Hardware Implementation Study of the SCFQ-CA and DRR-CA Scheduling Algorithms*

Raúl Martínez[1], Francisco J. Alfaro[2], José L. Sánchez[2], and José M. Claver[3]

[1] Intel-UPC Barcelona Research Center,
Barcelona, SPAIN 08034
raulmm@dsi.uclm.es
[2] Dpto. de Sistemas Informáticos. Univ. Castilla-La Mancha
Albacete, SPAIN 02071
{falfaro,jsanchez}@dsi.uclm.es
[3] Dpto de Informática. Univ. Valencia
Valencia, SPAIN 46100
jclaver@uv.es

Abstract. The provision of Quality of Service (QoS) in computing and communication environments has been the focus of much research in industry and academia during the last decades. A key component for networks with QoS support is the egress link scheduling algorithm. Apart from providing a good performance in terms of, for example, good end-to-end delay and fair bandwidth allocation, an ideal scheduling algorithm implemented in a high-performance network with QoS support should satisfy other important property which is to have a low computational and implementation complexity. This is especially important in high-performance networks due to their high speed and because switches are usually implemented in a single chip.

In [7] we proposed the Self-Clocked Fair Queuing Credit Aware (SCFQ-CA) and the Deficit Round Robin Credit Aware (DRR-CA) schedulers in order to adapt the SCFQ and DRR algorithms to networks with a link-level flow control mechanism. In this paper, we propose specific implementations of these two schedulers taking into account the characteristics of current high-performance networks. Moreover, we compare the complexity of these two algorithms in terms of silicon area and computation delay. In order to carry out this comparison, we have performed our own hardware implementation for the different schedulers. We have modeled the schedulers using the Handel-C language and employed the DK design suite tool from Celoxica in order to obtain hardware estimates on silicon area and arbitration time.

* This work has been jointly supported by the Spanish MEC and European Comission FEDER funds under grants Consolider Ingenio-2010 CSD2006-00046 and TIN2006-15516-C04-02 and by Junta de Comunidades de Castilla-La Mancha under grant PCC08-0078-9856. Raúl Martínez was with the University of Castilla-La Mancha when the main ideas of the paper where developed.

H. Sips, D. Epema, and H.-X. Lin (Eds.): Euro-Par 2009, LNCS 5704, pp. 1089–1100, 2009.

1 Introduction

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 [8]. 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 [3].

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, InfiniBand, or Advanced Switching (AS). An interesting survey with the QoS capabilities of these network technologies can be found in [10]. Common characteristics of the specifications of these network technologies intended to provide QoS are the use of a link-level flow control mechanism, which makes the networks lossless, a reduced set of Virtual Channels (VCs), and an egress link scheduler to arbitrate among the traffic transmitted in each VC. These last two mechanisms permit us to aggregate traffic with similar characteristics in the same VC and to provide each VC with a different treatment according to its requirements, at the style of the differentiated services (DiffServ) QoS model [1].

A key component for networks with QoS support is the output (or egress link) scheduling algorithm (also called service discipline), which selects the next packet to be transmitted and decides when it should be transmitted [4], [18]. A lot of possible scheduling algorithms have been proposed in the literature. However, most of these algorithms were proposed for the Internet scenario with a high number of possible flows, without taking into account a link-level flow control mechanism, and were intended mainly to be implemented by software. These characteristics differ from the requirements of current high-performance networks. In [7], we already gave a first step in adapting some scheduling algorithms to high-performance networks by proposing new versions of some schedulers in order to support a link-level flow control mechanism.

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 important property which is to have a low computational and implementation complexity [13]. 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 that 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 the scenario that we are addresing here is very different than for example IP routers with QoS support, where these algorithms are usually implemented by software instead of harware.

The fair queuing algorithms are an important kind of scheduling algorithms that 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 [4], [9], which is based on a fluid model that provides perfect instant fairness in bandwidth allocation and has many desirable properties [15], [9]. 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.

The "Sorted-priority" family of algorithms, which includes the Weighted Fair Queuing (WFQ) [4] and Self-Clock Fair Queueing SCFQ [6] algorithms, are known to offer good delay bounds [14]. This kind of scheduling algorithms assign each packet a tag and the scheduling is made based on the ordering of these tags. The main complexity factors in this kind of schedulers comes from tag calculation and tag sorting. Note that these algorithms were proposed mainly for the Internet scenario with a high number of possible flows. Moreover, a common problem in the sorted-priority approach is that tags cannot be reinitialized to zero until the system is completely empty and all the sessions are idle. The reason is that these tags depend on a common-reference virtual clock and are an increasing function of the time. 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 exhibits self-similar patterns which lead to heavily tailed buffer occupancy distributions.

To avoid the complexity of the sorted-priority approach, the Deficit Round Robin (DRR) algorithm [12] has been proposed. The aim of DRR is to implement fair queuing and achieve practically acceptable complexity at the expense of other performance metrics such as fairness and delay. Due to its computational simplicity, recent research in the Differentiated Services area proposes the DRR as a feasible solution for implementing the Expedited Forwarding Per-hop Behavior [5].

In this paper, we propose specific implementations (taking into account the characteristics of current high performance networks) of the SCFQ and DRR scheduling algorithms and compare their complexity in terms of silicon area and computation delay. In fact, we are going to consider the modified versions of these two algorithms that we proposed in [7] in order to maintain the fair bandwidth allocation in networks with a link-level flow control network, which is the case in most current high-performance network technologies. We have chosen the SCFQ algorithm as an example of "sorted-priority" algorithm and the DRR algorithm because of its very small computational complexity.

In [16] and [11] interesting implementations for the SCFQ scheduler are proposed. However, these implementations were designed for a high number of possible flows. Note that in our case there is going to be just a limited number

of VCs. This allows us to consider more efficient implementations. Moreover, the case of the SCFQ implementation [11] 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 the Handel-C language [2] and employed the DK design suite tool from Celoxica in order to obtain hardware estimates on silicon area and arbitration time.

The structure of the paper is as follows: Sections 2 and 3 present the DRR-CA and SCFQ-CA scheduling algorithms, respectively, state how we have adapted them to current high-performance network characteristics, and describe specific hardware implementation decisions. In Section 4 a comparison study on the implementation and computational complexity of the two schedulers is provided. Finally, some conclusions are given.

2 Implementation of the SCFQ-CA Scheduler

The Self-Clocked Weighted Fair Queuing (SCFQ) algorithm [6] is a variant of the well known Weighted Fair Queuing (WFQ) mechanism [4] which has a lower computational complexity. This objective is accomplished by adopting a different notion of the virtual time employed in WFQ. Instead of linking the 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 the virtual time that may make WFQ unfeasible in high-speed interconnection technologies.

Therefore, when a packet arrives, SCFQ uses the service tag of the packet currently in service as the virtual time to calculate the new packet tag. Thus, in this case the service tag of the k^{th} packet of the i^{th} flow is computed as

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

where L_i^k is the packet length, ϕ_i the weight assigned to its flow, and $S_{current}$ is the service tag of the packet being serviced. Figure 1 shows the pseudocode for the SCFQ algorithm.

In [7] we proposed a new version of this scheduler, the SCFQ Credit Aware (SCFQ-CA) algorithm, that takes into account the link level flow control mechanism and VCs instead of flows. The SCFQ-CA algorithm that we proposed works in the same way as the SCFQ algorithm, except in the following aspects:

- When a new packet arrives at a VC queue, a service tag is assigned 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 due to a lack of credits and receives enough credits to transmit again, a new service tag is assigned to the VC.

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)
        $\forall flow\ flow_{lastServiceTag} \leftarrow 0$
        $currentServiceTag \leftarrow 0$

Fig. 1. Pseudocode of the SCFQ scheduler

Note 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 until it arrives at the head of a VC. 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 at the head of the VC queue. Note that this makes 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^{th} VC.

With these modifications, we have already taken into account the presence of an egress link scheduler. However, we can take more advantage of the limited number of VCs to simplify the actual implementation. The SCFQ-CA algorithm, as the original SCFQ algorithm and most sorted-priority algorithms, 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-CA 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,

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

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 2. 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, which is the common trend in the last high-performance network proposals.

```
PACKET ARRIVAL(newPacket, VC):
if (newPacket is at the head in the queue of VC) and
      (The flow control does allow transmitting from VC))
            VC_serviceTag ← VC_sizeFirst / VC_reservedBandwidth
ARBITRATION:
while (There is at least one active VC)
      selectedVC ← Active VC with the minimum serviceTag
      currentServiceTag ← selectedVC_serviceTag
      Transmit packet from selectedVC
      ∀ active VC
            VC_serviceTag ← VC_serviceTag − currentServiceTag
      if ((There are more packets in the queue of selectedVC) and
            (The flow control does allow transmitting from selectedVC))
            selectedVC_serviceTag ← selectedVC_sizeFirst / selectedVC_reservedBandwidth
```

Fig. 2. Pseudocode of the improved SCFQ-CA scheduler

In order to implement this 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 before, we have solved the problem of the possible overflow of the service tags. Moreover, this modification entails a simplification of the service tag computation. 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, which obtains the selected VC in log(#VCs) cycles. The structure of the selector module is shown in Figure 3.

Note that, in order to calculate the packet service tag a division operation is performed. This operation requires an arithmetical unit that is not simple at all. Handel-C, which is the language that we have used to model the schedulers, offers a divisor operand that calculates the result in one cycle (as all the Handel-C

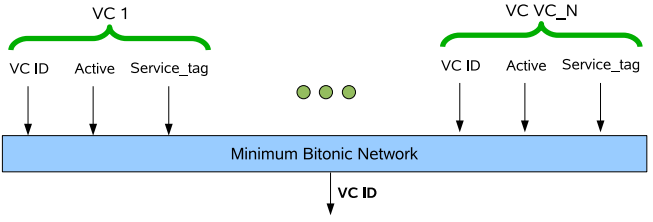

Fig. 3. Structure of the selector module for the SCFQ-CA scheduler

statements). 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 employed 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. With this new version the calculation of the time tag requires more cycles to be performed but reduces the cycle time and thus, the arbitration time, which is a more critical value. Therefore, the estimates that we show in Section 4 have been obtained using this second version.

3 Implementation of the DRR-CA Scheduler

The Deficit Round Robin (DRR) algorithm [12] is a variation of the Weighted Round Robin (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 flow is proportional to the bandwidth assigned to that flow. The sum of all

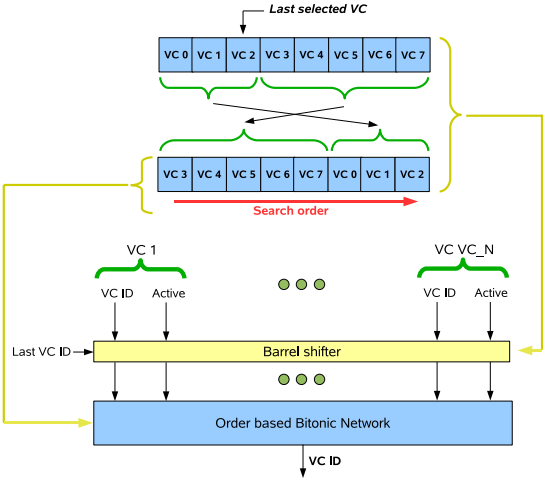

Fig. 4. Structure of the next VC to transmit selector in the DRR-CA scheduler

the quantums is called the frame length. For each flow, 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 flow. 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.

The DRR Credit Aware (DRR-CA) algorithm that we proposed in [7] works in the same way as the DRR algorithm, except in the following aspects:

- 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.

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. However, in order to make this search efficient, 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. The structure for this selector function is shown in Figure 4.

4 Hardware Estimates

In this section we analyze the implementation and computational complexity of the DRR-CA and SCFQ-CA schedulers. We have modeled these schedulers using Handel-C language [2] and employed the DK design suite tool from Celoxica in order to obtain hardware estimates on silicon area and arbitration time. Handel-C's level of design abstraction is above Register Transfer Level (RTL) languages, like VHDL and Verilog, but below behavioral. In Handel-C each assignment 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. Note that the Handel-C code that we have designed can actually be used to implement the 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, but to obtain the relative differences on

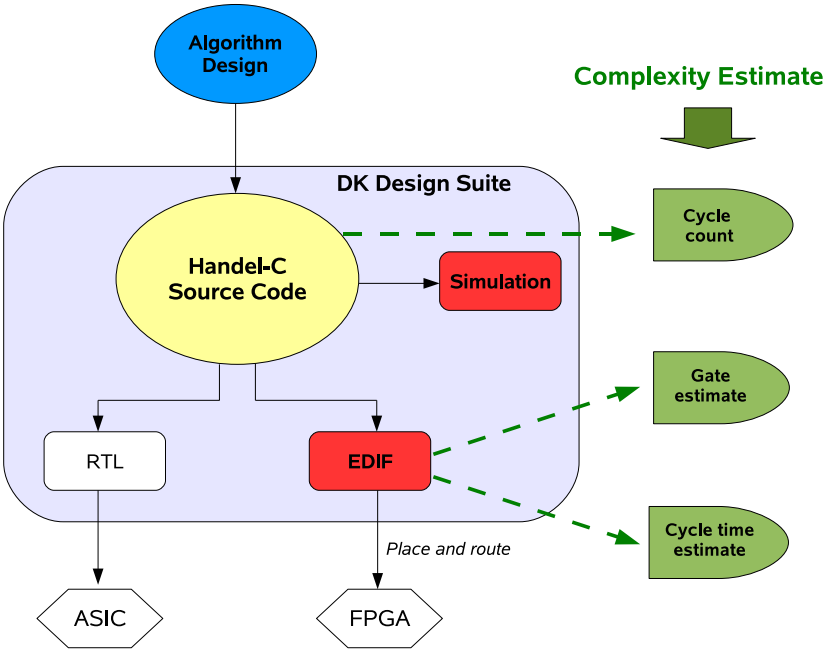

Fig. 5. Design flow with DK employing Handel-C

silicon area and arbitration time for the different schedulers and to measure the effect of the number of VCs and the Maximum Transfer Unit (MTU).

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

1. We have modeled in Handel-C a full egress queuing system, which includes the scheduling mechanism.
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 Electronic Design Interchange Format (EDIF) output for a Virtex 4 FPGA from Xilinx [17].

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 [17]. 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 of the different schedulers. Figure 5 reflects the design flow that we have followed.

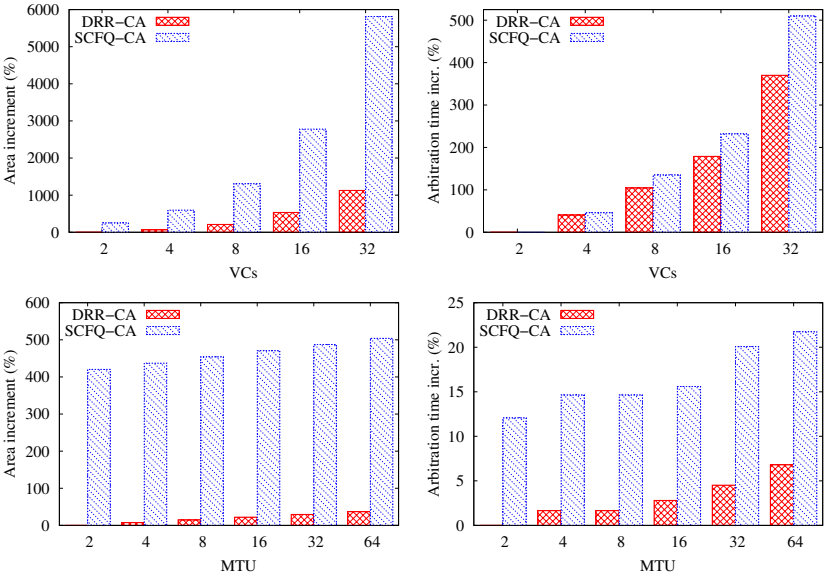

Fig. 6. Comparison of the silicon area and arbitration time required by the DRR-CA and the SCFQ-CA schedulers for different number of VCs and MTU

Figure 6 shows a comparison of the DRR-CA and SCFQ-CA algorithms and how the increment in the number of VCs and the MTU affects the complexity of the two schedulers. Specifically, we have varied the number of VCs with a fixed MTU of 32 and the MTU value with the number of VCs fixed to 8. The figure shows the percentage of increment in silicon area and arbitration time respect 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 the MTU).

Regarding the effect of the number of VCs, Figure 6 shows that this number 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 increase in the cycle time and the increase in the number of cycles required to compute the arbitration. Note also that the X axis is in logarithmic scale, thus a linear growth in data plot actually means a logarithmic data growth, and an exponential growth in data plot actually means a linear data growth. On the other hand, regarding the effect of the MTU, Figure 6 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.

Finally, this figure shows, as expected, that the DRR-CA scheduler is the simplest scheduler in terms of silicon area and arbitration time. The SCFQ-CA scheduler requires quite more silicon area than the DRR-CA scheduler. However, the difference in arbitration time is not so big. Nevertheles, as was shown in [7] for multimedia traffic the SCFQ-CA scheduler is able to provide much more tighter QoS requirements than the DRR-CA scheduler. These results are

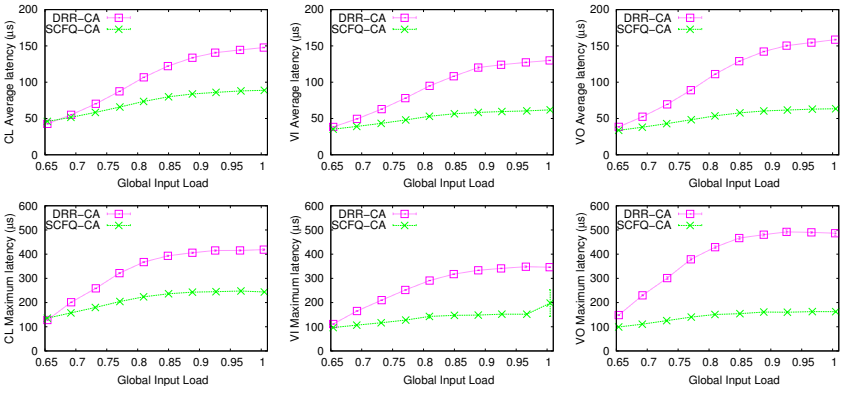

Fig. 7. Latency comparison for controled load (CL), video (VI), and voice (VO) traffic, for the DRR-CA and SCFQ-CA schedulers

depicted again in Figure 7, which shows the performance provided by the DRR-CA and SCFQ-CA schedulers to three types of multimedia traffic that are simultaneously injected into the network with different amounts of best-effort traffic.

5 Conclusions

Current high-performance networks with QoS support require egress link scheduling algorithms to be fast and simple. Moreover, most well-known algorithms were designed for managing a lot of flows in lossy networks. Therefore, it is necesary to propose specific implementations of the scheduling algorithms adapted to current high-performance requirements. In this work we have proposed implementations and performed a complexity study for two fair queueing scheduling algorithms, the DRR-CA and SCFQ-CA schedulers, taking into account these requirements, and with the objective of fullfilling the complexity constrains in high-performance networks.

The complexity estimates that we have obtained show that the SCFQ-CA scheduler is quite more complex than the DRR-CA in terms of silicon area. However, this algorithm provides much better performance in terms of latency. Therefore, the decision of choosing one of these algorithms for the switches of a real network will be determined by how sensitive to the QoS requirements are the applications that are expected to traverse that network and by the area that there is available to implement the schedulers at the egress links.

References

1. Blake, S., Back, D., Carlson, M., Davies, E., Wang, Z., Weiss, W.: An Architecture for Differentiated Services. Internet Request for Comment RFC 2475, Internet Engineering Task Force (December 1998)
2. Celoxica. Handel-C Language Reference Manual for DK4 (2005)

3. Cheng, L., Muralimanohar, N., Ramani, K., Balasubramonian, R., Carter, J.B.: Interconnect-aware coherence protocols for chip multiprocessors. In: ISCA, pp. 339–351. IEEE Computer Society, Los Alamitos (2006)
4. Demers, A., Keshav, S., Shenker, S.: Analysis and simulations of a fair queuing algorithm. In: SIGCOMM (1989)
5. Charny, A., et al.: Supplemental information for the new definition of EF PHB (Expedited Forwarding Per-Hop-Behavior). RFC 3247 (March 2002)
6. Golestani, S.J.: A self-clocked fair queueing scheme for broadband applications. In: INFOCOM (1994)
7. Martínez, R., Alfaro, F.J., Sánchez, J.L.: A framework to provide quality of service over Advanced Switching. IEEE Transactions on Parallel and Distributed Systems 19(8), 1111–1123 (2008)
8. Montessoro, P.L., Pierattoni, D.: Advanced research issues for tomorrow's multimedia networks. In: International Symposium on Information Technology, ITCC (2001)
9. Parekh, A.K., Gallager, R.G.: A generalized processor sharing approach to flow control in integrated services networks: The single-node case. In: IEEE/ACM Transactions on Networking (1993)
10. Reinemo, S.A., Skeie, T., Sødring, T., Lysne, O., Trudbakken, O.: An overview of QoS capabilities in InfiniBand, Advanced Switching Interconnect, and Ethernet. IEEE Communications Magazine 44(7), 32–38 (2006)
11. Rexford, J., Greenberg, A.G., Bonomi, F.: Hardware-efficient fair queueing architectures for high-speed networks. In: INFOCOM (2), pp. 638–646 (1996)
12. Shreedhar, M., Varghese, G.: Efficient fair queueing using deficit round robin. In: SIGCOMM, pp. 231–242 (1995)
13. Sivaraman, V.: End-to-Ent delay service in high speed packet networks using Erliest Deadline First Scheduling. PhD thesis, University of California (2000)
14. Stiliadis, D., Varma, A.: Latency-rate servers: A general model for analysis of traffic scheduling algorithms. In: IEEE/ACM Transactions on Networking (1998)
15. Turner, J.S.: New directions in communications (or which way to the information age). IEEE Communications 24(10), 8–15 (1986)
16. Vellore, P., Venkatesan, R.: Performance analysis of scheduling disciplines in hardware. In: Canadian Conference on Electrical and Computer Engineering (CCECE) (May 2004)
17. Xilinx. Virtex-4 family overview. Fact sheet DS112 (v2.0) (June 2007)
18. Zhang, H.: Service disciplines for guaranteed performance service in packet-switching networks (1995)