets as entities. It has been shown that scheduling algorithms can provide similar end-to-end delay bounds to GPS if their packet service does not significantly differs from GPS [PG93]. Examples of these approximations are Weighted Fair Queuing (WFQ) [DKS89], packet-by-packet GPS [PG93], Self-Clock Fair Queueing SCFQ [Gol94], Worst Case Weighted Fair Queuing (WF2Q) [BZ96], frame-based fair queuing [SV96], and Hierarchical Packet Fair Queuing [BZ97].

A real-world packet-by-packet service discipline typically consists of the following two functions:

1. **Tracking GPS time:** This function tracks the progress of GPS virtual time (described later) with respect to the real time. Its main objective is to estimate the GPS virtual start and finish times of a packet, which are the times that a packet should have started and finished to be served, respectively, if served by a GPS scheduler.

2. **Scheduling according to GPS clock:** This function schedules the packets based on the estimation of their GPS virtual finish/start times. For example, WFQ selects the packet with the lowest GPS virtual finish time among the packets currently in queue to be served.

The algorithms that follow this approach are included in the "Sorted-priority" family of algorithms. This kind of scheduling algorithms assign each packet a tag and scheduling is made based on the ordering of these tags. "Sorted-priority" algorithms are known to offer good delay bounds [SV98]. However, this family of algorithms suffers from two major problems. The first problem is that these algorithms require processing at line speeds for tag calculation and tag sorting. In other words, each time a packet arrives at a node, its

time tag is calculated and the packet is inserted at the appropriate position in the ordered list of packets waiting for transmission. This means that these algorithms require at least the complexity of a search algorithm in the list of queued packets: $O(log(N))$, where N is the maximum number of packets at the queue, or if the buffers are not shared, $O(log(J))$, where J is the number of active flows. The complexity of computing the GPS virtual finish times of the packets has long been believed to be $O(J)$ [PG93, SV96, SV98, CG01]. In [ZX04] and [XL05] a deeper discussion on this topic can be found.

The second problem that may happen in the sorted-priority approach is that the virtual clock cannot be reinitialized to zero until the system is completely empty and all the sessions are idle. The reason of this is that the time tag is an increasing function of the time and depends on a common-reference virtual clock, which in turns reflects the value of the time tag of previously served packets. In other words, it is impossible to reinitialize the virtual clock during the busy period, which, although statistically finite (if the traffic is constrained), can be extremely long, especially given that most communication traffic has been shown to exhibit self-similar patterns which lead to heavily tailed buffer occupancy distributions.

Therefore, for practical implementation of sorted-priority algorithms, very high-speed hardware needs to be designed to perform the sorting, and floating-point units must be involved in the computation of the time tags.

Weighted Fair Queuing (WFQ)

Weighted Fair Queuing (WFQ) [DKS89], also known as packet-by-packet GPS (PGPS) [PG93], is perhaps the best known scheduling discipline approximating the GPS system. The basic idea in WFQ is to emulate the GPS system by stamping each packet p that arrives at the egress link, with a virtual finish time F_p that represents the time at which the packet would depart under the reference GPS system. The packets are then served in increasing order of the time stamps. It should be noted, however, that at the time when the WFQ server becomes free, it may be that the next packet to depart under GPS has not yet arrived [Zha95]. For example, suppose that there is only one large packet in the WFQ system at time γ when the server becomes free. In order to be work-conserving, the server selects this packet for transmission. However, just after time γ a very small packet could arrive, such that it would be served under the GPS system before the large packet. Obviously, it is not possible for the server to be both work-conserving and serve the packets in exact order of F_p. Thus, an additional condition is needed to describe the functioning

of WFQ [PG93]: The server picks the first packet that would complete service in the GPS simulation if no additional packets were to arrive after time γ.

In WFQ, the calculation of virtual finish times is based on the simulation of the reference GPS system in the background. The GPS virtual time *V(t)*, as a function of real time t, is calculated as follows:

$$V(0) = 0$$

$$V(t_{j-1} + \gamma) = V(t_{j-1}) + \frac{\gamma}{\sum_{i \epsilon B_j(t)} \phi_i}$$

$$\gamma \leq t_j - t_{j-1},\ j = 2, 3, ...$$

Here $\{t_j\}_{j=1,2,...}$ are the times at which two types of events happen under GPS:

- The service starts for a new packet.
- The service finishes for a packet currently in queue.

In order to calculate each packet tag, let A_i^k be the real time that the k^{th} packet of the i^{th} session arrives and L_i^k be its length. Let S_i^k and F_i^k be the virtual times when it should have started and finished service under GPS, respectively. These two virtual times of a packet are calculated as soon as the packet arrives as:

$$S_i^k = max\{F_i^{k-1}, V(A_i^k)\}$$

$$F_i^k = S_i^k + \frac{L_i^k}{\phi_i}$$

The WFQ algorithm is one of the best approximations of GPS and thus, it offers very good latency bounds. However, the price to be paid for this advantage is the very high implementation and computation complexity of this scheduling mechanism. This complexity comes mainly from the cost of the real-time emulation of the GPS fluid system. Specifically, it comes from keeping track of the set of active flows [Gol94].

Worst-case Fair Weighted Fair Queuing (WF2Q)

Worst Case Weighted Fair Queuing (WF2Q) [BZ96] is a variant of WFQ that aims at emulating more accurately the GPS system. Whereas in WFQ only the finish times of packets in the GPS system are used for making scheduling decisions, in WF2Q also the start times are considered. More precisely, the WF2Q algorithm selects for transmission

a packet that would be finished first in the GPS system, from among those packets that have already started receiving service in GPS.

It is shown that the service order of packets under WFQ and WF2Q system can be different for the same traffic arrival pattern [Zha95]. Since WFQ may select a packet for transmission even if the packet has not started being served in GPS, WFQ can be far ahead of the GPS system. On the contrary, in WF2Q there is no such problem. According to [Zha95] the service provided by WF2Q and GPS can differ at most by one packet size. WF2Q nevertheless presents the same disadvantage as WFQ, namely the additional complexity introduced by the real-time emulation of the GPS fluid system.

Self-Clocked Fair Queuing (SCFQ)

The Self-Clocked Fair Queuing (SCFQ) algorithm [Gol94] defines fair queuing in a self-contained manner and avoids using a hypothetical queuing system as reference to determine the fair order of services. This objective is accomplished by adopting a different notion of virtual time. Instead of linking virtual time to the work progress in the GPS system, the SCFQ algorithm uses a virtual time function which depends on the progress of the work in the actual packet-based queuing system. This approach offers the advantage of removing the computation complexity associated to the evaluation of $V(t)$ that may make WFQ unfeasible in high-speed interconnection technologies.

Therefore, when a packet arrives, SCFQ uses the service tag (finish time in WFQ) of the packet currently in service as the $V(t)$ to calculate the new packet tag. Thus, in this case the service tag is computed as

$$S_i^k = max\{S_i^{k-1}, S_{current}\} + \frac{L_i^k}{\phi_i}$$

As stated before, the SCFQ algorithm avoids the emulation of a GPS system to maintain the *virtual time*. This reduces the computational complexity of the tag calculation. Therefore, the computational complexity of the SCFQ algorithm is lower than the complexity of the WFQ algorithm. However, the simplification in computation does not come without a cost: In some situations SCFQ can perform worse than WFQ and WF2Q. Figure 3.5 shows the pseudocode for the SCFQ algorithm.

PACKET ARRIVAL (newPacket,flow):
$newPacket_{serviceTag} \leftarrow max(currentServiceTag, flow_{lastServiceTag}) + \frac{newPacket_{size}}{flow_{reservedBandwidth}}$
$flow_{lastServiceTag} \leftarrow newPacket_{serviceTag}$

ARBITRATION:
while (There is at least one packet to transmit)
    $selectedPacket \leftarrow$ Packet with the minimum $serviceTag$
    $currentServiceTag \leftarrow selectedPacket_{serviceTag}$
    Transmit $selectedPacket$
    if (There are no more packets to transmit)
        $\forall flow$
            $flow_{lastServiceTag} \leftarrow 0$
        $currentServiceTag \leftarrow 0$

Figure 3.5: Pseudocode of the SCFQ scheduler.

Weighted Round Robin (WRR)

Weighted Round Robin (WRR) is a frame-based scheduling discipline that provides a simple way to emulate the GPS system. In the WRR, a list of flow weights is visited sequentially, each weight indicating the number of packets from the flow that can be transmitted. The WRR algorithm faces a problem if the average packet size of the different flows is different. In that case, the bandwidth that the flows obtain may not be proportional to the assigned weights. Therefore, the WRR algorithm does not work properly with variable packet sizes. However, today network technologies usually use variable packet sizes.

Deficit Round Robin (DRR)

The DRR algorithm [SV95] is a variation of the WRR algorithm that works on a proper way with variable packet sizes. In order to handle properly variable packet sizes, the DRR algorithm associates each queue with a *quantum* and a *deficit counter*. The quantum assigned to a queue is proportional to the bandwidth assigned to that queue. The deficit counter is set to 0 at the beginning. The scheduler visits sequentially each queue. For each queue, the scheduler transmits as many packets as the quantum allows. When a packet is transmitted, the quantum is reduced by the packet size. The unused quantum is saved in the deficit counter, representing the amount of quantum that the scheduler owes the queue. At the next round, the scheduler will add the previously saved quantum to the current quantum. When the queue has no packets to transmit, the quantum is

discarded, since the flow has wasted its opportunity to transmit packets. Figure 3.6 shows the pseudocode for this algorithm.

```
while (There is at least one packet to be transmitted)
    if ((There are no packets in the queue of selectedFlow) or
        (selectedFlow_sizeFirst > totalQuantum))
        deficitCounter_selectedFlow ← totalQuantum
        selectedFlow ← Next active flow
        totalQuantum ← deficitCounter_selectedFlow + quantum_selectedFlow
    totalQuantum = totalQuantum − selectedFlow_sizeFirst
    Transmit packet from selectedFlow
    if (There are no more packets in the queue of selectedFlow)
        totalQuantum ← 0
```

Figure 3.6: Pseudocode of the DRR scheduler.

A well-known problem of the WRR and DRR algorithms is that the latency and fairness depend on the frame length. The frame length in these algorithms is defined as the sum of all the weights in the WRR algorithm or the quantums in the DRR algorithm. The longer the frame is, the higher the latency and the worse the fairness. In order for DRR to exhibit lower latency and better fairness, the frame length should therefore be kept as small as possible. Unfortunately, given a set of flows, it is not possible to select the frame length arbitrarily. According to the implementation proposed in [SV95], DRR exhibits $O(1)$ complexity provided that each flow is allocated a quantum no smaller than the MTU. As observed in [KSP02], removing this hypothesis would entail operating at a complexity which can be as large as $O(N)$. Note that this restriction affects not only the weight assigned to the smallest flow, but to the rest of the flows in order to keep the proportions between them.

The complexity of the DRR algorithm is quite small. Provided that each flow is allocated a quantum no smaller than the MTU and if a list of active flows is maintained, the algorithm can cycle through the list knowing that it is always possible to transmit at least one packet from each flow. This means that there will never be a need to cycle through the entire table several times in order to gather enough weight for the transmission of a single packet. Each time a packet is transmitted, the algorithm must compute if more packets from the same flow can be transmitted or it must change to the next active flow. However, this computation can be performed with simple integer units.

3.5.2 Table-based schedulers

The "sorted-priority" fair queuing algorithms, like WfQ, WF2Q, and SCFQ, are known to offer very good delay [SV98]. However, their computational complexity is very high, making their implementation in high-speed networks rather difficult. The Deficit Round Robin (DRR) algorithm [SV95] has a very low computational complexity, but depending on the situation the latency that provides can be very bad.

On the other hand, in the table-based schedulers instead of serving packets of a flow in a single visit per frame, like in the WRR or DRR, the service is distributed throughout the entire frame. This approach is followed in [CM03] and in two of the last high-performance network interconnection proposals: Advanced Switching (AS) [Adv03] and InfiniBand (IBA) [Inf00]. These table-based schedulers are intended to provide a good latency performance with a low computational complexity.

List-based WRR

In this generalization of the classical WRR discipline, instead of serving packets of a flow in a single visit per frame, the service is distributed throughout the entire frame. For this, a list of flow identifiers, called "service list", is maintained. When scheduling is needed, the list, or table, is cycled through sequentially and a packet is transmitted from the flow indicated by the current table entry. The number of times that a flow identifier appears in the service list is proportional to its weight, but these appearances are not necessarily consecutive as in the classical WRR algorithm. Note that, the list-based WRR, as the original WRR, is intended for environments with fixed packet size.

In [CM03], three ways of distributing the flow identifiers to conform the service list are proposed: Simply Interleaved WRR, Uniformly Interleaved WRR, and WF2Q Interleaved WRR. These three possible ways of distributing the flow identifiers result in three different schedulers with different characteristics. Note that, in all the cases the proportion of table entries associated with each flow indicates the bandwidth assigned to each flow. Therefore, the difference between the three schedulers is in the way of distributing the flow identifiers among the table entries. These different forms of interleaving the flow identifiers result in different latency characteristics for the three schedulers.

In [CM03] it is shown that all the approaches are able to improve the performance of the classical WRR. However, the WF2Q Interleaved WRR approach offers the best properties. Any of the three list-based WRR approaches can be implemented in either, the InfiniBand or AS table-based schedulers. Note that, the proposed list-based WRR schemes

do not involve packet tag calculation and sorting, and hence, they have lower implementation complexity than the "sorted-priority" schemes. These reasons make promising this kind of schedulers.

In order to compute any service list for a list-based WRR scheduler, let be w_i the integer weight assigned to each flow i, J the number of flows, and $N = \sum_{i=1}^{N} w_i$ the number of entries of the service list.

Simply Interleaved WRR In order to compute the service list of this approach, we divide the service list in $W_{max} = max\{w_i\}_{i=1}^{J}$ sets of entries (bins). Session with weight w_i registers itself in the first w_i bins. Each bin will have at maximum one entry assigned to any given flow. A service list is then computed by listing all the sessions in the first bin, followed by all those in the second bin, and so on, up to the W_{max}th bin.

Uniformly Interleaved WRR In this approach, the number of bins equals the least common multiple (denoted by W_{LCM}) of $\{w_i\}_{i=1}^{J}$. Session i registers itself in every $(n \times (W_{LCM}/w_i))$th bin for $1 \leq n \leq w_i$. A service list is then computed, by listing the sessions bin after bin.

WF2Q Interleaved WRR In this approach, the service list is computed by assuming that all sessions are always backlogged and determining the sequence in which the packets are transmitted in the WF2Q scheme. The service list is then set equal to this sequence.

The InfiniBand table-based scheduler

InfiniBand uses Virtual Channels (VCs) to aggregate flows with similar characteristics and the arbitration is made at a VC level. The maximum number of unicast VCs that a port can implement is 16. InfiniBand defines a scheduler that uses two tables, one for scheduling packets from high-priority VCs and another for low-priority VCs. The maximum amount of data that can be transmitted from high-priority VCs before transmitting a packet from the low-priority VCs can be configured. Each table has up to 64 entries. Each entry contains a VC identifier and a weight, which is the number of units of 64 bytes to be transmitted from that VC. This weight must be in the range of 0 to 255, and is always rounded up as a whole packet. When arbitration is needed, the table is cycled through sequentially and a certain number of packets is transmitted from the VC indicated by the VC identifier depending on the entry weight.

Chapter 4

Advanced Switching Review

Advanced Switching (AS) [Adv05] is an open-standard fabric-interconnect technology based on PCI Express [PCI03], which is already replacing the extensively used Peripheral Component Interconnect (PCI) bus. The PCI bus has served industry well for the last ten years and is currently used extensively. However, the processors and I/O devices of today and tomorrow demand much higher I/O bandwidth than PCI 2.2 or PCI-X can deliver. The reason for this limited bandwidth is the parallel bus implementation. PCI Express eliminates the legacy shared bus-based architecture of PCI and introduces an improved and dedicated point-to-point interconnect. The primary strength behind PCI Express is in its support for legacy PCI while addressing its inadequacies.

AS is an extrapolation of PCI Express, borrowing its lower two architectural layers and including an optimized transaction layer to enable essential communication capabilities like peer-to-peer communication. In this chapter, we review the AS technology, focusing on those traffic management mechanisms that can be used to provide QoS.

4.1 Introduction

The widely adopted PCI uses a parallel bus at the physical layer and a load-store-based software usage model. Since PCI's introduction, its bus frequency and width have increased to satisfy the ever-increasing I/O demands of applications. Its extension, PCI-X, is backward compatible with PCI in terms of hardware and software interfaces. PCI-X delivers higher peak I/O performance and efficiency than PCI. However, the processors and I/O devices of today and tomorrow demand much higher I/O bandwidth than PCI or PCI-X can deliver.

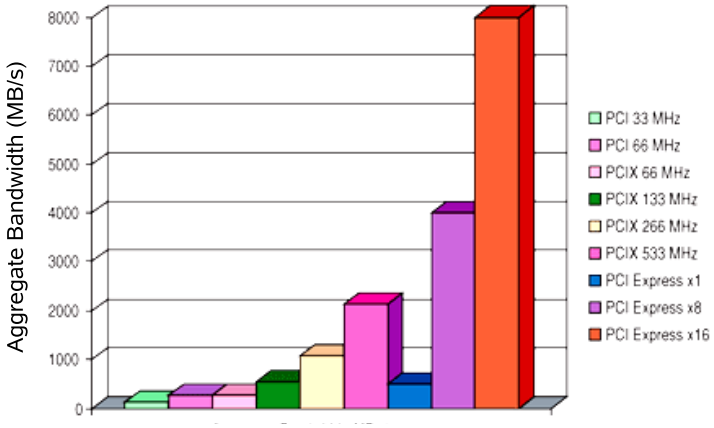


Figure 4.1: PCI, PCI-X, and PCI Express bandwidth comparisons.

In PCI and PCI-X architectures, signal skews exist in the underlying parallel physical interface; these limit bus frequency and width. Furthermore, as stated in Section 2.1.1, all the devices connected to a bus share its bandwidth. Therefore, PCI and PCI-X have limited bandwidth scalability.

Due to the widespread adoption and implementation of the PCI bus and the resulting investment in hardware and software, the industry is leaning toward an evolutionary rather than a replacement technology to protect its investments. In July 2002, the PCI Special Interest Group (PCI SIG) released the PCI Express specification v1.0 to its members. The v1.1 was published in march of 2005 [Adv05]. The specification defines a serial bus structure for chip-to-chip and add-in card applications, the functions provided today by the PCI interconnect.

PCI Express is a serial interconnect, which results in lower pin counts, lower power and full duplex transmission. It provides improvement in areas of scalability, reliability, and quality of service. Figure 4.1 shows a bandwidth comparison of the different PCI technologies. In terms of software, PCI Express is fully compatible with PCI at the application level. PCI Express addresses many of the limitations of PCI's parallel bus-based architecture and is well positioned to become the successor of the PCI interconnect for the PC, traditional server, and direct attached storage markets [Chr04]. By providing a manageable transition from PCI, PCI Express has steadily gained support from key industry players, who are making substantial product development investments.

However, the greatest asset of PCI Express, which is its compatibility with PCI, limits its use in complex systems. PCI Express is designed to operate in systems with a single host processor connected to a multitude of peripheral devices. Thus, it is limited in its

ability to handle multiprocessor applications, found in communications, storage, and blade servers, which have more sophisticated communication models, involving multiprocessing or peer-to-peer communication.

Therefore, during the development of the PCI Express specification, the industry realized that a certain class of applications would require a superset of the PCI Express features. The Advanced Switching Interconnect Special Interest Group (ASI SIG) was formed to develop a specification that would build this functionality on top of the PCI Express Physical and Data Link layers. Advanced Switching (AS) further enhances the capabilities of PCI Express by providing protocols suitable for a variety of applications, including multiprocessing and peer-to-peer computing.

In December 2003, the ASI SIG announced the approval and release of version 1.0 of the Advanced Switching core specification [Adv03]. Companies, such as Agere, Alcatel, Huawei, Intel, Siemens, Vitesse, and Xilinx were joined by other semiconductor vendors and major players in the communications and compute markets, all of which had expertise in developing advanced serial interconnects.

AS was targeted for applications such as converged servers, advanced storage, communication access/edge infrastructure, and blade servers, which until now have not been well served by industry standard interconnects. Instead, these applications have had to rely on proprietary solutions for the combination of high availability, distributed processing, QoS features, and multi Gbit/s performance.

As stated before, AS is built on the same physical and link layers as PCI Express technology. Moreover, it includes an optimized transaction layer to enable essential communication capabilities, including:

- Protocol encapsulation.
- Some mechanisms, which correctly used permit to provide QoS.
- Enhanced fail-over.
- High availability.
- Congestion and system management.

Moreover, direct unicast communication between any two nodes, or multicast communication between a source node and multiple designated destination nodes, are supported by the AS architecture.

AS provides a high level of flexibility for system architects, allowing a number of different I/O protocols to share the fabric. The protocol is identified in a header attached to the data packet. In addition, the data payload can contain either a native AS packet or encapsulate a packet in its native format, such as Ethernet, SONET, TCP/IP, or PCI Express.

In fact, the main advantage of AS lies in its innate ability to seamlessly co-exist with PCI Express devices. This is the result of having the same physical and link layers, which in turn greatly simplifies the bridging required between the two interconnects. This is a particular important feature to systems developers where ASI will serve as central switch fabric connecting PCI Express endpoints. Possible applications of this may be:

- Aggregation and dynamic reconfiguration of multiple PCI Express trees within a single switching element.
- PCI Express based endpoints virtualized across multiple hosts/processors.
- PCI Express based processors clustered together across several line cards within a box or rack.

Summing up, AS is an extrapolation of PCI Express, borrowing its lower two architectural layers from the PCI Express specification, but diverging at the transaction layer and in the marketplaces it intends to serve. Whereas PCI Express has already begun to reshape a new generation of PCs and traditional servers, AS was intended to proliferate in: Multiprocessor, peer-to-peer systems in the communications, storage, networking, servers, and embedded platform environments.

4.2 Layer architecture

As stated before, AS uses the same physical layer as PCI Express. It also shares much of the link layer. Additional Data Link Layer Packets (DLLPs) have been incorporated for the purpose of exchanging VC credit-flow control as well as congestion management messages between AS link partners. It also inherits the differentiated traffic classes (TCs) and VC concepts. Over the physical and link layers of PCI Express, AS implements an optimized transaction layer, providing a rich set of features and capabilities. The relationship between PCI Express and AS layer architecture is illustrated in Figure 4.2.

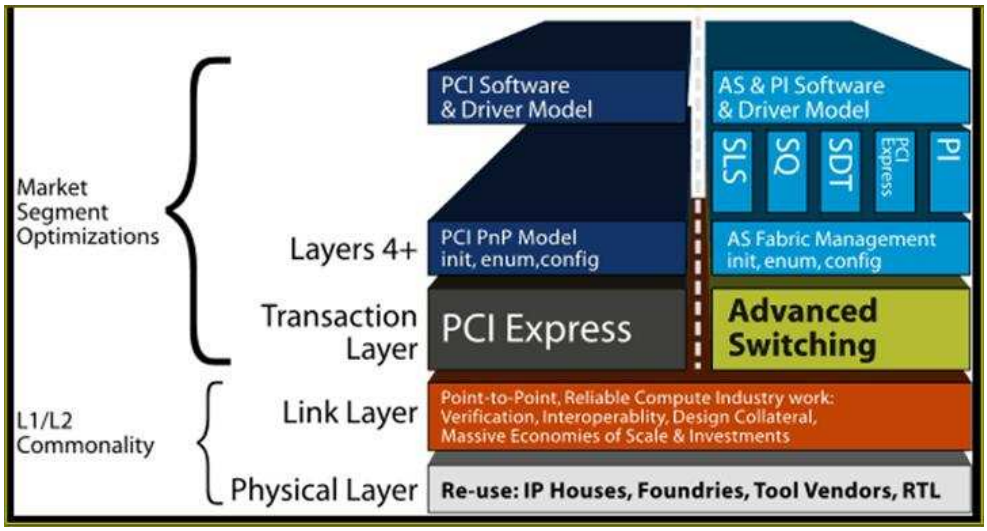


Figure 4.2: AS layer architecture.

4.2.1 Physical layer

The physical layer, which is the same that in the PCI Express interconnect, transports packets between the link layer of two AS network elements. It consists in a dual-simplex channel, which is implemented as a transmit pair and a receive pair, with an initial bandwidth of 2.5 Gb/s/direction. A data clock is embedded using the 8b/10b encoding scheme, which is also used in Fibre Channel and Gigabit Ethernet [ANS93, Sei98]. Note that with the 8b/10b encoding scheme the effective bandwidth is only 2 Gb/s/direction. Moreover, the physical layer attaches to the packets a start symbol and an end symbol.

The bandwidth of a link may be linearly scaled by adding signal pairs to form multiple lanes. The physical layer supports x1, x2, x4, x8, x16, or x32 lane widths. When several lanes are present, the data is split in bytes and each byte is transmitted, with 8b/10b encoding, across a separate lane. This data disassembly and reassembly is transparent to other layers. Note that this way of scaling the link bandwidth is different from a typical parallel approach where the bits belonging to the same byte would be transmitted using a different lane. During initialization, each AS link is set up following a negotiation of lane widths by the two agents at each end of the link. No firmware or operating system software is involved.

4.2.2 Link layer

The primary role of the link layer is to ensure reliable delivery of packets across the AS link. The link layer is responsible for data integrity and adds a sequence number and a CRC to the transaction layer. The link layer will automatically retry a packet that was signaled as corrupt.

A credit-based flow control protocol ensures that packets are only transmitted when there is enough buffer space at the other end to store them, making sure that no packet is dropped when congestion appears. This makes AS a lossless network. Flow control credits use a 64 bytes granularity. This flow control operates over all links including those between adjacent switch elements and between switch elements and endpoints (and between endpoints if there is no intervening switch).

4.2.3 Transaction layer

AS supports unicast and multicast traffic. For unicast traffic the AS transaction layer provides source-based routing versus the memory-mapped routing of PCI Express. By eliminating the top-down hierarchy with a single host structure of memory mapped routing, AS enables true peer-to-peer and multiprocessor environments in multiple topologies, including mesh, star, and dual star, which are topologies typically employed in blade servers and telecom systems. Figure 4.3 shows a simplified example of typical topologies for PCI Express and AS. Multicast routing enables a single packet generated by a source to be sent to multiple endpoints. Duplicate packets are generated at points along the fabric where the associated multicast distribution tree branches.

The packet size, when transmitting the packet between link partners, is determined by the number of bytes between the start and end symbols at the link layer. Cut-through forwarding[1] does not have the advantage of knowing the exact size of a packet until the end symbol has been received. Packets contain a *Credit Required* field to provide an indication of the size of the associated packet for cut-through routing purposes. This field indicates the number of credits necessary to hold the entire transaction layer AS packet.

The maximum size of an AS packet is 2176 bytes. Flow control credits use a 64 bytes granularity. The *Credit Required* field contains 5 bits. This results in a maximum *Credit Required* value of 34 and a maximum of 32 *Credit Required* encodings. The shortfall

[1] A node starts to send a packet before the packet has been completely received, as explained in Section 2.2.2.

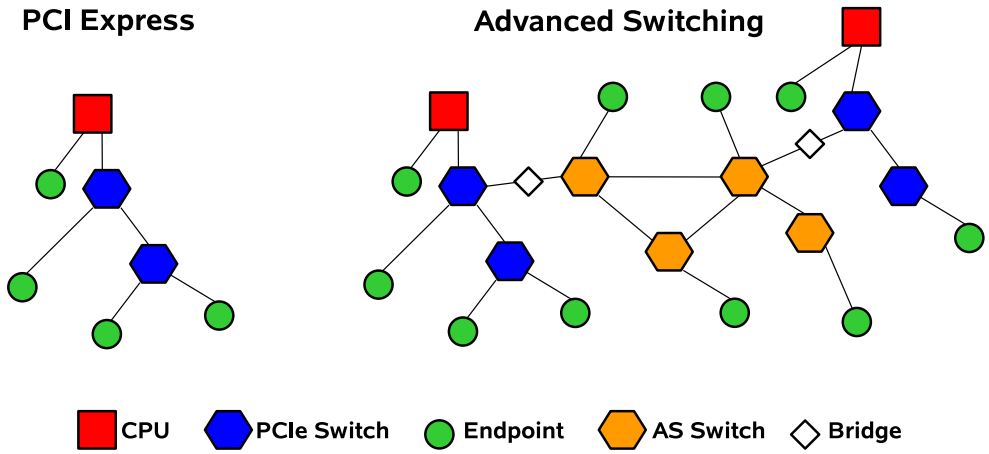


Figure 4.3: PCI Express and AS example topologies.

in credit reporting capability is handled by treating the three largest values: 32, 33, and 34 as if they are 34 for cut-through purposes.

AS encapsulates data packets and attaches a header that routes them through the fabric, regardless of the packet format. The header contains a Protocol Interface field that is used at the packet's destination to determine packet format. Thus, nearly any transport, network, or link layer protocol can be routed through an AS network. PCI Express packets are a particularly important format for system developers where AS will serve as a central switch fabric connecting PCI Express endpoints. There is a specific protocol interface designed to allow multiple-enabled PCI Express CPUs to connect transparently to multiple-enabled PCI Express I/O nodes through the AS fabric using PCI Express plug-and-play software.

4.3 Packet format and routing

Essentially, every AS packet contains two headers: One for fabric navigation (the route header) and the other for content (the Protocol Interface (PI) header). Moreover, in order to guarantee the transmission of the packet between link partners, additional information is attached to the packet by the data link layer and the physical layer of the transmitting link partner. This information is removed from the packet by the physical layer and the data link layer of the receiving link partner. Therefore, in addition to the explicit AS packet format, an AS packet at the physical layer contains a start symbol, sequence number, link CRC, and stop symbol. Figure 4.4 shows an AS packet with the physical and link layers information attached.

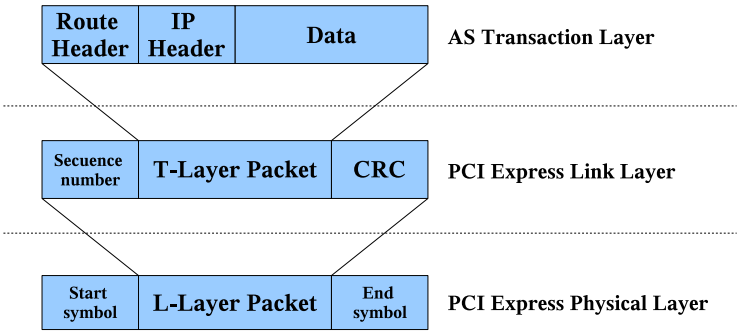


Figure 4.4: Structure of an Advanced Switching packet.

This split header model provides a great flexibility in data-carrying capabilities, both for existing protocols and for future protocols. A single AS fabric can concurrently carry an indeterminate number of independent data protocols. Moreover, the separation of routing information from the rest of the packet enables simple, high-performance and cost effective switch designs. Switches are concerned only with the routing information and, with some exceptions for path building and device management packets, do not care about the content of the payload, i.e., they are agnostic to the encapsulated protocol. There are two basic packet types for AS: Unicast packets and path-building packets. The information contained within an AS route header includes:

- Routing information (Turn Pool, Turn Pointer, and Direction).
- Traffic Class (TC).
- Deadlock avoidance information.
- Cut-Through 'Credits Required' information.
- Protocol Interface (PI) identifier.

4.3.1 Unicast packets

Unicast routing is used for sending a packet from a single origin to a single destination. AS employs source routing for unicast traffic and thus, as a unicast packet traverses through the fabric, there is no need for switches to use destination look-up tables to route the packet. This results in simpler switch design and negates latencies involved in such look-up schemes.

A unicast packet contains routing information in the form of a 31-bit turn pool, turn pointer, and direction flag. This information, which is included in the route header, is

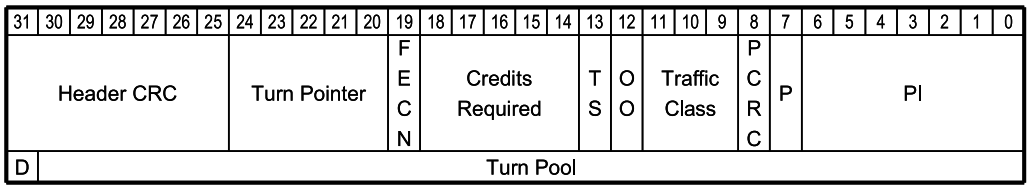


Figure 4.5: Unicast route header format.

used by switches to forward the unicast packet. The turn pool contains a certain number of *turns*. The turn value indicates the relative position of a switch's egress port from the ingress port at which the packet arrives. A turn is variable in length, ranging from 1 to 8 bits, depending on the port count of the switch immediately in its path. The turn pointer indicates which turn is currently active. Note that as a packet moves through the fabric, different turns through different switches are required to properly route the packet. For example, a packet traversing four 3-bit switches and one 4-bit switch would use 5 different turns (through 5 switches), consuming 16 of the available 31 bits in the turn pool (called the active portion of the turn pool). The remaining 15 bits would be unused. The 8-bit maximum for a turn correlates obviously to the 256-port maximum port count on an AS switch. The direction flag is used to indicate whether the packet is being forward-routed, from origin to terminus, or backward-routed, from terminus to origin.

The fields of a unicast packet header, which are shown in Figure 4.5, are:

- *Protocol Interface (PI).* This field identifies the type of the encapsulated packet.
- *Perishable (P).* The Perishable Flag indicates whether the associated packet may be silently discarded if it encounters congestion. Support for discarding packets is optional. The associated PI dictates the rules for setting this bit.
  - When 0: The packet cannot be discarded by fabric components unless they encounter a routing error within a fabric, or other error at destination.
  - When 1: Any node in a packet's path can choose to discard a packet if congestion prevents the timely forwarding of a packet.
- *Packet CRC (PCRC).* The Packet CRC Flag indicates whether a CRC has been appended to the packet's payload or the end of the PI header if no payload is included.
- *Traffic Class (TC).* This field indicates the traffic class of the associated packet.
- *Ordered-Only (OO).* The Ordered-Only flag, when set, indicates that the associated TC is to be routed through an Ordered-Only Unicast VC. When clear, this flag

indicates that the associated packet is to be routed through either the bypassable or ordered queue (depending on the value of the Type Specific flag) of a Bypass Capable Unicast VC.

- *Type Specific (TS).* The Type Specific Flag is reserved for Ordered-Only VCs and is the Bypassable flag for Bypass Capable VCs.
  - When 0: The packet consumes ordered credit and is not Bypassable.
  - When 1: The packet consumes bypass credit and is Bypassable.
- *Credits Required.* The credits required field indicates the number of credits that must be available to perform cut-through forwarding of the associated packet.
- *Forward Explicit Congestion Notification (FECN).* A packet's Forward Explicit Congestion Notification (FECN) Flag must be initialized to 0 by a packet's origin. As the packet is routed across a fabric, if the packet encounters congestion, then the FECN flag may be set. Once set, the flag remains set until it reaches its destination.
- *Turn Pointer.* For forward paths, this field references the position one greater (to the left) of the most significant bit of the next turn value (also the position of least significant bit of the previous turn). For backward paths, this field contains the position of the least significant bit of the next turn value.
- *Header CRC.* This field contains the CRC performed over the no mutable part of the route header.
- *Turn Pool.* This field contains the variable bit width turn values of a path specification.
- *Direction (D).* This flag, when clear, indicates that the Turn Pool is being traversed in the forward direction (left to right) or, when set, that the Turn Pool is being traversed in the backward direction (right to left).

4.3.2 Path building header

Path-building packets include variations for spanning tree and multicast functions. Path-building packets are constructed such that the receiving device is provided a path back through the fabric to the origin device. As a path-building packet traverses the fabric, the method that switches use to route them is based on whether the packet is further identified in its header as a spanning tree packet or a multicast packet.

Spanning tree packets, as their name says, are used for a spanning tree process. This process is initiated by a fabric manager elected from amongst various candidate devices. This process is part of an initial fabric discovery and involves a promiscuous generation, or blind broadcast of packets, which are used to identify topology, node capabilities and paths between all communicating devices, including a path from each node to the fabric manager for purposes of event/status notifications. Redundant paths are also identified during this process, but are placed into a blocked state unless needed in the case of a path failure or traffic congestion. Switches consume, then regenerate these packets to every other egress port they have, except the port at which the packet arrived (the ingress port) or any explicitly masked port. As a result, all nodes on the fabric receive spanning tree packets and are identified to the fabric manager.

The multicasting feature allows an endpoint to target a packet to multiple end systems. A multicast group index is carried on each multicast packet's route header. A multicast group uniquely identifies a set of switch egress ports for each switch hop on a multicast packet's path. A multicast group table in a switch is looked up, using packet's multicast group index. The packet is then replicated on each port contained in the multicast group. As stated before, in the process of traversal through the fabric, each multicast packet constructs a turn pool from the source, by recording turns within the switch. This provides a backward route to the multicast source end system for event notifications regarding this multicast packet.

4.3.3 Protocol interface

As stated before, an AS route header is not sufficient to define a data protocol, and must contain an encapsulated packet defined by the AS route header's PI field. The AS route header contains only enough information to manage the movement of a packet from one fabric location to another and to identify the contents (contained protocol) of the packet. Endpoints are the responsible to extract meaning from the content of the encapsulated packet payload based upon the packet's PI.

PI types are specified from PI-0 to PI-127, with PI-0 to PI-7 reserved for fabric services and PI-8 to PI-254 reserved for tunneling specific protocols. The 128 possible PIs are summarized in Table 4.1. Some specific PIs are:

- PI-2: Segmentation And Reassembly (SAR). The Maximum Transfer Unit (MTU) of an AS network is the smallest MTU supported among all the network elements. All packet sizes must be restricted according to the network MTU. If an endpoint needs

to transmit a larger packet, the packet must be split into packets of network MTU size. This involves keeping track of the multiple segments of the original packet and reassembling them at the destination.

- PI-8: PCI Express Encapsulation. This is the standard tunneling scheme for passing native PCI Express packets through the AS fabric, offering the simplicity of complete software compatibility with PCI Express peripherals within an AS environment. A system can contain a mix of PCI Express and AS components to offer the best features of both technologies.

- PI-9: Socket Data Transport (SDT). A low overhead protocol that provides direct hardware implementation of the well-known socket inter-processor communication interface's read/readv/readn and write/writev/writen data movement model. SDT will move massive amounts of data with minimal processor overhead.

- PI-10: Simple Load/Store (SLS). An extension of the PCI load/store model that offers a low overhead model for transporting data across the fabric. SLS provides a simple load/store abstraction that would allow PCI, PCI-X, PCI Express, HyperTransport, RapidIO, and virtually any other interconnect that used a load/store model to interoperate within an AS fabric via translation of their native protocol into the common SLS protocol. SLS is a trusted communication model that provides the advantages of efficiency, low overhead, and low latency.

- PI-11: Simple Queuing (SQ). A simple messaging protocol that uses queues in place of specific addresses to move messages across an AS fabric. SQ allows multiple endpoints to share a single queue resource, thus minimizing the context required for concurrent communication.

4.4 Virtual channels and traffic classes

AS fabric supports differentiated classes of service utilizing Traffic Class (TC) identifiers, Virtual Channels (VCs), and an egress link scheduling mechanism. As we will see in the next section, AS defines two egress link scheduling mechanisms. Manufacturers can choose between implement one of these mechanisms or implement their own proprietary egress link scheduler.

VCs provide a means of supporting multiple independent 'logical data flows' over a given common physical channel, i.e., the link. Conceptually, this involves multiplexing

Table 4.1: Protocol interface identifiers.

PI Index	Protocol Interface
0 (0:0) (0:1-127)	Path Building (Spanning Tree Generation) (Multicast)
1	Congestion Management (Flow ID messaging)
2	Segmentation and Reassembly (SAR)
3	Reserved for future AS Fabric Management Interfaces
4	Device Management
5	Event Reporting
6-7	Reserved for future AS Fabric Management Interfaces
8-95	ASI-SIG defined PIs
96-126	Vendor defined PIs
127	Invalid

different data flows onto a single physical Link. Packets moving through different VCs do not have any ordering requirements between them. As a result, packets moving in one VC are not subject to blocking conditions that may exist in other VCs.

A packet's Traffic Class Identifier (TC or TC ID) is transmitted unmodified from origin to destination through an AS fabric. The need for TCs arises because not all links define the same number of VCs. At each hop within an AS fabric, the TC ID contained in the packet's AS route header is used to apply appropriate VC selection. Packets with different TC IDs do not have ordering requirements between them. However, packets moving along a common path within the same VC remain ordered because the AS queue structure has no provisions for bypassing independent TCs within the same VC. As a result, packets with different TCs moving within the same single VC are subject to blocking conditions that may be caused by packets within that VC that have a different TC assignment.

AS supports up to 20 VCs of three different types: Up to 8 bypassable unicast VCs (BVCs), up to 8 ordered-only unicast VCs (OVCs), and up to 4 multicast VCs (MVCs). Table 4.2 shows a brief description of each type, the number of them that a AS element can implement and their identifiers. The bypassable VC with the highest identifier in each network element is called the Fabric Management Channel (FMC). Note that the link-level flow control is made at a VC level. This means that each VC has its own credit count for the credit-based flow control. Moreover, each VC type has its own MTU. The allowed MTU values for the bypassable VC type are 192, 320, 576, 1088, and 2176 bytes. The

Table 4.2: Advanced Switching VC Types.

Virtual Channel Type	Description	VC ID's
Bypass Capable Unicast (BVC)	Unicast VC with bypass capability, necessary for deadlock free tunneling of some, typically load/store, protocols	0-7
Ordered-Only Unicast (OVC)	Single Queue Unicast VC, suitable for message oriented "push" traffic	8-15
Multicast (MVC)	Single Queue Virtual Channel for Multicast "push" traffic	16-19

allowed MTU values for the ordered VC type are 64, 96, 128, 192, 320, 576, 1088, and 2176 bytes.

The BVCs are unicast VCs with bypass capability, necessary for deadlock-free tunneling of some protocols (typically load/store ones). This mechanism works in the following way: When a packet arrives at the VC it is stored in a FIFO queue. Once a packet reaches the head of this queue it is transmitted if there are enough flow control credits. If a bypassable packet is at the head of that queue but there are no enough flow control credits, it is moved to another queue where it waits until there are enough flow control credits. OVCs are FIFO queue unicast VCs.

This architecture with two Unicast queuing models supports robust, low latency transport of chip-to-chip protocols such as PCI and PCI Express as well as message oriented "push" protocols. These features enable ASI fabric to deliver a unified backplane solution for load/store and message based communications.

As stated before, a link-level credit-based flow control mechanism ensures that packets are never lost due to congestion. Credits are computed, per ordered queues (all VCs) and bypass queues (only BVCs), by the receiver end of the link and distributed upstream to the transmission side. Packets may only be transmitted if enough credits are available for the particular VC and queue into which the packet is grouped. Upon sending a packet, the transmission side debits its available credit account by an amount that reflects the packet size. As the receive side reclaims buffer space freed up as packets are forwarded, it returns the credits to the transmit side which in turn adds to its credit account. Link partners exchange credit information, via DLLPs.

The AS packet header contains a 3-bit field with a TC ID. This field permits to specify one of eight possible TCs. Since systems can be constructed with switches

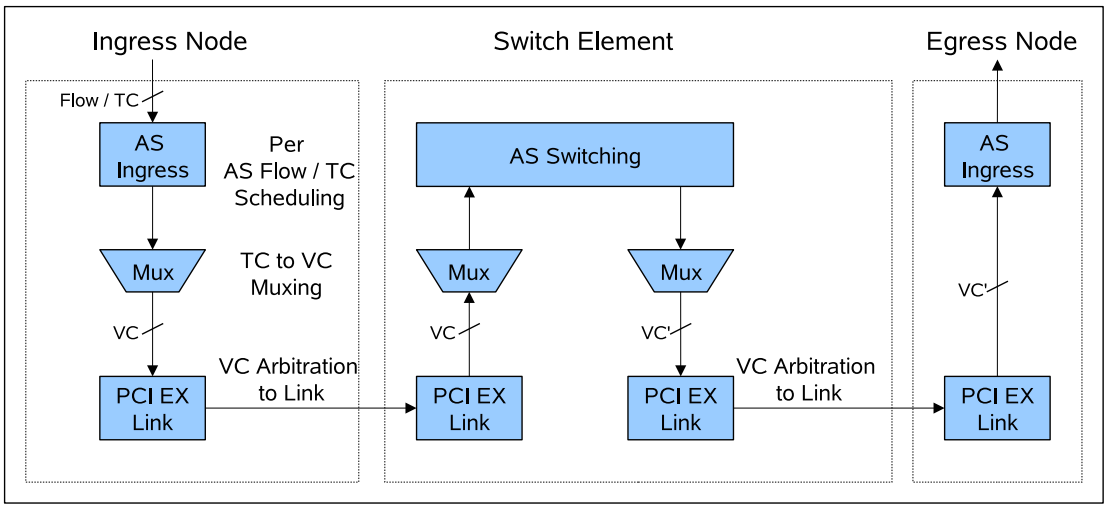


Figure 4.6: TC to VC Aggregation Model

supporting a different number of VCs, TC to VC mappings can change in each hop through a switch fabric. Thus, VCs themselves may be aggregated (when the next hop switch implements fewer VCs) and disaggregated (when the next hop switch implements more VCs). Figure 4.6 shows the TC to VC aggregation model. Each VC type (BVC, OVC, and MVC) is governed by a distinct TC/VC mapping.

4.5 Congestion management

The congestion management mechanism provided by AS tries to regulate traffic flows throughout an AS network to avoid overloading link and component capacities. Failure to successfully regulate traffic can result in excessive latency, lower throughput, and inhibit a switch's ability to provide QoS assurances. In short, an effective congestion management solution is needed if a fabric is going to support QoS and deliver predictable capacity. Even in cases where QoS is not supported, congestion management may be needed to avoid or mitigate head-of-line blocking and support expected traffic bandwidths.

The link-layer credit-based flow control and the VC architecture provide the foundation upon which AS congestion management is based. VCs provide separate logical paths for traffic, aggregating traffic flows by TC into per-VC queues. Flow control governs traffic flow between link partners on a per VC basis, modulating a link partner's ability to transmit packets based on the availability of storage (credit) on the connected link partner. AS congestion management defines several supplementary optional normative

mechanisms[2]. These enable the AS fabric developer to better manage congestion, support differentiated class-of-service, and support a diversity of applications. Table 4.3 shows a summary of the different mechanisms and their disposition (required[3], optional normative, or informative[4]).

These mechanisms are intended to provide the AS hardware the ability to support high throughput traffic with low and predictable latency. On the whole, they are reactive, responding dynamically to rapidly changing conditions within the AS fabric. Software mechanisms combined with hardware implemented within endpoint devices may also be employed to provide proactive control over the AS fabric behavior. Proactive mechanisms include admission control, provisioning, and adaptation. These mechanisms operate over longer time spans and require fabric management software and/or operator intervention. End-to-end congestion management (between endpoints) is not specifically defined within the AS specification.

Table 4.3: Congestion management mechanisms summary.

Mechanism	**Disposition**	**Comments**
Link-layer Credit-Based Flow Control	Required	The credit-based flow control is defined by PCI-Express Base Specification and adopted by AS, it may be viewed as the last line of Congestion Management defense. During congestion, flow control prevents packet losses by queuing packets in the absence of packet transfer credit.
Status-Based Flow Control	Optional Normative	This mechanism provides a one stage look-ahead view of the congestion landscape and causes the upstream egress scheduler to start/stop certain flows.
Discard of Status Feedback DLLPs	Required	If the status-based flow control sink function is not supported and enabled, then received Status Feedback DLLPs must be gracefully discarded upon reception.
MinBW Egress Link Scheduler	Optional Normative for switches	Packets from competing VC queues are selected for transmission to egress links in accordance with configured minimum bandwidth parameters.
VC Arbitration Table Scheduler	Optional Normative	The VC Arbitration Scheduler provides a packet based weighted round robin VC scheduler.

[2] An optional normative mechanisms is not required to be implemented in an AS device. If implemented, however, it must comply with the requirements specified.

[3] All required mechanisms must be implemented in an AS device.

[4] An informative mechanism is not required to be implemented in an AS device. Only informative information is given.

Table 4.3 (Continuation): Congestion management mechanisms summary.

Mechanism	Disposition	Comments
Packet Dropping	Optional Normative	An AS implementation may drop packets as a response to congestion if the perishable bit within the packet AS Route Header is set.
Endpoint Injection Rate Limiting	Optional Normative	An endpoint implementation may implement a specified form of injection rate limiting. The specified facility provides for up to 64K connection queues and a token bucket rate limiter for each queue.
Path Selection	Informative	Source endpoints should select paths through an AS fabric that are optimized to one or more criteria. For example, paths that are not congested may be preferred over congested paths.
Admission Control	Informative	Fabric management software may provide source endpoints congestion information with which they might accept/deny access to new traffic. A suitable response to congestion might be to deny new packet flows access to the switch fabric until congestion disappears.
Adaptation	Informative	Fabric management and endpoint software may move packet flows from congested paths to uncongested paths or change packet rates by adjusting token bucket average and peak rate parameters.
Disposition meaning		
Required		An AS device must implement this feature.
Optional Normative		This feature is not required to be implemented in an AS device. If implemented however, it must comply with the requirements specified.
Informative		This feature is not required to be implemented in an AS device. Only informative information is given.

4.5.1 Local status-based flow control

The status-based feedback mechanisms (referred to as status-based flow control) provide support for optimizing the flow of traffic across the links between any switch and its adjacent components. Special DLLPs pass buffer status from any switch to its immediate upstream switch or endpoint neighbor. This status information provides a one stage look-ahead view of the congestion landscape and causes the upstream egress scheduler

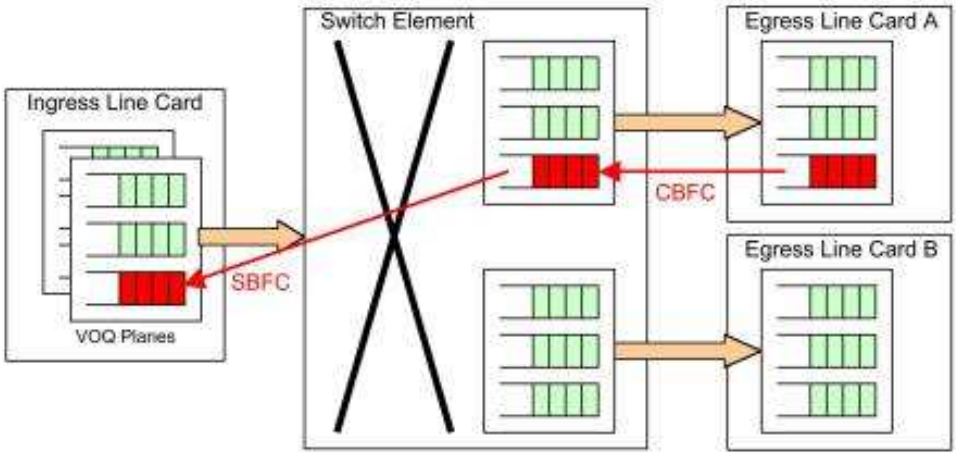


Figure 4.7: SBFC and CBFC interaction example.

to start/stop the flow identified within the DLLP. Figure 4.7 shows how this mechanism (SBFC in the figure) can be used along with the link-level flow control mechanism (CBFC in the figure) to achieve non-blocking operation in single stage switch fabrics that will be typically used in communication systems.

4.5.2 Egress link scheduling

With up to twenty VCs competing for bandwidth onto an egress link, it is the role of the Egress VC Scheduler to resolve this competition. The scheduler also handles DLLP traffic (for example the generated by the credit-based and status based flow control) and, where needed, distinguishes between the ordered and bypassable parts of the VCs. Figure 4.8 shows the structure of an egress link scheduler.

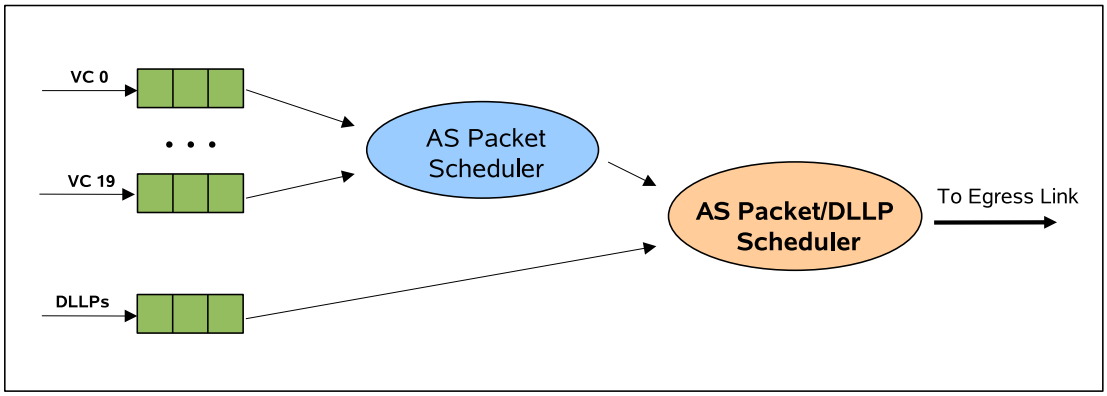


Figure 4.8: Structure of an egress link scheduler for a port with 20 VCs.

The AS Packet/DLLP Scheduler observes strict priority, allocating bandwidth exclusively to DLLPs as long as backlog exists within the DLLP queue. AS does not define a hardware mechanism to prevent DLLP traffic from starving AS Packet traffic. Rather, the AS architecture specifies DLLP sources to be well behaved AS network elements, self-limiting their DLLP generation to acceptable rates (small fraction of the link bandwidth).

Two optional normative egress link schedulers are defined for the AS Packet Scheduler[5]. The VC Arbitration Table scheduler is similar to that defined for PCI-Express. It provides packet-based Weighted Round Robin (WRR) servicing of the VCs. The Minimum Bandwidth (MinBW) scheduler is intended for more precise allocation of bandwidth, regardless of packet size, although the actual mechanism is not specified. A given implementation may choose either VC Arbitration Table scheduler, the recommended MinBW Allocation Scheduler or may implement a proprietary mechanism of its choosing.

When implementing the egress link scheduler, the interaction with the credit-based flow control must be taken into account. Packets from VCs that lack enough credits must not be scheduled. Thus, if the credits for a given VC have been exhausted, the VC scheduler must treat the corresponding queue as if it were empty. While this situation persists, the bandwidth ordinarily given to that queue is considered excess bandwidth and must be redistributed among queues for which corresponding VC credits are available.

Virtual Channel Arbitration Table Scheduler

The table scheduler provides an implementation of the WRR algorithm [KSC91]. The VC arbitration table is a register array with fixed-size entries of 8 bits. Each 8-bit table entry corresponds to a slot of a WRR arbitration period. Each 8-bit table entry contains a field of 5 bits with a VC identifier value and a reserved field of 3 bits. When arbitration is needed, the table is cycled through sequentially and a packet is transmitted from the VC indicated in the current table entry regardless of the packet size. If the current entry points to an empty VC, that entry is skipped. The number of entries may be 32, 64, 128, 256, 512, or 1024. Figure 4.9 shows an example of an arbitration table with 64 entries.

Minimum Bandwidth Egress Link Scheduler

The MinBW scheduler is intended for a more precise allocation of bandwidth regardless of the packet size. Figure 4.10 shows the organization of the MinBW scheduler. This

[5]In this work we will refer to this scheduler as the egress link scheduler without taking into account the DLLP/Packet scheduling.

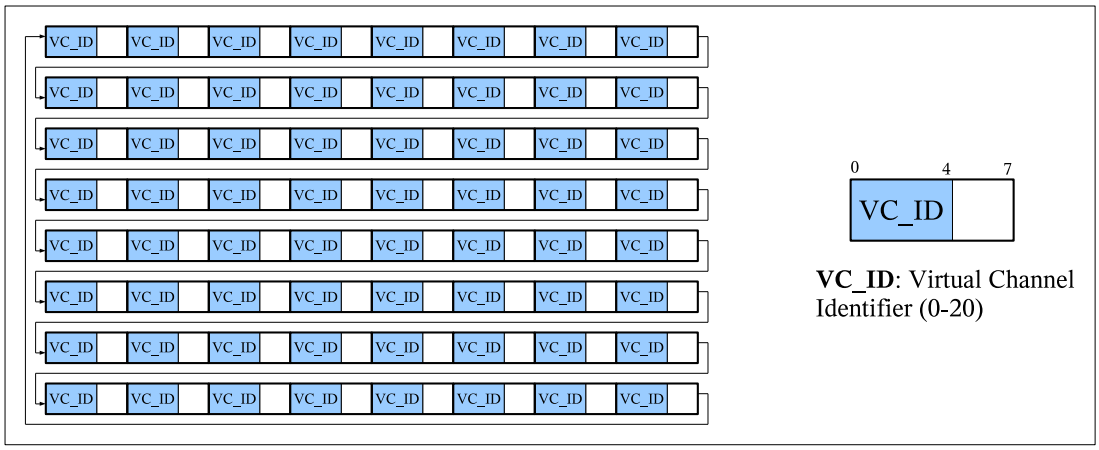


Figure 4.9: Example of an arbitration table with 64 entries.

scheduler consists of two parts: The first mechanism, or outer scheduler, provides the FMC with absolute priority, ahead of the other VCs, but with its bandwidth limited by a token bucket. The second mechanism, or inner scheduler, distributes bandwidth amongst the rest of the VCs according to a configurable set of weights. Each VC is assigned a weight between 1 and 4096, the link bandwidth fraction represented is calculated by multiplying this weigh by 1/4096. AS does not state a specific algorithm for the inner scheduler, but it must respect the following properties [Adv03]:

- Work conserving: If at least one VC has a packet available to be sent, it should be transmitted.
- Minimum bandwidth guarantee: Egress link bandwidth is allocated among the VCs in proportion to a set of configurable weights that represent the fraction of egress link bandwidth assigned to each VC.
- Bandwidth metering, not packet metering: The MinBW scheduler allocates link bandwidth to each VC taking into account packet sizes.
- Fair redistribution of unused bandwidth: Bandwidth left over, after all the VCs have consumed their configured bandwidth, must be redistributed among those VCs that have credits and packets to be transmitted in proportion to their bandwidth allocations.
- Memoryless: During the time that a VC has no packets to transmit, or credits to do so, it does not consume bandwidth and the scheduler must not save that VC's minimum bandwidth allocation for future use.

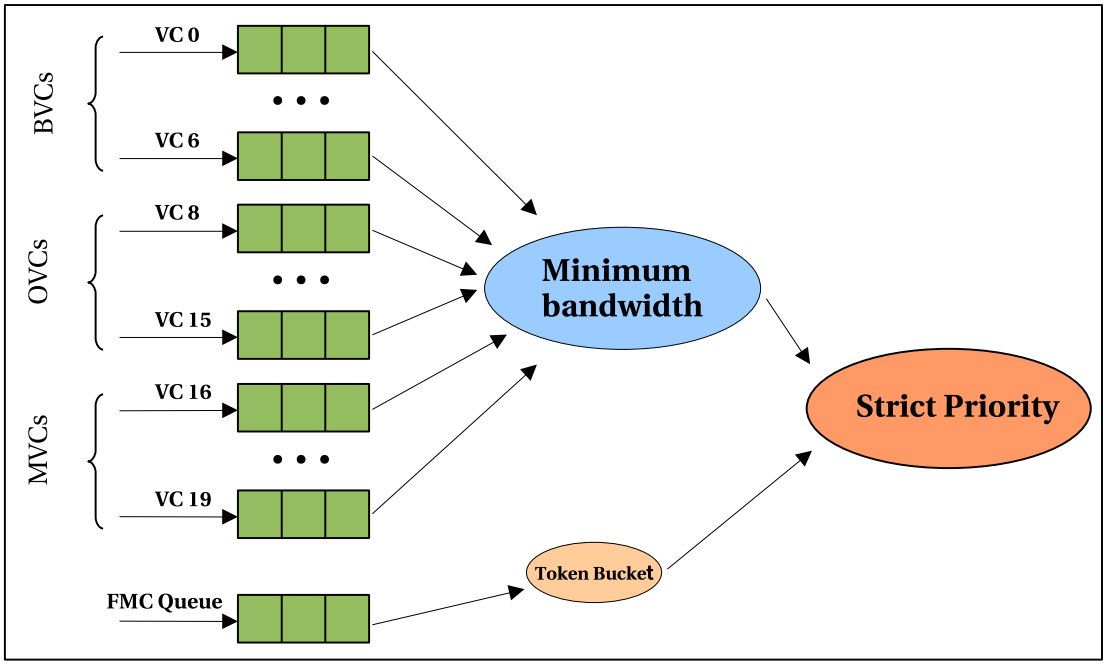


Figure 4.10: Structure of the MinBW scheduler. Example with 20 VCs.

The AS specification states that variants of Weighted Fair Queuing (WFQ) [DKS89] such as Self-Clocked Weighted Fair Queuing (SCFQ) [Gol94], and variants of Weighted Round Robin (WRR) [KSC91] such as Deficit Round Robin (DRR) [SV95] exhibit the desired properties of the inner MinBW scheduler. The AS specification also states that commonly employed scheduling algorithms, such as simple round robin or WRR, do not exhibit the desired properties of the MinBW scheduler and are, thus, not suitable for a MinBW scheduler implementation.

4.5.3 Endpoint source or injection rate limiting

At each source node, packets must be sorted into connection queues by TC and optionally by such additional criteria as destination and path. Allocation of bandwidth from each of the connection queues to the link feeding the AS fabric is controlled via a link scheduler (AS does not define this scheduler). If the AS device supports the Endpoint Injection Rate Limiting capability structure, then a token bucket must be paired with each connection queue to provide the needed rate limiting. Together the scheduler and optional token buckets shape the bandwidth from each connection queue to the AS fabric. Token buckets limit connection queues average transmission rate while allowing controlled burstiness.

Parameters associated with every connection queue/token bucket pair may be adjusted by software to constrain the bandwidth and burst size allowed of every connection queue.

4.5.4 Packet dropping

AS elements are permitted to drop packets in response to congestion. Only packets marked via the AS Route Header Perishable bit may be dropped. AS does not specify the manner in which congestion is detected. If an implementation determines that a packet with the Perishable flag set will excessively contribute to congestion, the packet may be dropped.

4.5.5 Admission control

Fabric management software may regulate access to the AS fabric, allowing new packet flows entry to the fabric only when sufficient resources are available. Fabric management software may track resource availability by monitoring (perhaps with the aid of the required and optional normative statistics counters) 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. In an implementation employing admission control, sources would only add new flows when permitted by the fabric management admission control module. Such software allows a new flow access to the AS fabric only if it can do so without creating congestion. If admitted, the software assigns a suitable bandwidth, TC and path to the traffic. As necessary, the software may reduce the bandwidth assigned to existing flows, or even terminate an existing flow, to accommodate a new flow.

4.5.6 Adaptation

Switch elements may maintain per-port and per-VC statistics (a minimal subset is required). This permits fabric management software to map the congestion. As part of a complete system, this enables the equipment operator and/or fabric management software to monitor AS fabric performance and identify chronic congestion hot spots. Once congestion hot spots are identified, several options exist. Among these are the following:

- Source rate limit parameters associated with the per-CQ token buckets may be reconfigured to reduce the average offered load to the fabric.
- New end-to-end flows may be routed along non-congested paths and existing flows may be re-routed around hot spots.

- New flows may be refused or lower priority active flows may be rate reduced to accommodate new flows with higher priority.

Chapter 5

Implementing the MinBW Scheduler

As stated in Section 4.5.2 in page 67, AS provides an optional normative scheduler called Minimum Bandwidth Egress Link Scheduler to resolve among the up to 20 VCs. The inner, or minimum bandwidth (MinBW), scheduler allocates all the bandwidth left over after the FMC is serviced. AS does not specify an algorithm or implementation for the MinBW scheduler but does impose constraints by requiring that certain properties hold. However, the AS specification states that there are several well-known scheduling algorithms that fit this model in a proper way and thus, can be used to implement the MinBW algorithm. However, the specification also states that, when implementing the egress link scheduler, the interaction with the credit-based flow control must be taken into account.

In this chapter we are going to discuss about the implementation of the MinBW scheduler. We will see that the traditional well-known scheduling algorithms, including those stated by the AS specification, must be adapted in order to be employed in this environment. Specifically, we present three new fair queuing scheduling algorithms that take into account the AS credit-based flow control and fulfill all the properties that the AS MinBW scheduler must have and, therefore, can be implemented in this new technology. These new algorithms are based on well-known scheduling algorithms (WFQ, SCFQ, and DRR). We have called these new algorithms: WFQ Credit Aware (WFQ-CA), SCFQ Credit Aware (SCFQ-CA), and DRR Credit Aware (DRR-CA).

5.1 Introduction

The Minimum Bandwidth Egress Link Scheduler consists of two parts: The first mechanism, or outer scheduler, provides the FMC with absolute priority, ahead of the other

VCs, but with its bandwidth limited by a token bucket. The second mechanism, or inner scheduler, distributes bandwidth amongst the rest of the VCs according to a configurable set of weights. Figure 5.1 shows the structure of this scheduler. AS does not state a specific algorithm for the inner scheduler, but it must respect certain properties (see Section 4.5.2 in 67): *Work conserving*, *bandwidth metering, not packet metering*, *minimum bandwidth guarantee*, *fair redistribution of unused bandwidth*, and *memoryless* [Adv05].

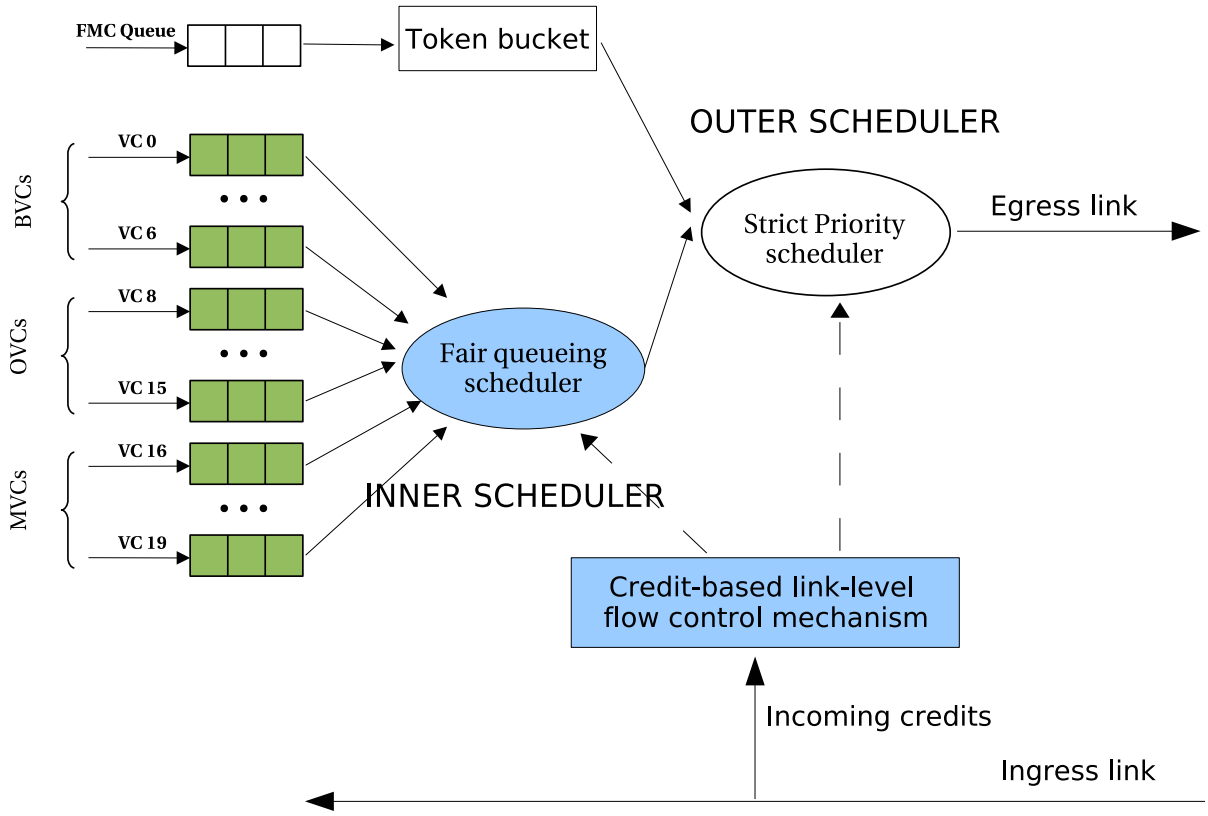


Figure 5.1: Minimum bandwidth egress link scheduler.

As stated in the AS specification, there are several well-known scheduling algorithms that fulfill these properties and thus, can be used to implement the MinBW algorithm. Specifically, the specification states that variants of WFQ such as SCFQ and variants of WRR such as DRR exhibit the desired properties. It also says that simple round robin or WRR (without deficit modifications) do not exhibit the desired properties of the MinBW scheduler and thus, are not suitable for a MinBW scheduler implementation.

In fact, analyzing the properties of the inner scheduler of the MinBW, we can state that they refer to an ideal fair-queuing model. As stated in Section 3.5.1 in page 3.5.1, in a fair-queuing system, supposing a service rate R, N flows, with the i^{th} flow having assigned a weight ϕ_i, during a given interval of time, the flow i receives a fair share bandwidth (B_i)

proportional to its weight ϕ_i:

$$B_i = \frac{\phi_i}{\sum_{j=1}^{V} \phi_j} \times R$$

where V is the set of flows with data in queue ($V \leq N$) during that interval of time.

However, apart from the previously stated properties that refer to a fair queuing behavior, the AS specification also states that, when implementing the egress link scheduler, the interaction with the credit-based link-level flow control must be taken into account. Note that in networks without a link-level flow control, packets are going to be transmitted when the scheduler decides, without taking into account if there is enough buffer at the other side of the link to store the packet or not, and thus, if the packet is going to be discarded or not. In networks in which the link-level flow control is made at the port level, like Gigabit Ethernet [Sei98], when the receiving buffers are full, the flow control is going to block the transmission from all the VCs of the port. When the flow control allows to transmit again because there is available space, all the VCs can transmit again as if nothing had happened.

However, in AS both the scheduling and the link-level flow control are made at the VC level. This means that the flow control can block a set of VCs because there is not enough buffer to store more packets from those VCs, but allows to transmit packets from the rest of VCs, which have enough buffer to store more packets. This means that the scheduler must have the ability to enable or disable the selection of a given VC based on the flow control information. In this situation:

- A VC is considered active only if:
  - The VC has some packet to be transmitted.
  - The link-level flow control mechanism allows to transmit packets from the VC. In the case of a credit-based flow control, this means that there are enough flow control credits to transmit the packet at the head of the VC queue.
- A VC can change its status from inactive to active when:
  - A new packet arrives at the VC queue.
  - A flow control packet with new flow control credits for the VC is received.
- A VC can change its status from active to inactive when:

– A packet is transmitted into the egress link, leaving the VC, after being selected by the scheduler. The change of status would actually happen if there is no other packet in that VC or there are one or more packets, but there are not enough credits to transmit the packet at the head of the queue.

The problem of most well-known scheduling algorithms for implementing the MinBW scheduler, including those that AS states as appropriate, is that they were designed without taking into account the existence of a flow control mechanism, and thus, they do not consider the possibility of having a subset of VCs with packets to transmit but without permission of the link-level flow control mechanism to do so. The reason is that they were originally proposed for networks that do not have link layer flow control, for example Internet or ATM. Note that, if a given VC has no enough credits to transmit the packet at the head of its queue and the scheduling mechanism only disables the selection of this VC, some problems may arise. For example, let us consider a "sorted-priority" algorithm like the WFQ and SCFQ schedulers in which each packet is stamped with a priority tag when it arrives at the scheduler. Packets are usually transmitted in an increasing order of this tag. However, in a certain moment a VC is disabled because of lack of flow control credits. Packets from the rest of VCs are going to be transmitted even if the values of their tags are bigger than the values of the packet tags of the blocked VC. Note that the packet tags belonging to the disabled VC are going to remain the same during all the blocked period. When the disabled VC achieves enough credits to transmit again, the tags of its packets are probably going to be smaller than the packet tags of the rest of VCs and thus, this VC is going to transmit several packets before the rest of VCs can transmit any other packet.

This situation is represented in Figures 5.2 and 5.3. These figures show the packets transmitted from two VCs (plotted in red and green) that have the same bandwidth reservation. In these figures we can see how the traffic of the red VC takes advantage over the green VC for the time that it has been disabled because of lack of flow control credits. This is going to produce a burst of red packets that is going to affect negatively the performance of the green traffic. In the case represented in Figure 5.2, which shows that the green VC is able to use the bandwidth left over by the red VC, this burst is probably going to affect the latency and jitter of the green traffic. In the case represented in Figure 5.3, the green traffic arrives at the scheduler at the same rate that it is transmitted and thus, is not able to take advantage of the bandwidth left over by the red VC. In this last case, the red burst is going to affect negatively not only the latency and jitter of the green VC but also its throughput performance.

Therefore, a blocked VC should not take advantage of the time that has been disabled by the flow control. In this line, regarding the interaction between the egress

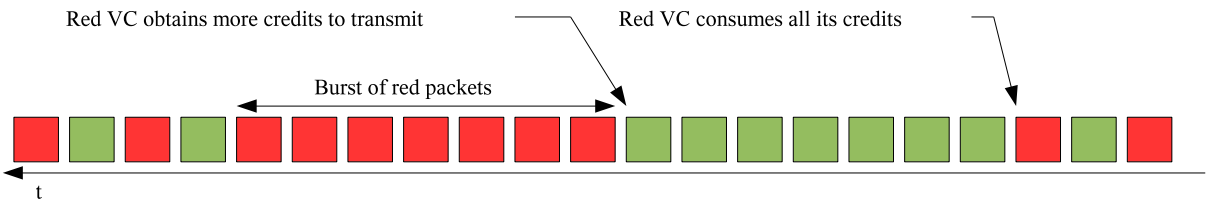


Figure 5.2: Example of a VC taking advantage of the time it has been blocked. Well behaved VC uses bandwidth left over by the blocked VC.

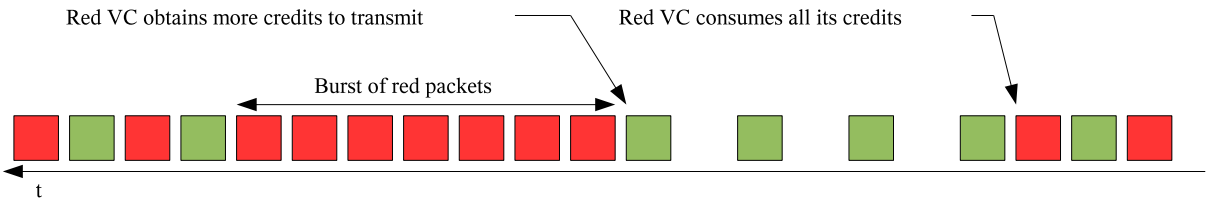


Figure 5.3: Example of a VC taking advantage of the time it has been blocked. Well behaved VC does not use bandwidth left over by the blocked VC.

link scheduling and the credit-based flow control mechanism, the AS specification states that: *Packets must not be scheduled to VCs that lack sufficient link-layer credit. Thus, if link-layer credit for a given VC has been exhausted, the VC scheduler must treat the corresponding queue as if it were empty. The bandwidth ordinarily given to that queue is considered excess bandwidth and must be redistributed among queues for which corresponding VC credit is available. Packets may not be scheduled from that queue until link credits become available.* Moreover, the *memoryless* property of the MinBW schedulers states that: *A non-backlogged queue does not consume bandwidth. During this time, the scheduler must not 'save' that queue MinBW allocation for future use. So, if a queue becomes backlogged after being non-backlogged for a while, it is only allotted its MinBW and its fair share of the excess bandwidth. It does not get to 'catch up' by also using the opportunities missed when it was non-backlogged.*

Figures 5.4 and 5.5 show the same scenario as Figures 5.2 and 5.3 except that the red VC does not employ the bandwidth that it lost when it was disabled by lack of credits. In this case we can see that the green VC takes advantage of the bandwidth left over by the red VC if it has the opportunity. However, the red VC, which we can consider bad-behaved, does not affect negatively the performance of the green VC, which we can consider well-behaved.

Summing up, if the scheduling mechanism does not take into account in a proper way the interaction with the flow control mechanism, the performance of those VCs that are not disabled by lack of flow control credits can be negatively affected by those flows that are disabled by the flow control mechanism. This negative effect violates the property

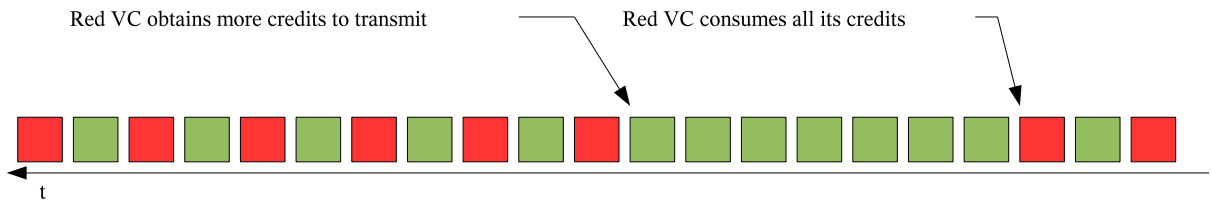


Figure 5.4: Example of a VC that does not takes advantage of the time it has been blocked. Well behaved VC uses bandwidth left over by the blocked VC.

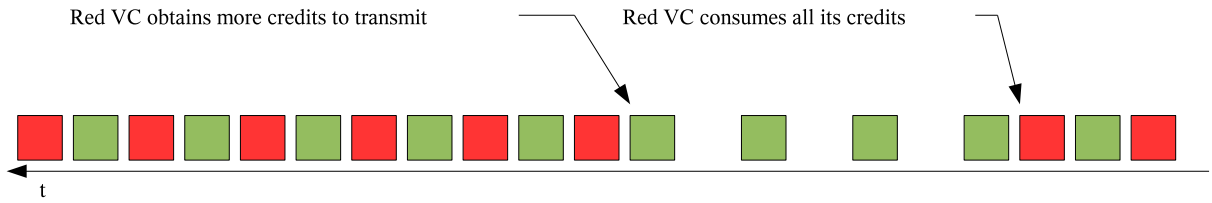


Figure 5.5: Example of a VC that does not takes advantage of the time it has been blocked. Well behaved VC does not use bandwidth left over by the blocked VC.

of protection that a scheduling algorithm should possess [Zha95] (see Section 3.5). Therefore, one of the main issues to consider when implementing the MinBW scheduler is its interaction with the AS credit-based flow control. As stated before, a given implementation of a scheduler is not allowed to select packets from a VC lacking transmission credits, nor it is allowed to 'save' this bandwidth for future use. However, most well-known scheduling algorithms do not take into account this issue. Moreover, most of them were designed to schedule among a high number of individual flows. In AS the scheduler must resolve only between up to 20 VCs. This last consideration permits us to simplify some scheduling algorithms.

Therefore, in order to employ well-known scheduling algorithms to implement the inner scheduling mechanism of the MinBW scheduler, those well-known algorithms must be adapted. Providing this behavior to the strict priority mechanism is not difficult, however, this issue is not so trivial for the fair queuing mechanism. The main aspects that must be taken into account to adapt these algorithms are:

- The simplifications that can be made when considering just a specific, small set of VCs instead of an undetermined, big set of flows.
- The proper interaction with the flow-control mechanisms. Specifically, in order to not 'save' bandwidth of inactive VCs for future use.

In the following sections we present three new fair queuing scheduling algorithms that take into account the AS credit-based flow control and fulfill all the properties that

the AS MinBW scheduler must have and, therefore, can be implemented in this technology. These new algorithms are based on some of the previously named well-known scheduling algorithms (WFQ, SCFQ, and DRR). We have called these new algorithms: WFQ Credit Aware (WFQ-CA), SCFQ Credit Aware (SCFQ-CA), and DRR Credit Aware (DRR-CA).

5.2 Weighted fair queuing credit aware (WFQ-CA)

The WFQ-CA scheduler is based on the WFQ scheduler [DKS89], which we have reviewed in Section 3.5.1. The WFQ-CA algorithm that we propose works in the same way as the WFQ algorithm, except in the following aspects:

- The GPS virtual time *V(t)* is actualized when one of the following events occurs:
  - A new packet is received by the scheduler in the egress link queues and must be stamped with its timestamp.
  - A packet has finished to be transmitted and the VC to which it belongs becomes inactive.
  - A flow control packet with new flow control credits is received and a packet that was inactive due to lack of flow control credits becomes active.
- When a new packet arrives at a VC queue, if there are enough credits to transmit the packet that is at the head of the VC, the new packet is stamped with its *virtual finishing time*.
- Packets are transmitted in an increasing order of timestamp, but only packets at the head of their queue and with enough credits to be transmitted are considered.
- When a VC is inactive because of lack of credits and receives enough credits to be able to transmit again, its packets are restamped, from the head to the tail, as if they had arrived in that instant.

These considerations take into account the interaction with the link-level flow control mechanism.

Furthermore, another aspect that must be taken into account is that the WFQ algorithm uses the real time to calculate the virtual time. Note that the real time includes the time used to transmit packets from the FMC VC (usually control packets), which are out of the control of the WFQ algorithm. The WFQ-CA algorithm fixes this problem by

not taking into account the time employed in sending control packets for calculating the virtual time. However, this is not a trivial task because events still may happen during that time. An event is anything that changes the scheduler state, namely the arrival or departure of a packet, or the arrival of a credit flow control message that changes a queue from inactive to active.

Figure 5.6 shows an example of how the $V(t)$ is calculated. The figure shows 7 events occurring in the system and two "gaps" (shadowed boxes) in the time line due to the transmission of control packets. The t line represents the real time of the system. The t' line represents the time that is actually being used to calculate $V(t)$ and when the events are considered to happen. Note that the events that happen during a time gap are considered to happen at the beginning of that gap.

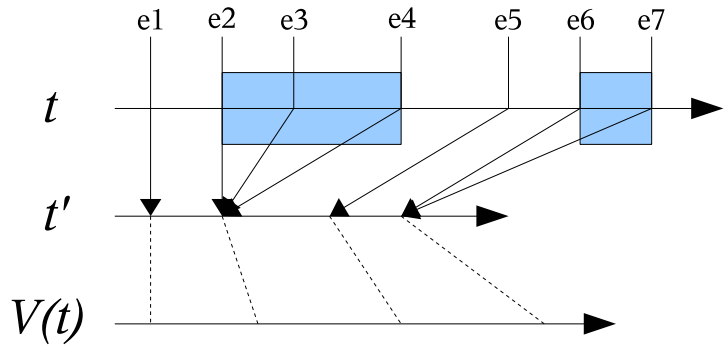


Figure 5.6: Time line in the MinBW WFQ-CA implementation.

If we compare the complexity of the WFQ-CA algorithm that we propose for implementing the AS MinBW scheduler with the original WFQ algorithm, it must be taken into account that in AS the scheduling is made at a VC level. This involves, for example, that the tag sorting process is much simpler than in other environments, where each flow is considered separately. In AS, the scheduler must consider only the packets at the head of each active VC. Only when a packet from a given VC is transmitted, the next packet in the same VC may be inserted in the sorted list of eligible packets (if they have enough credits to be transmitted). Therefore, in AS the maximum number of packets that the scheduler must take into account is twenty, which is the maximum number of VCs. Note that, in those environments where the scheduling is made at a flow level, the maximum number of packets that must be considered would be extremely higher than when using VCs. The other consideration when comparing the WFQ algorithm with the WFQ-CA algorithm is that the WFQ-CA algorithm adds the complexity of the restamping process, which may be a very costly process. This last issue may make this scheduler unfeasible in high-performance networks. This is the reason because we also propose the SCFQ-CA

and DRR-CA algorithms. Moreover, the WFQ-CA algorithm is useful for performance evaluation comparisons.

5.3 Self-clocked fair queuing credit aware (SCFQ-CA)

The SCFQ-CA scheduler is based on the SCFQ scheduler [Gol94], which we have reviewed in Section 3.5.1. As stated in that section, in the SCFQ algorithm each packet is stamped with a service tag that is computed as follows:

$$S_i^k = max\{S_i^{k-1}, S_{current}\} + \frac{L_i^k}{\phi_i}$$

Note that, $S_{current}$ is the service tag of the packet currently being transmitted and thus, the service tag of the packets that have already been transmitted is equal to or lower than $S_{current}$. Moreover, the service tag of the packets that have not already been transmitted are equal or bigger than $S_{current}$. Therefore, if the k^{th} packet of the VC i arrives at an empty queue, the service tag is computed as:

$$S_i^k = S_{current} + \frac{L_i^k}{\phi_i}$$

On the other hand, if the k^{th} packet of the VC i arrives at a queue with more packets, the service tag is computed as:

$$S_i^k = S_i^{k-1} + \frac{L_i^k}{\phi_i}$$

This means that once that there is at least one packet in a VC queue, the value of the service tags of the packets that arrive after this first packet depends only on the value of the precedent service tags and not on the value of $S_{current}$ at the arrival time. Therefore, we can wait to stamp a packet p_i^k with its service time until the packet that is before it in the VC queue, p_i^{k-1}, is being transmitted. Note that at this time the $S_{current}$ is equal to S_i^{k-1}.

This allows us to simplify in a high degree the original SCFQ algorithm by storing not a service tag per packet, but a service tag per flow or VC. This service tag represents the service tag of the packet of the VC queue. Note that this is going to make much easier and simpler to modify this algorithm to take into account a link-level flow control

mechanism. Each VC service tag is then computed as:

$$S_i = S_{current} + \frac{L_i^{first}}{\phi_i}$$

where L_i^{first} is the size of the packet at the head of the i VC.

The SCFQ-CA algorithm that we propose works in the same way as the SCFQ algorithm, except in the following aspects:

- Each active VC has associated a service tag.
- When a new packet arrives at a VC queue, that VC is assigned a service tag only if the arrived packet is at the head of the VC and there are enough credits to transmit it.
- When a packet is transmitted, if there are enough credits to transmit the next packet, the VC service tag is recalculated.
- When a VC is inactive because of lack of credits and receives enough credits to transmit again, the VC is assigned a new service tag.

The resulting scheduling algorithm is represented in the pseudocode shown in Figure 5.7.

As in the WFQ-CA case, the SCFQ-CA algorithm is simpler because in AS we consider VCs instead of single flows. Moreover, assigning a service tag per VC instead of per packet allows us to simplify in a high degree the scheduler management and also allows us to avoid the restamping process of the WFQ-CA algorithm.

However, the SCFQ algorithm still has the problem of the increasing tag values and the possible overflow of the registers used to store these values. Therefore, we propose a modification to the SCFQ scheduler that makes impossible this overflow. This modification consists in subtracting the service tag of the packet currently being transmitted to the rest of service tags. If we consider only a tag per VC, this means to subtract the service tag of the VC to which the packet being transmitted belongs to the rest of VCs service tags.

This limits the maximum value of the service tags while still maintaining the absolute differences among their values. This also means that $S_{current}$ is always equal to zero and thus,

$$S_i = \frac{L_i^{first}}{\phi_i}$$

```
PACKET ARRIVAL(newPacket,flow):
if (newPacket is at the head in the queue of flow) and
        (The flow control does allow transmitting from flow))
                flow_serviceTag ← currentServiceTag + flow_sizeFirst / flow_reservedBandwidth

ARBITRATION:
while (There is at least one active flow)
        selectedFlow ← Active flow with the minimum serviceTag
        currentServiceTag ← selectedFlow_serviceTag
        Transmit packet from selectedFlow
        if ((There are more packets in the queue of selectedFlow) and
                (The flow control does allow transmitting from selectedFlow))
                selectedFlow_serviceTag ← currentServiceTag + selectedFlow_sizeFirst / selectedFlow_reservedBandwidth
        else
                selectedFlow_serviceTag ← 0
                if (There are no active flows)
                        currentServiceTag ← 0
```

Figure 5.7: Pseudocode of the SCFQ-CA scheduler.

Moreover, the service tags are limited to a maximum value max_S: $max_S = \frac{MTU}{min_\phi}$ where MTU is the maximum packet size and min_ϕ is the minimum possible weight that can be assigned to a VC. The resulting SCFQ-CA scheduling algorithm is represented in the pseudocode shown in Figure 5.8. Note that this last modification adds the complexity of subtracting to all the service tags a certain value each time a packet is scheduled. This makes this modification feasible in hardware only when a few number of VCs is considered, like in the AS case.

5.4 Deficit round robin credit aware (DRR-CA)

The DRR-CA scheduler is based on the DRR scheduler [SV95], which we have reviewed in Section 3.5.1. The problem of the DRR scheduler when interacting with a link-level flow control mechanism is that, when we do not allow the selection of a flow or VC because of lack of flow control credits, if we still continue accumulating quantum for this VC in each round, then the blocked VC is going to take advantage of the time that has been blocked. In order to solve this problem, the DRR-CA algorithm that we propose works in the same way as the DRR algorithm, except in the following aspects:

PACKET ARRIVAL(newPacket,flow):
if (*newPacket* is at the head in the queue of $flow$) and
    (The flow control does allow transmitting from $flow$))
        $flow_{serviceTag} \leftarrow \frac{flow_{sizeFirst}}{flow_{reservedBandwidth}}$

ARBITRATION:
while (There is at least one active flow)
    $selectedFlow \leftarrow$ Active flow with the minimum $serviceTag$
    $currentServiceTag \leftarrow selectedFlow_{serviceTag}$
    Transmit packet from selectedFlow
    $\forall$ active $flow$
        $flow_{serviceTag} \leftarrow flow_{serviceTag} - currentServiceTag$
    if ((There are more packets in the queue of $selectedFlow$) and
        (The flow control does allow transmitting from $selectedFlow$))
        $selectedFlow_{serviceTag} \leftarrow \frac{selectedFlow_{sizeFirst}}{} selectedFlow_{reservedBandwidth}$

Figure 5.8: Pseudocode of the improved SCFQ-CA scheduler.

- A VC queue is considered active only if it has at least one packet to transmit and if there are enough credits to transmit the packet at the head of the VC.
- When a packet is transmitted, the next active VC is selected when any of the following conditions occurs:
  - There are no more packets from the current VC or there are not enough flow control credits for transmitting the packet that is at the head of the VC. In any of these two cases, the current VC becomes inactive, and its deficit counter becomes zero.
  - The remaining quantum is less than the size of the packet at the head of the current VC. In this case, its deficit counter becomes equal to the accumulated weight in that instant.

The resulting algorithm is expressed in the pseudocode shown in Figure 5.9.

If we compare the complexity of the DRR and DRR-CA algorithms, the main difference is that in the case of the DRR-CA algorithm the number of queues is equal to the number of VCs instead of the number of flows, and thus the complexity is even smaller. The only added complexity remains in taking into account the flow control in order to consider active or inactive a VC.

```
while (There is at least one active flow)
    if ((selectedFlow is not active) or (selectedFlow_sizeFirst > totalQuantum))
        deficitCounter_selectedFlow ← totalQuantum
        selectedFlow ← Next active flow
        totalQuantum ← deficitCounter_selectedFlow + quantum_selectedFlow
    totalQuantum = totalQuantum − selectedFlow_sizeFirst
    Transmit packet from selectedFlow
    if ((There are no packets in the queue of selectedFlow) or
        (The flow control does not allow transmitting from selectedFlow))
        totalQuantum ← 0
```

Figure 5.9: Pseudocode of the DRR-CA scheduler.

5.5 Summary

In this chapter we have highlighted the considerations and problems that must be taken into account when implementing the MinBW scheduler. Specifically, the interaction with the AS link-level flow control. The problem is that most well-known scheduling algorithms were designed without taking into account this. Therefore, we have presented three new fair queuing scheduling algorithms that take into account the AS credit-based flow control and fulfill all the properties that the AS MinBW scheduler must have and, therefore, can be implemented in this technology. These new algorithms are based on some of the previously named well-known scheduling algorithms (WFQ, SCFQ, and DRR). We have called these new algorithms: WFQ Credit Aware (WFQ-CA), SCFQ Credit Aware (SCFQ-CA), and DRR Credit Aware (DRR-CA).

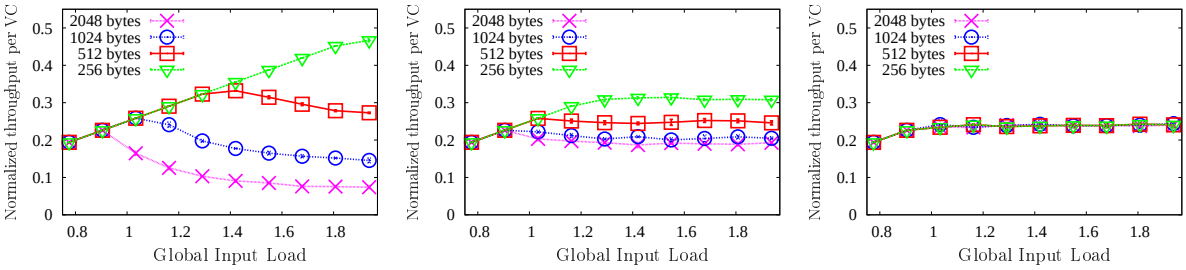
Chapter 6

The Deficit Table Scheduler

The main problem of the table-based schedulers mentioned in Section 3.5.2, including the AS table scheduler, is that they do not work in a proper way with variable packet sizes, as it is common in actual traffic. As we will show, if the average packet size of the flows[1] is different, the bandwidth the flows obtain may not be proportional to the number of table entries. We have proposed a new table-based scheduler that solves this problem [MAS06]. As far as we know, a table-based scheduler that is able to handle in a proper way variable packet sizes had not yet been proposed. We have called this scheduling algorithm Deficit Table scheduler, or just DTable scheduler, which is a mix between the already proposed table-based schedulers and the DRR algorithm. Table-based schedulers also face the problem of bounding the bandwidth and latency assignments. The number of table entries assigned to a flow determines the bandwidth assigned to that flow and the bandwidth that it has assigned determines the latency performance. If we want a flow to have a better latency performance we must assign it more bandwidth. This produces a waste of resources in some cases. In [MAS06] we also proposed a methodology to configure the DTable scheduler in such a way that it permits us to attend the bandwidth and latency requirements of the traffic with a certain degree of independence.

In this chapter, we review the DTable scheduling mechanism and its decoupling configuration methodology. Moreover, we show several possibilities in order to adapt the existing AS table scheduler into the DTable scheduler without modifying too much the AS specification.

[1] In this thesis we will use the term *flow* to refer both to a single flow or to an aggregated of several flows with similar characteristics.



(a) Basic table (AS) (b) Weighted table (IBA) (c) Deficit table

Figure 6.1: Performance of several table-based schedulers for flows with different packet size.

6.1 The DTable scheduling mechanism

As stated before, the table-based schedulers that have been proposed until now do not work in a proper way with variable packet sizes. Figure 6.1 shows the performance of various table-based schedulers, considering four Virtual Channels (VCs) in the network. Note that we use VCs to aggregate flows with similar characteristics and the arbitration is made at a VC level, as it is the case in AS or IBA technologies. In the example, the four VCs have the same number of assigned table entries (the same bandwidth reservation). Moreover, we inject an increasing amount of traffic at the same rate in all the VCs. However, the traffic injected in each VC has a different packet size. Note that in the figures we refer each VC according to the packet size that the flows associated to that VC use. The simulated architecture is the same as that used for the performance evaluation in Chapter 9.

Figure 6.1(a) shows the case of the AS table scheduler, which is cycled through. When a table entry is selected, a packet from the VC indicated in that entry is transmitted regardless of the packet size. As can be observed, when using the basic table scheduler, the VCs obtain a very different bandwidth because the traffic that traverses each VC has a different packet size. Therefore, although the same number of packets from each flow will be transmitted, the amount of information will not be the same.

The IBA's arbitration table works in a similar way than the AS table. However, it adds a weight to each entry. This weight indicates the amount of information to be transmitted from the VC associated to the table entry each time that the entry is selected. This weighted table solves the problem only partially because it allows a packet to be transmitted even requiring more weight than the remainder of a given table entry (exhausting it). Figure 6.1(b) shows the performance of a weighted table that works in this way. We have assigned all the entries the same weight: 2176 bytes (34 units of 64 bytes). As can

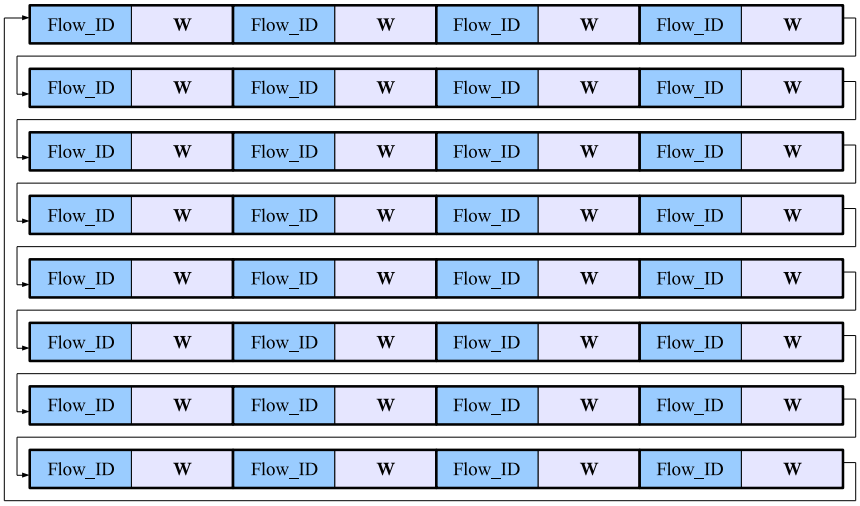


Figure 6.2: Example of an arbitration table with 32 entries for the DTable scheduler.

be seen, this scheduler presents a better performance than the basic table scheduler, but not an optimum performance.

We have proposed a new table-based scheduling algorithm that works properly with variable packet sizes [MAS06] (as can be seen in Figure 6.1(c)). We have called this algorithm Deficit Table scheduler, or just DTable scheduler, because it is a mix between the previously proposed table-based schedulers and the DRR algorithm. Our scheduler works in a similar way than the DRR algorithm but instead of serving packets of a flow in a single visit per frame, the quantum associated to each flow is distributed throughout the entire frame.

This new table-based scheduler defines an arbitration table in which each table entry has associated a flow identifier and an *entry weight*, which is usually expressed in flow control credits in networks with a credit-based link-level flow control (like AS and IBA). Moreover, each flow has assigned a *deficit counter* that is set to 0 at the beginning. Figure 6.2 shows an example of an arbitration table with 32 entries.

When scheduling is needed, the table is cycled through sequentially until an entry assigned to an active flow is found. A flow is considered active when it stores at least one packet and the flow control allows that flow to transmit packets. When a table entry is selected, the *accumulated weight* is computed. The accumulated weight is equal to the sum of the deficit counter for the selected flow and the current entry weight. The scheduler transmits as many packets from the active flow as the accumulated weight allows. When a packet is transmitted, the accumulated weight is reduced by the packet size.

The next active table entry is selected if the flow becomes inactive or the accumulated weight becomes smaller than the size of the packet at the head of the queue. In the first case, the remaining accumulated weight is discarded and the deficit counter is set to zero. In the second case, the unused accumulated weight is saved in the deficit counter, representing the weight that the scheduler owes the queue.

This behavior is represented in the pseudocode shown in Figure 6.3. Note that when using the scheduling algorithm the bandwidth assigned to the i^{th} flow ϕ_i with an arbitration table of N entries is:

$$\phi_i = \frac{\sum_{j=0}^{J} weight_j}{\sum_{n=0}^{N} weight_n}$$

where J is the set of table entries assigned to the i^{th} flow and *weight* is the entry weight assigned to a table entry.

```
while (There is at least one active flow)
    if ((selectedFlow is not active) or (selectedFlow_sizeFirst > accumulatedWeight))
        deficitCounter_selectedFlow ← accumulatedWeight
        tableEntry ← Next table entry assigned to an active flow
        selectedFlow ← tableEntry.flowIdentifier
        accumulatedWeight ← deficitCounter_selectedFlow + tableEntry.weight
    accumulatedWeight = accumulatedWeight − selectedFlow_sizeFirst
    Transmit packet from selectedFlow
    if ((There are no packets in the queue of selectedFlow) or
        (The flow control does not allow transmitting from selectedFlow))
        accumulatedWeight ← 0
```

Figure 6.3: Pseudocode of the DTable scheduler.

In order to keep the computational complexity low, we set the minimum value that a table entry can have associated to the Maximum Transfer Unit (MTU) of the network. This is the smallest value that ensures that there will never be necessary to cycle through the entire table several times in order to gather enough weight for the transmission of a single packet. This means that each time an entry from an active flow is selected, at least one packet is going to be transmitted from that flow. Note that this consideration is also made in the DRR algorithm definition [SV95]. Note also that in the IBA table-based

scheduler this issue is solved by rounding up to a whole packet the remaining weight in a table entry.

6.2 Providing QoS with the DTable scheduler

The easiest way of employing the DTable scheduler would be to assign all the table entries the same weight. This weight would be the general MTU of the network. In this case, the bandwidth assigned to the i^{th} flow, which has assigned n_i table entries, is: $\phi_i = n_i/N$, where N is the total number of entries of the table. Therefore, if we want to provide bandwidth requirements, we must assign each flow a number of table entries proportional to the bandwidth that we want to assign to that flow. Note that if we distribute all the entries belonging to the same flow in a consecutive way in the arbitration table, the performance of the scheduler is going to be similar to the DRR scheduler. As stated before, depending on the situation, the DRR algorithm can offer a bad latency performance [SV98]. Therefore, if we want to improve the latency performance provided by this scheduler, we can distribute the table entries as the WF2Q variant of the list-based Weighted Round Robin proposed by Chaskar and Madhow [CM03].

However, following the Chaskar and Madhow [CM03] approach we cannot differentiate among different levels of latency requirements. The WF2Q emulation tries to provide the best latency performance for all the flows given the amount of bandwidth that each flow has assigned. On the other hand, in [ASD04], the approach is different. Instead of having a set of flows with different bandwidth requirements and trying to provide all of them with the best possible latency, flows present different latency requirements and the table is filled in such a way that their requirements are achieved. In [ASD04], it is shown (in that case for InfiniBand) that controlling the maximum separation between any consecutive pair of entries assigned to the same flow, it is possible to control the latency of that flow. This is because this distance determines the maximum time that a packet at the head of a flow queue is going to wait until being transmitted. Note that this explains the different latency properties of the list-based WRR schedulers.

However, setting the distances among the table entries depending on the latency requirements faces the problem of bounding the bandwidth assignment to the latency requirements. If a maximum separation between any consecutive pair of table entries of a flow (or aggregated of flows with the same maximum separation requirement) is set, a certain number of them are being assigned, and hence a minimum bandwidth, to the flow in question. If the flow requires more bandwidth, we can assign more entries. However, to

Table 6.1: Arbitration table parameters.

$max\phi_i$	Maximum bandwidth assignable to the i^{th} flow
$min\phi_i$	Minimum bandwidth assignable to the i^{th} flow
ϕ_i	Bandwidth actually assigned to the i^{th} flow
N	Number of entries of the arbitration table
n_i	Number of entries assigned to the i^{th} flow
$GMTU$	General Maximum Transfer Unit
M	Maximum weight per table entry
$pool$	Bandwidth pool
k	Bandwidth pool decoupling parameter
w	Maximum weight decoupling parameter

assign to the most latency-restrictive flows a small amount of bandwidth is not possible because lower distances must be used for these flows and thus, a high number of table entries is devoted to them. This can be a problem because the most latency-restrictive traffic does not usually present a high bandwidth requirement.

Therefore, both approaches have the problem of bounding the bandwidth and latency assignments. We propose a methodology to configure the DTable scheduler that permits to decouple, at least partially, this bounding, allowing us to provide bandwidth and latency requirements with a certain independence among them. With this methodology we set the maximum distance between any consecutive pair of entries assigned to a flow depending on its latency requirement. Moreover, we set the weights of the table entries assigned to a flow depending on its bandwidth requirement. With this methodology we can assign the flows with a bandwidth varying between a minimum and a maximum value that depends not only on the number of table entries assigned to each flow, but also on other configuration parameters.

Supposing an arbitration table with N entries in a network with a certain general MTU $GMTU$, and supposing the i^{th} flow has assigned n_i table entries in order to fulfill its latency requirements, we would like to be able to assign the i^{th} flow a certain bandwidth ϕ_i in the most flexible possible way. This means that we would like the minimum bandwidth $min\phi_i$ that can be assigned to that flow to be as small as possible, and the maximum bandwidth $max\phi_i$ that can be assigned to that flow to be as large as possible. Table 6.1 shows all the involved parameters in the following statements.

Given the maximum weight M that can be assigned to a single table entry of a table with N entries, the maximum total amount of weight that can be distributed among

all the table entries is $M \times N$. However, we are going to fix in advance this total weight to a lower value. We are going to call this value *bandwidth pool*, or just *pool*. Note that the value of M is probably going to be given by the hardware implementation. However, we can always reduce this value by software, in order to accommodate it to our requirements. In this situation, the bandwidth assigned to the i^{th} flow is:

$$\phi_i = \frac{\sum_{j=0}^{J} weight_j}{pool}$$

where J is the set of table entries assigned to the i^{th} flow and $weight$ is the entry weight assigned to a table entry. Therefore, the minimum and maximum bandwidth that can be assigned to the i^{th} flow is:

$$min\phi_i = \frac{n_i \times GMTU}{pool}$$

$$max\phi_i = \frac{n_i \times M}{pool}$$

Let define M and *pool* in function of the GMTU and two configuration parameters w and k:

$$M = GMTU \times w$$

$$pool = N \times GMTU \times k$$

Note that $k \leq w$ because the bandwidth pool cannot be larger than $N \times M$. Note also that $w, k \geq 1$. In this way we can see that the minimum and maximum bandwidth that can be assigned to a flow depend not only on the proportion of table entries n_i that it has assigned, but also on the w and k parameters:

$$min\phi_i = \frac{n_i \times GMTU}{N \times GMTU \times k} = \frac{n_i}{N} \times \frac{1}{k}$$

$$max\phi_i = \frac{n_i \times GMTU \times w}{N \times GMTU \times k} = \frac{n_i}{N} \times \frac{w}{k}$$

When choosing the value of these parameters some considerations must be made. Note that the objective for this methodology is to decrease the minimum bandwidth and to increase the maximum bandwidth that can be assigned to a flow. In order to be able to assign a small amount of bandwidth to a flow with a high proportion of table entries, we can use a high value for the k parameter. However, the higher k is, the smaller the

maximum bandwidth that can be assigned, and thus, the flexibility to assign the bandwidth decreases. We can solve this by increasing the value of w.

Table 6.2 shows two different example scenarios, each one with a different pair of values for the w and k parameters: DTable4 ($k = 2$, $w = 4$) and DTable8 ($k = 4$, $w = 8$). Note that we refer the different DTable scenarios according to the w value used in each case. Table 6.2 shows the minimum and maximum bandwidth that can be assigned to 7 VCs with different proportion of table entries. This proportion of table entries corresponds to 7 VCs with different latency requirements, and thus, different distances between any pair of consecutive entries in the arbitration table. Note that we are going to consider the requirements of a VC as the requirements of the traffic that is going to be transmitted using that VC. We have called these VCs D2, D4, D8, D16, D32, D64, and D64', indicating the distance between any pair of consecutive table entries. Therefore, the D2 VC has stricter latency requirements than the D4 VC, the D4 VC than the D8 VC, and so on. As we can see, when we increase the k parameter, the minimum bandwidth decreases. However, to maintain the same maximum bandwidth in the two scenarios, we have had to increase the w parameter in the same proportion.

Table 6.2: Table configuration example with all the VCs having the same MTU.

		DTable4		**DTable8**	
		$k = 2$, $w = 4$		$k = 4$, $w = 8$	
VC	**%entries**	$min\phi_i$	$max\phi_i$	$min\phi_i$	$max\phi_i$
D2	50	0.25	1	0.125	1
D4	25	0.125	0.5	0.0625	0.5
D8	12.5	0.0625	0.25	0.03125	0.25
D16	6.25	0.03125	0.125	0.015625	0.125
D32	3.125	0.015625	0.0625	0.0078125	0.0625
D64	1.5625	0.0078125	0.03125	0.00390625	0.03125
D64'	1.5625	0.00708125	0.03125	0.00390625	0.03125
Total	100	0.5	2	0.25	2

However, increasing the value of the w parameter has two disadvantages. First of all, the memory resources to store each entry weight are going to be higher. Secondly, the latency of the flows is going to increase, because each entry is allowing more information to be transmitted, and thus, the maximum time between any consecutive pair of table entries will be higher.

It would be desirable to have a good flexibility when assigning the bandwidth to the flows but without increasing too much the w parameter. In order to achieve this, we

propose to use different MTUs for the different flows, instead of considering the general network MTU that the technology fixes for all the flows. This means that each flow has a specific MTU equal to or lower than the general MTU of the network and that we can assign each table entry a minimum weight equal to the specific MTU of the flow associated with that table entry. We can assign each flow a specific MTU by hardware or at the communication library level.

The advantage of having a flow with a specific MTU smaller than the general MTU is that we can assign a table entry a minimum weight equal to the new MTU. When we use the general MTU for all the flows we cannot do this. As stated before, in this case, the general MTU is the smallest value that ensures that there will never be necessary to cycle through the entire table several times in order to gather enough weight for the transmission of a single packet. Being able to assign the table entries of a flow with a weight smaller than the general MTU allows to decrease the minimum bandwidth that can be assigned to that flow. If the i^{th} flow uses a specific MTU of size MTU_i, the maximum bandwidth that can be assigned to that flow is the same:

$$max\phi_i = \frac{n_i \times M}{pool} = \frac{n_i \times GMTU \times w}{N \times GMTU \times k} = \frac{n_i}{N} \times \frac{w}{k}$$

However, the minimum bandwidth depends not only on the w and k parameters, but also on the proportion between the specific MTU and the general MTU:

$$min\phi_i = \frac{n_i \times MTU_i}{pool} = \frac{n_i \times MTU_i}{N \times GMTU \times k} = \frac{n_i}{N} \times \frac{MTU_i}{GMTU} \times \frac{1}{k}$$

Note that varying the w and k parameters affects the minimum and maximum bandwidth that can be assigned to all the flows. However, assigning to a flow a specific MTU smaller than the $GMTU$ only affects the minimum bandwidth of that flow.

Note that with this method we can achieve small minimum bandwidths with a low value for the k parameter. Note also that now k can be even lower than 1. This allows to use a small w and still getting big maximum bandwidths. Specifically, the minimum k value is:

$$pool = N \times GMTU \times k \geq \sum_{i=0}^{I} (n_i \times MTU_i)$$

$$k \geq \frac{\sum_{i=0}^{I} (n_i \times MTU_i)}{N \times GMTU}$$

where I is the number of flows considered by the scheduler.

Table 6.3 shows two different scenarios, each one with a different pair of values for the w and k parameters: DTable1 ($k = 0.5$, $w = 1$) and DTable2 ($k = 1$, $w = 2$). Note that in this case we also refer the different DTable scenarios according to the w value used in each case. This table shows the specific MTU per flow and the minimum and maximum bandwidth that can be assigned to the flows. If we compare these values with the values in Table 6.2, we can see that now we can assign a small amount of bandwidth to those flows with lots of entries with a small w parameter. In this way we have increased the flexibility without increasing the latency of the flows.

Table 6.3: Table configuration example with VCs having different MTUs.

			DTable1		**DTable2**	
			$k = 0.5$, $w = 1$		$k = 1$, $w = 2$	
VC	**%entries**	MTU_i	$min\phi_i$	$max\phi_i$	$min\phi_i$	$max\phi_i$
D2	50	$MTU/32$	0.03125	1	0.015625	1
D4	25	$MTU/32$	0.015625	0.5	0.0078125	0.5
D8	12.5	$MTU/16$	0.015625	0.25	0.0078125	0.25
D16	6.25	$MTU/8$	0.015625	0.125	0.0078125	0.125
D32	3.125	$MTU/4$	0.015625	0.0625	0.0078125	0.0625
D64	1.5625	$MTU/2$	0.015625	0.03125	0.0078125	0.03125
D64'	1.5625	MTU	0.03125	0.03125	0.015625	0.03125
Total	100		0.140625	2	0.07	2

In order to use a different MTU per VC, when a message from a given VC arrives at the network interface, if its size is greater than its specific MTU, the message is split in several packets of a maximum size given by the specific MTU of the VC, as can be seen in Figure 6.4. A possible disadvantage of assigning specific MTUs smaller than the general MTU could be that the bandwidth and latency overhead of fragmenting the original message in several packets could probably affect the performance of the flows. However, most restrictive latency flows (for example network control or voice traffic) usually present low bandwidth requirements, and small packet size. For example, in [TMdM00] several payload values for voice codec algorithms are shown. These values range from 20 bytes to 160 bytes. In that way, if we fix a small MTU for these VCs, no fragmentation will be usually necessary because, in fact, the packets of those VCs are already smaller than the new MTU. Therefore, the cornerstone of this proposal is to tune the specific MTU of each flow according to its specific characteristics.

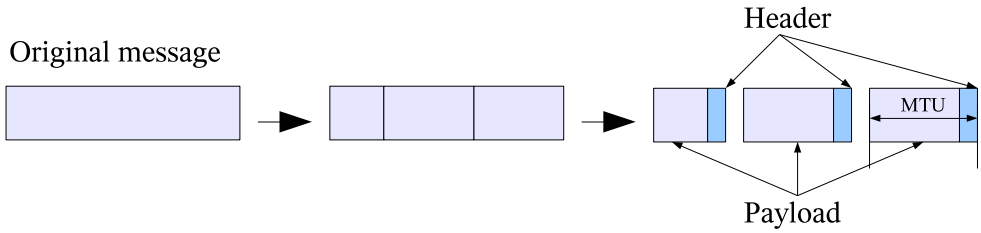


Figure 6.4: Process of message fragmentation into packets.

Summing up, with this decoupling configuration methodology we can configure the DTable scheduler in order to provide a flow with latency and bandwidth requirements in a partially independent way. Depending on the traffic pattern and the bandwidth and latency requirements of the different flows, the network manager must choose the most appropriate k, w, and specific MTU values, and distribute properly the bandwidth pool among the table entries, in order to provide the flows with their latency and bandwidth requirements in the most efficient way.

6.3 Adapting the AS table scheduler

As stated in Section 4.5.2, the AS arbitration table consists in a list of entries that contains a VC identifier. The entries do not have associated any weight as it is the case in the DTable scheduler. When arbitration is needed, the table is cycled through sequentially and a packet is transmitted from the VC indicated in the current table entry regardless of the packet size. This is the reason of the AS table scheduler problem with variable packet sizes.

In order to adapt the AS table scheduler into the DTable scheduler we must add the deficit counter mechanism and a way to associate each table entry with a weight. Adding the deficit counters associated to the VCs would require simple hardware modifications of the original AS table scheduler. However, this modification does not change the interface provided in the AS specification to configure the table scheduler. Note that these counters are set to zero at the beginning and are modified dynamically by the scheduler itself during the scheduling process, and thus they do not require any user configuration.

In this section, we show several possibilities to assign each table entry with a weight modifying as little as possible the AS specifications: To employ a constant value for all the entries, to use the 3-bit reserved field of each table entry, to modify the arbitration table

structure, and to use the same weight for all the entries of a VC. In the following sections, each one of these possibilities is studied.

6.3.1 Using a constant value for all the entries

The simplest way of implementing the DTable scheduler would be to assign all the table entries the same weight. Moreover, this modification would not alter the interface to configure the arbitration table as defined in the AS specification, only its behavior. This fixed weight would be the general MTU of the network. Note that if the three AS VC types (BVC, OVC, and MVC) have a different MTU, the bigest value should be employed. As stated before, this is the smallest value that ensures that there will never be necessary to cycle through the entire table several times in order to gather enough weight for the transmission of a single packet. In this way, when a new table entry is selected, the entry weight is computed as:

$$weight \leftarrow GMTU$$

This approach solves the AS table scheduler problem with variable packet sizes. However, this approach does not allow to employ our decoupling methodology. Therefore, this approach has the problem of bounding the bandwidth and latency assignments. Note that if all the table entries have assigned the same weight, all the table entries allow to transmit the same amount of information, and thus, the number of entries assigned to a VC establishes the minimum bandwidth assigned to that VC. Therefore, we have proposed three other possibilities to fully implement the DTable scheduler in AS. These alternatives are described in the following sections.

6.3.2 Using the 3-bit reserved field

As was stated in Section 4.5.2 in page 67, each entry of the AS arbitration table has 8 bits, being 5 of them for indicating the VC identifier and the other 3 bits are reserved. The approach that we use in this section consists in employing the 3-bit reserved field of each table entry to assign a weight to each entry. Figure 6.5 shows an example of an arbitration table with 64 entries following this approach. The problem of this implementation is that this field only allows us to specify a weight between 0 and 7. Moreover, note that the weight (the number of weight units) assigned to each table entry in the DTable scheduler represents the amount of information that each table entry allows to be transmitted. This weight could be expressed for example in bytes (1 weight unit = 1 byte). However, in

networks with a credit-based flow control, which is the case of AS, the flow control allows to transmit information with a granularity equal to the flow control credit size. Note that in order to transmit a packet of a given size, both the flow control and the scheduling mechanism must allow that packet to be transmitted. Therefore, expressing the weight of the table entries in flow control credits (1 weight unit = 1 flow control credit) is the logical option. However, as we will see, this is not always possible. Therefore, several considerations must be made.

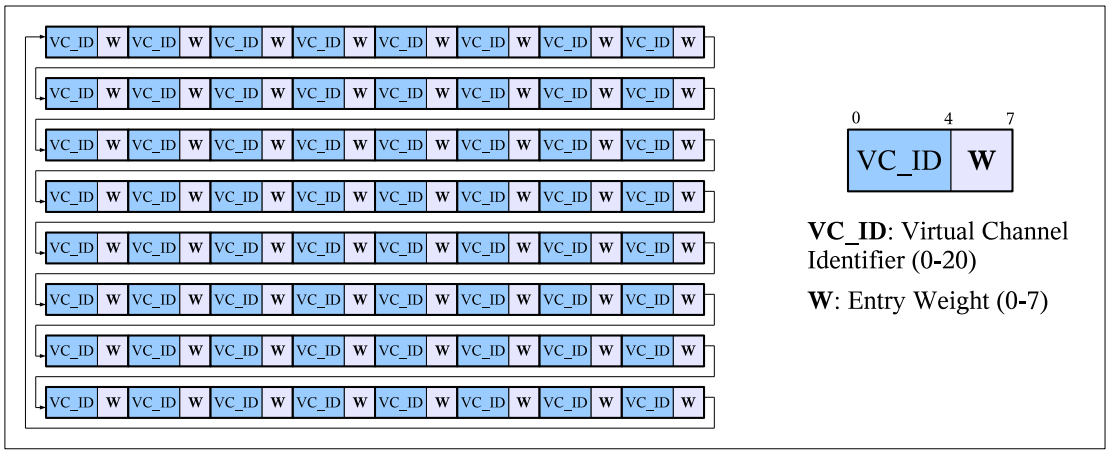


Figure 6.5: Example of using the 3-bit reserved field.

First of all, as stated before, the entry weight must represent at least the value of the general MTU. Therefore, a weight of 0 is not going to be used, and thus, we propose to consider the value 0 as the weight 1, the value 1 as the weight 2, etc. This allows us to specify a weight between 1 and 8 with the 3-bit field. Moreover, in AS, the GMTU can be up to 34 flow control credits (2176 bytes). Obviously, it is not possible to represent directly a value of at least 34 with just 3 bits. Therefore, when using the 3-bit reserved field to assign a weight to each entry, each weight unit will represent a weight equivalent to a certain number of flow control credits m. Therefore, when an entry is selected, its weight must be translated into its value in flow control credits:

$$weight \leftarrow (tableEntry.weight + 1) \times m$$

Note that, in this way, the maximum weight per entry expressed in weight units M' is 8. The maximum weight per entry expressed in flow control credits is:

$$M = M' \times m = 8 \times m$$

Table 6.4: Value of the m parameter with different combinations of $GMTU'$ and $GMTU$.

	$GMTU'$							
$GMTU$	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
3 (192 bytes)	3	2	1	1	1	1	1	1
5 (320 bytes)	5	3	2	1	1	1	1	1
9 (576 bytes)	9	5	3	3	2	2	2	2
17 (1088 bytes)	17	9	6	5	4	3	3	3
34 (2176 bytes)	34	17	12	9	7	3	5	4

If the value of the m parameter is not fixed by hardware, when configuring the DTable scheduler we must specify, apart from the VC identifier and weight of each table entry, this value of m. In this case, it would be necessary to implement an extra configuration parameter in the network elements for the egress link scheduler.

Furthermore, we must have a value in weight units for the GMTU and the specific MTUs, if applicable. Note, that these MTU values indicate the minimum weight in weight units that can be assigned to a table entry depending on its associated VC. Therefore, during the configuration phase, we must choose a value between 1 and 8 weight units to represent the different MTUs. Note that the value of the minimum bandwidth that can be assigned to a VC depends on these MTUs expressed in weight units:

$$min\phi_i = \frac{n_i}{N} \times \frac{MTU'_i}{GMTU'} \times \frac{1}{k}$$

Where $GMTU'$ and MTU'_i are the general and specific MTUs expressed in weight units. Note that the real MTU expressed in flow control credits and its equivalent value in weight units determine the minimum value of the m parameter:

$$GMTU \leq GMTU' \times m,\ MTU_i \leq MTU'_i \times m$$

$$GMTU',\ MTU'_i \in [1,8]; GMTU',\ MTU'_i \in \mathbb{N}$$

For example, if we choose to represent a MTU of 34 flow control credits with 3 weight units, each weight unit must represent at least 12 flow control credits ($12 \times 3 = 36 \geq 34$). We could employ a higher m value, but it would not allow us a higher flexibility and would increase the amount of information that each table entry allows to be transmitted and thus, it would unnecessarily affect in a negative way the latency performance. Table 6.4 shows the appropriated value of the m parameter with different combinations of $GMTU'$ and $GMTU$.

Table 6.5: Value of other configuration aspects when using the 3-bit option.

w	$\frac{8}{GMTU'}$
Minimum k	$\frac{\sum_{i=0}^{I}(n_i \times MTU'_i)}{N \times GMTU'}$
Minimum $\frac{MTU'_i}{GMTU'}$	$\frac{1}{GMTU'}$
Maximum granularity	$\frac{1}{8 \times N}$

Table 6.5 shows other configuration aspects when using the 3-bit implementation option. This implementation possibility limits the maximum weight per entry to 8, and thus the maximum value for the w parameter is also limited to 8 (in this case $GMTU$ would be 1). The values of the general MTU and the specific MTUs are also very limited (1-8). This limits in a high degree the possibility of decreasing the minimum bandwidth that can be assigned to a VC using a small specific MTU. Moreover, if we increase the value of the w parameter, the ratio $MTU'_i/GMTU'$ is even smaller. The bandwidth assignation granularity depends on the bandwidth pool. The maximum bandwidth pool is the maximum weight per table entry multiplied by the number of table entries, and thus, the maximum granularity is 1/(8×N).

Summing up, this possibility limits the possible values for the w parameter and the specific MTUs, and as a consequence limits the flexibility of the table configuration. However, the implementation of this option is quite simple.

6.3.3 Modifying the arbitration table format

Other possibility is to modify the structure of the arbitration table in order to dedicate a higher number of bits to the entry weight. Specifically, we propose to use two bytes per table entry, employing 5 bits for the VC identifier and up to 11 for the entry weight. Figure 6.6 shows an example of an arbitration table with 32 entries following this approach. Note that at least 6 bits are required to represent a MTU of 34 credits. If 6 or more bits are used, the weight field is big enough to directly employ it for storing the entry weight:

$$weight \leftarrow tableEntry.weight$$

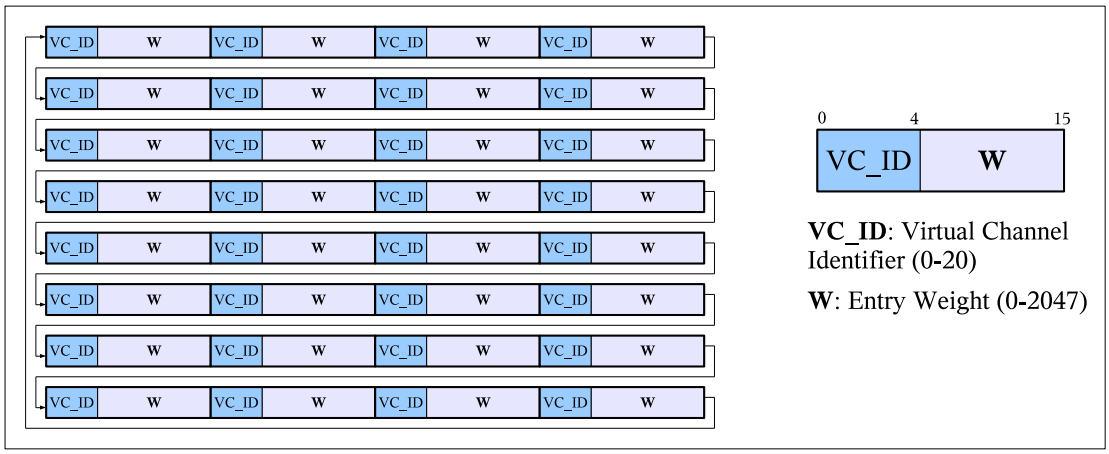


Figure 6.6: Example of modifying the arbitration table format.

Table 6.6: Number of bits assigned to the weight.

Number of bits	**6**	**7**	**8**	**9**	**10**	**11**
Entry weight	$1-64$	$1-128$	$1-256$	$1-512$	$1-1024$	$1-2048$
Maximum w	1.88	3.76	7.53	15.06	30.12	60.24
Maximum granularity	$\frac{1}{N \times 64}$	$\frac{1}{N \times 128}$	$\frac{1}{N \times 256}$	$\frac{1}{N \times 512}$	$\frac{1}{N \times 1024}$	$\frac{1}{N \times 2048}$

Depending on the actual number of bits assigned to the weight we can assign a different maximum value to the w parameter. Table 6.6 shows the maximum w value depending on the number of bits used for the weight. It also shows the allowed weight range per entry and the maximum bandwidth assignation granularity. With 11 bits, the entry weight can take a value between 1 and 2048, and thus, with a MTU of 34 flow control credits, the maximum w parameter is around 60 ($M = GMTU \times w$, $w = 2048/34$) and the maximum granularity is $1/(\mathrm{N}\times 2048)$.

This possibility allows a higher flexibility in the assignation of the w parameter and the specific MTU values. However, it requires the double of memory to store the arbitration table than the previous option for the same number of entries. Moreover, it requires to process two bytes per entry instead of only one.

6.3.4 Using only one weight per VC

The third possibility that we propose is to associate the same weight to all the entries assigned to a VC. Therefore, we only need to specify a weight per VC instead of per table entry. Figure 6.7 shows an example of an arbitration table with 64 entries that schedules 8 VCs following this approach. In order to change as little as possible the AS specification, a possibility is to specify the weight assigned to the entries of each VC employing the MinBW configuration structure, which provides 12 bits to specify a weight per each VC. This allows us to specify a weight between 1 and 4096, and thus, the maximum w value is around 120 ($M = GMTU \times w$, $w = 4096/34$). When a new table entry is selected, the accumulated weight is computed as:

$$weight \leftarrow weight_{selectedVC}$$

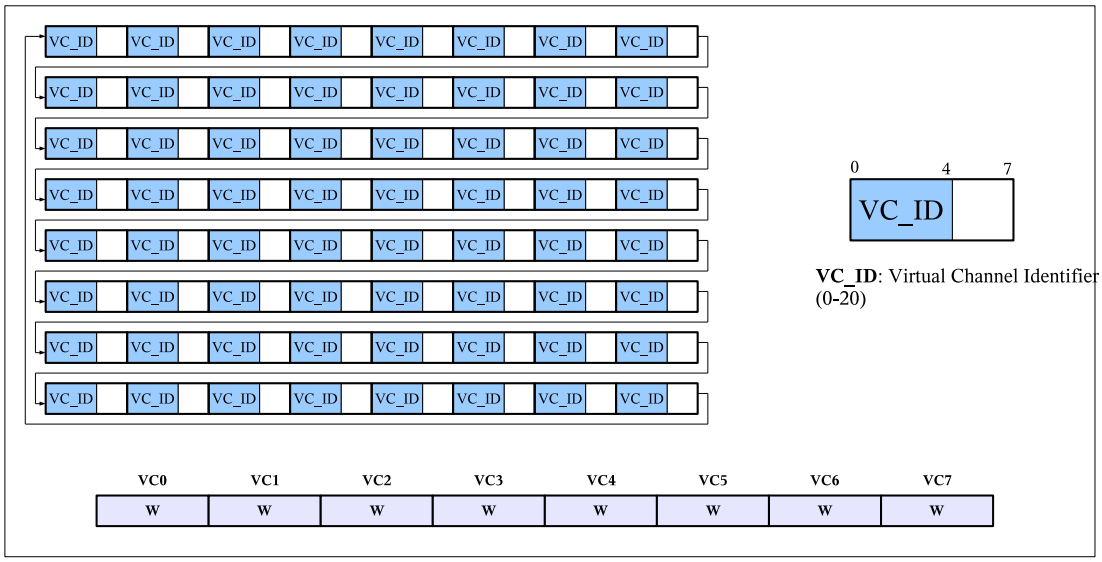


Figure 6.7: Example of using one weight per VC.

This possibility also allows us a higher flexibility in the assignation of the w parameter and the specific MTU values than the 3-bit option. The main disadvantage is that we cannot assign the weight units from the bandwidth pool between the table entries in a totally free way. We have to assign the weights in exact fractions of the number of entries of each VC. Therefore, the bandwidth assignation granularity is different for each VC and depends on the number of entries assigned to that VC: $n_i/(\mathrm{N}\times 4096)$.

Table 6.7: Summarized properties of the different possibilities to adapt the original AS table scheduler into the DTable scheduler.

	Constant	**3-bits**	**2 bytes**	**1 weight VC**
Maximum w	1	8	60	120
Minimum k	1	$\frac{\sum_{i=0}^{I}(n_i \times MTU_i)}{8 \times N}$	$\frac{\sum_{i=0}^{I}(n_i \times MTU_i)}{34 \times N}$	$\frac{\sum_{i=0}^{I}(n_i \times MTU_i)}{34 \times N}$
Minimum $\frac{MTU_i}{GMTU}$	1	$\frac{1}{8}$	$\frac{1}{34}$	$\frac{1}{34}$
Maximum granularity	$\frac{1}{N}$	$\frac{1}{8 \times N}$	$\frac{1}{2048 \times N}$	$\frac{n_i}{4096}$

6.3.5 Final considerations

In this section we have seen four possibilities to implement the DTable scheduler in AS. Table 6.7 shows a summary with some characteristics of the various approaches. As stated before, the simplest way of adapting the original AS table scheduler would be to assign all the table entries the same weight. However, this partial approach does not allow to employ our decoupling methodology. Using the 3-bit reserved field is probably the simplest possibility to implement a fully functional version of the DTable scheduler. However, it limits the possible values for the w parameter and the specific MTUs.

The possibility of using the same weight for all the entries of a VC allows us to use higher values for the w parameter and to choose freely the values for the specific MTUs. However, the bandwidth assignation granularity is different for each VC and depends on the number of entries assigned to that VC. The possibility of modifying the arbitration table structure does not present these problems, but it requires a higher amount of memory to store the arbitration table and needs to process two bytes, instead of just one, per table entry.

6.4 Summary

The main problem of the AS table-based scheduler is that it does not work in a proper way with variable packet sizes, as it is common in actual traffic. Moreover, it faces the problem of bounding the bandwidth and latency assignments. In this chapter we have presented a new table-based scheduler, which we have called DTable scheduler, that works in a proper

way with variable packet sizes. Moreover, we have proposed a configuration methodology that decouples at least partially the latency and bandwidth bounding.

With this decoupling configuration methodology we can configure the DTable scheduler in order to provide a VC with latency and bandwidth requirements in a partially independent way. Specifically, we assign the table entries among the VCs attending to the latency requirements of the traffic that traverse those VCs, and the weights of the table entries attending to the VC bandwidth requirements. The bandwidth that can be assigned to each VC depends not only on the proportion of table entries assigned to the VC, but also on two general decoupling configuration parameters and the specific MTU of that VC.

In order to adapt the AS table scheduler into the DTable scheduler we must add the deficit counter mechanism and a way to associate each table entry with a weight. In this chapter, we show several possibilities to assign each table entry with a weight modifying as little as possible the AS specifications: To employ a constant value for all the entries, to use the 3-bit reserved field of each table entry, to modify the arbitration table structure, and to use the same weight for all the entries of a VC.

Chapter 7

Hardware implementation study of the MinBW and DTable schedulers

As stated in Section 3.5 in page 36 the end-to-end delay, flexibility, and protection that a scheduler is able to provide are not the only parameters that must be taken into account when deciding which is the most appropriate scheduler in a high-performance network with QoS support. Other very important property that a scheduling mechanism should satisfy is to have a low complexity [Siv00].

We can measure the complexity of a scheduler based on two parameters: Silicon area required to implement the scheduling mechanism and time required to determine the next packet to be transmitted. A short scheduling time is an efficiency requirement. The next packet to be transmitted should be chosen during the transmission time of the last packet which was selected by the scheduler. This is necessary in order to be able to send packets one after another without letting gaps between them. This requirement takes more importance in high-performance networks due to their high speed. Moreover, switches of high-performance interconnection technologies are usually implemented in a single chip. Therefore, the silicon area required to implement the various switch elements is a key design feature. Note, that a scheduling algorithm must be implemented in each egress link and thus, the silicon area required to implement the scheduling algorithm should be as small as possible.

In this section we are going to analyze the implementation and computational complexity of the MinBW and DTable schedulers. In [VV04] and [RGB96] interesting implementations for the WFQ and SCFQ schedulers are proposed. However, this implementations were designed for a high number of possible flows. Note that in our case there

are going to be just a limited number of VCs. This allows to consider more efficient implementations. Moreover, the case of the SCFQ implementation [RGB96] was intended for fixed packet sizes, specifically, for an ATM environment.

Therefore, we have performed our own hardware implementation for the different schedulers. We have modeled the schedulers using Handel-C language [Cel05] and employed the DK design suite tool from Celoxica in order to obtain hardware estimates on silicon area and arbitration time. Note that the code that we have designed can actually be used to implement the DRR-CA and SCFQ-CA schedulers in a Field Programmable Gate Array (FPGA) or, if the appropriate conversion is made, in an Application Specific Integrated Circuit (ASIC). However, this has not been the objective of our work. Therefore, we have tried to implement the schedulers in an efficient way, but there could have probably been implemented more efficiently. Our objective has neither been to obtain explicit values for the silicon area and arbitration time of each scheduler. In fact, these values are very dependent on the specific FPGA or the implementation technology employed. We are more interested in the relative differences on silicon area and arbitration time for the different schedulers and the effect of some design parameters like the number of VCs or the MTU.

7.1 Handel-C and the DK design suite

As stated before, we have employed the Handel-C language to model and obtain hardware estimates for the different schedulers that we have considered. Handel-C is essentially an extended subset of the standard ANSI-C language, specifically extended for being used in hardware design (see Figure 7.1).

Handel-C's level of design abstraction is above Register Transfer Level (RTL) languages, like VHDL [Ash02] and Verilog [Pal03], but below behavioral. In Handel-C each assignment infers a register and takes one clock cycle to complete, so it is not a behavioral language in terms of timing. The source code completely describes the execution sequence and the most complex expression determines the clock period.

A comparison of Handel-C with RTL languages shows that the aims of these languages are quite different. RTL languages are designed for hardware engineers who want to create sophisticated circuits. They provide all constructs necessary to craft complex, tailor made hardware designs. By choosing the right elements and language constructs in the right order, the specialist can specify every single gate or flip-flop built and manipulate the propagation delays of signals throughout the system. On the other hand, RTL languages

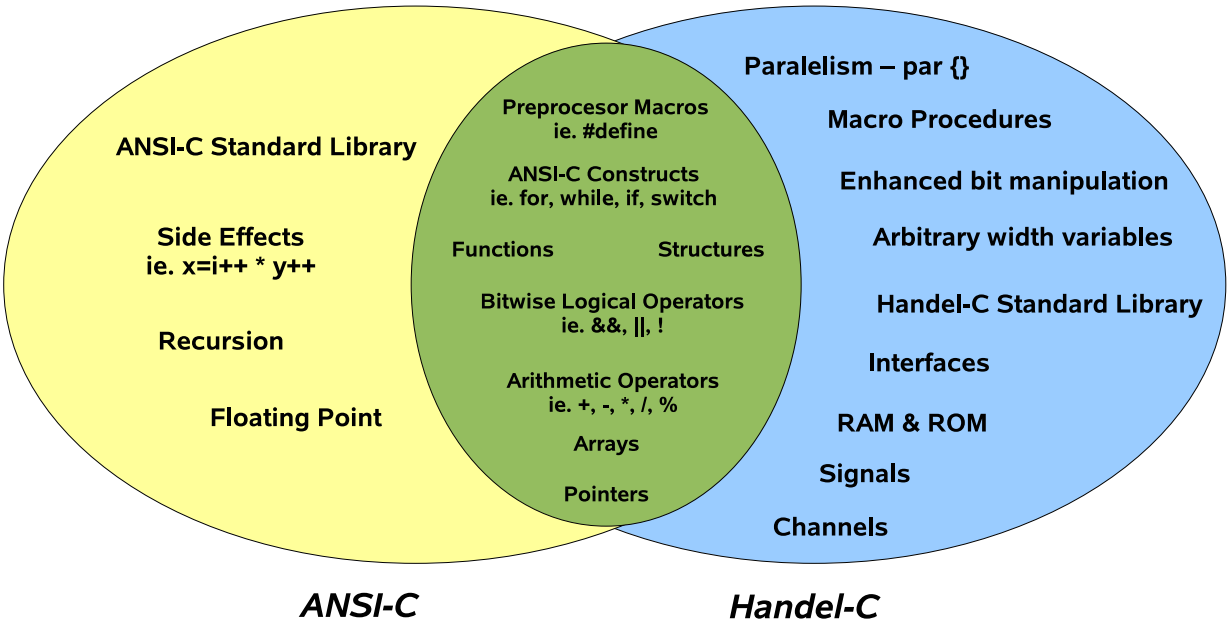


Figure 7.1: ANSI-C / Handel-C comparison.

expect that the developer knows about low-level hardware and requires him continuously thinking about the gate-level effects of every single code sequence.

In contrast to that, Handel-C is not designed to be a hardware description language, but a high-level programming language with hardware output. It doesn't provide highly specialized hardware features and allows only the design of digital synchronous circuits. Instead of trying to cover all potentially possible design particularities, its focus is on fast prototyping and optimizing at the algorithmic level. The low-level problems are hidden completely, all the gate-level decisions and optimization are done by the compiler so that the programmer can focus his mind on the task he wants to implement. As a consequence, hardware design using Handel-C resembles more to programming than to hardware engineering.

Handel-C closely corresponds with a typical software flow and provides the essential extensions required to describe hardware. These extensions include flexible data widths, parallel processing and communications between parallel threads. Sequential by default, Handel-C has a *par* construct. When a block of code is qualified by *par*, statements are executed concurrently and synchronized at the block end. This simple construct allows for the expression of mixed sequential and parallel flows in compact and readable code.

The Handel-C compiler comes packaged with the Celoxica DK design suite. The DK design suite supports several output targets:

- Debugger: The debugger provides in-depth features normally found only in software development. These features include breakpoints, single stepping, variable watches, and the ability to follow parallel threads of execution. The hardware designer can step through the design just like a software design system using this approach.

- EDIF: The second output target is the synthesis of a netlist for input to place and route tools. Place and route is the process of translating a netlist into a hardware layout. This output allows the design to be translated into configuration data for particular chips. When compiling the design for a hardware target, Handel-C emits the design in Electronic Design Interchange Format (EDIF).

- RTL (VHDL and Verilog): The RTL output preserves the hierarchy of the Handel-C source code allowing experienced engineers to verify at the RTL level. The compiler generates RTL with appropriate syntax and attributes for leading third party synthesis tools, timing simulators and ASIC design flows.

In order to obtain the hardware estimates in which we are interested:

1. We have modeled in Handel-C a full egress queuing system, including the scheduler.

2. We have validated the schedulers employing the simulation and debugging functionality of the DK design suite.

3. We have isolated the scheduler module in order to obtain estimates without influence of other modules.

4. We have obtained the EDIF output for a Virtex 4 FPGA from Xilinx [Xil07].

A cycle count is available from the Handel-C source code: Each statement in the Handel-C source code is executed in a single cycle in the resulting hardware design and thus, the number of cycles required to perform a given function can be deduced directly from the source code. Moreover, an estimate of gate count and cycle time is generated by the EDIF Handel-C compiler. The cycle time estimate is totally dependent on the specific target FPGA, in this case the Virtex 4 [Xil07], which is one of the last FPGA models provided by Xilinx [Xil]. However, as our objective is to obtain relative values instead of absolute ones, we consider that this approach is good enough to be able to compare the complexity in terms of silicon area and scheduling time of the different schedulers. Figure 7.2 reflects the design flow that we have followed.

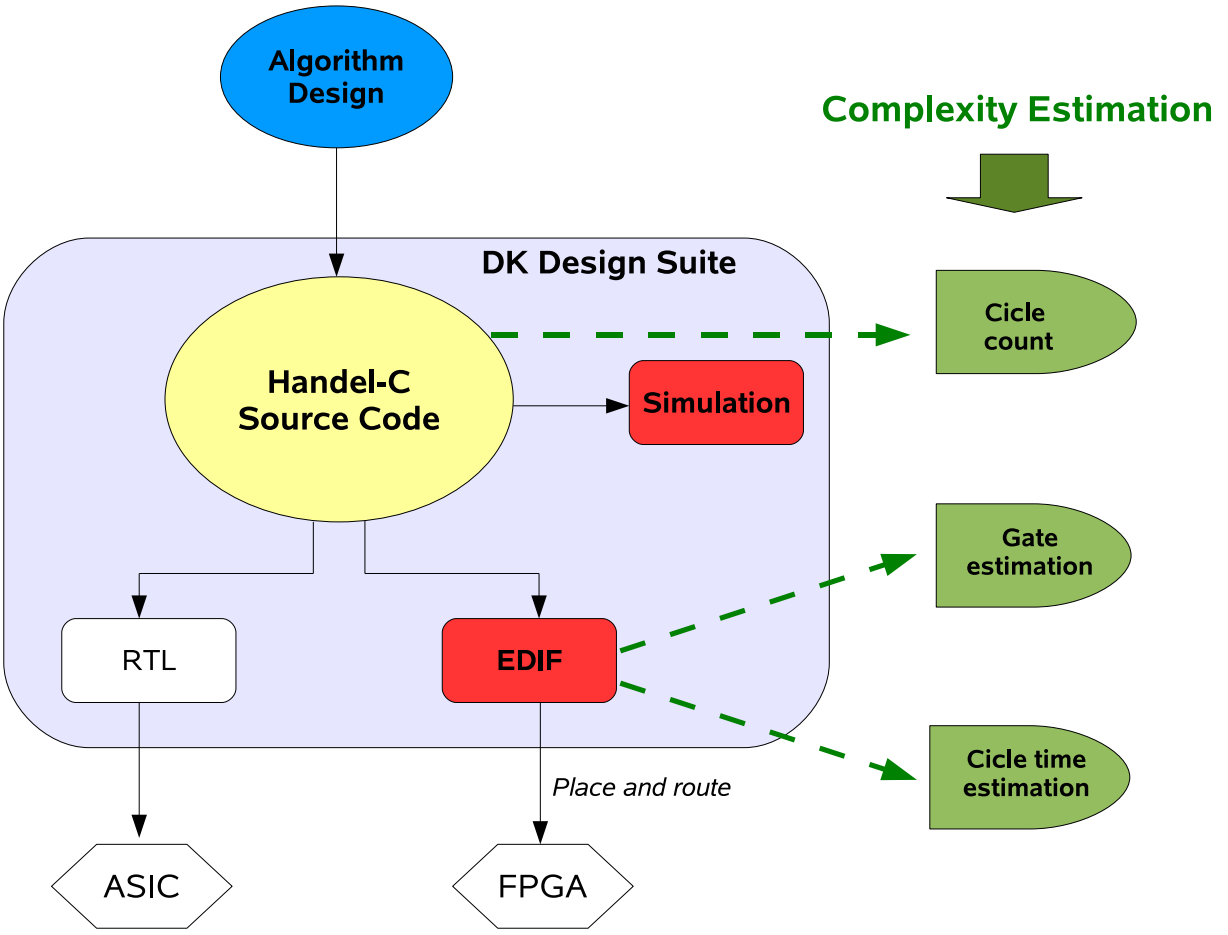


Figure 7.2: Design flow with DK employing Handel-C.

7.2 Modelling the egress queuing system

As stated in the previous section, in order to model the different schedulers, we have previously modeled a full egress link queuing system that could be part of an endnode or switch. We have done this in order to be able to test the rightness of our implementation. Figure 7.3 shows the different modules that compound the egress queuing system and their interactions. These modules are:

- **Traffic generator:** We need a traffic load in order to test the schedulers. We have developed a Constant Bit Rate (CBR) traffic generator in order to feed the VCs. We can assign each VC with a different traffic generator configured to produce packets at a different rate and with different packet size.

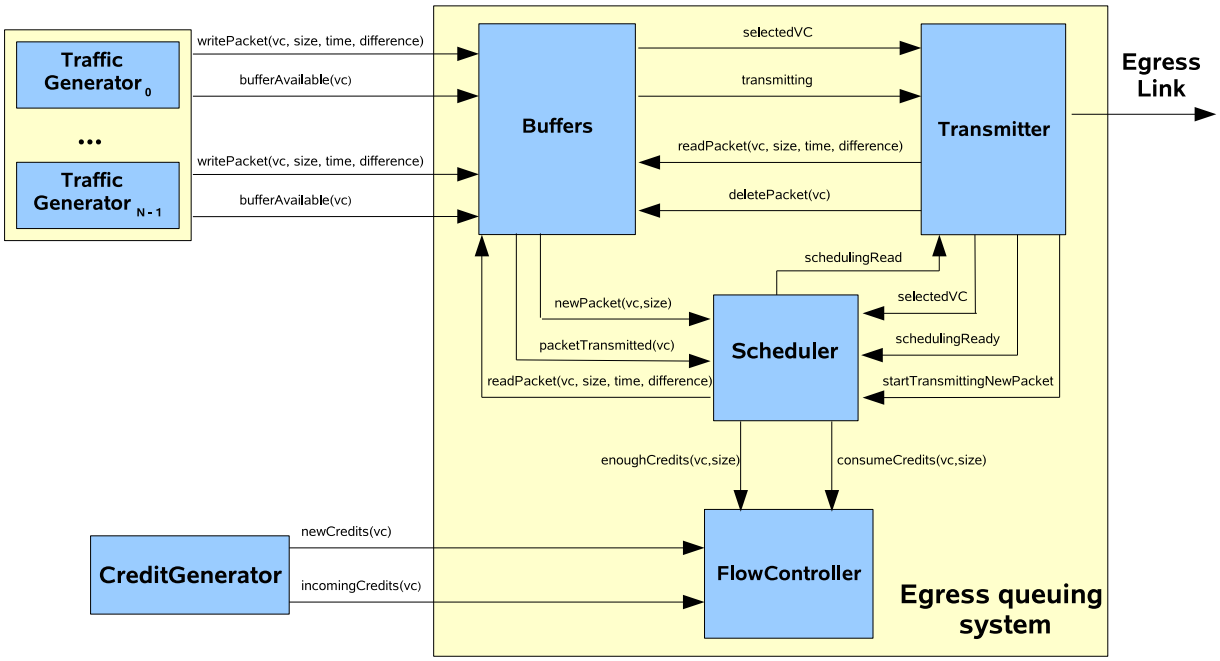


Figure 7.3: Egress link queuing system modules.

- **Buffers:** The buffers module is the responsible of managing the packets stored in each VC queue. It tracks the available space in each queue, notifies the scheduler the arrival of new packets, and frees space in the queues when packets are transmitted.

- **Transmitter:** The transmitter module injects into the egress link the packets that the scheduler indicates and deletes the information of those packets in the buffers.

- **Scheduler:** The scheduler module is the most important part to our objective. Its main function is to decide the next packet to be transmitted from an active VC. In order to do so, it keeps track of the set of active VCs by monitoring the packet at the head of each queue and the available number of flow control credits. Moreover, it consumes the flow control credits required by each transmitted packet. When a scheduling decision has finished it notifies that fact to the transmitter.

- **Flow controller:** The flow controller tracks the number of available flow control credits of each VC.

- **Credit generator:** Only one egress queuing system has been modeled, and thus in order to keep the system transmitting packets we need to renew the consumed flow control credits with a flow control credit generator module.

An advantage of using Handel-C to model the egress queuing system and the schedulers is that it allows parameterizing the design in an easy way. Through the use of constants and compiler commands we can generate outputs (for simulation, EDIF, or RTL targets) with, for example, variable number of VCs and packet MTU considered. In order to simplify the design, we have considered power of two values for the number of VCs and MTU. Moreover, we have considered packets to be of an integer number of flow control credits. Note that, in the case of the MinBW scheduler, each VC is going to have assigned a weight between 1 and 4096 (see Section 4.5.2 in page 67).

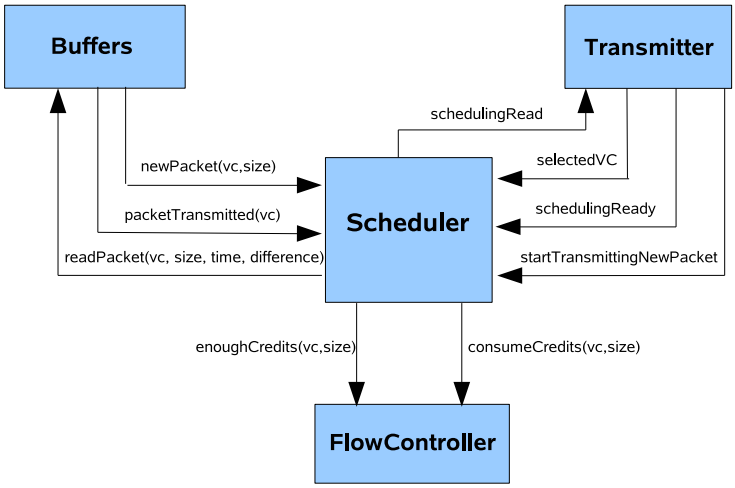


Figure 7.4: Scheduler module.

7.3 Hardware implementation of the MinBW scheduler

The scheduler module (see Figure 7.4) performs a variety of actions in synchronization with the rest of the modules in order to make a proper arbitration. The most important actions that it performs are probably the following:

- Selecting the next packet to be transmitted among the packets at the head of the active VCs.
- Stamping the packet with an appropriate tag (in the case of the sorted priority algorithms).

Moreover, these are the actions that differentiate one scheduler from the others. In the next sections we are going to show briefly the way in which we have implemented these functions for the different schedulers.

7.3.1 The DRR-CA scheduler

When a packet arrives at the head of a VC queue the scheduler receives a notification from the buffers. The DRR-CA scheduler just takes note of the packet size and actives the VC if there are enough flow control credits to transmit that packet. In order to select the next VC that can transmit packets, the scheduler must select the next active VC from the last selected VC in a list with all the VCs. The scheduler transmits packets from the same VC until the flow becomes inactive or there is no enough quantum to transmit more packets from that VC.

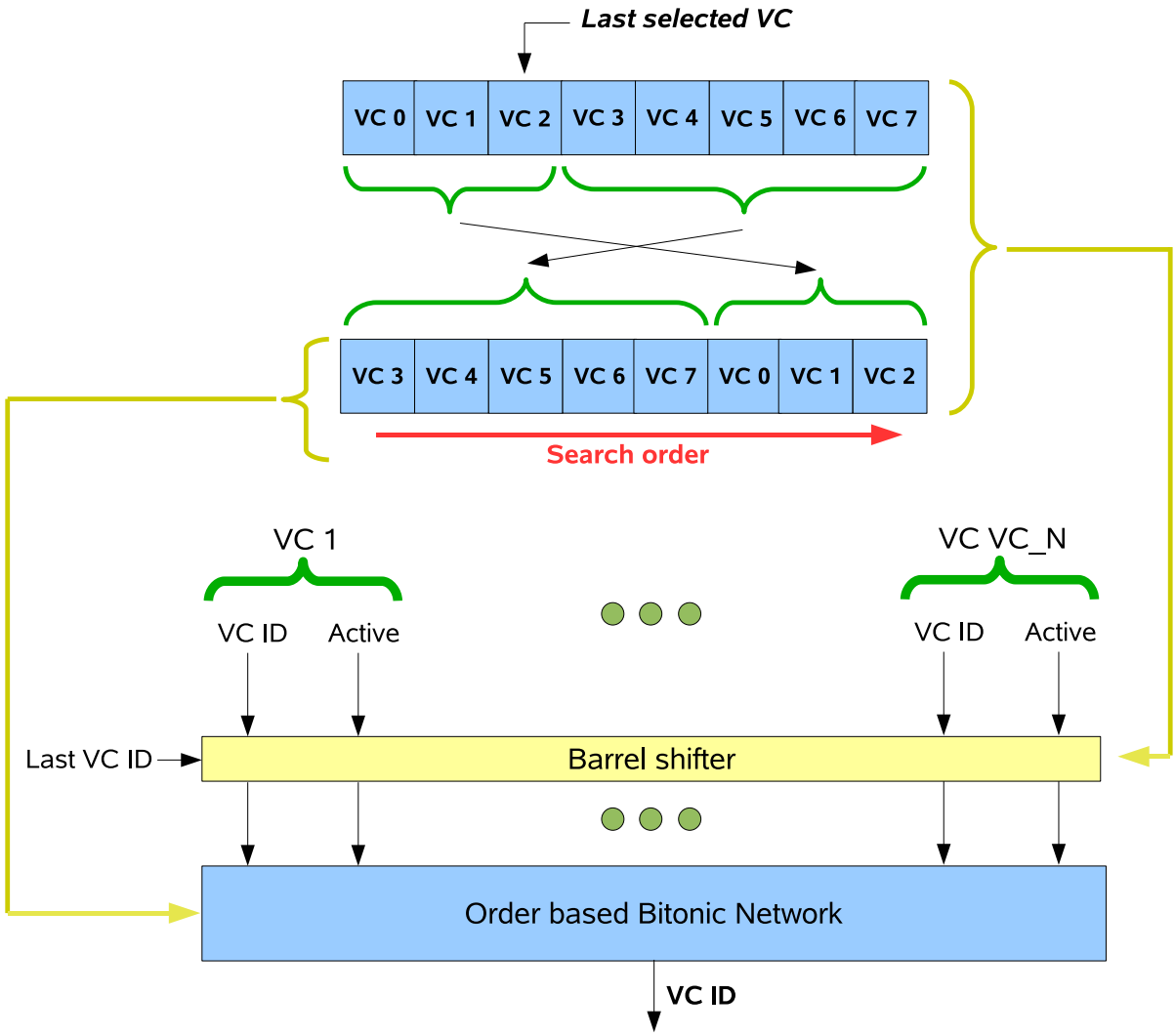


Figure 7.5: Structure of the module that selects the next VC to transmit in the DRR-CA scheduler.

A possible way of implementing the mechanism that selects the next active VC would be to check sequentially all the VCs in the list starting from the contiguous position of the last selected VC (see Figure 7.5). However, in order to make this search in an efficient way, we have implemented it with a *barrel shifter* connected to an *order based bitonic network*. The barrel shifter rearranges the list in the correct order of search and the bitonic network finds the first active VC in a logarithmic number of cycles. This structure for the selector function is shown in Figure 7.5.

7.3.2 The SCFQ-CA scheduler

When a new packet arrives at the SCFQ-CA scheduler, apart from taking note of the packet size and activating the VC if there are enough flow control credits to transmit that packet, this scheduler must calculate the packet service tag. As stated in Section 5.3 in page 81, we have solved the problem of the possible overflow of the service tags. Moreover, this modification entails a simplification of the computation of the service tag, which is:

$$S_i = \frac{L_i^{first}}{\phi_i}$$

However, this calculation consists in a division, and a divider is not a simple mathematical unit. Handel-C offers a divisor operand that calculates the result in one cycle (as all the Handel-C statements). Employing this operand makes the division very short in terms of number of cycles but, it makes the cycle time very long, and thus it makes the arbitration time quite long. Therefore, we have also implemented a version of the SCFQ-CA scheduler that employs a mathematical division unit that performs the division in several cycles. Specifically, it takes a number of cycles equal to the length of the operators plus one. This second version reduces the cycle time and thus, the arbitration time. However, the division requires much more cycles to be performed. It even requires more time to be performed because the cycle time is not reduced in the same proportion as the number of cycles is increased. We have called the SCFQ-CA version that performs the division in one cycle '*atomic SCFQ-CA*'. On the other hand, we have called the SCFQ-CA version that performs the division in several cycles '*segmented SCFQ-CA*'.

The advantage of the atomic SCFQ-CA is that it calculates the time tag in only one cycle, and thus it takes the same time than the DRR-CA scheduler, and as we will see, also the table scheduler, for processing a new packet. This makes very easy to compare both schedulers, because it is only necessary to confront the silicon area and arbitration time. However, in the segmented SCFQ-CA case processing a new packet takes much more

time, and without a full model of a switch the effect over the overall performance of this longer time is not easy to measure. We include this option in this study because it is a possibility that must be taken into account, but the comparison with the rest of schedulers is not so clear like in the atomic SCFQ-CA case.

In order to decide which is the next packet to be transmitted, the SCFQ-CA algorithm must choose the packet from the active VC with the smallest service tag. In order to do this in an efficient way, we have employed a bitonic network. The structure of the selector module is shown in Figure 7.6.

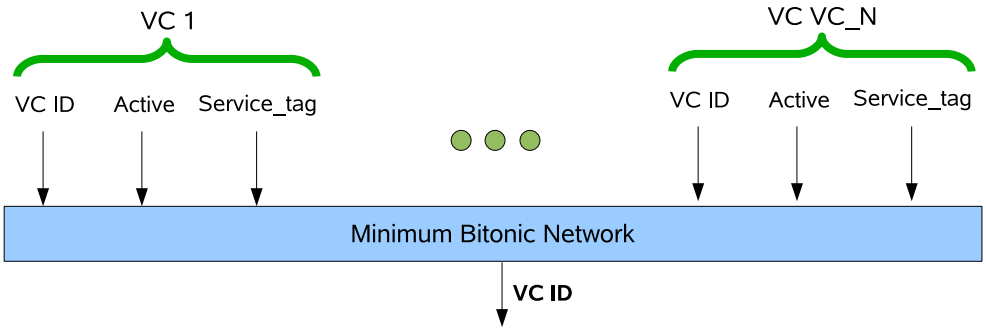


Figure 7.6: Structure of the selector module for the SCFQ-CA scheduler.

7.3.3 The WFQ-CA scheduler

The main complexity sources of the WFQ-CA scheduling algorithm are:

- **Maintaining updated the virtual clock.** As stated in Section 5.2 in page 79, the virtual clock of the WFQ-CA scheduler is updated each time that:
  - A new packet is received by the scheduler in the egress link queues and must be stamped with its timestamp.
  - A packet has finished to be transmitted and the VC to which it belongs becomes inactive.
  - A flow control packet with new flow control credits is received and a packet that was inactive due to lack of flow control credits becomes active.

  As stated in Section 3.5.1 in page 41 the virtual clock is calculated as:

$$V(t_{j-1}+\gamma) = V(t_{j-1}) + \frac{\gamma}{\sum_{i \epsilon B_j(t)} \phi_i}$$

This entails that each time one of those events happen, several sums and one division must be performed. If various events concur in the same cycle or in very close ones, the process of these events may concatenate, affecting the time required to calculate the timestamps.

- **Calculating the virtual finishing time.** As stated in Section 3.5.1, the timestamp of each packet, its virtual finishing time, is calculated based on the virtual clock, the timestamp of the previous packet of the same VC, and the packet size:

$$F_i^k = max\{F_i^{k-1}, V(A_i^k)\} + \frac{L_i^k}{\phi_i}$$

  This entails one comparison and one division. This would be a little more complex than in the SCFQ-CA case due to the comparison, but could be implemented without too much problem.

- **Selecting the next packet to be transmitted among the active VCs.** As it has been shown for the SCFQ-CA case, due to the small number of VCs this can be done with a bitonic network in a relatively short time. If the scheduler would work at a flow level, this would entail much more complexity.

- **Restamping the virtual finishing time tags for those packets in VCs that have been activated again after receiving more flow control credits.** This restamping process is necessary to protect the rest of VCs against those VCs which become inactive due to lack of flow control credits. However, it entails to recalculate the timestamp of all the packets in the queue and thus, it may require a lot of time to be performed.

- **Avoiding the overflow of the registers used to store the virtual clock and the timestamps.** As stated in Section 3.5.1, the virtual clock cannot be reinitialized to zero until the system is completely empty and all the sessions are idle. This is a problem because the value of the virtual clock is an increasing function of the time and thus, it can overflow during long busy periods. This problem can be mitigated employing big registers to store the virtual clock and the timestamps. However, there is no total guarantee that there is not going to be an overflow. Moreover, the use of bigger registers entails mathematical units that require more silicon area and time to calculate the results.

Summing up, we believe that this scheduling algorithm is too complex to be implemented in a high-performance network because of the reasons outlined before. Or at least,

it is much more complex than the SCFQ-CA scheduler. Moreover, it would be very difficult, if not impossible, to measure the effect on the time required to compute the virtual clock and the restamping process without a full hardware design for the switches, which is out of the scope of this work. Therefore, we are going to employ the WFQ-CA scheduler in the Performance Evaluation Chapter in order to compare the performance of this scheduler in terms of throughput, latency, and jitter with the rest of schedulers, especially with the SCFQ-CA. However, we are not going to design a hardware model for this scheduler nor are we going to obtain hardware estimates for it.

7.3.4 Hardware estimates for the DRR-CA and SCFQ-CA schedulers

As stated before, once that the schedulers have been validated through simulation with the debugger functionality of the DK design suite, we have isolated the scheduler module in order to compile it for the EDIF output. In this way the hardware estimates obtained, like the cycle time, are not going to be influenced by the rest of modules. Table 7.1 shows the number of cycles required by the DRR-CA and SCFQ-CA schedulers to perform the arbitration. Therefore, the arbitration time depends on the cycle time and on the number of VCs (VC_N in the table).

Table 7.1: Arbitration time in cycles for the DRR-CA and SCFQ-CA schedulers.

Scheduler	**Number of cycles**
DRR-CA	$log_2(VC\_N) + 3$
Atomic SCFQ-CA	$log_2(VC\_N) + 2$
Segmented SCFQ-CA	$log_2(VC\_N) + 2$

Figure 7.7 shows how the increment in the number of VCs and the MTU affects the silicon area and the arbitration time of the DRR-CA and SCFQ-CA schedulers. Specifically, it shows the increment in these complexity indices respect the simplest case for each scheduler (2 VCs and a MTU of 2). When varying the number of VCs, we have used a MTU of 32 and when varying the MTU we have considered 8 VCs.

Regarding the effect of the VCs, Figure 7.7 shows that the number of VCs influences dramatically the silicon area and arbitration time required by the DRR-CA and SCFQ-CA schedulers. Note that in the case of the arbitration time, the increment is due to both, the

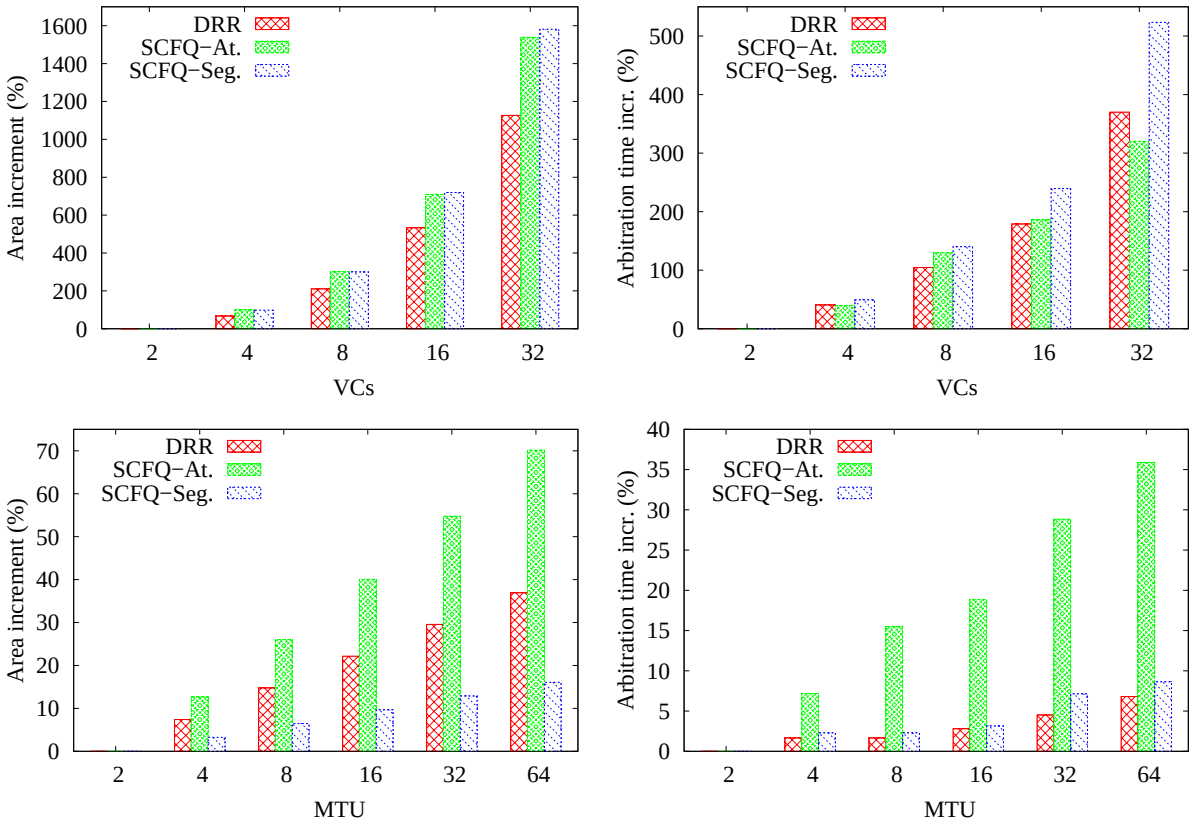


Figure 7.7: Effect of the number of VCs and MTU over the silicon area and arbitration time required by the DRR-CA and the SCFQ-CA schedulers.

increase in the cycle time and the increase in the number of cycles required to compute the arbitration.

On the other hand, regarding the effect of the MTU, Figure 7.7 shows that the increase in silicon area and time when increasing the MTU is not so important if compared with the effect of the number of VCs. The atomic variant of the SCFQ-CA scheduler is the most affected by this parameter. Increasing the MTU from 2 to 64 increases the silicon area required by this scheduler 70% and the arbitration time 37%. The reason of this is that the value of the MTU affects the size of the division operation required to calculate the SCFQ-CA service tag and thus, it affects in a higher degree the atomic version of the SCFQ, which requires a lot of silicon area and increases in a high degree the cycle time in order to perform the division in a single cycle.

Figure 7.8 shows the same results than Figure 7.7 except that in this case, the increment is relative to the silicon area and arbitration time required by the DRR-CA scheduler with 2 VCs (when varying the number of VCs) and a MTU of 2 (when varying

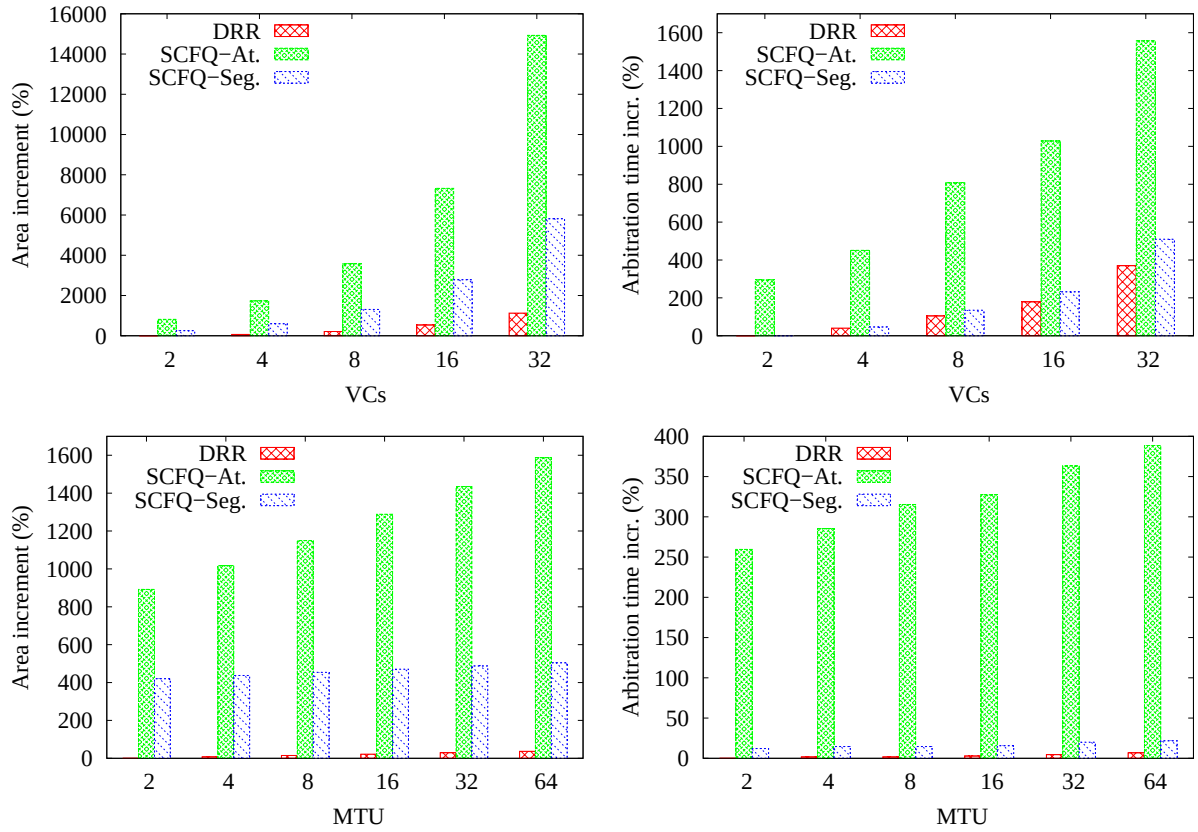


Figure 7.8: Comparison of the silicon area and arbitration time required by the DRR-CA and the SCFQ-CA schedulers.

the MTU). This allows us to compare the silicon area and the arbitration time required by the different schedulers for different design parameters.

Figure 7.8 shows, as expected, that the DRR-CA scheduler is the simplest scheduler in terms of silicon area and arbitration time. On the other hand, the atomic version of the SCFQ-CA scheduler requires much more silicon area and arbitration time than the DRR-CA or the segmented SCFQ-CA schedulers. Figure 7.8 also shows that the segmented SCFQ-CA scheduler requires also much more silicon area than the DRR-CA scheduler. However, the difference in arbitration time is not so big. Finally, this figure shows that the difference among the atomic SCFQ-CA scheduler and the other two scheduler increases with the MTU.

7.4 Hardware implementation of the DTable scheduler

When a new packet is notified to the DTable, this scheduler just takes note of the packet size and actives the VC if there are enough flow control packets to transmit that packet (it makes the same as the DRR-CA scheduler). As in the DRR-CA case, this scheduler transmits from the same selected VC until the VC becomes inactive or the remaining weight entry is not enough to transmit the packet at the head of the VC queue. In order to select a new VC to transmit from, the arbitration table must be looked over sequentially searching for the next active entry and skipping those entries that refer to a VC without packets or credits to transmit. Although the checking of each entry can be made with very simple computational units, in the worst case all the table must be looked over in order to find the next active entry.

In order to make the process faster, several entries of the table can be read simultaneously at the expense of increasing the silicon area and probably the cycle time. This algorithm also requires the memory necessary to store the arbitration table. However, this algorithm has not the problem of the increasing tag value and does not need mathematical division units to calculate any packet tag of sorted priority algorithms.

The arbitration table can be stored in specialized memory blocks, like the SRAM block that can be found in most FPGAs models, or in an array of registers. A possible way to read several entries simultaneously in an efficient way is to split the register array or memory block in several subblocks and read one entry of each of these subblocks in the same cycle. We have called the number of simultaneous table entries read in a single cycle the *parallelization grade*.

Figure 7.9 shows the structure of the mechanism that we have implemented to obtain the next active table entry. First of all we read a certain number of consecutive table entries from the last selected table entry equal to the parallelization grade. The next cycle, we check if any of those entries refers to an active VC. At the same time, the next 'parallelization grade' entries are read. When the mechanism realizes that at least one entry is active in the set of table entries, the process stops and a bitonic network is employed to calculate which is the first active entry in the subblock.

Table 7.2 shows the number of cycles required to make the arbitration decision in both cases, when the table is cycle through sequentially or, when various entries are processed at the same time. Note that in the DTable scheduler case, the number of cycles required to complete the arbitration is variable and depends on how far from the last selected entry is the next selected entry. When the load of the network is low, more cycles will be probably required in average to found the next table entry. When the load of the

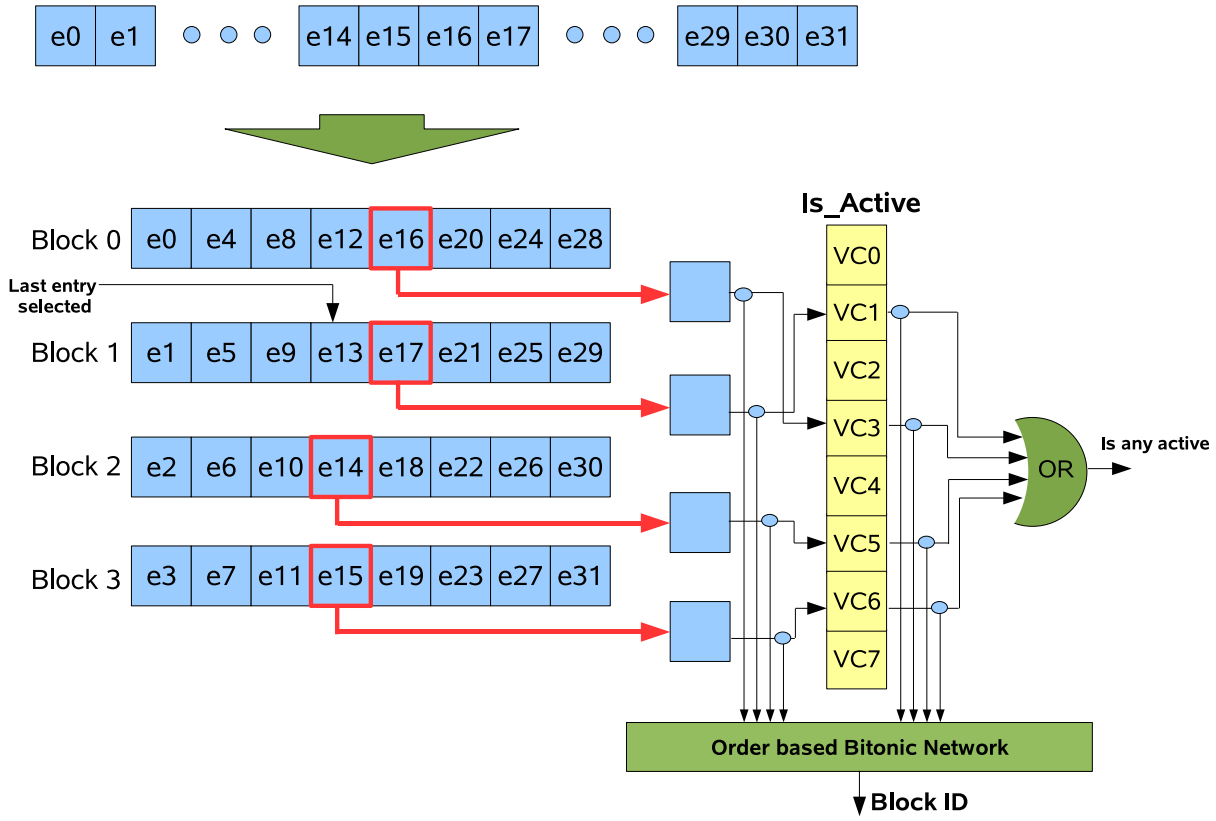


Figure 7.9: Structure of the selector module for the parallel table scheduler.

Table 7.2: Arbitration time in cycles for sequential and parallel implementations of the DTable scheduler.

Scheduler	Number of cycles
Table (Sequential search)	$[1 - \#Entries] + 2$
Table (Parallel search)	$[1 - \frac{\#Entries}{Parallel\_Grade}] + log_2(Parallel\_Grade) + 3$

network is high, most VCs will be active anytime, and thus the average number of cycles will be very small.

7.4.1 Hardware estimates for the DTable scheduler

In order to obtain hardware estimates of the DTable scheduler we have considered, apart from the number of VCs and the MTU, the number of table entries and the parallelization grade as design parameters. Moreover, we have also calculated hardware estimates to

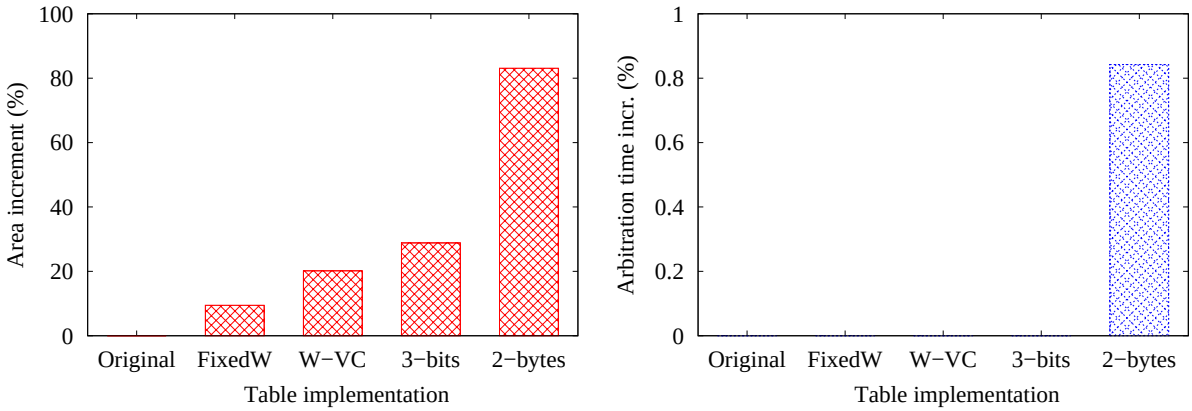


Figure 7.10: Complexity comparison of the different possible implementations of the DTable scheduler.

compare the original AS table with the possible implementations of the DTable scheduler shown in Section 6.3 in page 97.

Figure 7.10 shows the difference in silicon area and arbitration time of the different table possibilities. Note that the increment in time refers to both, the minimum and maximum arbitration time required by the scheduler. Specifically, the figure shows the increment in silicon area and time respect the original AS table scheduler. In all the cases, a table of 128 entries with a parallelization grade of 16, 8 VCs, and a MTU of 32 is considered. Figure 7.10 shows that employing a fixed weight for all the table entries (*FixedW*), which solves the problem of the original table scheduler with variable packet size, only requires 10% more silicon area than the original AS table scheduler (*Original*). If we want to be able to employ the decoupling configuration methodology we can choose between using a weight per each VC (*W-VC*), using the three reserved bits of each table entry (*3-bits*), or using two bytes to store the VC identifier and the table entry (2-bytes). Figure 7.10 shows that the 2-bytes option is the most demanding one. This option requires 80% more silicon area than the original AS table compared with the 35% of the 3-bit option. Moreover, the arbitration time is slightly higher (0.85%) than in the rest of the cases, which have the same arbitration time, and thus the increment is 0%. In the rest of this work we will show statistics of the 2-bytes DTable option. It can be considered the worst case for all the table implementations. Moreover, this is the possibility that provides the best flexibility and granularity.

Figure 7.11 shows the effect of the number of VCs over the complexity of a DTable with 128 entries, a parallelization grade of 16, and a MTU of 32. Specifically, it shows the increment in silicon area and arbitration time required respect the 2-VC case. This figure

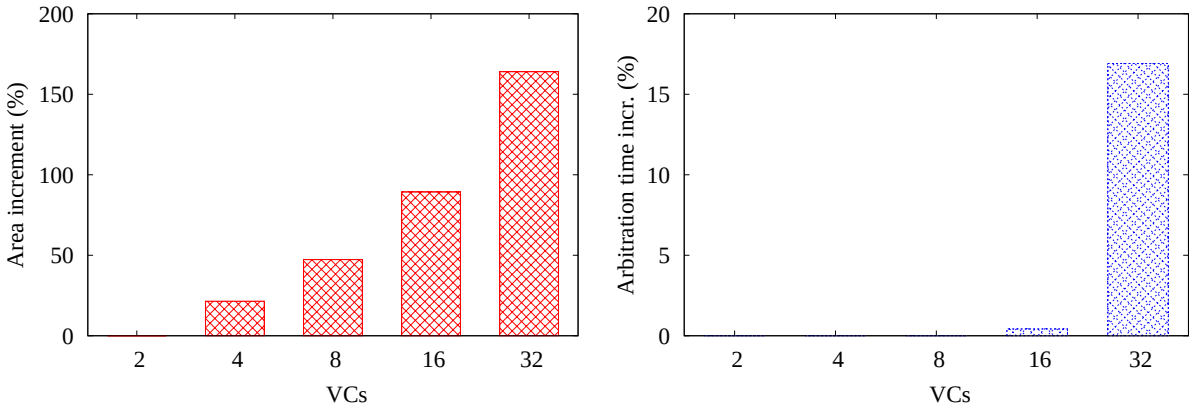


Figure 7.11: Effect of the number of VCs over the silicon area and arbitration time required by the DTable scheduler.

shows that this parameter affects in a high degree the silicon area required and, when the number of VCs is very high, a little the arbitration time. However, the effect is not so dramatic as in the DRR-CA and SCFQ-CA cases. Note that for 2 to 8 VCs the arbitration time is the same, and thus the increment is 0%. The reason because the number of VCs does not affect as much the complexity as in the DRR or SCFQ cases is that in the DTable case the scheduling is made over the arbitration table and not over a list of VCs, like in the DRR-CA case where we search for the next active VC, or the SCFQ-CA, where we search for the VC with the minimum service tag.

Figure 7.12 shows the effect of the MTU value over the complexity of a DTable with 128 entries, a parallelization grade of 16, and 8 VCs. Specifically, it shows the increment in silicon area and arbitration time required respect the 2-MTU case. This figure shows that the MTU is almost irrelevant for the silicon area and arbitration time required by this scheduler.

Figure 7.13 shows the effect of the number of table entries over the complexity of a DTable with a parallelization grade of 16, when the MTU is 32 and there are 8 VCs. Specifically, this figure shows the increment in silicon area, cycle time, and minimum and maximum time required to perform the arbitration respect the silicon area and minimum time required in the 32-entry case.

Figure 7.13 shows that this parameter affects in a high degree both the silicon area and the arbitration time. The increment in the silicon area is due to the increment in the space required to store the arbitration table and the extra logic to handle it. The increment in the arbitration time is due to the increment in the cycle time, but also to the extra number of cycles required to process a bigger table. Specifically, the increment

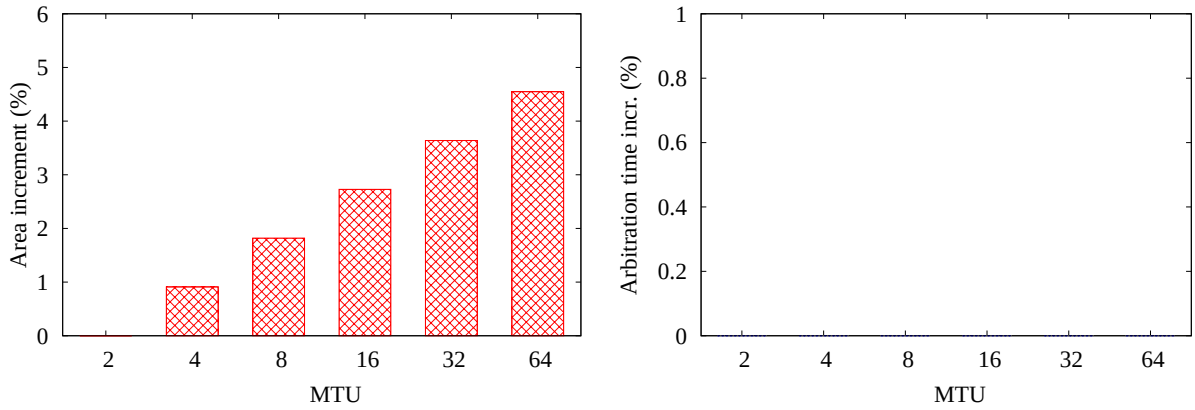


Figure 7.12: Effect of the number of the MTU over the silicon area and arbitration time required by the DTable scheduler.

in the cycle time determines the increment in the minimum time required to make the arbitration. Note that we use the same parallelization grade in all the cases and thus, the same minimum number of cycles is required to perform the arbitration (see Table 7.2). On the other hand, the maximum number of required cycles increases with the table size and thus, the maximum required time increases dramatically.

A way to reduce the arbitration time is to increase the parallelization grade. Figure 7.14 shows the effect of this parameter over a DTable of 128 entries, 8 VCs, and a MTU of 32. Specifically, this figure shows the increment in silicon area, cycle time, and minimum and maximum time required to perform the arbitration, respect the silicon area and minimum time required when the parallelization grade is 1 (sequential search) This figure shows that increasing the parallelization grade also increases in a high degree the silicon area required. This extra area is not so exacerbate when we increase only a bit the parallelization grade. However, if we increase the value of this parameter a lot, the silicon area increases much faster. Given a certain number of entries (128 in this case), the effect of increasing the parallelization grade is to reduce the maximum number of cycles required to perform the arbitration at the cost of increasing the minimum number of cycles required (see Table 7.2). This effect is shown in Figure 7.14. However, this figure shows that increasing too much the parallelization grade affects in a negative way both the minimum and maximum arbitration time because of the increment in the cycle time.

Until now we have shown the individual effects of varying the value of the different design parameters over a basic configuration of a 2-bytes DTable with 128 entries, a parallelization grade of 16, a MTU of 32, and 8 VCs. Figure 7.15 shows a more general picture in which we observe the effect of varying the number of VCs for every table size.

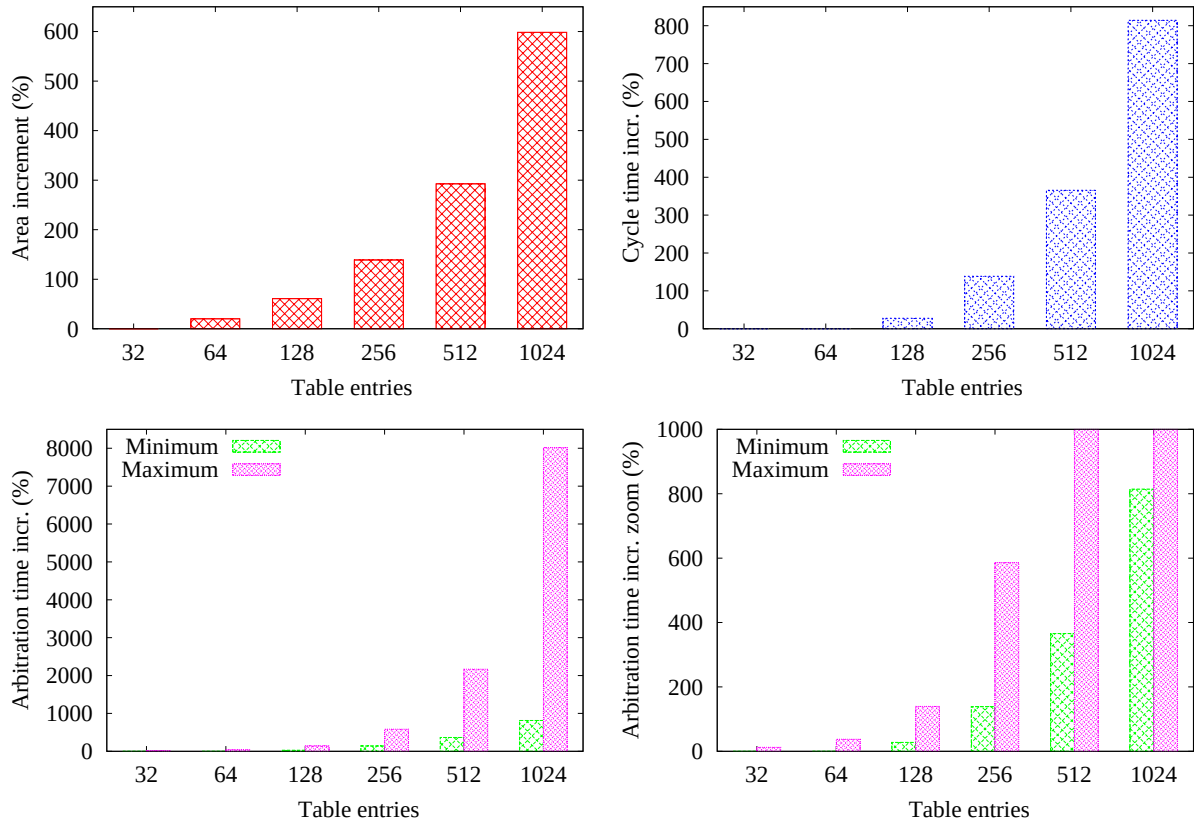


Figure 7.13: Effect of the number of table entries over the silicon area and arbitration time required by the DTable scheduler.

At the same time we vary the parallelization grade in order to keep constant and equal to 16 the number of cycles required to process all the table entries (number of entries / parallelization grade = 16). Note that even with this last consideration, the number of cycles is not the same in each combination of number of entries and parallelization grade (see Table 7.3). The increments shown are respect a DTable with 32 entries and 2 VCs.

Figure 7.15 shows that when the number of table entries grows, the silicon area required increases dramatically due to the accumulated effect of the increment on the table size and the parallelization grade. However, even increasing the parallelization grade the arbitration time also grows a lot due to the increment on the cycle time. A smaller arbitration time could be achieved increasing more the parallelization grade, however, this would increase even more the silicon area required. Figure 7.15 also shows that the number of VCs is only relevant for the arbitration time for small arbitration table sizes. When the arbitration table has lots of entries, the number of VCs does not affect the cycle time and thus, the arbitration time.

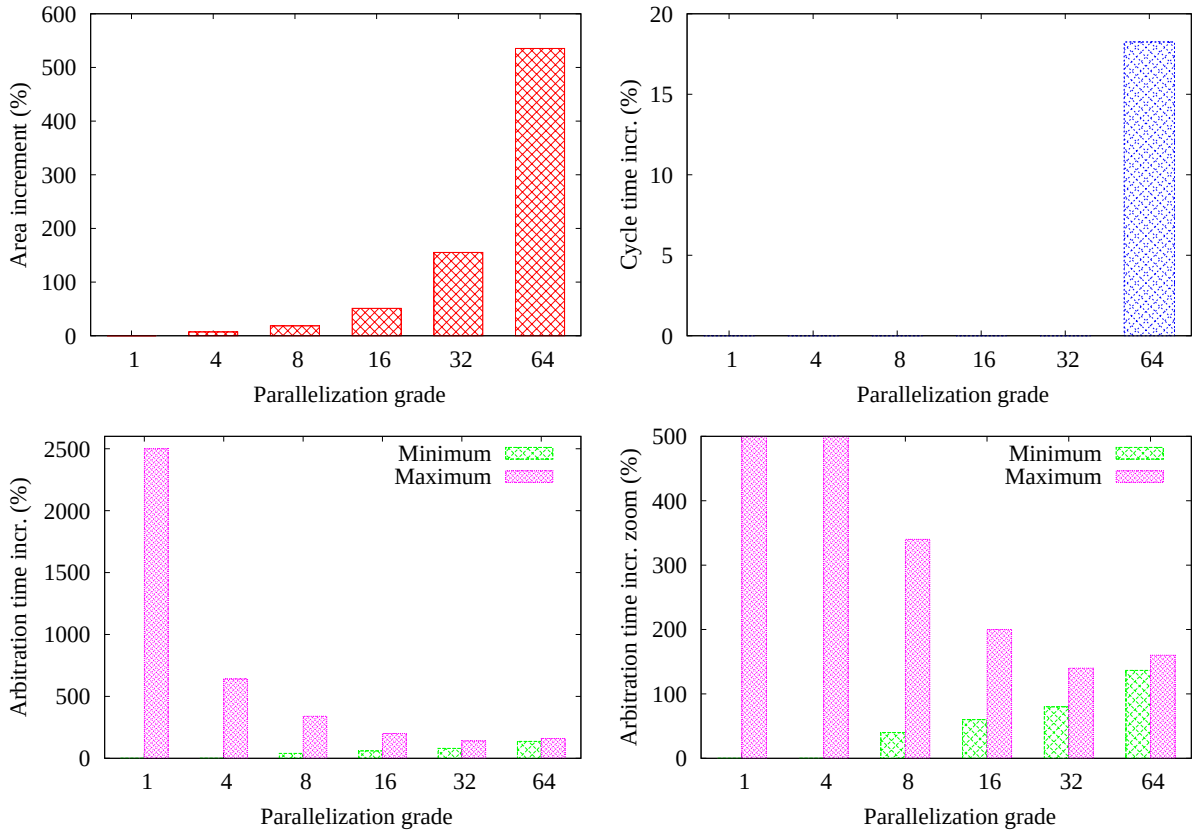


Figure 7.14: Effect of the parallelization grade over the silicon area and arbitration time required by the DTable scheduler.

Table 7.3: Combination of values for the table entries and parallelization grade and arbitration time in cycles.

Number of table entries	Parallelization grade	Arbitration time (cycles)
32	4	6 - 13
64	8	7 - 14
128	16	8 - 15
256	32	9 - 16
512	64	10 - 17
1024	128	11 - 18

7.5 Comparing the MinBW and DTable schedulers

In the previous sections we have shown how the different design parameters affect the complexity, in terms of silicon area and arbitration time, of the DRR-CA and SCFQ-CA

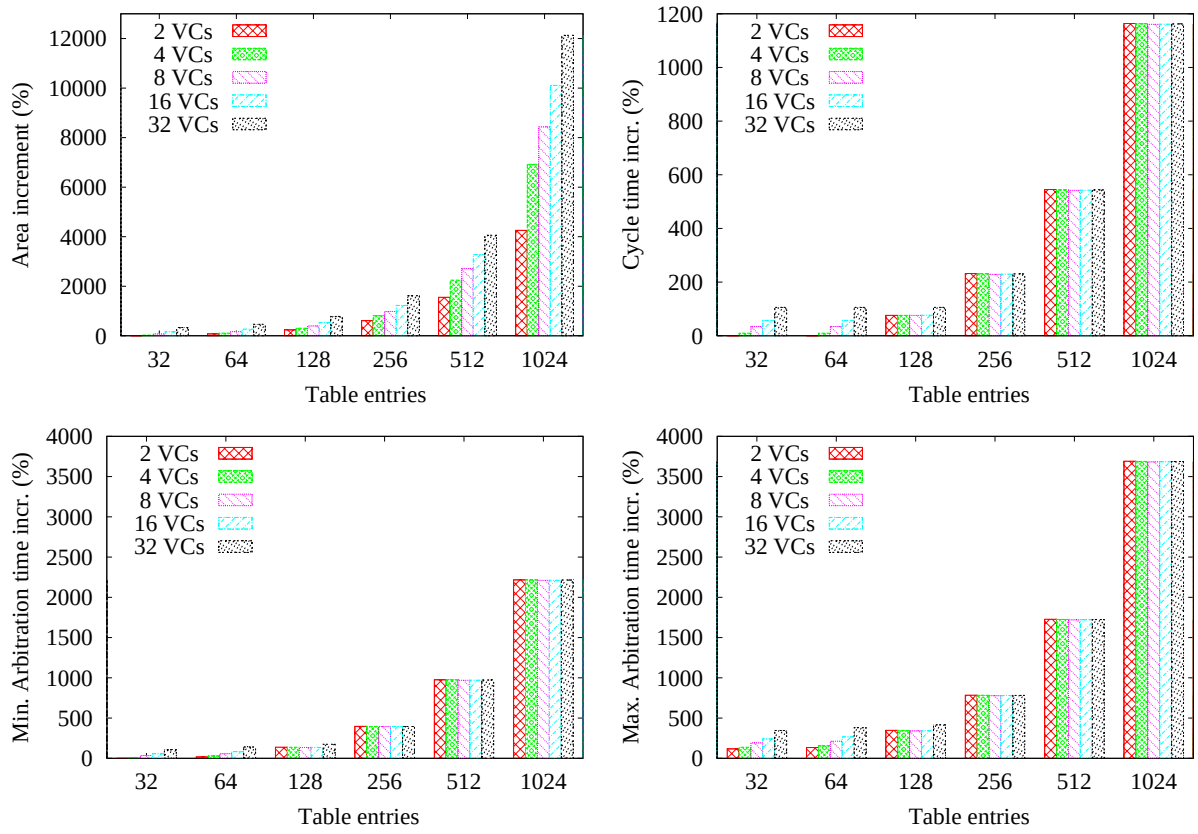


Figure 7.15: Silicon area and arbitration time increment for the combined effect of the number of table entries and number of VCs for the DTable scheduler.

implementations of the MinBW scheduler and the DTable scheduler. In this section we are going to compare the complexity of these schedulers.

Figures 7.16 and 7.17 show a comparison of the silicon area and arbitration time of the different schedulers required with different number of VCs and, in the case of the DTable scheduler, different number of table entries (we have also kept number of entries / parallelization grade = 16). Note that not all the possible combinations of number of VCs and number of table entries make sense. If we have a lot of VCs, we will probably need more table entries to accommodate appropriately all those VCs. Note, for example, that in an extreme case where we have 32 VCs and 32 entries, we should assign each VC to a given table entry and we would not be able to make any latency differentiation. On the other hand, if we have very few VCs, it would be a waste of resources to employ a lot of table entries. Therefore, we have only shown the combination of 2 and 4 VCs with 32, 64, and 128 table entries, and 16 and 32 VCs with 256, 512, and 1024 table entries. For the 8-VC case we show the interaction with the possible table sizes. Moreover, we have split

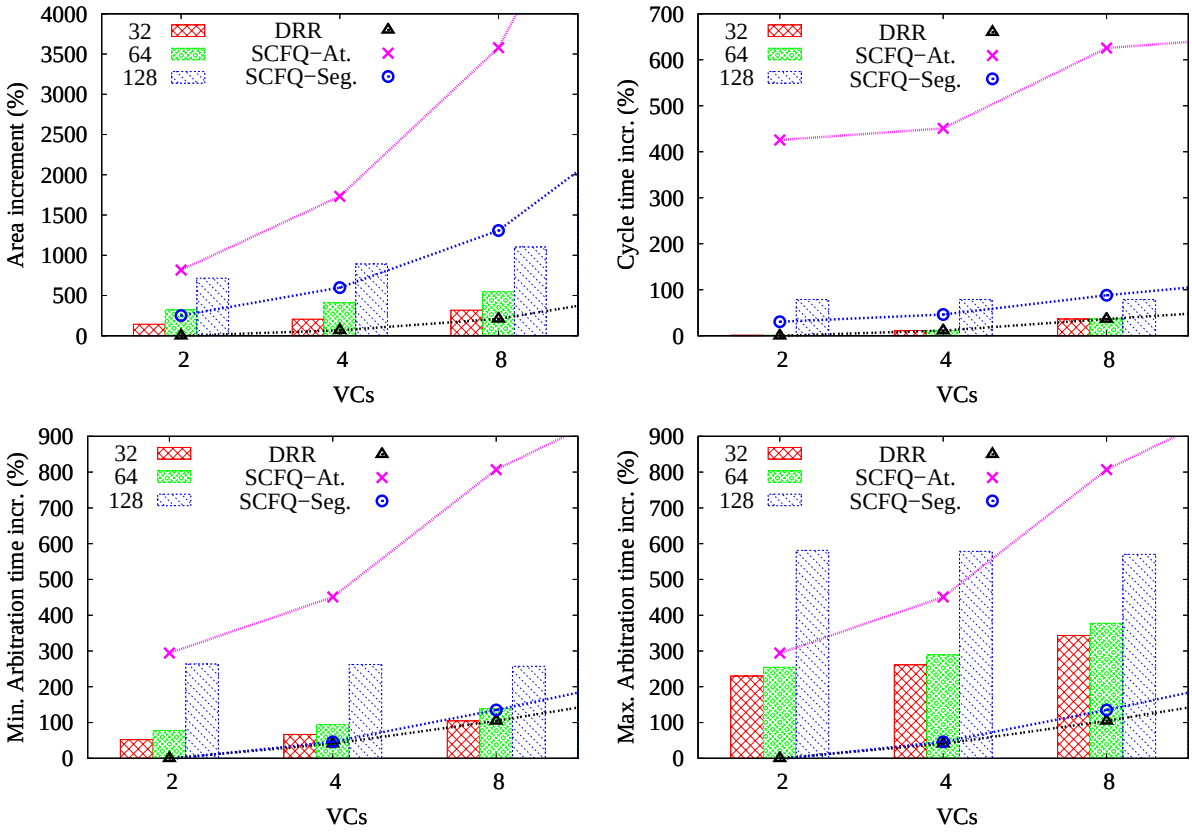


Figure 7.16: Silicon area and arbitration time comparison of the different schedulers with a small number of VCs.

the data in two separate figures in order to show them more clearly. Both figures show the increment on silicon area and minimum and maximum arbitration time required respect the DRR-CA with 2 VCs.

Figure 7.16 shows the comparison of the schedulers for a small number of VCs (2-8) and a small number of table entries (32-128). This figure shows that, as expected, the DRR-CA is the simplest scheduler in terms of both, silicon area and arbitration time. The atomic version of the SCFQ-CA scheduler is the most demanding implementation also in both aspects. Regarding the DTable scheduler and the segmented version of the SCFQ-CA scheduler, Figure 7.16 shows that in general the DTable scheduler requires less silicon area than the segmented SCFQ-CA scheduler. On the other hand, the SCFQ-CA scheduler is faster than the DTable scheduler. However, as stated before, in this comparison we do not take into account the extra time required by the segmented SCFQ-CA scheduler to compute the service tag.

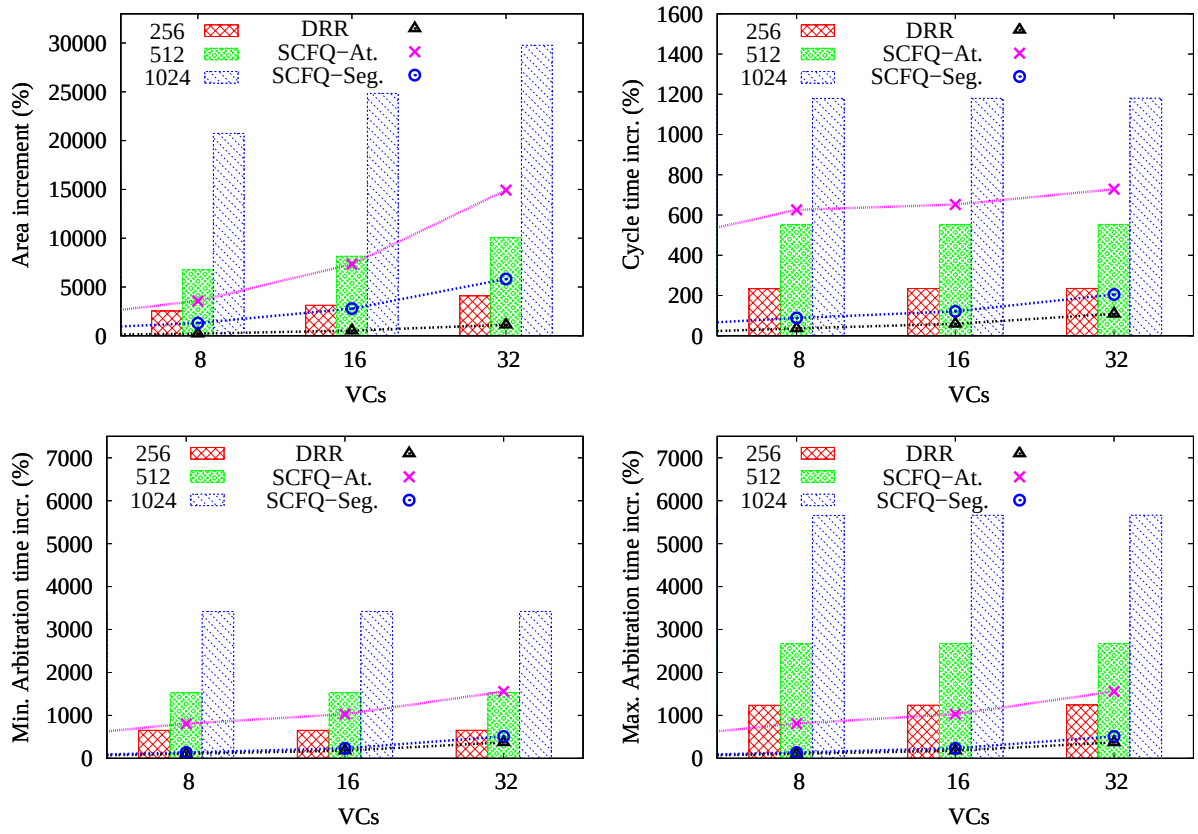


Figure 7.17: Silicon area and arbitration time comparison of the different schedulers with a high number of VCs.

Figure 7.17 shows the comparison of the schedulers for a high number of VCs (8-32) and a high number of table entries (256-1024). This figure shows that the DTable scheduler is the most complex when the number of table entries is 1024. When the table has 512 entries, only if it has 32 VCs it requires less silicon area than the atomic SCFQ-CA scheduler. When the DTable arbitration table has 256 entries this scheduler requires less silicon area than the atomic SCFQ-CA case. The time required in this case by the atomic SCFQ-CA case is in general higher than the minimum time required by the DTable scheduler but smaller than the maximum time. In almost all the cases the segmented SCFQ-CA case and the DRR-CA schedulers require less silicon area than the DTable with a size between 256 and 1024 entries.

7.6 Summary

In this chapter we have studied the complexity of the different possibilities for the MinBW scheduler that we propose in Chapter 5 and the DTable scheduler that we propose in Chapter 6. In order to do so we have implemented the schedulers in Handel-C and obtained hardware statistics employing the DK design suite tool.

We have studied the complexity in terms of silicon area and time required to perform the scheduling. We have obtained hardware estimates for these indices taking into account different values for some design parameters. We have considered the number of VCs and the MTU in all the cases. Moreover, for the DTable scheduler we have also considered the size of the table in terms of table entries and the parallelization grade, which is the number of table entries that we read each cycle. Furthermore, we have also compared the complexity of the different implementation options for the DTable scheduler.

The hardware estimates that we have obtained have shown that the cost of modifying the original AS table to handle in a proper way variable packet sizes is very small (around 10% increment in silicon area). If we want to fully implement the DTable scheduler and being able to apply the decoupling configuration at maximum, we only need to double the silicon area required. This increment compared with the entailed to increase the number of table entries or the parallelization grade is quite small.

The hardware estimates obtained also show that, as expected, the DRR-CA scheduler is the simplest one. The DTable scheduler is in general the most complex option when implementing large arbitration tables, which are required when there are a high number of VCs. However, the DTable scheduler can be a good option, at least in terms of silicon area, when a small number of table entries is implemented (32-256) if compared with the SCFQ-CA scheduler.

Chapter 8

Configuration of the AS mechanisms to provide QoS

AS provides several mechanisms for traffic differentiation and congestion management that can be used to provide QoS. However, the AS specification does not specify how to use these mechanisms in order to do so. In this chapter, we show a general framework to provide QoS over AS that uses some of the AS mechanisms reviewed in Chapter 4. Specifically, we present a traffic classification based on bandwidth and latency requirements, show how to configure the AS egress link scheduler that we studied in Chapters 5 and 6, and employ an admission control (AC) mechanism to ensure QoS provision.

8.1 Traffic classification

As stated before, AS switches differentiate the traffic at a VC level rather than at a flow level. The number of VCs is rather limited if compared with the possible number of flows that can traverse the network in a given moment. Therefore, in order to provide QoS over AS, a limited set of Service Classes (SCs) with different requirements must be specified. When a flow obtains access to the AS fabric, it will be assigned a SC depending on its characteristics.

If there are enough VCs we will devote a separate VC to the aggregated traffic of each existing SC. Note that the maximum number of unicast SCs supported by AS that we can define is 16, which is the maximum number of unicast VCs. Each SC will be identified in the packet header with the Traffic Class (TC) field, which can identify up to

8 TCs, and the Ordered-Only flag, which indicates if the packet must be routed through the Ordered-Only Unicast VCs or the Bypassable Unicast VCs.

In order to define the different SCs, we propose a traffic classification based on three network parameters: Bandwidth, latency, and jitter. In this way, this classification is similar to the one presented by Pelissier [Pel00]. Note that we do not consider packet loss because AS is a lossless network due to its link-level flow control. We distinguish between three broad categories of traffic:

- Network Control traffic: High-priority traffic to maintain and support the network infrastructure. One SC will be dedicated to this kind of traffic.

- QoS traffic: This traffic has explicit minimum bandwidth, maximum latency, and/or jitter requirements. Various QoS SCs can be defined with different specific requirements. This category can be divided into two groups:

  - Traffic which requires a given minimum bandwidth and must be delivered with a maximum latency and/or jitter in order for the data to be useful. Examples of such data streams include video conference, interactive audio, and video on demand.

  - Traffic which requires a given minimum bandwidth but is not particularly sensitive to latency or jitter. An example of this kind of traffic could be a non-interactive playback of a video clip.

- Best-effort traffic: This traffic accounts for the majority of the traffic handled by data communication networks today, like file and printing services, web browsing, disk backup activities, etc. This traffic tends to be bursty in nature and largely insensitive to both bandwidth and latency. Best-effort SCs are only characterized by the differing priority among each other.

The mapping of application requirements into appropriate SCs can be accomplished in two steps. In the first step, the application-level QoS parameters are mapped to a set of network-level QoS parameters such as latency, jitter, and bandwidth. In the second step, these network-level parameters are mapped to one of the available SCs.

Note that, in the case of the latency requirements, as stated in Section 3.1, the maximum delay that can be allowed in the network depends on the latency overhead produced by the other layers of the communication protocol stack employed.

8.2 Scheduler configuration

The schedulers must be properly configured at the different network elements to provide the different SCs with a differentiated treatment. Specifically, we are going to configure the schedulers in order to provide just bandwidth or bandwidth and latency simultaneously. Note that, although they are not totally correlated, if we limit the maximum latency performance, we are indirectly limiting the maximum jitter performance and thus, we can translate any maximum jitter requirement into a maximum latency requirement.

As stated before, if there are enough VCs we will devote a separate VC to the aggregated traffic of each existing SC. The bandwidth that each VC should be assigned depends on the requirements of the SC it has assigned. We should provide the network control SC with enough bandwidth to manage the maximum expected amount of control traffic. QoS VCs should be assigned at least a bandwidth equal to the minimum bandwidth requirements of the QoS SCs. Finally, the bandwidth intended for the best-effort SCs should be assigned among them according to their different priority in order to provide them with a differentiated performance.

However, it is well-known that interconnection networks are unable to achieve 100% global throughput. Therefore, not all the bandwidth can be distributed among the VCs, thereby requiring a certain bandwidth to be left unassigned. We propose to assign the network control VC with this bandwidth that should be left unassigned. Moreover, we propose not to assign best-effort VCs with all the bandwidth that is intended for this class of traffic. We propose instead to assign them only a small amount of bandwidth proportional to their relative priority. The rest of the best-effort bandwidth will also be assigned to the network control VC. In this way the network control VC will have been assigned more bandwidth than it actually requires. However, by doing so, we achieve a better performance of the network control traffic. We also achieve a better performance and a better resilience against unexpected transient congestion due to bursty traffic of the QoS VCs. Note that the bandwidth unused by the control and QoS VCs is redistributed by the scheduler among the rest of VCs, including the best-effort VCs, and thus they are going to take advantage of the bandwidth left over by the other VCs.

If any of the egress links do not implement as many VCs as SCs we have defined, several SCs should be aggregated into the same VC in the affected links. The schedulers that serve those links must provide to each VC the most restrictive QoS requirements of the SCs that it has assigned. This entails providing a minimum bandwidth equal to the sum of minimum bandwidth of the SCs and a maximum latency equal to the minimum maximum latency of the SCs. In the following sections we will show how to configure the

two normative AS schedulers, the MinBW scheduler and the table scheduler, to provide the flows aggregated in the different VCs with bandwidth and latency requirements.

8.2.1 Configuring the MinBW scheduler

In Chapter 5 we have outlined the requirements that must be taken into account to design a possible implementation for this scheduler. Moreover, we have proposed three valid scheduling algorithms: The DRR-CA, the SCFQ-CA, and the WFQ-CA. Providing minimum bandwidth requirements to a VC with the MinBW scheduler is as easy as assigning to that VC a weight equal to the proportion of the egress link bandwidth that it needs. The control SC will be assigned to the FMC in order to achieve the maximum priority, and thus no bandwidth will be assigned explicitly to this SC. However, this bandwidth cannot be assigned to any other VC but left unassigned.

Parekh and Gallager [PG94] analyzed the performance of a queuing network with an ideal fair queuing service discipline and derived upper bounds on the end-to-end delays when the input traffic streams conform to the leaky bucket characterization. In this work, we are not going to conform the traffic to a given pattern, but on the basis of that study, we could assign a higher amount of bandwidth than is needed to those VCs with high latency requirements, in order to obtain a better average and maximum latency performance.

8.2.2 Configuring the fixed weighted DTable scheduler

As stated in Section 6.3.1 in page 98 the simplest way of implementing the DTable scheduler, and solving the AS table problem with variable packet sizes, is to assign each table entry a fixed constant weight. In this case, the minimum bandwidth assigned to a VC is proportional to the number of entries assigned to that VC. However, one of the main advantages of the table scheduler is that it allows us to configure not only the number of table entries assigned to each queue or VC, but also the distribution of the entries assigned to each queue.

Note that although we can assign the network control SC to the FMC when using the table scheduler, this VC does not have maximum priority like in the MinBW case, so we will consider this VC as any other VC with traffic of high latency requirements.

As stated in Section 6.2 in page 91 there are two possible ways of configuring this table-based scheduler:

- If our objective is to provide only bandwidth requirements, we can distribute the table entries as the WF2Q variant of the list-based Weighted Round Robin proposed by Chaskar and Madhow [CM03]. As stated in Section 3.5.2 in page 46 this approach tries to improve the latency performance of all the SCs by emulating the order of transmission if the WF2Q would be implemented.
- If our goal is to provide also latency requirements to any or all the VCs:
  - We can assign the table entries taking into account the maximum distance between any consecutive pair of entries devoted to the VCs with latency requirements (network control and QoS SCs with latency requirements) [ASD04]. We can assign more entries to those VCs that require more bandwidth than they are assigned due to the maximum distance distribution.
  - The rest of table entries can be distributed among those VCs that do not have latency requirements. We can assign those entries consecutively in the remaining gaps or can interleave the entries of the various VCs like in the list-based Weighted Round Robin in order to improve the latency performance.

Note that the original AS table scheduler would be configured in the same way. However, due to the original AS table scheduler problem with variable packet sizes, no guarantees on bandwidth can be provided and thus neither in latency.

8.2.3 Configuring the fully DTable scheduler

As stated before, we can only provide bandwidth requirements with the fixed weighted DTable scheduler. We can provide also latency requirements but at the cost of bounding the bandwidth and latency assignments, which probably entails wasting resources (see Section 6.2). If we want to be able to employ our decoupling configuration methodology, we need to implement a full version of the DTable scheduler. Note that, with our decoupling methodology, the bandwidth that can be assigned to a VC depends not only on the proportion of table entries that it has assigned but also, on two decoupling parameters and the specific MTU of the VC. In Section 6.3 we have proposed three possible ways to adapt the AS table scheduler: To use the 3-bit reserved field of each table entry, to modify the arbitration table structure, and to use the same weight for all the entries of a VC.

When employing a full version of the DTable scheduler, we must first assign the table entries like in the fixed weighted DTable case attending to the latency requirements of the SCs with latency requirements and the bandwidth of the rest of SCs. After that, the bandwidth assignment is performed assigning each entry or VC the appropriate weight.

When selecting the maximum distance for each SC, it must be taken into account that the latency performance depends on the w parameter, which indicates the maximum weight that can be assigned to a table entry. Therefore, an option is to establish the maximum distance between any consecutive pair of entries of the SCs taking into account the maximum w value allowed by the hardware implementation of the DTable scheduler. In this way, when configuring the DTable scheduler, we can choose a smaller w value to improve the latency performance, but, in any case, the maximum latency requirements are going to be guaranteed.

In order to assign a given VC with a minimum bandwidth, the amount of weight units from the bandwidth pool assigned to the VC table entries must accomplish with the proportion of desired egress link bandwidth. Therefore, when we know the maximum distance between two consecutive table entries, and thus, the number of entries, and the amount of bandwidth that we want to assign to each VC, we must choose the w and k parameters that make possible that distribution of bandwidth among the various VCs.

Moreover, we can limit the MTU of some VCs in order to have a smaller minimum bandwidth for those VCs and for being able to use smaller k values. We can assign each VC a different MTU at a communication library level, but this would entail to add complexity to the AS communication protocols. On the other hand, we can take advantage of the AS characteristics to simplify the process. As stated before, AS allows us to establish two different MTUs for the two unicast VC types. Therefore, we can have two sets of VCs with two different MTUs and we can assign the SCs to the VCs taking into account this. Note that those SCs that have high latency requirements, and thus require more table entries, usually have small bandwidth requirements and use small packets. Therefore, we can assign these SCs to the VCs with the smallest MTU.

8.3 Admission control

In order to provide the different SCs with their QoS requirements even at very high network loads, the different network resources must be managed in a proper way. The objective is that the network control SC obtains a good latency; the SCs with bandwidth requirements obtain the amount that they need; the SCs with latency requirements do not exceed the maximum allowed; finally, the best-effort SCs obtain a different bandwidth and latency performance in accordance with their different priority.

In a lossless network like AS, congested packets are not thrown away and thus, the loss-rate due to congestion is zero. This has the advantage of avoiding retransmissions

that would severely affect the latency and jitter performance of the flows. On the case of applications with packet loss resilience, it would allow to reduce the overhead due to the encoding techniques used to minimize the impact of errors.

On the other hand, as stated in Section 3.4 in page 30, lossless networks have other problems, being the most important the creation of congestion (or saturation) trees [PN85]. These congestion trees may produce a dramatic network performance degradation, affecting not only the flows traversing the original point of congestion, but other flows that share common upstream links.

However, if the congestion is not persistent, the congestion situation will dissipate after a short period of time and packets will reach their destinations. Depending on the latency introduced by the congestion a packet may or may not meet their QoS requirements.

Our goals are to efficiently move traffic separated into differentiated SCs and to avoid congestion problems within one or more of these classes even as traffic volume approaches the AS fabric capacity. The way of achieving this is by using an admission control (AC) technique. The AC decides whether a new connection is accepted or rejected and ensures that the acceptance of additional traffic into a network cannot create congestion.

Note that in order to provide QoS guarantees, an AC mechanism must be used. Without an AC it is only possible to obtain a scheme of priorities where some SCs would have a higher priority than others, but no guarantee could be given.

AS specification just cites the AC as a possible mechanism to be used, but does not give any indication of how to implement it. However, probe-based algorithms are limited by a traffic awareness that is restricted to the traversal route while fluctuating traffic patterns, especially within a busy network, provide limited temporal information describing the network load. On the other hand, in the measurement-based approach, the collection and calculation of statistical data can be both costly to gather and process.

Therefore, we propose to use an *a priori-based* AC, see Section 3.4.6 in page 34. Specifically, we propose to employ an AC mechanism that relays on additive effective bandwidths to take the accept/reject decision. This solution assumes that both topology and routing information about network is available. In AS, this information is obtained by the network manager during the initialization network process. Moreover, the flows must use the same path during all their life. This is possible in AS due to its source-based routing. We will call this AC mechanism *bandwidth broker*. As we will see, the bandwidth broker configuration is intimately linked with the scheduler configuration.

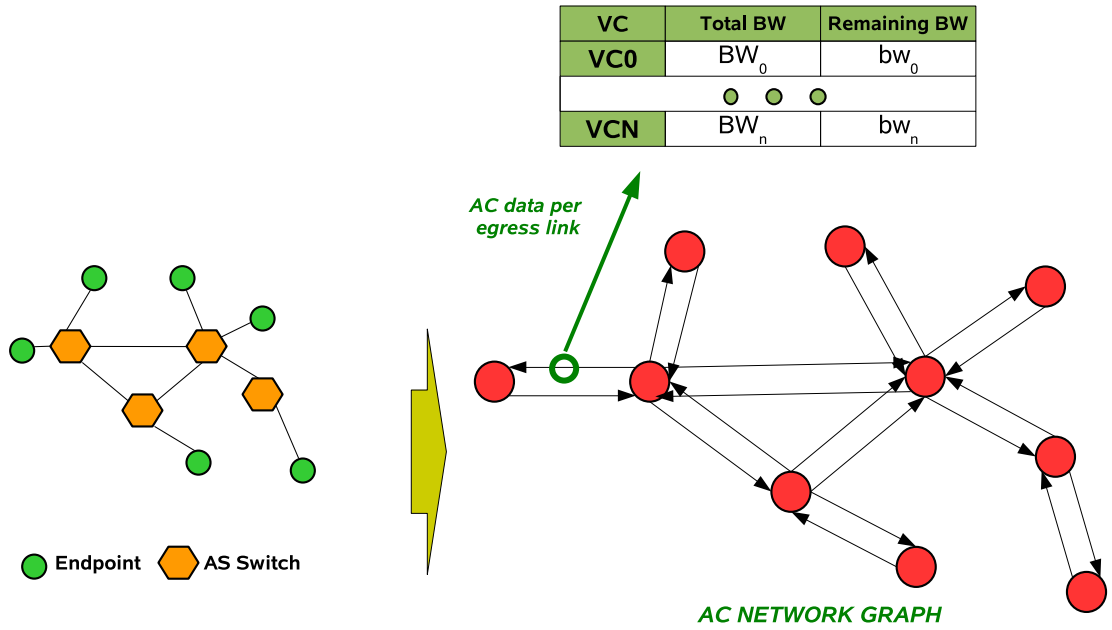


Figure 8.1: Example of the graph required by the bandwidth broker.

8.3.1 The bandwidth broker mechanism

The bandwidth broker mechanism must maintain a graph of the network egress links reporting the available free bandwidth per VC on each link. Figure 8.1 shows an example of such a graph. Note that the bandwidth allocation performed by the egress link scheduler is made at a VC level and not at a SC level. When a new connection tries to get access to the network, an effective bandwidth requirement is assigned to it. Then, the bandwidth broker checks if there is enough bandwidth available all along the path of that connection. This means to check if there is available bandwidth for the VC that the connection is going to employ in each link depending on its SC and the number of VCs employed in that link. If all the links have enough bandwidth to accommodate this new connection, the required bandwidth is subtracted from the available bandwidth for the appropriate VC of those links and the new connection is accepted. If any of the links has not enough bandwidth to accommodate the new flow the connection is rejected.

Note that the AS source-based routing allows this AC approach to not need specific flow information in the switches in order to make sure that each flow uses always the same path through the network. Switches must only maintain the configuration of the output schedulers, which is made at a VC level. In this way, this AC approach is an end-to-end mechanism that can be implemented in a centralized manner, which has all the brokering information in a single host, or in a distributed manner. In [HS05] a distributed

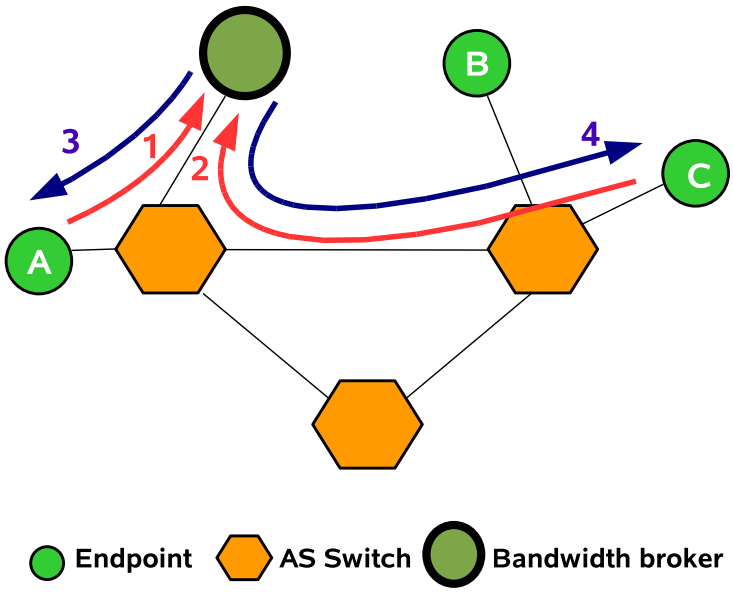


Figure 8.2: Example of a centralized bandwidth broker AC mechanism.

bandwidth broker that takes advantage of the AS multicast capability to keep actualized the state graph is proposed. In this chapter, we will suppose that the network manager acts as a centralized bandwidth broker. Figure 8.2 shows an example of a centralized bandwidth broker. In this example the bandwidth broker attends sequentially two different requests for establishing new connections, answering the sources of the new connections with the decision of accepting or rejecting the requests.

8.3.2 Brokered and unbrokered traffic

One of the main problems of employing an AC mechanism is the connection establishment procedure overhead. Applying this mechanism when trying to initiate every single flow can produce an excessive overhead. However, as stated before, AS defines the credit-based flow control and the scheduling mechanisms at the VC level. This provides a certain degree of isolation to the traffic traversing one VC regarding the traffic of the rest of VCs. Specifically, it allows devoting a certain minimum proportion of the link bandwidth to each VC. This allows us to apply the AC mechanism to avoid the appearance of congestion trees only within a subset of VCs. Therefore, even in the case that congestion trees appear in the rest of VCs, the traffic of the managed VCs will not be affected.

Therefore, we propose to apply the AC mechanism only to those VCs employed by the QoS SCs, which are the VCs that actually require guarantees in terms of explicit

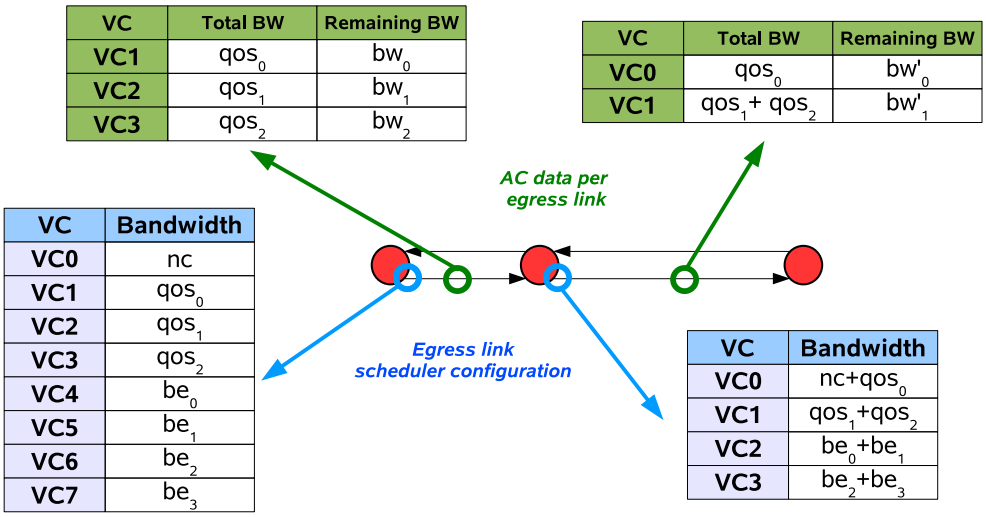


Figure 8.3: Example of a network with different number of VCs it its links.

QoS indices, and not to the control SC or the best-effort SCs. Note that, although the control traffic has high latency requirements, its latency constraints are not so explicit. Moreover, we can assume that the amount of control traffic that is going to traverse the network is going to be quite small. And thus, taking into account the maximum amount of expected control traffic, the scheduling algorithm can assign the network SC with an *a priori* amount of bandwidth.

Figure 8.3 shows an example in which we have 8 SCs (NC, QoS0, QoS1, QoS2, BE0, BE1, BE2, and BE3) and links with 8 and 4 VCs. This figure shows that the bandwidth broker only handles the traffic traversed through the VCs devoted to QoS0, QoS1, and QoS2, which are the SCs with explicit QoS requirements. Figure 8.3 also shows an example of how traffic from the different SCs could be aggregated in a smaller set of VCs and the effect over the bandwidth broker. Note that we cannot combine brokered and unbrokered traffic in the same VC. If we would do this, the unbrokered traffic should become brokered and should be handled by the bandwidth broker. The only exception to this is the network control traffic that can share a VC with QoS traffic because is expected to be below a certain level.

Figure 8.3 also shows the interaction between the bandwidth broker and the egress link scheduler. The maximum link bandwidth that the bandwidth broker can distribute among the connections is determined by the minimum bandwidth assigned to that VC by the egress link scheduler.

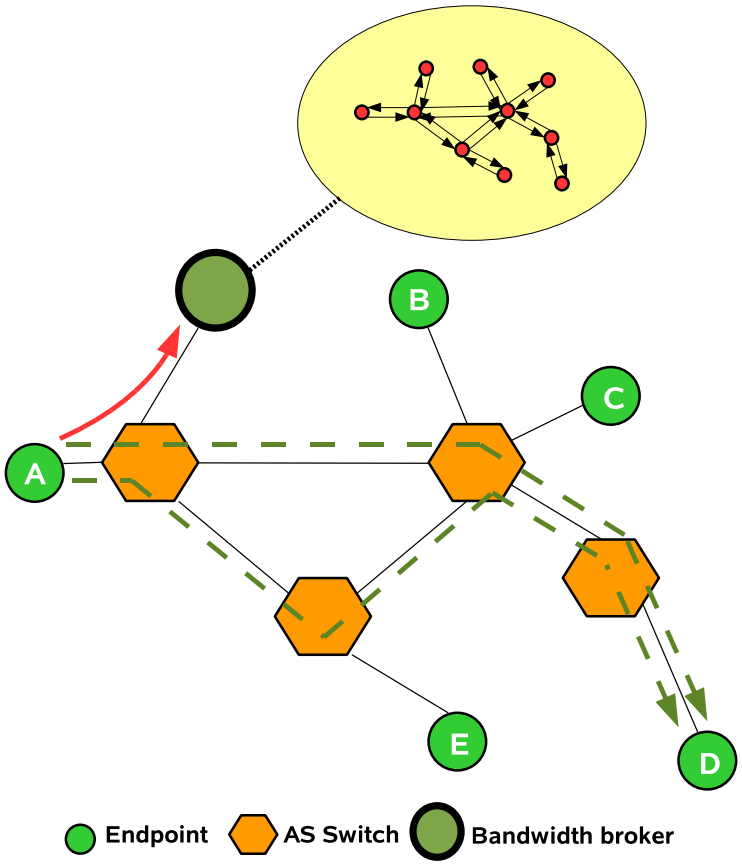


Figure 8.4: Example of multiple possible paths to the same destination.

8.3.3 Path selection and load balancing

As stated before, during the discovery process the network manager obtains knowledge about the topology of the network. With this knowledge and employing an appropriate routing algorithm, the network manager establishes the possible paths among any source and destination. In the AC process previously described the first path that meets the bandwidth requirements is selected as the path for the new connection.

However, the AC mechanism can also be employed to implement a load balancing mechanism. In this case, the AC mechanism, or other management mechanism with the AC support, would be the responsible for selecting the best path attending to the load of the different paths that are allowed by the AC mechanism. This would allow us to provide a better performance by balancing the load of the network.

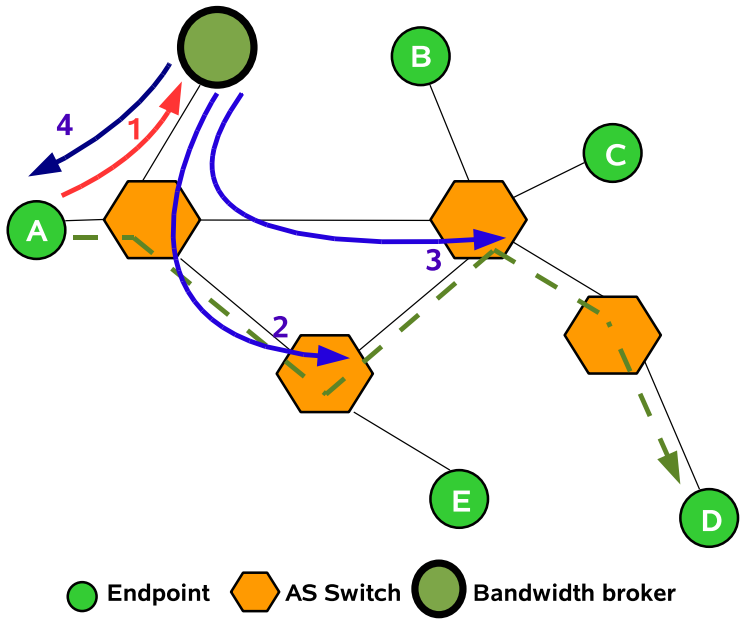


Figure 8.5: Example of dynamic bandwidth distribution.

Figure 8.4 shows an example in which node A requests permission to establish a new connection with node D. There exists two possible paths between node A and node D and thus, the bandwidth broker must select which of the two paths is employed.

.

8.4 Schedulers and bandwidth broker management

There are two possible ways of configuring the schedulers at the network elements and the bandwidth broker. The first possibility is to configure the schedulers and the bandwidth broker in advance, defining a set of SCs with a different minimum bandwidth and maximum latency reservation [RSJS03]. This would entail assigning each VC with a specific weight in the case of the MinBW scheduler, or assigning each table entry with a given VC identifier and weight, in the table scheduler case. The bandwidth broker would be configured attending to the bandwidth assignments for the QoS VCs. This distribution would be made taking into account the requirements and expected amount of traffic of the SCs that traverse each VC.

The second possibility [ASD03] is to configure the schedulers and bandwidth broker in accordance with the connection requirements in a dynamic way. An initial configuration would be made like in the previous case. However, if the bandwidth broker mechanism

determines that there is no path with enough bandwidth available for a new connection, a network management mechanism may modify the configuration of the schedulers in the path to accommodate the new request if there is available bandwidth from other VCs. Of course, the bandwidth broker would be also actualized with the new bandwidth distribution. Note that this modification is only necessary if the resources actually need to be moved from one VC to other VC. This second approach allows more flexibility and a more accurate use of the resources.

In the case of employing the DTable scheduler, this second possibility can be implemented in two ways.

- We can modify both the distribution of the table entries and the weights assigned to them.

- In the second one, we fix the distribution of the table entries, and thus the maximum latency performance properties of each VC, and modify the bandwidth assignation in a dynamic way. To do this, we distribute the weight units from the bandwidth pool among the VCs in a dynamic way taking into account the minimum and maximum bandwidth that the decoupling configuration methodology allows us to assign to each VC. In this last case the reconfiguration of the arbitration table is much faster than if we modify also the distribution of the table entries.

Figure 8.5 shows an example in which node A requests a new connection up to node D. The current configuration of the schedulers would not allow this new connection to be established, and thus, in a static configuration situation, the new connection should be rejected. However, in this dynamic configuration environment the network manager modifies two of the schedulers in the path of the new connection in order to accommodate it.

8.5 Summary

In this chapter we propose how to configure some of the AS mechanisms to provide QoS requirements based on bandwidth, latency, and jitter requirements. In order to do so, we have presented a traffic classification based on those QoS parameters. Specifically we distinguish among three broad categories of traffic: Network control, QoS, and best-effort traffic. In order to provide QoS over AS we must define a set of Service Classes (SCs) that fit in any of those traffic categories. The SCs must then be assigned to the different VCs,

which are the units that are going to be considered by the egress link scheduling and the link-level flow control mechanism.

In this chapter we have also show how to configure the MinBW and table schedulers in order to provide the VCs with their requirements. Finally, we propose to employ an admission control (AC) mechanism to provide the QoS SCs with their requirements. Specifically, we propose to employ an *a priori* AC mechanism that relays on additive effective bandwidths to take the accept/reject decision.

Chapter 9

Performance evaluation

In this chapter, we evaluate thoroughly our proposals, comparing the performance of the four possible scheduling mechanisms that we have proposed for AS: The DTable scheduler and the three possible implementations of the MinBW scheduler. Specifically, we compare their throughput, latency, and jitter performance. For this purpose, we have developed a detailed simulator that allows us to model the network at the register transfer level following the AS specification.

9.1 Simulated architecture

In order to test our proposals we have simulated a perfect-shuffle Bidirectional Multi-stage Interconnection Network (BMIN) with 64 endpoints connected using 48 8-port switches (3 stages of 16 switches). This network topology is shown in Figure 9.1. In AS any topology is possible, but we have used a MIN because it is a common solution for interconnection in current high-performance environments [TB03]. In our tests, the link bandwidth is 2.5 Gb/s but, with the AS 8b/10b encoding scheme, the maximum effective bandwidth for data traffic is only 2 Gb/s.

Figure 9.2 shows the switch model that we have employed. We have chosen a combined input-output buffer architecture with a crossbar to connect the buffers. This is the architecture employed in the AS StarGen's *Merlin* switch [Sta04], see Figure 9.3. The Merlin switch was one of the few commercial products that appeared before the ASISIG was disbanded.

Virtual output queuing has been implemented to solve the head-of-line blocking problem at switch level [AOS93]. We are assuming some internal speed-up (x1.5) for the

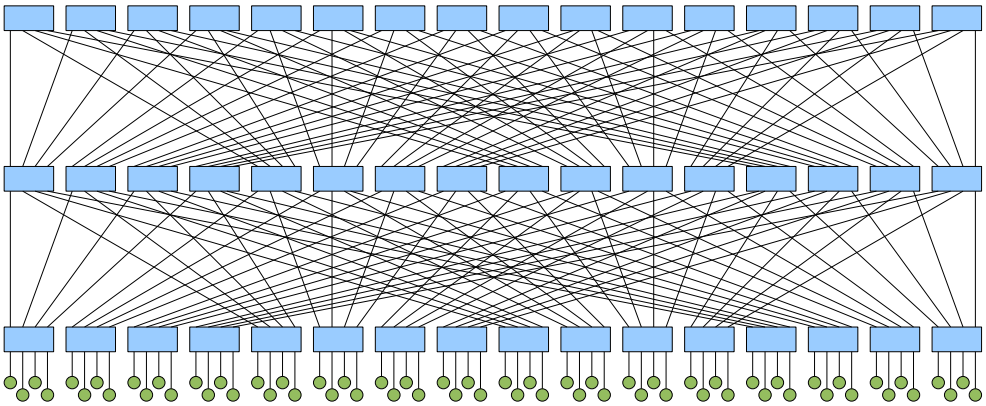


Figure 9.1: Perfect-shuffle BMIN with 64 end-points.

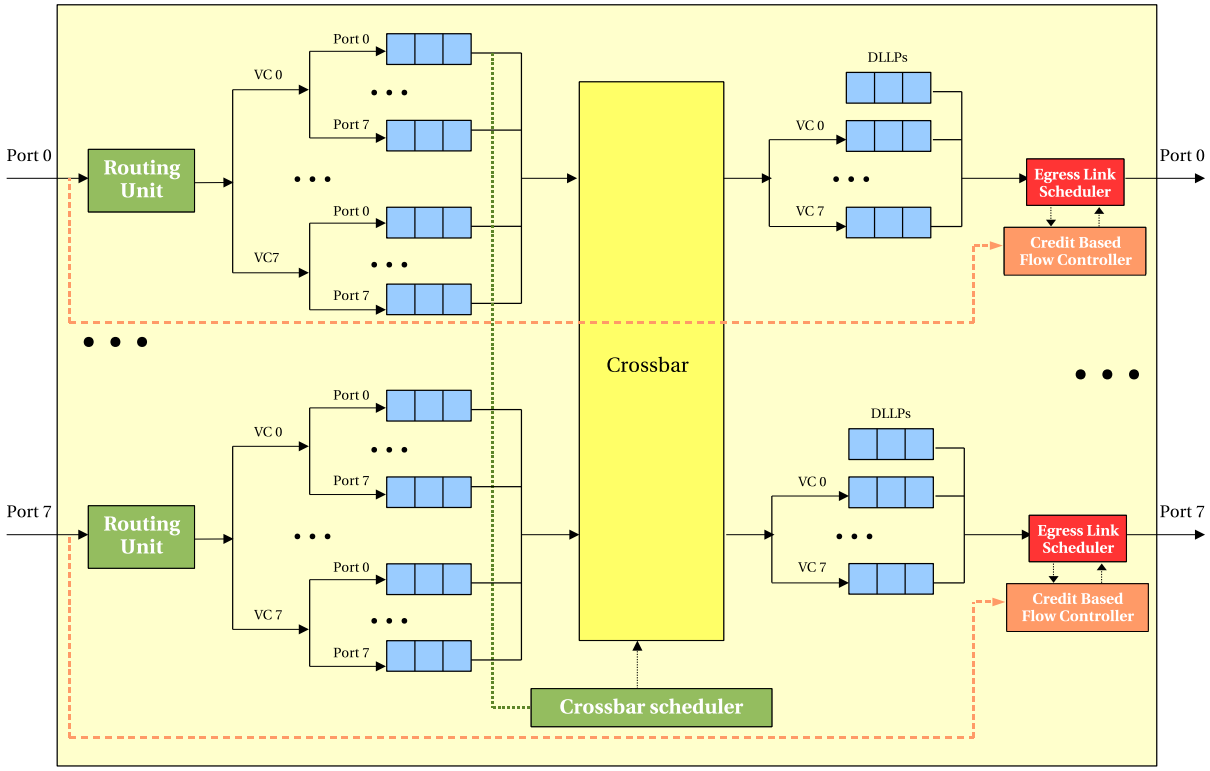


Figure 9.2: Switch model.

crossbar, as it is usually the case in most commercial switches [KPS04, KPC99]. AS gives us the freedom to use any algorithm to schedule the crossbar, so we have implemented a round-robin scheduler. The time that a packet header takes to cross the switch without any load is 145 ns, which is based on the unloaded cut-through latency of the AS StarGen's *Merlin* switch.

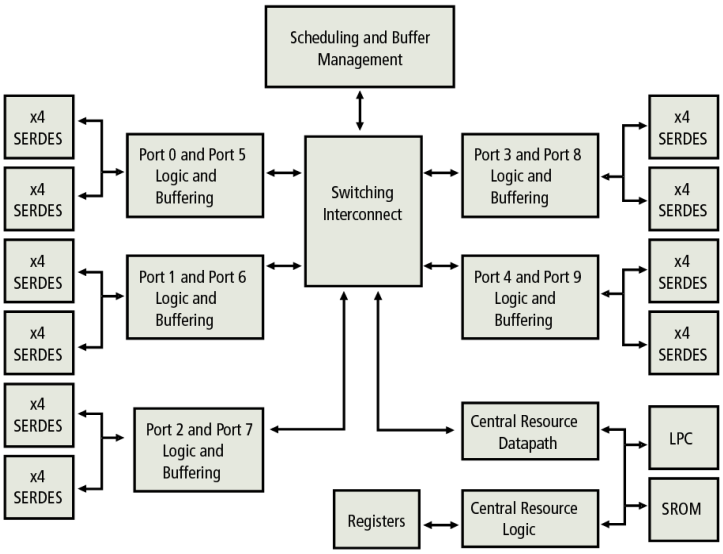


Figure 9.3: Merlin switch functional block diagram.

As stated before, a credit-based flow control protocol ensures that packets are only transmitted when there is enough buffer space at the other end to store them, making sure that no packets are dropped when congestion appears. Virtual Channels (VCs) are used to aggregate flows with similar characteristics and the flow control and the arbitration is made at VC level.

The MTU is 2176 bytes. The credit-based flow control unit is 64 bytes, and thus, the MTU corresponds to 34 credits. The buffer capacity is 17408 bytes (8×MTU) per VC both at the input and at the output ports of the switches. If an application tries to inject a packet into the endpoint but the appropriate buffer is full, we suppose that the packet is stored in a queue of pending packets at the application layer. When enough space is available, the pending packets are transferred to the endpoint. Therefore, endpoints can be considered as having unlimited buffer space. Figure 9.4 shows the endpoint model that we have employed, which is a simplified version of the switch model.

9.2 Performance metrics

Most figures of this performance evaluation show the average values and the confidence intervals at 90% confidence level of ten different simulations performed at a given input load. We have considered the next QoS indices for this performance evaluation:

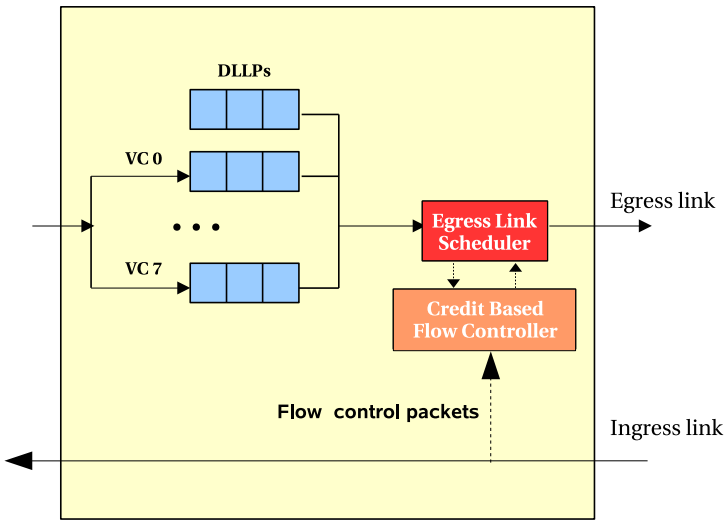


Figure 9.4: Endpoint model.

- 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.
  - Some application messages are larger than network 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, specifically, the 99th percentile.
  - For some load points we also show the cumulative distribution function (CDF) of the latency, which represents the probability of a packet achieving a latency equal to or lower than a certain value.
- Jitter. The jitter measures the variation of latency. However, there is not actually a consensus on how to measure jitter. We use the absolute difference between the delays of two consecutive packets belonging to the same connection [ECT05]. 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.

- Information loss. No statistics on packet loss are given because, as it has been said, AS employs a credit-based flow control mechanism to avoid dropping packets.

9.3 Latency differentiation provided by the schedulers

In this section, we study the capacity of the different schedulers to provide a differentiated latency performance to the various Service Classes (SCs). Specifically, we compare the latency performance of the different DTable scenarios with a different w parameter (DTable1, DTable2, DTable4, and DTable8) showed in Section 6.2 in page 91 with the performance provided by the SCFQ-CA and the DRR-CA schedulers. In order to do so, the traffic pattern of all the SCs must be the same to make in each scenario a fair comparison.

9.3.1 Simulated scenario and scheduler configuration

We have considered 7 VCs with different distances between any pair of consecutive entries in the arbitration table. In a real case we would assign the traffic flows to these VCs depending on their latency requirements. Note that we are going to consider the requirements of a VC as the requirements of the traffic that is going to be transmitted using that VC. We have called these VCs D2, D4, D8, D16, D32, D64, and D64', indicating the distance between any pair of consecutive table entries. Therefore, D2 has stricter latency requirements than D4, D4 than D8, and so on. A table of 64 entries has been used in the simulations. Note that in these tests we have employed a MTU of 32 flow control credits for simplicity.

As stated before, we are going to compare the performance of the DTable scheduler using different values for the w parameter (DTable1, DTable2, DTable4, and DTable8) with the performance of the SCFQ-CA and DRR-CA algorithms. Tables 9.1 and 9.2 show the percentage of entries assigned to each VC and the minimum and maximum bandwidth that can be assigned to each VC in each scenario. This values depend on the values of the w and k parameters, and the specific MTU value of each VC. Note that all the scenarios have the same maximum bandwidth values, differing only in the minimum bandwidth values.

Table 9.1: DTable4 and DTable8 configuration scenarios.

		DTable4		**DTable8**	
		$k = 2$, $w = 4$		$k = 4$, $w = 8$	
VC	**%entries**	$min\phi_i$	$max\phi_i$	$min\phi_i$	$max\phi_i$
D2	50	0.25	1	0.125	1
D4	25	0.125	0.5	0.0625	0.5
D8	12.5	0.0625	0.25	0.03125	0.25
D16	6.25	0.03125	0.125	0.015625	0.125
D32	3.125	0.015625	0.0625	0.0078125	0.0625
D64	1.5625	0.0078125	0.03125	0.00390625	0.03125
D64'	1.5625	0.00708125	0.03125	0.00390625	0.03125
Total	100	0.5	2	0.25	2

Table 9.2: DTable1 and DTable2 configuration scenarios.

			DTable1		**DTable2**	
			$k = 0.5$, $w = 1$		$k = 1$, $w = 2$	
VC	**%entries**	MTU_i	$min\phi_i$	$max\phi_i$	$min\phi_i$	$max\phi_i$
D2	50	$MTU/32$	0.03125	1	0.015625	1
D4	25	$MTU/32$	0.015625	0.5	0.0078125	0.5
D8	12.5	$MTU/16$	0.015625	0.25	0.0078125	0.25
D16	6.25	$MTU/8$	0.015625	0.125	0.0078125	0.125
D32	3.125	$MTU/4$	0.015625	0.0625	0.0078125	0.0625
D64	1.5625	$MTU/2$	0.015625	0.03125	0.0078125	0.03125
D64'	1.5625	MTU	0.03125	0.03125	0.015625	0.03125
Total	100		0.140625	2	0.07	2

Table 9.3: Bandwidth configuration of the DTable scheduler scenarios.

		DTable1		**DTable2**		**DTable4**		**DTable8**	
VC	ϕ_i	E. w.	T. w.	E. w.	T. w.	E. w.	T. w.	E. w.	T. w.
D2	25	8	256	16	512	32	1024	64	2048
D4	25	16	256	32	512	64	1024	128	2048
D8	25	32	256	64	512	128	1024	256	2048
D16	12.5	32	128	64	256	128	512	256	1024
D32	6.25	32	64	64	128	128	256	256	512
D64	3.125	32	32	64	64	128	128	256	256
D64'	3.125	32	32	64	64	128	128	256	256
Total	100		1024		2048		4096		8196

Table 9.3 shows the amount of bandwidth ϕ_i that we have actually assigned to each VC. This table also shows the configuration of the different DTable scenarios. Specifically, this table shows the total weight (T. w.) that we have distributed among the table entries of each VC and the weight assigned to each table entry (E. w.) of each VC. For example, in the DTable1 case, the bandwidth pool is 1024 credits ($k = 0.5$), and thus, in order to assign 25% of bandwidth to this VC, 256 credits must be assigned to it. Therefore, 8 credits have been assigned to each one of its 32 table entries.

Regarding the configuration of the SCFQ-CA and DRR-CA schedulers, in order to be able to compare the different schedulers in a fair way, we are going to perform the same bandwidth assignation as in the DTable case. Specifically, we have assigned each VC a weight equal to the *total weight per VC* that we have in the DTable1 case. These weights can be directly translated into a proportion in the SCFQ-CA scheduler. In the case of the DRR the weight must be translated into quantum units. The minimum weight should be translated into an amount of information equal to the MTU, and the rest of weights should be translated proportionally. However, note that the minimum weight is 32, which actually is the MTU. Therefore, the weights can be directly translated into quantum units expressed in flow control credits.

We are going to inject an increasing amount of traffic of all the VCs and study the throughput and latency performance of the different possibilities at different network load levels. The traffic load is composed of self-similar point-to-point flows of 1 Mb/s. The destination pattern is uniform in order to fully load the network. The packets size is governed by a Pareto distribution, as recommended in [Jai91]. In this way, many small-sized packets are generated, with an occasional packet of large size. The minimum payload size is 56 bytes, the maximum 2040 bytes, and the average 176 bytes, which represents enough packet size variability. The AS packet header size is 8 bytes. The periods between packets are modeled with a Poisson distribution.

9.3.2 Simulation results

The figures of this section show the average values and the confidence intervals at 90% confidence level of ten different simulations performed at a given input load. For each simulation we obtain the normalized average throughput and the average message latency. Note that in the DTable1 and DTable2 scenarios we use specific MTUs for the VCs that are smaller than the general MTU. Therefore, in these cases, a message can be split in several packets. In the rest of cases (DTable4, DTable8, SCFQ, and DRR) a message is going to

be transmitted in only one packet. Note that in the DTable1 and DTable2 scenarios we consider the latency of the message as a whole.

Figure 9.5 shows the normalized injection rate of the aggregated of flows associated with each VC and the normalized throughput results per VC of the DTable1 scenario. The rest of scenarios for the DTable scheduler and the DRR-CA and SCFQ-CA schedulers obtain similar throughput results. As we can see, when the load is low, all the VCs obtain the bandwidth they inject. However, when the load is high (around 95%) the VCs do not yield a corresponding result, obtaining a bandwidth proportional to their assigned bandwidth. Note that the VCs do not obtain all the bandwidth that they were supposed to have assigned because the network is not able to provide 100% throughput.

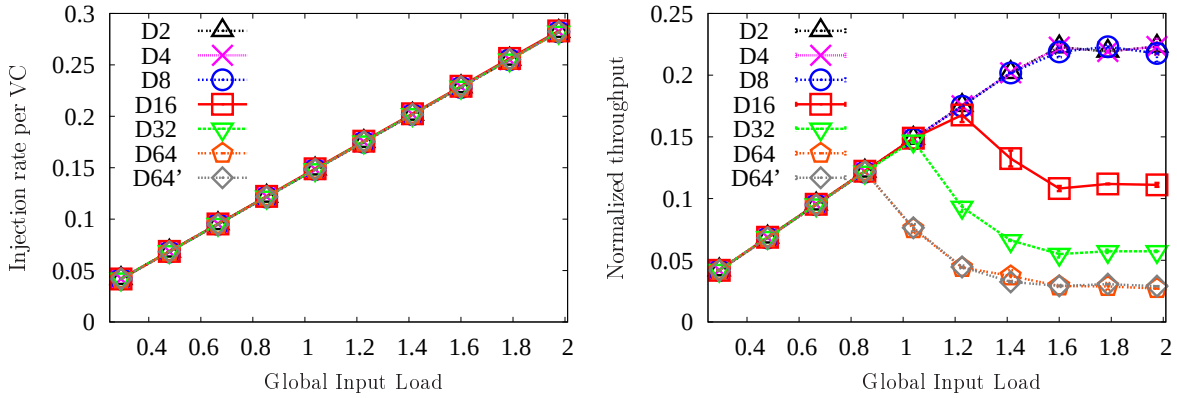


Figure 9.5: Normalized injection rate and throughput per VC of the DTable1 scenario.

Figure 9.6 shows the average latency performance. When the load is very low, all the VCs present a similar low latency. This is because at this load level there are few packets being transmitted through the network, and thus, there are few conflicts between them. However, when the load increases, the latency also increases because some packets must wait in the buffers until others have been transmitted. It is at this point when the scheduling algorithm assumes an important role and the VCs obtain a different latency depending on the scheduler configuration. However, when the load of the VC begins to outstrip its throughput, the latency of the scheduler starts to grow very fast. This is because the buffers used for that VC begin to be full. Finally, the buffers become completely full and the latency stabilizes at a given value which depends on the buffers' size and the bandwidth assigned to that VC.

Note that when using the SCFQ-CA algorithm those VCs that have assigned the same bandwidth (in this case the D2, D4, and D8 VCs, and the D64 and D64' VCs) obtain the same latency performance. In the case of the DRR-CA algorithm, all the VCs obtain a similar latency performance until a VC reaches the point when its load begins to outstrip

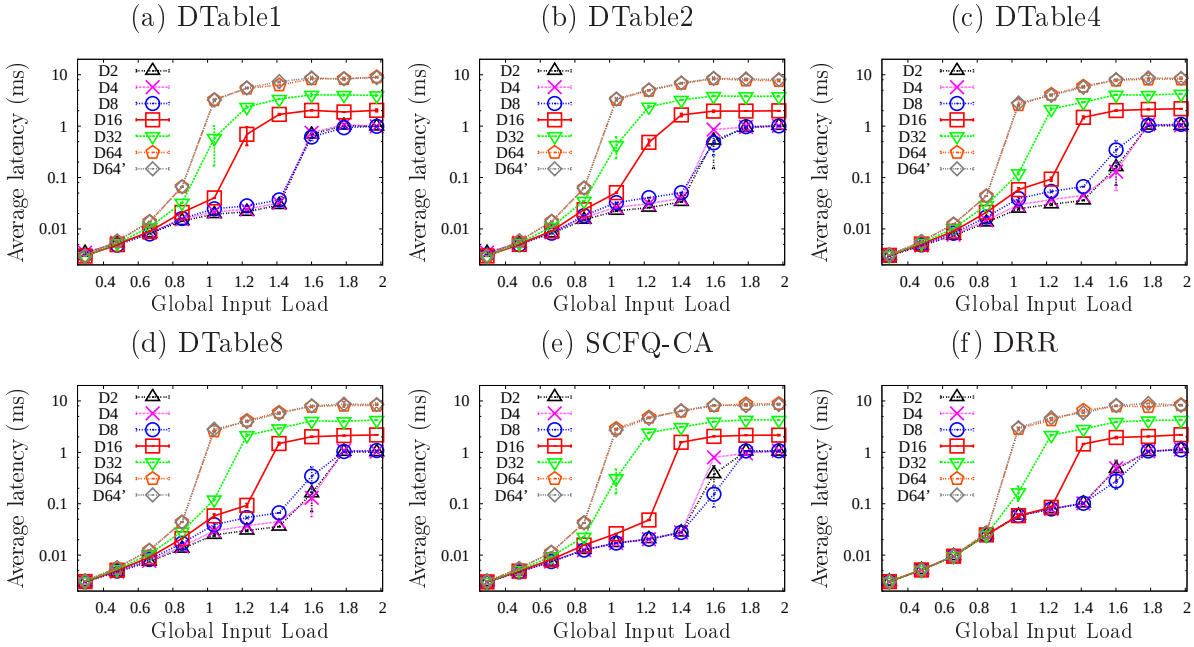


Figure 9.6: Average latency per VC of the different scheduling scenarios.

its throughput. In that point, the latency of that VC grows very fast and obtains a different latency performance. This happens for all the VCs as load grows. When using the DTable scheduler, all the VCs, including those with the same bandwidth assignment, obtain a different latency performance depending on the separation between any consecutive pair of their table entries. The smaller the distance, the better latency performance they obtain.

These different latency performance behaviors are explained by the fact that the maximum time that a packet at the head of a VC queue is going to wait until being transmitted is different depending on the scheduler algorithm. In the case of the SCFQ-CA algorithm, this time is proportional to the assigned bandwidth. In the case of the DTable scheduler, we can control this time by controlling the maximum separation between any consecutive pair of entries assigned to the same VC. In this way, we provide some VCs with a better latency performance and other VCs with a worse latency performance. In the case of the DRR-CA algorithm, the latency performance depends more on the frame length than on the quantum that each VC has been assigned. This is because when the quantum for a VC has been expended sending packets, all the frame must be cycled through before sending more packets of the same VC.

Finally, Figure 9.7 shows the percentage of improvement on average latency of the SCFQ-CA algorithm over the four possibilities of the DTable scheduler and the DRR-CA algorithm. Observing this figure we can analyze the DTable performance comparing it not only with the SCFQ-CA scheduler, but also with the DRR-CA scheduler. Moreover, we

can compare the difference between using the same general MTU for all the VCs or using specific MTUs for the VCs. This figure shows that, in general, the SCFQ-CA algorithm provides a better latency performance than the DTable scheduler in all the cases. However, this algorithm is the most complex. The DRR-CA provides a worse performance than the DTable scheduler for the most latency restrictive VCs and better for the less latency restrictive VCs. This is because with the DTable scheduler we can provide a different level of latency performance to the VCs, priorizing those VCs with higher latency requirements. This is not possible with the DRR-CA algorithm. Regarding the different scenarios of the DTable scheduler we can see that DTable1 provides a better latency performance than DTable2, and DTable4 than DTable8. This is because in general, the higher the value of the w parameter, the worse the latency performance. However, the effect of splitting the messages in several packets must also be taken into account.

Table 9.4 shows the bandwidth overhead per VC that is produced by using specific MTUs smaller than the general MTU. This packetization also has effect on the latency of the message. Note that each packet must be processed by the network elements (routing, scheduling, etc.). Moreover, if a table entry allows us to transmit a small number of packets of the new MTU size, it is possible that in order to transmit all the packets belonging to the same message more than one table entry must be used, and thus, the latency increases. Figure 9.7 shows clearly the first effect when considering a low load for the D2 and D4 VCs. In this case, the latency of the DTable1 and DTable2 scenarios is rather worse than for the others cases. We obtain a better latency for DTable1 and DTable2 than DTable4 and DTable8 when the latency is high for the D2, D4, and D8 VCs. However, for the rest of VCs we obtain a worse latency because the specific MTUs are higher and the weight assigned to the table entries lower. Note that this bad effect of the excessive packetization would disappear in a real case if the MTU of each VC is selected on the basis of the specific average message size of the flows that the VC would use.

Table 9.4: Packetization bandwidth overhead per VC with average packet size of 176 bytes.

VC	**D2**	**D4**	**D8**	**D16**	**D32**	**D64**	**D64'**
MTU_i (bytes)	64	64	128	256	512	1024	2048
Overhead (%)	11.7	11.7	3.82	1	0.4	0.06	0

Summing up, the DTable scheduler provides a worse latency performance than the SCFQ-CA algorithm. The percentage of difference depends on the DTable scenario. Moreover, it provides the most preferential VCs (those which have been assigned a shorter distance between any consecutive pair of entries) with a better latency performance than the DRR-CA algorithm. However, it provides the least preferential VCs with a worse latency than the DRR-CA algorithm. This means that the DTable scheduler is able to

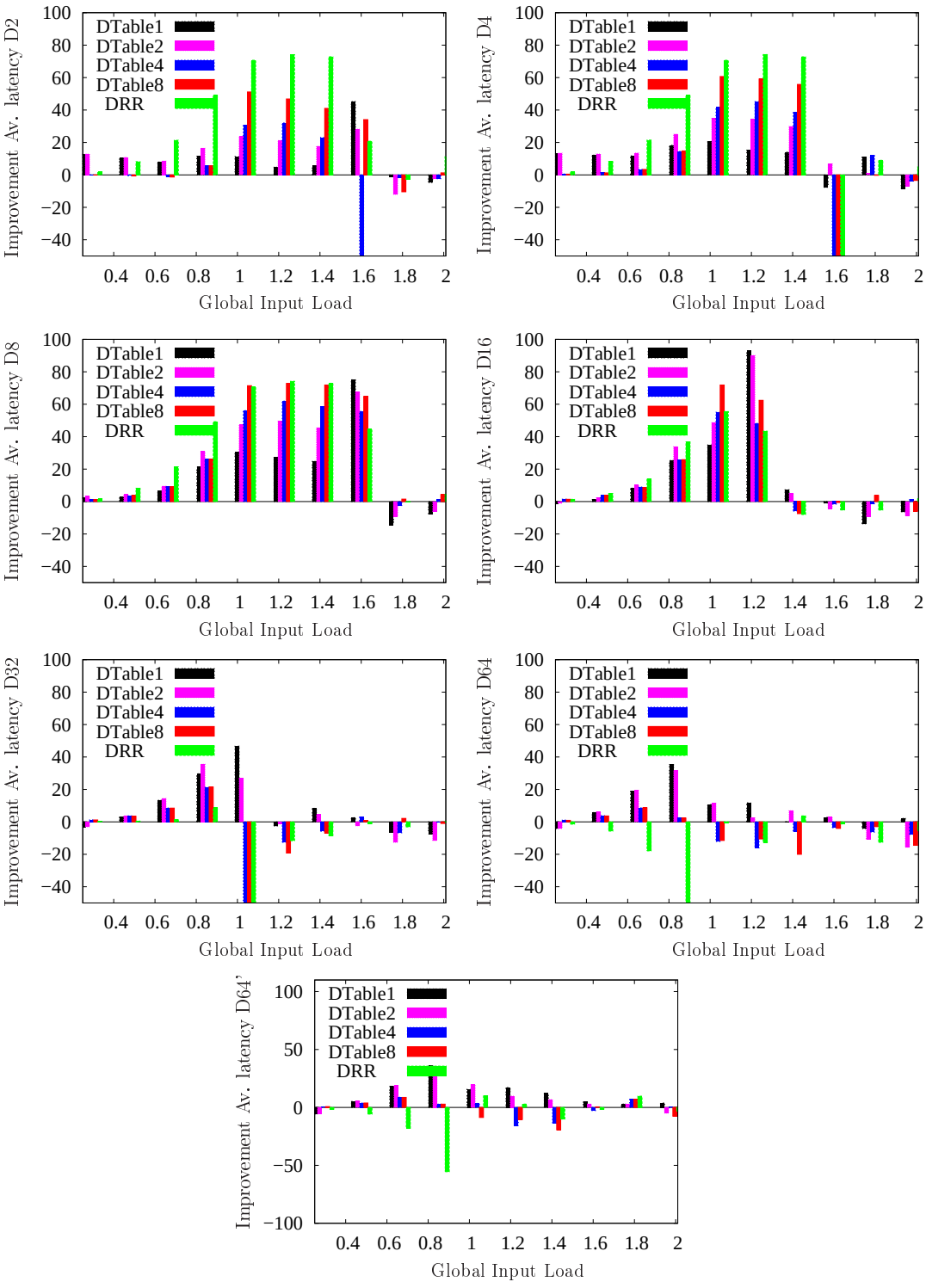


Figure 9.7: Average latency improvement of the SCFQ-CA algorithm over the other schedulers considered.

provide a better latency performance to those flows that really need it. Moreover, we have shown the negative effect on the latency of increasing the value of the w parameter.

9.4 Performance evaluation in a multimedia scenario

In this section we will show an evaluation of our proposals in a multimedia scenario. Our intention is to show that with an AC mechanism for controlling the QoS traffic and a relatively small amount of control traffic (as is usually the case), the QoS requirements of the different SCs are met whatever the load of best-effort traffic.

9.4.1 Traffic model

The IEEE standard 802.1D-2004 [IEE04] defines 7 traffic types at the Annex G, which are appropriate for this study. Table 3.1 in page 28 shows each traffic type and its requirements. We can classify each of these traffic types in one of the three broad categories of traffic that we have presented in Section 8.1 in page 133. We consider the VO, VI, and CL traffic types as QoS traffic, the EE, BE, and BK traffic types as best-effort traffic, and, of course, the NC traffic as network control traffic. We will consider each of these traffic types as a Service Class (SC). In this way, the workload will be composed of 7 SCs and each one of them will be assigned to a different VC, the NC SC being assigned to the FMC.

The packets from each traffic type are simulated according to different distributions, as can be seen in Table 9.5. VO, VI, and CL SCs are composed of point-to-point connections of the given bandwidth. VO and CL SCs are generated following a Constant Bit Rate (CBR) distribution. In [TMdM00] several payload values for voice codec algorithms are shown. These values range from 20 bytes to 160 bytes. We have selected a payload of 160 bytes for the VO SC traffic. In the case of VI SC, MPEG-4 traces are used to generate the size of each frame. Each frame is injected into the network interfaces every 40 ms. If the frame size is bigger than the MTU, the frame is split into several packets which are injected all along the frame time. The traffic of the best-effort SCs is generated according to a Bursts60 distribution [CK04]. This traffic is composed of bursts of 60 packets heading to the same destination. The packet size is governed by a Pareto distribution, as recommended in [Jai91]. In this way, many small size packets are generated, with an occasional large size packet. The periods between bursts are modeled with a Poisson distribution. The Bursts60 pattern models worst-case real traffic scenarios. The NC SC is generated in the

Table 9.5: Traffic pattern of the multimedia SCs considered.

Type	**SC**	**Traffic pattern**	**Packet size**
Control	Network control (NC)	Bursts1	up to 256B
QoS	Voice (VO)	64 Kb/s CBR connections	168B
QoS	Video (VI)	3.42 Mb/s MPEG-4 traces	up to 2176B
QoS	Controlled load (CL)	750 kb/s CBR connections	2176B
Best-effort	Excellent-effort (EE)	Bursts60	up to 2176B
Best-effort	Best-effort (BE)	Bursts60	up to 2176B
Best-effort	Background (BK)	Bursts60	up to 2176B

same way than the Burst60 traffic but with only one packet burst. For all the cases, the destination pattern is uniform in order to fully load the network.

Note that the traffic model that we use in this performance evaluation is based on a multimedia environment. However, we use a wide range of traffic behaviors, and thus the results obtained with this kind of traffic can be generalized to other environments with other kind of traffic with QoS requirements.

9.4.2 Simulated scenario and scheduler configuration

We suppose a scenario in which the goal is to dedicate around 5% of the egress link bandwidth to voice traffic (a lot but low-bandwidth requiring connections), around 40% of bandwidth to video traffic (a lot and high-bandwidth requiring connections), around 20-25% of bandwidth to controlled load, and the remaining bandwidth to best-effort traffic. Moreover, we expect that the maximum network control bandwidth to be around 1%. These percentages are intended to represent a multimedia scenario with a realistic combination of traffic from applications with very different requirements.

Note that depending on the burstiness of the pattern traffic, we may need to reserve more bandwidth than the average rate that we want to assign to a SC. This is true in this scenario for the video traffic. If we only assign the VI VC[1] 40% of bandwidth we would not be able to establish video connections with a total average injection rate of the 40% of the bandwidth. If we would do that, this SC would not probably meet its bandwidth and latency requirements due to the congestion. Therefore, we will assign the VI VC around 50% of the link bandwidth.

[1]We are going to refer each VC with the name of the SC that accommodates. Moreover, we will refer to the requirements of the SCs as the requirements of their VCs.

Table 9.6: Application of the decoupling methodology. $N = 128$, $GMTU = 34$, $w = 2$, $k = 0.5$.

VC	**Distance**	**#entries**	**%entries**	MTU_i **(Bytes)**	$min\phi_i$	$max\phi_i$
NC	2	64	50	3	0.088	2
VO	4	32	25	3	0.044	1
VI	8	16	12.5	34	0.25	0.5
CL	16	8	6.25	34	0.125	0.25
EE	32	5	3.906	34	0.078	0.156
BE	64	2	1.562	34	0.031	0.062
BK	128	1	0.781	34	0.015	0.031
	Total	128	100		0.631	4

The table scheduler must be properly configured to provide the SCs with their bandwidth and latency requirements. A table of 128 entries has been used. Specifically, we have employed the *2 bytes* modification of the AS table scheduler, which is the one that provides a higher flexibility and granularity (see Section 6.3 in page 97). We could have also employed the *3 bit* or the *one weight per VC* options but, it would have been a bit more complex to configure.

Table 9.6 shows the distribution of the table entries among the SCs. It shows the maximum distance between any consecutive pair of entries, the number of table entries, and the percentage of entries that this entails for each SC. We have assigned a distance of 2, 4, and 8 to the NC, VO, and VI VCs respectively, attending to their different latency requirements. Note that this entails assigning 112 entries. We have distributed 8 entries among the best-effort SCs attending to the different priority among them. Finally, we have assigned the remaining 8 entries to the CL SC. For the CL SC and the best-effort SCs we could have assigned the entries sequentially in the free gaps of the table, but to achieve better latency results for these SCs we have assigned their entries minimizing the distance between any pair of consecutive entries. Figure 9.8 shows the final distribution of the VC identifiers among the table entries.

In order to have a higher level of flexibility to distribute the bandwidth among the VCs, we have assigned the NC and VO VCs a specific MTU as small as the expected packet sizes of these SCs allow. Specifically, we have assigned a MTU of 192 bytes to these VCs. Note that these VCs have very high latency requirements but they actually need a small amount of bandwidth.

NC	3	VO	3	NC	3	VI	68	NC	3	VO	3	NC	3	CL	64
NC	3	VO	3	NC	3	VI	68	NC	3	VO	3	NC	3	EE	34
NC	3	VO	3	NC	3	VI	68	NC	3	VO	3	NC	3	CL	64
NC	3	VO	3	NC	3	VI	68	NC	3	VO	4	NC	3	BE	34
NC	3	VO	3	NC	3	VI	68	NC	3	VO	4	NC	3	CL	64
NC	3	VO	3	NC	3	VI	68	NC	3	VO	4	NC	3	EE	34
NC	3	VO	3	NC	3	VI	68	NC	3	VO	4	NC	3	CL	64
NC	3	VO	3	NC	3	VI	68	NC	3	VO	4	NC	3	EE	34
NC	3	VO	3	NC	3	VI	68	NC	3	VO	4	NC	3	CL	64
NC	3	VO	3	NC	3	VI	68	NC	3	VO	4	NC	3	EE	34
NC	3	VO	3	NC	3	VI	68	NC	3	VO	4	NC	3	CL	64
NC	3	VO	3	NC	3	VI	68	NC	3	VO	4	NC	4	BE	34
NC	3	VO	3	NC	3	VI	68	NC	3	VO	4	NC	4	CL	63
NC	3	VO	3	NC	3	VI	68	NC	3	VO	4	NC	4	EE	34
NC	3	VO	3	NC	3	VI	68	NC	3	VO	4	NC	4	CL	63
NC	3	VO	3	NC	3	VI	68	NC	3	VO	4	NC	4	BK	34

Figure 9.8: Table configuration for the basic multimedia scenario.

The next step to configure the DTable scheduler is to choose a proper value for the w and k parameters. We have chosen the value of these parameters taking into account mainly that we want to assign the VI VC a bandwidth several times higher than the actual proportion of table entries assigned. Moreover, we want to assign the NC VC, which has assigned a very high proportion of table entries, a quite small proportion of bandwidth. However, we want to assign a value to the w parameter as small as possible in order to obtain a good latency performance. We have finally chosen a value of 2 for k and a value of 0.5 for w. This combination allows us to assign each VC a bandwidth in the desired range, except for the NC VC to which we must assign at least 8.8% bandwidth. However, note that this amount of bandwidth is not going to be wasted because it is well-known that interconnection networks are unable to achieve 100% global throughput. Moreover, the bandwidth left by the network control traffic is going to be distributed among the rest of VCs, specifically, the best-effort VCs. Table 9.6 shows the minimum and maximum bandwidth that we can assign to each VC with this configuration.

Table 9.7 shows the final bandwidth assignation among the VCs. This table also shows the total weight that we have distributed among the table entries of each VC and the weight assigned to each table entry of each VC. This weight assignation is also shown in Figure 9.8.

Table 9.7: Bandwidth configuration of the DTable scheduler.

VC	ϕ_i	# entries	Entry Weight	Total Weight
NC	0.091	64	59×3, 5×4	197
VO	0.05	32	19×3, 13×4	109
VI	0.5	16	16×68	1088
CL	0.234	8	6×64, 2×63	510
EE	0.078	5	5×34	170
BE	0.031	2	2×34	68
BK	0.016	1	1×34	34
Total	1	128		2176

Regarding the configuration of the MinBW scheduler, in order to be able to compare the different schedulers in a fair way, we are going to perform the same bandwidth assignation as in the DTable case. Specifically, we have assigned each VC a weight equal to the *total weight per VC* that we have in the DTable case. These weights can be directly translated into a proportion in the SCFQ-CA and WFQ-CA schedulers. In the case of the DRR the weight must be translated into quantum units. The minimum weight should be translated into an amount of information equal to the MTU, and the rest of weights should be translated proportionally. However, note that the minimum weight is 34, which actually is the MTU. Therefore, the weights can be directly translated into quantum units expressed in flow control credits.

Finally, as stated in Section 8.3 we are going to consider an admission control mechanism that ensures that the VO, VI, and CL VCs are not oversubscribed. This means that the sum of the average injection rate of the flows that traverse these VCs is smaller than or equal to the bandwidth that these VCs have reserved. In the case of the VI VC we are going to allow a smaller amount of bandwidth than it has reserved because of the high degree of burstiness of the video traffic. We also suppose that the amount of control traffic in the network is going to be under a certain maximum. On the other hand, we do not make any assumption about best-effort traffic.

In this scenario, we are going to inject a fixed amount of control traffic (NC SC) and QoS traffic (VO, VI, and CL SCs) all the time, and we gradually increase the amount of best-effort traffic (EE, BE, and BK SCs). The amount of QoS traffic to be injected is the maximum allowed by the AC mechanism. Table 9.8 and Figure 9.9 show the normalized injection rate of each VC.

Table 9.8: Bandwidth assigned and injection rate per VC.

VC	ϕ_i	Minimum	Maximum
NC	0.091	0.01	0.01
VO	0.05	0.05	0.05
VI	0.5	0.37	0.37
CL	0.234	0.22	0.22
EE	0.078	0.001	0.12
BE	0.031	0.001	0.12
BK	0.016	0.001	0.12
Total	1	0.653	1.01

9.4.3 Simulation results of the basic multimedia scenario

Figure 9.9 gives a general overview of the performance when using the DTable scheduler. It shows the injection rate, throughput, average latency, and the 99th percentile of the CDF of latency. We do not show similar figures for the rest of schedulers because the throughput performance is the same for all the schedulers. Moreover, although the specific latency values are different, the general tendencies for the other mechanisms are the same. And thus, the comments that we are going to make based on this figure can be generalized to the rest of schedulers. If we compare the injection and the throughput results, we can see that the NC and the QoS SCs obtain all the bandwidth they inject. However, when the network load is high (around 85%), the best-effort SCs do not yield a corresponding result. From that input load, these SCs obtain a bandwidth proportional to their priority.

Regarding the latency performance, Figure 9.9 shows that the latency (average and 99th percentile) of the NC and QoS SCs grows with the load until they reach a certain value. Once this value is reached the latency remains more or less constant. However, the average latency of best-effort SCs continuously grows with the load. Furthermore, it can be seen that best-effort SCs obtain different average and maximum latency according to their different priority. In that sense, for example, the BK SC obtains a worse latency and starts to increase its latency sooner than the BE and EE SCs. Note that although the control and QoS SCs employ separate VCs, the growing best-effort traffic slightly affects their performance.

Figures 9.10, 9.11, 9.12, 9.13, and 9.14 show a more detailed comparison of the performance provided by each scheduler to the different SCs. Regarding the control SC, Figure 9.10 shows statistics on average latency, the 99th percentile of the CDF of latency, and the CDF of latency for the point of maximum load. This figure shows that the three

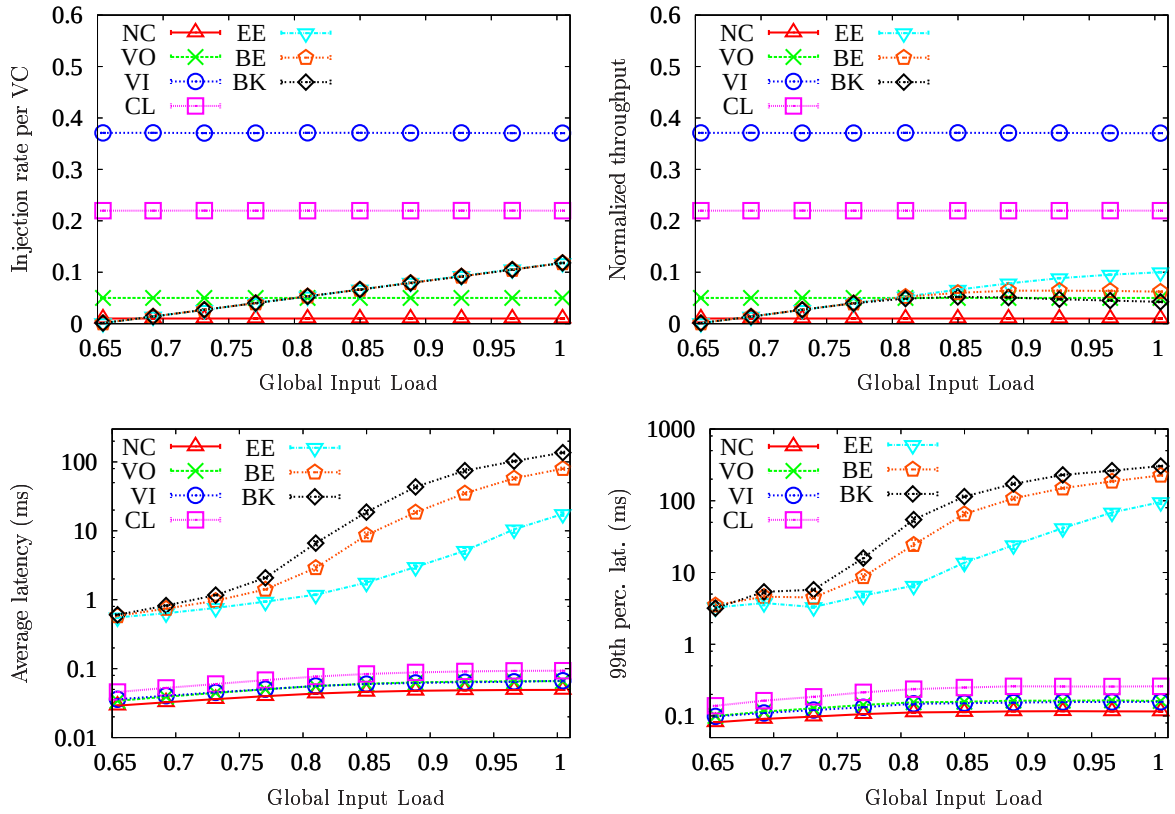


Figure 9.9: Performance per SC of the DTable scheduler.

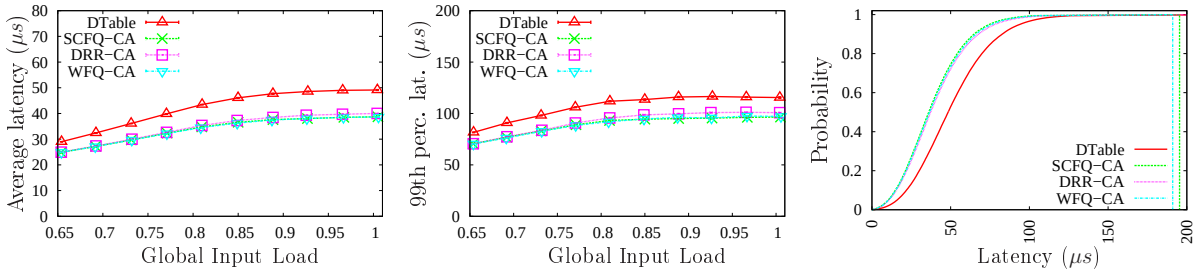


Figure 9.10: Latency performance comparison for the network control SC.

possibilities for the MinBW scheduler provide a similar performance, which is better than the performance provided by the DTable scheduler. This is because, as stated before, the MinBW scheduler employs a strict priority mechanism to schedule the FMC, which is the VC that we have assigned to the control SC. However, in the DTable scheduler case the control traffic does not have strict priority and it must compete with the other traffic.

Regarding the VO and VI SCs, Figures 9.11 and 9.12 show statistics on average latency, the 99th percentile of the CDF of latency, and the CDF of latency for the point

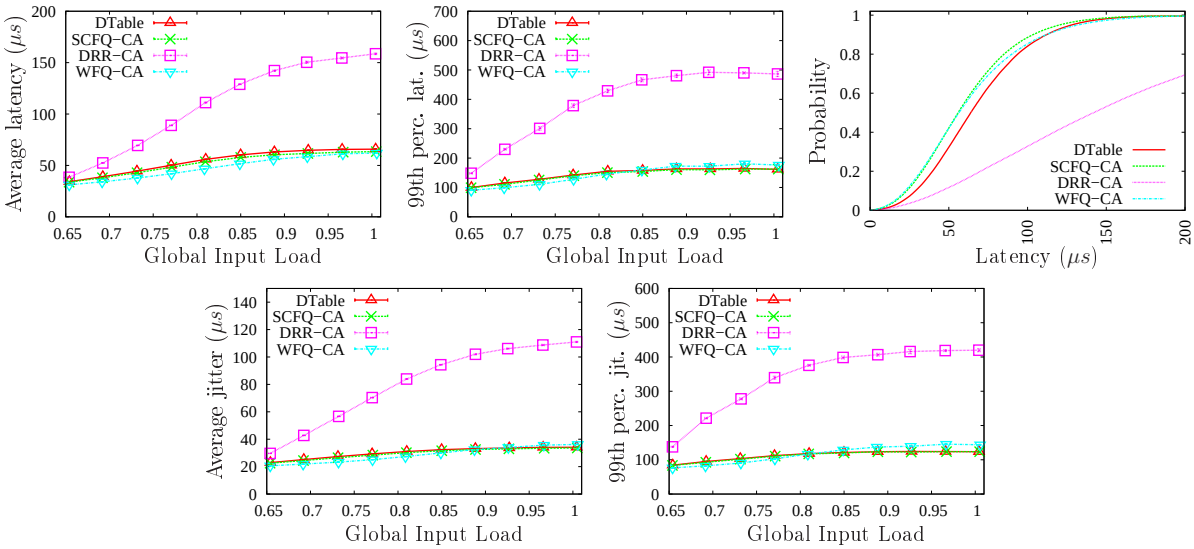


Figure 9.11: Latency performance comparison for the voice SC.

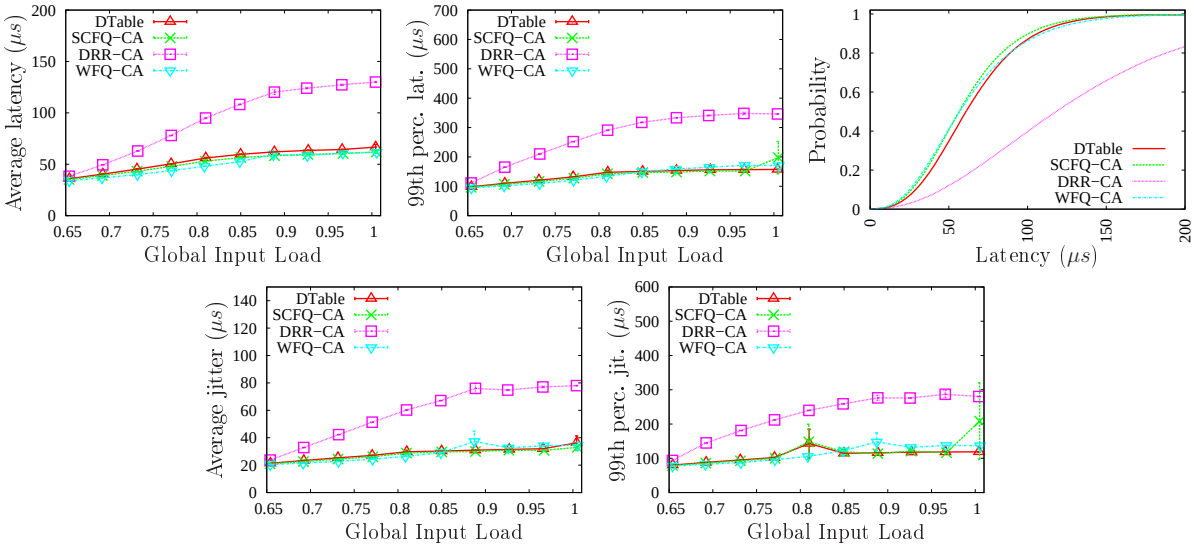


Figure 9.12: Latency performance comparison for the video SC.

of maximum load. They also show the average jitter and the 99th percentile of the CDF of jitter. Figures 9.11 and 9.12 show that the DRR-CA variant of the MinBW provides the worst latency and jitter performance of the four schedulers. If we compare the WFQ-CA and the SCFQ-CA MinBW variants, and the DTable scheduler, we can see that they provide a similar performance. In some cases, we can even see that the different lines and/or confidence intervals are overlapped. However, the CDF shows that the DTable scheduler provides a slightly worse latency performance.

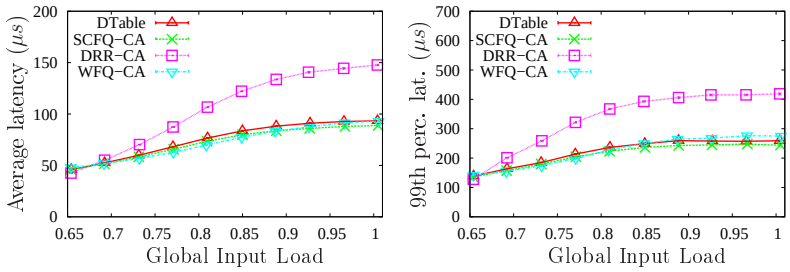


Figure 9.13: Latency performance comparison for the controlled load SC.

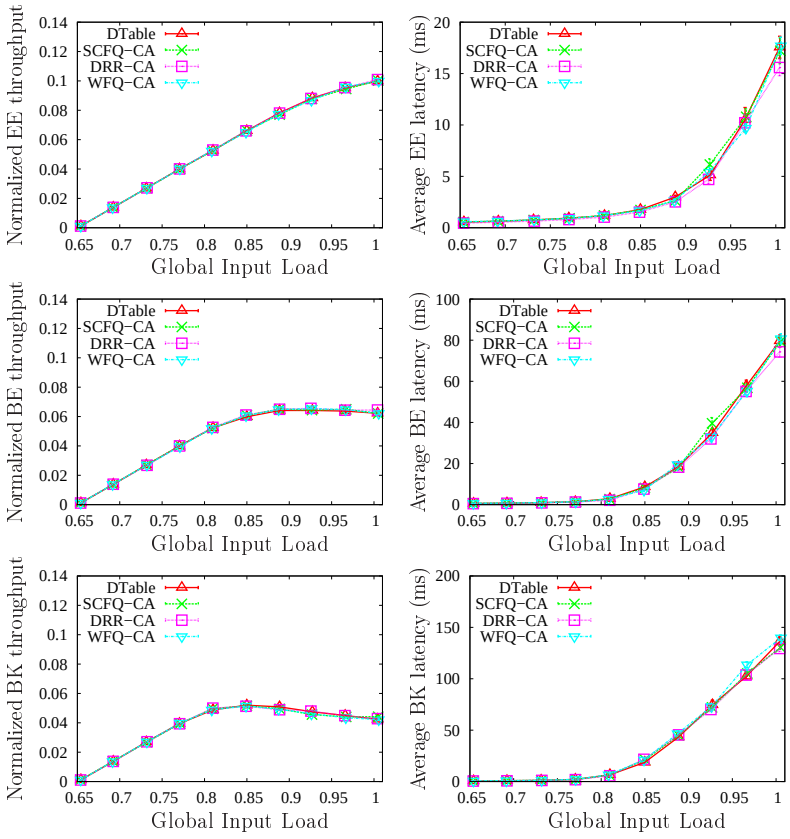


Figure 9.14: Latency performance comparison for the best-effort SCs.

Regarding the CL SC Figure 9.13 shows statistics on average latency and the 99th percentile of the CDF of latency. This figure shows, like in the VO and VI SCs cases, that the DRR scheduler provides the worst latency performance.

Regarding the best-effort SCs Figure 9.14 shows statistics on throughput and average latency. This figure shows more clearly than Figure 9.9 that the best-effort SCs are provided a throughput and latency performance which depend on the priority of each SC. This figure also shows that all the schedulers provide a similar performance to these SCs.

Summing up, simulation results of the basic multimedia scenario have shown that with an AC mechanism for controlling the QoS traffic and a relatively small amount of control traffic, we can control the throughput and latency performance (and thus, the jitter performance) of the QoS traffic whatever the load of best-effort traffic. Moreover, simulation results have shown that the DRR scheduler provides by far the worst performance of the four schedulers considered. Note also that the performance of this scheduler depends on the frame length, and thus, in other scenarios, the performance could be even worse. On the other hand, the SCFQ-CA and WFQ-CA schedulers provide practically the same performance. The DTable scheduler provides, except for the network control traffic, a performance only slightly worse than the SCFQ-CA and WFQ-CA schedulers.

9.4.4 Effect of the video injection rate

In the basic multimedia scenario, which we have evaluated in the previous section, we have allowed less video connections in the network than the percentage of reserved bandwidth would have permitted if we would have only considered the average injection rate of each video connection. Specifically, the VI VC has been assigned 50% of the link bandwidth but, we have injected only 37% of video traffic into the network. This configuration emulates that the bandwidth broker assigns an effective bandwidth to each connection depending on its traffic pattern characteristics, mainly the average and peak injection rate. The effective bandwidth is the parameter that is actually taken into account to reserve bandwidth in each link. As stated in Section 3.4.6 in page 34, there are several proposals in the literature to calculate the effective bandwidth. It is out of the scope of this thesis to decide which of these proposals is the most appropriate. However, in this section, we would like to further study the effect of the calculation of the effective bandwidth in an intuitive way. Specifically, we are going to show the effect of varying the amount of video traffic that is permitted into the network.

Moreover, we are going to show the effect of the video trace selected to generate the video traffic. In the basic multimedia scenario we have employed a trace of the *Paris* video sequence to generate all the video traffic, each video connection starting at an aleatory frame. In this section we are going to compare the performance employing this video trace with other three typical video traces. The characteristics of the sequences that we have employed are shown in Table 9.9. We have chosen these sequences because they are popular in the evaluation of video performance [Vid].

As stated before, we are going to show the effect over the performance of varying the video injection rate. In order to keep the global injection rate constant we are going to vary

Table 9.9: Video sequences for performance evaluation.

Name	**CIF/QCIF**	**Bandwidth**
Highway	QCIF	0.492 Mb/s
Paris	CIF	3.42 Mb/s
Mobile	CIF	9.71 Mb/s
Funny	CCIF	13.00 Mb/s

also the injection rate of the best-effort traffic. Table 9.10 shows the amount of bandwidth from each SC that we inject in each simulation point of this performance evaluation.

Table 9.10: Bandwidth assigned and injection rate per VC of each simulation point.

		Injection rate								
VC	ϕ_i	**0.25**	**0.28**	**0.31**	**0.34**	**0.37**	**0.40**	**0.43**	**0.46**	**0.49**
NC	0.091	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
VO	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05
VI	0.5	*0.25*	*0.28*	*0.31*	*0.34*	*0.37*	*0.40*	*0.43*	*0.46*	*0.49*
CL	0.234	0.22	0.22	0.22	0.22	0.22	0.22	0.22	0.22	0.22
EE	0.078	0.16	0.15	0.14	0.13	0.12	0.11	0.10	0.09	0.08
BE	0.031	0.16	0.15	0.14	0.13	0.12	0.11	0.10	0.09	0.08
BK	0.016	0.16	0.15	0.14	0.13	0.12	0.11	0.10	0.09	0.08
Total	1	1.01	1.01	1.01	1.01	1.01	1.01	1.01	1.01	1.01

Simulation results have shown that there is no significant difference among the performance of the various schedulers. Therefore, we are going to show results only for the SCFQ-CA scheduler. Figure 9.15 shows the injection rate, throughput, average latency and the 99th percentile of the CDF of latency provided by the SCFQ-CA scheduler when the *Paris* sequence is employed. Note that this is the sequence employed in the basic multimedia scenario. The performance obtained when employing the other sequences follows the same trend. This figure shows that when the VI SC injection rate is very high the VI SC throughput is lower than the injection rate. This is because when the video injection rate is too high, even if is lower than the bandwidth assigned to the VI SC, the congestion produced by the bursts of video traffic makes the network unable to provide all the bandwidth that this SC has been assigned.

Figure 9.15 also shows that before this video load point in which congestion makes the network unable to provide all the required throughput, the latency of the video packets has already grown a lot. Therefore, generalizing to any traffic type, this study shows the

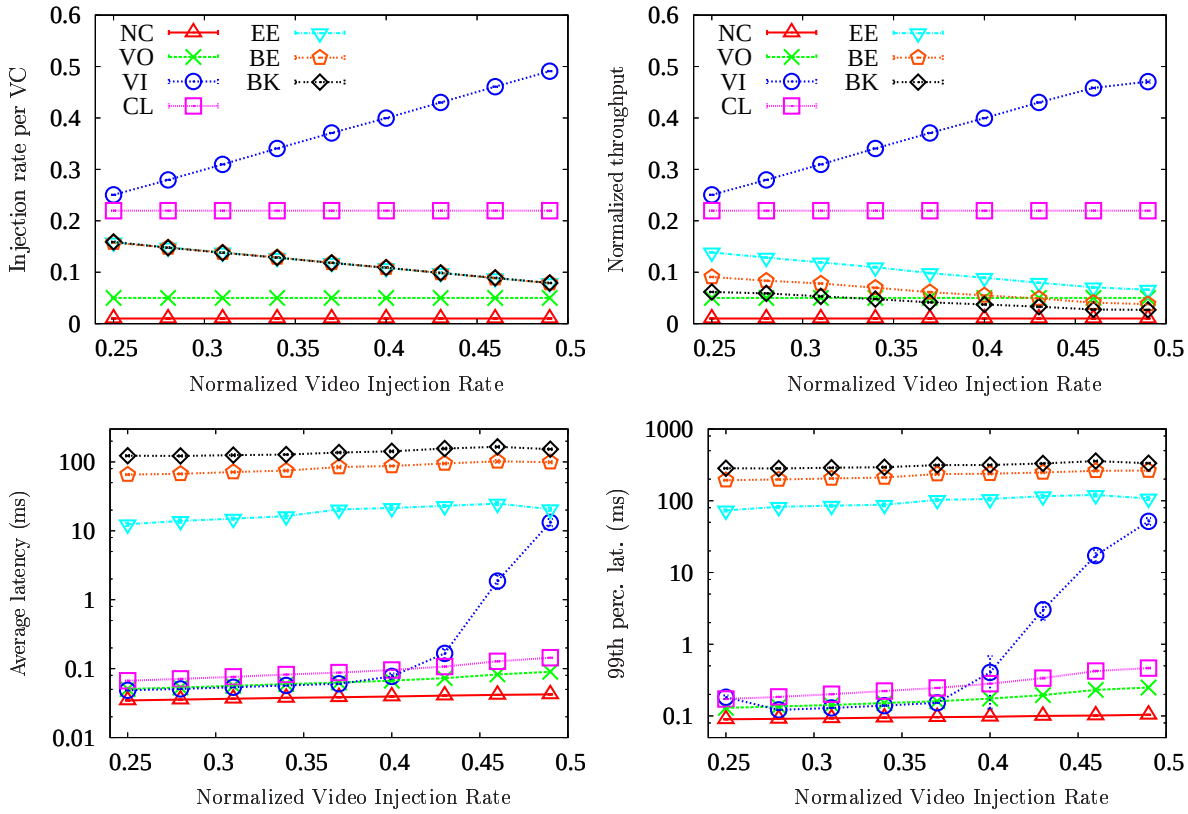


Figure 9.15: Performance per SC of the SCFQ scheduler when varying the VI SC injection rate and employing the *Paris* sequence.

necessity of the bandwidth broker to carefully limit the traffic injection. Note that in the basic multimedia scenario we injected 0.37% of video traffic, which is just the injection rate value before the video latency starts to grow a lot. However, taking into account the latency video requirements, we could have chosen a higher value.

Figure 9.15 also shows that the NC SC and the rest of QoS SCs are slightly affected by the amount of video traffic injected. This is because the amount of bandwidth left over by the VI SC traffic is distributed among the rest of VCs. When the video injection rate decreases, the amount of bandwidth that is distributed among the rest of VCs is bigger and thus, the performance of the rest of VCs is better.

Figure 9.16 shows a comparison of the performance obtained by the VI SC depending on the video sequence employed. Specifically, this figure shows statistics on throughput, average latency, the 99th percentile of the CDF of latency, average jitter, and the 99th percentile of the CDF of jitter. Note that the throughput performance is compared with the video injection rate. This figure shows that the performance of the VI SC, at the same

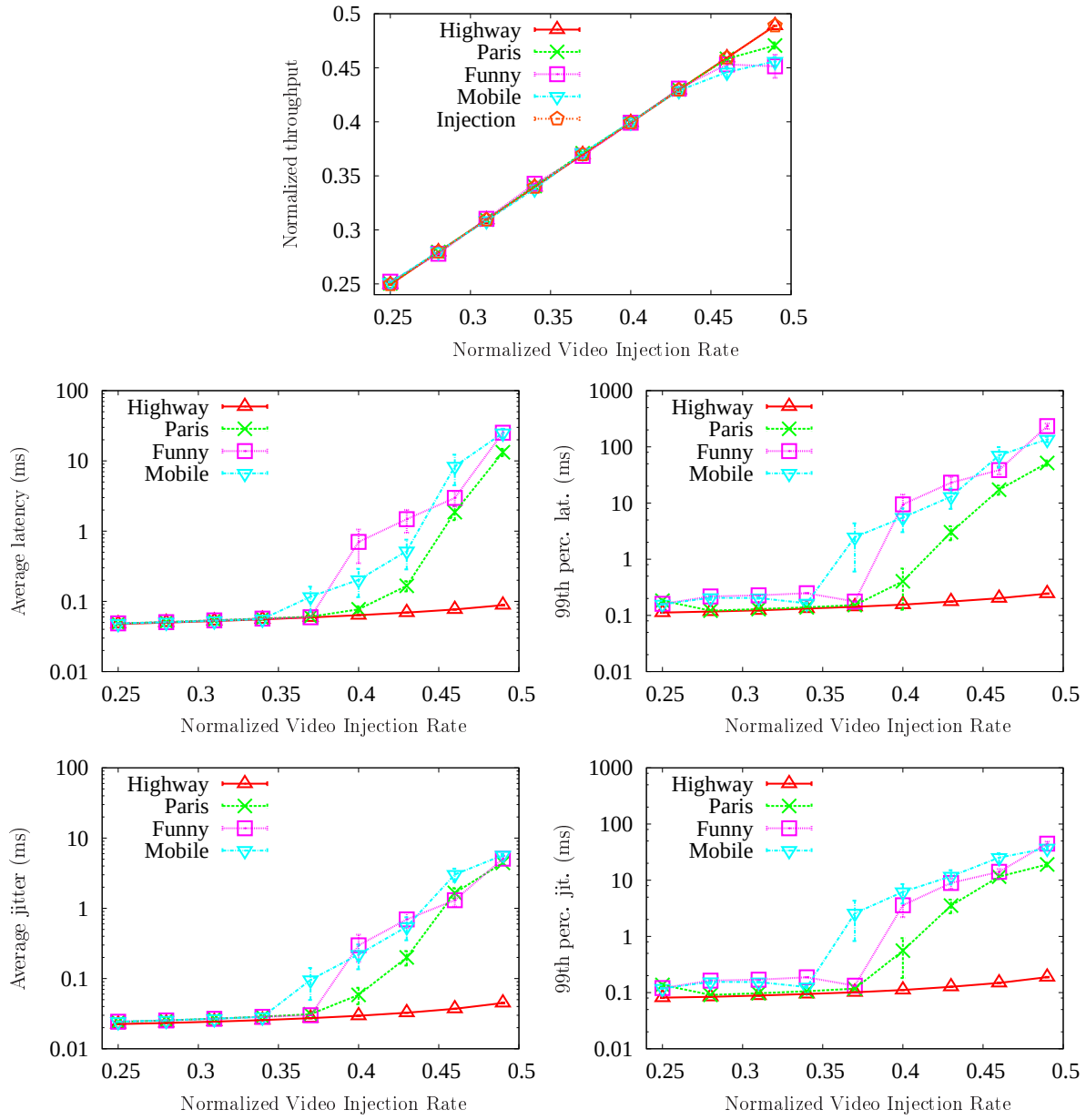


Figure 9.16: Latency performance comparison for the video SC when varying the VI SC injection rate.

injection rate, depends in a high degree on the video sequence. For example, the network is able to handle much more easily the *Highway* sequence than the Paris, Funny, and Mobile sequences, which are more bandwidth demanding sequences with bigger traffic bursts.

Summing up, in this section we have shown in an intuitive way that the bandwidth broker must carefully calculate the effective bandwidth of each connection in order to control in a proper way the throughput, latency, and jitter provided to the SCs.

9.4.5 Effect of the buffer size

In the basic multimedia scenario, which we have evaluated in Section 9.4.3 we have employed a buffer size in the switches equal to 8 times the MTU. In this section, we are going to show the effect of using other buffer sizes. Specifically, we are going to test the following buffer sizes: 1, 2, 4, 6, 8, 10, and 12 times the MTU. For all the cases we are going to inject the same traffic than in the maximum load point of the basic multimedia scenario (see Figure 9.17).

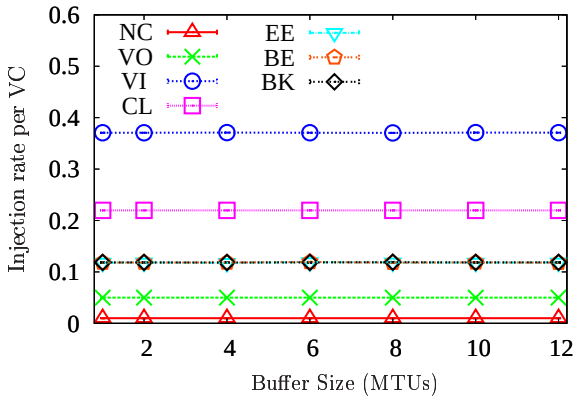


Figure 9.17: Injection rate per SC when varying the buffer size.

Figure 9.18 shows a general overview of the performance when using the DTable, DRR-CA, and SCFQ-CA schedulers. We do not show statistics for the WFQ-CA scheduler because they are practically the same than in the SCFQ-CA case. Specifically, Figure 9.18 shows statistics on throughput, average latency and the 99th percentile of the CDF of latency. This figure shows that for very small buffer sizes the performance provided by the schedulers is quite different than when the buffer size is bigger. This difference is bigger for the DRR-CA scheduler. Specifically, the VI and CL SCs are affected in a negative way, both in throughput and latency performance. On the other hand, the best-effort SCs take advantage of the bad performance of the VI and CL SCs and are affected positively.

Figures 9.19 and 9.20 show a more detailed comparison of the performance provided by the different schedulers to the VI and CL SCs. Figure 9.19 shows that, when the buffer size is only 1 time the MTU, the schedulers cannot provide the VI SC with all the bandwidth

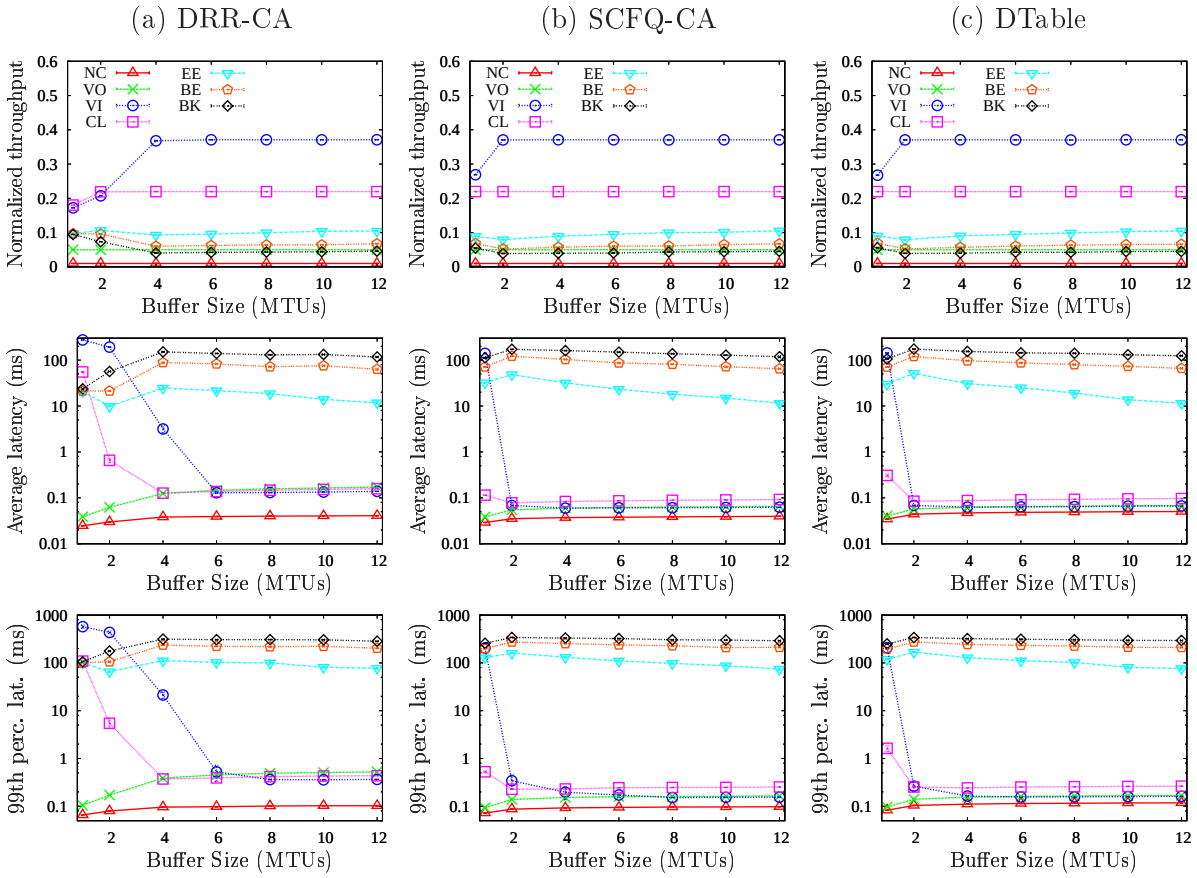


Figure 9.18: Performance per SC of the DRR, SCFQ, and DTable schedulers when varying the buffer size.

that it requires. Note that this also makes the latency to grow a lot. This is because the video traffic is composed of bursts of large packets and thus, with so a small buffer, the link level flow control mechanism is going to block and unblock the VI VC a lot of times, making impossible to take advantage of all its assigned bandwidth. This is going also to affect not the throughput but the latency of the CL SC, as it is shown in Figure 9.20.

Figures 9.19 and 9.20 also show that, in the DRR case, the throughput obtained by the VI and CL SCs is not equal to their injection rate until bigger buffer sizes than in the rest of cases are employed. In the same way, the latency obtained by these SCs is very high until quite big buffer sizes are employed. The reason of this is that in the DRR-CA scheduler case VCs are cycled through and a certain amount of packets are transmitted from each VC depending on the assigned proportion of link bandwidth. If the difference among the bandwidth assigned to each VC is big, the amount of information that should be transmitted from the VCs that have been assigned a high bandwidth is also quite big. If the buffers are too small, there are not going to be enough packets stored in the switch

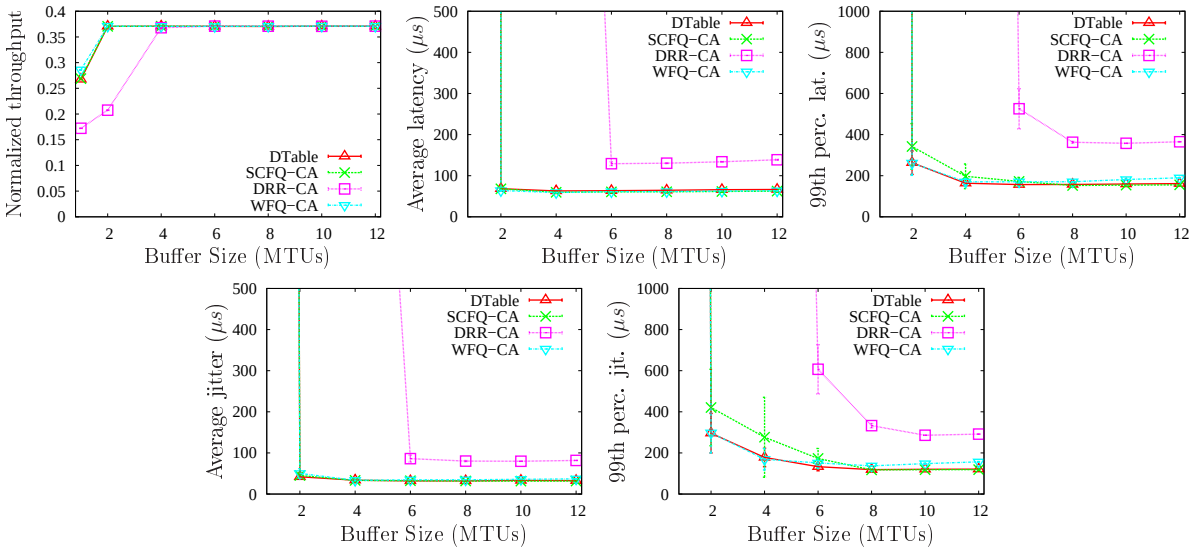


Figure 9.19: Latency performance comparison for the video SC when varying the buffer size.

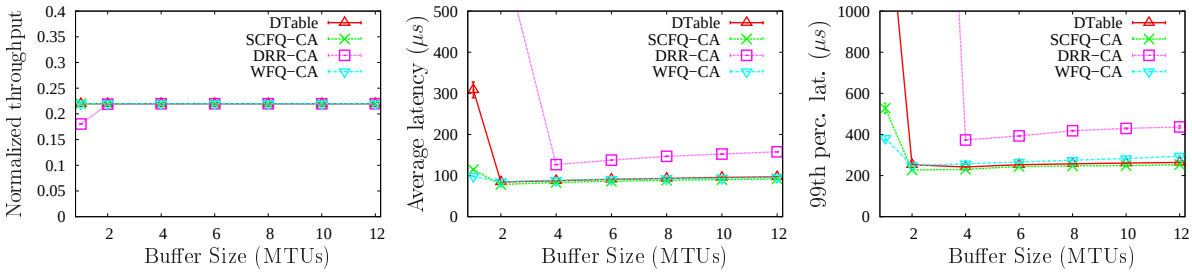


Figure 9.20: Latency performance comparison for the controlled load SC when varying the buffer size.

to satisfy with the required amount of information to transmit and thus, the next active VC is going to be selected after transmitting the last available packet. This is going to produce that the affected SC is not going to be able to take advantage of all the bandwidth that it has actually been assigned. In the current scenario, the VI and CL SCs, which are the SCs that have been assigned most of the link bandwidth, are not able to use all their assigned bandwidth and thus, their throughput and latency are affected negatively.

Summing up, in this section we have shown that all the schedulers, except the DRR, work in a proper way when the size of the switch buffers is at least twice the MTU. On the other hand, depending on the bandwidth assignation, the DRR may require very big buffer sizes to provide a proper throughput and latency performance.

9.4.6 Effect of the MTU size

In the basic multimedia scenario, which we have evaluated in Section 9.4.3 we have employed a MTU of 34, which is the maximum possible value in AS. In this section we are going to study the effect of using other MTU values. Specifically, we are going to test MTUs equal to 3, 5, 9, 17, and 34 flow control credits, which are the possible MTU values for both the bypassable and ordered unicast VCs in AS.

Having a smaller general MTU value would allow us to assign less bandwidth to the best-effort SCs in the DTable scheduler case. However, in order to make a fair comparison, we are going to keep the same bandwidth distribution among the VCs for all the MTU cases than in the basic multimedia scenario. Note that, even with smaller MTU values, the configuration of the SCFQ-CA, WFQ-CA, and DTable schedulers remains the same. In the case of the SCFQ-CA and WFQ-CA schedulers, this is clear because their configuration do not depend on the MTU value. However, in the case of the DTable scheduler, the MTU value could actually have allow to change the weight assigned to the table entries. Nevertheless, in this scenario, in which we have assigned a specific MTU value of 3 flow control credits to the NC and VO SCs, the weights must remain the same because we already employ such minimum weight to configure the weights assigned to their entries. The entry weights assigned to the rest of SCs must remain the same in order to keep the bandwidth proportion.

On the other hand, the configuration of the DRR-CA scheduler is actually affected by the change in the MTU value. In order to configure this scheduler, we must assign at least a quantum equal to the MTU to the VC with the least bandwidth assignation and a proportional quantum to the rest of VCs. Therefore, using a smaller MTU value entails a smaller frame length.

Regarding the simulated traffic patterns, varying the MTU only affects the generation of the CL SC traffic. The traffic of this SC is emulated employing CBR traffic and thus, in order to keep it as CBR traffic, packets from each CL connection are generated of the MTU size. The rest of SCs generate packets of the same size than in the basic multimedia scenario and are split, if necessary, in smaller packets depending on the MTU employed.

Figure 9.21 shows the general effect of varying the MTU value from 3 to 34 when the SCFQ is employed. This figure shows that the bigger the MTU value is, the worse the latency performance of the control and QoS SCs is. This trend is followed by the rest of schedulers. This is because when the MTU is smaller the best-effort packets are in general smaller and they interfere less with the rest of SCs.

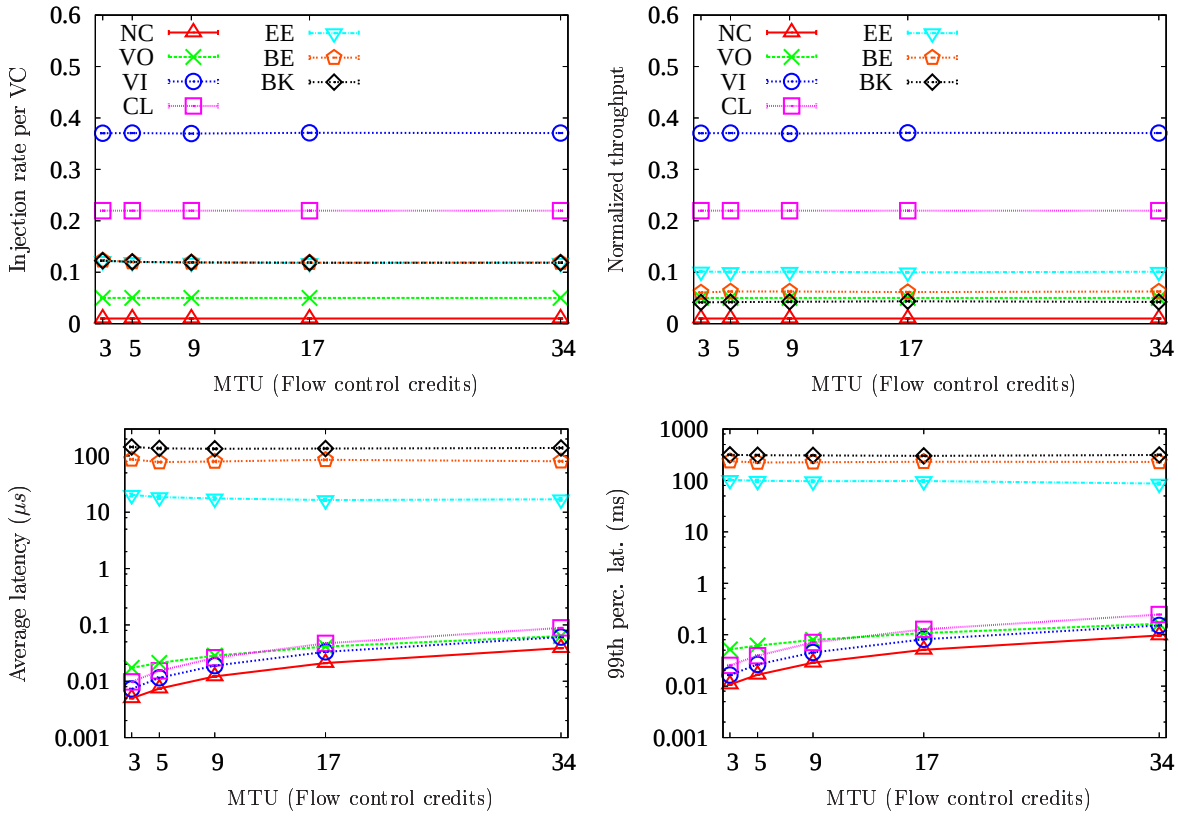


Figure 9.21: Performance per SC of the SCFQ-CA scheduler when varying the MTU value.

Figure 9.22 shows a more detailed comparison of the performance provided by the different schedulers. Specifically, it shows the average latency obtained by the control and QoS SCs. The most important feature shown by this figure is that, the smaller the MTU value is, the closer the performance that the DRR scheduler provides is to the provided by the rest of schedulers. This is because, as stated before, the smaller the MTU is, the smaller the DRR frame length, and thus, the better performance it obtains. Moreover, in this simulation scenario, we have kept the buffer size invariable and thus, the relative buffer size increases in relation with the MTU. This, as it has been shown in the previous section, improves the DRR latency performance.

Note, however, that employing small MTU values does not totally solve the problem of the DRR scheduler to provide QoS based on bandwidth and latency requirements. Employing small MTU values only alleviates the problem because depending on the bandwidth distribution configuration and the buffer size, the DRR scheduler is not going to be able to provide the desired performance.

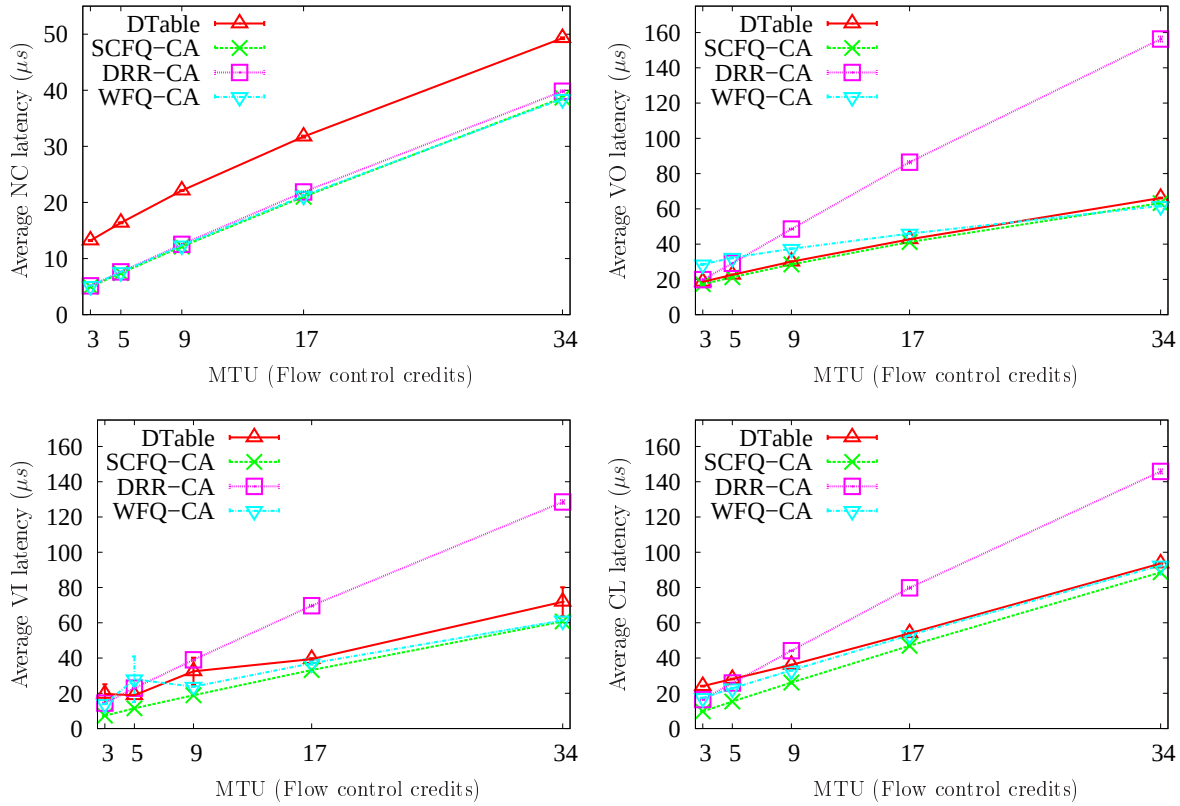


Figure 9.22: Latency performance comparison for the NC, VO, VI, and CL SCs when varying the MTU value.

9.4.7 Effect of the network size

In the basic multimedia scenario, which we have evaluated in Section 9.4.3 we have employed a perfect-shuffle Bidirectional Multi-stage Interconnection Network (BMIN) with 64 endpoints connected using 48 8-port switches (3 stages of 16 switches). In this section we are going to study the effect of using other network sizes. Specifically, we are going to simulate networks with 16, 32, 64, 128, 256, and 512 endpoints that are connected following a shuffle pattern. Table 9.11 shows the number of 8-port switches and stages that each network size involves.

Simulation results have shown that there is no significant difference among the performance of the various schedulers. Therefore, we are going to show results only for the SCFQ-CA scheduler. Figure 9.23 shows the injection rate, throughput, average latency and the 99th percentile of the CDF of latency provided by the SCFQ-CA scheduler for different network sizes. This figure shows that there is only a slight increase in the latency performance due to the increment in the network size. Therefore, we can say that, at least

Table 9.11: Characteristics of the networks considered.

Endpoints	Switches	Stages
16	8	2
32	24	3
64	48	3
128	128	4
256	256	4
512	640	5

with a shuffle BMIN topology, the network size is not a determinant parameter in the performance provided by the schedulers.

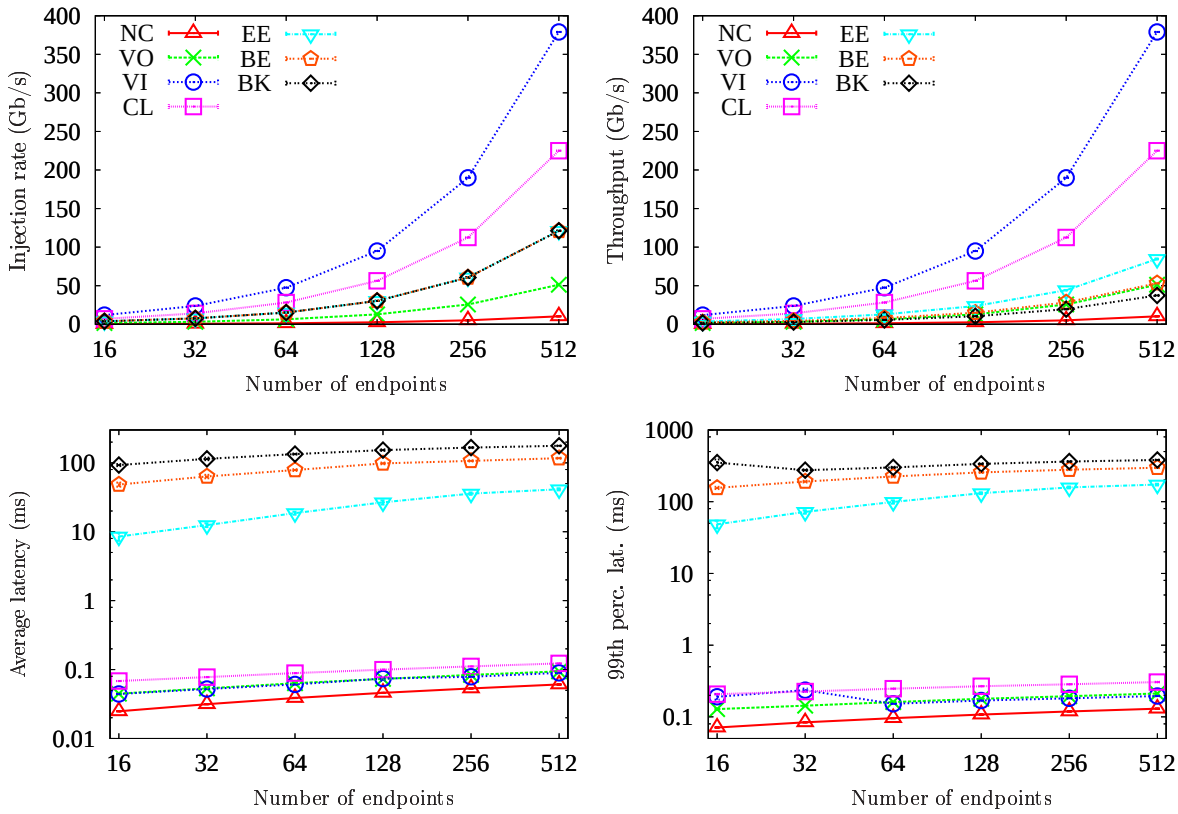


Figure 9.23: Performance per SC of the SCFQ scheduler when varying the network size

9.5 Conclusions

In this chapter we have evaluated the performance of our proposals in two different scenarios. The first scenario was intended to show the capacity of the various schedulers to provide latency differentiation. Moreover, this first scenario has been used to show the effect of the w parameter of our decoupling methodology on the latency performance provided by the DTable scheduler.

Simulation results have shown that the higher the value of the w parameter, the worse latency performance the DTable scheduler provides. We have shown that using different specific MTUs increments the flexibility of our decoupling methodology without the need of increasing the parameter too much. However, the excessive packetization of the messages may produce a negative effect on the performance of the flows. Therefore, the specific MTUs should be assigned taking into account the characteristics of the traffic, specifically, the size of the packets.

Regarding the capacity to provide latency differentiation, simulation results have shown that the DRR-CA scheduler is not able to provide any latency differentiation. It provides all the SCs with the same latency performance with independence of the assigned bandwidth. The DRR-CA scheduler is only able to provide a different saturation point to each SC depending on the bandwidth assignment. The SCFQ-CA scheduler provides those SCs that have been assigned the same bandwidth with the same latency performance. On the other hand, those SCs that have been assigned a different bandwidth receive a different latency performance. Finally, the DTable scheduler is able to provide a different latency performance to those SCs that have been assigned a different maximum distance between any pair of consecutive table entries. This is true even when the SCs have been assigned the same bandwidth.

The second scenario was intended to evaluate the performance of our proposals to provide QoS in an environment with a realistic mix of different traffic classes. Specifically, we have simulated a multimedia scenario. Simulation results show that with the bandwidth broker for controlling the QoS traffic and a relatively small amount of control traffic, we can control the throughput, latency, and jitter performance of the QoS traffic whatever the load of best-effort traffic.

Simulation results have also shown that the DRR scheduler provides by far the worst latency performance of the four schedulers considered. Moreover, when the buffer size is small the DRR scheduler is not even able to provide proper bandwidth guarantees. The performance provided by the DRR scheduler improves for small MTU values. However this only alleviates the problem. Note also that the performance of this scheduler depends

on the frame length, and thus, in other scenarios, the performance could be even worse. On the other hand, the SCFQ-CA and WFQ-CA schedulers provide practically the same performance. The DTable scheduler provides, except for the network control traffic, a performance only slightly worse than the performance obtained with the SCFQ-CA and WFQ-CA schedulers.

With these two simulation scenarios we have been able to study the capacity of the different schedulers to provide QoS requirements based on bandwidth, latency, and jitter. However, as stated in Section 3.5 in page 36, the performance that a scheduler is able to provide is not the only parameter that must be taken into account when deciding which is the most appropriate scheduler in a high-performance network with QoS support. Other very important property that a scheduling mechanism should satisfy is to have a low complexity [Siv00].

In Chapter 7 we have study the implementation and computational complexity of the different schedulers. Apart from the results shown in that chapter, Figure 9.24 shows a comparison of the complexity of the different schedulers with 8 VCs, a MTU of 32 flow control credits, and for the case of the DTable scheduler, an arbitration table of 128 entries. These values for the design parameters approximate the values employed in the two simulation scenarios. Specifically, this figure shows the increment in silicon area and minimum and maximum time required to perform the arbitration respect the silicon area and minimum time required by the DRR-CA scheduler.

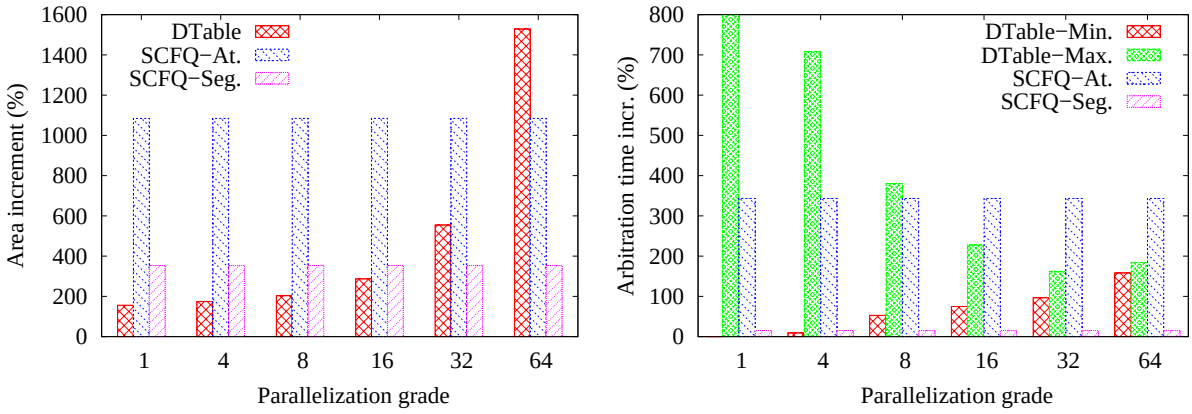


Figure 9.24: Silicon area and arbitration time comparison of the different schedulers with 8 VCs.

Figure 9.24 shows that the DTable scheduler requires less silicon area than both implementations of the SCFQ-CA scheduler (atomic and segmented) when the parallelization grade is less than 32. Regarding the arbitration time, this figure shows that the minimum

arbitration time of the DTable scheduler is smaller than the atomic SCFQ-CA scheduler. This would probably involve a faster arbitration time when the injection rate of all the SCs is high. However, the maximum arbitration time is higher than the arbitration time required by the atomic SCFQ-CA scheduler unless we increase the parallelization grade at least up to 16. This value for the parallelization grade seems to be the most appropriate. This is because it allows a faster arbitration than the atomic SCFQ-CA scheduler with less silicon area than both the atomic and segmented implementations of the SCFQ-CA scheduler. The segmented SCFQ-CA scheduler requires in general less time to perform the arbitration but, as stated in Section 7.3.2 in page 115, it requires more time to calculate the packet tags, and thus the comparison is no so direct and would require further study. In any case, Figure 9.24 shows that the DTable scheduler requires less silicon area to be implemented and thus, in some situations where this parameter is critical, the DTable scheduler can be the best option.

Summing up, if we consider the results that the simulations have shown and the analysis on the complexity performed in Section 7.5 in page 127 and also in this section, we can conclude that the DRR-CA scheduler is the simplest of all the schedulers but, it is not appropriate to correctly provide QoS requirements. On the other hand, the DTable scheduler can be a good possibility when the number of table entries required is not too high. Finally, the WFQ-CA scheduler only provides a slightly better performance than the SCFQ-CA scheduler and it is rather more complex than the SCFQ-CA scheduler. Therefore, the SCFQ-CA scheduler is probably the best option when the number of VCs is very high and thus, a too high number of table entries would be required for the DTable scheduler.

Chapter 10

Conclusions and future work

In this chapter we summarize the work done and discuss which are the main contributions of our work, which publications have followed, and which are the directions of future work.

10.1 Conclusions and contributions

At the beginning of this work, we set some objectives. These objectives were summarized in Section 1.3 in page 3 and now we review them and show in which degree they have been accomplished.

1. Studying the previous work. During the development of this thesis, a deep understanding of the operation of high-performance interconnects and Quality of Service (QoS) has been achieved. A summary of this can be found in Chapters 2 and 3.

2. Studying the Advanced Switching (AS) specification. A deep study of the specification, especially of the mechanisms intended to provide QoS requirements has been performed. A summary of this can be found in Chapter 4.

3. Developing a simulation tool to model high-performance networks. A general high performance network simulator has been developed in conjunction with Alejandro Martínez. This tool is a new development based on previous simulation tools used in the research group during many years. This simulator has been employed to obtain the performance results shown in Chapter 9. The development of this simulation tool has also helped to improve the understanding of the way of working of interconnection networks.

4. Proposing possible implementations for the MinBW scheduler. We have highlighted the considerations and problems that must be taken into account when implementing the MinBW scheduler. Specifically, the interaction with the AS link-level flow control. Most well-known scheduling algorithms were designed without taking into account this. We have presented in Chapter 5 three possible implementations for the MinBW scheduler, which are based on well-known scheduler algorithms, that fulfill all the requirements for the MinBW scheduler.

5. Solving the problem of the AS table scheduler with variable packet sizes. We have solved the problem of the table scheduler by proposing to incorporate a deficit mechanism, which makes the AS table scheduler to work in a proper way with variable packet sizes. The modified table-based scheduler, which we have called DTable, has been presented in Chapter 6.

6. Decoupling the bounding between the bandwidth and latency assignments of the table scheduler. The DTable scheduler may incorporate, apart from the deficit mechanism that solves the problem with variable packet sizes, a way to indicate a weight per each table entry. We have proposed a decoupling configuration methodology that assigns the table entries among the virtual channels attending to the latency requirements of the service classes, and the weights of the table entries attending to the service classes bandwidth requirements. In this way we achieve to decouple, at least partially, the bandwidth and latency assignation. We have presented this decoupling methodology in Chapter 6.

7. Studying the hardware complexity of the different schedulers. We have modeled the different schedulers in Handel-C, a high level hardware design language, in order to be able to obtain hardware estimates about the complexity of the schedulers in terms of silicon area and arbitration time. We have compared and analyzed the hardware requirements of the different schedulers with different values for the design parameters. We have presented this study in Chapter 7.

8. Proposing a general framework for providing QoS over AS. We have presented a traffic classification attending to bandwidth, latency, and jitter requirements. Moreover, we have proposed how to configure the schedulers and an admission control mechanism in order to provide QoS based on these requirements. We have presented these proposals in Chapter 8.

9. Evaluating our proposals from the performance point of view. We have evaluated the performance of our proposals with our simulation tool. We have considered

the traditional QoS indices such as latency, jitter, and throughput. We have also compared the performance provided by the different schedulers. We have presented the main results of this performance evaluation in Chapter 9.

Therefore, we consider that all the objectives initially proposed are satisfactory accomplished.

10.2 Applicability of our proposals in other technologies

Although our proposals have been intended for being applied in systems based on AS, they can be applied to other present and future network technologies. Our study on the problems of the interaction between the link-level flow control mechanism and the egress link scheduling mechanism is equally valid to any technology that performs both, the scheduling and the flow control, at a virtual channel level. Therefore, our credit aware versions of the DRR, SCFQ, and WFQ scheduling algorithms can be directly implemented in such technologies. Moreover, those new algorithms can be used as guidelines to adapt other well-known scheduling algorithms to interact in a proper way with the flow control mechanism.

The DTable scheduler can be implemented in any network technology in which the egress link scheduling is performed at a virtual channel level. This scheduler would allow providing bandwidth and latency requirements to the traffic that traverses each virtual channel with a high degree of independence. Specifically, it could be easily implemented in InfiniBand just including the deficit mechanisms in the InfiniBand table-based scheduler. This would allow the InfiniBand scheduler to work in a proper way with variable packet sizes. Note that, although the DTable based scheduler can actually be used at a flow level, in order to handle a high number of flows, an arbitration table with a lot of entries would probably be required an thus, the DTable scheduler would require too much silicon area to be implemented and its arbitration time would be too high.

The bandwidth broker admission control mechanism, which we have proposed to employ in order to be able to provide QoS guarantees, can be employed in any technology with adaptative source routing or at least deterministic routing. Note, that the requirement is that all the packets belonging to the same connection must traverse the same path through the network.

10.3 Publications

The different proposals, developments, and results compiled in this thesis have yield to several articles that have been published in journals or presented in international conferences and published in their proceedings. In the following, we show all these publications and give a brief description of their main contributions.

10.3.1 International journals

- Martínez Vicente, Alejandro; *Martínez Moráis, Raúl*; Alfaro Cortés, Francisco J.; Sánchez García, José L. **A Low-Cost Strategy to Provide Full QoS Support in Advanced Switching Networks.** Journal of Systems Architecture. July 2007. Impact: 0.402 (JCR 2005).

  In this paper, we compare the performance of the mechanisms provided by AS with a novel proposal to reduce the number of virtual channels.

- *Martínez Moráis, Raúl*; Alfaro Cortés, Francisco J.; Sánchez García, José L. **A framework to provide Quality of Service over Advanced Switching.** IEEE Transactional Parallel and Distributed Systems (TPDS). State: **Under major revision**.

  In this paper, which is under major revision, we present our general proposals to provide QoS over AS in a comprehensive way. We also present the DRR-CA, SCFQ-CA, and WFQ-CA implementations of the MinBW scheduler. Moreover, we employ the DTable scheduler with a fixed weight for all the table entries for the AS table scheduler.

- *Martínez Moráis, Raúl*; Alfaro Cortés, Francisco J.; Sánchez García, José L. **Providing QoS based on bandwidth and latency requirements with the Deficit Table scheduler.** IEEE Transactions on Computers (TC). State: **Under first revision**.

  In this paper, which is under first revision, we thoroughly review the DTable scheduler and our configuration methodology. Moreover, we show the advantages of our decoupling configuration methodology over the emulation of some "sorted-priority" algorithm like the WF2Q algorithm.

10.3.2 International conference with proceedings published by LNCS

- *Martínez Moráis, Raúl*; Alfaro Cortés, Francisco J.; Sánchez García, José L. **Improving the Flexibility of the Deficit Table Schedulers.** Lecture Notes in Computer Science Vol. 4297 (Proceedings of the International Conference on High Performance Computing, HiPC), December 2006. Acceptance rate: 52/282 = 18.4%.

  In this paper, we propose to employ a different specific MTU per each virtual channel in order to improve the flexibility of our DTable decoupling algorithm. Moreover, we compare the performance of the DTable scheduler with the performance of the SCFQ-CA and DRR-CA schedulers.

- *Martínez Moráis, Raúl*; Alfaro Cortés, Francisco J.; Sánchez García, José L. **Studying several proposals for the adaptation of the DTable scheduler to Advanced Switching.** Lecture Notes in Computer Science Vol. 4330 (Proceedings of the 2006 International Symposium on Parallel and Distributed Processing and Applications, ISPA), December 2006. Acceptance rate: 81/270 = 30%.

  In this paper, we present three different possibilities to implement a full version of the DTable scheduler in AS.

- Sï¿½dring, Thomas; *Martínez Moráis, Raúl*; Horn, Geir. **A Statistical Approach to Traffic Management in Source Routed Loss-Less Networks.** Lecture Notes in Computer Science Vol. 4208 (Proceedings of the High Performance Computing and Communications, HPCC), September 2006. Acceptance rate: 95/328 = 28.96%.

  In this paper, we present a traffic management mechanism to provide QoS to two service classes with only one virtual channel over AS.

10.3.3 International conference with proceedings published by IEEE

- *Martínez Moráis, Raúl*; Alfaro Cortés, Francisco J.; Sánchez García, José L. **Providing Quality of Service over Advanced Switching.** International Conference on Parallel and Distributed Systems (ICPADS), July 2006. Acceptance rate: 64/185 = 35%.

  In this paper, we present a first approach of our general proposals to provide QoS over AS. We also present the SCFQ-CA implementation of the MinBW scheduler. Moreover, we employ the DTable scheduler with a fixed weight for all the table entries for the AS table scheduler.

- *Martínez Moráis, Raúl*; Alfaro Cortés, Francisco J.; Sánchez García, José L. **Implementing the Advanced Switching Minimum Bandwidth Egress Link Scheduler.** 5th IEEE International Symposium on Network Computing and Applications (NCA), July 2006. Acceptance rate: 35%.

  In this paper, we present a first version of our credit aware versions of the DRR, SCFQ, and WFQ scheduling algorithms.

- *Martínez Moráis, Raúl*; Alfaro Cortés, Francisco J.; Sánchez García, José L. **Decoupling the Bandwidth and Latency Bounding for Table-based Schedulers.** 2006 International Conference on Parallel Processing (ICPP), August 2006. Acceptance rate: 64/200 = 32%.

  In this paper, we present the DTable scheduler and its decoupling configuration methodology.

- *Martínez Moráis, Raúl*; Alfaro Cortés, Francisco J.; Sánchez García, José L. **Evaluating Several Implementations for the AS Minimum Bandwidth Egress Link Scheduler.** 15th International Conference on Computer Communications and Networks (ICCCN), October 2006. Acceptance rate: 71/221 = 32.12%.

  In this paper, we thoroughly review the DRR-CA, SCFQ-CA, and SCFQ-CA scheduler algorithms. Moreover, we make a theoretical study on their complexity and compare their performance not only in terms of bandwidth and latency, but also jitter.

- *Martínez Moráis, Raúl*; Alfaro Cortés, Francisco J.; Sánchez García, José L. **Comparing the latency performance of the DTable and DRR schedulers.** Workshop on Communications Architecture for Clusters (CAC), proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium (IPDPS), March 2007. Acceptance rate: 10/31 = 32.25%.

  In this paper, we thoroughly compare the performance and characteristics of the DTable scheduler with the DRR scheduler

10.3.4 Other international publications

- *Martínez Moráis, Raúl*; Alfaro Cortés, Francisco J.; Sánchez García, José L.; Skeie, Tor. A First Approach to Provide QoS in Advanced Switching. Poster in the 12th IEEE International Conference on High Performance Computing (HiPC), December 2005.

In this paper, we present a very first approach of how to provide QoS over AS with fixed packet sizes.

10.4 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:

- **Proposing more implementations for the MinBW scheduler.** In this work we have proposed three possible implementations for the MinBW scheduler based on well-known scheduling algorithms. It would be interesting to study the effect of the link-level flow control in other well-known scheduling algorithms and propose modifications when needed to solve the problems that may arise.

- **Analytical study of the properties of the different schedulers.** In this work, we have evaluated the performance of the different schedulers by simulation. However it would be very interesting to perform an analytical study of the different schedulers in order to obtain their formal characteristics. Specifically, regarding to their latency characteristics. In the case of the MinBW scheduler, this study would be focused on determining the effect of the flow control mechanism over the formal properties of the well-known schedulers that we have considered. In the case of the DTable scheduler, this study would be focused on obtaining expressions that would indicate the latency bounding that can be provided with the DTable scheduler depending on the maximum distance between any consecutive pair of table entries.

- **Tuning of the connection admission control.** When a request for a new connection is performed the admission control must try to reserve the connection an amount of bandwidth all along its path. This reservation should be done based on the average latency and burstiness of the new connection. In this work we have configured the admission control in an intuitive way. We have tested several possible load values for the video traffic, which is the most problematic kind of traffic, and we have chosen the most appropriate load value. As stated in Section 3.4.6 a lot of different works have been presented on how to make this bandwidth reservation. This is the reason because we have not studied this issue more deeply. However, it would be interesting to evaluate and tune the performance of several of those proposals in this environment.

- **Evaluation of the complexity of the proposed schedulers in a full hardware system model.** In this work, we have modeled a full egress queuing system in Handel-C to test that the scheduler implementations are working in a proper way. However, this implementation is just an emulation of a real egress queuing system. It would be very interesting to model a full real system, or to implement our schedulers in an existing model, that would include endpoints and switches. This would allow us to study the real interaction of the scheduler with the rest of components of an endpoint or switch. Specifically, it would allow us to obtain a more realistic cycle time and also information on the effect of the time required to stamp the packets in the SCFQ-CA scheduler.

- **Evaluation of the complexity of the proposed schedulers in an ASIC platform.** In this work, we have obtained hardware estimates about the complexity of the schedulers that we have proposed, by obtaining the estimation on how many NAND gates and how much time would require the arbitration in a specific FPGA. Although we have obtained the hardware estimates for all the schedulers comparing the results for the same FPGA, it can still be some kind of dependence because of the specific FPGA architecture and features. It would be interesting to obtain estimates for an Application Specific Integrated Circuit (ASIC). These estimates would be independent of any specific FPGA.

- **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 nor 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.

- **Improvement of multimedia traffic.** When we have modeled 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 treated in this thesis.

Bibliography

[Adv03] Advanced Switching Interconnect Special Interest Group. *Advanced Switching core architecture specification. Revision 1.0*, December 2003.

[Adv05] Advanced Switching Interconnect Special Interest Group. *Advanced Switching core architecture specification. Revision 1.1*, March 2005.

[ANS93] ANSI. *Fibre channel physical and signaling interface, Rev. 4.2.* Technical report, X3T9.3 Task Group, October 1993.

[AOS93] 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.

[ASD03] F. J. Alfaro, J. L. Sánchez, J. Duato. *A new proposal to fill in the InfiniBand arbitration tables.* In IEEE Int. Conference on Parallel Processing (ICPP), pp. 133 – 140, October 2003.

[ASD04] F. J. Alfaro, J. L. Sánchez, J. Duato. *QoS in InfiniBand subnetworks.* IEEE Transactions on Parallel and Distributed Systems, Vol. 15, No. 9, pp. 810–823, September 2004.

[Ash02] P. J. Ashenden. *The Designer's Guide to VHDL.* Morgan Kaufmann; 2nd edition, 2002.

[BAP03] J. Beecroft, D. Addison, F. Petrini, M. McLaren. *Quadrics QsNet II: A network for supercomputing applications.* In In Hot Chips 15, Stanford University, Palo Alto, CA, August 2003. Available from `http://www.c3.lanl.gov/~fabrizio/papers/hot03.pdf`.

[BBC98] 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, December 1998.

[BCF95] N. J. Boden, D. Cohen, R. E. Felderman. *Myrinet – A Gigabit per Second Local Area Network*. IEEE Micro, pp. 29–36, February 1995.

[BCN99] D. Bull, N. Conagarajah, A. Nix. *Insights into Mobile Multimedia Communications*. Academic Press, 1999.

[BCS94] 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.

[Ber98] Y. Bernet. *A Framework for Differentiated Services*. Internet draft 2275, Internet Engineering Task Force, May 1998.

[BKS00] L. Breslau, E. W. Knightly, S. Shenker, I. Stoica, H. Zhang. *Endpoint admission control: Architectural issues and performance*. In SIGCOMM, pp. 57–69, 2000.

[Bla00] U. Black. *QoS in Wide Area Networks*. Prentice Hall Series in Advanced Communications Technologies. Prentice Hall, 2000.

[BZ96] J. Bennett, H. Zhang. *WF2Q: Worst-case fair weighted fair queueing*. INFOCOM, 1996.

[BZ97] J. C. R. Bennett, H. Zhang. *Hierarchical packet fair queueing algorithms*. IEEE/ACM Trans. Netw., Vol. 5, No. 5, pp. 675–689, 1997.

[BZB97] 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, September 1997.

[Cel05] Celoxica. *Handel-C Language Reference Manual for DK4*, 2005.

[CG01] H. Chao, X. Guo. *Quality of Service Control in High-Speed Networks*. Wiley, 2001.

[Chr04] S. Christo. Markets converge on advanced switching, may 2004. RTC Magazine. `www.rtcmagazine.com/home/article.php?id=100018`.

[CK04] N. Chrysos, M. Katevenis. *Multiple priorities in a two-lane buffered crossbar*. In Proceedings of the IEEE Globecom 2004 Conference, November 2004.

[CM03] H. M. Chaskar, U. Madhow. *Fair scheduling with tunable latency: A round-robin approach*. IEEE/ACM Transactions on Networking, Vol. 11, No. 4, pp. 592–601, 2003.

[CMR06] L. Cheng, N. Muralimanohar, K. Ramani, R. Balasubramonian, J. B. Carter. *Interconnect-aware coherence protocols for chip multiprocessors.* In ISCA, pp. 339–351. IEEE Computer Society, 2006.

[CS99] D. Chalmers, M. Sloman. *A survey of Quality of Service in mobile computing environments.* IEEE Communications Surveys and Tutorials, Vol. 2, No. 2, 1999.

[DCD98] W. Dally, P. Carvey, L. Dennison. *Architecture of the Avici Terabit switch/router.* In Proceedings of the 6th Symposium on Hot Interconnects, 1998.

[DKS89] A. Demers, S. Keshav, S. Shenker. *Analysis and simulations of a fair queuing algorithm.* In SIGCOMM, 1989.

[DYL02] J. Duato, S. Yalamanchili, N. Lionel. *Interconnection networks. An engineering approach.* Morgan Kaufmann Publishers Inc., 2002.

[ECT05] I. Elhanany, D. Chiou, V. Tabatabaee, R. Noro, A. Poursepanj. *The network processing forum switch fabric benchmark specifications: An overview.* IEEE Network, March 2005.

[EGBS03] 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, 2003.

[FBT01] V. Firoiu, J-Y. Le Boudec, D. Towsley, Z-L. Zhang. *Advances in internet Quality of Service.* Technical report, National Science Foundation, 2001.

[FH98] P. Ferguson, G. Huston. *Quality of Service: delivering QoS on the Internet and in corporate networks.* John Wiley & Sons, Inc., 1998.

[FJ93] S. Floyd, V. Jacobson. *Random early detection gateways for congestion avoidance.* IEEE/ACM Transactions on Networking, Vol. 1, No. 4, pp. 397–413, 1993.

[For95] ATM Forum. *ATM Forum traffic management specification. Version 4.0*, May 1995.

[GG99] N. Giroux, S. Ganti. *Quality of Service in ATM Networks.* Prentice Hall, 1999.

[GGK99] P. Goyal, A. Greenberg, C. R. Kalmanek, W. T. Marshall, P. Mishra, D. Nortz, K. K. Ramakrishnan. *Integration of call signaling and resource management for IP telephony.* IEEE Internet Computing, Vol. 3, No. 3, pp. 44–52, 1999.

[GM92] A. G. Greenberg, N. Madras. *How fair is fair queuing.* J. ACM, Vol. 39, No. 3, pp. 568–598, 1992.

[Gol94] S. J. Golestani. *A self-clocked fair queueing scheme for broadband applications.* In INFOCOM, 1994.

[GQF06] P. J. García, F. J. Quiles, J. Flich, J. Duato, I. Johnson, F. Naven. *Efficient, scalable congestion management for interconnection networks.* IEEE Micro, Vol. 26, No. 5, pp. 52–66, 2006.

[GT99] M. Grossglauser, D. N. C. Tse. *A framework for robust measurement-based admission control.* IEEE/ACM Transactions on Networking, Vol. 7, No. 3, pp. 293–309, 1999.

[GTP04] S. Georgoulas, P. Trimintzios, G. Pavlou. *Joint measurement and traffic descriptor-based admission control at real-time traffic aggregation points.* Paris, France, June 2004.

[Hal01] F. Halsall. *Multimedia Communications: Applications, Networks, Protocols and Standard.* Addison-Wesley, 2001.

[HK88] 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, December 1988.

[HS05] G. Horn, T. Sødring. *SH: A simple distributed bandwidth broker for source-routed loss-less networks.* In Computer, Networks and Information Security. IASTED, 2005.

[IEE04] IEEE. *802.1D-2004: Standard for local and metropolitan area networks.* `http://grouper.ieee.org/groups/802/1/`, 2004.

[Inf00] InfiniBand Trade Association. *InfiniBand architecture specification volume 1. Release 1.0*, October 2000.

[Jai91] R. Jain. *The art of computer system performance analysis: Techniques for experimental design, measurement, simulation and modeling.* John Wiley and Sons, Inc., 1991.

[KHM87] 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.

[KLC98] 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.

[KPC99] P. Krishna, N. Patel, A. Charny, R. Simcoe. *On the speedup required for work-conserving crossbar switches.* IEEE J. Sel. Areas in Communications, Vol. 17, No. 6, pp. 1057–1066, June 1999.

[KPS04] M. Katevenis, G. Passas, D. Simos, I Papaefstathiou, N. Chrysos. *Variable packet size buffered crossbar (cicq) switches.* In IEEE Int. Conference on Communications (ICC 2004), pp. 20–24, Paris, France, June 2004.

[KS99] E.W. Knightly, N. B. Shroff. *Admission control for statistical QoS: Theory and practice.* IEEE Network, Vol. 13, No. 2, pp. 20–29, 1999.

[KSC91] M. Katevenis, S. Sidiropoulos, C. Courcoubetis. *Weighted round-robin cell multiplexing in a general-purpose ATM switch chip.* IEEE Journal on Selected Areas in Communications, October 1991.

[KSP02] S. S. Kanhere, H. Sethu, A. B. Parekh. *Fair and efficient packet scheduling using elastic round robin.* IEEE Transactions on Parallel and Distributed Systems, 2002.

[LY99] J. Liebeherr, E. Yilmaz. *Workconserving vs.non-workconserving packet scheduling: An issue revisited.* In IEEE/IFIP International Workshop on Quality of Service (IWQOS), May 1999.

[LZ99] T. K. Lee, M. Zukerman. *An ef@bookChao2001, author = H. Chao and X. Guo, title = Quality of Service Control in High-Speed Networks, publisher = Wiley, year = 2001, ficiency study of different model-based and measurement-based connection admission control techniques using heterogeneous traffic sources.* 1999.

[LZ01] T. K. Lee, M. Zukerman. *Admission control schemes for bursty multimedia traffic.* Alazka, USA, April 2001.

[MAS06] R. Martínez, F. J. Alfaro, J.L. Sánchez. *Decoupling the bandwidth and latency bounding for table-based schedulers.* International Conference on Parallel Processing (ICPP), August 2006.

[MK03] D. Mayhew, V. Krishnan. *PCI Express and Advanced Switching: Evolutionary path to building next generation interconnects.* In Hot Interconnects: 10th Symposium on High Performance Interconnects, 2003.

[MP01] P. L. Montessoro, D. Pierattoni. *Advanced research issues for tomorrow's multimedia networks.* In International Symposium on Information Technology (ITCC), 2001.

[Pal03] S. Palnitkar. *Verilog HDL.* Prentice Hall PTR; 2 edition, 2003.

[Par92] C. Partridge. *A proposed flow specification.* Internet Request for Comment RFC 1363, Internet Engineering Task Force, September 1992.

[Par05] K. I Park. *QoS in Packet Networks.* Springer, 2005.

[PCI03] PCI SIG. *PCI Express base architecture specification. Revision 1.0a*, April 2003.

[Pel00] J. Pelissier. *Providing Quality of Service over Infiniband architecture fabrics.* In Proceedings of the 8th Symposium on Hot Interconnects, August 2000.

[PG93] A. K. Parekh, R. G. Gallager. *A generalized processor sharing approach to flow control in integrated services networks: The single-node case.* IEEE/ACM Transactions on Networking, 1993.

[PG94] A. K. Parekh, R. G. Gallagher. *A generalized processor sharing approach to flow control in integrated services networks: The multiple node case.* IEEE/ACM Transactions on Networking, 1994.

[PN85] G. Pfister, A. Norton. *Hot spot contention and combining in multistage interconnection networks.* IEEE Transactions on Computers, Vol. C-34, 10, pp. 943–948, 1985.

[RGB96] J. Rexford, A. G. Greenberg, F. Bonomi. *Hardware-efficient fair queueing architectures for high-speed networks.* In INFOCOM (2), pp. 638–646, 1996.

[RS05] Sven-Arne Reinemo, Tor Skeie. *Ethernet as a lossless deadlock free system area network.* In ISPA, pp. 901–914, 2005.

[RSJS03] S.A. Reinemo, F.O. Sem-Jacobsen, T. Skeie, O. Lysne. *Admission control for diffserv based Quality of Service in cut-through networks.* In Proceedings of the 10th Int. Conference on High Performance Computing, December 2003.

[RSS06] Sven-A. Reinemo, T. Skeie, T. Sødring, O. Lysne, O. Tørudbakken. *An overview of QoS capabilities in infiniband, advanced switching interconnect, and ethernet.* IEEE Communications Magazine, Vol. 44, No. 7, pp. 32–38, 2006.

[Sei98] R. Seifert. *Gigabit Ethernet: Technology and Applications for High-Speed LANs.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998.

[Siv00] V. Sivaraman. *End-to-Ent delay service in high speed packet networks using Erliest Deadline First Scheduling.* PhD thesis, University of California, 2000.

[Sta04] StarGen. *StarGen's Merlin switch*, 2004. `http://www.stargen.com/products/merlin_switch.shtml`.

[Sti96] D. Stiliadis. *Traffic scheduling in packet-switched networks: Analysis, design, and implementation.* PhD thesis, University of California, 1996.

[SV95] M. Shreedhar, G. Varghese. *Efficient fair queueing using deficit round robin.* In SIGCOMM, pp. 231–242, 1995.

[SV96] D. Stiliadis, A. Varma. *Design and analysis of frame-based fair queueing: A new traffic scheduling algorithm for packet-switched networks.* SIGMETRICS Perform. Eval. Rev., Vol. 24, No. 1, pp. 104–115, 1996.

[SV98] D. Stiliadis, A. Varma. *Latency-rate servers: A general model for analysis of traffic scheduling algorithms.* IEEE/ACM Transactions on Networking, 1998.

[TB03] D. Tutsch, M. Brenner. *MINSimulate - a multistage interconnection network simulator.* In 17th European Simulation Multiconference: Foundations for Successful Modelling and Simulation (ESM'03), pp. 211–216, 2003.

[TMdM00] A. Tyagi, J. K. Muppala, H. de Meer. *VoIP support on differentiated services using expedited forwarding.* In IEEE International Performance, Computing, and Communications Conference (IPCCC), February 2000.

[Tur86] J. S. Turner. *New directions in communications (or which way to the information age).* IEEE Communications, Vol. 24, No. 10, pp. 8–15, October 1986.

[Vid] Video Traces Research Group. *YUV Video Sequences*. `http://trace.eas.asu.edu/yuv/index.html`.

[VV04] P. Vellore, R. Venkatesan. *Performance analysis of scheduling disciplines in hardware*. In Canadian Conference on Electrical and Computer Engineering (CCECE), May 2004.

[Wan01] Z. Wang. *Internet QoS: Architecture and Mechanisms for Quality of Service*. Morgan Kaufmann, 2001.

[WZ98] Y. Wang, Q. Zhu. *Error control and concealment for video communication: A review*. Proceedings of the IEEE, Vol. 86, No. 5, pp. 974–997, May 1998.

[Xil] Xilinx, Inc. `http://www.xilinx.com`.

[Xil07] Xilinx. Virtex-4 family overview. Fact sheet DS112 (v2.0), June 2007.

[XL05] Jun Xu, Richard J. Lipton. *On fundamental tradeoffs between delay bounds and computational complexity in packet scheduling algorithms*. IEEE/ACM Trans. Netw., Vol. 13, No. 1, pp. 15–28, 2005.

[XN99] X. Xiao, L.M. Ni. *Internet QoS: A Big Picture*. IEEE Network Magazine, pp. 8–18, March 1999.

[Zha95] H. Zhang. Service disciplines for guaranteed performance service in packet-switching networks, 1995.

[ZX04] Q. Zhao, J. Xu. *On the computational complexity of maintaining gps clock in packet scheduling*. In IEEE INFOCOM, March 2004.