Electrical and Electronic Engineering 2012, 2(2): 68-77
DOI: 10.5923/j.eee.20120202.13

Design of a Qos-Based Reconfigurable Priority Active Queue Management for IP Networks

Hattab Guesmi[1,2,*], Rached Tourki[1]

[1]Electronic and microelectronic laboratory, Monastir university, Monastir, Tunisia
[2]Faculty of sciences, Jazan university, Jazan, KSA

Abstract This paper presents a reconfigurable and scalable architecture of a high-performance IP switch to improve network quality of service (QoS). Quality of services, in terms of delay, through-put and loss rate, can be provided by using a mechanism support like scheduling and buffer management architecture of packet switching IP networks. The proposed architecture consists of a new memory management data structure based on circular linked lists. The linked lists include different priorities levels with a pipelined organization for the reconfigurable priority active queues management. The architecture also scales dynamically to support a large number of priority levels and a large queue size. The new data structure enables us to configure the architecture based on network service domain. Detailed description of new data structures of the proposed algorithms and their corresponding implementations are presented as well.

Keywords Diffserv, IP Switch, Qos, Psoc, CBWFQ, AQM, WFQ, Reconfigurable Architecture, RPAQM

1. Introduction

The future growth of the Internet requires design and development of high-speed IP switch that forward exponentially increasing volume of traffic and provide QoS guarantees at the same time. In this respect, Quality of service (QoS) guarantees in term of delay, throughput, and loss rate can be provided by using a service discipline at switching nodes in packet switching networks. This paper deals with the important class service disciplines used for output queued switches like the Class Based Weighted Fair Queuing and Weighted Fair Priority Queuing Techniques. In these service disciplines, packets are assigned transmission deadlines (or priority indices), and the packet with the earliest deadline (or highest priority) is served at first. This approach requires a priority queue manager for sorting priority levels and managing queues. Packets from various connections are interleavingly served so that each connection's QoS requirements can be guaranteed by using services disciplines and priority queues. In order to achieve QoS guarantees in practice, the priority queue manager must be able to support a large buffer size in high-speed networks.

The design of an AQM that implements the deadline-ordered service disciplines with a large number of priority levels, a large buffer size, and a large number of input links is not easy to achieve for high-speed packet switching networks. In fact, stored-priority disciplines must always choose the packet with smallest deadline or highest priority among all prioritized packets in the queue to be served at first. If the number of stored packets is M, then the complexity is $O(\log M)$. The bottleneck comes from the high cost of maintaining a high-speed sorting mechanism associated with implementing these service disciplines. Moreover, to improve the network performance, a large number of input links to an AQM is necessary. However, many proposed priority queue designs have limitations in term of priority levels number, buffer size, and input links number or bandwidth are introduced in the literature[1, 2, 3]. In our approach, we propose priority AQM architecture for deadline-ordered service disciplines that can be used in output-buffered switches to provide QoS guarantees in high-speed packet switching networks. In this respect, we have to use an insert mechanism in a sorted priority list to maintain the highest priority cell in the top instead of commonly used shift, compare and RAM-based search mechanisms. This paper, also, presents a novel, highly-scalable architecture for a PAQM. Section two of this paper introduces quality of services basics and mechanisms that support the QoS. High-performance, QoS-capable data structure of the PQM that is the priority circular linked list is represented in section three. Then, the implementation of the component supporting the QoS management is presented in Section four. In section five, design results are presented. Then we present the reconfiguration of the architecture to be used in different networks domain and finally, the design results related to our proposed architecture is presented with some important conclusions and remarks.

* Corresponding author:
Hattab.guesmi@fsm.rnu.tn (Hattab Guesmi)

2. Quality of Service Mechanism'S Support

QoS (Quality of Service) is a hot topic in both academic and industrial fields for many years. QoS means a series of service requirements that networks should satisfy while delivering data. It can be represented by: delay, delay jitter, loss rate, bandwidth, etc[1]. QoS control is to provide consistent, predictable and controllable data delivery service, and to satisfy different application requirements. In fact, the application of the QoS philosophy has to guarantee different received packets related to different level of services.

Table 1. QoS parameters and mechanisms

QoS parameters	QoS mechanisms	Guaranteed	Statistical	Best effort
Loss	Buffer management	Fixed buffer allocation based on the peak rate	Shared buffers based on average rate	No guaranteed buffer allocation
Throughput	Regulation (flow shaping)	Eligibility time based on a peak rate	Eligibility time based on average rate	No regulation resources committed
Delay and jitter	Scheduling	Flow always scheduled at eligibility time	Flow scheduled at eligibility time resources permitting	Flow scheduled if scheduler idle

There're many mechanisms to support QoS, such as resource reservation (RSVP), admission control in Integrated Services (IntServ) and traffic shaping/marking in Differentiated Services (DiffServ). The major difference between IntServ and Diffserv architecture consists on the granularity of service differentiation. The IntServ concept lies in resource reservation where each application requests service level in term of service rate or end-to-end delay. Consequently, the network has to accept or reject requests according to resources availability. However, the IntServ approach faces potential problems concerning scalability and manageability, since all routers must maintain per-flow state. The main strength of DiffServ, as proposed by the IETF Differentiated Services Working Group[4, 5], is that it allows IP traffic to be classified into a finite number of service classes that receive different routing treatments. Routers at the network edges classify packets into predefined service classes based on requirements and characteristics of associated application. Core routers forward each packet according to its class service. DiffServ model provides service differentiation on each node (Per-Hop behaviors) for large aggregates of network traffic. DiffServ achieves scalability and manageability by providing QoS to aggregate traffic (not for each application flow). While the common and key ones are buffer management and packet scheduling, called interestedly "Active Queue Management". Buffer management determines how to allocate buffers and whether to drop an arriving packet according to certain policy, which mainly influences the loss rate and fairness. In this respect, packet scheduling adapts politic transmission of stored packets. It serves to control the resource distribution between classes of service. This operation is carried out with flow isolation and priority levels assignment to find and send highest priority packets first. In fact, packet scheduling mainly influences the bandwidth, delay/jitter, and fairness. There has been a great amount of research work on packet scheduling in the past years and many algorithms appeared. The key ideas of most packet scheduling algorithms consist in computing an index for each queue and sort them. The scheduling decision is performed by selecting the queue with the minimum or maximum value of these indexes. WRR and DRR are kinds of round robin strategy which are easy to implement, but have weakness in providing delay guarantee. EDF (Earliest Deadline First) and its variants are based on queuing delay. Their key ideas consist on allocating a delay parameter D_i to each queue as the delay up bound, where each arrived packet is tagged with the time stamp $T_i = A_i + D_i$: A_i represents the arrival time. Consequently, every time the packet with the minimum T_i has to be scheduled. In the other way, another category of algorithms called PFQ (Packet Fair Queuing) are based on service rate. Their key ideas lies in maintaining the virtual system time Vi(t), the virtual start time Si(t) and the virtual finish time Fi(t) for each queue. Si(t) or Fi(t) is computed and the queue with its maximum or minimum value is scheduled. One weakness of the PFQ algorithms consists on coupling the service rate (bandwidth) and the delay that results in the inflexible resource allocation. Floyd and Jacobson have proposed a link-sharing and resource allocation scheme called class-based queuing (CBQ) which employs DRR queuing algorithm and differentiate flows into different queue classes. Each queue is serviced in round-robin fashion and receives bandwidth equal to its allocated share. However, the research work on buffer management and packet scheduling are mostly separated where they consider only one or some performance metrics which is insufficient[6-8]. Since buffer management allows the manipulation of en-queuing and packet scheduling concerns the manipulation of de-queuing, which have a tight relationship. Consequently, both buffer management and packet scheduling mechanisms (which means the active queue management) have many effects on almost of the performance metrics are unable to meet the QoS requirement of today's applications.

The table 1 summarizes the characteristics in terms of loss, throughput, delay and jitter related to each traffic control strategy. We remark that the guaranteed service requires a specific treatment in order to satisfy the required service compared to the best effort one. Where the QoS mechanisms that support QoS parameters is the active queue management which consist of buffer management, traffic regulation and scheduling the more interesting one in term of performance but needs an important effort to be implemented.

3. Data Structure of the Priority Circular Linked List (P-CLL)

3.1. Description of the Active Queue Manger

The active queue management requires a specific organized data structure related to the selcted scheduling algorithm, since it presents its data base. The data structure must be structured in a way that all stored packets are visible and accessible so that the decision have to be fast and effective according to the QoS required for each flow. Thus, basic operations like creation, access and release of a service flow are achieved in a dynamic way according to the number of declared flows and the free memory spaces. Consequently, the memory spaces have to be adjustable according to the dynamic parameters of the loss management algorithm. The number of flows per class is not fixed statically, but can be managed dynamically according to the allocated memory. To meet these needs, we have used the priority circular linked list which ensures the dynamic queues management and the reconfiguration of the architecture parameteres. The architecture of the active queue manager consists primarily on a priority assignement unit, a P-CLL manager and a queue controller as depicted in the Figure 1. In this respect, the priority asignement unit stamps the incoming packets with a certain priority value, which is decided depending upon the scheduling algorithm in use. The P-CLL manager responsible for the queue managmenet contains the priority circular linked list, where each elements in the P-CLL represents an active priority level. The queue controller maintains a lookup matrix with entries corresponding to each priority level. Each priority level gather all packets have the same priority. We refer to this list as a priority list. Thus, our proposed structure is one of per-priority queuing rather than per-flow queuing, and is more general in the sense that it can handle many priorities of the same flow. It should be noted that at any time, the P-CLL contains only the active priority levels for which the priority list is non-empty[9, 1].

Generally, the RPAQM maintains a logical queue for each flow or session in the data memory. Each queue can be implemented in a linked list with head and tail pointers pointing to its head of line (HOL) and tail of line (TOL) packets. An idle queue may also be needed to maintain the idle priority levels. When a packet arrives at the system, it is stored in the corresponding queue. The scheduler queue prioritizes all HOL packets, or all eligible HOL packets if a shaper-scheduler is implemented, based on their finish times. It then chooses the packet with the smallest finish time to transmit first. This requires fast sorting or searching operations and it is one of the challenges in designing a packet scheduler. In general, all the HOL packets are first stored in the shaper queue. Only those that are currently eligible can be moved to the scheduler queue. Some efficient mechanism is needed to compare the system virtual time with the start times of packets in the shaper queue (i.e., performing the eligibility test) and then move eligible packets to the scheduler queue. In the worst case, there may be a maximum of packets that become eligible. Suppose the scheduler queue selects the HOL packet of queue; it determines the head pointer associated with queue and then reads out the packet using the head pointer. There are more design issues, such as handling time-stamp overflow and time-stamp aging problems.

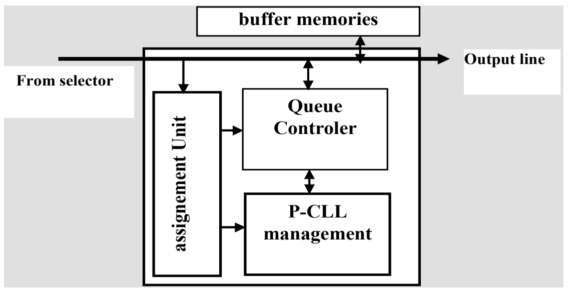

Figure 1. the output queue manager architecture

3.2. Data Structure of the Active Queue Management

In order to increase the design performance of the per-priority queuing, we have retained the priority circular linked list combined with a dedicated data structure which implements the reconfigurable priority active queue manager. The P-CLL contains only active priorities levels. In this respect, data related to each class of service is stored in a per-connection data queues, so that each data queue represents a cell in the priority circular linked list, can be served separately.

When a new packet needs to be inserted into the queue, the priority assignment unit stamps the packet with a suitable priority value. The queue controller determines whether a priority list already exists for the stamped priority value. If it does, it simply adds the new packets to the corresponding priority list. However, if the list does not exist, the queue controller creates a new priority list. It also signals the P-CLL manager to perform an en-queue operation, which inserts the new priority value into the P-CLL in a sorted manner. This operation is done to make sure that the highest priority stays at the top of the P-CLL so that when a de-queuing of a packet is required, the priority list with the highest priority can be rightly accessed. In the other hand, when a packet needs to be removed from the queue, the P-CLL manager determines the non-empty priority list with the highest priority by looking at the topmost element of the P-CLL and sends this priority value to the queue controller. The queue controller accesses the corresponding priority list and removes a single packet from it. When the priority list become empty, the P-CLL manager initiates a de-queue operation which removes the topmost element from the P-CLL while making sure that the P-CLL remains sorted.

The P-CLL data structure maintains various priority lists sorted in the circular linked list. Each en-queue operation requires N times to be serviced, where N is the size of the P-CLL. However, the emulation of the circular linked list which implements the priority queuing mechanism is easy to make and it provides a big save in term of time constraints. Furthermore, this structure allows a pipelined implementation compared with the conventional circular linked list and provides a constant time operations. In order to improve the performance of our P-CLL, we implement the method of insert based on a content addressable memory where each priority queuing operation is performed in only one time.

3.3. The P-CLL Based Priority Queuing Solution

Using a conventional circular linked list, queuing operations require O(n) steps, where n is the number of elements in the circular linked list; (these cannot be easily pipelined). On the other hand, architectures such as the systolic array and binary heap architectures can be pipelined[6, 10] but have extremely high hardware requirements. To support the pipeline organization for queuing operations using a circular linked list, a specific data structure is required. This will be able to offer constant priority queuing operations with a low hardware cost. Two main operations have been defined regarding the en-queuing and de-queuing operations.

● **The en-queuing operation:** To en-queue a new value of priority in the circular linked list of the P-CLL, we need to find a free cell. This operation is achieved by exploring the valid data path from the matrix T values which allows us to find the inserted cell address pointer. Then, we carry out the updated list to be added to the P-CLL in a sorted manner. The insertion position is identified by finding the first least weak priority level of non-empty priority list coming just after this level. Seeking the matrix T from the inserted value address and find the first case contain '1' coming after this address corresponding to cell which has the least weak priority. The address of priority level founded represents the insertion position of the priority level. The addition operation consists on reading and writing the bonding pointers of the two cells (inserted cell and founded cell) and updating the class descriptor service parameters. The operation of research requires N time, where N is the priority levels number. However, in the P-CLL, the research operation needs only the conversion time of the priority level into address to fetch the first non empty cell. We can summarize the required steps for inserting a new priority list as the following:

Step1:
Request an available free cell
Read the pointer at the matrix address
Step2:
Stamp the packet with a priority value
Update the flag in the matrix
Step3:
Determine the insertion position
Insert the priority list

● **The de-queuing operation:** The de-queuing operation consists of extracting the cell pointed by the header pointer from the P-CLL since it has the highest priority value. Then, modify the case in the matrix representing the priority level (flag) to '0' making the priority level inactive and update the class descriptor service parameters. The de-queue operation are summarized below:

Step1:
Read the head pointer of the circular linked list
Add the free cell to the free cell list
Step2:
Update the circular linked list
Update the status flag
Update the file capacity

4. Related Works

Extensive research has been done on the next generation of high-speed routers. Nick Mckeown's group at Stanford University did intensive research on high speed switching[11, 3, 12]. R. Bhagwan proposed a Design of a high-speed packet switch for fine-grained quality of service guarantees and a fast and scalable priority queue architecture for high-speed networks switches[13]. H. Jonathan et al. have proposed a design for packet-fair queuing schedulers using a RAM-based searching engine[14]. Many other papers and proposals are dealing with some key issues in real-time packet scheduling, fine grain QoS control, high-speed switches for the data path and so on[9, 2]. We present two techniques which implements data structure of the AQM such as:

● **The calendar queue** uses the search-based approach to reduce implementation complexity. In the search-based approach, time stamps are quantized into integers and are used as the address for the priority queue. Each memory entity may contain a validity bit and two pointers pointing to the head and tail of an associated linked list called the timing queue which links the indexes, such as, of each session, where the time stamps of the HOL packets are the same. Therefore, all the HOL packets are pre-sorted when their corresponding session indexes are stored in the calendar queue. Finding the next packet with the minimum time stamp is equivalent to finding the nonempty timing queue with the smallest address. In the search-based approach, the time complexity of sorting time stamps is traded with space complexity, which is determined by the maximum value of the time stamp, say for instance.

● **The Hierarchical Searching** (RSE) reorganizes and stores all the bits in the calendar queue. Its main function is to find a nonempty timing queue that has the smallest finish time (address) and to output its address, which is the output of the RSE (read operations). It is used to fetch the queue index which in turn locates the pointer pointing to queue's HOL packet.

In a calendar queue, a validity-bit is associated with each timing queue, indicating whether this queue is empty or not. Since packets are automatically sorted based on their corresponding locations in the calendar queue, finding the next packet to be transmitted is equivalent to searching the first bit in the calendar queue. The key concept of hierarchical searching is extended and generalized from the one in the

PCAM chip by dividing the total validity-bits in the basic searching into multiple groups, which forms a tree data structure, where is the maximum value of time stamp. Each group consists of a number of bits, so another bit string can be constructed at the upper level with its length equal to the number of groups at the bottom level. Each bit at the upper level represents a group at the bottom level with its value equal to the logical OR of all the bits in the group. Further grouping can be performed recursively until the new string can be placed in a register. Suppose levels are formed from the original bit string. There are bits at level, and each of its groups has bits.

The services disciplines implemented in our architecture such as CBWFQ and WFPQ also needs to maintain another priority queue called shaper queue is implemented as a multitude of priority lists. Each list is associated with a distinct value of start time common to all queued packets in this list. Using the search-based approach, we can construct a 2-D calendar queue based on the start times of the queued packets, where the start time and time stamp are used as the column and row addresses, respectively, and is the maximum value of the start times. All packets with the same start time are placed in the same column addressed by and also are sorted according to their time stamps. Hence, each column represents a priority list. Each bit in a column can be located by its unique address. Performing the eligibility test is equivalent to using the system virtual time as the column address to find the nonempty column(s), with their addresses ranging from the previous value until the current value of the system virtual time, and then moving packets in these column(s) to the scheduler queue.

5. Implementation of the RPAQM

All recently proposed packet-scheduling algorithms for output-buffered switches that support quality of services (QoS) transmit packets in some priority order, e, g, according to dead-lines, virtual finishing times, eligibility times, or other time stamps that are associated with a packet[5, 10, 15]. Since maintaining a sorted priority queue introduces significant overhead, much emphasis on QoS scheduler design is put on method to simplify the task of maintaining a priority queue. The two main metrics for measuring the performances of a scheduling algorithm: throughput and delay. In our architecture we use fast and scalable pipelined priority queue architecture for use in high-performance switches with support for fine-grained quality of services guarantees. Each output port is maintained using an output port manager. The output port manager implements mechanisms that support the QoS such as buffer management and scheduling which are called reconfigurable priority active queue management (RPAQM). The architecture implementation includes the following blocs: the priority assignment Unit, the P-CLL manager Unit, the queue controller Unit and the controller[4, 16].

5.1. Priority assignment module (PA)

This module stamps packets by a label calculated according to the implemented scheduling algorithm. This label determines the transmission order of the data packets to the output line (Figure 2). This module needs to know the three following factors:

- The weight allotted to the concerned class of service: the concept of weight determines the band-width percentage that the class is seen allotting.
- The length of the packet;
- The interaction with the other concurrent active classes.

In order to stamps packets by labels, this unit calculates the virtual finish time and the virtual start time according to the scheduling algorithm (CBWFQ). The calculated value related to each assigned packet level represents the transmission order (packets priority). According to these values, packets are stored in suitable queues. Since our architecture implements N_{cs} classes of services, the calculation of priority values needs a dynamic model uses a data base in order to describe the state of each service class and each stored packet. This data base is constructed in the P-CLL manager module to be consulted and updated by this unit after any operation (emission or reception) in order to provide the correct priority values. The time stamp of a packet is the sum of its virtual start time and the time needed to transmit this packet at its reserved bandwidth. Packets are served by an increasing order of their time stamps. The implementation cost of a CBWFQ algorithm is determined by two components: computing the system virtual time function and maintaining the relative ordering of the packets via their time stamps in a priority queue mechanism.

A Priority Assignment server has N queues to store all arrived flows. Each queue i has a minimum bandwidth allocation, denoted by r_i, where i = 0, 1, 2, …, N-1. During any time interval when there are exactly n (n inferior or equal to N) non-empty queues, the server serves the n HOL packets corresponding to the non-empty queues simultaneously. To approximate the priority level (label), CBWFQ algorithm maintains a system virtual time function V(t), a virtual start time Si(t), and a virtual finish time Fi(t) for each queue i. Si(t) and Fi(t) are updated on arrival of the HOL packet for each queue. A packet departure occurs when its last bit is sent out, while an HOL packet arrival occurs in either of two cases: (1) a previously empty queue has an incoming packet that immediately becomes the HOL or (2) the packet next to the previous HOL packet in a nonempty queue immediately becomes the HOL after its predecessor departs. Obviously, a packet departure and a packet arrival in Case 2 could happen at the same time. Therefore

$$S_i(t)=\begin{cases}\max\left\{V(t),F_i(t^-)\right\}, & \text{for packet arrival in case } 1\\ F_i(t^-), & \text{for packet arrival in case } 2\end{cases}$$

$$F_i(t)=S_i(t)+\frac{L_i^k}{r_i}$$

Where Fi(t) is the finish time of queue before the update and L_i^k is the length of the HOL packet for queue. The way of determining V(t) is the major distinction among proposed WFQ algorithms[1, 17].

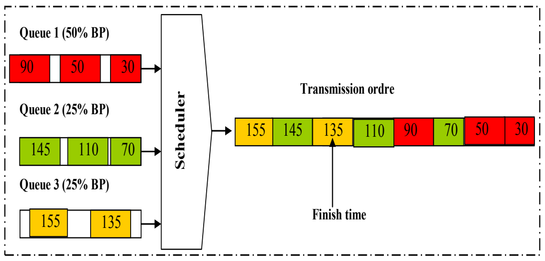

Figure 2: Stored stamped packets in a sorted manner with CBWFQ algorithm

5.2. P-CLL Manager Module (PM)

This module implements the selected data structure which is based on the priority circular linked list P-CLL. It consists of a circular linked list based on the priority levels. Each cell of these descriptors of the priority levels has an active priority level which gathers all packets having the same priority level in a circular linked list which is called packet list (Figure 3). Every cell related to the packet list contains two fields: a bound pointer to its packet list and an address pointer to the memory space which stores the packet. The cell of the priority list contains several fields which are: a head and a tail pointers associated to the priority lists (data packets having the same priority value), a bound pointer to its class list and an additional field which describes the priority level value. In the other hand, the service class descriptor cell contains the associated control parameters related to each class such as the queue average size, the probability of reject etc...

In this respect, our design of the P-CLL manager unit uses two separated memories. The first one contains the class of service descriptors where the second it contains the priority values list. Thus, the first memory is organized as Ncs blocs (Ncs: number of service classes), where each bloc has Npc fields (Npc: number of control parameters of each service class). The second memory which implements the priority linked list levels is organized as Npv * 4 fields (Npv: number of priority values; one field for the priority value, one for the bound pointer and two fields for the head and tail pointer). The variation of the memory size depends only on the number of service classes for the first memory and the number of the priority levels belonging to the second memory. The size of these memories varies in a linear way according to the increase of the number of the priority levels managed by this architecture and the services classes' number.

We used two separated memories in our design for this reasons: Firstly, our architecture will be scalable by increasing the memory size with a simple adding of space memory to increase the number of priority levels related the number of service class. Second, the number of priority values belonging to each service class is reserved in a dynamic way according to the need for each class of service (number of flows treated in each service class): allowance on request or dynamic adjustment of the band-width between the classes of service. Then, the architecture will be reconfigurable according to the network domain. And finally, The en-queue and de-queue operations of a priority level are simple and rapid because the priority level represents its address in the memory, so the access to every priority level needs only the time to convert the priority level to an address[15, 18].

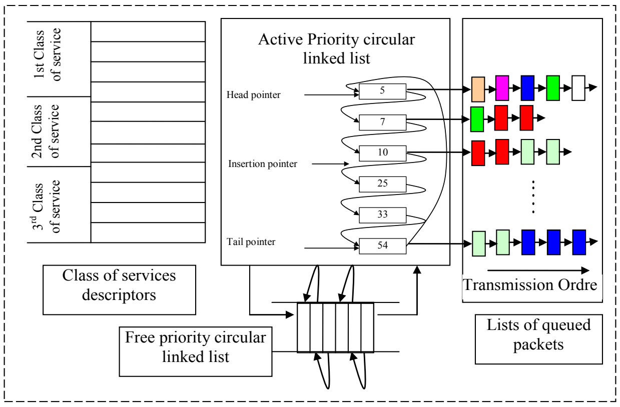

Figure 3. Structure of the priority circular linked list

5.3. Queue Controller Module (QC)

This module is used to control all priority levels related to the active queue management: which can be switched between the active and the inactive state. It is represented by a NxN matrix which stores the state of all priority values. Every level is described in a case which contain “0” (inactive) or “1” (active). Every case is addressed by the priority level. When an assigned packet is received, this module seeks in the matrix if the priority value is active. If yes, so this module launches an addition operation requested by this packet with its corresponding priority list. The addition of the packet is the access to the priority level list which is addressed by the priority level to add the packet to its packet list. In the other case, this module inserts (en-queue) this priority value in its suitable place in the sorted circular linked list. During the en-queuing operation, QC makes level active (case = “1”) in the matrix and seeks the first active priority level in the increasing order. This level represents the insertion position in the P-CLL which remains the list sorted. During the emission, if this packet is the last in the packet list, therefore it launches a de-queue operation to make the level of priority inactive (case = “0”) in the matrix. The de-queue operation is achieved by the update of the P-CLL.

The design of the queue controller module uses M parallel memories (Figure 4) organized as NxN matrix (NxN= N_{pv} is the number of priority levels). In this respect, each priority level state is described by one bit (case), if the bit = “1” indicate that the priority level is active else (bit = “0”) priority level is inactive (list of queued packets is empty). The memory size of this component is equal to the priority level number; this capacity increases if the priority values number increase. It represents the priority level number treated by this architecture[6, 19].

This design facilitates the access to all priority level because the priority level represents the address in the matrix (position in line and position in column). This upgrades the throughput of our architecture because it is related with the en-queue and de-queue operations. The en-queue operation needs only few cycles because at one read cycle we can determine the state of N levels which are stored in M parallel memories. So founding the insertion position is rapid because we use M comparators to determine the first active priority level. The comparators output the results at one read cycle. After five read cycles we declare that the priority level will be inserted in the tail of the P-CLL (if the difference between the two finish times is greater than 1280 the level is posted in the tail of the P-CLL).

In our design we define the priority levels number as 65536 level, so the QC is implemented in four parallel memories (256x64 bits) to form a 256x256 matrix. The most significant byte of the priority level represents the line number and the least significant byte represents the column number in the matrix.

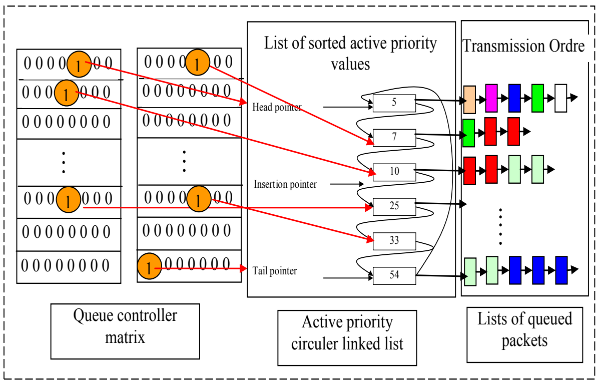

Figure 4. Structure of the queue controller module

5.4. The Controller Module

This module ensures the general reconfiguration of the IP switch to respond the need of the network administrator. This architecture will be used in several domains: DiffServ domain, IntServ domain and in very high-speed networks domain. For these reasons, our architecture is implemented in a reconfigurable FPGA due to the capacity and the speed of FPGAs computing. This implementation also enables us to reconfigure the IP switch to be used in several network domains. The controller implements the control parameters of this architecture. It uses a specific data structure related to the circular linked list organization which facilitates the reconfiguration of the general controller so that the architecture ensures the packet switching according to the network domain and class of service requirements[17].

6. Reconfiguration of the IP Switch'S Data Structure

Our proposed architecture of the reconfigurable priority active queue management[20] is designed in order to be reconfigurable according to the network's domain. This architecture integrates many sophisticated functions to guarantee the QoS (buffer management and scheduling). In each domain we need a data structure to optimize the IP switch architecture for delivering packets in required manner.

6.1. Data Structure in DiffServ (QoS optimization) Domain

The data structure which ensures the QoS guarantees in this domain is based on QoS management. This structure uses three classes of services: guaranteed service, assured service and best effort service. The data structure described previously manages N_C classes of service, so if we take $N_c = 3$, the switch has to manage only three classes of service. Each class of service inter-connects a sorted priority list pertaining to the same class. In each level of priority a list of sorted packets is stored which is depicted in figure 3. This structure makes possible to store packets in the order of highest priority first and ensures a dynamic band-width adjustment which enables us to optimize the use of the available band-width. Each class of service has a percentage of the available band-width according to its needs. CBWFQ scheduling algorithm is the best one to guarantee QoS in DiffServ domain, its implementation needs to configure the algorithm of management to uses three classes of service and to fix their percentage in band-width. Data structure described in section III is oriented to this networks domain.

6.2. Data Structure in IntServ Domain

The data structure related to the IntServ domain uses two classes of service such as guaranteed service and best effort service. Consequently, packets are treated in a different way according to these two classes indicated below. The corresponding data structure is managed by the scheduling algorithm and the queue management algorithm to send and store packets according to the implemented services. In this case, queue management algorithm is used for all possible configurations, but the Weighted Fair Priority Queuing (WFPQ) is used as the scheduling algorithm to this domain. Our basic architecture maintains N_c classes of service implemented in a circular linked list structure with N_c services class's descriptors. As a result, it is quite simple to change the context of our application to pass from the configuration of N_c service classes to the two service classes' one. Figure 5 represent the data structure of this domain interconnecting two classes of services: Best Effort class contains only one level priority and guaranteed service class gathers many priority levels. Each priority level gathers packets having the same priority. This configuration is carried out if we adjust the selection algorithm to assigns packets only to the two classes of services and assigns (0%) of bandwidth to the other classes. Then in the emission of packets, the scheduling algorithm sends highest priority packets first, where highest priority levels belong to the guaranteed service class (PQ). Packets in this class are treated in a different way according to the WFQ approach.

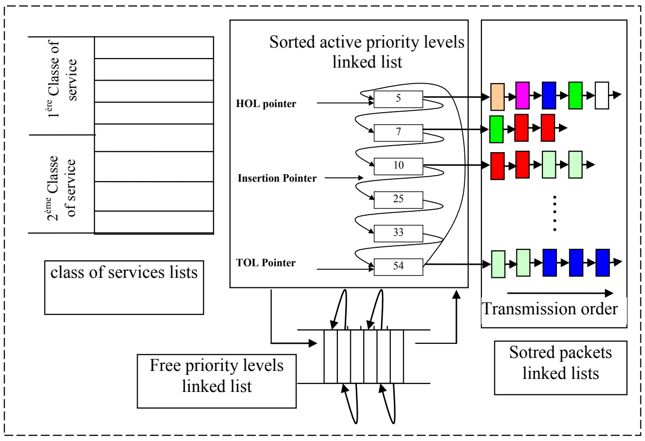

Figure 5. data structure in IntServ domain

6.3. Data Structure in very High-speed Networks Domain

The data structure in high-speed networks uses only one service class which is the best effort. Packets are treated in very high-speed since parameters of QoS are respected. Therefore, the scheduling algorithm to be implemented in this domain is based on the FIFO algorithm, where packets are treated in the order of their arrival (First In First Out). The reconfiguration of the RPAQM requires the following modifications:

● Make the storage average size of the other classes to zero (0%). Packets are stored in the Best effort class of service which has the 100% of the available bandwidth.

● Assign the same priority value to all arrived packets to be stored in the same priority level list instead of requiring several priority level lists. Packets are treated in identical manner according to their arrival order.

The data structure in this domain is implemented using only one circular linked list pointed by two pointers; a head pointer which points packets to be sent by the scheduling algorithm and a tail pointer which ensures the storage of the arrived packets. Arrived packets are stored in the tail of the list whereas packets to be sent are in heading of the linked list. The figure 6 shows the data structure related to the RPAQM used in high-speed networks. It uses three lists which are the class of service list (contain control parameters), the active priority list used to store all arrived packets in FIFO techniques, and the linked list gather all received packets. The linked list of free priority levels is not used.

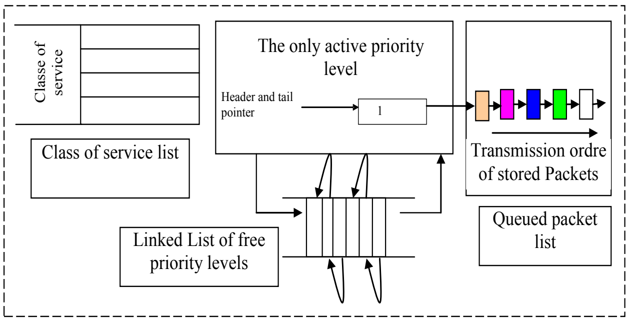
Classe of service
Class of service list
The only active priority level
Header and tail pointer
1
Transmission ordre of stored Packets
Queued packet list
Linked List of free priority levels

Figure 6. data structure in high-speed networks

7. Design Results of the Proposed RPAQM

7.1. Design Methodology

In general, the process of designing a system will proceed from a behavioral to a physical representation, gaining implementation details along the way. High level synthesis converts a behavioral specification of a digital system into an equivalent RTL design that meets a set of stated performance constraint[2]. The designer describes his system with a high level specification at one of abstraction levels. This description with a HDL (Hardware Description Language) is synthesized using existent synthesis tool allowing passage to the next abstraction level until reaching either integration in ASIC or implementation in FPGA. After design verification, a design compiler is used to perform logic synthesis. The logic synthesis tool starts with two kinds of information: an RTL specification given in VHDL and a functional unit library that can include complex functional units. The RTL description accesses these function blocks through VHDL procedure calls. For each procedure or function used, the library must include at least one functional unit able to execute the corresponding operation[20, 21].

7.2. Results

For the implementation of RPAQM architecture, a description in VHDL is carried out at the RTL level and is simulated using ModelSim simulator. This approach has permitted to evaluate the behavior of each component alone as well as the interaction of the overall architecture with all interconnected components. The implemented components can integrate basic standard elements with a known behavior such as adders, comparators, MUX, etc, or other elements where their behaviors have to be defined like sequencer, specific interface and telecommunication operators. Our proposed solutions have to give answers for all questions related to the design specification and the time constraints required by the application context. In order to emulate the environment of our architecture, we have used some file of test containing the entries stimuli with the overall VHDL description to constitute the validation virtual prototyping IP switching system.

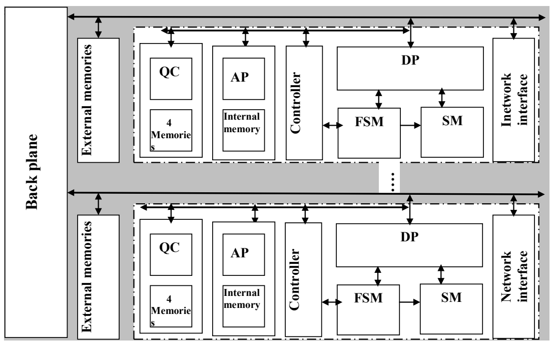

Figure 7. Architecture of the RPAQM in the RTL level

Our architected uses hybrid model of control which combines the control dominated and the memory dominated model. This target architecture has to respond to both the time constraints and the memory saving for the overall data computing operations. Furthermore, in order to make better the organization of our architecture in order to be easily extensible, we have control part, the data path part and the interface part. This separation is also justified to automate the process of design thereafter. Consequently, our proposed RPAQM architecture of the integrated protocols related to IP switch consists on a set of interface and control components. This architecture is capable to transmit data and control information in two directions between distant communication entities. This architecture as depicted in Figure 7includes the following parts:

■ **The memory part** Our proposed architecture is based on the QoS dynamic management. This brings us to define the suitable data structure and to choose the appropriate technique. Memories are used to implement selected techniques for QoS optimization.

■ **The finite state machine (FSM):** it receives the primitive of the system and defines the primitive execution order.

■ **The sequencer (SM)** it allows the generation of all necessary commands and control signals for the data path operation.

■ **The data path (DP):** In order to perform all operations required by the entire architecture, this module contains an interface on one hand to registers and on other hand to dedicated and standard operators (counters addition, etc…).

■ **The controller:** To reconfigure architecture according to the need for the QoS management, and to administer these memories a general controller is designed to assure the dynamic QoS management.

■ **Assignment priority (AP)**: implements the dynamic model of the priority level calculus.

■ **Queue controller (QC)**: implements the queue controller module.

■ **The network interface:** This unit is responsible for the reception/emission of data from the network interface/buffers.

Table 2. synthesis results of the AQM architecture

Logic Utilisation	Used	Avalable	Utilisation
Number of Slice Registres	3806	207360	1%
Number of Slice LUTs	26818	207360	12%
Number of fully used LUT-FF pairs	3308	27316	12%
Number of bonded IOBs	845	1200	70%
Number of BUFG/BUFGCTRLs	16	32	50%
Number of DSP48Es	14	192	7%

The design is simulated and synthesized using ISE design foundation. The results of these operations are presented in table 2. The final ASIC has been implemented using an FPGA Virtex 5. The number of slice Luts used is 20126, and the number of slice registers used is 3806. This circuit operates with clock frequency of 250 MHz and allows the cells transfer on 2 Gbit/s. It can process 65536 priority levels and requires three types of cache memories: the first one has 128x32Bits size to be used by the Assignment Priority (AP) module; the second one has 256x64 bits size. The Queue Controller (QC) module uses four parallel memories to construct a 256x256 matrix which holds 65536 priority levels. The third memory has 64x32 bits size used to reconfigure the FPGA, it has the parameters of each service domain to be configured and installed.

8. Conclusions

QoS guarantees in high-speed packet switched networks still remain a huge challenge for network designers. Since most of service disciplines for providing QoS are deadline-ordered and specified for output queues, RPAQM becomes an essential component for high performance packet networks. The RPAQM must be designed not only to support many priority levels, large buffer size, many input links, and high bandwidth but also to support scalability and reconfigurability.

In this paper, we have proposed a scalable and a reconfigurable PAQM architecture for IP switches that can handle a large number of priorities in different network domains. In addition, we have proposed three architectures in which scalable PAQM's can be reconfigured to support QoS guarantees. The IP switch can therefore be used efficiently in a high-speed packet switch providing fine-grained quality of service guarantees. In addition, to be very fast and reconfigurable, the architecture also scales very well to a large number of priority levels and to large queue sizes. We remarks that the proposed architectures (data structures in different domains) have not great difference in term of hardware description and conception, but they have great difference in active queue management in packet networks switches. Switching packets represents the essential parameters to guarantee QoS which are the scheduling algorithm and buffer management. These parameters are configured differently in the proposed architectures which are based in the implemented data structure (P-CLL). Data structure enables us to support these configurations which increase performances architectures cited below. However, in the first architecture we implement the CBWFQ scheduling algorithm, the second architecture uses the WFPQ algorithm and the last architecture uses the FIFO algorithm. Our current implementation is described in VHDL language at RTL levels, synthesized in a low level and mapped to FPGA architecture.

REFERENCES

[1] N. Ni, L-N Bhuyan "fair scheduling and buffer management in internet router" article, conference on computer computing, INFOCOM2002, vol3, New York, June 2002.

[2] N. I. S. Hwang, B. Hwang, P. M. Chang, C.Y. Wang,

"QoS-Aware Active Queue Management for Multimedia Services over the Internet", Proceedings of the International Multi-Conference of Engineers and Computer Scientists, vol. 2, Hong Kong, 2010.

[3] R. Bhagwan and B. Lin, "Design of a high-speed packet switch for fine-grained quality of service guarantees", *journal*, IEEE International conference on communication (ICC'00), New Orleans, vol3, pp 1430-1434, 2000.

[4] S. Paul, A. J. Pan, and R. Jain "Architecture of the future network and the next generation internet: a survey", journal, Computer communications, volume 34, issue 1, 15 January 2011.

[5] F. Ren, C. Lin, B. Wei, "a robust active queue management algorithm in large delay networks", article, computer communications 28 (485-493), 2005.

[6] Z. Mammeri, "framework for parameter mapping to provide end-to-end QoS guarantees in IntServ/DiffServ architectures", article, computer communications 28 (1074-1092), 2005.

[7] P. Giaccone "Queueing and scheduling algorithms for performance routers" *thesis* of polytechnic university of Torino, Itali, February 2002.

[8] S. Floyd, R. Gummadi, and S. Shenker "adaptive RED: an algorithm for increasing the robustness of RED's active queue management" *rapport*, longer technical report, 1 august 2001.

[9] B. Abbasov and, S. Korukoglu, "Effective RED: An algorithm to improve RED's performance by reducing packet loss", Journal of Network and Computer Applications, vol. 32, pp. 703-709, May 2009.

[10] S. Wallner, "A configurable system-on-chip architecture for embedded and real-time applications: concepts, design and realization" article, Journal of Systems Architecture, Volume 51, Issues 6-7, Pages 350-367, June-July 2005.

[11] M. franklin, P. Crowley, H. Hadimioglu, P. Onufryk "network processor design", book, computer sciences, issue and practice volume 2, 2004.

[12] Michael V. Lau, S. Shieh, P-F. Wang, B. Smith, D. Lee, J. Chao, B. Shung and C-C Shih, "Gigabit Ethernet switches using a shared buffer architecture", article, IEEE Communication magazine, p-p 76-84, December 2003.

[13] A. C-K. Kam "efficient scheduling algorithms for quality of services guarantees in the internet" thesis of Massachust technology institute, April 2000.

[14] N. Seitz, NTIA/ITS, "ITU-T QoS standards for IP-based networks" article, IEEE communication magazine, pp 82-89, June 2003.

[15] S. Pillalamarri, S. Ghosh, "high-speed networks: definition and fundamental attributes", article, computer communications 28 (956-966), 2005.

[16] H. Jonanthan Chao, Yau-Ren Jenq, Xiaolei Gue and Cheuk H. Lam "design of packet-faire queuing schedulers using a RAM-based searching engine" IEEE *journal* on selected areas in communications. Vol. 17, No. 6, pp. 1105-1125, June 1999.

[17] K. Nisar, A. M.Said and H. Hasbullah, "An efficient Voice Priority Queue (VPQ) Scheduler Architectures and Algorithm for VoIP over WLAN Networks", Computer Science Letters, vol.2, Sept. 2010.

[18] A. Baghdadi, « Exploration et conception systématique d'architectures multiprocesseurs monopuces dédiées à des applications spécifiques », thesis of national polytechnic institute of Grenoble, 14 mai 2002.

[19] P. Gupta and N. Mckeown "algorithms for packet classification", *journal*, IEEE Network, mars 2001.

[20] H. Guesmi, R. Djemal, B. Bouallegue, J-P. Diguet, R. Tourki, "high performance architecture of integrated protocols for encoded video application", article, computer standards and interface 26 (301-315), 2004.

[21] H. Guesmi, R. Djemal, B. Bouallegue, H. Youssef et R. Tourki, " Architecture d'un routeur IP de haute performance optimisée pour la différentiation de qualité de service", Premier Congres International IEEE de Signaux, Circuits Et Systèmes (SCS'04), Monastir - Tunisia, 18 - 21 March 2004.