SRR: An *O*(1) Time Complexity Packet Scheduler for Flows in Multi-Service Packet Networks

Guo Chuanxiong
Inst. of Comm. Eng.
P.O.Box 110, 2 Biaoying, Yudao st.
Nanjing, 210016, China
xguo@ieee.org

ABSTRACT
In this paper, we present a novel fair queueing scheme, which we call Smoothed Round Robin (SRR). Ordinary round robin schedulers are well known for their burstiness in the scheduling output. In order to overcome this problem, SRR codes the weights of the flows into binary vectors to form a Weight Matrix, then uses a Weight Spread Sequence (WSS), which is specially designed to distribute the output more evenly, to schedule packets by scanning the Weight Matrix. By using the WSS and the Weight Matrix, SRR can emulate the Generalized Processor Sharing (GPS) well. It possesses better short-term fairness and schedule delay properties in comparison with various round robin schedulers. At the same time, it preserves *O*(1) time complexity by avoiding the time-stamp maintenance employed in various Fair Queueing schedulers. Simulation and implementation experiments show that SRR can provide good average end-to-end delay for soft real-time services. SRR can also be implemented in high-speed networks to provide QoS for its simplicity and low time complexity.

Keywords
QoS, packet scheduler, fair queueing, time complexity, end-to-end delay, high-speed networks.

1. INTRODUCTION
With the expanding of the Internet, more and more services besides the traditional Best Effort services are added into the network. Video and audio conferencing, remote medical caring are some of the examples. It is expected that more services to be introduced in the near future. Different types of services have different characteristics, and generally have different requirements. For example, video conferencing is a kind of VBR service that requires broad bandwidth and low end-to-end delay bound, while traditional data services do not have explicit QoS requirements. Even with the rapid increasing rate of the transmission medium, certain kind of isolation is needed to satisfy the QoS requirements of the competing *flows* (as defined in [29] , a flow is a stream of packets that traverse the same route from the source to the destination, and that require the same grade of transmission service. Flows can be further aggregated into *classes*). Many mechanisms on how to provide QoS support for packet networks have been proposed in [3], [8], [13], [19], [29], [30]. One of the most important parts of these mechanisms is a packet scheduler. A packet scheduler's task is to decide which packet to be transmitted when the output link is idle. Traditional routers use First Come First Serve (FCFS) scheduler to schedule packets. FCFS does not distinguish different flows. Thus, it does not provide any kind of isolation among them. We only consider schedulers that distinguish different flows or classes in this paper.

Generally, packet scheduler should have the following properties:

1. low time complexity to choose and forward a packet;

2. treats different flows fairly;

3. provides low worst case delay and delay variation;

4. it should be simple enough to be implemented efficiently.

The simplicity and time complexity properties always collide with the fairness and delay bound properties. Schedulers with short-term fairness and strict delay bound generally have high time complexity and are hard to be implemented. *O*(1) time complexity schemes are easy to be implemented, but they generally fail to provide short-term fairness and low local delay bound.

In time-stamp based schedulers (one of the two kinds of well studied work-conserving scheduling algorithms), a virtual time clock is maintained to emulate the ideal Generalized Processor Sharing (GPS [19]). Traditional Weighted Fair Queuing (WFQ) [11](PGPS [19]) has low local delay bound and good fairness, but its time complexity is *O*(*N*) (*N* is the number of the active flows). Variants of WFQ such as Virtual-Clock [29], WF^2Q [1], Start-time FQ [16], FFQ, SPFQ [25], Time-shift FQ [9] use different methods to calculate the time-stamp, but still have at least *O*(*logN*) time complexity. Since the best known algorithm to insert a number into a sorted array needs *O*(*logN*), it is unlikely that a time-stamp based scheduler with *O*(1) time complexity can be found. However, an *O*(*logN*) scheduler is not good enough for a high-speed link. For example, it takes approximate 0.08us to transmit a 100 bytes length packet for a 10G*bps* link. That means an *O*(*logN*) scheduler must finish the packet selection in 0.08us regardless of the number of flows. The situation becomes even worse when the capacity of the output link is 40G*bps* or higher.

This work was supported by the 863 project of China under contract number 863-300-02-04-99

SIGCOMM'01, August 27-31, 2001, San Diego, California, USA.

On the other hand, another kind of work-conserving schedulers, the round robin schemes are simple to be implemented and have $O(1)$ time complexity, but they are well known for their output burstiness and short-term unfairness. Deficit Round Robin (DRR) [23] and Carry-Over Round Robin (CORR) [22] are typical round robin schedulers. In these kinds of round robin schedulers, the schedulers will serve a flow for a continuous period of time in proportion to the weight of the flow, resulting in a highly burst scheduling output for each flow. Thus, these kinds of round robin schedulers are considered not suitable to provide QoS in packet networks.

In this paper, we present a Smoothed Round Robin (SRR) scheduler to overcome the shortcomings of the ordinary round robin schedulers. SRR has short-term fairness and certain schedule delay bound, as well as $O(1)$ time complexity. A Weight Spread Sequence (WSS) and a Weight Matrix are used as two key data structures of the scheduler. The weights of the flows are coded into binary vectors to form a Weight Matrix, then SRR uses the corresponding WSS to scan the Weight Matrix. WSS is a specially designed sequence that can distribute the output traffic of each flow evenly. Thus, SRR can emulate GPS as the various time-stamp based schedulers do. Since it does not need to maintain any tags or states, SRR can achieve $O(1)$ time complexity, short-term fairness, and certain delay bound at the same time.

In the following paragraphs, an example is illustrated to show how SRR works. The concepts of WSS, Weight Matrix are used without definitions. The formal definitions will be given in Section 2.

Suppose there are four flows with fixed packet size, named f_1, f_2, f_3, f_4, with rates $r_1 = 64kbps$, $r_2 = 256kbps$, $r_3 = 512kbps$, $r_4 = 192kbps$. The packet sizes of the flows are 512 bytes; all the four flows are backlogged. And the bandwidth of the output link $C = 2^{20} = 1Mbps$. The corresponding weights of the flows are $w_1 = 1$, $w_2 = 4$, $w_3 = 8$, $w_4 = 3$. By coding the weights into binary vectors, we have $WV_1 = \{0,0,0,1\}$, $WV_2 = \{0,1,0,0\}$, $WV_3 = \{1,0,0,0\}$, $WV_4 = \{0,0,1,1\}$ for the four flows respectively. According to the binary vectors, the Weight Matrix corresponding to flow f_1, f_2, f_3, f_4 is,

$$WM = \begin{bmatrix} WV_1 \\ WV_2 \\ WV_3 \\ WV_4 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}.$$

We number the columns of this WM from left to right as $column_3, column_2, column_1$ *and* $column_0$.

The corresponding WSS (which will be defined in Section 2.1) to this Weight Matrix is,

$$1,2,1,3,1,2,1,4,1,2,1,3,1,2,1.$$

SRR then scans the WSS term by term, when the value of the term is i, the $column_{4-i}$ is chosen. In $column_{4-i}$, the scheduler will scan the terms from top to bottom, when the term is not 0, the scheduler will serve the corresponding flow. That is, the flows will be served in the following service sequence,

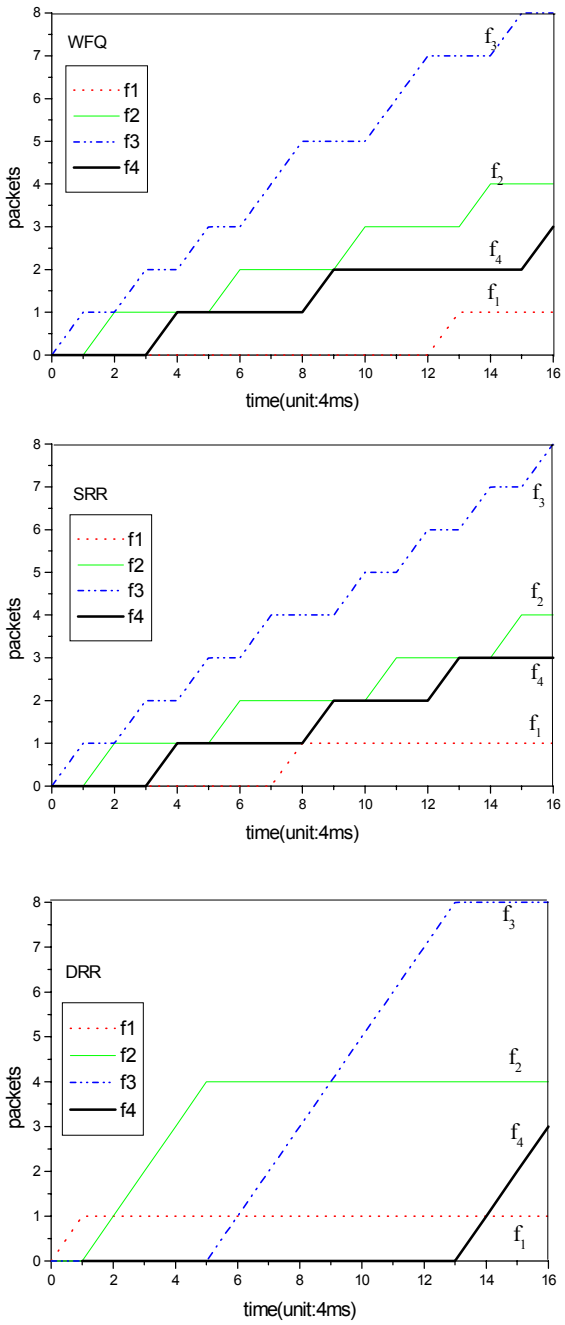

Figure 1. Service curves of the three schedulers

$f_3, f_2, f_3, f_4, f_3, f_2, f_3, f_1, f_4, f_3, f_2, f_3, f_4, f_3, f_2, f_3.$

The service curves of the flows in SRR are shown in Figure 1 together with that in WFQ and DRR. From the above example, we can observe that SRR emulates the GPS quite well. Both WFQ and SRR perform much better than DRR in that they have much better delay bound and short-term fairness. It is also easy to observe that the outputs of f_2, f_3, f_4 in DRR are very bursty.

In Section 2, the definition of a set of Weight Spread Sequences (WSS) and their properties are presented first, then the definition of the Weight Matrix is given. In Section 3, the formal description of SRR is described. The fairness, schedule delay bound, scalability, space and time complexity of SRR are analyzed in Section 4. In section 5, simulation experiments are designed to compare the end-to-end delay property of SRR with that of WFQ and DRR, simulation experiments show that SRR can provide good average end-to-end delay for soft real-time services such as IP telephony. This paper concludes with Section 6.

2. THE WEIGHT SPREAD SEQUENCE AND THE WEIGHT MATRIX

2.1 The Weight Spread Sequence

Definition 1: A set of Weight Spread Sequences (WSS) is defined recursively as,

1) The first WSS $S^1 = 1$,

2) The *k*th WSS is, $S^k = \{a_i\} = S^{k-1}, k, S^{k-1}$.
$1 \le i \le 2^k - 1$ for $k > 1$ (1)

The set corresponding to sequence S^k is $\{1,2,3,\ldots,k\}$. len_k is defined as the total number of terms of the *k*th WSS. The terms of the *k*th WSS can be arranged in a circle so that term a_{2^k-1} is next to term a_1.

The *distance* between two terms a_m and a_n of S^k is defined to be,

$$\min[(n-m) \bmod (2^k - 1), (m-n) \bmod (2^k - 1)].$$

Two terms a_m, a_n ($m>n$) are called two adjacent occurrences of element $i (1 \le i < k)$ if,

1) $a_m = a_n = i$, and,

2) $a_j \ne i$ for $j \in (n,m)$ or $a_j \ne i$ for $j \in (m, 2^k - 1] \cup [1, n)$. For the former case, the *chain* between two adjacent occurrences a_m, a_n for element i is $\{a_{n+1}, a_{n+2}, \ldots, a_{m-1}\}$, for the latter case, the *chain* is $\{a_{m+1}, a_{m+2}, \ldots, a_{2^k-1}, a_1, a_2, \ldots, a_{n-1}\}$.

For example, from Definition 1, the 5th WSS is

$$1,2,1,3,1,2,1,4,1,2,1,3,1,2,1,5,1,2,1,3,1,2,1,4,1,2,1,3,1,2,1.$$

So, for this sequence, $len_5 = 31$, the *distance* between two adjacent occurrences of element 3 is 7 or 8.

The following Propositions are properties of WSS.

Proposition 1: The first $(2^{k-1} - 1)$ terms of a *k*th WSS forms a (*k*-1)th WSS. And a (2*k*)th WSS can be constructed from a *k*th WSS and a (*k*+1)th WSS in the following way,

1. adding (*k*-1) to every term of the (*k*+1)th WSS;
2. replacing the terms a_i where $a_i = k$ of the newly formed (*k*+1)th WSS with the *k*th WSS. The newly formed sequence is a (2*k*)th WSS.

Proposition 2: The total number of terms of S^k is $2^k - 1$ (i.e., $len_k = 2^k - 1$), and the number of the occurrences of element $i (1 \le i \le k)$ is 2^{k-i}.

Proposition 3: The *chain* between two adjacent occurrences $a_m, a_n (m > n)$ of element i in S^k is,

$$\begin{cases} S^{i-1}, S^{i-1} & if \quad m-n > (n+2^k-1)-m; \\ S^{i-1}, x, S^{i-1} & if \quad m-n < (n+2^k-1)-m. \end{cases} \quad (2)$$

where $1 < i < k$, and $i < x \le k$. The proof of Proposition 3 is given in the Appendix A.

The following proposition is obvious in view of Proposition 3.

Proposition 4: The *distance* between two adjacent occurrences of element $i (1 \le i < k)$ in S^k is either 2^i or $2^i - 1$.

Since there is only one occurrence of element *k* in S^k, the distance of element *k* is defined to be $2^k - 1$.

2.2 The Weight Matrix

In various Fair Queueing Schedulers, each flow is assigned a weight in proportion to its reserved rate. In this paper, we assume that the set of weights is $\{1,2,3,4,5\ldots,2^k - 1\}$. By adjusting the value of *k*, rate allocation schemes with different range can be accommodated. For example, for *k*=16, if the granularity of rate is 1*bps*, then the set of rates corresponding to the set of weights is $\{1bps, 2bps, 3bps, \cdots, 64kbps\}$. For *k*=32, the set of rates is $\{1bps, 2bps, 3bps, \cdots, 4Gbps\}$.

The weight of $flow_f$ can be coded as,

$$w_f = \sum_{n=0}^{k-1} a_{f,n} 2^n \quad \text{where } a_{f,n} \in \{0,1\}.$$

Definition 2: The binary coefficients $a_{f,n}$ of w_f form a Weight Vector of $flow_f$, which is defined as,

$$WV_f = \{a_{f,(k-1)}, a_{f,(k-2)}, \ldots, a_{f,0}\}. \quad (3)$$

Definition 3: The Weight Matrix corresponding to flows $f_1, f_2, \ldots, f_N$ is defined as

$$WM = \begin{bmatrix} WV_1 \\ WV_2 \\ \vdots \\ WV_N \end{bmatrix} = \begin{bmatrix} a_{1,(k-1)} & a_{1,(k-2)} & \cdots & a_{1,0} \\ a_{2,(k-1)} & a_{2,(k-2)} & \cdots & a_{2,0} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N,(k-1)} & a_{N,(k-2)} & \cdots & a_{N,0} \end{bmatrix}. \quad (4)$$

where $a_{i,j} \in \{0,1\}$, and $1 \le i \le N$, $0 \le j \le (k-1)$. We number the column of the Weight Matrix from left to right as $column_{k-1}, column_{k-2}, \ldots, column_0$.

3. THE SMOOTHED ROUND ROBIN SCHEDULER

We combine the *k*th WSS and the $N \times k$ Weight Matrix to form the Smoothed Round Robin (SRR) scheduler. The basic idea of SRR is based on the scanning of the WSS and the corresponding Weight Matrix. The WSS is scanned term by term. When the current term is element i, $column_{k-i}$ of the Weight Matrix is selected. For each occurrence of $a_{f,(k-i)} = 1$ in this column, packet from flow corresponding to the row of $a_{f,(k-i)}$ is scheduled.

In the following of this section, the formal description of SRR is given.

3.1 Formal Description of SRR

In this paper, we focus on packet scheduler, and consider the packet classification the function of the *packet classifier*, the assignment of weights to flows the function of the *admission controller*, and that all the input packets are queued to their corresponding queues by the *packet enqueuer*. The tasks of the packet scheduler are to choose and forward packets and to maintain related data structures of the scheduler.

In a packet network, if the packet length of a flow is greater than the Maximum Transmission Unit (MTU) of the output link, the system will fragmentize the packet into small pieces. Therefore, we assume that the maximum packet length of all the flows is the MTU of the output link, and denote it as $L_{\max}$.

In SRR, we assume that the maximum order of WSS is $K_{\max}$. When $K_{\max}$ =32, if the bandwidth assignment granularity is 1bps, the set of rates that can be provided by SRR is $\{1bps, 2bps, 3bps, \cdots, 4Gbps\}$; if the granularity is 1kbps, the set of rates is $\{1kbps, 2kbps, 3kbps, \cdots, 4Tbps\}$.

We assume that a flow can be deleted explicitly by a command (i.e., by some kinds of signaling protocols) or implicitly by SRR when the queue corresponding to that flow is empty.

We adopt the following notations for the scheduler:

$K_{\max}$	The maximum order of the WSS used by SRR;
M	Weight Matrix of all the active flows;
S^k	The *k*th WSS currently used by the scheduler;
k	The order of the current WSS used by SRR;
P_c	Index of the current scanning position of the WSS, ranging from 1 to $2^k - 1$;
$queue_f$	Queue of the received packets of $flow_f$, which is a FIFO;
P_f	Packet that is at the head of $queue_f$;
L_f	Length of P_f;
w_f	Weight of $flow_f$, it is a normalized value according to the bandwidth assignment granularity;
$deficit_f$	A register to memorize how many bytes $flow_f$ should bring to the next round;
DL_i	The *i*th double link, $0 \le i < K_{\max}$. There are $K_{\max}$ double links in SRR;
P_{dl}	Pointer to a node of a double link;
$L_{\max}$	The upper bound of packet's length of the output link;
C	Normalized (according to the bandwidth assignment granularity) bandwidth of the output link.

We use the following 3 pieces of pseudo C code in Figure 2 to illustrate the scheduler. There are 3 asynchronous *actions*, namely, **Schedule**, **Add_flow**, **Del_flow**. Each action is triggered by some events. **Schedule** is the main part of the scheduler, it is invoked whenever the output link enters a *busy-period*. **Add_flow** is invoked when a new flow arrives. **Del_flow** is the action taking place when the flow is deleted explicitly or dead (i.e., the queue of the corresponding flow is empty).

```
Schedule{
local variable: f, col; /*f, col are the current row, column
number of M, respectively*/
P_c=1; P_dl=head_{k-1}->next; /*initialization*/

while(in busy-period){
    f=P_dl ->fid;
    deficit_f = deficit_f + L_max;
    while(deficit_f > 0){
        if(L_f <= deficit_f ){
            dequeue(P_f);
            send(P_f);
            deficit_f = deficit_f - L_f;
            if(queue_f is empty){
                Del_flow(f);
                break;
            }
        }elsebreak;
    }
    if(P_dl->next!=tail_col){
        P_dl=P_dl->next;
    }else{
```

```
loop:  Pc=Pc+1;
       if(Pc==2^k) Pc=1;
       col=k-S^k[Pc];/*get the corresponding column number */
       if(DL_col is empty) goto loop;
       P_dl=head_col->next; /* points to the first non-zero term of
this column*/
       }
    }
}
```

Add_flow(*w*) { /* w is the weight assigned to this flow */

local variable: f;

*f=get_flowId( ) ; /*get a new flow ID for this flow */*

assign deficit$_f$, queue$_f$; / deficit$_f$=0, queue$_f$ is empty */*

use w to form a Weight Vector as shown in Equation 3;

add the vector to the last row of Matrix M;

insert nodes into DL_0, DL_1, …, DL_{Kmax-1} according to the coefficients of w;

if(new columns are added into M)

update k;

}

Del_flow(f) { /* f is the flow ID of this flow */

remove the corresponding row from M;

remove deficit$_f$, queue$_f$;

remove nodes from DL_0, DL_1, …, DL_{Kmax-1} according to the coefficients of w_f ;

if(empty columns are deleted from M){

update k;

$P_c = P_c \bmod (2^k)$;

}

}

Figure 2. Description of SRR

The Weight Matrix is adjusted dynamically in SRR. When a new flow comes, a new row will be added into *M* as the last row. If the weight of this new flow $w_f = \sum_{n=0}^{j-1} a_{f,(j-1)} 2^n$ $(j > k, a_{f,(j-1)} = 1)$, new columns numbered $column_{j-1}, clolumn_{j-2}, \ldots, column_k$ will be added into *M*, and the order of the WSS is adjusted to *j* (i.e., *k*=*j*). When a flow leaves SRR, the corresponding row of *M* will be deleted. If the $column_{k-1}, clolumn_{k-2}, \ldots, column_i$ become empty, these columns will be removed from *M*, then the order of the WSS is adjusted to *i* (i.e., *k*=*i*).

$S^k (1 \le k \le K_{\max})$ is the *k*th WSS defined by equation (1), the order of the WSS used in SRR is adjusted dynamically according to the column number of *M*.

$deficit_f$ is borrowed from DRR [23] to memorize the bytes that $flow_f$ can bring into the next round.

There are $K_{\max}$ double links named DL_0, $DL_1, \ldots, DL_{K_{\max}-1}$ in SRR. DL_i is used to link the non-zero terms of $column_i$ of *M*. Each DL_i has a head, and a tail. Each node of the link has three fields, *next*, *prev* and *fid*. *next* is a pointer points to the next node, *prev* is a pointer points to the previous node, and *fid* is a field contains the flow id. DL_i is defined to be empty if all the terms of $column_i$ are zero. These links are used here to reduce the time complexity of SRR. Double link data structure is chosen to reduce the time complexity of flow deletion.

The *busy-period* in **Schedule** has the same meaning as that in [19]. At the beginning of SRR, *M* and the double links are empty. When the first flow comes, **Add_flow** will be called. Then **Schedule** will be invoked. After **Del_flow** deletes the last flow, the system enters idle state waiting for the next busy-period. Since **Add_flow** and **Del_flow** need to update at most *k* double links, their time complexities are in proportion to *k*, which is the current order of WSS used by SRR.

Since *M* is adjusted dynamically according to the weights of flows in SRR, SRR has the following property.

Proposition 5: The double link DL_{k-1} is not empty in SRR, where *k* is the order of the WSS currently in use in SRR.

In the following section, properties such as the long-term and short-term fairness, schedule delay bound, scalability, time and space complexity of SRR will be analyzed.

4. PROPERTIES OF SRR

Since SRR always forwards packets when there are active flows in the system, it is work-conserving.

SRR finishes a *round* when it starts from the first term of the *k*th WSS, and after visiting all the $2^k - 1$ terms, back to the beginning of the sequence again.

Theorem 1: SRR visits $flow_f$ w_f times in a round, where w_f is the weight of $flow_f$.

Proof: From Proposition 2 and the description of SRR, $column_i$ (and therefore all the terms belong to $column_i$) of the Weight Matrix *M* will be visited 2^i times in a round. Since $w_f = \sum_{n=0}^{k-1} a_{f,n} 2^n$, where $a_{f,n}$ belongs to $column_n$, $flow_f$ will be visited w_f times in a round. □

Thus each flow gets its share in a round according to its weight.

The following lemma is obvious according to the working procedure of SRR.

Lemma 1: Suppose $flow_f$ is backlogged, and has been visited by SRR *x* times from time 0 to *t*, and $S_f(0,t)$ denotes the bytes served by SRR of $flow_f$, then,

$$(x-1)L_{\max} < S_f(0,t) \le xL_{\max} \quad (5)$$

4.1 Fairness of the Scheduler

Let $V_f(0,t)$ the times $flow_f$ visited by SRR from time 0 to *t*, and τ the time the scheduler finishes a round. From Theorem 1, it is easy to see that at the end of a round, for any pair of active flows *f, g*, the following equation holds,

Lemma 2:

$$\left|\frac{V_f(0,\tau)}{w_f} - \frac{V_g(0,\tau)}{w_g}\right| = 0 \quad (6)$$

Lemma 2 shows the long-term fairness of SRR. However, SRR can provide more than this. For any pair of active flows *f, g*, we have the following theorem.

Theorem 2: For any pair of backlogged flows *f, g* in SRR,

$$\left|w_f V_g(0,t) - w_g V_f(0,t)\right| \le \frac{k}{2}\max(w_f, w_g) \quad (7)$$

k is the order of the current WSS used by SRR. The proof of Theorem 2 is given in the Appendix B.

From Theorem 2, there exists following corollary.

Corollary 1: For any pair of backlogged flows *f, g* in SRR,

$$\left|\frac{S_f(0,t)}{w_f} - \frac{S_g(0,t)}{w_g}\right| < \frac{(k+2)L_{\max}}{2\min(w_f, w_g)} \quad (8)$$

where $S_f(0,t)$, $S_g(0,t)$ denote the service received by $flow_f$, $flow_g$ from time 0 to *t*, respectively.

Proof: From Lemma 1, the following 2 inequalities hold,

$$(V_f(0,t)-1)L_{\max} < S_f(0,t) \le V_f(0,t)L_{\max}$$
$$(V_g(0,t)-1)L_{\max} < S_g(0,t) \le V_g(0,t)L_{\max}$$

Thus,

$$\frac{S_g(0,t)}{w_g} - \frac{S_f(0,t)}{w_f} < \frac{V_g(0,t)L_{\max}}{w_g} - \frac{(V_f(0,t)-1)L_{\max}}{w_f} = \left(\frac{V_g(0,t)}{w_g} - \frac{V_f(0,t)}{w_f}\right)L_{\max} + \frac{L_{\max}}{w_f}$$
$$\frac{S_f(0,t)}{w_f} - \frac{S_g(0,t)}{w_g} < \frac{V_f(0,t)L_{\max}}{w_f} - \frac{(V_g(0,t)-1)L_{\max}}{w_g} = \left(\frac{V_f(0,t)}{w_f} - \frac{V_g(0,t)}{w_g}\right)L_{\max} + \frac{L_{\max}}{w_g}$$

so,

$$\left|\frac{S_f(0,t)}{w_f} - \frac{S_g(0,t)}{w_g}\right| < \left|\frac{V_f(0,t)}{w_f} - \frac{V_g(0,t)}{w_g}\right|L_{\max} + \frac{L_{\max}}{\min(w_f, w_g)}$$
$$\le \frac{kL_{\max}}{2\min(w_f, w_g)} + \frac{L_{\max}}{\min(w_f, w_g)} = \frac{(k+2)L_{\max}}{2\min(w_f, w_g)}.$$

□

4.2 Schedule Delay Bound of SRR

If T_a^p is the time a packet becomes the head of $queue_f$, and T_d^p is the time that the scheduler finishes transmitting the packet, we name the schedule delay for this packet, $D_f^p = T_d^p - T_a^p$. We further name the maximum value of D_f^p the scheduler delay bound of $flow_f$, that is,

$$D_f = \max(D_f^p), \quad where \quad p \in flow_f.$$

As to D_f, we have the following theorem.

Theorem 3: Suppose there are N flows, numbered from 1 to N in SRR. The weight assigned to $flow_f$ is w_f, and $\sum_{f=1}^{N} w_f \le C$, $w_f = \sum_{n=0}^{i} a_{f,n} 2^n$, where $a_{f,i} = 1$, and $i \le k-1$. The schedule delay bound of $flow_f$,

$$D_f < \frac{2L_{\max}}{w_f} + (N-1)\frac{2L_{\max}}{C} \quad (9)$$

Proof: According to SRR, a packet becomes the head of a flow if it is the head of a new flow or the packets before it have left the system. When a packet becomes the head of a flow, it will be served when SRR visits the flow again. A flow is visited when one of its coefficients $a_{f,n}(a_{f,n} \ne 0)$ is visited by SRR. So the delay bound of a flow is the maximum value of the intervals between two adjacent visits by SRR. Let $count$ be the sum of times that each non-zero terms of *M* is visited by SRR during this interval. According to the value of w_f, there are two cases.

1. $2^i \le w_f < 2^{i+1} - 1$.

From Proposition 3, there must exists a y, where $y < i$, and $a_{f,y} = 0$. The chain between two terms of element ($k-i$) that includes element ($k-y$) is $S^{k-i-1},(k-y),S^{k-i-1}$. In this case, $flow_f$ will be visited again after SRR visits the columns mapped by $S^{k-i-1},(k-y),S^{k-i-1}$ and the $column_i$. Thus,

$$count = 2\sum_{m=1}^{N} a_{m,i+1} + 2^2 \sum_{m=1}^{N} a_{m,i+2} + \dots$$
$$+ 2^{k-i-1} \sum_{m=1}^{N} a_{m,k-1} + \sum_{m=1}^{N} a_{m,y} + \sum_{m=1}^{N} a_{m,i}$$
$$= \sum_{n=i}^{k-1} (2^{n-i} \sum_{m=1}^{N} a_{m,n}) + \sum_{m=1}^{N} a_{m,y}$$
$$= \frac{1}{2^i} \sum_{n=i}^{k-1} (2^n \sum_{m=1}^{N} a_{m,n}) + \sum_{m=1}^{N} a_{m,y}$$
$$= \frac{1}{2^i} (\sum_{n=0}^{k-1} 2^n \sum_{m=1}^{N} a_{m,n} - \sum_{n=0}^{i-1} 2^n \sum_{m=1}^{N} a_{m,n}) + \sum_{m=1}^{N} a_{m,y}$$
$$\le \frac{1}{2^i} (C - \sum_{n=0}^{i-1} 2^n \sum_{m=1}^{N} a_{m,n}) + \sum_{m=1}^{N} a_{m,y}.$$

Thus,

$$D_f^P < \frac{L_{\max}}{C} [\frac{1}{2^i} (C - \sum_{n=0}^{i-1} 2^n \sum_{m=1}^{N} a_{m,n}) + \sum_{m=1}^{N} a_{m,y}] + \frac{(N-1)L_{\max}}{C}$$
$$< \frac{2L_{\max}}{w_f} - \frac{L_{\max}}{C} (\frac{1}{2^i} \sum_{n=0}^{i-1} 2^n \sum_{m=1}^{N} a_{m,n} - \sum_{m=1}^{N} a_{m,y} - (N-1))$$

The $(N-1)L_{\max}$ is the maximum deficit the other $(N-1)$ flows can bring into this interval.

2. $w_f = 2^{i+1} - 1$.

In this case, the chain with the maximum length between two adjacent occurrences of element $(k-i)$ is S^{k-i-1}, S^{k-i-1}. So,

$$count = 2\sum_{m=1}^{N} a_{m,i+1} + 2^2 \sum_{m=1}^{N} a_{m,i+2} + \dots$$
$$+ 2^{k-i-1} \sum_{m=1}^{N} a_{m,k-1} + \sum_{m=1}^{N} a_{m,i}$$
$$= \sum_{n=i}^{k-1} (2^{n-i} \sum_{m=1}^{N} a_{m,n}) = \frac{1}{2^i} \sum_{n=i}^{k-1} (2^n \sum_{m=1}^{N} a_{m,n})$$
$$= \frac{1}{2^i} (\sum_{n=0}^{k-1} 2^n \sum_{m=1}^{N} a_{m,n} - \sum_{n=0}^{i-1} 2^n \sum_{m=1}^{N} a_{m,n})$$
$$\le \frac{1}{2^i} (C - \sum_{n=0}^{i-1} 2^n \sum_{m=1}^{N} a_{m,n}).$$

Thus,

$$D_f^P < \frac{2L_{\max}}{w_f} - \frac{L_{\max}}{2^i C} \sum_{n=0}^{i-1} 2^n \sum_{m=1}^{N} a_{m,n} + \frac{(N-1)L_{\max}}{C}$$

In the above 2 cases,

$$D_f = \max(D_f^P) < \frac{2L_{\max}}{w_f} + \frac{2L_{\max}(N-1)}{C}. \qquad \square$$

So D_f is not only in inverse proportion to the weight of the flow, but also in direct proportion to the total number of the flows in SRR. Thus it fails to provide a strictly rate-proportional delay bound. However, the delay bound is still much better than that of DRR.

However, we have $D_f < \frac{2L_{\max}}{w_f}$ if $\frac{1}{2^i} \sum_{n=0}^{i-1} 2^n \sum_{m=1}^{N} a_{m,n} - \sum_{m=1}^{N} a_{m,y} = 0$ and no flows bring deficit to the next round. Thus, on average case, the delay bound is only in inverse proportion to the weight of the flow.

It should be noted that D_f is different from the local delay bound concept used in WFQ and its variants, where the departure time of a packet is compared with the departure time under GPS [19]. D_f is similar to the concept of WFI of [1,2]. It has been shown in [1] that D_f (or WFI) of WFQ is in proportion to N, where N is the number of active flows. Thus D_f (or WFI) of SRR is similar to that of WFQ .

It also should be noted that SRR fails to provide the inequality $F' - F \le \frac{L_{\max}}{C}$ as WFQ does[1]. For example, suppose the packet length is 1, and *C=16*, there are 8 flows numbered as $f_1, f_2, \cdots, f_8$ with weight 1. When SRR is serving f_1, a new flow f_9 with weight 8 comes. In this case, for the first packet of f_9 in SRR, $F' - F = \frac{N-1}{C} + \frac{1}{C} - \frac{1}{w_9} = \frac{7}{16} > \frac{L_{\max}}{C} = \frac{1}{16}$.

4.3 Scalability of SRR

The scalability of SRR is illustrated in the following aspects.

1. Different rate ranges can be accommodated with the WSS of the same order by adjusting the rate granularity. For example, when the granularity of rate is 1K*bps*, and $K_{\max}$ *=16* (i.e., the order of the WSS is 16), the set of rates is $\{1kbps, 2kbps, 3kbps, \cdots, 64Mbps\}$. When the rate granularity is 1M*bps*, the corresponding set of rate is $\{1Mbps, 2Mbps, 3Mbps, \cdots, 64Gbps\}$. Thus, similar WSS can be used in both core routers (switches) and edge routers (switches).
2. SRR can be used in output links with variable bandwidth capacity. According to its working procedure, SRR can provide fairness among competing flows even when the bandwidth of the output link varies from time to time.
3. SRR works well regardless of the number of flows. Since the time complexity of SRR is strictly *O*(1) (which will be proven in the next subsection), SRR works well even with a large number of flows. This makes SRR an attractive scheduler for high-speed networks where time complexity is the most important factor.

[1] The inequality is Theorem 1.1 of reference [17] (in page 25).

4.4 Complexity of SRR

From the first part of Proposition 1, we know that the WSSs with order $\{1,2,3,\cdots,K_{\max}-1\}$ are contained in the $K_{\max}th$ WSS. Thus, only one $K_{\max}th$ WSS is needed in SRR. When $K_{\max}$ =16, the space needed to store the corresponding WSS is 64k bytes (each term of WSS occupies one byte). However, since the length of the WSS increases exponentially with the order of the WSS, it becomes impractical to store the whole sequence statically when $K_{\max}$ becomes very large. This problem can be overcome by using the last part of Proposition 1. By constructing a (2*k*)*th* WSS from a *kth* and a (*k*+1)*th* WSS, the space needed can be reduced from 2^{2k} to 3×2^{k}.

We believe that a 32*th* WSS is enough for current and future packet networks (it can provide up to 4T*bps* rate with granularity of 1k*bps*). Thus, under this condition, the space complexity of SRR is $c+O(N\times K_{\max})$, $K_{\max}\le 32$ and $c=3\times 2^{16}$. *c* is the space needed to store a 16*th* and a 17*th* WSS, $O(N\times K_{\max})$ is the space needed to store the $K_{\max}$ double links.

We have the following theorem for the time complexity of SRR.

Theorem 4: The SRR packet Scheduler needs *O*(1) time to choose a packet for transmission, *O*(*k*) time to add or delete a flow, where *k* is the order of WSS currently used by SRR.

Proof: SRR uses the **schedule** action in Figure 2 to choose a packet for transmission. It takes the scheduler *O*(1) time to choose the flow *f*. Then since $L_{\max}\ge L_f$, schedule will transmit at least one packet for flow *f*. After serving *f*, the schedule will update P_{dl}. If the end of DL_{col} is not reached, one sentence is needed to update P_{dl}. If the end of DL_{col} is met, schedule will update P_c to get the new column number of *M*, it may enter the *loop* code. However, according to the WSS, $column_{k-1}$ of *M* will be visited at least once in every 2 times. According to Proposition 5, DL_{k-1} is not empty. Thus, the *loop* code can be executed at most 2 times. Thus, the code that updates P_{dl} and P_c needs *O*(1) time. Thus, SRR needs *O*(1) time to choose a packet for transmission.

Since **Add_flow** and **Del_flow** need to update the *k* double links when flows come and leave SRR, their time complexities are *O*(*k*) □

It should be noted that if a flow is always not backlogged, the **Add_flow** and **Del_flow** will be invoked once per packet. Though in SRR (which uses a fixed number of weights, $2^{K_{\max}}$) the time complexities of **Add_flow** and **Del_flow** are constant values (at most $O(K_{\max})$), it does introduce a burden that may be comparable to the *O*(*logN*) incremental step in the time-stamp based schemes.

In [5], we propose to use a timer to delay the deletion of an inactive flow. However, such a mechanism will make SRR not a strictly *O*(1) scheme to forward a packet. We also show in [5] that it is difficult to choose the time-out value of the timer. It is a question needs further investigation.

5. SIMULATION

In this paper, we use simulation to compare the end-to-end delay property of SRR with that of WFQ and DRR. For more simulation experiments (such as local delay bound, queue delay and fairness) please refer to [5].

5.1 Simulation Configuration

The tool we used in our simulation experiment is *ns* [31], to which we added WFQ, SRR scheduling classes, and revised the

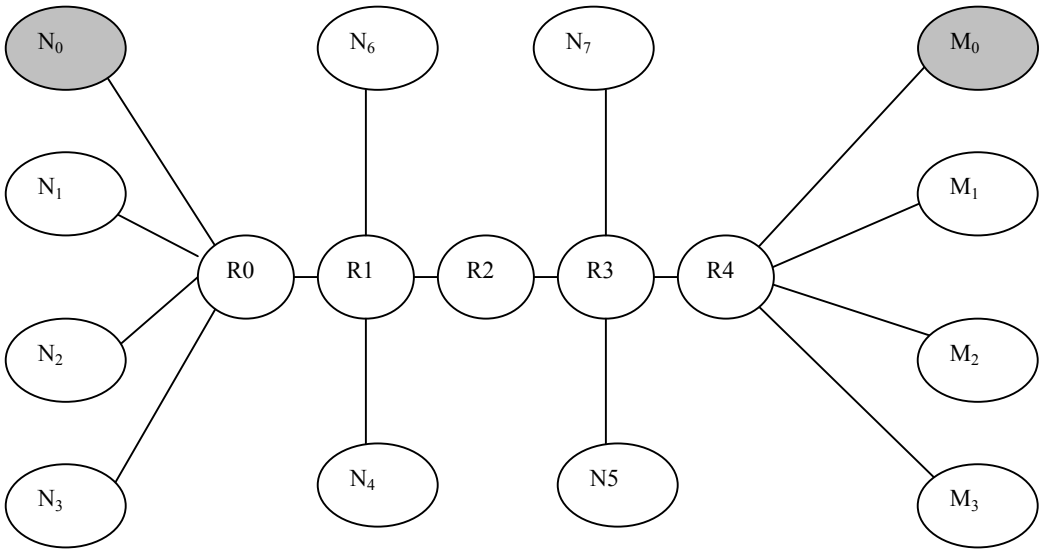

Figure 3. Network topology of the simulation experiment

DRR[2] scheduling class.

As shown in Figure 3, the above network topology is designed to compare the end-to-end delay property of SRR with that of WFQ and DRR. There are 12 hosts (N0-N3, M0-M3, and N4-N7), and 5 routers (R0-R4). The transmission delay and bandwidth capacities of the links are shown in the following table.

Table 1 Transmission delay and bandwidth parameters of the links.

Links	Transmission delay (ms)	Bandwidth (Mbps)
N[0-3] – R0	0.03	10
R0-R1	0.1	6
R1-R2	3	15.5
R2-R3	3	100
R3-R4	0.1	10
R4-M[0-3]	0.03	10
N[4,6]-R1	0.03	10
N[5,7]-R3	0.03	10

In this simulation, R0, R4 are edge routers, R1-R3 are core routers. A packet from N0 to M0 will traverse 6 links.

The following traffic traces are used in this experiment,

1. There are 10 *CBR* flows numbered from 1 to 10 between N0 and M0. The rates of the 10 flows are 10k*bps*, 10k*bps*, 20k*bps*, 20k*bps*, 40k*bps*, 80k*bps*, 80k*bps*, 160k*bps*, 260k*bps*, 320k*bps* respectively. The *CBR* flows simulate the real-time audio service here.
2. There are 2 *ftp* flows between N1 and M1. The total rates of the two flows are 2M*bps*. These 2 flows are best effort streams. The best effort streams in this experiment are mapped to flow 0.

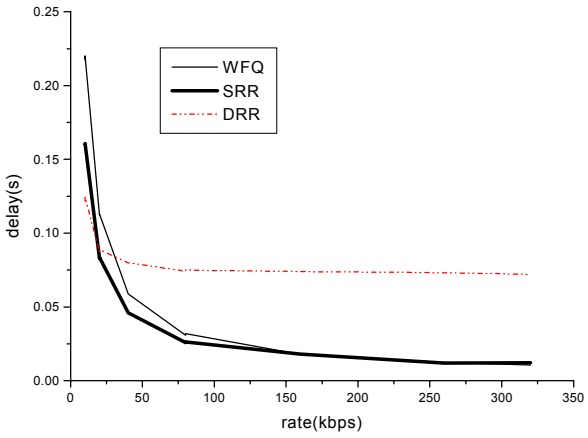

Figure 4a. The average delay of the CBR flows

3. There are 2 real-time video streams numbered as flow 11, 12 between N2 and M2. The total rates of the 2 streams are 1.1Mbps. The video streams are gotten from [21], one is a cartoon movie named *simpsons* (with average rate 464*kbps*), the other is a movie named *golden finger* (with average rate 608*kbps*). The videos were compressed using an MPEG-1 compliant encoder. The quantization values were: I=10, P=14, and B=18 using the pattern *IBBPBBPBBPBB*, which gives a group of picture (GOP) size of 12.
4. There are 10 flows numbered from 13 to 22 with *Pareto* distribution between N3 and M3. The rate of each flow is 200k*bps*. These flows simulate services with long-range dependency.
5. There is a *ftp* stream between N5 and N6, and a *telnet* stream between N4 and N7. These flows are best effort services used to consume the redundant bandwidth of the network.

In this experiment, we measure the end-to-end delays of the ten CBR flows under different scheduling schemes (i.e., WFQ, SRR, DRR).

5.2 Simulation Results

The average and maximum end-to-end delays of flows 1 to 10 are shown in Figure 4a and Figure 4b.

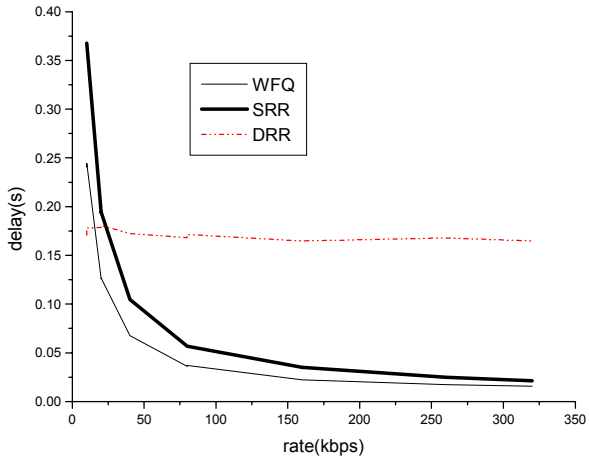

Figure 4b. The maximum delay of the CBR flows

This experiment shows that the end-to-end delay property of SRR is similar to that of WFQ. The worst-case and average end-to-end delays of SRR and WFQ decrease with the increasing of the flow rate. This experiment also shows that the worst-case end-to-end delay property of SRR is worse than that of WFQ (which conforms with Theorem 3 of this paper), and the average delay property of SRR is a little better than that of WFQ. For example, as to the flow 6, the worst-case end-to-end delays under WFQ, SRR, and DRR are 36.16*ms*, 56.61*ms* and 168*ms*, and the average delays are 30.78*ms*, 25.93*ms*, and 70.20*ms* respectively. Thus, as to flow 6,

[2] In ns2.1.b5, the implementation of DRR does not interpret the algorithm in a right way. It deletes a flow when it *deque*s its last packet. However, a flow should be deleted only when the last bit of the last packet left the transmission interface.

$$D_{\max}^{WFQ} : D_{\max}^{SRR} : D_{\max}^{DRR} = 1 : 1.57 : 4.65$$
$$D_{mean}^{WFQ} : D_{mean}^{SRR} : D_{mean}^{DRR} = 1 : 0.84 : 2.28$$

Therefore, the end-to-end delay of SRR is very similar to that of WFQ. Both WFQ and SRR perform much better than DRR. If flow 6 is a real-time IP telephony stream, it will work well under WFQ and SRR. However, it will not work well under DRR for the large worst-case end-to-end delay.

The experiment also shows that the maximum and average end-to-end delays of DRR change little with different flow rates. The average delay is about 80*ms*, and the maximum delay is about 170*ms* for DRR. This experiment shows that the worst-case delay of DRR makes it not suitable for services with certain delay bound requirements, such as IP telephony.

Thus, SRR is a qualified scheduler for services that do not have strict end-to-end delay requirements, such as IP telephony, and adaptive real-time services.

6. CONCLUSION

We have proposed SRR and examined its properties in this paper. A Weight Spread Sequence and a Weight Matrix are introduced as two main data structures of the SRR. With the use of the WSS and the Weight Matrix, the output of SRR is distributed more evenly than that of the ordinary round robin schedulers. SRR can provide strictly $O(1)$ time complexity, short-term fairness, and certain schedule delay bound at the same time.

SRR is attractive for its low time complexity and simplicity. It only needs to store a static WSS, to maintain $K_{\max}$ double links, and to assign a deficit counter for each flow. Thus it can be implemented in high-speed links at low cost, where efficiency and time complexity are the most important factors. It should be noted that SRR fails to provide a strict local delay bound. Thus, it is not suitable for those applications where strict end-to-end delay bound is needed (i.e., guaranteed services). However, simulations show that SRR can provide good average and certain worst-case end-to-end delay bounds, thus it is an appropriate scheduler for services where strict delay bound is not required (such as IP telephony and adaptive real-time services).

Though it is still elusive that whether an ideal packet scheduler with strict rate-proportional delay bound, short-term fairness, and $O(1)$ time complexity exists, this paper introduces a new idea to avoid the $O(\log N)$ limits of various time-stamp based schedulers while still maintaining short-term fairness and certain schedule delay bound.

We have implemented and tested the SRR in the Linux Kernel 2.2.5, the implementation indicates that SRR introduces little cost to the TCP/IP stack, and the experiment results are consonant with that of the simulation results. Our experiments also show that SRR is a suitable scheduling algorithm for the AF PHB of DiffServ.

7. ACKNOWLEDGMENTS

Prof. W. Qi gave the name of SRR and WSS. The author would like to thank him for his constructive comments and generous help for improving the organization and presentation of this paper. The author also would like to thank J. Wang, Y. Sun for implementing the SRR in the Linux kernel, J. Chen, and C. Lin for their valuable comments. Finally, we thank both the anonymous reviewers and Roch Guerin for their efforts.

8. REFERENCES

[1] J. Bennet, and H. Zhang, "WF^{2}Q: worst case fair weighted fair queueing," in *Proc. Infocom'96*, 1996.

[2] J. Bennett, and H. Zhang, "Hierarchical Packet Fair Queueing Algorithms," in *Proc. SIGCOMM'9*6, 1996.

[3] S. Blake, *et. al. "*An Architecture for Differentiated Services," *RFC 247*5, Dec. 1998.

[4] J. Bolot, and T. Turletti, "Experience with Control Mechanisms for Packet Video," in *Proc. SIGCOMM'9*7, 1997.

[5] Guo Chuanxiong, "A SRR packet scheduler for flows in multi-service packet networks," Ph.D. thesis, Inst. of Comm. Eng. of China, April, 2000.

[6] D. Clark, and Wenjia Fang, "Explicit Allocation of Best-Effort Packet Delivery Service," *IEEE/ACM Trans. Networkin*g, vol. 6, Aug. 1998.

[7] D. Clark, "The Design Philosophy of the DARPA Internet Protocols," in *Proc. SIGCOMM'8*8, 1988.

[8] D. Clark, S. Shenker, and L. Zhang, "Supporting Real-time Applications in an Integrated Services Packet Network: Architecture and Mechanism," in *Proc. SIGCOMM'92,* 1992.

[9] J. Cobb, M. Gouda, and A. El-Nahas, "Time-Shift Scheduling—Fair Scheduling of Flows in High-Speed Networks," *IEEE/ACM Trans. Networkin*g, vol. 6, June, 1998.

[10] J. A. Cobb, and M. G. Gouda, "Flow Theory," *IEEE/ACM Trans. Networkin*g, vol.5, Oct. 1997.

[11] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm," in *Proc. SIGCOMM'8*9, 1989.

[12] S. Floyd, and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Trans. Networkin*g, vol. 1, Aug. 1993.

[13] S. Floyd, and V. Jacobson, "Link-share and Resource Management Models for Packet Networks," *IEEE/ACM Trans. Networkin*g, vol. 3, Aug. 1995.

[14] S. Floyd, and K. Fall, "Promoting the Use of End-to-End Congestion Control in the Internet," *IEEE/ACM Trans. Networkin*g, vol. 7, Aug. 1999.

[15] L. Georgiadis, R. Guerin, and R. Rajan, "Efficient Support of Delay and Rate Guarantees in an Internet," in *Proc. SIGCOMM'9*6, 1996.

[16] P. Goyal, H. M. Vin, and H Cheng, "Start-Time Fair Queueing: A Scheduling Algorithm for Integrated Services Packet Switching Networks," *IEEE/ACM Trans. Networkin*g, vol. 5, 1997.

[17] Pawn Goyal, and H. Vin, "Generalized Guaranteed Rate Scheduling Algorithms: A Framework," *IEEE/ACM Trans. Networkin*g, vol. 5, 1997.

[18] S. R. McCanne, "Scalable Compression and Transmission of Internet Multicast Video," Ph.D. thesis, UC. Berkeley, Dec. 1996.

[19] A. Parekh, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Network," Ph.D. thesis, Dept. Elect. Eng. and Comput. Sci., M.I.T., Feb. 1992.

[20] V. Paxson, and S. Floyd, "Why we don't know how to simulate the Internet," from: ftp://ftp.ee.lbl.gov/papers/wsc97.ps.

[21] O. Rose, "Traffic Modeling of Variable Bit Rate MPEG Video and its Impacts on ATM Networks," Ph.D. thesis, Wurezburger, Bericht, 02/97.

[22] D. Saha, S. Mukherjee, and S. Tripathi, "Carry-Over Round Robin: A Simple Cell Scheduling Mechanism for ATM Networks," *IEEE/ACM Trans. Networkin*g, vol.6, Dec. 1998.

[23] M. Shreedhar and G. Varghese, "Efficient Fair Queuing using Deficit Round Robin," in *Proc. SIGCOMM'9*5, 1995.

[24] D. Stiliadis, and A. Varma, "Rate-Proportional Servers: A Design Methodology for Fair Queueing Algorithms," *IEEE/ACM Trans. Networkin*g, vol. 6, Apr. 1998.

[25] D. Stiliadis, and A. Varma, "Efficient Fair Queueing Algorithms for Packet-Switched Networks, " *IEEE/ACM Trans. Networkin*g, vol. 6, Apr. 1998.

[26] A. Varma, and D. Stiliadis, "Hardware Implementation of Fair Queuing Algorithms for Asynchronous Transfer Mode Networks," *IEEE Com. Mag*., vol. 35, Dec. 1997.

[27] W. Weiss, "QoS with differentiated services," *Bell-labs Technical Journal*, oct-dec 1998.

[28] W. Willingers, and V. Paxson, "Where Mathematics meets the Internet," *Notes of the American Mathematical Societ*y, vol.45, Aug.1998.

[29] L. Zhang, "A New Architecture for Packet Switching Network Protocols," Ph.D. thesis, Dept. Elect. Eng. and Comput Sci., M.I.T., Aug. 1989.

[30] L. Zhang, S. Deering, D. Estrin, S. Shenker and D. Zappala, "RSVP: A New Resource ReServation Protocol," *IEEE Networ*k, 1993.

[31] The VINT Project, "ns Notes and Documentation", from http://www-mash.cs.berkeley.edu/ns/nsDoc.ps.gz.

APPENDIX

A. Proof of Proposition 3

Proof: 1, when $i = k-1$, the *k*th WSS

$S^k = S^{k-1}, k, S^{k-1} = S^{k-2}, (k-1), S^{k-2}, k, S^{k-2}, (k-1), S^{k-2}.$

Thus, the chains between two adjacent occurrences of element (*k*-1) are S^{k-2}, k, S^{k-2} or S^{k-2}, S^{k-2}.

2. Suppose the Proposition is correct for two adjacent occurrences of element $i (1 < i \le k-1)$, this means the WSS can be expressed as,

$$S^k = S^{i-1}, i, \ldots, i, S^{i-1}, x, S^{i-1}, i, \ldots, i, S^{i-1}$$

since $S^{i-1} = S^{i-2}, (i-1), S^{i-2}$,

$S^k = S^{i-2}, (i-1), S^{i-2}, i, \ldots, i, S^{i-2}, (i-1),$
$S^{i-2}, x, S^{i-2}, (i-1), S^{i-2}, i, \ldots, i, S^{i-2}, (i-1), S^{i-2}.$

Thus, the chain between two adjacent occurrences of element $(i-1)$ is,

$$S^{i-2}, S^{i-2} \text{ or } S^{i-2}, x, S^{i-2}.$$

Thus, the proposition follows by induction.

B. Proof of Theorem 2

Before the proof, we observe that the maximum value of $\left| w_f V_g(0,t) - w_g V_f(0,t) \right|$ only relates to flow *f* and flow *g* themselves in SRR. Though other flows may change the time distribution of V_f, V_g, they will not affect the service sequence of flow *f* and flow *g*. Thus, only the service sequence includes *f, g* is used in the proof. It is obvious that the theorem is correct when $w_f = 1$ or $w_g = 1$.

From Lemma 2, we know that at the end of each round, $\left| w_f V_g(0,\tau) - w_g V_f(0,\tau) \right| = 0$. Thus we only need to prove Theorem 2 in its first round. Under this condition, $V_g \le w_g, V_f \le w_f$.

Proof: We prove this theorem by induction.

1. When $k = 2, w_f, w_g \in \{1,2,3\}$. It is easy to prove that for all the 9 combinations of w_f, w_g,

$$\left| w_f V_g(0,t) - w_g V_f(0,t) \right| \le \max(w_f, w_g).$$

2. Suppose that the inequality is correct using a k*th* WSS, that is for any pair of w_f, w_g,

$$\left| w_f V_g(0,t) - w_g V_f(0,t) \right| \le \frac{k}{2} \max(w_f, w_g).$$

For any pair of f', g' using a (*k*+1)th WSS, w_f' and w_g' can be expressed as,

$w_f' = 2w_f + a_{f',0}$, $w_g' = 2w_g + a_{g',0}$, where $w_f' > 1, w_g' > 1, a_{f',0}, a_{g',0} \in \{0,1\}$.

Thus, the service sequence of flow f', g' can be expressed as,

$$S_{(k+1)}(f', g') = S_k(f, g), \{a_{f',0}.f, a_{g',0}.g\}, S_k(f, g).$$

We name the subsequence before $\{a_{f',0}.f, a_{g',0}.g\}$ the left part of $S_{(k+1)}(f', g')$, and the subsequence after $\{a_{f',0}.f, a_{g',0}.g\}$ the right part of $S_{(k+1)}(f', g')$.

With different values of $a_{f',0}, a_{g',0}$, there are 4 cases:

1). $a_{f',0} = 0, \quad a_{g',0} = 0$; 2). $a_{f',0} = 1, \quad a_{g',0} = 0$;

3). $a_{f',0} = 0, \quad a_{g',0} = 1$; 4). $a_{f',0} = 1, \quad a_{g',0} = 1$.

The 4 cases can be proven with similar method. Thus, we only show the proof of the last case.

As to this case, when $V_{f'} = V_f, V_{g'} = V_g$,

$$\left|w_{f'}V_{g'} - w_{g'}V_{f'}\right| = \left|(2w_f + 1)V_g - (2w_g + 1)V_f\right|$$

$$\le \left|2w_f V_g - 2w_g V_f\right| + \left|V_g - V_f\right| < \frac{k+1}{2}\max(w_{f'}, w_{g'}).$$

When $V_{f'} = w_f + 1, V_{g'} = w_g$,

$$\left|w_{f'}V_{g'} - w_{g'}V_{f'}\right| = \left|(2w_f + 1)w_g - (2w_g + 1)(w_f + 1)\right|$$

$$= \left|w_g + w_f + 1\right| < \frac{k+1}{2}\max(w_{f'}, w_{g'}).$$

When $V_{f'} = w_f + 1, V_{g'} = w_g + 1$,

$$\left|w_{f'}V_{g'} - w_{g'}V_{f'}\right| = \left|(2w_f + 1)(w_g + 1) - (2w_g + 1)(w_f + 1)\right|$$

$$= \left|w_g - w_f\right| < \frac{k+1}{2}\max(w_{f'}, w_{g'}).$$

When $V_{f'} = w_f + 1 + V_f, V_{g'} = w_g + 1 + V_g$,

$$\left|w_{f'}V_{g'} - w_{g'}V_{f'}\right| = \left|(2w_f + 1)(w_g + 1 + V_g) - (2w_g + 1)(w_f + 1 + V_f)\right|$$

$$= \left|2w_g V_f - 2w_f V_g + w_g - V_g - w_f + V_f\right|$$

$$\le \left|2w_g V_f - 2w_f V_g\right| + \left|w_g - V_g - w_f + V_f\right|$$

$$< \left|2w_g V_f - 2w_f V_g\right| + \max(w_f, w_g) \le \frac{k+1}{2}\max(w_{f'}, w_{g'}).$$

So, for all the 4 cases,

$$\left|w_{f'}V_{g'} - w_{g'}V_{f'}\right| \le \frac{k+1}{2}\max(w_{f'}, w_{g'}).$$

Thus, Theorem 2 follows by induction.