Improved Smoothed Round Robin Schedulers for High-Speed Packet Networks

Chuanxiong Guo, chguo@microsoft.com

***Abstract*— The Smoothed Round Robin (SRR) [7] packet scheduler is attractive for use in high-speed networks due to its very low time complexity, but it is not suitable for real-time applications since it cannot provide tight delay bound. In this paper, we present two improved algorithms based on SRR, namely SRR$^+$ and SRR$^\#$, which are based on novel matrix transform techniques. By transforming the irregular Weight Matrix of SRR into triangular and diagonal ones, SRR$^+$ and SRR$^\#$ are able to evenly interleave flows based on their reserved rates even in skewed weight distributions. SRR$^+$ and SRR$^\#$ provide bounded delay, whereas are still of low space and time complexities and are simple to implement in high-speed networks. The properties of SRR$^+$ and SRR$^\#$ are addressed in detail through analysis and simulations. SRR$^+$ and SRR$^\#$, together with SRR and the recently developed G-3 scheduler [9] form a full spectrum of schedulers that provide tradeoffs among delay, time complexity, and space complexity.**

I. Introduction

The Internet has become to be a global information infrastructure that must support different applications such as voice-over-IP, video streaming, video conferencing, etc. The traditional Internet, however, only provides a "best-effort" service, and does not provide guarantees on whether a packet will be delivered, when a packet will arrive at its destination, and how much bandwidth an application can get.

To extend the "best-effort" Internet to support different types of applications (e.g., real-time applications), many service models have been proposed (e.g., [2], [3]). One of the key technologies in routers to differentiate applications according to their requirements is a packet scheduler. A packet scheduler decides which packet to serve when the output link has finished transmitting the previous packet. Due to their ability to provide fair bandwidth sharing and end-to-end delay bound, Fair Queueing (FQ) packet schedulers have been studied extensively in the last decade [1], [5], [12], [14], [15], [17].

The bandwidth of a single network interface now reaches up to 100 Gb/s, which means a packet scheduler needs to handle hundreds of millions of packets per-second. Therefore, packet schedulers are required to be of extreme low time complexity ($O(1)$ time complexity is preferred). Round-robin based schedulers visit and serve flows in a network interface circularly and repeatedly, and generally have very low time complexity due to their "round-robin" nature. Hence, round-robin based schedulers are suitable for implementation in high-speed routers.

This work was performed before the author joined MSRA and was supported in part by the NSFC of China under contract No. 60503049.

Round-robin schedulers, however, are well known for their scheduling burstiness. For example, for three flows f_1-f_3 with weights 4, 2, 1, the service sequence of Weighted Round Robin (WRR) in one round is $f_1f_1f_1f_1f_2f_2f_3$, whereas a much smoother service sequence is $f_1f_2f_1f_3f_1f_2f_1$. Burstiness can result in large delay for a flow even if it has a large weight (consider the case that when a flow with large weight arrives, this flow must wait until all the other flows have been visited).

To reduce the scheduling burstiness, we have proposed a Smoothed Round Robin (SRR) packet scheduler in our previous work [7]. SRR codes the weights of the flows into binary vectors to form a Weight Matrix (WM), and then uses a Weight Spread Sequence (WSS), which is specially designed to distribute the scheduling output more evenly, to schedule packets by scanning the Weight Matrix. SRR possesses better short-term fairness and scheduling delay properties in comparison with WRR. At the same time, SRR preserves $O(1)$ time complexity due to its specifically designed data structures. The worst-case delay of SRR, however, is in proportion to the number of active flows in the scheduler. SRR therefore is not suitable for real-time applications that need tight end-to-end delay bound.

In this paper, we present two novel improvements of SRR, namely SRR$^+$ and SRR$^\#$, by transforming the Weight Matrix of SRR into more regular ones. The transforms result in better scheduling output and smaller delay bound, with only moderate increases in time and space complexities. SRR, SRR$^+$, SRR$^\#$, and the recently developed G-3 scheduler [9] provide a full spectrum of schedulers that have different delay, time-complexity, and space-complexity properties.

The rest of the paper is organized as follows. We provide background materials and review SRR and its data structures in Section II. We then present SRR$^+$ and SRR$^\#$ and analyze their properties in Section III and IV, respectively. We use simulations to demonstrate the delay properties of SRR$^+$ and SRR$^\#$ in Section V. Related work is discussed in Section VI and the paper concludes in Section VII.

II. Packet Scheduling Background and SRR

This section is the base for the design and analysis of SRR$^+$ and SRR$^\#$. The major content of this section is reproduced from [9]. It is included here to make this paper self-contained.

A. Packet Scheduling Background

Modern high-speed routers are composed of many network interfaces that are interconnected by a switching fabric (e.g.,

a crossbar). The operation of the switching fabric is divided into small time-slots. Packets of variable sizes are divided into small fixed sizes (e.g., 64 bytes) in the input network interfaces and reassembled in the output network interfaces. For this reason, we assume that packets are of the same size L in the rest of the paper.

Each network interface uses a packet scheduling algorithm to decide which packet to serve among all the competing packets in that network interface. In an FQ packet scheduler, there are many active flows that compete for the output bandwidth. We assume there are N flows in the scheduler, named as $f_0, f_1, \cdots, f_{N-1}$. The output bandwidth of the link is C. For flow f_i, there is a reserved rate r_i. For schedulability reason, we need $\sum_{i=0}^{N-1} r_i \leq C$. A flow f_i is assigned a weight w_i which is in proportion to its rate r_i. In this paper, without losing generality, we assume $w_i = r_i$.

A packet scheduler is composed of three parts: A *flow_add* procedure to accept a new flow, a *flow_delete* to remove an old flow, and a *schedule* to decide which packet to serve once the previous packet has been transmitted. A packet scheduler needs to invoke the *schedule* procedure for each packet. The *schedule* procedure is therefore required to be of low time-complexity. A real-time flow (e.g., VoIP) on average lasts for tens of seconds and transmits at least hundreds or thousands of packets per-second. The *flow_add* and *flow_delete* procedures are therefore much less time-critical. A time-complexity of approximately $O(logN)$ should be good for *flow_add* and *flow_delete*.

In an $ideal$ scheduler, for a flow f with rate r_f, the total bytes that has been transmitted at time t is denoted as $S_f^{id}(t) = r_f(t - t_0)$, where $t \geq t_0$ and t_0 is the arrival time of flow f. It is easy to observe that, in the ideal scheduler, the ith $(i > 0)$ packet of f will be finished transmission at time $t_f^{id}(i) = t_0 + \frac{i \times L}{r_f}$, where L is the packet size. We also denote the bits transmitted of f at time t in a real packet scheduler ps as $S_f^{ps}(t)$. We call $S_f^{id}(t)$ ideal service curve of f and S_f^{ps} service curve of f under packet scheduler ps, respectively. Due to the packetized nature of a packet network, the finish time $t_f^{ps}(i)$ of the ith packet of flow f in a real packet scheduler ps may be different from $t_f^{id}(i)$ in the ideal scheduler.

Definition 1: We say that a packet scheduler ps provides bounded delay for flow f, if the finish time $t_f^{ps}(i)$ of the ith $(i > 0)$ packet satisfies

$$t_f^{ps}(i) \leq t_0 + \frac{i \times L}{r_f} + d_f^{ps} \tag{1}$$

where d_f^{ps} is defined as

$$d_f^{ps} = \max_{i>0}\{t_f^{ps}(i) - t_f^{id}(i)\} \leq const \tag{2}$$

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

Suppose packet scheduler ps provides bounded delay d_f^{ps} for flow f; d_f^{ps} is defined in Definition 1. For flow f with reserved rate r_f, its ideal serve curve and real service curve under ps has the following relationship:

$$S^{ps}(t) \geq S^{id}(t - d_f^{ps} - \frac{L}{r_f} + \frac{L}{C}) \tag{3}$$

And if $S^{ps}(t) \geq S^{id}(t - \tau)$, we have

$$d_f^{ps} \leq \tau \tag{4}$$

The correctness of Equation (4) is apparent and the correctness of Equation (3) is proved in [9].

B. Smoothed Round Robin

In order to reduce scheduling burstiness of WRR, Smoothed Round Robin (SRR) [7] introduces two data structures: Weight Spread Sequence (WSS) and Weight Matrix (WM).

In SRR, the weight of f_i is represented as $w_i = r_i = \sum_{j=0}^{k-1}\{a_{i,j}2^j\}$, where $k \leq \lfloor \log_2 C \rfloor + 1$. The binary coefficients $a_{i,j}$ of the N active flows are used to form a $(N \times k)$ Weight Matrix as below.

$$WM = \begin{bmatrix} a_{0,(k-1)} & a_{0,(k-2)} & \cdots & a_{0,0} \\ a_{1,(k-1)} & a_{1,(k-2)} & \cdots & a_{1,0} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N-1,(k-1)} & a_{N-1,(k-2)} & \cdots & a_{N-1,0} \end{bmatrix} \tag{5}$$

where $a_{i,j} \in \{0, 1\}$, $0 \leq i \leq N-1$, and $0 \leq j < k$. The columns of the Weight Matrix are numbered from left to right as col_{k-1}, col_{k-2}, $\cdots$, col_0, respectively. Note that the weight of terms in column i is 2^i. We further denote $y_j = \sum_{i=0}^{N-1} a_{i,j}$ for $0 \leq j \leq k-1$. For this WM matrix, we have

$$0 \leq y_j \leq N \tag{6}$$

and

$$\sum_{i=0}^{N-1} r_i = \sum_{i=0}^{N-1}\sum_{j=0}^{k-1}\{a_{i,j}2^j\} = \sum_{j=0}^{k-1}\{y_j 2^j\} \leq C \tag{7}$$

where C is the bandwidth of the output link.

In SRR, the Weight Matrix is then scanned by a specially designed Weight Spread Sequence (WSS). A set of WSS is defined as

$$WSS^k = \{a_i^k | 1 \leq i \leq 2^k - 1\} \tag{8}$$

where a_i^k is defined recursively as

$$a_i^k = \begin{cases} a_i^{k-1}, & 1 \leq i \leq 2^{k-1} - 1 \\ k, & i = 2^{k-1} \\ a_{i-2^{k-1}}^{k-1}, & 2^{k-1} < i \leq 2^k - 1 \end{cases} \tag{9}$$

for $k > 1$, and $\text{WSS}^1 = \{a_1^1 = 1\}$. We call k the *order* of sequence WSS^k. It is apparent that the set of elements of WSS^k is $\{1, 2, 3, \cdots, k\}$ and the size of WSS^k is $2^k - 1$. From the definition, we get $\text{WSS}^1 = \{1\}$, $\text{WSS}^2 = \{1, 2, 1\}$, $\text{WSS}^3 = \{1, 2, 1, 3, 1, 2, 1\}$, etc. Note that element i appears 2^{k-i} times in WSS^k.

For a Weight Matrix that has k columns, there is a corresponding WSS of order k. The Weight Matrix and the WSS sequence are then combined to design the SRR packet scheduler. In SRR, WSS is scanned term by term pointed by a

pointer p_w. p_w starts from 1 and is incremented by one when one term of WSS is scanned. The index of the next WSS item that pointed by p_w is calculated by $\{1+p_w \, mod \, (2^k-1)\}$. That is, when p_w reaches the last term of WSS, it starts from the first term again. When the current term of WSS pointed by p_w is element i, column col_{k-i} of the Weight Matrix is selected. For each occurrence $a_{f,k-i} \neq 0$ in this column, packet from flow corresponding to the row of $a_{f,k-i}$ is then scheduled. Due to the properties of WM and WSS, it is easy to observe that a flow with reserved rate r is visited r times in a round.

When flows arrive or leave, they need to be added or removed from the Weight Matrix. The number of columns of the Weight Matrix, k, may change accordingly. The maximum value of k is $\lfloor \log_2 C \rfloor + 1$. The dynamic change of k does not affect the time complexities of SRR [7] and the new schedulers we present in this paper. For description simplicity, we denote the number of columns of all the matrices as k.

Due to the simplicity of the scheduling procedure, SRR achieves $O(1)$ time-complexity for packet scheduling. SRR is also able to provide good short-term fairness among two competing flows [7]. SRR, however, cannot provide low delay bound. This can be illustrated by the skewed weight distribution example as follows. Suppose the output bandwidth is 2^n, and there are 2^{n-1} flows $f_1 - f_{2^{n-1}}$ with reserved rate 1 in the beginning. Then a flow f_0 with reserved rate 2^{n-1} comes when the scheduler is serving the first flow f_1. f_0 must wait for all the other 2^{n-1} flows before its first packet can be served, even its reserved rate is much higher than all the other flows.

In fact, the above observation can be quantitative analyzed. In SRR, when a non-zero term $a_{i,j}$ of the Weight Matrix is visited, we say that the corresponding flow f_i is assigned a *time-slot*. We use a variable $TS(t)$ to count the number of time-slots assigned in SRR at time t. TS is set to 0 when SRR starts (i.e., $TS(0) = 0$), and is increased by 1 when a non-zero term of the Weight Matrix is visited.

Suppose a flow f with weight 2^n, i.e., $r = 2^n$, arrives at the scheduler at time t_0, and the corresponding element $k-n$ in WSS has been scanned j times at t_0. We denote the value of TS at the time when f has been served i times as $TS(t_i)$.

Definition 2: We define $TS_n(i) = TS(t_i) - TS(t_0)$, which is the number of assigned time-slots in SRR from the time when flow f (with rate 2^n) arrives to the time when f is served $i \, (i \geq 1)$ times.

As to $TS_n(i)$, we have lemma as follows.

Lemma 1: For a flow f with rate 2^n in SRR, from the time when f arrives at the scheduler to the time when f is served i times, the value of $TS_n(i)$ is bounded by

$$TS_n(i) \leq \frac{iC}{2^n} + \theta(n)N. \tag{10}$$

where $-n < \theta(n) < n$.

Based on Lemma 1, we get the following results on the delay property of SRR.

Lemma 2: For a flow f with rate 2^n $(0 \leq n < k)$, its delay bound in SRR is

$$d_f^\star \leq \theta(n) \times N \times \frac{L}{C} \tag{11}$$

Similar result exists for flows with arbitrary rate. We have the following theorem:

Theorem 1: In SRR, for a flow f with rate $r_f = w_f = \sum_{i=1}^{m} 2^{n_i}$ $(0 \leq n_1 < n_2, \cdots, < n_m \leq k-1)$,

$$d_f^\star < \theta(n_m) \times N \times \frac{L}{C} + (m-1)\frac{L}{r_f} \tag{12}$$

where C is the bandwidth of the output link, m is the number of non-zero coefficients of r_f.

The proofs of Lemma 1 and 2 and Theorem 1 can be found in [8], and are omitted here due to space limitation. Theorem 1 shows that the delay of SRR is in direct proportion to the number of active flows.

III. SRR$^+$

A. The SRR$^+$ Scheduler

The key idea of SRR$^+$ is to introduce a matrix transform to normalize the Weight Matrix, so that the weight matrix becomes more regular and the skewed weight distributions can be handled. In SRR$^+$, we sum up the weights of each column of the Weight Matrix to compose k new flows $f_0^+, f_1^+, \cdots, f_{k-1}^+$. The weight of flow $f_i^+ (0 \leq i \leq k-1)$ is therefore

$$w_i^+ = 2^i \sum_{j=0}^{N-1} a_{j,i} = 2^i y_i = \sum_{j=0}^{k-1} a_{i,j}^+ 2^j \tag{13}$$

We then use the weights of the k composed flows to form a new $k \times k$ square matrix, WM$^+$, as illustrated in Figure 1.

Procedure 1: Matrix transform to generate WM^+
Input: a $N \times k$ matrix WM;
Output: a $k \times k$ matrix WM^+;
for $(i = 0; \, i < k; \, i++)$ {
  $w_i^+ = 2^i \sum_{j=0}^{N-1} a_{j,i} = \sum_{j=0}^{k-1} a_{i,j}^+ 2^j$;
  use $a_{i,j}^+ (0 \leq j < k)$ to form the ith row of WM^+;
}

Fig. 1. Procedure to generate the WM$^+$ matrix.

For WM^+, the following lemma holds.

Lemma 3: For a $(k \times k)$ WM$^+$ matrix, $a_{i,j}^+ = 0$ for $j < i$.

Proof: From Equation (13), we have $w_i^+ = 2^i y_i$. The binary coefficients $a_{i,j}^+$ therefore is 0 for $0 \leq j < i \leq (k-1)$. The lemma is therefore proved. ■

From Lemma 3, we know that WM$^+$ is a triangular matrix, and can be represented as follows.

$$WM^+ = \begin{bmatrix} a_{0,(k-1)}^+ & a_{0,(k-2)}^+ & \cdots & a_{0,1}^+ & a_{0,0}^+ \\ a_{1,(k-1)}^+ & a_{1,(k-2)}^+ & \cdots & a_{1,1}^+ & 0 \\ \vdots & \vdots & \ddots & \vdots & \\ a_{(k-1),(k-1)}^+ & 0 & \cdots & 0 & 0 \end{bmatrix} \tag{14}$$

By this matrix transforming, we transform the irregular $(N \times k)$ matrix WM$^+$ into a regular $(k \times k)$ matrix WM$^+$.

As in [7], in order to reduce time complexity, we introduce doubly linked list (DL) to link the non-zero terms of WM and WM$^+$. Each node of a DL list has three fields: $next$, $prev$, and fid, with $next$ points to the next node, $prev$ points to the previous node, and fid is the flow id. For WM$^+$, the non-zero terms of column i are linked together to form doubly linked list DL_i^+, with the head node named $head^+[i]$, and the tail node named $tail^+[i]$. The $prev$ field of the head node and the $next$ field of the tail node are set to $NULL$. When the terms of a column i are all 0, $head^+[i] = NULL$

The double links $\{DL_i | 0 \leq i < k\}$ that correspond to WM have the same structure as that of WM$^+$, except that the $next$ field of the last node of a double link points to the first node instead of $NULL$ (i.e., the double linked lists DL_is of WM are circular lists).

The formal description of SRR$^+$ is given in Fig. 2. In SRR$^+$, there is a pointer p_w points to the current scanned position of the WSS sequence, p_w is initialized to 1. There is a pointer P^+ points to the current scanned non-zero term of the WM$^+$ matrix. For each DL_i, there is also a pointer $P[i]$ to remember the current scanned term of DL_i. $P[i]$ is initialized to point to the first node of DL_i.

SRR^+ then performs scheduling as follows. SRR$^+$ scans the WSS sequence term by term. When the current WSS term is i, column $col^+ = k - i$ of the WM$^+$ matrix is selected (line 1). Then from the head to tail, SRR$^+$ traverses $DL^+[col^+]$ (lines 2-3). SRR^+ then gets the *fid* of the current node of $DL^+[col^+]$ (line 4), uses this *fid* as an index to the column number of the WM matrix, and gets the flow id f of the current node of that column of WM (line 5). Flow f is then selected.

If f is backlogged (i.e., there are packets queued in the system) (line 6), *ServFlow* is then invoked to serve the flow (line 7). The task of ServeFlow(f) is to dequeue the first packet of f from the queue, then transmit it.

If f is not backlogged, *idle_sched* is called (line 8-9). In *idle_sched*, the schedule opportunity is assigned to the best-effort flow. In SRR$^+$, the best-effort flow is the flow that does not have explicit bandwidth requirement.

Lines 10, 11, 13 are to adjust the pointers $P[col]$, P^+, and p_w.

When a new flow f comes, *flow_add* is called. *flow_add* first expresses the rate of f into binary form (line 14), then for each non-zero binary coefficients a_i (line 15), the ith row of WM$^+$ needs to be regenerated by calling *gen_row_wm*$^+$ (line 16), and a new node needs to be appended to the tail of DL_i (i.e., the list that represents the ith column of WM) (line 17). If new columns are added into WM$^+$, then the order of the WSS sequence and the value of p_w need adjustment (lines 19-22). The order of WSS is adjusted to the column number of WM$^+$. When the order of WSS increases from k to j, the net effect is that we multiple p_w by 2^{j-k}.

When a flow f leaves, *flow_delete* is invoked. For each non-zero binary coefficient a_i (lines 23-24), the ith row of WM$^+$ needs to be regenerated (line 25), and the node represents a_i in DL_i needs to be deleted (line 26). If empty columns are removed from WM$^+$, the order of WSS and the value of p_w also needs to be adjusted (lines 28-31).

```
Schedule:
   while (in busy-period) {
1    i = WSS^k[p_w]; col^+ = k − i;
2    P^+ = head^+[col^+];
3    while (P^+ ≠ NULL) {
4       col = P^+ → fid;
5       f = P[col] → fid;
6       if ( f is backlogged)
7          ServeFlow(f);
8       else
9          idle_sched();
10      P[col] = P[col] → next;
11      P^+ = P^+ → next;
12   }
13   p_w = 1 + p_w mod (2^k − 1);
   }
flow_add(f, r_f)
14    r_f = Σ_{i=0}^{k−1} a_i 2^i;
15    for (each binary coefficient a_i ≠ 0) {
16       gen_row_WM^+(i);
17       append a new node at the tail of DL_i;
18    }
19    if(new columns are added into WM^+){
20       p_w = p_w 2^{j−k}; /* j is the new order of WSS */
21       k = j;
22    }
flow_delete(f, r_f):
23    r_f = Σ_{i=0}^{k−1} a_i 2^i;
24    for (each binary coefficient a_i ≠ 0) {
25       gen_row_wm^+(i);
26       delete node that represents a_i from DL_i;
27    }
28    if(empty columns are removed from WM^+) {
29       p_w = ⌈p_w / 2^{k−j}⌉ /* j is the new order of WSS */;
30       k = j;
31    }
```

Fig. 2. Description of the SRR$^+$ packet scheduler.

In both *flow_add* and *flow_delete*, *gen_row_wm*$^+$ is used to update a row of WM$^+$. The *gen_row_wm*$^+$ procedure is illustrated as follows.

```
gen_row_wm^+(i):
1    w_prev = Σ_{j=0}^{k−1} a^+_{i,j} 2^j;
2    w_curr = Σ_{j=0}^{k−1} b^+_{i,j} 2^j;
3    for (j = 0; j < k; j++) {
4       if (a^+_{i,j} == 1 and b^+_{i,j} == 0) {
5          if (P^+ points to a^+_{i,j})
6             P^+ = P^+ → next;
7          remove node that represents a^+_{i,j} from DL^+_j;
8       }
```

```
9       if (a+_{i,j} == 0 and b+_{i,j} == 1)
10          insert new node that represents b+_{i,j} to DL+_j;
11    }
```

Row i of WM$^+$ represents the sum of weights of the terms in column i of WM. In *gen_row_wm*$^+$, we use w_{prev} and w_{curr} to represent the weights of column i of WM before and after adjustment, respectively (lines 1-2). We then compare the binary coefficients of w_{prev} and w_{curr} (line 3). If the previous coefficient $a_{i,j} = 1$ and the current coefficient $b^+_{i,j} = 0$, then the node that represents $a^+_{i,j}$ needs to be removed from DL^+_j (line 7). In this case, we also need to judge if P^+ points to an empty node. If that is the case, we need to put P^+ into the right place (lines 5-6). If $a^+_{i,j} = 0$ and $b^+_{i,j} = 1$, new node needs to be inserted into DL^+_j (lines 9-10).

We use an example to illustrate how SRR$^+$ works. Suppose we have 7 flows numbered from f_0 to f_6 with rate 1, 3 flows numbered from f_7 to f_9 with rate 2, 1 flow f_{10} with rate 4. Then the corresponding WM and WM$^+$ matrices are as follows.

$$WM = \begin{bmatrix} 0 & 0 & 1 \\ & \cdots & \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \rightarrow WM^+ = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

Since the generated WM$^+$ has 3 columns, a WSS of order 3, $\{1,2,1,3,1,2,1\}$, is used. The first term of WSS^3 is 1, based on the scheduling procedure of SRR$^+$, column 3-1=2 of WM$^+$ is selected. SRR$^+$ then scans column 2 of WM$^+$ from top to bottom. The first non-zero term is at row 0, then column 0 of WM is selected, the first non-zero term of column 0 corresponds to f_0, f_0 is therefore chosen and served. The second non-zero term of column 2 of WM$^+$ is at row 1, then column 1 of WM is selected, the first non-zero term is at row 7, f_7 is then selected and served. Similarly, the next flow to serve is then f_{10}. After that, SRR$^+$ has scanned all non-zero terms of column 2 of WM$^+$. It then increases p_w to scan the next term of WSS, $\cdots$. Using the scheduling procedure, we get the service sequence in a round

$$f_0 f_7 f_{10} f_1 f_8 f_2 f_9 f_{10} f_3 f_4 f_7 f_{10} f_5 f_8 f_6 f_9 f_{10},$$

Whereas the corresponding service sequence of SRR is

$$f_{10} f_7 f_8 f_9 f_{10} f_0 f_1 f_2 f_3 f_4 f_5 f_6 f_{10} f_7 f_8 f_9 f_{10}$$

From the above two sequences, we observe that the low weight flows $f_0 - f_6$ are interleaved more evenly in SRR$^+$ than in SRR.

Readers can also check that SRR$^+$ solves the skewed weight distribution example in Section II. The scheduling output of that example is $f_1 f_0 f_2 f_0 \cdots f_{2^{n-1}} f_0$. f_0 therefore needs to wait for only one flow instead of 2^{n-1} flows before its first packet gets served.

B. Delay Property of SRR$^+$

Lemma 4: When WM is fixed, for a flow f with rate 2^n $(0 \le n \le k-1)$ in SRR$^+$,

$$d^+_f \le \theta(k-1) \times k \times \frac{L}{C}. \tag{15}$$

Proof: Based on the constructing procedure of WM$^+$, flow f must be represented by a term $a^+_{g,m}$ $(0 \le g \le k-1, m \ge n)$ in WM$^+$.

Flow f will be served once when $a^+_{g,m}$ is served every 2^{m-n} times. We therefore have

$$T_{f,n}(i) = T_{g,m}(i2^{m-n} - \delta)$$

where δ is a constant, and $0 \le \delta < 2^{m-n}$.

From Equation (10), we have

$$T_{g,m}(i) \le \frac{iC}{2^m} + \theta(m) \times k,$$

since the maximum number of flows in WM^+ is k.

Thus,

$$\begin{aligned} T_{f,n}(i) &\le \tfrac{(i2^{m-n}-\delta)C}{2^m} + \theta(m) \times k \\ &= \tfrac{i \times C}{2^n} + \theta(m) \times k - \tfrac{\delta C}{2^m} \\ &\le \tfrac{i \times C}{2^n} + \theta(m) \times k \end{aligned}$$

Therefore,

$$\begin{aligned} d^+_f(i) &= t_0 + T_{f,n}(i)\tfrac{L}{C} - (t_0 + i\tfrac{L}{2^n}) \\ &\le \theta(m) \times k \times \tfrac{L}{C} \end{aligned}$$

Since $m \le k-1$, we therefore have

$$d^+_f \le \theta(k-1) \times k \times \frac{L}{C}.$$

■

Based on Lemma 4, we get delay bound for flows with arbitrary reserved rates. We have theorem as follows.

Theorem 2: When WM is fixed, for a flow f with rate $r_f = w_f = \sum_{l=1}^{m} 2^{n_l}$ $(0 \le n_1 < n_2, \cdots, n_m \le k-1)$, its delay bound is

$$d^+_f \le \theta(k-1) \times \frac{kL}{C} + \frac{mL}{r_f} - \frac{L}{C} \tag{16}$$

Proof: Our proof strategy is to decompose the flow f into a set of flows $\{f_1, f_2, \cdots, f_m\}$ with rates $\{2^{n_1}, 2^{n_2}, \cdots, 2^{n_m}\}$. Then by using Lemma 4 and Equation 3, the service curve of flow f_i, S_{f_i}, has inequality as follows.

$$S_{f_i}(t) > S^{id}_{f_i}(t - d^+_{f_i} - \frac{L}{2^{n_i}} + \frac{L}{C})$$

The service curve of f can be gotten by summing up the service curves of the m flows. Therefore, in SRR$^+$

$$\begin{aligned} S_f(t) &= \textstyle\sum_{i=1}^{m} S_{f_i}(t) > \sum_{i=1}^{m}\{2^{n_i}(t - d^+_{f_i} - \frac{L}{2^{n_i}} + \frac{L}{C})\} \\ &\ge \textstyle\sum_{i=1}^{m}\{2^{n_i}t\} - \sum_{i=1}^{m}\{2^{n_i}\theta(k-1) \times k \times \frac{L}{C}\} \\ &\quad -mL + \textstyle\sum_{i=1}^{m}\{2^{n_i}\frac{L}{C}\} \\ &= r_f(t - \theta(k-1)\tfrac{k \times L}{C} - \tfrac{mL}{r_f} + \tfrac{L}{C}) \end{aligned}$$

By using Equation (4), we therefore get

$$d^+_f \le \theta(k-1) \times \frac{kL}{C} + \frac{mL}{r_f} - \frac{L}{C}$$

The theorem is therefore proved. ■

C. Time and Space Complexities of SRR$^+$

From Fig. 2, it is easy to observe that *Schedule* can be carried out in strict $O(1)$ time to choose a flow to serve. The time complexities of *flow_add* and *flow_delete* are all $O(k^2)$. This is because that when a flow is added or removed, the WM^+ matrix needs to be re-generated. In the worst-case, one needs k^2 steps to re-generate this $k \times k$ matrix. The time complexity of SRR$^+$ is therefore comparable with that of SRR.

The space complexity of SRR$^+$ is $O(2^k + N \times k + k^2)$, where 2^k is the space to store the WSS sequence, $N \times k$ is the space to hold the WM matrix, and k^2 is the space to store the $k \times k$ WM$^+$ matrix. $O(N \times k)$ is inevitable for all FQ packet schedulers, since it is the lower bound to hold the reserved rates for all the active flows. The space needed to hold the WSS sequence increases exponentially as k increases. Similar with SRR, SRR$^+$ can use a WSS sequence of order $n+1$ to produce a WSS sequence of order $2n$. By using this technique, the space to store the WSS sequence reduces from $O(2^k)$ to $O(2^{\frac{k}{2}})$, and the scheduler needs one additional step to access the terms of the WSS sequence [7].

As compared with SRR, SRR$^+$ needs additional k^2 space to store the WM$^+$ matrix. This additional space is not an issue, since k^2 is a small number even when k is as large as 32.

SRR$^+$ therefore provides much better delay guarantee than SRR, with minimal increase in space and time complexities.

IV. SRR$^\#$

A. The SRR$^\#$ Scheduler

The Matrix Transforming idea explored in SRR$^+$ can be performed more than once. In what follows we present a new scheduler, which we call SRR$^\#$ by iteratively transforming the Weight Matrix to more regular forms.

We introduce a new definition as follows.

Definition 3: A $k \times k$ matrix is defined to be a *diagonal* matrix, if its terms $a_{i,j} (0 \le i, j \le k-1)$ satisfy

$$a_{i,j} = \begin{cases} 0, & i \neq j \\ 0 \text{ or } 1, & i = j \end{cases} \tag{17}$$

It is easy to observe that the sum of each column of a *diagonal* matrix, $y_i \le 1 \, (0 \le i \le k-1)$.

We then use Procedure 2 as depicted in Fig. 3 to construct a serial of weight matrices and a diagonal matrix $WM^\#$.

Procedure 2: Iterative matrix transforms to generate $WM^\#$
Input: $WM^0 = WM^+$;
Output: $WM^\#$ and $\{WM^i | 1 \le i \le k-1\}$;
for $(i = 1; i < k; i++)$ {
  for $(j = 0; j < k; j++)$ {
    $w_j = 2^j \sum_{l=0}^{k-1} a_{l,j}^{i-1}$;
    use w_j to generate jth row of WM^i;
  }
}
$WM^\# = WM^i$;

Fig. 3. Procedure to produce a serial of transformed matrices and the WM$^\#$ matrix.

In Procedure 2, WM^+ is the matrix that defined by Equation (14), $a_{l,j}^i$ is the term at row l and column j of WM^i. It is possible to generate WM$^\#$ in less than $k-1$ iterations for certain input matrices, but $k-1$ iterations is needed in the worst-case. We have the following lemma for WM$^\#$:

Lemma 5: The output matrix WM$^\#$ of Procedure 2 is a $diagonal$ matrix.

Proof: First we note that the matrix gotten after one matrix transform of a triangular matrix is still a triangular matrix.

From Lemma 3, we know that for a WM$^+$ matrix, $a_{j,0}^+ = 0$ for $j \neq 0$. Therefore, after the first matrix transform, we have $a_{0,l}^1 = 0$ for $l \neq 0$. And Since WM^1 is also a triangular matrix, we have $a_{j,0}^1 = 0$ for $j \neq 0$.

Similarly, we get that after the second iteration, $a_{1,l}^2 = a_{j,1}^2 = 0$ for $j \neq 1, l \neq 1$. And iteratively, we have $a_{(i-1),l}^i = a_{j,(i-1)}^i = 0$ for $l \neq i-1$ and $j \neq i-1$ after the ith iteration.

We therefore get a diagonal matrix after $k-1$ matrix iterations. Since WM$^\#$ = WM^{k-1}, WM$^\#$ is therefore a diagonal matrix.

■

$WM^\#$ therefore can be denoted as follows.

$$WM^\# = \begin{bmatrix} 0 & 0 & \cdots & 0 & a_{0,0}^\# \\ 0 & 0 & \cdots & a_{1,1}^\# & 0 \\ \vdots & \vdots & \ddots & \vdots & \\ a_{k-1,(k-1)}^\# & 0 & \cdots & 0 & 0 \end{bmatrix} \tag{18}$$

where $a_{i,i}^\# \in \{0, 1\}$ $(0 \le i \le k-1)$ and $a_{(k-1),(k-1)}^\# = 1$.

There are $k+1$ matrices in Procedure 2: WM, WM^0, WM^1, $\cdots$, WM^{k-1} (WM^0 is WM$^+$ and WM^{k-1} is WM$^\#$). From the proof of Lemma 5, we get that WM^i is more "regular" than WM^{i-1}. We are therefore able to provide a serial of schedulers by using these matrices. For example, by using WM^1, WM$^+$, and WM, a new scheduler can be designed. These scheduler are similar in spirit. We only present SRR$^\#$, which uses all the matrices and provides the best delay bound.

Similar with SRR$^+$, SRR$^\#$ uses doubly linked lists to denote the matrices. For each column col of WM^i, there is a pointer $P^i[col]$ points to the current scanned node. The formal description of SRR$^\#$ is given in Fig. 4.

The *Schedule* procedure is similar with that of SRR$^+$, except that more matrices need to be visited. When the current term of WSS is i, column $k-i$ of WM$^\#$ is selected (line 1). Since the only term that can be non-zero is $a_{i,i}^\#$ in this column, SRR$^\#$ checks this term directly (line 2). If the term is not zero, SRR$^\#$ then iteratively visit the $k-1$ matrices. The column number of the next visited matrix is identified by the *fid* of the current DL node (lines 3-5). After that, the column number of WM can be identified, and the flow to be served is selected (line 7). The rest of *Schedule* is similar with that of SRR$^+$ (lines 8-14).

When a new flow f with reserved rate r_f comes, *flow_add* is called. WM and the rest k matrices are regenerated (lines 16-18). The order of WSS and the pointer p_w is adjusted accordingly (lines 19-22).

When a flow f leaves, *flow_delete* is invoked to remove the flow. The matrices are updated accordingly (lines 24-26) and the order of WSS and the pointer p_w are also updated (27-30). Note that the details of adjusting the pointers $P^i[col]$ in *flow_add* and *flow_delete* are omitted in Fig. 4 for description simplicity. These details are similar to those in Fig. 2.

Schedule:
while (in busy-period) {
1 $i = WSS^k[p_w]$; $i = col = k - i$;
2 if ($a^{\#}_{i,i} == 1$) {
3 for ($j = k-1; j \geq 0; j--$) {
4 $col = P^j[col] \rightarrow fid$;
5 $P^j[col] = P^j[col] \rightarrow next$;
6 }
7 $f = P[col] \rightarrow fid$;
8 if(f is backlogged)
9 ServeFlow(f);
10 else
11 idle_sched();
12 $P[col] = P[col] \rightarrow next$;
13 }
14 $p_w = 1 + p_w \, mod \, (2^k - 1)$;
}
flow_add(f, r_f):
15 $r_f = \sum_{i=0}^{k-1} a_i 2^i$;
16 use the coefficients a_i to update WM;
17 generate WM$^+$ using Procedure I;
18 generate WM$^{\#}$ and WM^{i}s using Procedure II;
19 if(new columns are added into WM$^{\#}$){
20 $p_w = p_w 2^{j-k}$; /* j is the new order of WSS */
21 $k = j$;
22 }
flow_delete(f, r_f):
23 $r_f = \sum_{i=0}^{k-1} a_i 2^i$;
24 use the coefficients a_i to update WM;
25 generate WM$^+$ using Procedure I;
26 generate WM$^{\#}$ and WM^{i}s using Procedure II;
27 if(empty columns are deleted from WM$^{\#}$) {
28 $p_w = \lceil \frac{p_w}{2^{k-j}} \rceil$ /* j is the new order of WSS */;
29 $k = j$;
30 }

Fig. 4. Description of the SRR$^{\#}$ packet scheduler.

We use the same example as in Section III to illustrate how SRR$^{\#}$ works. The matrices in SRR$^{\#}$ are generated as follows.

$$WM \rightarrow WM^+ \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \rightarrow WM^1 \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

$$\rightarrow WM^2 \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \rightarrow WM^3 \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\rightarrow WM^{\#} \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Since the number of columns of WM$^{\#}$ is 5, a WSS sequence of order 5 is then used. SRR$^{\#}$ performs scheduling as follows. The first term of WSS^5 is 1, column 4 of WM$^{\#}$ is selected. The first (in fact, the only) non-zero term of column 4 of WM$^{\#}$ is at row 4. Column 4 of WM^3 is then selected. The first (in fact, the only) non-zero term of column 4 of WM^3 is at row 3. Column 3 of WM^2 is then selected. The first non-zero term of column 3 of WM^2 is at row 2. Column 2 of WM^1 is then selected. The first non-zero term of of column 2 of WM^1 is at row 1. Column 1 of WM$^+$ is then selected. The first non-zero term of column 1 of WM$^+$ is at row 0. Column 0 of WM is then selected. The first non-zero term of column 0 is at row 0, f_0 is therefor selected and served. Similarly, the next flow to serve is f_1. The service sequence of SRR$^{\#}$ in a round is

$$f_0 f_1 f_7 f_{10} f_8 f_2 f_9 f_{10} f_3 f_4 f_5 f_7 f_{10} f_8 f_6 f_9 f_{10}$$

B. Delay Property of SRR$^{\#}$

Lemma 6: When WM is fixed, for a flow f with rate 2^n $(0 \leq n \leq k-1)$ in SRR$^{\#}$,

$$d_f^{\#} \leq \theta(k-1) \times \frac{L}{C} \tag{19}$$

The proof of this lemma is similar with that of Lemma 4. Note that the delay provided by (19) is better than (15) of SRR$^+$. The reason is that WM$^{\#}$ is more "regular" than WM$^+$ (note that WM$^{\#}$ is a diagonal matrix whereas WM$^+$ is a triangular one). Better delay bound is the motivation that we transform WM$^+$ into WM$^{\#}$.

Based on Lemma 6, for flow with arbitrary rate, we have the following theorem.

Theorem 3: When WM is fixed, for a flow f with rate $r_f = w_f = \sum_{l=1}^{m} 2^{n_l}$ $(0 \leq n_1 < n_2, \cdots, n_m \leq k-1)$, its d_f^+ is bounded by

$$d_f^{\#} \leq \theta(k-1) \times \frac{L}{C} + \frac{mL}{r_f} - \frac{L}{C} \tag{20}$$

The proof of this theorem is similar with that of Theorem 2. Theorem 3 shows that the delay bound of SRR$^{\#}$ is similar with that of G-3 and SRR$^+$. From (16) and (20), we get that the delay bound of SRR$^{\#}$ is reduced by $\theta(k-1)\frac{(k-1)L}{C}$ as compared with that of SRR$^+$. The significance of the reduction, however, depends on the values of C and r_f. When C is large or r_f is small, the major contribution of the delay comes from $\frac{L}{r_f}$. For low bit rate flows, SRR$^{\#}$ may not be a good choice, since the gain in delay bound may not be comparable with the loss in time complexity.

Note that in Theorems 2 and 3, we require that the Weight Matrix is fixed. The WM$^+$ and WM$^{\#}$ matrices may change as flows arrive and leave. The locations of the time-slots assigned to a flow may change accordingly, which may result in increased worst-case delays. We leave the delay analysis when flows dynamically join and leave for further research.

C. Time and Space Complexities of SRR$^{\#}$

The time complexity of *Schedule* is $O(k)$ since SRR$^{\#}$ needs to traverse the k matrices to identify which column of WM to visit and then decide which flow to serve. This is a significant increase as compared with $O(1)$ time complexity of SRR^{+}, since packet scheduling must be performed for every packet. The time complexities of *flow_add* and *flow_delete* are $O(k^3)$, since when a flow is added or removed, the k matrices must be regenerated.

The space complexity of SRR$^{\#}$ is $O(2^k + N \times k + k^3)$, where k^3 is the space needed to store the k generated $k \times k$ matrices from the Weight Matrix. As compared with SRR^{+}, the increased space is to store the generated matrices from WM^{+} to WM$^{\#}$. k^3 is also not a big number. For example, when $k = 32$, k^3 is only 32768. Hence, the space complexity of SRR$^{\#}$ is also not an issue.

V. Simulation

We have implemented SRR^{+} and SRR$^{\#}$ in *NS*$_2$ [11]. In the network as depicted in Fig. 5, the bandwidths and propagation delays of R_0R_1 and R_1R_2 are 10 Mb/s and 10ms, respectively. The bandwidths and delays are 100 Mb/s and 1 ms for the rest of the links. The MTU of the network is 200 bytes and the reserved rate of a flow is set to its sending rate.

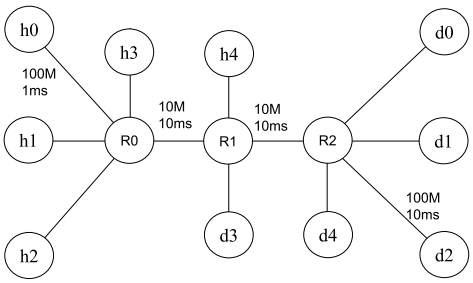

Fig. 5. Network topology of the simulation.

We setup a CBR flow f_1 between h_0 and d_0, a CBR flow f_2 with rate 1024 Kb/s between h_1 and d_1. There are 500 flows with rate 16 Kb/s between h_2 to d_2. There are also two best-effort flows (f_0) from h_3 to d_3 and h_4 to d_4 to occupy the unallocated bandwidth. The two best-effort flows are generated from two Pareto sources with mean on and off time 100 ms, and $\alpha = 1.5$. The average rate of each Pareto source is 2 Mb/s, which is larger than the unallocated bandwidth of the network.

We then vary the rate of f_1 from 32kb/s to 1024k/bs and measure its end-to-end delays under G-3, SRR^{+}, SRR$^{\#}$, and SRR, respectively. The result is shown in Fig. 6.

Fig. 6 shows that G-3, SRR^{+}, and SRR$^{\#}$ outperform SRR significantly. This is not a surprise, since the delay bounds of all the schedulers except SRR are not depend on the number of active flows in the scheduler. The maximum delay of G-3, SRR^{+}, and $SRR^{\#}$ is strictly in inverse proportion to the reserved rate of the flow. For example, the maximum delays of f_1 under G-3, SRR^{+}, and SRR$^{\#}$ are (88ms, 101ms, 104ms) and (25.6ms, 25.8ms, 24.9ms) for rates 32kb/s and

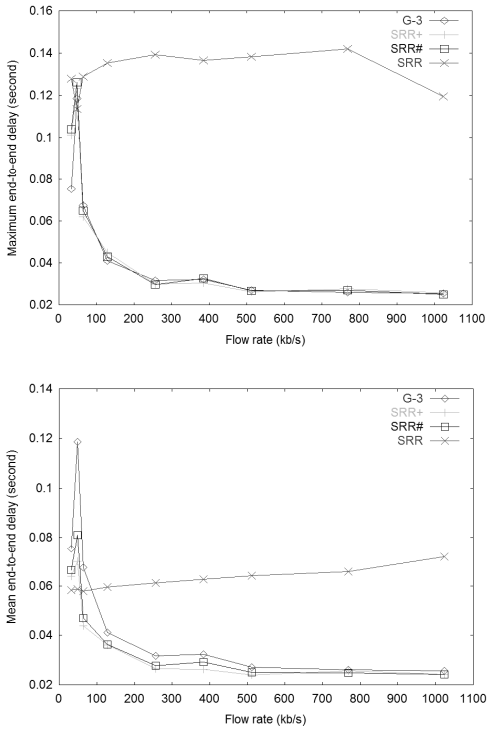

Fig. 6. Maximum and mean end-to-end delays for different reserved flow rates under G-3, SRR, SRR^{+}, and SRR$^{\#}$.

1024kb/s, respectively. All are smaller than their respective delay bounds. The simulation conforms to our analysis that the delay properties of G-3, SRR^{+}, and SRR$^{\#}$ are similar, and is not affected by the number of active flows.

VI. Related Work

Due to their low time-complexity nature, round-robin packet schedulers are very suitable to be implemented in high-speed routers to provide bandwidth and delay guarantees. For this reason, many round-robin packet schedulers have been proposed recently [4], [6], [7], [9], [10], [13], [14], [16], [18].

To reduce the burstiness of WSS, in Aliquem [10] and PDRR [16], instead of using only one linked list, multiple lists are introduced. Flows are placed to different lists based on their weights. The schedulers visit the lists according to certain rules. The schedulers are able to provide better delay bound as compared with WSS. But the scheduling time complexities of these schedulers are larger than $O(1)$ and the delay bounds are still in proportion to the number of active flows.

Recently, several round-robin schedulers try to improve WSS using a hybrid approach [4], [13], [18]. These schedulers cluster flows with similar weights into a same group (or class). A time-stamp based approach is used for inter-group scheduling, and a round-robin based approach is used for intra-group scheduling. These schedulers generally reduce the output burstiness and are still of low time complexity. For example, GRRR [4] achieves $O(1)$ time complexity for packet scheduling and a delay bound that is in proportion to g^2, where g is the number of groups. The matrix transform introduced in SRR^{+} can be considered a specific form of

grouping: instead of assigning a flow to one group, SRR$^+$ assigns the flow to multiple "groups" (i.e., multiple rows of WM$^+$) according to the non-zero coefficients of the reserved rate. The "grouping" method used in SRR$^+$ is therefore more accurate. Moreover, SRR$^+$ does not need to maintain time-stamps to track the progress of each group, resulting in more efficient implementation for high-speed routers (e.g., SRR$^+$ does not need to perform division to calculate time-stamps and comparison of time-stamps as compared with GRRR and STRR).

RRR [6] uses a binary tree structure to track the bandwidth allocation, and performs scheduling by recursively traversing the tree. RRR is able to achieve bounded delay. Different from the tree structure in RRR, the Weight Matrix used in SRR$^+$ and SRR$^\#$ achieves a serial of packet schedulers that provide tradeoffs among delay, space-complexity, and time complexity.

The recently developed G-3 [9] scheduler achieves both $O(1)$ time complexity and bounded end-to-end delay. G-3 is built over SRR [7] and RRR [6]. G-3 takes advantages of two key data structures, Weight Spread Sequence (WSS, which is introduced in SRR) and Weighted Binary Tree (WBT, which is introduced in RRR). To achieve $O(1)$ time complexity, G-3 introduces a time-slot array (TArray) to expand each weighted binary tree (WBT). It also introduces a novel *binary_reversal* operation to update the TArrays when flows arrive and leave. The space needed for the TArrays is $C/gran$, where C is the bandwidth of the output link and $gran$ is the bandwidth allocation granularity. The space complexity of G-3 is therefore $O(C+N)$. When C is large, the space complexity of G-3 may be much higher than that of SRR$^+$ and SRR$^\#$. For example, for a 40Gb/s network interface with rate granularity of 1Kb/s, the space needed to store the Time-slot Arrays is 160 MB (assume 4 bytes to store a flow id).

G-3, however, has one advantage that SRR$^+$ and SRR$^\#$ do not have. The time-slots assigned to a flow is fixed in that the addition and removal of flows do not affect the time-slot locations of an existing flow. *Schedule*, *flow_add*, and *flow_delete* therefore can be executed simultaneously, whereas in SRR$^+$ and SRR$^\#$, *Schedule* must synchronize with *flow_add* and *flow_delete*, since the matrices that are used by *Schedule* must be regenerated when flows arrive and leave. And matrices regenerations may change the time-slot locations of an existing flow and affect its delay. From these viewpoints, G-3 is simpler and has better delay property than SRR$^+$ and SRR$^\#$.

SRR, SRR$^+$, SRR$^\#$, and G-3 all use the WSS sequence and Weight Matrix as their central data structures (G-3 also introduces other data structures). They provide a full spectrum of round-robin schedulers that have different delay, space complexity, and time complexity tradeoffs. When strict delay bound is not needed and $O(1)$ time-complexity is required, SRR is a good choice; when strict delay bound is needed, $O(1)$ time-complexity is required, and space is not an issue, G-3 is the best choice; when space is an issue and low time-complexity and delay bound is needed, SRR$^+$ is the choice; when space is an issue, delay bound is needed, but low time-complexity is not required, SRR$^\#$ can be considered.

VII. Conclusion

We have presented two packet schedulers, SRR$^+$ and SRR$^\#$, which significantly improve the delay bound of the original SRR scheduler. By transforming the Weight Matrix into more regular triangular and diagonal ones, the burstiness of scheduling output can be significantly reduced even for skewed weight distributions, whereas the scheduling simplicity is preserved. Both analysis and simulations show the superior delay properties of SRR$^+$ and SRR$^\#$. As compared with the recently developed G-3 packet scheduler, SRR$^+$ and SRR$^\#$ have much lower space complexity, but have larger time complexity and delay bound. These three schedulers together with SRR provide a full spectrum of fair queueing packet schedulers that have different delay, space-complexity and time-complexity tradeoffs for high-speed networks.

References

[1] Jon C. R. Bennett and Hui Zhang. WF^{2}Q: Worst-case fair weighted fair queueing. In *Proc. infocom*, 1996.

[2] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. *An architecture for differentiated services (RFC 2475)*, Dec. 1998.

[3] R. Braden, D. Clark, and S. Shenker. *Integrated services in the internet architecture: an overview (RFC 1633)*, June 1994.

[4] Bogdan Caprita, Wong Chun Chan, Jason Nieh, Clifford SteinC, and Haoqiang Zheng. Group ratio round-robin: O(1) proportional share scheduling for uniprocessor and multiprocessor systems. In *Proc. USENIX Annual Technical Conference*, 2005.

[5] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *Proc. SIGCOMM*, 1989.

[6] Rahul Garg and Xiaoqiang Chen. RRR: Recursive round robin scheduler. *Computer Networks*, 31(18):1951–1966, Aug. 1999.

[7] Chuanxiong Guo. SRR: An O(1) time complexity packet scheduler for flows in multi-service packet networks. *IEEE/ACM trans. Networking*, 12(6):1144–1155, Dec. 2004.

[8] Chuanxiong Guo. G-3: An $O(1)$ time complexity packet scheduler that provides bounded end-to-end delay. Technical report, Inst. of Commu. Engi., June 2006. http://edu.jsvnet.com/%7Exguo/tr_g3.pdf.

[9] Chuanxiong Guo. G-3: An $O(1)$ time complexity packet scheduler that provides bounded end-to-end delay. In *Proc. infocom*, 2007.

[10] Luciano Lenzini, Enzo Mingozzi, and Giovanni Stea. Tradeoffs between low complexity, low latency, and fairness with deficit round-robin schedulers. *IEEE/ACM trans. Networking*, 12(4):681–693, Aug. 2004.

[11] NS2. The NS network simulator. http://www.isi.edu/nsnam/ns.

[12] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services network: The single node case. *IEEE/ACM trans. Networking*, 1(3), June 1993.

[13] Sriram Ramabhadran and Joseph Pasquale. Stratified round robin: A low complexity packet scheduler with bandwidth fairness and bounded delay. In *Proc. SIGCOMM*, 2003.

[14] M. Shreedhar and G. Varghese. Efficient fair queuing using deficit round robin. In *Proc. SIGCOMM*, pages 231–242, 1995.

[15] Dimitrios Stiliadis and Anujan Varma. Latency-rate servers: A general model for analysis of traffic scheduling algorithms. *IEEE/ACM trans. Networking*, 6(5):611–624, Oct. 1998.

[16] S.-C. Tsao and Y.-D. Lin. Pre-order deficit round-robin: A new scheduling algorithm for packet switched networks. *Computer Networks*, 35:287–305, Feb. 2001.

[17] A. Varma and D. Stiliadis. Hardware implementation of fair queuing algorithms for asynchronous transfer mode networks. *IEEE Communication Magazine*, 35(12):54–68, Dec. 1997.

[18] Xin Yuan and ZhenHai Duan. FRR: A proportional and worst-case fair round robin scheduler. In *Proc. infocom*, pages 831–842, 2005.