2009

Scheduling Architectures for DiffServ Networks with Input Queuing Switches

Mei Yang
University of Nevada, Las Vegas, mei.yang@unlv.edu

Henry Selvaraj
University of Nevada, Las Vegas, henry.selvaraj@unlv.edu

Enyue Lu
Salisbury University, ealu@salisbury.edu

Jianping Wang
City University of Hong Kong, jianwang@cityu.edu.hk

S. Q. Zheng
The University of Texas at Dallas, sizheng@utdallas.edu

See next page for additional authors

Scheduling Architectures for DiffServ Networks with Input Queuing Switches

Mei Yang[†], Henry Selvaraj[†], Enyue Lu[‡], Jianping Wang[⋆], S. Q. Zheng[*], and Yingtao Jiang[†]

[†] Department of Electrical and Computer Engineering

University of Nevada, Las Vegas, Las Vegas, NV 89154

[‡] Dept. of Mathematics and Computer Science

Salisbury University, Salisbury, MD 21801

[⋆] Department of Computer Science

City University of Hong Kong, Hong Kong

[*] Department of Computer Science

The University of Texas at Dallas, Richardson, TX 75080

E-mail:[†]meiyang@egr.unlv.edu, selvaraj@unlv.nevada.edu, yingtao@egr.unlv.edu,

[‡]ealu@salisbury.edu, [⋆]jianwang@cityu.edu.hk, [*]sizheng@utdallas.edu

Abstract

Due to its simplicity and scalability, the differentiated services (DiffServ) model is expected to be widely deployed across wired and wireless networks. Though DiffServ supporting scheduling algorithms for output-queuing (OQ) switches have been widely studied, there are few DiffServ scheduling algorithms for input-queuing (IQ) switches in the literature. In this paper, we propose two DiffServ scheduling algorithms for DiffServ networks with IQ switches: the dynamic DiffServ scheduling (DDS) algorithm and the hierarchical DiffServ scheduling (HDS) algorithm. The basic idea of DDS and HDS is to schedule EF and AF traffic according to their minimum service rates with the reserved bandwidth and schedule AF and BE traffic fairly with the excess bandwidth. Both DDS and HDS find a maximal weight matching but in different ways. DDS employs a centralized scheduling scheme. HDS features a hierarchical scheduling scheme that consists of two levels of schedulers: the central scheduler and port schedulers. Using such a hierarchical scheme, the implementation complexity and the amount of information needs to be transmitted between input ports and the central scheduler for HDS are dramatically reduced compared with DDS. Through simulations, we show that both DDS and HDS provide minimum bandwidth guarantees for EF and AF traffic as well as fair bandwidth allocation for BE traffic. The delay and jitter performance of DDS is close to that of PQWRR, an existing DiffServ supporting scheduling algorithm for OQ switches. The tradeoff of the simpler implementation scheme of HDS is its slightly worse delay performance compared with DDS.

Keyword: Quality of service, DiffServ, scheduling, input-queuing switches

I. Introduction

The rapid growth of the Internet and wireless communications has driven the demand for wired/wireless broadband Internet access with quality of service (QoS) support. The two main approaches to provide QoS are: Integrated Services (IntServ) [4] and Differentiated Services (DiffServ) [3]. Fine-grained QoS guarantees can be achieved by IntServ. However, the scalability of the IntServ model is limited due to the per-flow reservation and heavy signaling overhead [12]. The DiffServ model is proposed to meet different QoS requirements for various types of clients and network applications. It addresses scalability by a coarse-grain differentiation model.

The DiffServ model [3] is orientated toward edge-to-edge service across a single domain. Traffic is classified into a limited number of service classes according to the service level agreement (SLA) with the network provider. The flow-based traffic classification and conditioning is pushed to edge routers of the domain. Core routers of the domain do not need to maintain per-flow state information, but only need to forward packets according to the per hop behavior (PHB) associated with each service class, which is identified by the DiffServ code point (DSCP) field in the header of each packet. The DiffServ model matches the heterogeneous feature of the Internet and it is capable of providing end-to-end QoS guarantees by bilateral agreements between neighboring domain owners [5]. Due to its simplicity and scalability, DiffServ is expected to be widely deployed across wired and wireless networks [2], [12].

Currently, the IETF defines a set of PHBs which include Expedited Forwarding (EF) PHB, Assured Forwarding (AF) PHB group, and Best Effort (BE) PHB. The EF PHB provides low loss, low delay, low jitter, assured bandwidth, and end-to-end service through the DiffServ domain. The EF PHB is ideally suitable for voice over IP (VoIP), audio-, video- streaming, and other real-time applications. The AF PHB group provides services with minimum rate guarantee and low loss rate [9]. Four AF classes (AF1, AF2, AF3, and AF4) are defined and each class has three levels of drop precedence [1], [9], [22]. The level of forwarding assurance of an IP packet belonging to an AF class depends on the amount of resources allocated to the AF class, the current load of the AF class, and the drop precedence of the packet. AF PHBs are suitable for

network management protocols, such as Telnet, SMTP, FTP, HTTP. All data packets belonging to the BE class are not policed and are forwarded with the best effort.

The implementation of PHBs relies much on the scheduling and queuing schemes used in DiffServ compliant switches and routers. In order to provide premium service to EF traffic, packets belonging to EF class should be served prior to packets belonging to other classes. Meanwhile, to prevent the influence of damaging EF traffic to other traffic, the service rate (bandwidth) for EF traffic should be limited to its peak information rate (PIR). For each AF class, a minimum service rate, referred as committed information rate (CIR), should be guaranteed. On the other hand, to avoid starvation of BE traffic, backlogged BE queues should be served if excess bandwidth is available. In practice, we desire those scheduling and queuing schemes which are efficient in providing differentiated services for different traffic classes, with high throughput, and simple in implementation.

Existing DiffServ supporting scheduling schemes for output-queuing (OQ) switches include priority queuing (PQ), weighted round-robin (WRR), PQWRR [19], [25], and class-based queuing (CBQ) [11], [18]. CBQ ensures explicit rate control for each traffic class by the rate control mechanisms functioned at two schedulers: the general scheduler and the link-sharing scheduler [8]. Compared with PQ and WRR, PQWRR delivers the minimum delay and jitter for EF traffic and provides better bandwidth allocation for AF traffic and BE traffic by priority scheduling of EF traffic and non-EF traffic, and weighted round-robin scheduling of AF traffic and BE traffic. In terms of implementation, PQWRR is simple and more practical than CBQ. Nevertheless, these schemes all assume OQ switch architectures which are not scalable for high line rates and/or large numbers of ports due to the speed limitation of the switching fabric and memories.

Compared with OQ switches, input queuing (IQ) switches are more scalable and practical since they only need the switching fabric and memories to run at the line rate. We hence focus our study on DiffServ supporting scheduling algorithms for IQ switches. Many QoS supporting scheduling algorithms have been proposed for IQ switches. Most of them are maximal weight matching (MWM) based algorithms with different definitions of the weight, such as algorithms with the weight defined as a function of queue length

(e.g. the successive incremental matching over multiple ports (SIMP) algorithm [23], the longest normalized queue first (LNQF) algorithm [16], the worst-case longest port first (LPF), and prioritized LPF algorithms [24]), algorithms with the weight defined as credits of bandwidth [13], and algorithms with the weight defined as time difference [6]. Another noticeable QoS scheduling algorithm is the hierarchical scheduling algorithm [15], which combines a dynamic algorithm which is used to determine input-output matchings and a static algorithm which is used to select a request in the granted input port. However, due to the lack of bandwidth reservation schemes, all these algorithms do not provide bandwidth or delay guarantee for each traffic class. Although the distributed mutlilayered scheduler (DMS) [7] for multistage switches can provide delay bounds for EF flows and guaranteed bandwidth for AF flows, the complex structure of DMS and maintenance of per-flow queues prevent its practical use. In [10], the Adaptive Weighted Fair Queueing with Priority (AWFQP) scheduler attempts to provide QoS guarantees to EF, AF, and BE classes with two levels of schedulers: the Priority Queueing Scheduler and the Fair Queueing Scheduler in the first level, and the Adaptive Queueing Scheduler in the second level.

In this paper, we propose two DiffServ scheduling algorithms for IQ switches: the dynamic DiffServ scheduling (DDS) algorithm and the hierarchical DiffServ scheduling (HDS) algorithm, to provide dynamic bandwidth allocation for DiffServ classes. The basic idea of DDS and HDS is to schedule EF and AF traffic according to their minimum service rates with the reserved bandwidth and schedule AF and BE traffic fairly with the excess bandwidth. Both DDS and HDS find a maximal weight matching but in different ways. DDS employs a centralized scheduling scheme. HDS features a hierarchical scheduling scheme that consists of two levels of schedulers: the central scheduler and port schedulers. Using such a hierarchical scheme, the implementation complexity and the amount of information needs to be transmitted between input ports and the central scheduler for HDS are dramatically reduced compared with DDS. Through simulations, we evaluate the performance of DDS and HDS under bursty arrivals and compare them with PQWRR. We show that both DDS and HDS provide minimum bandwidth guarantees for EF and AF traffic as well as fair bandwidth allocation for BE traffic. DDS also achieves the delay and jitter performance for EF traffic close

to that of PQWRR and the delay performance for AF traffic better than that of PQWRR at high loads.

The rest of the paper is organized as follows. Section II introduces the IQ switch architecture. Section III describes the preliminaries for both algorithms. Section IV presents the DDS algorithm. Section V presents the HDS algorithm. Section VI discusses the simulation results and comparison with PQWRR. Section VII concludes the paper.

II. IQ Switch Architecture

Figure 1 shows an $N \times N$ IQ switch architecture. We assume that all data packets arriving at the switch are segmented into fixed-size cells, transmitted through the switching fabric, and reassembled back into original data packets before they leave the switch. We also assume that time is slotted such that one cell slot is equal to the transmission time of one cell on the input/output line. To remove head-of-line (HOL) blocking, each input port maintains N groups of virtual output queues (VOQs), and each group of VOQs is used to buffer cells destined for an output port.

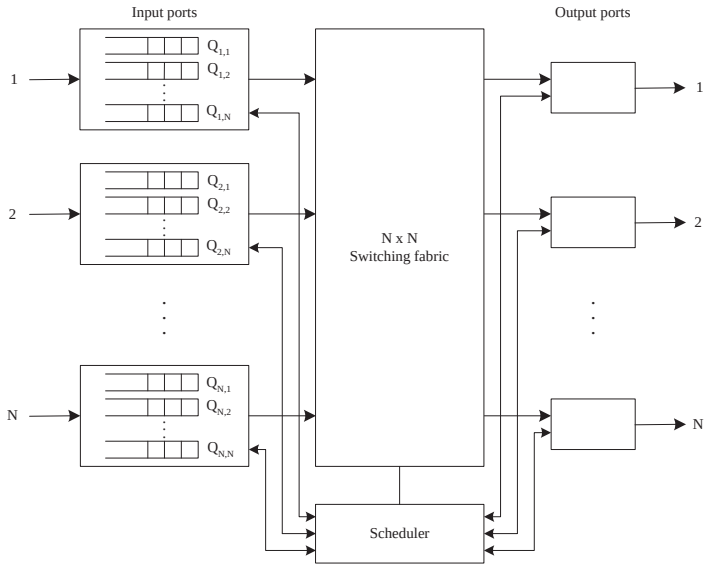

Fig. 1. The IQ switch architecture.

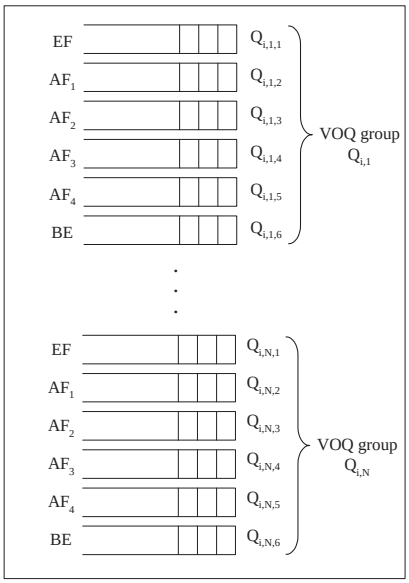

Fig. 2. Queuing scheme at input port I_i.

A VOQ group is composed of K VOQs, each dedicated to buffering cells of a DiffServ class. Figure 2 shows the queuing scheme used at input port I_i, $1 \leq i \leq N$, in which a separate FIFO queue $Q_{i,j,k}$ is used to buffer cells belonging to traffic class k, $1 \leq k \leq K$, and destined for output port O_j, $1 \leq j \leq N$. For the

DiffServ model, we have $K = 6$ with $k = 1$ to 6 representing the classes of EF, AF1, AF2, AF3, AF4, and BE respectively. When a cell arrives at an input (port), it is classified based on its DSCP field and output port address, and buffered in the VOQ corresponding to its traffic class and output (port).

In each cell slot, a scheduling algorithm is needed to determine which N cells in the N^2K VOQs to be transmitted through the switching fabric. In the following, we assume that scheduling in the current cell slot is based on the VOQ status of the previous cell slot, and switching in the current cell slot is based on the scheduling decision made by the previous cell slot.

III. Preliminaries

Three factors need to be considered when designing a DiffServ supporting scheduling algorithm for IQ switches. First, to provide minimum bandwidth guarantees for EF and AF classes, the scheduling algorithm needs to consider the PIR for EF class and CIRs for four AF classes. Meanwhile, to avoid starvation of BE class, backlogged queues should be served if the excess bandwidth is available. Hence, class differentiation and bandwidth reservation and measurement schemes need to be introduced in the scheduling algorithm. Second, the switch throughput should be kept as much as possible. Third, the scheduling algorithm should be simple in implementation.

In the next two sections, we present the dynamic DiffServ scheduling algorithm and the hierarchical DiffServ scheduling algorithm. The service discipline of DDS and HDS is the same: If the reserved bandwidth is available, it serves EF or AF traffic first so that the PIR for EF class and the CIR for each AF class are guaranteed; otherwise, it serves non-EF traffic fairly so that BE traffic is not starved. The difference between DDS and HDS is the scheduling scheme used to find a maximal weight matching. DDS employs a centralized scheme, while HDS features a hierarchical scheme. Before we present each algorithm, we first introduce the bandwidth reservation and measurement schemes at each output port.

We use L to denote the bandwidth of each output link, which is divided into two categories, reserved bandwidth and excess bandwidth (e.g., 90% as the reserved bandwidth and 10% as the excess bandwidth). The reserved bandwidth is further divided into five parts, each corresponding to the guaranteed bandwidth

for a non-BE DiffServ class. To provide bandwidth guarantees for AF classes in a finer granularity and enforce smooth AF traffic, we introduce the time unit of $frame$, which is composed of T time slots. Each output port O_j, $1 \leq j \leq N$, maintains the following variables.

- $R_{j,k}$ denotes the reserved (guaranteed) bandwidth for class k, where $1 \leq k \leq K-1$. $R_{j,1}$ = PIR for EF class, $R_{j,k}$ = CIR for AF$(k-1)$ class, $2 \leq k \leq K-1$, and $\sum_{k=1}^{K-1} R_{j,k} \leq 1$.
- $C_{j,k}$ denotes the cell counter for class k. $C_{j,1}$ counts the number of EF cells up to the current slot, and $C_{j,k}$, $2 \leq k \leq K-1$, counts the number of AF$(k-1)$ cells transmitted in the current frame. We set $C_{j,1} = 0$ at cell slot $t = 0$, and $C_{j,k} = 0$ at cell slot $t \bmod T = 0$ for $2 \leq k \leq K-1$.
- $S_{j,k}$ denotes the bandwidth utilization status for class k. $S_{j,k} = 1$ if $C_{j,1}/t < R_{j,1}$ for EF class or $C_{j,k}/T < R_{j,k}$ for AF$(k-1)$ class, $2 \leq k \leq K-1$; $S_{j,k} = 0$ otherwise.

At the beginning of each cell slot, each output port O_j, $1 \leq j \leq N$, sends S_j to the central scheduler. Each input port I_i, $1 \leq i \leq N$, collects the waiting time of the HOL cell of each non-empty VOQ $Q_{i,j,k}$ as $w_{i,j,k} = t - t'_{i,j,k}$, where $t'_{i,j,k}$ is the entering time slot of the HOL cell. We use a mapping function to map the weight value into the range of 0 to $2^{b_k} - 1$, where b_k is the number of bits used to represent the weight range of traffic class k. In this paper, we use a saturation function which is defined as follows.

$$f(w_{i,j,k}) = \begin{cases} w_{i,j,k} & \text{if } 0 \leq w_{i,j,k} < 2^{b_k}, \\ 2^{b_k} - 1 & \text{otherwise.} \end{cases} \tag{1}$$

IV. The DDS Algorithm

The DDS algorithm finds a maximal weight matching in a centralized way. At the start of each cell slot, each input port I_i sends a weighted vector with NK values to the scheduler. For each VOQ group $Q_{i,j}$, a weighted request vector $V_{i,j}$ is constructed as $(f(w_{i,j,1}), f(w_{i,j,2}), \cdots, f(w_{i,j,K}))$.

A. The DDS Algorithm

The DDS algorithm works iteratively, with each iteration consisting of the following three steps.

Step 1: **Request.** Each unmatched input I_i sends request vectors $V_{i,j}$'s to their corresponding outputs.

Step 2: **Grant.** For each unmatched output O_j, once it receives at least one non-zero request vector, it grants one input as follows.

- If $S_{j,1} = 1$ or $S_{j,k} = 1$ for $2 \leq k \leq K-1$, it grants the input with $\max\{f(w_{i,j,k})|\ f(w_{i,j,k}) > 0, 1 \leq i \leq N\}$ starting from $k = 1$ to $K-1$; otherwise, it grants the input with $\max\{f(w_{i,j,k}) \mid f(w_{i,j,k}) > 0, 1 \leq i \leq N, 2 \leq k \leq K\}$.
- If $f(w_{i',j,k'}) > 0$ is selected for some traffic class k' of input $I_{i'}$, it sends $I_{i'}$ a grant vector with the k'-th entry equal to $f(w_{i',j,k'})$ and other entries equal to '0', and other inputs zero grant vectors (all entries of the vector are set as '0').

Step 3: **Accept.** For each input I_i that receives at least one non-zero grant vector, it selects the output with $\max\{f(w_{i,j,k}) \mid f(w_{i,j,k}) > 0, 1 \leq j \leq N\}$ starting from $k = 1$ to K. The accepted output is notified of the acceptance.

As described in the grant step, if the reserved bandwidth is available, the DDS algorithm allocates the reserved bandwidth to EF and AF traffic by serving the request with the highest weight value of the highest priority class; otherwise, it allocates the excess bandwidth to AF and BE traffic fairly by serving the request with the highest weight value among AF classes and BE class. Additionally, the DDS algorithm is starvation-free since the weight is generated based on the waiting time of the HOL cell and the excess bandwidth is shared by AF and BE traffic fairly.

Note that in grant and accept steps, there might be ties, i.e. requests with equal weights. Ties may exist among different traffic classes, or among different inputs/outputs. To ensure fairness, we break ties by making selections desynchornizedly. We set the selection starting position of each output or input in the static round-robin way. For example, at cell slot t, O_j starts its selection of inputs from $(j + t) \bmod N$ and its selection of classes from $(t \bmod (K-1)) + 1$, and I_i starts its selection of outputs from $(i + t) \bmod N$. Compared with breaking ties randomly [20], static round-robin is much easier to implement.

Figure 3 shows an example of the DDS algorithm for a 4×4 switch. In the current cell slot, the bandwidth utilization vector S_j for each output O_j is given in the second row. In the request step, each input I_i sends

	O_1	O_2	O_3	O_4
s_j	(1, 1, 1, 1, 1)	(1, 1, 1, 1, 1)	(1, 0, 1, 0, 1)	(0, 0, 0, 0, 1)
I_1	(2, 3, 0, 1, 2, 0) (0, 0, 0, 0, 0, 0)	(0, 2, 0, 1, 2, 3) (0, 0, 0, 0, 0, 0)	(0, 0, 2, 1, 2, 0) (0, 0, 2, 0, 0, 0)	(2, 0, 3, 1, 0, 1) (0, 0, 0, 0, 0, 0)
I_2	(3, 2, 1, 0, 0, 4) (3, 0, 0, 0, 0, 0)	(3, 3, 1, 0, 0, 6) (3, 0, 0, 0, 0, 0)	(0, 2, 1, 3, 0, 5) (0, 0, 0, 0, 0, 0)	(0, 2, 0, 1, 0, 4) (0, 0, 0, 0, 0, 0)
I_3	(1, 0, 1, 0, 2, 3) (0, 0, 0, 0, 0, 0)	(2, 1, 0, 3, 2, 0) (0, 0, 0, 0, 0, 0)	(0, 1, 1, 2, 0, 4) (0, 0, 0, 0, 0, 0)	(3, 1, 2, 0, 0, 2) (0, 0, 0, 0, 0, 0)
I_4	(0, 1, 0, 2, 3, 2) (0, 0, 0, 0, 0, 0)	(1, 0, 3, 2, 0, 4) (0, 0, 0, 0, 0, 0)	(0, 3, 0, 1, 3, 7) (0, 0, 0, 0, 0, 0)	(1, 3, 0, 1, 0, 7) (0, 0, 0, 0, 0, 7)

Granted request

Accepted grant

Fig. 3. An example of the DDS algorithm.

a request vector to each output O_j as shown in the first vector of each cell. In the grant step, O_1 grants the EF request from I_2 since the reserved bandwidth for EF class is still available and I_2 has the largest EF request among all inputs. For the same reason, O_2 grants the EF request from I_2. O_3 grants the EF request from I_1 since there is no EF request to O_3, the reserved bandwidth for AF1 class is used up, and I_1 has the largest AF2 request among all inputs. O_4 grants the BE request from I_4 since the reserved bandwidths for all non-BE classes are used up and the BE request from I_4 is the largest among all non-EF requests from all inputs. The grant received at each input is shown as the second vector in each cell. In the accept step, I_1 accepts the grant from O_3. Having two grants with the same value, I_2 accepts one according to tie-breaking scheme, for instance, the grant from O_2. I_4 accepts the grant from O_4. In the first iteration, three pairs of inputs and outputs are matched. More iterations can be conducted to enlarge the number of matched inputs and outputs.

The core of the DDS algorithm is a maximal weight matching algorithm. The number of iterations needed to converge is at most N. Through simulations, we find that on average $\log N$ iterations are adequate to achieve satisfying performance.

B. Hardware Implementation Scheme of DDS

To implement the DDS algorithm, one can use the scheduler architecture shown in Figure 4 (a), in which each input/output is associated with an arbitration component. As shown in Figure 4 (b) and (c), each arbitration component can be constructed by K copies N-input comparator-trees [20], each being used to

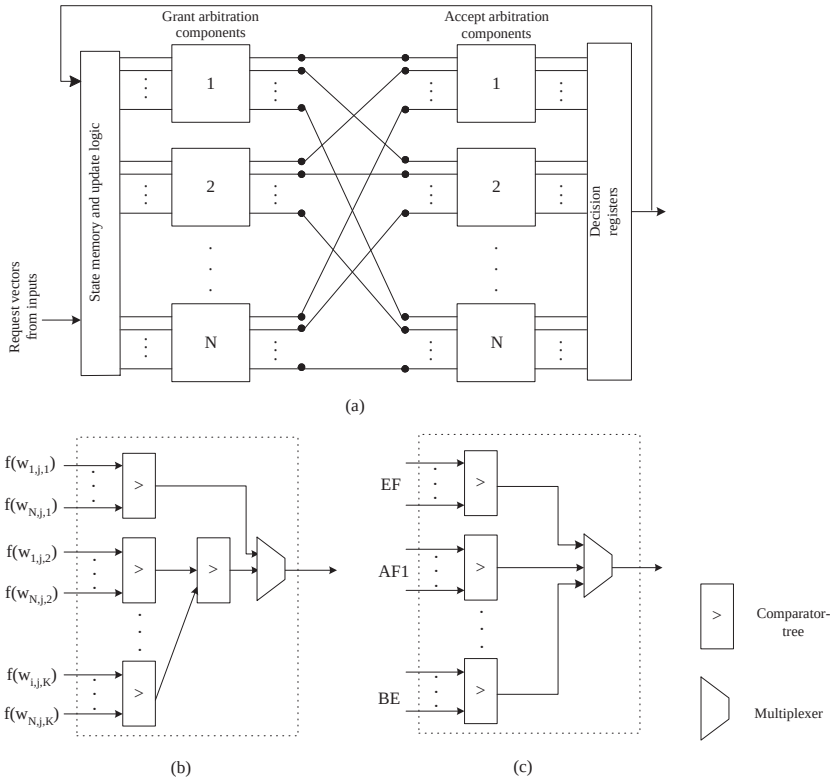

Fig. 4. (a) Block diagram of a DDS scheduler. (b) The grant arbitration component for output O_j. (c) The accept arbitration component for input I_i.

find the maximum weight value for a class k, $1 \leq k \leq K$. One more comparator-tree is needed for each grant arbitration component to choose the maximum weight value of all classes. Each grant or accept arbitration component has $O(\log N \log b)$-gate delay, where $b = \max\{b_k \mid 1 \leq k \leq K\}$. Such an implementation of the DDS algorithm has $O(\log^2 N \log b)$-gate delay. Each arbitration component consumes $O(NKb)$ gates since each comparator tree is composed of $O(Kb)$ gates. The whole DDS scheduler consumes $O(N^2Kb)$ gates.

V. The HDS Algorithm

As we can see from the previous section, the construction of the DDS scheduler is complex. In order to reduce the implementation complexity of the scheduler, we extend the idea of hierarchical scheduling [15] and propose the hierarchical DiffServ scheduling algorithm. The HDS algorithm separates the tasks of providing differentiated services and maximizing switch throughput by employing two levels of schedulers.

One level is the central scheduler which is designed to maximize the switch throughput by computing a maximal size matching (MSM) between input ports and output ports. The other level is formed by input port schedulers which provide differentiated services by serving cells belonging to different classes dynamically. In light of the idea of exhaustive matching [17], the central scheduler employs a three-phase exhaustive MSM algorithm. At the granted input port, the service policy changes according to the bandwidth utilization at the destined output port such that minimum bandwidth guarantees for EF and AF classes and fair bandwidth allocation for BE class are provided.

In the HDS algorithm, at the start of each cell slot, each input port I_i only needs to send a $2N$-bit vector P_i to the central scheduler, where $P_{i,j} = 2$ if I_i has more than one EF cells in VOQ group $Q_{i,j}$, $P_{i,j} = 1$ if I_i has at least one cell in VOQ group $Q_{i,j}$, and $P_{i,j} = 0$ otherwise.

A. The HDS Algorithm

The HDS algorithm works in two stages.

Stage I: The central scheduler finds a maximal size matching in a three-phase exhaustive scheme iteratively. We assume that each input port I_i has an accept pointer a_i indicating the accept starting position, and each output port O_j has a grant pointer g_j indicating the grant starting position. Each iteration of stage I consists of the following three steps.

Step 1: **Request.** Each I_i sends a request to every O_j for which it has a queued cell.

Step 2: **Grant.** If an unmatched O_j receives any request, it selects one request to grant starting from the input port that g_j points to in a round-robin manner. For *the first iteration*, if $P_{i,j} = 2$ for some I_i, g_j is updated to i, otherwise, g_j is updated to one beyond the granted input port.

Step 3: **Accept.** If an unmatched I_i receives any grant, it selects one grant to accept starting from the output port that a_i points to in a round-robin manner. a_i is updated to the accepted output port.

After Stage I finishes, the central scheduler will send to each input port I_i an N-bit grant vector G_i, and S_j if there exists $G_{i,j} = 1$ for some j.

Stage II: For each input I_i that receives a non-zero grant vector (assuming that $G_{i,j} = 1$), if $\sum_{k=1}^{K-1} S_{j,k} f(w_{i,j,k}) \neq 0$, then it will select $Q_{i,j,k}$ such that $S_{j,k} = 1$ starting from $k = 1$ to $K - 1$; otherwise, it will select $Q_{i,j,k}$ with $\max\{f(w_{i,j,k}) \mid f(w_{i,j,k}) > 0, 2 \leq k \leq K\}$.

Figure 5 illustrates an example of the exhaustive scheduling algorithm used at stage I for a 4×4 switch. At the beginning of the cell slot, grant pointers are set as $g_1 = 1$, $g_2 = 3$, $g_3 = 3$, and $g_4 = 2$, and accept pointers are set as $a_1 = 2$, $a_2 = 4$, $a_3 = 3$, and $a_4 = 1$. Given the request matrix P, in the request step, each input port I_i sends a request to each output O_j with $P_{i,j} > 0$ for $1 \leq i, j \leq 4$ as shown in Fig. 5 (a). As shown in Fig. 5 (b), in the grant step, each output grants one request starting from its grant pointer and updates its grant pointer accordingly. Notice that O_3 grants the request from I_3 and let g_3 stay at I_3 since $P_{3,3} = 2$. In the accept step, each input port accepts one grant starting from its accept pointer and updates its accept pointer to the accepted output port as shown in Fig. 5 (c). The generated grant matrix G is shown in the figure. Using such a pointer updating scheme, in the next cell slot, request from VOQ group $Q_{3,3}$ will continue to be favored, thereby serving EF traffic with the highest priority.

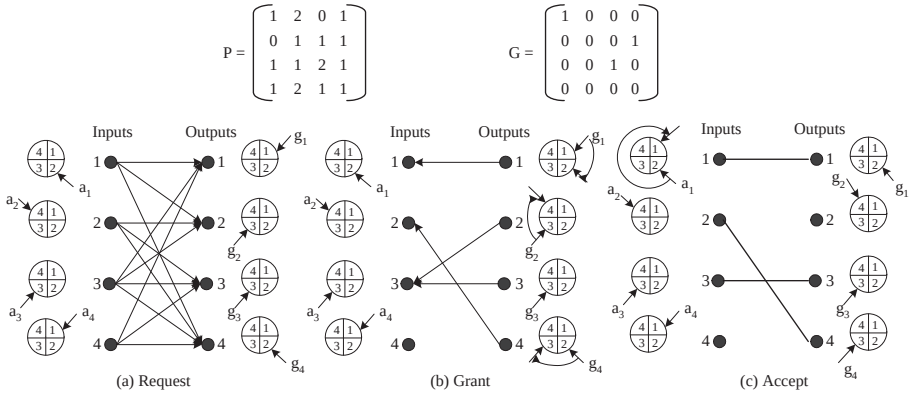

Fig. 5. An example of the exhaustive scheduling algorithm used at the central scheduler.

Similar to the DDS algorithm, the HDS algorithm also finds a maximal weight matching. However, different from the DDS algorithm, the HDS algorithm distributes the selection of the highest weight request to each input port, hence simplifies the operation at the central scheduler. In each cell slot, the central scheduler only needs to find a maximal size matching. As one can understand, the tradeoff of the two-level scheduling is that the maximal weight matching found by the HDS algorithm may not be as good as the one

found by the DDS algorithm in terms of the total weight.

B. Hardware Implementation Scheme of HDS

To implement the central scheduler, one can use the scheduler architecture shown in Figure 6 (a), in which each input/output is associated with an arbiter, which is responsible for selecting one out of N requests. Each arbiter can be implemented by the parallel round-robin arbiter (PRRA) proposed in [26], which has $O(\log N)$-gate delay and consumes $O(N)$ gates. We find through simulations that on average $\log N$ iterations are adequate to achieve satisfying performance. Hence, the first stage of the HDS algorithm can be implemented in $O(\log^2 N)$-gate delay and $O(N^2)$ gates.

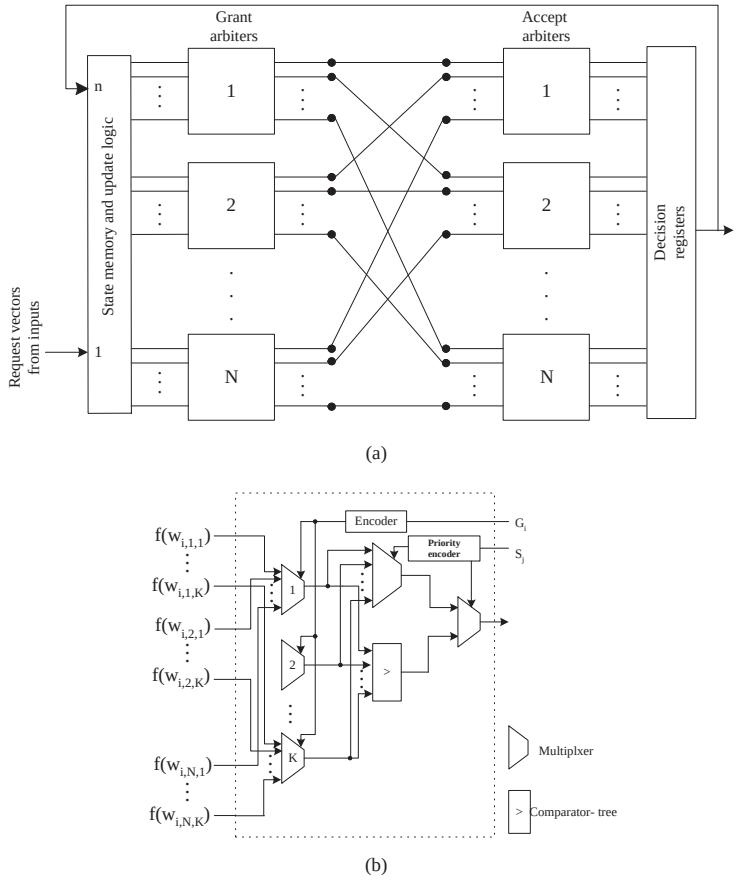

Fig. 6. (a) Block diagram of the central scheduler. (b) Block diagram of a port scheduler.

As shown in Figure 6 (b), each port scheduler majorly consists of K N-input multiplexers, one K-input multiplexer, and one K-input comparator-tree, which is responsible for selecting the maximum weight value among all traffic classes of the same VOQ group. Each port scheduler has $O(\log N + \log K \log b)$-gate delay,

where $b = \max\{b_k \mid 1 \leq k \leq K\}$, and consumes $O(NK + Kb)$ gates.

The total delay of such an implementation of the HDS algorithm is $O(\log^2 N + \log K \log b)$-gate delay, which is faster than the implementation of the DDS algorithm, $O(\log^2 N \log b)$-gate delay. The total number of gates needed for the HDS scheduler is $O(N^2K + NKb)$, which is also smaller than that of the DDS scheduler, $O(N^2Kb)$ gates.

In addition, the amount of information to be transmitted between each input port and the central scheduler in the HDS algorithm is much less than in the DDS algorithm. In each cell slot, in the HDS algorithm, each input port only needs to send $2N$ bits to the central scheduler and the central scheduler only needs to send $N + K$ bits back to each input port, while in the DDS algorithm, each input port needs to send NKb bits to the scheduler and the scheduler needs to send back NK bits to each input port. Table I summaries the difference of implementation complexity between HDS and DDS.

Algorithm	**Time (gate delay)**	**Area (number of gates)**	**Bits sent from each input**	**Bits sent back to each input**
HDS	$O(\log^2 N + \log K \log b)$	$O(N^2K + NKb)$	$2N$	$N + K$
DDS	$O(\log^2 N \log b)$	$O(N^2Kb)$	NKb	NK

TABLE I

COMPARISON OF THE IMPLEMENTATION COMPLEXITY OF HDS AND DDS.

VI. PERFORMANCE EVALUATION

We evaluate the performance of the DDS and HDS algorithms in two aspects: fairness and efficiency, where fairness is measured by the received bandwidth and efficiency is measured by the average cell delay and delay jitter. The cell delay is the queuing delay that a cell encounters in the switch. For EF traffic, we also consider its delay jitter performance, which is defined as the difference between the cell delays of two consecutive cells. To validate our evaluation, we compare the performance of the DDS and HDS algorithms with that of the PQWRR algorithm for OQ switches.

A cell-based simulator is developed and simulations have been conducted assuming that all queue sizes are infinite. In our simulations, we consider bursty traffic arrivals using 2-state modulated Markov-chain sources [21]. Each source alternately generates a burst of full cells (all with the same destination) followed by an idle period of empty cells. The number of cells in each burst or idle period is geometrically distributed. Let $E(B)$ and $E(D)$ be the average burst length and the average idle length in terms of the number of cells respectively. Then, we have $E(D) = E(B)(1-\rho)/\rho$, where ρ is the load of each input source. We assume that the destination of each burst is uniformly distributed.

In all the simulations, we assume that the average cell arrival rates of EF class and AF classes to each output link are 18%, 24%, 20%, 16%, and 12% respectively by default. To ensure guaranteed service to EF traffic, we set its PIR a little more than its arrival rate [11], e.g. $R_{j,1} = 18\% \times 1.1 = 19.8\%$. The CIRs for AF1 through AF4 to each output port are 24%, 20%, 16%, and 12% respectively. In the following simulations, we assume the frame size is 1000 and $b_k = 4$ for all $1 \le k \le K$.

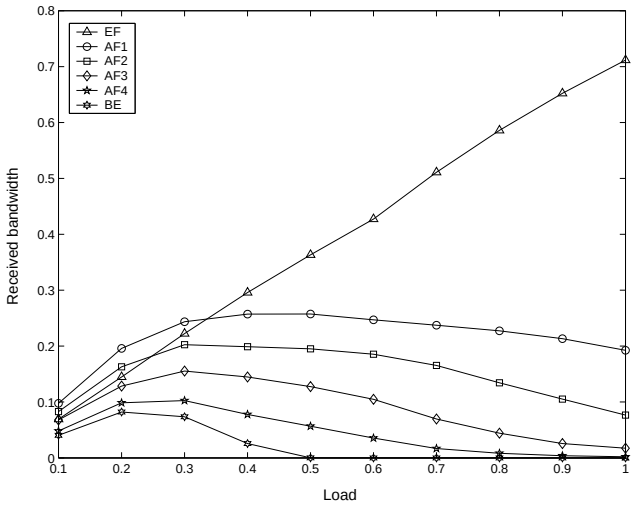

Fig. 7. Received bandwidth using PQWRR.

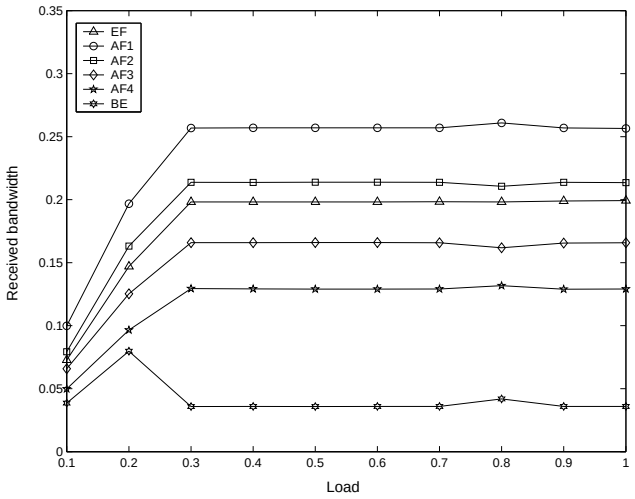

Fig. 8. Received bandwidth using DDS.

A. Bandwidth Allocation

First, we evaluate the effectiveness of the DDS and HDS algorithms supporting dynamic bandwidth allocation when a link is overloaded. We assume a 4×4 switch, the average burst length $E(B) = 32$, and the number of iterations allowed for DDS and the Stage I of HDS is 4. We assume that output link 1 is the

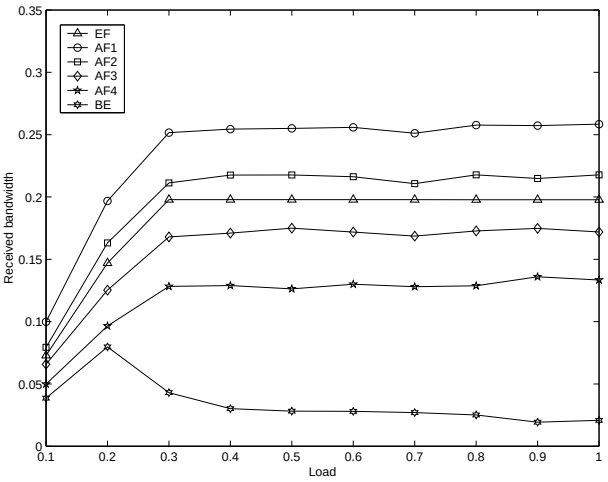

Fig. 9. Received bandwidth using HDS.

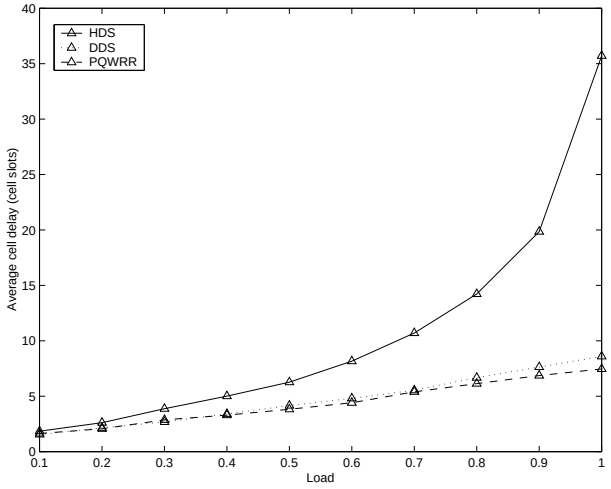

Fig. 10. Delay performance of EF traffic.

overloaded link and we vary the load to each VOQ group destined for output link 1 from 0.1 to 1.0.

Figures 7 to 9 show the received bandwidth of each traffic class for PQWRR, DDS, and HDS respectively. For a load below 0.25, the received bandwidth of each traffic class is able to keep up with its arrival rate for three schemes. However, for a load beyond 0.25, the received bandwidth of EF traffic by PQWRR still follows the arrival rate without regarding to the limitation of its PIR. For a load beyond 0.30, due to the influence of damaging EF traffic, the received bandwidth of AF traffic by PQWRR is degrading dramatically, and BE traffic cannot get any service at all.

On the other hand, DDS and HDS guarantee but limit the received bandwidth of EF traffic to its PIR, 19.8%, assure the CIR for each AF traffic, and avoid the starvation of BE traffic when the load is greater than 0.25. For example, when the load is at 0.40, the bandwidth received by EF, AF1, AF2, AF3, AF4, and BE traffic for DDS is 19.8%, 25.70%, 21.37%, 16.60%, 12.92%, and 3.6% respectively, while for HDS is 19.8%, 25.45%, 21.76%, 17.10%, 12.89%, and 3.0% respectively. Such bandwidth distributions conform to the design goal of DDS and HDS, which is to provide minimum bandwidth guarantees for non-BE classes and fair bandwidth allocation for BE class.

B. Delay Performance

Next, we examine the delay performance of DDS and HDS using simulations of a 16×16 switch under bursty arrivals assuming $E(B) = 32$ and the destination of each burst uniformly distributed. The number of iterations allowed for DDS and HDS is set as 4. Figure 10 shows the average cell delay vs. load of EF traffic for DDS, HDS, and PQWRR. The average cell delay of EF traffic using DDS is very close to that using PQWRR. The average cell delay of EF traffic using HDS is not as good as that using DDS and PQWRR. Figure 11 shows the jitter distribution of EF traffic at load 0.90 for DDS, HDS, and PQWRR. For DDS and HDS, over 90% EF traffic has jitter less than 1 cell slot, which is comparable to PQWRR.

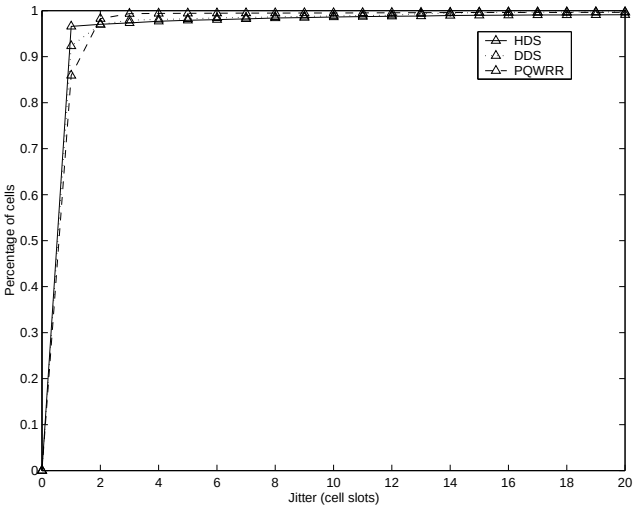

Fig. 11. EF jitter distribution.

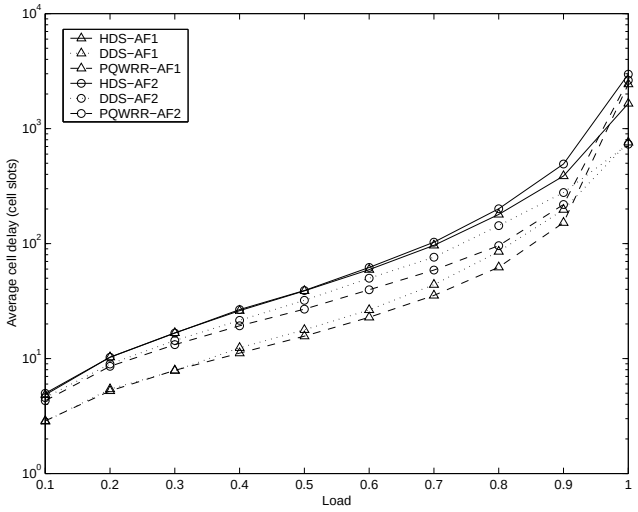

Fig. 12. Delay performance of AF1 and AF2 traffic.

Figure 12 shows the average cell delay vs. load of AF1 and AF2 traffic for DDS, HDS, and PQWRR. Figure 13 shows the average cell delay vs. load of AF3 and AF4 traffic for DDS, HDS, and PQWRR. The average cell delay of each AF class using DDS is close to that using PQWRR for loads below 0.95. For loads over 0.95, DDS performs even better than PQWRR. The reason is that DDS uses a function of the waiting time as the weight but PQWRR uses the queue length as the weight. In Figure 13, for loads lower than 0.60, HDS performs close to PQWRR. With loads going up, the performance of HDS is degrading. Figure 14 shows the average cell delay vs. load of BE traffic for DDS, HDS, and PQWRR. For loads lower than 0.90, HDS performs better than DDS and PQWRR. In general, DDS outperforms HDS in delay performance. This

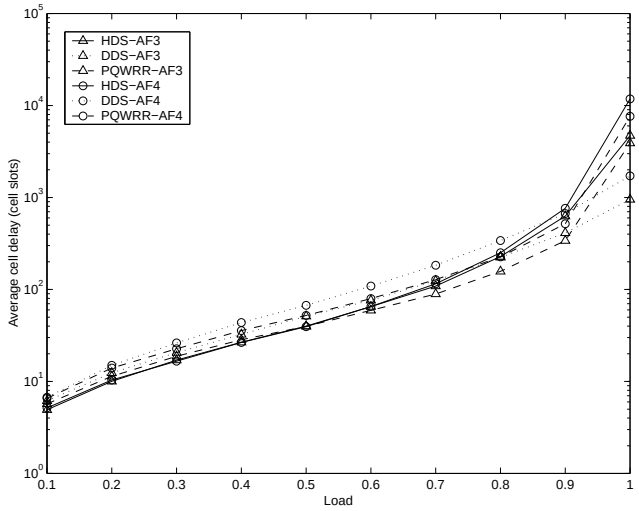

Fig. 13. Delay performance of AF3 and AF4 traffic.

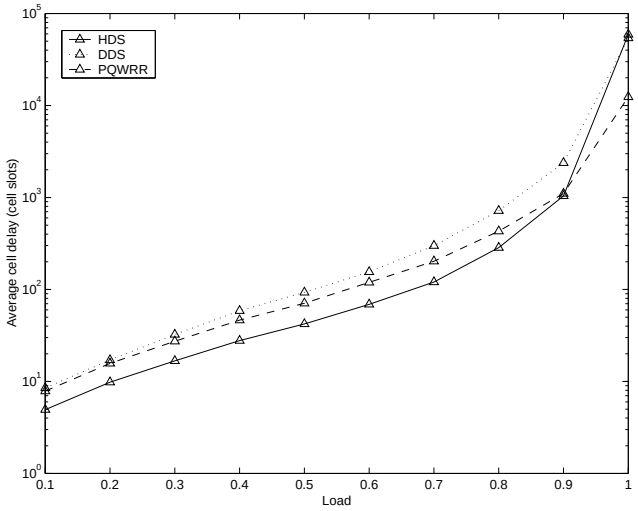

Fig. 14. Delay performance of BE traffic

is consistent with our intuition that using a centralized scheme DDS tends to find a larger weight maximal weight matching than HDS.

In the worst case, N iterations are needed for DDS to find a maximal weight matching. Similarly, at most N iterations are needed for the central scheduler of HDS to find a maximal size matching. However, the number of iterations allowed in one cell slot is limited in reality. Figures 15 and 16 show the effect of the number of iterations allowed on the average cell delay of AF1 traffic using DDS and HDS respectively. We can see that DDS or HDS with 2 iterations achieves significant performance improvement over DDS or HDS with 1 iteration. The performance of DDS or HDS with 4 iterations is very close to the performance of DDS or HDS with 16 iterations. That is why we set the number of iterations allowed as 4 for previous simulations on 16×16 switches.

The purpose of using frame is to smooth bandwidth sharing of AF traffic in a finer way. As we can understand, the smaller the frame size, the finer bandwidth sharing. However, smaller frame size may introduce longer cell delay. Figure 17 and Figure 18 show the influence of different frame sizes to the average cell delay of AF classes for DDS and HDS respectively. It shows that the performance of classes AF1 and AF2 improves, while the performance of classes AF3 and AF4 degrades as the frame size increasing from 1000 to 10000. In the previous simulations, we set the frame size at 1000.

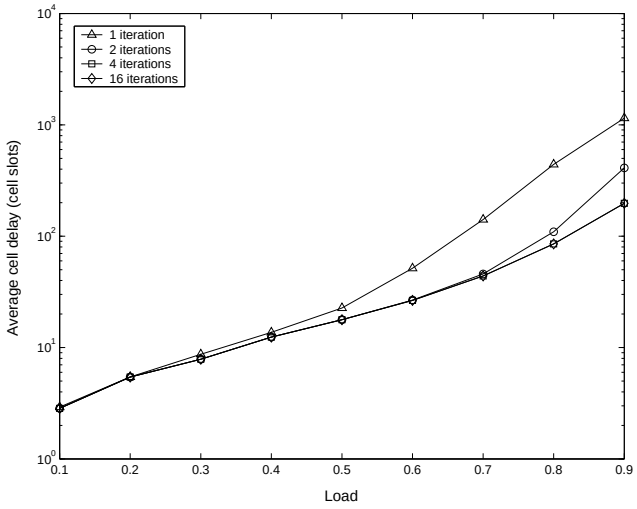

Fig. 15. Delay performance of AF1 traffic with different number of iterations allowed using DDS.

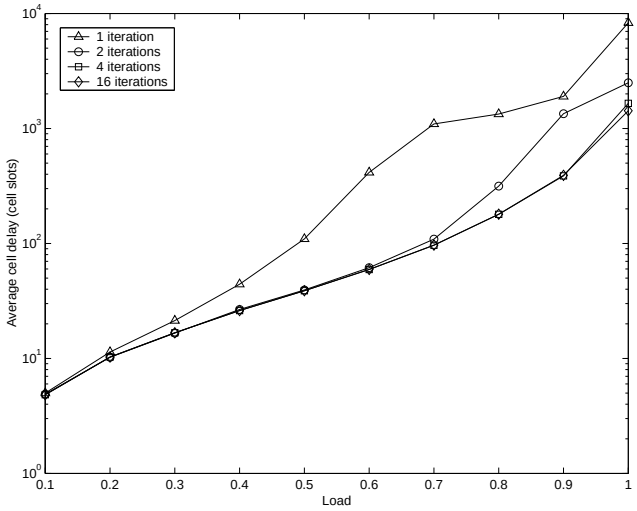

Fig. 16. Delay performance of AF1 traffic with different number of iterations allowed using HDS.

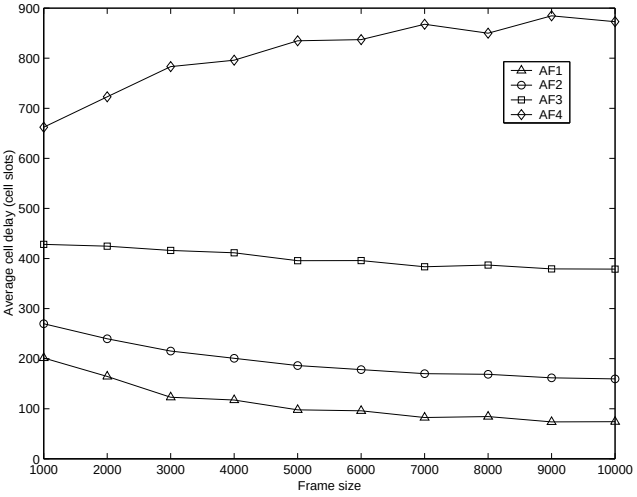

Fig. 17. Delay performance of AF1 traffic vs. different frame sizes using DDS.

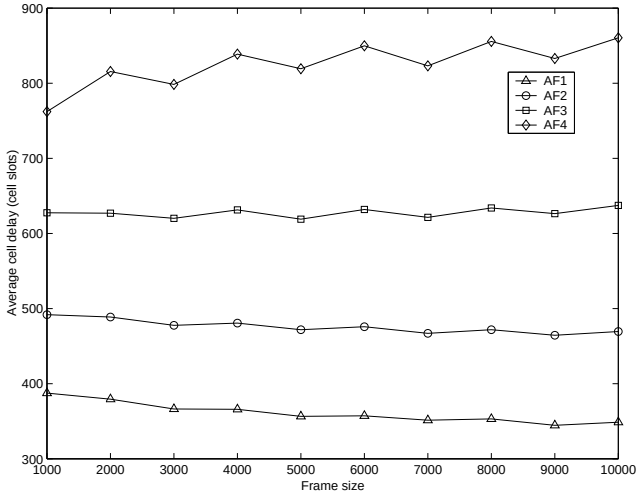

Fig. 18. Delay performance of AF1 traffic vs. different frame sizes using HDS.

VII. Conclusion

In this paper, we proposed the dynamic DiffServ scheduling (DDS) algorithm and the hierarchical DiffServ scheduling (HDS) algorithm, to support dynamic bandwidth allocation for DiffServ classes on IQ switches. With bandwidth measurement scheme at output ports, both DDS and HDS provide minimum bandwidth guarantees for EF and AF traffic with the reserved bandwidth as well as fair bandwidth allocation for BE traffic with the excess bandwidth. We show that DDS is starvation-free since it generates the weight

based on the waiting time of the head-of-line cell instead of the queue length. Compared with DDS, the advantage of HDS is that the implementation complexity and the amount of information needs to be transmitted between each input port and the central scheduler are much reduced by using a hierarchical scheme. The tradeoff of HDS is its slightly worse delay performance compared with DDS, as shown in the simulation results. Since IQ switches are more scalable than OQ switches, HDS and DDS are very useful to implement DiffServ model and other differentiated service models, such as the Olympic service [9].

REFERENCES

[1] D. Adami, S. Giordano, M. Pagano, and R. Secchi: Optimization of scheduling algorithms parameters in a DiffServ environment. Symposium on Applications and the Internet Workshops, 2005, pp. 276-279.

[2] A. Bader, G. Karagiannis, L. Westberg, et. al. “QoS signaling across heterogeneous wired/wireless networks: resource managment in DiffServ using the NSIS protocol suite. International Conference on Quality of Service in Heterogeneous Wired/Wireless Networks 2005, pp. 51-56.

[3] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss: An architecture for differentiated services. IETF RFC 2475, Dec. 1998.

[4] R. Braden, D. Clark, and S. Shenker: Integrated services in the Internet architecture: an overview. IETF RFC 1633, 1994.

[5] B. Carpenter and K. Nichols: Differentiated services in the Internet. Proceedings of the IEEE. 2002, vol. 90, no. 9, pp. 1479-1494.

[6] C. Chen and M. Komatsu: An adaptive scheduler to provide QoS guarantees in an input-buffered switch. International Conference on Communications, 2002, vol. 2, pp. 1118-1122.

[7] F. Chiussi and A. Francini: A distributed scheduling architecture for scalable packet switches. IEEE Journal of Selected Areas in Communications 2000, vol. 18, no. 12, pp. 2665-2683.

[8] S. Floyd and V. Jacobson: Link-sharing and resource management models for packet switches. IEEE/ACM Transactions on Networking 1995, vol. 3, no. 4, pp. 365-386.

[9] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski: Assured forwarding PHB group. IETF RFC 2597, 1999.

[10] I.-S. Hwang, B.-J. Hwang, and C.-S. Ding: Adaptive weighted fair queueing with priority (AWFQP) scheduler for DiffServ networks. Journal of Informatics & Electronics 2008, vol. 2, no. 2, pp. 15-19.

[11] V. Jacobson, K. Nichols, and K. Poduri: An expedited forwarding PHB group. IETF RFC 2598, 1999.

[12] H. Jiang, W. Zhuang, X. Shen, A. Abdrabou, and P. Wang. Differentiated services for wireless mesh backbone. IEEE Communications Magazine 2006, vol. 44, no. 7, pp. 113-119.

[13] A. Kam and K. Sui: Linear complexity algorithms for QoS support in input-queued switches with no speedup. IEEE Journal of Selected Areas in Communications 1999, vol. 17, no. 6, pp. 1040-1056.

[14] N. D. Kiameso, H. Hassanein, H. T. Mouftah: Analysis of prioritized scheduling of assured forwarding in DiffServ Architectures. IEEE International Conference on Local Computer Networks, 2003, pp. 614.

[15] H. Kim, K. Kim, and Y. Lee: Hierachical scheduling algorithm for QoS guarantee in MIQ switches. IEEE Electronic Letters 2000, vol. 36, no. 18, pp. 1594-1595.

[16] S. Li and N. Ansari: Provisioning QoS features for input-queued ATM switches. Electronics Letters 1998, vol. 34, no. 19, pp. 1826-1827.

[17] Y. Li, S. Panwar, and H. J. Chao: The dual round-robin matching with exhaustive service. IEEE Workshop on High Performance Switching and Routing, 2002, pp. 58-63.

[18] G. Mamais, M. Markaki, G. Politis, and I. S. Venieris: Efficient buffer management and scheduling in a combined IntServ and DiffServ architecture: a performance study. International Conference on ATM, 1999, pp. 236-242.

[19] J. Mao, W. M. Moh, and B. Wei: PQWRR scheduling algorithm in supporting of DiffServ. International Conference on Communications, 2001, vol. 3, pp. 679-684.

[20] N. Mckeown: Scheduling algorithms for input-buffered cell switches. Ph. D. Thesis, Univerity of California at Berkeley, 1995.

[21] N. Mckeown: The iSLIP scheduling algorithm for input-queued switches. IEEE/ACM Transactions on Networking 1999, vol. 7., no. 2, pp. 188-201.

[22] T. Minagawa and T. Kitami: Packet size based dynamic scheduling for assured services in DiffServ network. Electronics and Communications in Japan 2004, vol. 88, no. 1, pp. 12-20.

[23] R. Schoenen, G. Post, and G. Sander: Prioritized arbitration for input-queued switches with 100% throughput. IEEE ATM Workshop, 1999, pp. 253-258.

[24] M. Song and M. Alam: Two scheduling algorithms for input-queued switches guaranteeing voice QoS. IEEE GLOBECOM, 2001, pp. 92-96.

[25] Y. Zhang and P. G. Harrison: Performance of a priority-weighted round robin mechanisms for differentiated service networks. IEEE International Conference on Computer Communications and Networks, 2007, pp. 1198-1203.

[26] S. Q. Zheng, M. Yang, J. Blanton, P. Golla, and D. Verchere: A simple and fast parallel round-robin arbiter for high-speed switch control and scheduling. IEEE Midwest Symposium on Circuits and Systems, 2002, pp. 671-674.