A DYNAMICALLY RECONFIGURABLE QUEUE SCHEDULER

Christoforos Kachris, Stamatis Vassiliadis[*]

Computer Engineering Laboratory
Delft University of Technology
The Netherlands
{kachris, stamatis}@ce.et.tudelft.nl

ABSTRACT

In this paper we present the design and implementation of a dynamically reconfigurable system for packet queue scheduling. Two widely accepted queue schedulers have been implemented in reconfigurable logic in a way that can be interchanged based on the number of active queues and the Quality of Service (QoS) requirements. The first algorithm is the Deficit Weighted Round Robin (DWRR) that is used to support up to 2K queues. The second algorithm is the Worst-case Fair Weighted Fair Queuing (WF2Q+) algorithm that is more accurate and closer to the ideal scheduling but the computation and memory requirements are higher. This scheme is used when the number of active queues is up to 341 queues or classes of services (CoS). The performance evaluation shows that we can exchange these algorithms, thus obtaining higher accuracy, depending on the variant and the characteristics of the network traffic with negligible performance degradation due to the dynamic reconfiguration overhead.

1. INTRODUCTION

The exponential increase of the bandwidth in the Internet has created the need for congestion management and congestion avoidance in the network devices. In addition, the requirements in terms of latency, jitter and bandwidth are different for each application in the Internet (e.g. the video and the voice traffic has higher priority over non-time critical applications such as file transfers traffic.). Hence, simple algorithms for the scheduling of packets (such as the round robin algorithm) are being abandoned in favor of more advanced scheduling algorithms that can support queues with different weights.

There are several algorithms that have been proposed for the scheduling of queues with different priorities [1-3]. Each algorithm has its own advantages and disadvantages such as the complexity of the algorithm, the latency and the error in bandwidth allocation. For example, the Deficit Weighted Round Robin (DWRR) can be easily implemented both in hardware and in software, it can support a large number of queues but it has the drawback that it does not provide end-to-end delay guarantees as precise as other scheduling schemes. On the other hand, a more accurate algorithm, called Worst-case Fair Weighted Fair Queuing (WF2Q+) can achieve better results in terms of quality (lower delay, and end-to-end delay guarantees) but it does not scale well for large number of queues [5].

The analysis of the Internet traffic in [9] shows how the number of queues, the average packet size and the type of applications with different requirements (jitter, latency, and bandwidth) changes over time. One of the figures shows a detailed traffic analysis of how the number of flows change over 24-hour and 7-day time range. Using dynamically reconfigurable systems in the network devices, we can match the configuration of the network device to the network traffic (in this case to the number of active queues). In this work, we have chosen to use the DWRR algorithm when the number of active queues is large. In the case, that the number of active queues is smaller, the more accurate but more complex in terms of hardware resources WF2Q+ algorithm is used. Hence, the dynamic reconfiguration of the queue scheduler in a network device can be used to achieve better QoS requirements depending on the number of active queues with negligible performance degradation in terms of bandwidth allocation. The main contributions of this paper are:

- The efficient mapping of two widely used packet schedulers (DWRR, WF2Q+) in reconfigurable logic
- The implementation of the scheduler into a partially dynamic reconfigurable scheme
- The performance evaluation of a system that can use two queue scheduler (taking into account the reconfiguration overhead) depending on the number of active queues

This paper is organized in the following way. Section 2 presents the design and the implementation of the dynamically reconfigurable scheduler. Section 3 presents the performance evaluation and the results in terms of QoS for the proposed scheme. Finally, Section 4 presents the conclusions of the proposed scheme.

2. DESIGN ORGANIZATION

The block diagram of the system is shown in Fig. 1. The system consists of a Queue Table Controller, a Queue Table that holds the queue information and a Queue

*This work was supported by Sandbridge Technologies, Inc.

Scheduler that can be partially reconfigured either as a DWRR scheduler or as a WF2Q+ scheduler. Furthermore, a FIFO is used that contains the QueueID of the active queues and it is used by the DWRR scheme. The scheduler can be either the DWRR or the WF2Q+ depending on the number of active queues. The Queue Table Controller receives the information from a Queue engine such as the Queue Manager Reference Design form Xilinx [6]. This Queue Manager can support up to 64K queues and eight traffic classes. The current implementation of the dynamic queue scheduler is targeting mainly network access devices and not high-performance core routers (that usually supports thousands of queues) and it is designed to exploit the efficiency of dynamic reconfiguration in the schedulers, hence it can support up to 2K queues (the QueueID is 11 bits). The Queue Table (a 2-port 2Kx64bits RAM) holds the information for each queue: the size of the packet in the head of the queue (16 bits), the number of packets there are in each queue (16 bits), the DWRR weight (16 bits), and the WF2Q+ weight (16 bits). Using 16-bits weight we can support up to 64K distinct weights, enough for most of the applications. When the queue scheduler selects a queue, the QueueID is sent to the Queue Table Controller which updates the Queue Table. In the case of the DWRR the scheduler reads the next active queue from the FIFO. If the queue is eligible then the Table Controller checks if it is the last one. Otherwise, it writes the queue back to the FIFO of the active queues.

2.1 Deficit Weighted Round Robin Scheduling

The Deficit Weighted Round Robin (DWRR) was proposed by M. Shreedhar and G. Vargeshe in 1995 [7] and it was initially designed to address the limitations of the Weighted Round Robin and the Weighted Fair Queuing models. In DWRR queuing, each queue is configured with two parameters:

- A weight that defines the percentage of the output port bandwidth allocated to the queue
- A Deficit Counter that specifies the total number of bytes that the queue is permitted to transmit each time that it is visited by the scheduler.

Hence, the algorithm of the system is rather easy to implement. Each time a queue is visited by the scheduler the deficit counter of the corresponding queue is accumulated with the weight of the specific queue. If the new deficit counter is bigger than the size of the packet at the head of the queue then the packet is transmitted and the size of the packet is subtracted from the deficit counter. Otherwise, the scheduler visits the next active queue (active queue is a queue that has at least one packet to send). The implementation of this scheduler is straight forward. It only needs an adder and a comparator and a table that holds the deficit counter for each queue (in our case a 2Kx16bits RAM). The drawback of this algorithm is

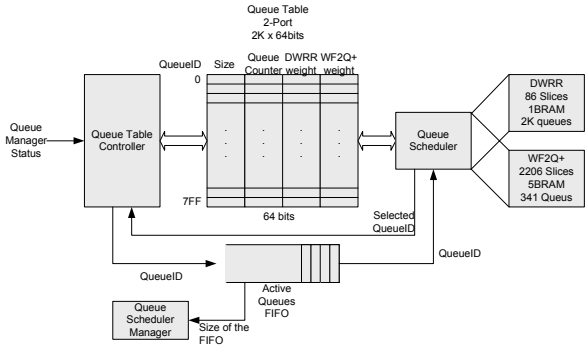

Fig. 1. Overview of the system

that many queues can be visited that are not eligible to be transmitted if the counter is less than the size of the packet. Hence, the time it takes for the scheduler to decide which queue will be transmitted is unknown when the weight of the queue is smaller than the size of the packets. To overcome, this drawback, a number of queues can be examined at the same time and then use a priority encoder to decide which queue will be transmitted.

2.2 Worst-case Fair Weighted Fair Queuing+ Scheduling

The Worst-case Fair Weighted Fair Queuing+ algorithm was proposed by J. Bennett and H Zhang in 1996 [4]. The Weighted Fair Scheduling algorithm tries to allocate services more accurately than the other algorithms and it is closer to the ideal case of Generalized Processor Sharing (GPS). In the case of the GPS all the queues are sliced into bits and each queue is allocated exactly the corresponding bandwidth. Since, the slicing of packets into bits can not been implemented efficiently, the WFQ algorithm assigns a finish time, to each packet, equal to the virtual time that the packet should be finished transmitted if the algorithm was the GPS. In this case, the finish time of each packet arriving at the Queue Manager should be stored. Instead, in the Worst-case Fair Weighted Fair Queuing+ algorithm, the scheduler assigns a start time and a finish time only to the packet that it is in the head of the queue. Hence, for each queue the following parameters are used:

- The weight of the queue
- The start time of the head packet of the queue
- The finish time of the head packet of the queue
- The virtual time

The WF2Q+ algorithm works in the following way. Each time a new packet is assigned as a head packet of a queue, the start time of this packet is configured to be either the current virtual time if the scheduler is idle or the time at which the packet that is being sent will be finished. The finish time of the packet is assigned as the start time plus the time to send this packet using the weight of this queue, as it is shown in the following equations (Eq. 1, Eq.2).

$$queue(i).start\_t = \begin{cases} virtual\_t, scheduler = idle \\ finish\_t, scheduler \neq idle \end{cases} \quad (1)$$

$$queue(i).finish\_t = queue(i).start\_t + \frac{size(i)}{queue(i).weight} \quad (2)$$

$$virtual\_time(t) = \max\{virtual\_time(t+\tau), \min\{start\_t\}\} \quad (3)$$

The scheduler selects the queue with the minimum finish time and then updates the virtual time using either the minimum start time of the queues or the current virtual time plus the time to transmit the selected packet as it is shown in Eq.3. The main advantage of this scheme is that the system operates without a priori knowledge of the traffic or assumptions about the packet size. The main disadvantage is the required computations and the state information that should be stored in a memory [5]. The algorithm requires finding the minimum start and finishing time for the active queues. If the number of active queues is large then the minimum functions can slow down the performance of the scheduler if we examine each queue separately (O(N)). On the other hand, if we use a tree structure the area in hardware would be too big. Hence, we use a heap sort scheme, which is a more efficient implementation, to find the minimum start and finish time from a number of active queues.

The heap sort scheme that is has been designed can compare 4 different values at the same time. To map efficiently the heap sort algorithm in the current design the following scheme was used. A 128x128bits RAM is used to store 4 words as it is shown in Fig. 3. Each word contains the QueueID (16 bits) and the finish time (16 bits). The minimum finish time is stored at address 0x0. At address 0x1 is stored the next level of the heap sort (4 values), at addresses 0x2, 0x3, 0x4 and 0x5 are stored the 3rd level, etc. Using this scheme we need 1 cycle to find the queue with the minimum finish time and 5 clock cycles to update the heap sort table.

Each time a new entry has to be inserted into the heap memory, the scheme depicted in Fig.2 is used. The new finish time is inserted into the 16-bits "level register" and the memory data from the same level is loaded. If any of the four entries is empty ('E' bit in the scheme) then this space is selected to store the new data. Otherwise, it compares the value of the new entry with the other 4 finish times and if any of them is smaller (and the lower entries are not full; 'F' in the scheme) then the new entry is inserted into this space and the previous one is forwarded to the next-level register. If the new entry is bigger than all of the values in the memory then the new entry is forwarded to the next-level register. The current design can support up to 341 queues (1+4+16+64+256 =341 elements; each row of the table holds 4 elements). A similar heap sort has been used to sort the start time of the queues. But in this case, the minimum start time does not have to be extracted from the table.

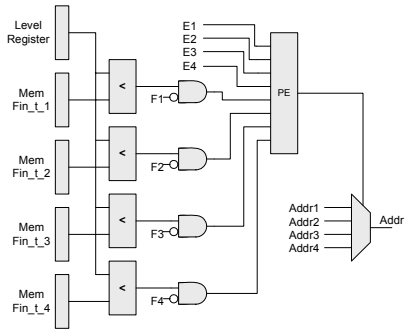

Fig. 2. The WF2Q+ address generator

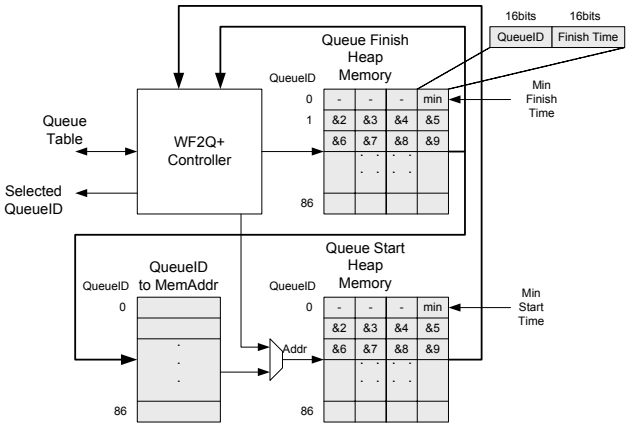

Fig. 3. The WF2Q+ scheduler

Table 1. Resource allocation

Unit	Slices	BRAMs
Table Controller	127	8
DWRR	86	1
WF2Q+	2206	5

The scheduler only reads the minimum start time but it does not extract it from the table. Each time a queue is selected from the Finish Time Table, the corresponding Start time entry must be deleted from the Start time table. Hence, a small Look-Up Table is used to map the QueueID to the address of the Start time Table as it is shown in Fig. 3. Thus, the time to update the Start Time Table takes 1-5 clock cycles.

The current design has been implemented into a Xilinx Virtex-II Pro XC2VP30 Platform. The design methodology for partially dynamically reconfigurable designs was used to floorplan the system [8]. The system was partitioned into a static and a reconfigurable area. The signals that are used for the communication of the static and the reconfigurable area have to be placed using specific tri-state buffers (bus macros). The signals that are used in this design are the signals for the second port of the Queue Table, the signals to read the active queue FIFO and the selected QueueID (12 bits). Table 1 illustrates the resources that each unit occupies.

3. PERFORMANCE EVALUATION

The main drawback of dynamic reconfiguration is the timing overhead, because during the reconfiguration the part of the device that is being reconfigured can not be used. This overhead would cause too much delay in the packets waiting to be transmitted. In addition, the implementation of the schedulers shows that the DWRR occupies much less area than the WF2Q+ scheduler. Hence, when the DWRR scheme is selected, the remaining area would be empty. Thus, in our design the DWRR scheme is used in the static area, while the WF2Q+ scheme is placed in the reconfigurable area as a page-able IP module. When the DWRR is used this spare area can be used for other IP modules such as encryption or compression units used in common network processing applications. In this case we examine the overhead of the dynamic reconfiguration in the allocation of the bandwidth, during the transfer of the scheduler from DWRR to WF2Q+ and vice versa. In the first case, when the system switch to the WF2Q+ algorithm then the scheduler configures all the active queues as if they have just arrived in the system (virtual time = 0, start time=0, finish time = start time + size/weight). In the second case, the DWRR counter for all the active queues is set to zero.

Fig. 4 shows the impact of the dynamic reconfiguration overhead in the allocation of the bandwidth for the DWRR and the WF2Q+ scheduler for several active queues (32-256 for the WF2Q+ and 100-1000 for the DWRR) using uniform distributed random packet sizes. The figure shows the difference in the band-width allocation between the ideal allocation (GPS) and the real allocation for several thresholds of reconfiguration. In the case of the DWRR scheduler, when the dynamic reconfiguration takes place after 1000 or 2000 packets then the error in bandwidth allocation varies from 0.02% to 0.06%. If the network is more stable, e.g. the number of active queues does not change very often, the error in bandwidth allocation becomes almost negligible. For example, if the number of active queues changes at least every 5000 packets then the error in bandwidth allocation is almost 0.01% which is almost the same to the static version in which a DWRR scheduler is always used.

In the case of the WF2Q+ scheduler, the impact of the reconfiguration is also negligible (0.01% to 0.04%) in the bandwidth allocation, as it is shown in Fig. 4. This scheme also shows that both of the scheduler can allocate quite accurately the bandwidth, since the error in bandwidth allocation for the static version is almost 0.01% compared to the ideal case, although in terms of latency the WF2Q+ outperforms the DWRR according to [1]. It is obvious that a network topology in which the number of active queues changes more rarely (more than every 10000 packets, which is a logical assumption according to [9]) then the impact of the reconfiguration would be infinitesimal.

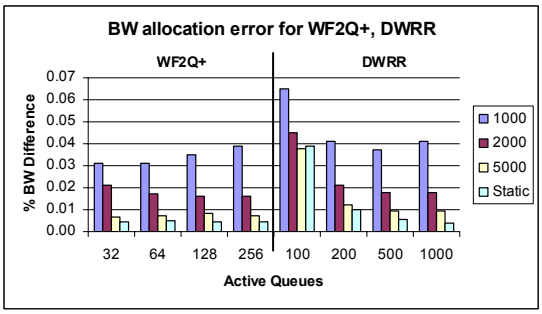

Fig. 4. Bandwidth allocation error

4. CONLUSIONS

In this paper we investigate the use of dynamic reconfiguration to adapt the queue scheduler of a network processing unit to the requirements of the active queues. As it is shown, the overhead of the reconfiguration causes almost no performance degradation in terms of bandwidth allocation, while the use of different schedulers has a major impact on the delay and the jitter for several applications [1, 2]. The timing overhead of the dynamic reconfiguration can be overcome using the DWRR as a static scheduler, hence the transfer of the scheduler from the DWRR to the WF2Q+ scheduler can be achieved in only one clock cycle, when the reconfiguration of the module will be over. This paper shows that the dynamic reconfiguration of the current FPGAs can improve the performance of the network devices in terms of scheduling accuracy when they are used efficiently by trying to adapt to the varied number of active queues.

REFERENCES

[1] C. Semeria, “Supporting Differentiated Service Classes: Queue Scheduling Disciplines, White Paper, Juniper Networks, Inc. Dec. 2001

[2] “Congestion Management Overview”, White Paper, Cisco, Inc.

[3] “Bringing Comprehensive Quality of Service Capabilities to Next-Generation Networks”, White Paper, Freescale Semiconductors, Inc.

[4] J.C.R. Bennet, H.Zhang, “Hierarchical Packet Fair Queuing Algorithms”, in *Proc. ACM SIGCOMM’96*, pp. 675-689, August 1996

[5] D. Comer, “Network System Design using Network Processors”, Pearson Prentice Hall, 2004

[6] H. Fallside, “Queue Manager Reference Design”, Application Note 511, Xilinx Inc., March 2004

[7] M. Shreedhar, G. Varghese, “Efficient Fair Queuing using Deficit Round Robin”, in *Proc. ACM SIGCOMM’95*, Vol. 25, No. 4, pp.231-242, October 1995

[8] “Two flows for partial reconfiguration: Module based or Difference Based”, Application Note, Xilinx Inc., September 2004

[9] K. Thompson, G. Miller, R. Wilder, “Wide-Area Internet Traffic Patterns and Characteristics”, *IEEE Network*, vol. 11, no.6, Nov.-Dec. 1997