ELSEVIER

Computer Networks 35 (2001) 287–305

COMPUTER NETWORKS

www.elsevier.com/locate/comnet

Pre-order Deficit Round Robin: a new scheduling algorithm for packet-switched networks [☆]

Shih-Chiang Tsao [a,1], Ying-Dar Lin [b,*]

[a] *Telecommunication Laboratories, Chunghwa Telecom Co., Ltd, Taoyuan, Taiwan*

[b] *Department of Computer and Information Science, National Chiao Tung University, 1001 Ta Hsueh Road, Hsinchu 30050, Taiwan*

Received 11 February 2000; accepted 31 July 2000

Responsible Editor: E. Knightly

Abstract

In recent years, many packet fair queueing algorithms have been proposed to approximate generalized processor sharing (GPS). Most of them provide a low end-to-end delay bound and ensure that all connections share the link in a fair manner. However, scalability and simplicity are two significant issues in practice. Deficit Round Robin (DRR) requires only O(1) work to process a packet and is simple enough to be implemented in hardware. However, its large latency and unfair behavior are not tolerated. In this work, a new scheme, Pre-order Deficit Round Robin, is described, which overcomes the problems of DRR. A *limited* number, Z, of *priority queues* are placed behind the DRR structure to reorder the transmission sequence to approximate packet by packet generalized processor sharing (PGPS). We provide an analysis on latency and fairness, which shows our scheme as a better alternative to DRR. In most cases PDRR has a per-packet time complexity of O(1), and O(log Z) in other specific cases. Simulation results are also provided to further illustrate its average behavior. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: Packet scheduling; Fair queueing; Round robin; Deficit

1. Introduction

In recent years, many packet scheduling algorithms, which aim to approximate generalized processor sharing (GPS) [1] have been proposed [2–7]. Stiliadis and Verma [8] presented a broad class of schedulers to describe their common architecture and provided a systematic analysis regarding their latency and fairness properties. Generally, for the class of schedulers, there are two major operations, which are the key factors in determining the implementation complexity. One is the maintainence of virtual time and the other selects the most eligible packet to send out next among all active flow queues.

[☆] This work was supported in part by MOE Program of Excellence Research, 89-E-FA04-1-4.

[*] Corresponding author. Tel.: +886-3-573-1899; fax: +886-3-572-1490.

E-mail addresses: weafon@cis.nctu.edu.tw (S.-C. Tsao), ydlin@cis.nctu.edi.tw (Y.-D. Lin).

[1] Tel.: +886-3-424-5085.

To reduce the complexity of the above two major operations in this class of schedulers, many scheduling algorithms were proposed [3–7,9–11]. Some algorithms can reduce complexity to O(1) [9–11]. However, their schemes and analytical results can only be applied to fixed packet sizes and, hence, are only suitable for ATM networks. Others can handle variable-length packets and offer good scheduling properties such as fairness and latency. However their complexities are still dependent on N, e.g., $O(\log N)$, where N is the number of flows [3,4,7]. Among them, Deficit Round Robin (DRR) [5] (a credit-based version of Round Robin) is an extreme case. It requires only O(1) work to process a packet and is amenable to variable-length packets. But its latency and fairness [8] are higher than others, such as self-clocked fair queueing (SCFQ) [3] which has $O(\log N)$ per-packet time complexity. How it performs these two operations with only O(1) work is most interesting. In contrast, the problems cause DRR's intolerable properties in latency and fairness which are what we have attempted to overcome.

Herein, a new scheduling algorithm, Pre-order Deficit Round Robin (PDRR), is proposed whose objective is to solve the problems of DRR while keeping the advantage of O(1) per-packet time complexity in most cases and remaining amenable to variable-length packets. This goal is achieved through two improvements. First, a *Pre-order Queueing* module is appended to the original architecture of DRR. The new design solves the phenomenon of small packets waiting too long to be transmitted, which is caused by sending large packets at inappropriate time. Secondly, the *quantum update* operation was separated from the *dequeue* operation of DRR and thereby enable the packet to be considered for sending out in this round upon arrival. In the original DRR, the packet is refrained from being sent out until the flow's turn arrives. In the worst scenario, it might just miss the opportunity and therefore not be sent out in this round. The above two improvements enable the server to *reorder* the transmission sequence of DRR in one round and send out a packet in an eligible time as early as it may.

Next, we analyzed our PDRR algorithm with respect to three measures, including latency, fairness and per-packet time complexity. The analysis confirms that PDRR offers better performance in latency and fairness than DRR and lower time complexity than SCFQ. Finally, through simulation results, we also demonstrate the average behavior of PDRR.

The rest of the paper is organized as follows. Section 2 illustrates the problems of DRR. Section 3 presents a new scheme, PDRR, to resolve these problems. Section 4 gets the analytical results of PDRR on latency, fairness, and per-packet time complexity, and Section 5 presents simulation results demonstrating the average behaviors. Section 6 describes some related works and finally, in Section 7 we summarize our work and illustrate future directions.

2. Motivation

2.1. Deficit Round Robin

DRR [5], proposed by Shreedhar and Varghese is a simple scheduling algorithm. Herein, its structure is conceptually depicted. For additional details, readers are referred to [5]. Examples to explain what its problems are and solutions are also provided.

The server in DRR *rotationally* selects packets to send out from all flows that have queued packets. DRR maintains a *service list* to keep the flow sequence being served in a *round* and to avoid examining empty queues. If a flow has no packets in its queue, its identifier will be deleted from the service list. The next time a packet arrives to the flow that has an empty queue, the identifier of the flow will be added to the tail of the list.

We now come to see the number of packets can that be sent once a flow is served. For each flow, two variables, *Quantum* and *DeficitCounter*, are maintained. *Quantum* is the amount of credits in byte allocated to a flow within the period of one round. $Quantum_i$ for flow i can be derived as

$$Quantum_i = \frac{r_i}{C} \times F, \tag{1}$$

where r_i is the rate allocated to flow *i*, *C* the link service rate, and *F* is the frame size that represents the summation of *Quantum*'s for all flows. $DeficitCounter_i^{j-1}$ accumulates the residual $Quantum_i$ of flow *i* in the $(j-1)$th round. The next time that flow *i* is served, it can send out additional $DeficitCounter_i^{j-1}$ bytes of data in the *j*th round. By this rule, once flow *i* is served, two steps are performed. First, the server updates $DeficitCounter_i^j$ as

$$DeficitCounter_i^j = DeficitCounter_i^{j-1} + Quantum_i.$$

Second, it verifies the size of the packet at the queue head of flow *i*. If the size is smaller than $DeficitCounter_i^j$, $DeficitCounter_i^j$ is decreased by this packet size and the packet is sent out. The server repeats to execute this operation until the size of the head packet is larger than $DeficitCounter_i^j$, that is there are insufficient credits to serve the subsequent packet, or there are no remaining packets in the queue of flow *i*. In the former case, the time it takes to transmit this packet is delayed and the residual value in $DeficitCounter_i^j$ is held until the subsequent turn arrives. Again, the next time flow *i* gets its turn, it can send out additional $DeficitCounter_i^j$ bytes of data in addition to $Quantum_i$ bytes. In the latter case, $DeficitCounter_i^j$ is reset to zero, that is the residual credits remaining from the previous round cannot be carried over to serve the following burst. Or it may delay the service to other flows.

2.2. Quantum size

Deciding *Quantum* size is an important issue. According to Shreedhar and Varghese [5], if we expect that the work for DRR is O(1) per packet, then the *Quantum* for a flow should be larger than the maximum packet size within the flow so that at least one packet per backlogged flow can be served in a round. Besides, according to the meaning of *Quantum*, for any two flows *i* and *j*, we have

$$\frac{Quantum_i}{Quantum_j} = \frac{r_i}{r_j}. \tag{2}$$

To satisfy the above two constraints for all flows, the *Quantum* of a flow might be very large and many times its maximum packet size.

An example is used to further illustrate the above problem. Assume that there are four flows, sharing the same link, whose traffic parameters are depicted in Table 1 and the link capacity is 160 Mbps (megabits per second). Let us consider the determination of *Quantum*. Under the above two constraints, flow *B* that has the maximum ratio of maximum packet size to reserved rate was selected as the base and its *Quantum* was set to its maximum packet size, 640. The *Quantum* of alternate flows are set according to (2). Notably, the *Quantum* size of flow *D* is 25.6 times of the maximum packet size of flow *D*.

Table 1
The traffic parameters and quantum size of four flows

Flow ID	Reserved rate (Mbps)	Traffic type	Maximum packet size (byte)	Ratio of max packet size to reserved rate	Quantum size (byte)
A	12.8	CBR	400	250	512
B	16	CBR	640	320	640
C	64	CBR	800	100	2560
D	64	CBR	100	12.5	2560

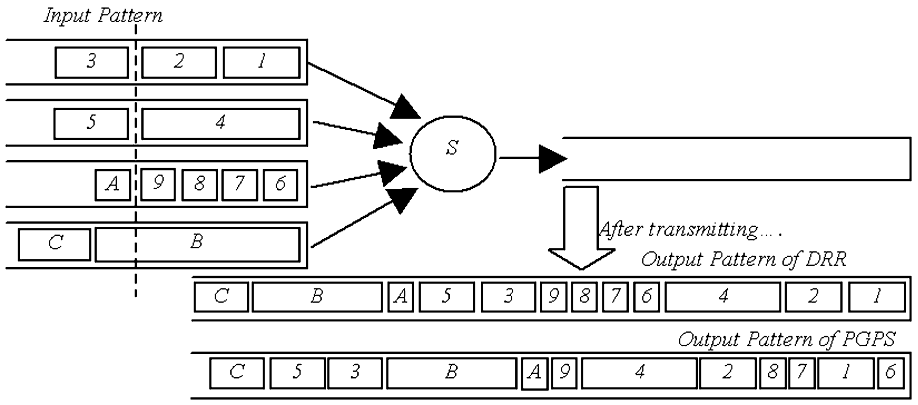


Fig. 1. Input and output patterns of packets in DRR and WFQ. Packets on the right-hand side of the broken line are allowed to be sent out in the first round. Packet *B* is excluded in the first round.

2.3. Problems and causality of DRR

Fig. 1 depicts the problems as the *Quantum* size reaches many times the size of the maximum packet. Suppose that there are four flows requesting the same amount of bandwidth and have fixed, but heterogeneous packet sizes. The same *Quantum* is assigned to all of them and, according to the rules in Section 2.2, the *Quantum* should be equal to the *largest maximum* packet size among all flows. Assume that packets 1, 4, 6, and *B* arrive at the same time and all have greedy flow sources. Namely, all flows are heavily backlogged. By comparing the output pattern in DRR with that in WFQ, three problems are observed. First, packets 1, 4 and 6 should be transmitted according to the sequence of service completions in the fluid server such as 6, 1 and 4. However, DRR only considers whether a packet could be sent out in a round and does not care for their eligible transmission sequence. Second, packets 6, 7, 8 and 9 are sent out in a batch, which in terms of latency and fairness is not a well behavior. Third, the transmission time of packet *B* with size slightly greater than the residual credits of this round is delayed until the next turn of this flow, after all other flows finish their transmissions in the second round. In fact, in contrast with the result in WFQ, it may wait too long. The delay increases with the frame size and a larger *Quantum* size produces larger frame size.

Thus, all three above mentioned problems are due to the large *Quantum* size, as compared with the maximum packet size. Reducing the *Quantum* size is an intuitive solution. However, it is ineffective as it would often cause the server to discover nothing to send out after querying all flows. Thus, the *Quantum* size is not reduced. Instead, a *Pre-order Queueing* module is appended to the original DRR scheduler to divide *Quantum* into several parts, which makes the flow to spend its *Quantum* in pieces within a round. Furthermore, for the packet described in the third problem, its transmission sequence is decided only according to the size of its portion over the residual credit of the previous round, which enables the server to treat it as fair as those packets of smaller size than *Quantum*.

3. Pre-order Deficit Round Robin

3.1. Architecture

A new algorithm, *Pre-order Deficit Round Robin* (PDRR), is presented herein. Fig. 2 illustrates an overview of the PDRR architecture. An additional module, *Pre-order Queueing*, which consists of a

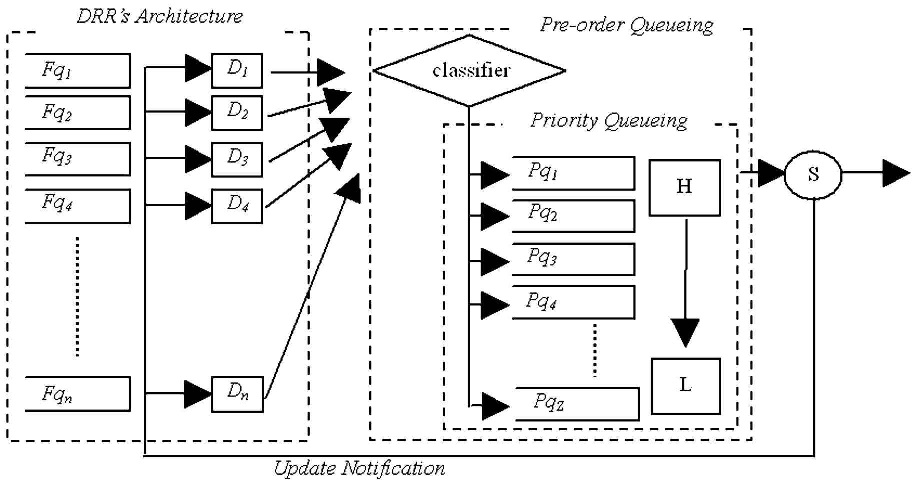


Fig. 2. The architecture of PDRR. Fq_i denotes the queue of flow *i*. Pq_j denotes the priority queue *j* in *Pre-order Queueing*. D_i denotes the $DeficitCounter_i$. *S* is the server.

classifier sub-module and a priority queueing sub-module, is appended to the original architecture of DRR so as to pre-order the service order of DRR and overcome the problems described in Section 2.3. Fig. 2 shows that the priority queueing sub-module has a limited number of priority queues, Z. Let Pq_1, $Pq_2, \ldots, Pq_Z$ denote the priority queues. As the server is ready to transmit, it picks up one packet from Pq_j with the smallest j among all nonempty priority queues. In one round, even when the server has sent out all packets in Pq_{j+1}, if one packet arrives at Pq_j, this packet is sent out before any other packets in Pq_l where l is larger than j.

The classifier sub-module decides which priority queue a packet should enter. Since the server picks packets from Pq_j with the smallest j among all nonempty ones, *Pre-order Queueing* in effect determines the order of packets in a round that appear in the output link. Prior to describing our classifier, we show why PDRR can approximate PGPS. Assume that all flows begin heavily backlogged at time t and review the output pattern of WFQ (Fig. 1). The server in WFQ selects packets to send out according to the sequence of service completions within the *fluid* server. That is, the packet with the smallest virtual finishing timestamp among the packets at the heads of all flow queues can be sent out first. Under the above heavy backlog assumption, the virtual finishing timestamp of a packet is computed as

$$TS_i^m = TS_i^{m-1} + \frac{L_i^m}{r_i}, \tag{3}$$

where TS_i^m denotes the timestamp of the mth packet of flow i after time t and, for all i, TS_i^0 is set to zero at time t, r_i denotes the allocated rate of flow i, and L_i^m denotes the size of the mth packet of flow i after time t. Eq. (3), by substituting Acc_i^m for $TS_i^m \times r_i$, is equivalent to

$$\frac{Acc_i^m}{Quantum_i} = \frac{Acc_i^{m-1} + L_i^m}{Quantum_i}, \tag{4}$$

where Acc_i^m denotes the accumulated amount of data with in a byte that flow i has sent out after transmitting the mth packet after time t. Assume that all m packets could be transmitted in the kth round. Eq. (4), by replacing Acc_i^m with $DeficitCounter_i^0 - DeficitCounter_i^m$, is equivalent to

$$\frac{DeficitCounter_i^m}{Quantum_i} = \frac{DeficitCounter_i^{m-1} - L_i^m}{Quantum_i}, \tag{5}$$

where $DeficitCounter_i^m$ denotes the residual credits of flow i in this round *after* it puts the mth packet into the *Pre-order Queueing*. To further illustrate the equivalence, the following definition is required:

Definition 1. The *Quantum Availability*, QA_i^m, of the packet P_i^m is the ratio of its $DeficitCounter_i^m$ to $Quantum_i$, i.e.,

$$QA_i^m = \frac{DeficitCounter_i^m}{Quantum_i}. \tag{6}$$

Lemma 1. *For any packet P_i^m, its Quantum Availability, QA_i^m, satisfies*

$$0 \leqslant QA_i^m < 1.$$

Proof. Readers are referred to Appendix A. □

Lemma 2. *For the packet with the smallest timestamp in one round, its QA is the largest.*

Proof. Readers are referred to Appendix A. □

According to the Smallest virtual Finishing time First policy [12], to approximate the sequence of service completions in GPS, the server should select the packet with the smallest timestamp to send out. From the Lemma 2, our architecture selects the packet with the largest QA within one round to send out. However, it costs too much time to find the packet with the largest QA among all packets that could be sent out in this round. Thus, our classifier only classifies packets to several classes according to their QA and places them into the corresponding priority queues. Assume that there are Z priority queues and, hence, Z classes. For the mth packet of flow i that can be sent out in this round, its class n_i^m can be derived as

$$n_i^m = Z - \lfloor QA_i^m \times Z \rfloor = Z - \left\lfloor \frac{DeficitCounter_i^m}{Pqg_i} \right\rfloor, \tag{7}$$

where $DeficitCounter_i^m$ denotes the residual credits in byte for flow i in the kth round *after* the mth packet is placed into a priority queue, and Pqg_i denotes the granularity of priority queue for flow i derived as

$$Pqg_i = \frac{Quantum_i}{Z}. \tag{8}$$

3.2. Algorithm

Our scheduling algorithm is described formally herein. That is, it repeatedly performs the following three modules: *PKT_Pass*, *PKT_Arrival* and *PKT_Departure*. *PKT_Pass* manages the update of the *DeficitCounter* for each flow and classifies eligible packets into the priority queueing sub-module from their *Fq*. *PKT_Arrival* simply places each arrival packet into its corresponding *Fq*. *PKT_Departure* declares the beginning of a new round and notifies *PKT_Pass* to process non-empty *Fq*'s. Once there are packets in the priority queueing sub-module, it repeatedly picks the packet from Pq_j with the smallest j, among non-empty priority queues, to serve. A *min heap*, a complete binary tree in which the key value in each node is no larger than the key values in its children (if any), is used to enable *PKT_Departure* to efficiently obtain the information about the smallest *j*.

Prior to examining these pseudo codes of the three operations, Tables 2 and 3 contain some necessary definitions.

Below is our *PKT_Pass* and *PKT_Arrival* pseudo code:

```
PKT_Arrival module
$i = ExtractFlow(p)$                  // Get the flow # of packet $p$
$Enqueue(p, (Fq_i)$
If $NumItem (Fq_i) = 1$ Then          // The equal implies that $Fq_i$.is empty before $p$ was
  $SendMsg$(PKT_Pass, $i$)            // placed into it, and we need to notify PKT_Pass
                                      // to handle the flow $i$.
PKT_Pass module
While(TRUE)
{$i = WaitMsg()$                      // Wait until a pkt is placed to the empty $Fq_i$.
If $Round_i \neq Round_{sys}$         // Non-equivalent implies a new round is arrival.
{  $Round_i = Round_{sys}$
  $DC_i = \mathrm{Max}(DC_i, Quantum_i)$
}
While $DC_i > 0$ and $NonEmpty(Fq_i)$  // Classify eligible packets into PQ.
{   $PktSize = Size(Head(Fq_i))$      // Get the size of the pkt at head of $Fq_i$.
    If ($PktSize \leqslant DC_i$) Then  // if flow $i$ credits are enough to send out pkts.
   {   $DC_i = DC_i - PktSize$        // Take out the used credits
      $j = Z - (DC_i/Pqg_i)$          // Compute the $j$
      $Enqueue(Dequeue(Fq_i), Pq_j)$  // Move the head packet of $Fq_i$ to eligible $Pq_j$.
       If $NumItem(Pq_j) = 1$ Then    // It implies that $Pq_j$ is empty and $j$ is non-existed in
                                      // the min heap before the last packet was placed into it.
          $MH\_Insert(j)$             // Insert $j$ to min heap
   }
}
If $NonEmpty(Fq_i)$ Then              // Imply the residual credits are not enough.
 {  $Enqueue(i, AckList)$
    If $NumItem(AckList)$=1 Then $SetEvent(EV_{actlist})$
 }
}
```

On arrival of packet *p*, *PKT_Arrival* obtains its flow identification *i* and places it into Fq_i. It then notifies *PKT_Pass* if there are no other packets in Fq_i. For each packet that could be sent out in this round, *PKT_Pass* decides to which class *j* it belongs and takes it into the corresponding priority queue Pq_j from its

Table 2
The operations used in *PKT_Arrival* and *PKT_Departure* and *PKT_Pass* procedures

Operation	Description
Enqueue, *Dequeue*, *NonEmpty*, *Empty*, *Head*	The standard *Queue* operations
NumItem(*A*)	Return the number of entries in the *A*
MH_Insert(*x*), *MH_Delete*(), *MH_Empty*()	The standard operations of the min heap
MinHeapRoot	The minimum value among nodes of the min heap
MH_Lock, *MH_Unlock*	After locking, only one module can access the min heap
SetEvent(*ev*)	Signal the event *ev*. The event will remain until signaled someone releases it
EventCase = *WaitEvents*($ev1, ev2, \ldots$)	Once any event is in the signaled state, return it to *EventCase* and release it. Note *ev*1 is prior to *ev*2
SendMsg(x, y)	Send message along with value y to x
y = *WaitMsg*()	Wait message and store the received value in y

Table 3
The variables used in *PKT_Arrival* and *PKT_Departure* and *PKT_Pass* procedures

Variable	Description
Fq_i	The queue of flow $i, i = 1, \ldots, N$
Pq_j	Priority queue $j, j = 1, \ldots, Z$
$Quantum_i$	The created allocated to flow i in one round
DC_i	The *DeficitCounter* of that flow i
Pqg_i	The granularity of priority queue for flow i
Z	The number of priority queues
$Round_{sys}$	The identification of the system current round
$Round_i$	The identification of the round of flow i
$EV_{minheap}$	A event signaled as one key was placed into the empty min heap
$EV_{actlist}$	A event signaled as one item was placed into the empty ActList

Fq. If it is the first packet of Pq_j, the function *MH_Insert* along with parameter j is called to indicate that Pq_j is not empty. Alternately, if there are packets which cannot be sent out in this round, due to the insufficient value of *DeficitCounter*, a flow identifier is inserted into *AckList*. *PKT_Departure* knows, from *ActList*, which flow queues have remaining packets in this round and accumulate their residual credits for the next round.

We now describe our *PKT_Departure* as the following pseudo codes

```
**PKT_Departure module**
While(TRUE)
{ If $MH\_Empty()$ Then              // Imply no packets can be placed into PQ
 {  $Round_{sys} = Round_{sys} + 1$        // Announce arrival of the new round
    $EC = WaitEvents(EV_{minheap}, EV_{actlist})$          // wait until a pkt was placed in PQ or any Fq's
   If $(EC = EV_{actlist})$ Then       // Imply there are pkts without sending out at last round.
  {  $NumAckList = NumItem(AckList)$      // There are NumAckList non-empty Fq's at last round.
     While$(NumAckList > 0)$              // For non-empty Fq's at last round, accumulate
    {  $i = Dequeue(AckList)$             //  their residual credits for using at this round.
       $DC_i = DC_i + Quantum_i$
       $SendMsg(\textbf{PKT\_Pass}, i)$            //Ack PKT_Pass to pass pkts of $Fq_i$ into PQ.
      $NumAckList = NumAckList - 1$
    }
    $WaitEvent(EV_{minheap})$                // Pause if no packets in PQ.
  }  // IF $EC = EV_{actlist}$
 } // IF MH_EMPTY
 $WaitEvent(ServerIdle)$                // Pause if the server is sending out a packet.
 $MH\_Lock()$    //To avoid the MinHeapRoot being modified as MH_Delete() is involved.
 If $Empty(Pq_{MinHeapRoot})$ Then $MH\_Delete()$
 $MH\_Unlock()$
 $Send(Dequeue(Pq_{MinHeapRoot}))$          // Send out the pkt in the $Pq_j$ with the smallest j
}                                       // among non-empty $Pq_j$.
```

Once packets are pushed into the priority queueing sub-module by *PKT_Pass*, the status of the event $EV_{minheap}$ is also be signaled to notifiy *PKT_Departure* to send out a packet drawn from $Pq_{MinHeapRoot}$. After the transmission is complete, ServerIdle is signaled and *PKT_Departure* repeats the last action until the $Pq_{MinHeapRoot}$ is empty. At this point, the function *MH_Delete* deletes the root node of the min heap and sets *MinHeapRoot* to the smallest j among residual nodes. When the min heap is empty, i.e., all Pq_j's are empty, *PKT_Departure* declares the arrival of a new round by adding 1 to $Round_{sys}$. For all non-empty *Fq*'s, i.e.,

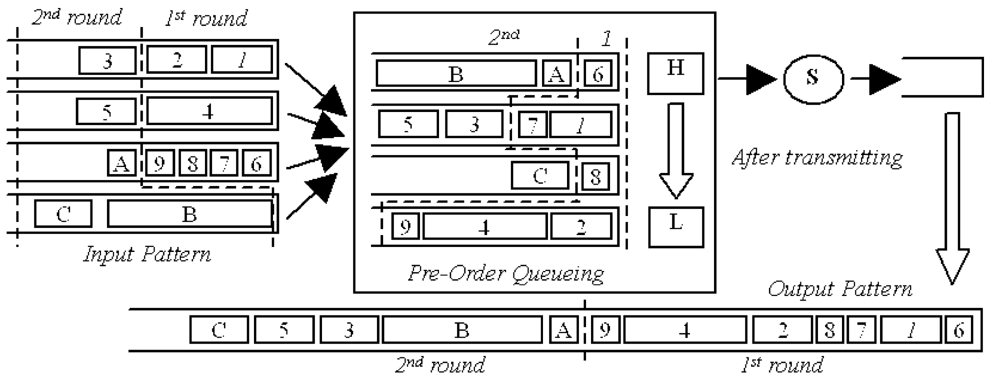


Fig. 3. The transmission state of packets in PDRR.

flows with packets remaining in the last round, it updates their *DeficitCounter*'s according to their sequence in the *AckList* and requests that *PKT_Pass* classify the eligible packets into the priority queueing sub-module. Then, it waits again until new packets are inserted to the priorityqueueing sub-module.

3.3. Example

An example that uses the same input pattern and assumption as in Fig. 1 is given to show that our algorithm can overcome the problems of DRR in Section 2.3. In this example, packets are classified into 4 classes, $Z = 4$. Suppose that for all flows their *Quantum*'s are equal to 400 and the size of packet B is 500. Clearly, packet B could not be sent out in the first round. However, in the next round, $DeficitCounter^B$ would be equal to 300, i.e., $400 + 400 - 500$. According to (7), the packet would be classified into the first class, i.e., $4 - \lfloor 300/(400/4) \rfloor$ and could be sent out at the beginning of the next round. Other packets are put into the priority queueing sub-module according to the same rule. Fig. 3 presents the result, which is the same as that in WFQ (Fig. 1) and is unlike that in DRR. Under the assumption that all flows are heavily backlogged and there are enough priority queues, our algorithm is expected to have behavior similar to that of WFQ.

4. Analytical results

In this section, the performance of PDRR is analyzed in terms of delay bound and throughput fairness. Finally, PDRR is revealed to have an O(1) per-packet time complexity in most cases. In high-speed networks, low per-packet time complexity is an important criteria for routers and switches.

4.1. Delay bound

Consider a queueing system with a single server of rate C.

Definition 2. *A backlogged period for flow i* is any period of time during which flow i is continuously backlogged in the system.

Let t_0 be the beginning of a backlogged period of flow i and t_k indicate the time that the kth round in PDRR is completed. $W_i(\tau, t_k)$ denotes the service offered to flow i in the interval $(\tau, t_k]$ by the server and L_i is the maximum packet size of flow i.

Lemma 3. *Under PDRR, if flow i is continuously backlogged in the interval $(t_0, t_k]$, then at the end of the kth round,*

$$W_i(t_0, t_k) \geqslant k\phi_i - D_i^k, \tag{9}$$

where D_i^k is the value of the DeficitCounter$_i$ at the end of the kth round and ϕ_i is Quantum$_i$.

Proof. Readers are referred to Appendix A. □

Lemma 4. *Let t_0 be the beginning of a backlogged period of flow i in PDRR. At any time t during the backlogged period,*

$$W_i(t_0, t) \geqslant \max\left(0, r_i\left(t - t_0 - \frac{(2 + (1/Z))F - \phi_i}{C}\right)\right), \tag{10}$$

where F is equal to $\sum_1^{\#flow} \phi_i$, Z is the number of priority queues, r_i is the rate allocated to the flow i.

Proof. Readers are referred to Appendix A. □

Theorem 1 (PDRR belongs to *LR*). *The PDRR server belongs to LR with latency θ^{PDRR} less than or equal to*

$$\frac{(2 + (1/Z))F - \phi_i}{C}. \tag{11}$$

Proof. The proof is straightforward from Lemma 4 and the definition of *LR* server [8]. □

According to (1), replacing F with $\phi_i C/r_i$ in (11), we also have

$$\theta^{\mathrm{PDRR}} \leqslant \left(2 + \frac{1}{Z}\right)\frac{\phi_i}{r_i} - \frac{2\phi_i}{C}. \tag{12}$$

Eq. (11) shows that PDRR evidentially improves latency, as opposed to the DRR whose latency is $(3F - 2\phi_i)/C$. Furthermore, in the worst case, if the form of θ^{SCFQ} [8] is translated to the form of θ^{PDRR}, the latency of PDRR is revealed as similar to that of SCFQ, $(2F - \phi_i)/C$. From (12), it reveals the latency of PDRR is inversely dependent with the allocated bandwidth, and independent of the number of actived flows The major reason for DRR to have worse latency is that it only considers whether a packet could be sent out in a round and ignores the information provided in the quantum consumed by a packet. The latter information can be used to *re-order* the packets sent out in a round.

Theorem 2 (Delay bound). *Suppose that the scheduling algorithm at the server is PDRR and the traffic of flow i conforms to a leaky bucket with parameters (σ_i, ρ_i), where σ_i and ρ_i denote the burstiness and average rate of the flow i. Assume the rate allocated to the flow i is equal to ρ_i. If Delay$_i$ is the delay of any packet of flow i, then*

$$Delay_i \leqslant \frac{\sigma_i}{\rho_i} + \frac{(2 + (1/Z))F - \phi_i}{C}. \tag{13}$$

Proof. The proof is straightforward from Theorem 1 and the analysis results of *LR* server [8]. □

4.2. Throughput fairness

The *fairness* parameter herein is based on the definition presented by Golestani [3]. However, derivation skills in [8] were followed to obtain ours.

Theorem 3. *For a PDRR scheduler,*

$$Fairness^{\mathrm{PDRR}} = \frac{(2+(1/Z))F}{C}, \tag{14}$$

where $Fairness^{\mathrm{PDRR}}$ *is the fairness of the server PDRR.*

Proof. Readers are referred to Appendix A. □

Thus, $Fairness^{\mathrm{PDRR}}$ is smaller than $Fairness^{\mathrm{DRR}}$, which is $3F/C$ [8]. The fairness and latency of other scheduling algorithms have been analyzed in [8].

4.3. Per-packet time complexity

Table 4 summarizes the complexity of *PKT_Arrival*, *PKT_Pass*, *PKT_Departure* when a non-empty or empty priority queue is accessed. For each packet, *PKT_Arrival* inserts it into its corresponding *Fq* and *PKT_Pass* takes it from its *Fq* to the Pq_j, where j is found in a constant number of operations. *PKT_Departure* simply repeats to pick a packet from the $Pq_{MinHeapRoot}$ whose *MinHeapRoot* always presents the smallest j among non-empty Pq_j's. As the min heap operations are not invoked under the assumption that each accessed *Pq* is non-empty, all the complexities of the above operations are O(1). When the $Pq_{MinHeapRoot}$ is empty, a delete operation of the min heap is invoked by *PKT_Departure* to get the new *MinHeapRoot*. Basically, we cannot continue to send out packets until the end of reheapification loop whose time complexity is $\mathrm{O}(\log Z)$, where Z is the maximum number of keys present in the min heap, i.e., the number of non-empty *Pq*'s at that moment. A similar situation also occurs as *PKT_Pass* must insert a new j into the min heap.

However, there are sufficient reasons to say that packets can be sent out individually and have low delay possibility by operations of the min heap. First, the min heap operation in our algorithm is involved only when the *Pq*, which it tries to access is empty, unlike the sorted-priority algorithms, e.g., SCFQ and WFQ, where the two operations, insert and delete, are repeatedly invoked once the server tries to send out a packet. Secondly, the scalar of the min heap is small whose maximum number of keys are the number of *Pq*'s instead of the number of flows. Moreover, the approach, presented in [13], customarily allows concurrent insertions and deletions on the heap. According to the concept, *PKT_Departure*, after getting the next smallest value j immediately via one comparison between the two leaf keys of the root, can start to send out packets in the Pq_j concurrently during the period of the reheapification loop. In brief, the time complexity of PDRR is O(1) in most cases and $\mathrm{O}(\log Z)$ in some special cases. That is obviously lower than the complexity of SCFQ, $\mathrm{O}(\log N)$ where N is the number of flows.

Table 4
The complexity of PDRR

Operation case	*PKT_Arrival*	*PKT_Pass*	*PKT_Departure*
Pq non-empty	O(1)	O(1)	O(1)
Pq empty	O(1)	$\mathrm{O}(\log Z)$	$\mathrm{O}(\log Z)$

5. Simulation results

Simulation results are shown in this section to compare the delay performance of PDRR with those of DRR and SCFQ. The link bandwidth, i.e., server capacity, is assumed to be 80 Mbps, shared by 20 flows. These 20 flows are divided in two groups, GA and GB. As the goal of this simulation is not to compare the sorted-priority scheduling algorithms, which are in a different class from ours, only SCFQ was choosen, which is the easiest to implement among all sorted-priority algorithms, in addition to DRR, to compare with PDRR.

5.1. Bursty transmission

The following experiment was performed to show the problem of bursty transmission by DRR and the improvement of PDRR. Assume that all traffic sources are at a constant bit rate (CBR) with fixed packet size. Table 5 indicates the traffic parameters of the two groups, GA and GB. As the bit rates of GA and GB are equal and the maximum packet size among both groups is 500 bytes, the same *Quantum* size of 500 bytes was allocated them. Notably, the packet arrival rate of GA flows is 10 times the size that GB's is. In this experiment, 10 priority queues were placed in the *Pre-order Queueing* of PDRR.

The delay times of a particular GA flow under DRR, PDRR, and SCFQ, respectively were measured. In Fig. 4, it is observed that packets of flow within DRR are sent out in a batch once the flow is served. The packets that just missed the transmission chance, receive the largest delay. However, in PDRR the flow is able to spend its *Quantum* in several pieces, so that packets could be sent out uniformly. Another observation of this special case is that packets in SCFQ suffer high delay jitter. Due to GA flows having a 10 times packet arrival rate than GBs do. For GA packets that arrive while the server is serving a large packet of GBs, their virtual arrival times would be equal, which causes the server not to send them out in their actual arrival sequence.

5.2. Packet size

To further discuss the average behavior of PDRR, the traffic source is assumed to be MMPP shaped by a leaky bucket whose on/off rates are both 1000 1/μs and bit rate is 4 Mbps. In this experiment, GA flows are

Table 5
The traffic parameters and *quantum* size of two groups

Group	Traffic type	Bit rate (Mbps)	Packet size (byte)	Quantum size (byte)
GA	CBR	4	50	500
GB	CBR	4	500	500

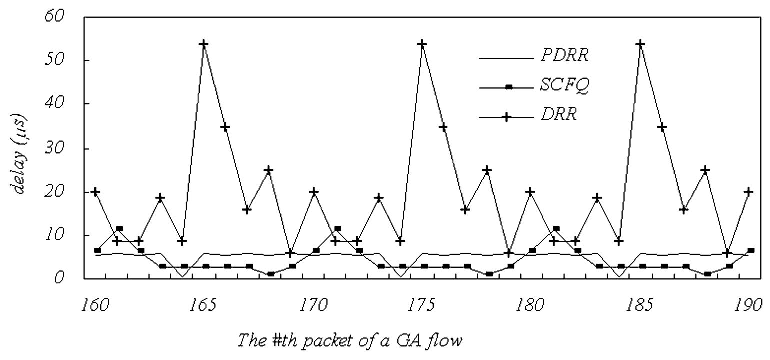


Fig. 4. Packet delay of a constant bit rate source.

assigned a larger packet size than GB flows, however, all request the same bandwidth. In Fig. 5, a GA flow in DRR has a larger average delay then that in PDRR and SCFQ is observed. The result for GB flow is similar. Additionally, with the increase of GA's packet size to that of GB's, the curve of PDRR in Fig. 5 becomes closer to that of SCFQ, which means that PDRR works like SCFQ, especially with heterogeneous traffic sources. This is due to the more heterogeneous the traffic sources are, the more bursty the transmission behavior of GA flows are. Fig. 6 shows that as the ratio of GA's packet size to GB's is increased, small packets in PDRR and SCFQ perform better than large packets. However, this is not so evident for the case of DRR. That is the server in PDRR considers the information provided in the quantum consumed by a packet and reorders the transmission sequence of packets in one round. This is similar to a server in SCFQ considering the timestamp of a packet and then sending out the packet with the smallest timestamp. However, in DRR, the server only considers whether a packet could be sent out and ignores the transmission order of packets in one round.

5.3. Bit rate

Under the environment of flows with heterogeneous bandwidth requirements, the flow with high bit rate in DRR still has the bursty transmission behavior. Thus, our algorithm also could perform better than DRR because the *Pre-order Queueing* module can reduce the forenamed problem by transmitting packets

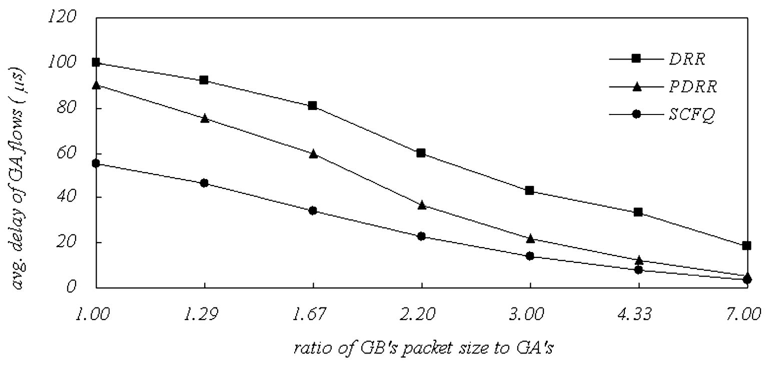


Fig. 5. The average delay of GA flows in DRR, PDRR, and SCFQ.

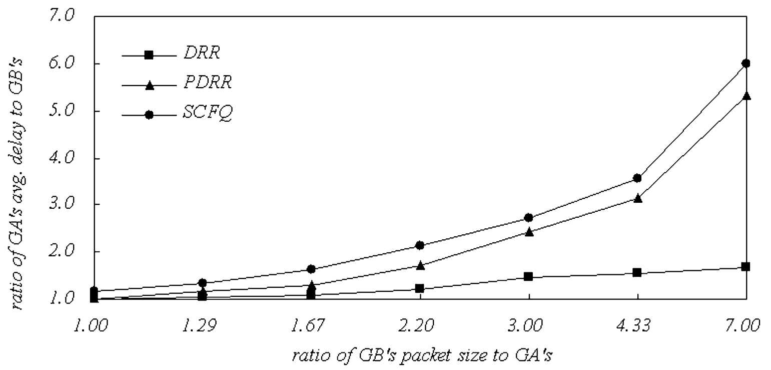


Fig. 6. The performance ratio of GA to GB flows in DRR, PDRR, and SCFQ.

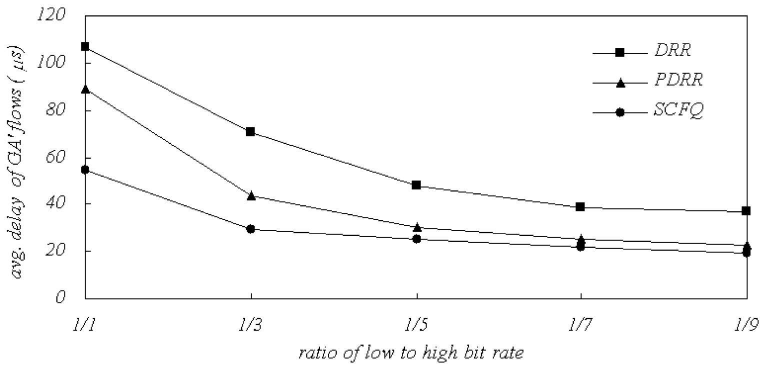


Fig. 7. Average delay of GA flows under heterogeneous bandwidth requirements in DRR, PDRR, and SCFQ.

more uniformly within a round. GA flows are assumed to have a higher bit rate than that of GB flows. Fig. 7 shows that the average delay of GA flows in PDRR is lower than that in DRR. Furthermore, the result for GB flows is similar to that in Fig. 7.

5.4. Number of priority queues

The ideal number priority queues for a specific traffic environment is discussed herein. As described in Section 3.1, the *Pre-order Queueing* enables a flow to use its *Quantum* uniformly within a round, especially when its *Quantum* is several times its maximum packet size. For flow i, the *Pre-order Queueing* with $(Quantum_i/L_i)$ priority queues can reach the above goal. Thus, for a specific traffic environment, Z priority queues are enough, where Z is equal to the maximum value of $(Quantum_i/L_i)$ among all flows, i.e., $\max_i(Quantum_i/L_i)$. Fig. 8 shows, through an experiment, the relationship between Z and $\max_i(Quantum_i/L_i)$. For easier observation, all traffic sources are assumed as CBR with fixed packet size. Each curve in Fig. 8 shows the average delay of the flow whose $(Quantum_i/L_i)$ equals to $\max_i(Quantum_i/L_i)$. For each line, this flow receives the smallest average delay when $Z = \max_i(Quantum_i/L_i)$, which confirms our hypothesis.

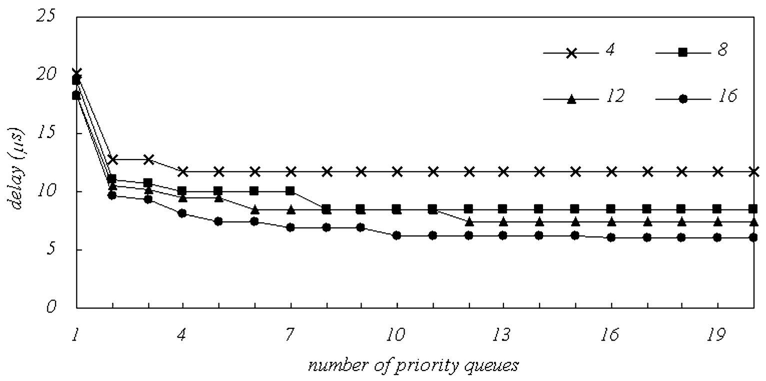


Fig. 8. The optimal number of pre-order queues under different $\max_i(Quantum_i/L_i)$.

6. Related works

Uniform Round Robin (URR) [9] is a scheduling algorithm for ATM networks, improved from Weighted Round Robin (WRR) [10]. Some of its goals are the same as our algorithm. However, its frame and packet sizes are *fixed*. That is, within a frame there is a fixed number of slots. For a slot allocated to one flow, if the flow has no data waiting to be sent out, the slot will be wasted. Namely, it is not a work-conserving scheduling algorithm and the residual bandwidth cannot be shared by those active flows. Several work-conserving versions of WRR are designed in [11], however, their currenct schemes and analytical results are only applicable to fixed-length packets. In fact, latency and fairness, the two properities of scheduler, are affected seriously as the packet-length is variable. That is what the PDRR has attempted to overcome.

Calendar queue [14] is a technique for reducing the implementation complexity of GPS-like schedulers. Its concept, similar to ours, classifies packets. However, it does so according to their timestamps. In fact, for the packets currently in the system, the maximum interval between any two timestamps could be very large, which depends upon the fairness behavior of the underlying scheduler. To accommodate the large time interval, *calendar queue* must maintain a large number of queues to reduce the granularity of a class. In our PDRR, however, the range of the classified value is *fixed* and *small*. Thus, less memory space is needed to obtain the same classification granularity as that in the *calendar queue*. Concerning any GPS-like schemes combined with calendar queue, the complexity is reduced to O(1). However their latency and fairness are affected. The comparison and discussion are however, beyond the scope of this work.

7. Conclusion and future work

In this work, a rule for deciding the *Quantum* size of a flow was offered and several problems in DRR such as bursty transmission and inappropriate transmission sequence that causes a large delay bound and serious unfair behavior, were presented. A new algorithm, PDRR, was proposed, which adds a new structure, *Pre-order Queueing*, into the DRR scheduler. It reorders the transmission sequence of the packets that could be sent out in one round, according to the quantum consumption status of its flow within this round. This thereby enables PDRR to approximate PGPS and overcome said DDR problems.

From the analysis of PDRR on delay and fairness and simulation results, our algorithm is able to avoid bursty transmissions and overcome the serious problem of DRR. Namely, packets that cannot be sent out within the current round may get too high a penalty. Meanwhile, PDRR still maintains the advantages of DRR, i.e., per-packet time complexity independent from an increase in flow number and ability to handle variable-length packets. Furthermore, we have also shown that the treatment of PDRR to packets with different sizes is similar to that of the sorted-priority scheduling, such as SCFQ.

There remains work to be done in the future. First, more simulation cases, such as flows with different allocated bandwidth but having the same distribution of packet sizes, are being investigated. Secondly, traffic types, other than CBR and MMPP, could be applied to PDRR to observe its general behavior. Finally, we plan to implement our scheduling algorithm on a Linux-based router to further evaluate its performance within a real environment.

Appendix A. Proofs of primary results

Proof of Lemma 1. This is proved by showing that for any packet P_i^m, its $DeficitCounter_i^m$ must be positive and smaller than $Quantum_i$. From the description in Section 3.2, the value in *DeficitCounter* cannot be negative and could only increase by $Quantum_i$ in the *UpdateOneFlow* procedure. Assume that there is insufficient credit to send out the packet P_i^m, i.e., the $DeficitCounter_i^{m-1}$ is smaller than L_i^m, the size of this packet. After updating and sending out the packet P_i^m, $DeficitCounter_i^m = DeficitCounter_i^{m-1} + Quantum_i - L_i^m$. As $DeficitCounter_i^{m-1} < L_i^m$, $DeficitCounter_i^m$ must be smaller than $Quantum_i$. From the above description, the lemma is proved. □

Proof of Lemma 2. According to (6), Eq. (5) is equivalent to

$$QA_i^m = QA_i^{m-1} - \frac{L_i^m}{Quantum_i} \tag{A.1}$$

and since $L_i^m > 0, Quantum_i > 0$ and $r_i > 0$, from (3) and (A.1) for any m

$$TS_i^m > TS_i^{m-1} \quad \text{and} \quad QA_i^{m-1} > QA_i^m.$$

In the same round, for the packet P_i^m with the smallest timestamp, its m is smallest among all packets, and the QA_i^m of the packet with the smallest m. Thus, for the packet with the smallest timestamp in one round, its QA is the largest. □

Proof of Lemma 3. As our algorithm only modifies the service sequence of the packets in a DRR round, the packets in DRR that could be sent out in a round during a backlogged period, they can still be sent out in the same PDRR round. Thus, PDRR also has this property of DRR that has been shown in [5]. □

Proof of Lemma 4. For each time interval $(t_{k-1}, t_k]$, we can write

$$t_k - t_{k-1} \leqslant \frac{1}{C}\left(F + \sum_{j=1}^{N} D_j^{k-1} - \sum_{j=1}^{N} D_j^k\right). \tag{A.2}$$

By summing over $k-1$,

$$t_{k-1} - t_0 \leqslant (k-1)\frac{F}{C} + \frac{1}{C}\sum_{j=1}^{N} D_j^0 - \frac{1}{C}\sum_{j=1}^{N} D_j^{k-1}. \tag{A.3}$$

Assume there are two packets, P_i^A and P_i^B, in the Fq_i whose sizes are L_i^A and L_i^B, ($L_i^B = \phi_i$), respectively and only P_i^A can be sent out at the $(k-1)$th round. All other flows exhaust their *DeficitCounter*. Thus, $D_i^{K-1} = \phi_i - \Delta$, where $0 < \Delta \leqslant \phi_i$, $D_j^{K-1} = 0$ for $j \neq i$, and

$$t_{k-1} - t_0 \leqslant (k-1)\frac{F}{C} + \frac{F - \phi_i + \Delta}{C}. \tag{A.4}$$

Under this assumption, in the kth round P_i^B would be placed into the Pq_n where

$$\begin{aligned} n &= Z - \left\lfloor \frac{D_i^K}{Pqg_i} \right\rfloor = Z - \left\lfloor \frac{D_i^{K-1} + \phi_i - L_i^B}{\phi_i/Z} \right\rfloor = Z - \left\lfloor \frac{D_i^{K-1}}{\phi_i} Z \right\rfloor = Z - \left\lfloor \frac{\phi_i - \Delta}{\phi_i} Z \right\rfloor \\ &= Z - \left\lfloor Z - \frac{\Delta}{\phi_i} Z \right\rfloor = \left\lceil \frac{\Delta}{\phi_i} Z \right\rceil \end{aligned} \tag{A.5}$$

and the maximum amount of data that could be served before P_i^B is $((n/Z)F - \phi_i)$. Thus, for any time t from the beginning of the kth round until the P_i^B is served,

$$\begin{aligned} t - t_0 &\leqslant \frac{(n/Z)F - \phi_i}{C} + t_{k-1} - t_0 \\ &\leqslant (k-1)\frac{F}{C} + \frac{F - \phi_i + \Delta}{C} + \frac{(n/Z)F - \phi_i}{C}, \end{aligned} \tag{A.6}$$

or equivalently,

$$k-1 \geqslant \frac{(t-t_0)C}{F} + \frac{2\phi_i - \Delta}{F} - \frac{n}{Z} - 1. \tag{A.7}$$

Replacing k with $k-1$ in (9) and defining r_i , the reserved rate of flow i, as $(\phi_i C/F)$, we have

$$\begin{aligned} W_i(t_0, t_{k-1}) &\geqslant \phi_i\left(\frac{(t-t_0)C}{F} + \frac{2\phi_i - \Delta}{F} - \frac{n}{Z} - 1\right) - D_i^{k-1} \\ &= r_i\left(t - t_0 + \frac{2\phi_i - \Delta}{C} - \frac{n}{Z} \times \frac{F}{C} - \frac{F}{C} - \frac{D_i^{k-1}}{r_i}\right) \\ &= r_i\left(t - t_0 - \left(1 + \frac{n}{Z} + \frac{D_i^{k-1}}{\phi_i}\right)\frac{F}{C} + \frac{2\phi_i - \Delta}{C}\right) \\ &= r_i\left(t - t_0 - \frac{(1 + (n/Z) + (D_i^{k-1}/\phi_i))F - 2\phi_i + \Delta}{C}\right). \end{aligned} \tag{A.8}$$

Replacing D_i^{k-1} with $\phi_i - \Delta$ and since

$$\frac{n}{Z} - \frac{\Delta}{\phi_i} = \left\lceil \frac{\Delta}{\phi_i} Z \right\rceil \leqslant \frac{1}{Z} - \frac{\Delta}{\phi_i}\frac{1}{Z}$$

for any Δ,

$$\begin{aligned} W_i(t_0, t_{k-1}) &\geqslant r_i\left(t - t_0 - \frac{(2 + (n/Z) - (\Delta/\phi_i))F - 2\phi_i + \Delta}{C}\right), \\ &\geqslant r_i\left(t - t_0 - \frac{(2 + (1/Z))F - \phi_i}{C}\right). \end{aligned} \tag{A.9}$$

In the worst case, flow i was the last to be updated during the kth round and its packet L_i is inserted into the tail of Pq_n. We distinguish two cases:

Case 1. At time t *before* the time that flow i is served in the kth round, i.e.,

$$t_{k-1} < t \leqslant t_{k-1} + \frac{n}{Z}F - \phi_i,$$

we obtain

$$W_i(t_0, t) = W_i(t_0, t_{k-1}). \tag{A.10}$$

Case 2. At time t *after* the time that flow i starts being served in the kth round, i.e.,

$$t_{k-1} + \frac{n}{Z}F - \phi_i < t \leqslant t_{ki},$$

we obtain

$$W_i(t_0, t) = W_i(t_0, t_{k-1}) + W_i(t_{k-1}, t) \geqslant W_i(t_0, t_{k-1}). \tag{A.11}$$

Thus, for any time t,

$$W_i(t_0, t) \geqslant \max\left(0, r_i\left(t - t_0 - \frac{(2 + (1/Z))F - \phi_i}{C}\right)\right). \quad \square \tag{A.12}$$

Proof of Theorem 3. Assume that it is at the beginning of the kth round and there are two packets P_i^A and P_i^B, in the Fq_i whose sizes are L_i^A and ϕ_i, respectively and only P_i^A can be sent out at the kth round. The packet P_i^B's class is n, which implies it will enter the nth priority queue. In the $(k+1)$ round, all packets of another flow j whose classes are larger than n, could not be sent out before the time t when P_i^B is sent out, as the server always selects packets from the nonempty Pq_j with the smallest j. Thus, before t, for another flow j,

$$W_j(t_0, t) \leqslant k\phi_j + \frac{n}{Z}\phi_j. \tag{A.13}$$

Also as $t_k < t < t_{k+1}$, from (9) for flow i,

$$W_i(t_0, t) \geqslant (k-1)\phi_i + \Delta. \tag{A.14}$$

From (A.5), (A.13) and (A.14) we can easily conclude that

$$\begin{aligned} &\left|\frac{W_i(t_0,t)}{r_i} - \frac{W_j(t_0,t)}{r_j}\right| \\ &\quad\leqslant \left(k + \frac{n}{Z}\right)\frac{\phi_j}{r_j} - (k-1)\frac{\phi_i}{r_i} - \frac{\Delta}{r_i} \\ &\quad\leqslant \left(1 + \frac{n}{Z} - \frac{\Delta}{\phi_i}\right)\frac{F}{C} \leqslant \left(1 + \frac{1}{Z}\right)\frac{F}{C}. \end{aligned} \tag{A.15}$$

This bound applies to time intervals that began at t_0. For any arbitrary interval,

$$\left|\frac{W_i(t_1,t_2)}{r_i} - \frac{W_j(t_1,t_2)}{r_j}\right| \leqslant \left(2 + \frac{1}{Z}\right)\frac{F}{C}. \tag{A.16}$$

Thus, for any two flows i and j,

$$Fairness^{\mathrm{PDRR}} = \frac{(2 + (1/Z))F}{C}. \qquad \square \tag{A.17}$$

References

[1] A.K. Parekh, R.G. Gallager, A generalized processor sharing approach to flow control in integrated services networks: The single-node case, IEEE/ACM Trans. Networking (1993) 344–357.

[2] J.C.R. Bennett, H. Zhang, WF^{2}Q: Worst fair weighted fair queueing, in: Proceedings of the IEEE INFOCOM'96, San Francisco, CA, March 1996, pp. 120–128.

[3] S.J. Golestani, A self-clocked fair queueing scheme for broadband applications, in: Proceedings of the INFOCOM'94, April 1994, pp. 636–646.

[4] L. Zhang, Virtual Clock: A new traffic control algorithm for packet switching, ACM Trans. Comput Systems 9 (2) (1991) 101–124.

[5] M. Shreedhar, G. Varghese, Efficient fair queueing using Deficit Round Robin, IEEE/ACM Trans. Networking 4 (3) (1996) 375–385.

[6] D. Stiliadis, A. Varma, Efficient fair queueing algorithms for packet-switched networks, IEEE/ACM Trans. Networking 6 (2) (1998) 175–185.

[7] S. Suri, G. Varghese, G. Chandranmenon, Leap forward virtual clock: A new fair queuing scheme with guaranteed delays and throughput fairness, in: Proceedings of the INFOCOM'97, 1997, pp. 557–562.

[8] D. Stiliadis, A. Varma, Latency-rate servers: A general model for analysis of traffic scheduling algorithms, IEEE/ACM Trans. Networking 6 (5) (1998) 611–624.

[9] N. Matsufuru, R. Aibara, Efficient fair queueing for ATM networks using Uniform Round Robin, in: Proceedings of the INFOCOM'99, 1999, pp. 389–397.

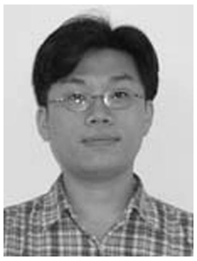
[10] M. Katevenis, S. Sidiropoulos, C. Courcoubetis, Weighted Round-Robin cell multiplexing in a general-purpose ATM switch chip, IEEE J. Selected Areas Commun. 9 (8) (1991) 1265–1279.

[11] H.M. Chaskar, U. Madhow, Fair scheduling with tunable latency: A Round Robin approach, in: IEEE Globecom'99, 1999, pp. 1328–1333.

[12] J.C.R. Bennett, D.C. Stephens, H. Zhang, High speed, scalable, and accurate implementation of fair queueing algorithms in ATM networks, in: Proceedings of the ICNP'97, 1997, pp. 7–14.

[13] V. Nageshwara Rao, V. Kumar, Concurrent access of priority queues, IEEE Trans. Comput. 37 (12) (1988) 1657–1665.

[14] J.L. Rexford, A.G. Greenberg, F.G. Bonomi, Hardware-efficient fair queueing architectures for high-speed networks, in: Proceedings of the INFOCOM '96, 1996, pp. 638–646.



Shih-Chiang Tsao was born in Hsinchu, Taiwan, in 1975. He received the B.S. and M.S. degrees in Computer and Information Science from National Chiao Tung University in 1997 and 1999, respectively. Since 1999, he worked in the Telecommunication Laboratories at Chunghwa Telecom Co., Ltd, mainly to capture and analyze switch performance. His research interests include scheduling algorithms, quality of services, and TCP bandwidth control. He has implemented QoS scheduling mechanisms in a remote access server, and is currently involved with a project on enforcing bandwidth policies.

Ying-Dar Lin was born in Hsi-Lo, South Taiwan, in 1965. He received the Bachelor's degree in Computer Science and Information Engineering from National Taiwan University in 1988, and M.S. and Ph.D. degrees in Computer Science from the University of California, Los Angeles in 1990 and 1993, respectively. At UCLA Computer Science Department, he worked as a Research Assistant from 1989 to 1993 and worked as a Teaching Assistant from 1991 to 1992. In the summers of 1987 and 1991, he was a technical staff member in IBM Taiwan and Bell Communications Research, respectively. He joined the faculty of the Department of Computer and Information Science at National Chiao Tung University in August 1993 and is a Professor since 1999. His research interests include design, analysis, and implementation of network protocols and algorithms, wire-speed switching and routing, quality of services, and intranet services. He has authored two books. He is a member of ACM and IEEE. He can be reached at http://www.cis.nctu.edu.tw/~ydlin.