

HOBRP: A hardware optimized packet scheduler that provides tunable end-to-end delay bound

Ruisheng Wang* Youjian Zhao† Hongtao Guan‡ Guanghui Yang§
Department of Computer Science & Technology, Tsinghua University
Beijing, P.R.China, 100084
{wangrs06*, zhaoyj†, ght‡, ygh07§}@csnet1.cs.tsinghua.edu.cn

Abstract—A packet scheduler is a primary component of the improved Quality of Service (QoS) model for today’s Internet. Although many fair packet schedulers have been proposed through theoretical consideration, practical high-speed packet schedulers remain elementary. The disparity arises because existent schedulers either lack of necessary QoS guarantee or have an unacceptable cost of computation and storage. In this paper, we propose a simple and efficient packet scheduler called Hardware Optimized Bit Reversal Permutation (HOBRP) based scheduler. Besides some common merits including low time- and space-complexity, bounded end-to-end delay guarantee and constant fairness index that many well-known schedulers have already owned, our HOBRP still possesses two additional features: One is that the end-to-end delay bound of HOBRP is tunable, which makes itself flexible enough to provide different levels of delay bounds for diverse types of application flows. The other is that all the operations and structures used by HOBRP are very simple and easy to be pipelined and paralleled, which benefits an intuitive high-speed hardware design scheme.

I. INTRODUCTION

With the development of Internet, the amount of non best-effort services, such as voice over IP and video conferencing, are taking an increasing proportion of Internet traffic. How to efficiently satisfy QoS requirements (expressed in terms of delay, delay-jitter, throughput and loss rate) of these services is becoming a challenge for the design of Internet architecture. Many frameworks, such as Diffserv [1] and IntServ [2], have been proposed to enable multi-service network. One of the key parts of these frameworks is a packet scheduler, which decides the service sequences of these application flows.

An ideal packet scheduler is expected to have two properties. One is to be simple, which means a packet scheduler should have low time- and space- complexity and a simple hardware implementation. This makes a packet scheduler available for high-speed network environments with tens of thousand flows. The other is to provide QoS guarantee including flow isolation, fairness, low delay and delay-jitter for various network services, which makes these services work well.

The current packet scheduling algorithms can generally be classified into two categories: timestamp based scheduling and frame based scheduling. Timestamp based scheduling algorithms, including traditional Weighted Fair Queueing (WFQ

[3]) (PGPS [4]) and its variants, such as Virtual Clock [5], WF^2Q [6], $L - WF^2Q$ [7], Self-Clocked FQ [8], Start-Time FQ [9] FFQ, SPFQ [10] and Time-Shift FQ [11], maintain a virtual clock to emulate the ideal Generalized Processor Sharing (GPS [4]). Usually these algorithms have a low local delay and short-term fairness. However, no matter what method they use to calculate the timestamp, these algorithms still have at least time complexity $O(\log N)$ [12] (this is because the best known algorithm to insert a number into a sorted array needs $O(\log N)$), which is not good enough for high-speed network interface. For example, the available time to forward a 64-bytes cell for a 40Gbps network interface is only 12.8 ns, which is very time critical.

Frame based (or Round-Robin based) scheduling is to serve flows in a round or frame, the classical algorithms of which are WRR, DRR [13], Aliquem [14], CORR [15], RRR [16], SRR [17]. Although frame based scheduling usually have $O(1)$ scheduling complexity, most of them lack of short-term fairness and bounded end-to-end delay compared to timestamp based ones.

Recently, several hybrid schedulers have been proposed, such as BSFQ [18], GR3 [19], FRR [20] and Stratified RR [21]. These schedulers cluster flows with similar weights into a same group (or class). A timestamp based approach is used for inter-group scheduling, and a round-robin based approach is used for intra-group scheduling. Although these algorithms generally have short-term fairness and are still of $O(1)$ scheduling time complexity, some operations, such as adding or deleting a flow, are still complex and not suitable for hardware implementation.

In this paper, we propose a simple and efficient frame based packet scheduler named Hardware Optimized Bit Reversal Permutation (HOBRP) based scheduler. The proposed scheduler has following merits.

- 1) (*fairness*) HOBRP provides constant Proportional Fairness Index (PFI) [8] and Worst-case Fairness Index (WFI) [6].
- 2) (*tunable delay bound*) HOBRP provides bounded end-to-end delay. Moreover, its delay bound can be shortened if the ratio of Guaranteed traffic to Overall volume (G/O) is less than 1. In particular, if G/O is no larger than 1/2, the GPS-relative delay of HOBRP scheduling can be less than the transmission time of one packet, which is

This work is supported by National Natural Science Foundation of China under Grant No. 60773150 and 863 project 2007AA01Z219.

the optimum delay bound in practical packet scheduling environment.

3) (implementation complexities)

- a) The scheduling time complexity of HOBRP is $O(\log k)$ and its time complexity to add or delete a flow is $O(k)$, where k is the order of link bandwidth ($C = 2^k$).
- b) The space complexity of HOBRP is $O((k+1)N)$, where N is the number of active flows.
- c) Most importantly, the operations and structures used in HOBRP, such as binary searching and circular linked list array, are very suitable for pipelining and paralleling, which benefits an intuitive high-speed hardware design scheme.

The rest of paper is organized as follows: We provide a background material about packet scheduler and introduce a novel Bit Reversal Permutation (BRP) based scheduling in Section II. We then present Hardware Optimized BRP (HOBRP) scheduler in Section III and analyze its properties in Section IV. Hardware implementation issues about HOBRP are discussed in Section V. Section VI demonstrates the delay and short-term throughput property of HOBRP by using simulations. And the whole paper concludes in Section VII.

II. BACKGROUND AND BRP

A. Packet Scheduling Background

Modern high-speed routers are composed of multiple network interfaces that are usually interconnected by a crossbar fabric. The switch fabric is operated in a slotted and synchronized fashion. Packets of various sizes are split into fixed-size (e.g. 64 bytes) parts in the input network interface, and then reassembled at the output network interface. For this reason, we assume that packets are of the same size L in the rest of the paper.

A packet scheduler is composed of three parts. A *flow_add* procedure to accept a new flow, a *flow_delete* procedure to remove an old flow, and a *schedule* procedure to decide which packet to serve once the previous packet has been transmitted. The *schedule* procedure is invoked at every timeslot, which needs a low time complexity operation. The *flow_add* and *flow_delete* procedures are less time critical, since a media flow usually lasts for tens of seconds.

The basic scenario used in this paper is as follows: There are N flows (named as f_1, f_2, \dots, f_N) competing for an output network interface. The capacity (C) of the network interface is normalized as 2^k ($C = 2^k$), and the granularity of rate allocation is 1. The reserved rate of flow f_i is r_i (r_i is a positive integer). For schedulability reason, we need $\sum_{i=1}^N r_i \leq C$. Our packet scheduler operates by dividing time into frames of same size. A frame contains 2^k timeslots and a flow f is allocated r_f timeslots per frame.

In an *ideal* scheduler, for a flow f with rate r_f , the number of packets that has been transmitted at timeslot t is denoted as $S_f^{id}(t) = r_f(t - t_0)/C$, where $t \geq t_0$ and t_0 is the arrival

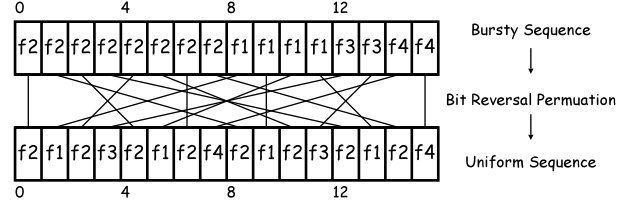


Figure 1: An example of Bit Reversal Permutation

timeslot of flow f . We also denote the number of packets transmitted of f at timeslot t in a real packet scheduler ps as $S_f^{ps}(t)$.

Definition 1: We say that a packet scheduler ps provides bounded delay for flow f , if $S_f^{ps}(t)$ satisfies

$$S_f^{id}(t) \leq S_f^{ps}(t + d_f^{ps}), d_f^{ps} \leq const$$

where $const$ is a small constant that is not related to N , the number of active flows in the scheduler.

B. Bit Reversal Permutation

Our packet scheduler is based on an efficient operation called Bit Reversal Permutation (BRP). We assume that there are N flows and the rate (r_i) of flow f_i is 2^{n_i} ($1 \leq i \leq N$). We sorts these N flows in descending order by their rate and allocate r_i consecutive timeslots in a frame to flow f_i in turns. The scheduling sequence acquired in this way is called *Bursty Sequence* (BS). As is shown in Figure 1, the capacity of outside link is 16, and there are 4 flows (f_1, f_2, f_3, f_4) with rate 4, 8, 2, 2 respectively. The flows are sorted into f_2, f_1, f_3, f_4 . And we allocate f_2 with 8 timeslots, f_1 with 4 timeslots, f_3 with 2 timeslots and f_4 with 2 timeslot in turns. The corresponding *Bursty Sequence* is shown in the top of Figure 1.

Then we make a *Bit Reversal Permutation* (BRP), which allocate the flow at the timeslot $(a_0 a_1 \dots a_{k-1})_2$ in BS with the timeslot $(a_{k-1} a_{k-2} \dots a_0)_2$ in *Uniform Sequence* (US). And we schedule flows according to US in BRP based scheduling. As is illustrated in Figure 1, the US transformed from BS through BRP is $f_2, f_1, f_2, f_3, f_2, f_1, f_2, f_4, f_2, f_1, f_2, f_3, f_2, f_1, f_2, f_4$. An obvious characteristic of US is that all the flows are spread out smoothly over the whole frame.

Lemma 1 (BRP): For a flow f with rate $r_f = 2^{n_f}$ in BRP scheduling, the number of packets it transmitted during time interval $[0, t]$

$$\frac{r_f t}{C} - 1 < S_f^{BRP}(t) < \frac{r_f t}{C} + 1$$

where $C = 2^k$.

Proof: For a flow f with rate 2^{n_f} , its timeslots in BS are from $a_0 a_1 \dots a_{k-n_f-1} 0 \dots 0$ to $a_0 a_1 \dots a_{k-n_f-1} 1 \dots 1$. In BRP scheduling, it will be scheduled at timeslot $0 \dots 0 a_{k-n_f-1} \dots a_1 a_0$, $0 \dots 1 a_{k-n_f-1} \dots a_1 a_0$, \dots , $1 \dots 1 a_{k-n_f-1} \dots a_1 a_0$ respectively. The scheduling interval is 2^{k-n_f} . Therefore, in a time range $[0, t]$, the flow can be

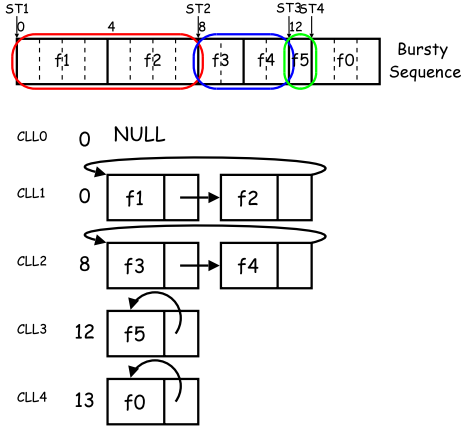


Figure 2: An example to show how HOBRP works

scheduled at least $\left\lfloor \frac{t}{2^{k-n_f}} \right\rfloor$ and at most $\left\lceil \frac{t}{2^{k-n_f}} \right\rceil$ times. So, we have

$$\frac{t}{2^{k-n_f}} - 1 < S_f^{BRP}(t) < \frac{t}{2^{k-n_f}} + 1$$

Since $C = 2^k$ and $r_f = 2^{n_f}$, we can get the lemma above. ■

III. HARDWARE OPTIMIZED BRP BASED SCHEDULER

Although BRP scheduler has bounded service curve $S_f^{BRP}(t)$, its hardware implementation complexity is too high. When a flow is added or deleted, it needs to recompute the BS and US. And the space complexity of BS is relatively high $O(2^k)$. In this section, we present a Hardware Optimized Bit Reversal Permutation (HOBRP) based scheduler which has low hardware implementation complexity and the same delay performance as BRP scheduler. Moreover, extended HOBRP is introduced to support flows with arbitrary rate (i.e. r_f can be an arbitrary positive integer). We start from the introduction of the novel structure used in HOBRP scheduling.

A. Circular Linked List Array (CLLA)

We divide the flows in BS into $k+1$ groups. The rates of the flows in the same group are the same integral power of two. All the flows in the same group are linked into a *Circular Linked List (CLL)*. As is shown in Figure 2, the capacity of outside link is 16, and there are 5 flows (f_1, f_2, f_3, f_4, f_5) with rate 4, 4, 2, 2, 1 respectively. In their corresponding BS, flows are divided into 3 groups: (f_1, f_2) with rate 4, (f_3, f_4) with rate 2, (f_5) with rate 1. The remain bandwidth is for best-effort flow f_0 which belongs to the last group. Thus, we have 4 non-empty CLLs. All the CLLs consists of *Circular Linked List Array (CLLA)*.

A formal description of CLLA is like this: *Circular Linked List Array (CLLA)* is an array of $k+1$ *Circular Linked Lists (CLLs)* denoted as $CLL_0, CLL_1, \dots, CLL_k$. CLL_i links all the flows with rate $r_f = 2^{k-i-1}$. And each CLL is associated with a *Starting Timeslot (ST)* and a *Scheduling Pointer (SP)*.

ST_i indicates the first timeslot of CLL_i in its corresponding BS. For example, in Figure 2, $ST_0 - ST_4$ are 0, 0, 8, 12, 13 respectively. SP_i points to the next flow to be scheduled in CLL_i . We define the *Range* of CLL_i as $[ST_i, ST_{i+1} - 1]$. Flow f_0 is denoted as best-effort flow and is always linked into CLL_k .

HOBRP scheduler contains a *Circular Counter (CC)* whose max value is 2^k . The initial value of CC is 0 and its value increases one per timeslot. At timeslot t , if the value of CC ($CC(t)$) is $(a_1 a_2 a_3 a_4)_2$, we first select the i th CLL (CLL_i) whose range contains $(a_4 a_3 a_2 a_1)_2$ by binary searching. And then schedule the flow pointed by SP_i . Then SP_i moves to the next node in CLL_i . For example, at time t , $CC = (0001)_2$, since $(1000)_2 = 8$ falls into CLL_2 's range $[8, 11]$. Suppose the flow to which SP_2 points is f_3 . Therefore, we choose f_3 to serve at time t , and SP_2 moves to the next flow f_4 in CLL_2 . According to the operations depicted above, the scheduling sequence in the example is $f_1, f_3, f_2, f_5, f_1, f_4, f_2, f_0, f_1, f_3, f_2, f_0, f_1, f_4, f_2, f_0$.

Lemma 2 (HOBRP): For a flow f with rate $r_f = 2^{n_f}$ in HOBRP scheduling, the number of packets it transmitted during time interval $[0, t]$

$$\frac{r_f t}{C} - 1 < S_f^{HOBRP}(t) < \frac{r_f t}{C} + 1$$

where $C = 2^k$.

Proof: According to HOBRP scheduling, a flow f with rate $r_f = 2^{n_f}$ will be linked into CLL_{k-n_f-1} . we suppose that there are m flows in CLL_{k-n_f-1} , and correspondingly the range of CLL_{k-n_f-1} is $[a2^{n_f}, (a+m)2^{n_f})$. According to HOBRP scheduling, CLL_{k-n_f-1} will be visited m times every 2^{k-n_f} time-slots. Correspondingly, flow f will be scheduled once every 2^{k-n_f} timeslots. So, we have

$$\frac{t}{2^{k-n_f}} - 1 < S_f^{HOBRP}(t) < \frac{t}{2^{k-n_f}} + 1$$

substitution of C and r_f , we can get the result above. ■

B. Extended HOBRP

In basic HOBRP scheduling depicted above, the value of rate r_f is constrained to an integral power of two, so we extend basic HOBRP scheduling to support flow f with rate r_f which can be an arbitrary positive integer in following two ways.

(Splitting Algorithm) For a flow f with rate $r_f = \sum_{l=1}^m 2^{n_l}$, $0 \leq n_1 < n_2, \dots, n_m \leq k-1$, it can be views as the composition of m separate subflows $\{f_1, \dots, f_m\}$ with rates $\{2^{n_1}, 2^{n_2}, \dots, 2^{n_m}\}$. So we can just serve these m subflows under basic HOBRP scheduling.

Lemma 3 (EHOBRP-SA): For a flow f with rate r_f (r_f is an integer and $0 < r_f < 2^k$) in Extended HOBRP scheduling, the number of packets transmitted under Splitting Algorithm (SA) during time interval $[0, t]$

$$\frac{r_f t}{C} - k < S_f^{H-SA}(t) < \frac{r_f t}{C} + k$$

Proof: Suppose the rate of flow f is $r_f = \sum_{l=1}^m 2^{n_l}$, $0 \leq n_1 < n_2, \dots, n_m \leq k-1$, and split flow f into m ($m \leq k$)

subflows $\{f_1, \dots, f_m\}$ with rates $\{2^{n_1}, 2^{n_2}, \dots, 2^{n_m}\}$. And we have

$$S_f^{H-SA}(t) = \sum_{l=1}^m S_{f_l}^{HOBRP}(t)$$

substitution of $S_{f_l}^{HOBRP}(t)$, we have

$$\frac{r_f t}{C} - m < S_f^{H-SA}(t) < \frac{r_f t}{C} + m$$

since $m \leq k$, we can get the lemma above. ■

(Over-allocation Algorithm) We allocate $R_f = 2^{n_f}$ timeslots to flow f with rate r_f in a frame under basic HOBRP scheduling, which satisfies $2^{n_f-1} < r_f \leq R_f = 2^{n_f}$ and use a deficit counter to decide whether to schedule a packet belongs to flow f or best effort flows f_0 at every allocated timeslot.

Allocate $R_f (= 2^{n_f})$ timeslots to flow f with rate r_f under basic HOBRP scheduling
Denote DC_f as the deficit counter of flow f . At timeslot t_0 , $DC_f = 0$
In every timeslot that has been allocated to flow f

- 1: $DC_f = DC_f + \frac{r_f}{R_f}$.
- 2: **if** $DC_f > 0$ **then**
- 3: serve a packet of flow f
- 4: $DC_f = DC_f - 1$
- 5: **else**
- 6: serve a packet of best effort flows f_0
- 7: **end if**

Figure 3: Over-allocation Algorithm

As is shown in Figure 3, in every timeslot that has been allocated to flow f under basic HOBRP scheduling, the deficit counter of flow f is first increased by r_f/R_f , and then if it is positive, we serve a packet of flow f and decrease its deficit counter by 1. Otherwise, we serve a packet that belongs to best effort flows.

Lemma 4 (EHOBPR-OA): For a flow f with rate r_f (r_f is an integer and $0 < r_f < 2^k$) in Extended HOBRP scheduling, the number of packets transmitted under Over-allocation Algorithm (OA) during time interval $[0, t]$

$$\frac{r_f t}{C} - \frac{r_f}{R_f} < S_f^{H-OA}(t) < \frac{r_f t}{C} + \frac{r_f}{R_f} + 1$$

Proof: According to the OA, we have

$$S_f^{H-OA}(t) \geq \left\lceil \frac{r_f}{R_f} \left\lfloor \frac{t}{C/R_f} \right\rfloor \right\rceil > \frac{r_f}{R_f} \left(\frac{t}{C/R_f} - 1 \right)$$

and

$$S_f^{H-OA}(t) \leq \left\lceil \frac{r_f}{R_f} \left\lfloor \frac{t}{C/R_f} \right\rfloor \right\rceil < \frac{r_f}{R_f} \left(\frac{t}{C/R_f} + 1 \right) + 1$$

After some manipulations, we can get the lemma above. ■

From the analysis above, we know that Over-allocation Algorithm (OA) has a tighter bound of service curve $S_f(t)$ than Splitting Algorithm (SA) which implies that OA has

a better performance bound than SA in terms of delay and fairness index¹. However, OA needs some timeslots to serve best effort flows which implies that the ratio of Guaranteed traffic to Overall volume (G/O) is less than 1. In the worst case, G/O can only achieve approximately $\min_f \{r_f/R_f\} \approx 1/2$. In order to make a tradeoff between G/O and performance, we further propose an i-Splitting and Over-allocation Algorithm (iSOA) which can provide a tunable performance bound.

(i-Splitting and Over-allocation Algorithm) For a flow f with rate $r_f = \sum_{l=1}^m 2^{n_l}$, $0 \leq n_1 < n_2, \dots, n_m \leq k-1$, iSOA defines that flow f can be split into at most i ($i < m$) subflows $\{f_1, \dots, f_i\}$ with rate $\{2^{n_m}, 2^{n_{m-1}}, \dots, 2^{n_{m-i+2}}, 2^{n_{m-i+1}+1}\}$. So the total timeslots allocated for flow f in iSOA is $R_f = \sum_{l=m-i+2}^m 2^{n_l} + 2^{n_{m-i+1}+1} > r_f$ and G/O is larger than r_f/R_f . We first allocate R_f timeslots to flow f with rate r_f in a frame under SA scheduling. Then we use a deficit counter to decide whether to serve flow f or best-effort flow f_0 in every allocated timeslot, which is the same as OA.

Lemma 5 (EHBRP-iSOA): For a flow f with rate r_f (r_f is an integer and $0 < r_f < 2^k$) in Extended HOBRP scheduling, the number of packets transmitted under i-Splitting and Over-allocation Algorithm (iSOA) during time interval $[0, t]$

$$\frac{r_f t}{C} - i \frac{r_f}{R_f} < S_f^{H-iSOA}(t) < \frac{r_f t}{C} + i \frac{r_f}{R_f} + 1 \quad (1)$$

Proof: According to the iSOA, we have

$$\begin{aligned} S_f^{H-iSOA}(t) &\geq \left\lceil \frac{r_f}{R_f} \left(\sum_{l=1}^{l=i} \left\lfloor \frac{t}{C/R_l} \right\rfloor \right) \right\rceil \\ &> \frac{r_f}{R_f} \left(\frac{t}{C/R_f} - i \right) \end{aligned}$$

and

$$\begin{aligned} S_f^{H-iSOA}(t) &\leq \left\lceil \frac{r_f}{R_f} \left(\sum_{l=1}^{l=i} \left\lfloor \frac{t}{C/R_f} \right\rfloor \right) \right\rceil \\ &< \frac{r_f}{R_f} \left(\frac{t}{C/R_f} + i \right) + 1 \end{aligned}$$

After some simplifications, we can get the lemma above. ■

Theorem 1 (G/O): For a flow f with rate $r_f = \sum_{l=1}^m 2^{n_l}$ ($0 \leq n_1 < n_2, \dots, n_m \leq k-1$) in Extended HOBRP scheduling, the ratio of Guaranteed traffic to Overall volume (G/O) of flow f under iSOA

$$1 - \frac{1}{2^i} \leq (G/O)_f \leq 1$$

Proof: According to the iSOA, we have

$$\begin{aligned} (G/O)_f &= \frac{r_f}{R_f} = \frac{\sum_{l=1}^m 2^{n_l}}{\sum_{l=m-i+2}^m 2^{n_l} + 2^{n_{m-i+1}+1}} \\ &= 1 - \frac{2^{n_{m-i+1}} - \sum_{l=1}^{m-i} 2^{n_l}}{\sum_{l=m-i+2}^m 2^{n_l} + 2^{n_{m-i+1}+1}} \geq 1 - \frac{2^{n_{m-i+1}}}{2^{n_{m+1}}} \geq 1 - \frac{1}{2^i} \end{aligned}$$

¹The details of its performance analysis will be discussed in Section IV

and obviously, $G/O = r_f/R_f \leq 1$. Hence, proved. ■

Form Lemma 5 and Theorem 1, we have following conclusions: In EHOBRRP-iSOA scheduling, if a flow can be split into more subflows (i.e the parameter i of iSOA is greater), the G/O of the flow can be higher, which means more bandwidth is available for guaranteed traffic. At the extreme case, when a flow can be split into at most k flows (i.e. $i = k$), iSOA will be degraded into SA. On the contrary, if a flow can be split into less subflows, (i.e the parameter i of iSOA is smaller), the bound of serve curse $S_f(t)$ will be tighter, which implies a better performance bound in terms of delay and fairness. In the extreme situation, when a flow can not be split (i.e. $i = 1$), iSOA will be degraded into OA.

C. Description of HOBRP

In this section, we give a formal and complete description of HOBRP scheduling. In the rest of the paper, we refer to ‘‘Extended HOBRP with iSOA’’ as HOBRP. In HOBRP scheduler, we assume that a flow is added into the scheduler by a *Call Admission Controller* (CAC) and removed from the scheduler by a signaling protocol, and packets are classified into different flows by a packet classifier. And we allocate the unallocated bandwidth to the best-effort flow f_0 .

Definition 2: For a k bit binary number $i = a_{k-1}a_{k-2}\dots a_0$, where $a_j \in \{0, 1\}$ and $0 \leq j < k$, its bit reversal number is defined as

$$BR(i, k) = a_0a_1\dots a_{k-1}$$

From the definition, we get $BR((011)_2, 3) = (110)_2 = 6$ and $BR((0001)_2, 4) = (1000)_2 = 8$.

The scheduling procedure is like this (Figure 4): When there are packets in system (line 1), Scheduler selects the CLL whose range contains the value of $BR(CC, k)$ through binary searching (line 2-10). Then it selects the flow f pointed by the corresponding SP_{low_val} (line 11). Next, the deficit counter DC_f of flow f increases by r_f/R_f (line 12). If DC_f is positive (line 13), we uses *ServeFlow* to transmit a packet for flow f (line 14) and decrease DC_f by one (line 15). Otherwise, we serve a packet belongs to best-effort flow f_0 (line 17). Line 19 and 20 are to update the pointers SP of selected CLL and Circular Counter (CC) of the HOBRP scheduler, respectively.

Add_Flow is invoked when a new flow is accepted by the CAC controller. It examines all the $CLLs$ in turns. For CLL_j , *Add_Flow* insert flow f to the CLL if its coefficient of 2^{k-j-1} is greater than 0 (i.e. equals to 1) (line 3-4). At the same time, ST_{j+1} is updated according to the value of R_f (line 6).

When a flow leaves the scheduler, *Del_Flow* is called. *Del_Flow* examines all the $CLLs$ in turns. For CLL_j , flow f , if it exists, is removed from CLL_j (line 3-4). And ST_i is updated according to the value of R_f at the same time (line 6).

IV. PROPERTIES OF HOBRP

This section analyzes the properties of HOBRP. In particular, we are interested in the fairness, delay and its implementation complexities.

For flow f with rate r_f ,

R_f : the number of timeslots allocated to flow f per frame under iSOA.

DC_f : the deficit counter of flow f

ST_i : the starting timeslot of CLL_i

SP_i : the scheduling pointer of CLL_i

CC : the circular counter of the scheduler

Schedule:

```

1: while the scheduler in busy period do
2:   low_val = 0
3:   high_val = k + 1
4:   for i = 1 to ⌈log(k + 1)⌉ do
5:     if BR(CC, k) ≥ ST⌊ $\frac{low\_val+high\_val}{2}$ ⌋ then
6:       low_val = ⌊ $\frac{low\_val+high\_val}{2}$ ⌋
7:     else
8:       high_val = ⌊ $\frac{low\_val+high\_val}{2}$ ⌋
9:     end if
10:  end for
11:  f = SPlow_val → flow
12:  DCf = DCf + rf/Rf
13:  if DCf > 0 then
14:    ServeFlow(f)
15:    DCf = DCf - 1
16:  else
17:    ServeBestEffortFlow
18:  end if
19:  SPi = SPi → next
20:  CC = (CC + 1) % 2k
21: end while

```

Add_Flow(f, R_f)

```

1: Rf = ∑i=0k-1 bi2i, where bi ∈ {0, 1}
2: for j = 0 to k - 1 do
3:   if bk-j-1 > 0 then
4:     Insert f to the CLLj
5:   end if
6:   STj+1+ = ∑i=k-j-1k-1 bi2i
7: end for

```

Del_Flow(f, R_f)

```

1: Rf = ∑i=0k-1 bi2i, where bi ∈ {0, 1}
2: for j = 0 to k - 1 do
3:   if bk-j-1 > 0 then
4:     Delete f from CLLj
5:   end if
6:   STj+1- = ∑i=k-j-1k-1 bi2i
7: end for

```

Figure 4: Description of the HOBRP packet scheduler

A. Fairness

1) *Proportional Fairness Index (PFI)*: Proportional Fairness Index (PFI) [8] essentially requires that the difference between the normalized service received by any two backlogged flows f_i and f_j , over any time period (t_1, t_2) , be bounded by

a small constant.

Theorem 2 (PFI): In any time period (t_1, t_2) during which flows f_i and f_j are backlogged,

$$PFI = \left| \frac{S_i(t_1, t_2)}{r_i} - \frac{S_j(t_1, t_2)}{r_j} \right| < \frac{2m}{R_i} + \frac{2n}{R_j} + \frac{1}{r_i} + \frac{1}{r_j}$$

where m, n are the parameters of HOBRP for flow f_i, f_j respectively.

Proof: Form the definition of $S(t)$, we know $S(t_1, t_2) = S(t_1) - S(t_2)$, substitution of $S(t)$ of (1) into the equation, we have

$$\frac{r_i(t_1 - t_2)}{C} - 2m \frac{r_i}{R_i} - 1 < S_i(t_1, t_2) < \frac{r_i(t_1 - t_2)}{C} + 2m \frac{r_i}{R_i} + 1$$

in the same way,

$$\frac{r_j(t_1 - t_2)}{C} - 2n \frac{r_j}{R_j} - 1 < S_j(t_1, t_2) < \frac{r_j(t_1 - t_2)}{C} + 2n \frac{r_j}{R_j} + 1$$

hence,

$$\left| \frac{S_i(t_1, t_2)}{r_i} - \frac{S_j(t_1, t_2)}{r_j} \right| < \frac{2m}{R_i} + \frac{2n}{R_j} + \frac{1}{r_i} + \frac{1}{r_j}$$

2) *Worst-case Fairness Index (WFI):* Worst-case Fairness Index (WFI) [6] is more refined notion of fairness. Rather than comparing the relative amounts of service received by two flows f_i and f_g , it compares the service received by a single flow f_i to the service it would receive in the ideal case.

Theorem 3 (WFI): In HOBRP scheduling, the Worst-case Fairness Index (WFI)

$$WFI = \frac{S_f^{id}(t) - S_f^{HOBRRP}(t)}{r_f} < \frac{i}{R_f} + \frac{1}{r_f}$$

Proof: Substitution of $S_f^{id}(t) = r_f t / C$, and $S_f^{HOBRRP}(t)$ of (1) into the definition of WFI, we can easily get the theorem above. ■

B. Delay

1) Local Delay:

Theorem 4 (Local delay): For a flow f with rate r_f (r_f is an integer and $0 < r_f < 2^k$) in HOBRRP-iSOA scheduling, its delay bound

$$d_f^{HOBRRP} < \frac{iC}{R_f}$$

Proof: Let d_f^{HOBRRP} be the bound delay of f between ideal scheduler and BRP scheduler, which means a packet served at time t in ideal scheduler will be served at time $t + d_f^{HOBRRP}$ in BRP scheduler.

$$S_f^{id}(t) = S_f^{HOBRRP}(t + d_f^{HOBRRP})$$

Substitution of $S_f^{HOBRRP}(t)$ of (1) into the equation above, after some manipulations, yields

$$\begin{aligned} \frac{r_f(t + d_f^{HOBRRP})}{C} - i \frac{r_f}{R_f} &< S_f^{id}(t) \\ \Rightarrow \frac{r_f(t + d_f^{HOBRRP})}{C} - i \frac{r_f}{R_f} &< \frac{r_f t}{C} \\ \Rightarrow d_f^{HOBRRP} &< \frac{iC}{R_f} \end{aligned}$$

From the theorem above, we know that the delay bound of HOBRRP is tunable. If a network application like VoIP is very delay sensitive, we can use the scheduler with a small parameter i to achieve a low delay bound at the cost of low G/O. At the extreme case ($i = 1$), the local GPS-relative delay d_f^{HOBRRP} of flow f is less than $\frac{C}{R_f} < \frac{C}{r_f}$. This implies that flow f in HOBRRP scheduling is at most one packet lagged behind the same flow in ideal GPS scheduling, which is the best delay bound among practical schedulers.

2) End-to-end delay:

Corollary 1 (Bounded end-to-end delay): When a flow with reserved rate r is constrained by a leaky bucket (σ, r) and served by a network of M cascading HOBRRP nodes, its end-to-end delay bound is

$$D \leq \frac{\sigma}{r} + \sum_{i=1}^M d(i)$$

where $d(i)$ denotes the delay bound at node i and is calculated from Theorem 4.

C. Time and Space Complexities

Theorem 5: Time and Space complexities of HOBRRP scheduler are :

- 1) HOBRRP needs $O(\log k)$ to choose a packet for transmission.
- 2) HOBRRP needs $O(k)$ to add a flow and $O(k)$ to delete a flow.
- 3) The space complexity of HOBRRP is $O((k+1)N)$.

where k is the order of link bandwidth.

Proof: First of all, in *Schedule* procedure, line 6 and line 8 are executed $\lceil \log(k+1) \rceil$ times in all. Line 11, 12, 19 and 20 are executed one time respectively. And line 14 and 15 or line 17 is executed one time in a scheduling. So, *Schedule* needs $\lceil \log(k+1) \rceil + 6$ operations to serve a flow. The time complexity of *Schedule* procedure is $O(\log k)$.

Secondly, in both *Add_Flow* and *Del_Flow* procedures, line 4 is executed at most k times and line 6 is executed k times. So, both *Add_Flow* and *Del_Flow* need at most $2k$ operations to add or delete a flow, the time complexities of these two procedures are $O(k)$.

Thirdly, since a flow can be inserted into at most k CLLs, the whole structure of CLLA needs at most kN space. And HOBRRP still need k STs, k SPs, N DCs and a CC. So the total space needed by HOBRRP is $(k+1)N + 2k + 1$ (i.e. $O((k+1)N)$). ■

V. HARDWARE IMPLEMENTATION

In this section, we provide a hardware design scheme of HOBRP and demonstrate that our scheduler is very suitable for high-speed hardware implementation.

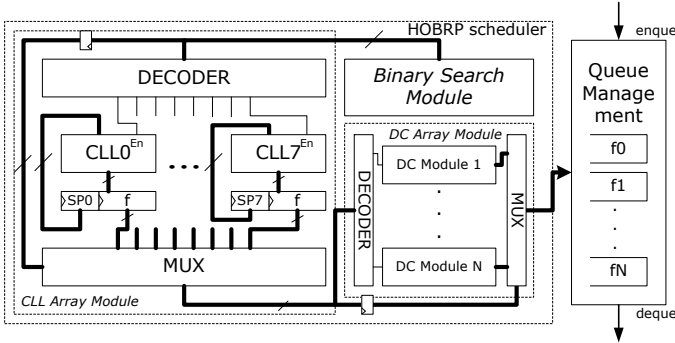


Figure 5: Overview of hardware design proposal of HOBRP

As is shown in Figure 5, the whole system consists of two parts: a Queue Management (QM) and a HOBRP scheduler. In QM, each queue is mapped to a flow. If a packet of flow f arrives, it will be appended to Queue Q_f . And when the output link is idle, QM will select the queue Q_f indicated by HOBRP scheduler, and send the packet at the head of Q_f to the outside link.

HOBRP scheduler is composed of three modules: Binary Search Module (BSM), Circular Linked List Array Module (CLLAM) and Deficit Counter Array Module (DCAM). As the start of HOBRP, BSM selects the CLL according to the comparison between $BR(CC, k)$ and k STs by binary searching. And CLLAM exports the flow number f pointed by SP of the selected CLL . At last, DCAM decides whether to serve a guaranteed flow f or best-effort flow f_0 on the basis of the value of DC_f .

A. Binary Search Module (BSM)

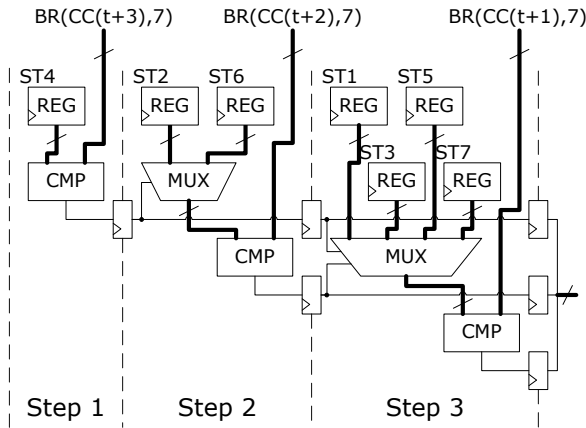


Figure 6: Hardware design of Binary Search Module

There are k STs and a CC in BSM. Figure 6 shows the inside structure of a BSM with $k = 7$. The whole

searching process can be pipelined. We suppose $CC = (a_6 a_5 a_4 a_3 a_2 a_1 a_0)_2$. In the first cycle, $(a_0 a_1 a_2 a_3 a_4 a_5 a_6)_2$ is compared with ST_4 . According to the comparison result of step 1, $(a_0 a_1 a_2 a_3 a_4 a_5 a_6)_2$ is compared with ST_2 or ST_6 in the second cycle. And in the third cycle, $(a_0 a_1 a_2 a_3 a_4 a_5 a_6)_2$ is compared with one of ST_1, ST_3, ST_5 and ST_7 according to the comparison results of step 1 and step 2. The combination of three comparison results is the output of the BSM, which represents a CLL number.

If a flow f is added or deleted, all STs will be updated in three steps, which coordinates with pipelined searching process. First step is to change ST_4 according to the rate of flow f . Second, ST_2 and ST_6 . Third, ST_1, ST_3, ST_5 and ST_7 .

B. CLL Array Module (CLLAM)

CLLAM contains k CLL s and only one CLL will be selected at a time. A CLL can be considered as a RAM which contains N units. Each unit of CLL consists of a flow number and the address of the next unit (i.e. SP). In each cycle, the CLL chosen by the decoder will replicate the content of the unit pointed by SP to the corresponding register. And flow number in the register will be exported to DCAM in the next cycle. Flow adding or deleting process is just to insert or delete nodes in the corresponding CLL .

C. DC Array Module (DCAM)

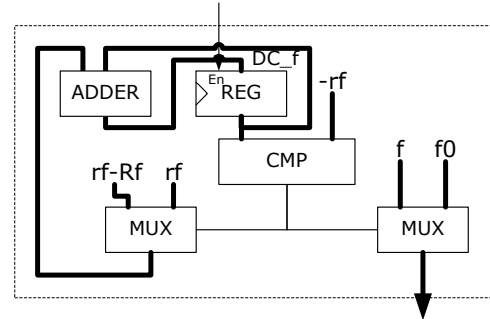


Figure 7: Hardware design of Deficit Counter Module

In DCAM, there is an array of Deficit Counter Modules (DCM_1, \dots, DCM_N). Like CLLAM, DCAM selects a DCM in a cycle, and exports the output of selected DCM to the QM in the next cycle. DCM_f is used to decide whether to serve flow f or best-effort flow f_0 in the allocated timeslot. Figure 7 shows the inside structure of DCM. In every cycle, DC_f is compared with $-r_f$. If $DC_f > -r_f$, $DC_f = DC_f + (r_f - R_f)$ and flow number f is exported. Otherwise, $DC_f = DC_f + r_f$ and meantime the module exports flow number f_0 . The value of $DC_f, -r_f, r_f - R_f, R_f$ will be initialized when a flow f is added or deleted.

From the synchronous sequential circuit depicted above, we can see that HOBRP scheduler is very easy to be pipelined and paralleled. This makes the operations in a cycle simple

enough to be finished in several nanoseconds by hardware². Therefore, we believe that our HOBPR is an ideal alternative scheduler for high speed network interface with large amounts of application flows.

VI. SIMULATION

We have implemented HOBPR in NS2 [22] and compare it with two other schedulers: WF^2Q [6] and SRR [17]. WF^2Q is chosen because it is an example of a time-stamp based scheduler with provably good delay properties (almost identical service to GPS differing by no more than one maximum size packet). While SRR is chosen because it is a round-robin based scheduler with comparable implementation complexity as HOBPR.

In the network as depicted in Figure 8, the bandwidths and propagation delays of R_0R_1 and R_1R_2 are 10 Mb/s and 10 ms, respectively. The bandwidths and delays are 100 Mb/s and 1 ms for the rest links. The MTU of the network is 250 bytes and the reserved rate of a flow is set to its sending rate.

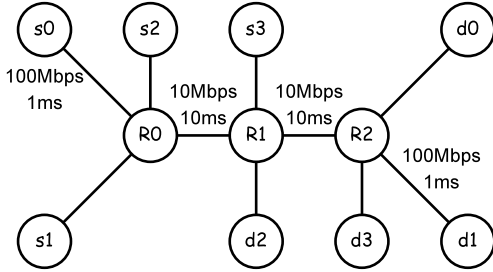
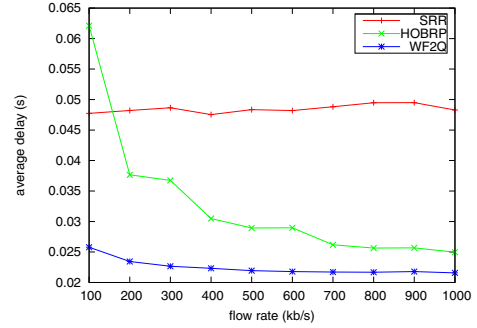


Figure 8: Network topology of the simulation

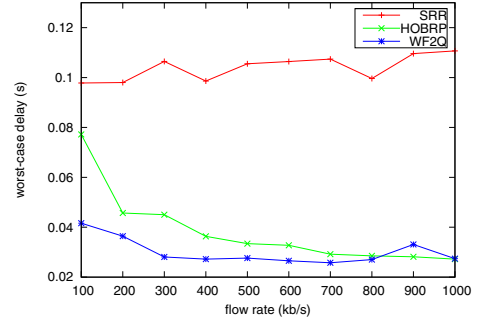
We setup a CBR flow f_1 between s_0 and d_0 and 500 CBR flows with rate 16 Kb/s from s_1 to d_1 . There are also two best-effort flows (f_0) from s_2 to d_2 and s_3 to d_3 to occupy the unallocated bandwidth. The first one is an exponential on/off flow with mean on and off time 100ms. The second one is generated from a Pareto source with mean on and off time 100 ms, and shape parameter $\alpha = 1.5$. The average rate of both flows are 2Mb/s, which is larger than the unallocated bandwidth of the network.

In the first experiment, we measured the average and worst-case end-to-end delay of flow f_1 whose reserved rate is varied from 100Kb/s to 1000Kb/s in step of 100Kb/s under the HOBPR (1-SOA), SRR and WF^2Q . From Figure 9, we can see that both average and worst-case delay of SRR are relatively insensitive to the reserved bandwidth of the flow. This is because its delay bound is not only in inverse proportion to the weight of the flow, but also in direct proportion to the total number of active flows in SRR [17]. In the case of HOBPR, the delays are inversely proportional to the reserved rate of the flow, i.e., the higher the reserved rate, the lower the average and worst-case delays. It can be seen from Figure 9 that both average and worst-case delays of HOBPR are much better than that of SRR and closely mirror that of WF^2Q . This is despite

²The DECODER and MUX shown in Figure 5 can be further pipelined.



(a) Average delay



(b) Worst-case delay

Figure 9: End-to-end delay of SRR, HOBPR and WF^2Q

the fact that the implementation complexity of HOBPR is quite low as discussed earlier.

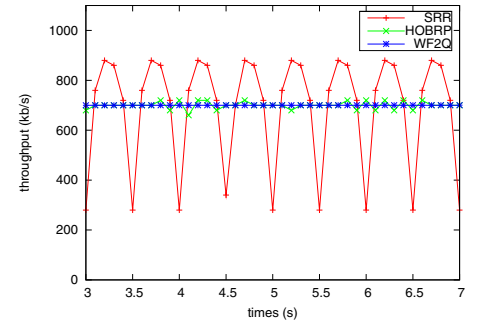


Figure 10: Short-term throughput of SRR, HOBPR and WF^2Q

The second experiment is to demonstrate the short-term throughput property of HOBPR (1-SOA). Figure 10 shows the short-term throughput of flow f_1 (700Kb/s) with different scheduling schemes. Each point in Figure 10 represents the throughput in an interval of 100ms. It can be seen from Figure 10 that the short-term throughput for SRR exhibits heavy fluctuations. The flow may significantly under-perform (down to 300Kb/s) or over-perform (up to 900 Kb/s) during some intervals. On the other hand, WF^2Q and HOBPR yield much better short-term throughputs: within each interval of 100ms, the throughputs are always close to the ideal rate. This experiment demonstrates that HOBPR has a better short-term

throughput property than SRR and is immune to impacts of many low-speed flows on the high-speed flows.

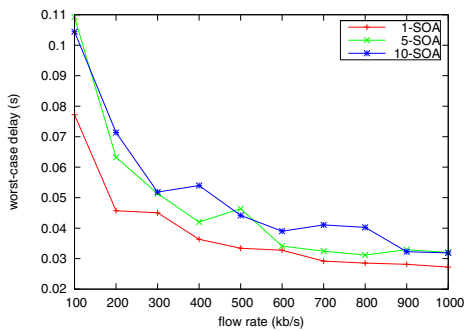


Figure 11: Worst-case end-to-end delay of HOBRP with different i-SoA

In the third experiment, we observe the worst-case delay of flow f_1 under HOBRP (i-SoA) with different parameter i . It can be seen from Figure 11 that HOBRP with a smaller i yields a lower worst-case delay. This is consistent with the theoretical result that the delay bound of HOBRP is proportional to i .

VII. CONCLUSION

In this paper, we proposed a novel framed based packet scheduler called Hardware Optimized Bit Reserved Permutation (HOBRP) based scheduler. Our HOBRP is based on a simple Bit Reversal Permutation that maps Bursty Sequence (BS) into Uniform Sequence (US). And US is an ideal scheduling sequence for the flows in the system. Moreover, by using Circular Linked List Array to replace BS structure, HOBRP not only reduces the space complexity, but also simplifies the operations of adding or deleting a flow.

HOBRP provides almost all the performance guarantee that an ideal scheduler should have: end-to-end delay bound, constant proportional and worst-case fairness index. All these performance guarantees were proved by both theoretical analysis and simulation comparisons.

In addition to these common merits mentioned above, HOBRP still owns following two unique features.

(tunable end-to-end delay bound) With *i-Splitting and Over-allocation Algorithm* (iSOA), HOBRP is able to make a tradeoff between performance bound and available bandwidth for guaranteed traffic (i.e the ratio of guaranteed traffic to overall volume, G/O). Through sacrificing some G/O, HOBRP can provide an optimum delay bound among practical packet schedulers, which makes itself flexible enough to deal with various of application flows with diverse performance requirements.

(intuitive hardware design scheme) We exhibit a synchronous sequential circuit design of HOBRP. By pipelining and paralleling, we demonstrate that the operations in each cycle are simple and can be finished within several nanoseconds. So we believe that our HOBRP is an ideal alternative scheduler for high-speed network interface with tens of thousand of flows.

REFERENCES

- [1] D. Black, S. Blake, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated services (RFC 2475)," 1998.
- [2] R. Braden, D. Clark, and S. Shenker, "Integrated services in the internet architecture: an overview (RFC 1633)," 1994.
- [3] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," *SIGCOMM Comput. Commun. Rev.*, vol. 19, no. 4, pp. 1–12, 1989.
- [4] A. Parekh and R. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: the single-node case," *Networking, IEEE/ACM Transactions on*, vol. 1, no. 3, pp. 344–357, 1993.
- [5] L. Zhang, "A new architecture for packet switching network protocols," Ph.D. dissertation, Dept. Elect. Eng. and Comput. Sci., M.I.T., 1989.
- [6] J. Bennett and H. Zhang, "WF2Q: worst-case fair weighted fair queueing," in *INFOCOM '96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation. Proceedings IEEE*, vol. 1, 1996, pp. 120–128 vol.1.
- [7] P. Valente, "Exact GPS simulation and optimal fair scheduling with logarithmic complexity," *Networking, IEEE/ACM Transactions on*, vol. 15, no. 6, pp. 1454–1466, 2007.
- [8] S. Golestani, "A self-clocked fair queueing scheme for broadband applications," in *INFOCOM '94. Networking for Global Communications., 13th Proceedings IEEE*, 1994, pp. 636–646 vol.2.
- [9] P. Goyal, H. Vin, and H. Cheng, "Start-time fair queueing: a scheduling algorithm for integrated services packet switching networks," *Networking, IEEE/ACM Transactions on*, vol. 5, no. 5, pp. 690–704, 1997.
- [10] D. Stiliadis and A. Varma, "Efficient fair queueing algorithms for packet-switched networks," *Networking, IEEE/ACM Transactions on*, vol. 6, no. 2, pp. 175–185, 1998.
- [11] J. Cobb, M. Gouda, and A. El-Nahas, "Time-shift scheduling-fair scheduling of flows in high-speed networks," *Networking, IEEE/ACM Transactions on*, vol. 6, no. 3, pp. 274–285, 1998.
- [12] J. Xu and R. Lipton, "On fundamental tradeoffs between delay bounds and computational complexity in packet scheduling algorithms," *Networking, IEEE/ACM Transactions on*, vol. 13, no. 1, pp. 15–28, 2005.
- [13] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round-robin," *Networking, IEEE/ACM Transactions on*, vol. 4, no. 3, pp. 375–385, Jun. 1996.
- [14] L. Lenzini, E. Mingozzi, and G. Stea, "Aliquem: a novel DRR implementation to achieve better latency and fairness at $o(1)$ complexity," in *Quality of Service, 2002. Tenth IEEE International Workshop on*, 2002, pp. 77–86.
- [15] D. Saha, S. Mukherjee, and S. Tripathi, "Carry-over round robin: a simple cell scheduling mechanism for ATM networks," *Networking, IEEE/ACM Transactions on*, vol. 6, no. 6, pp. 779–796, 1998.
- [16] R. Garg and X. Chen, "RRR: recursive round robin scheduler," in *Global Telecommunications Conference, 1998. GLOBECOM 98. The Bridge to Global Integration. IEEE*, vol. 1, 1998, pp. 422–432 vol.1.
- [17] C. Guo, "SRR: an $o(1)$ time-complexity packet scheduler for flows in multiservice packet networks," *Networking, IEEE/ACM Transactions on*, vol. 12, no. 6, pp. 1144–1155, 2004.
- [18] S. Cheung and C. Pencea, "BSFQ: bin sort fair queueing," in *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3, 2002, pp. 1640–1649 vol.3.
- [19] B. Caprita, W. C. Chan, J. Nieh, C. Stein, and H. Zheng, "Group ratio round-robin: $O(1)$ proportional share scheduling for uniprocessor and multiprocessor systems," in *Proceedings of the annual conference on USENIX Annual Technical Conference*. Anaheim, CA: USENIX Association, 2005, pp. 36–36. [Online]. Available: <http://portal.acm.org/citation.cfm?id=1247396>
- [20] X. Yuan and Z. Duan, "FRR: a proportional and worst-case fair round robin scheduler," *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 2, pp. 831–842 vol. 2, Mar. 2005.
- [21] S. Ramabhadran and J. Pasquale, "The stratified round robin scheduler: Design, analysis and implementation," *Networking, IEEE/ACM Transactions on*, vol. 14, no. 6, pp. 1362–1373, 2006.
- [22] NS2, "The NS network simulator." [Online]. Available: <http://www.isi.edu/nsnam/ns>