G-3: An $O(1)$ Time Complexity Packet Scheduler That Provides Bounded End-to-End Delay

Chuanxiong Guo, xguo@ieee.org
Institute of Communications Engineering, Nanjing, China

***Abstract*— In this paper, we present an $O(1)$ time-complexity packet scheduling algorithm which we call G-3 that provides bounded end-to-end delay for fixed size packet networks. G-3 is built over two round-robin schedulers SRR [10] and RRR [7] and several novel data structures. In G-3, bounded delay is provided by evenly distributing the binary coded weight of a flow into a Square Weight Matrix (SWM) and several Perfect Weighted Binary Trees (PWBTs). In order to achieve $O(1)$ time complexity, the SWM matrix is further spread by a Weight Spread Sequence (WSS) and each PWBT tree is spread by a corresponding Time-Slot Sequence (TSS), respectively. G-3 then performs packet scheduling by sequential scanning the WSS and TSS sequences. G-3 can be implemented in high-speed packet networks to provide bandwidth guarantee, fairness, and bounded delay due to its $O(1)$ time complexity.**

I. Introduction

The Internet has become an information infrastructure of the world. As an infrastructure, it is expected to support various applications such as voice over IP, video conferencing, and file sharing, which have very different bandwidth, delay, jitter, and packet loss requirements. Many frameworks have been proposed to enable a multi-service Internet (e.g., [2], [3]). One of the key technologies for multi-service support is a packet scheduler. A packet scheduler decides which flow to serve when the output link has finished transmitting the previous packet. A good packet scheduler should provide bounded end-to-end delay, fair bandwidth sharing, low time complexity, and is simple to implement. Due to the importance of packet scheduler, packet scheduling algorithms have been studied extensively ever since [6] and [13].

One category of packet scheduling algorithms maintain time-stamps for each flow (e.g., Weighted Fair Queueing (WFQ) [6]). Using WFQ as an example, the time-stamp tagged to a flow is calculated from an ideal fluid Generalized Processor Sharing (GPS) model. WFQ always serves the flow with the largest time-stamp and it provides bounded end-to-end delay [15]. The time complexity of WFQ is $O(N)$, where N is the number of active flows in the scheduler. Many schemes were then proposed to either reduce time-complexity or improve performance of WFQ, e.g., [1], [5], [8], [9], [18]. Most of these algorithms provide bounded end-to-end delay and all of them have at least $O(\log N)$ time complexity. $O(\log N)$, however, does not scale well for high-speed networks which only have nanoseconds to make scheduling decisions. For example, for a 40 Gb/s (OC-768) link, a packet of size 200 bytes will be transmitted in 0.04 us. When $N = 10^6$, $\log_2 N = 20$, which means that a WFQ scheduler must perform each comparison in less than 2 nanoseconds!

Another category of algorithms use round-robin instead of time-stamp to perform scheduling (e.g., RRR [7], SRR [10], Aliquem [12], and DRR [17]). Due to the simplicity nature of round-robin, most of the round-robin schedulers have $O(1)$ time-complexity. But they generally cannot provide bounded delay.

Recently, there emerged new scheduling algorithms by integrating time-stamp and round-robin schemes (GR^3 [4], STRR [16], FRR [21]). These algorithms divide flows into different groups according to their weights, and use time-stamp based schedulers for inter-group scheduling and round-robin based schedulers for intra-group scheduling. However, the number of groups in these algorithms is at least $O(\log N)$, these algorithms therefore can hardly be considered $O(1)$ time complexity schemes. Moreover, the delays of these schemes are either not clear or in proportion to N.

Since all the proposed schemes have either $O(\log N)$ time complexity and bounded delay or $O(1)$ time complexity and unbounded delay, it is therefore natural to ask if $O(1)$ time complexity schedulers cannot provide bounded delay. In [20], the authors showed that under the comparison computing model, which is used by the time-stamp based schemes, the lower time complexity bound to provide bounded delay is $\Omega(\log N)$. However, round-robin schedulers do *not* use comparison for scheduling, it is therefore still unknown if there exists $O(1)$ time complexity, non-time-stamp based schedulers that provide bounded delay.

In this paper, we propose a round-robin packet scheduler, which we call G-3, that has both $O(1)$ time complexity and bounded delay for fixed size packet networks. G-3 is built over two previously proposed schedulers SRR [10], which has $O(1)$ time complexity but unbounded delay, and RRR [7], which provides good delay property but has time complexity comparable to $O(\log N)$. The novelty of G-3 lies in several newly introduced data structures. In G-3, each flow is assigned a weight that is in proportion to its reserved rate. The weight is represented in its binary form and distributed into a Square Weight Matrix (SWM) and several Perfect Weighted Binary Trees (PWBT). The weights of the flows are evenly distributed into SWM and PWBTs, so that bounded delay is achieved. G-3 then spreads the SWM matrix by a Weight Spread Sequence (WSS, which is introduced in SRR) and each PWBT tree by a Time-Slot Sequence (TSS). G-3 achieves $O(1)$ time complexity by reducing packet scheduling to sequential scanning

the WSS and TSS sequences. We also have designed variants of G-3 to handle variable packet size (see [11]), these variants are not further discussed in this paper due to space limitation. To the best of our knowledge, G-3 is the first $O(1)$ time-complexity scheduler that provides bounded end-to-end delay.

The rest of the paper is organized as follows. In Section II, we overview SRR and RRR and provide analytical delay bounds for SRR. In Section III, we introduce several key data structures and then formally present G-3. The delay and space-time complexities of G-3 are analyzed in Section IV. Section V demonstrates the delay property of G-3 using simulations. Section VI concludes the paper.

II. Smoothed Round Robin and Recursive Round Robin

We assume there are N flows, numbered from f_0 to f_{N-1}, in the scheduler. The reserved rate for flow f_i is r_i. A flow f_i is assigned a weight w_i which is in proportion to its rate r_i. Without losing generality, we assume $w_i = r_i$. We assume packets are of the same size L.

In an *ideal* scheduler, for a flow f with reserved rate r, the total bytes that have been transmitted at time t is $S^{id}(t-t_0) = r(t-t_0)$, where $t_0 < t$ is the arrival time of flow f. $S^{id}(t-t_0)$ may be less than $r(t-t_0)$ if the arrival bits of a flow is less than $r(t-t_0)$. If this is the case, the flow can be split into several flows which are non-overlapped in their time duration, each with the property $S^{id}(t-t_0) = r(t-t_0)$. It is easy to observe that, in the ideal scheduler, the ith $(i > 0)$ packet of f will be finished transmission at time $t_i^{id} = t_0 + \frac{i \times L}{r}$. We further denote the bits transmitted for f at time t in a real packet scheduler ps as $S^{ps}(t-t_0)$. We call $S^{id}(t-t_0)$ the ideal service curve of f and $S^{ps}(t-t_0)$ the service curve of f under ps, respectively.

Definition 1: A packet scheduler ps provides bounded delay for flow f, if the finish time $t_i^{ps}(i > 0)$ satisfies

$$t_i^{ps} \leq t_0 + \frac{i \times L}{r} + d^{ps} \tag{1}$$

where r is the reserved rate of f, and d^{ps} is defined as

$$d^{ps} = \max_{i>0}\{d_i^{ps}\} = \max_{i>0}\{t_i^{ps} - t_i^{id}\} \leq const \tag{2}$$

where $const$ is a constant that is independent of the number of active flows N. Definition 1 is identical to say that $S^{ps}(t-t_0) \geq S^{id}(t-t_0-\tau)$, where τ is a constant. The adoption of Definition 1 can simplify the analysis in the rest of the paper. In next subsections, we briefly overview SRR and RRR and provide several new results of SRR which will be used in analyzing G-3.

A. Overview of the Smoothed Round Robin

There are two key data structures in SRR [10]: Weight Spread Sequence (WSS) and Weight Matrix (WM).

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

$$WM = \begin{bmatrix} a_{0,(k-1)} & a_{0,(k-2)} & \cdots & a_{0,0} \\ a_{1,(k-1)} & a_{1,(k-2)} & \cdots & a_{1,0} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N-1,(k-1)} & a_{N-1,(k-2)} & \cdots & a_{N-1,0} \end{bmatrix} \tag{3}$$

where $a_{i,j} \in \{0,1\}$, $0 \leq i \leq N-1$, and $0 \leq j \leq k-1$. The columns of the Weight Matrix are numbered from left to right as col_{k-1}, col_{k-2}, $\cdots$, col_0, respectively. 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

$$0 \leq y_j \leq N \tag{4}$$

and

$$\sum_{i=0}^{N-1} r_i = \sum_{i=0}^{N-1}\sum_{j=0}^{k-1}\{a_{i,j}2^j\} \leq C \tag{5}$$

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

$$WSS^k = \{a_i^k | 1 \leq i \leq 2^k - 1\} \tag{6}$$

where a_i^k is defined recursively as

$$a_i^k = \begin{cases} a_i^{k-1}, & 1 \leq i \leq 2^{k-1}-1 \\ k, & i = 2^{k-1} \\ a_{i-2^{k-1}}^{k-1}, & 2^{k-1} < i \leq 2^k - 1 \end{cases} \tag{7}$$

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 $2^k - 1$.

SRR then scans the terms of WSS cyclically. When the value of the current scanned term is j, column $k-j$ of the Weight Matrix is selected. For each occurrence $a_{i,k-j} \neq 0$ in this column, packet from f_i is then transmitted. Due to the desired properties of WSS and WM, flows are served fairly according to their weights.

Readers are referred to [10] for details of SRR. Though SRR has strict $O(1)$ time complexity for packet scheduling, it does not provide bounded delay. We will show analytically that the delay bound of SRR is in proportion to the number of active flows in next subsection (see Theorem 1).

B. Delay property of SRR

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

$$TS_n(i) \leq \frac{iC}{2^n} + \theta(n)N. \tag{8}$$

where $\theta(n) < n$. The proof of Lemma 1 is given in [11] and is omitted here due to space limitation. From Lemma 1, we have theorem as follows.

Theorem 1: In SRR, for a flow f with rate $r = w = 2^n$ $(0 \leq n < k)$, its delay bound

$$d^{srr} \leq \theta(n) \times N \times \frac{L}{C}. \tag{9}$$

Proof: The time when f finishes serving its ith packet, t_i, is

$$\begin{aligned} t_i &= t_0 + TS_n(i)\tfrac{L}{C} \\ &\leq t_0 + \tfrac{iL}{2^n} + \theta(n) \times N \times \tfrac{L}{C} \end{aligned}$$

Therefore

$$d_i^{srr} = t_i - (t_0 + \tfrac{iL}{2^n}) \leq \theta(n) \times N \times \tfrac{L}{C}$$

for all $i > 0$. ■

In fact, we have lemma for flows with arbitrary rates as follows.

Lemma 2: In SRR, for a flow f with rate $r = w = \sum_{i=1}^{m} 2^{n_i}$ $(0 \leq n_1 < n_2, \cdots, < n_m \leq k-1)$, its delay bound

$$d^{srr} < \theta(n_m) \times N \times \frac{L}{C} + (m-1)\frac{L}{r} \tag{10}$$

where m is the number of non-zero binary coefficients of r.

The proof of Lemma 2 is also given in [11]. Both Theorem 1 and Lemma 2 show that d^{srr} is in proportion to N. SRR therefore does not provide bounded delay.

C. Overview of the Recursive Round Robin

In RRR [7], the output bandwidth is normalized to 1, and the weight of a flow is $w = \frac{r}{C}$. The normalized weight is further represented as $w \approx \sum_{j=1}^{g} b_j 2^{-j}$, where $b_j \in \{0, 1\}$ is the coefficient of w.

The core data structure of RRR is a Weighted Binary Tree (WBT). This WBT tree has $g+1$ levels, with root node $v_{0,0}$ at level 0, and the two children of $v_{0,0}$, namely $v_{1,0}$ and $v_{1,1}$, at level 1, etc. A node of the WBT tree is denoted as $v_{l,i}$, where l indicates its level and i represents its id at level l. We have $0 \leq l \leq g$ and $0 \leq i < 2^l$. An intermediate node $v_{l,i}$ has a left child $v_{l+1,2i}$ and a right child $v_{l+1,2i+1}$. See Fig. 1 for an example of such a WBT tree. A node at level l of the WBT tree is associated with a weight 2^{-l}. The root node $v_{0,0}$, therefore, has weight 1, and its children $v_{1,0}$ and $v_{1,1}$ have weights $\frac{1}{2}$, respectively.

We use an example as illustrated in Fig. 1 to show how the WBT tree is formed. Suppose the scheduler is idle at the beginning (i.e., the WBT tree has only one $v_{0,0}$ node), then a flow f_1 with rate $\frac{1}{16}$ comes. Since the only unallocated node is $v_{0,0}$, RRR splits it into two nodes $v_{1,0}$ and $v_{1,1}$. Since the weight of $v_{1,0}$, $\frac{1}{2}$, is larger than $\frac{1}{16}$, RRR then splits $v_{1,0}$ into $v_{2,0}$ and $v_{2,1}$. Similarly, $v_{2,0}$ is split into $v_{3,0}$ and $v_{3,1}$, $v_{3,0}$ is split into $v_{4,0}$ and $v_{4,1}$. Since the weight of $v_{4,0}$ is $\frac{1}{16}$, $v_{4,0}$ is therefore allocated to f_1.

Fig. 1 shows the WBT tree after flows $f_1 - f_4$ have been added. The weights of the flows are $w_1 = \frac{1}{16}$, $w_2 = \frac{1}{8}$, $w_3 = w_4 = \frac{1}{4}$, respectively. In RRR, the unallocated leaf nodes of the WBT tree are allocated to a special flow f_0, which is an idle flow to consume the unused bandwidth. After adding these flows in, $v_{4,0}$, $v_{3,1}$, $v_{2,1}$, and $v_{2,2}$ are allocated to flows f_1, f_2, f_3, f_4, respectively. $v_{4,1}$ and $v_{2,3}$ are allocated to the idle flow f_0.

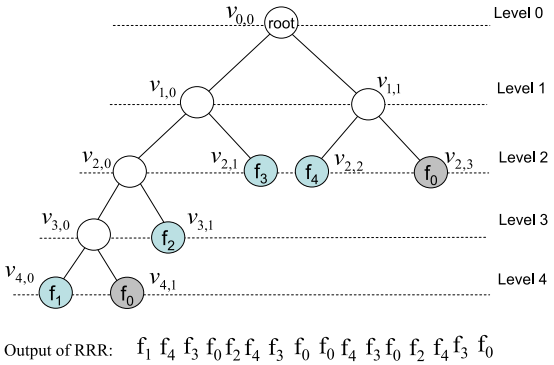

Fig. 1. An example to illustrate how RRR works. The core data structure of RRR is a Weighted Binary Tree (WBT).

After the formation of the WBT tree, scheduling is performed as illustrated in Fig. 2. Each intermediate node $v_{i,j}$ in the WBT tree is associated with a boolean variable $flip(v_{i,j})$. $flip(v_{i,j})$ is initialized to 0 and changes its value each time $v_{i,j}$ is visited.

The scheduling procedure of RRR:

```
1  while (the scheduler in busy-period) {
2      v = v_{0,0};
3      while (v is an intermediate node){
4          if (flip(v) == 0)
5              tmp = left_child(v);
6          else
7              tmp = right_child(v);
8          flip(v) = flip(v) XOR 1;
9          v = tmp;
10     }
11     f = flow id that corresponds to v;
12     ServeFlow (f);
13 }
```

Fig. 2. The scheduling procedure of RRR.

Using the scheduling procedure to serve the tree in Fig. 1, we get the following output service sequence in one round:

$$f_1 f_4 f_3 f_0 f_2 f_4 f_3 f_0 f_0 f_4 f_3 f_0 f_2 f_4 f_3 f_0.$$

In [7], the authors showed that, in RRR, for a flow f with weight $w = \sum_{i=1}^{m} 2^{-n_i}$ $(0 < n_1 < n_2, \cdots, < n_m \leq g)$, its

delay bound

$$d^{rrr} \leq \frac{m \times L}{r} \tag{11}$$

where m is the number of non-zero coefficients of w.

RRR has two issues. First, the value of m depends on both r and C. Since the bandwidth may not be known in advance, RRR hence cannot control the value of m to small values. For instance, for a flow with rate $r = 32kb/s$, $C = 10mb/s$, and $g = 20$, its $m = 6$. When $L = 200$ bytes, the delay bound d^{rrr} is as large as 300 ms!

Second and most importantly, the time complexity of RRR is high. According to Fig. 2, RRR needs g steps to select a packet, where g is the depth of the WBT tree. The time complexity for packet scheduling is therefore g. This time complexity does not have advantage over the time-stamp based algorithms, whose time complexity is generally $O(\log N)$, since g is at least as large as $\log_2 N$. A naive approach to reduce the time-complexity of RRR is to pre-compute and store the scheduling sequence in advance. This approach, however, does not solve the problem since the time complexity of pre-computing is as high as $O(C)$ ($\gg N$) and pre-computing needs to be performed for each flow arrival and departure.

III. The G-3 packet scheduling algorithm

Before we present G-3, we first introduce several novel data structures: Square Weight Matrix (SWM), Time-Slot Sequence (TSS), and Time-slot Array (TArray).

A. Square Weight Matrix

Definition 3: A $k \times k$ weight matrix is a Square Weight Matrix (SWM), if it can be represented as

$$SWM = \begin{bmatrix} 0 & 0 & \cdots & 0 & a_0 \\ 0 & 0 & \cdots & a_1 & 0 \\ \vdots & \vdots & \ddots & \vdots & \\ a_{k-1} & 0 & \cdots & 0 & 0 \end{bmatrix} \tag{12}$$

where $a_i \in \{0, 1\}$ and $a_{k-1} = 1$.

Lemma 3: Suppose a SWM is scheduled by SRR, then for a flow f with rate 2^n $(0 \leq n \leq k-1)$ in this SWM, its delay bound

$$d \leq \theta(n) \times \frac{L}{C} \leq \theta(k-1) \times \frac{L}{C}. \tag{13}$$

This lemma follows directly from Theorem 1, since the sum of each column of a SWM $y_j \leq 1$ $(0 \leq j \leq k-1)$.

B. Time-Slot Sequence and Time-Slot Array

The binary tree used in RRR provides good scheduling output, but experiences scheduling time complexity comparable to $O(\log N)$. In order to reduce time complexity, we first extend the binary tree of RRR into a Perfect Weighted Binary Tree (PWBT) by adding descendent nodes to the leaf nodes that are not at the lowest level. Fig. 3 shows such a PWBT tree constructed from the binary tree of Fig. 1. An intermediate node of a PWBT tree is said to *own* a leaf node if the leaf node is its descendent. For example, in Fig. 3, node $v_{2,1}$ owns $v_{4,4}$, $v_{4,5}$, $v_{4,6}$, and $v_{4,7}$. When an intermediate node in the PWBT tree is allocated to a flow f, all the leaf nodes owned by that node are also allocated to f. In Fig. 3, since node $v_{2,1}$ is allocated to f_3, then $v_{4,4}$, $v_{4,5}$, $v_{4,6}$, and $v_{4,7}$ are allocated to f_3 consequently.

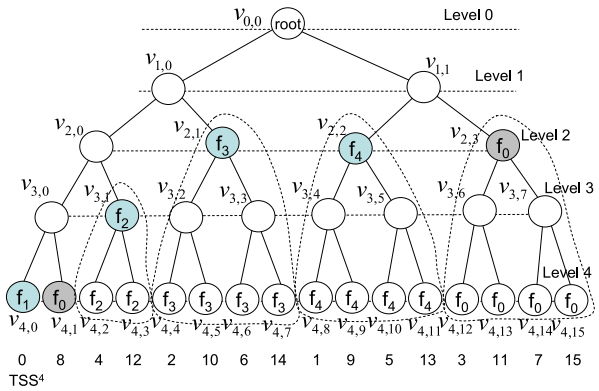

Fig. 3. The Perfect Weighted Binary Tree (PWBT) constructed from the binary tree of RRR.

From the scheduling procedure, RRR will first visit $v_{4,0}$, then visits $v_{4,8}$, $v_{4,4}$, $\cdots$, $v_{4,7}$, $v_{4,15}$. That is, the indices of the visited leaf nodes can be described as follows.

$$0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15, \tag{14}$$

In fact, the above sequence can be described by a newly introduced set of sequences which we call Time-Slot Sequence (TSS). TSS is defined as follows.

Definition 4: A set of Time-Slot Sequence (TSS) is defined as

$$TSS^n = \{b_i^n | 0 \leq i < 2^n\} \tag{15}$$

where $n \geq 0$ is the order of the corresponding TSS sequence and b_i^n is the ith term of TSS^n, and TSS^0 = $\{b_0^0 = 0\}$. The ith term of TSS^n, b_i^n $(n > 0)$, is defined recursively as

$$b_i^n = \begin{cases} 2b_i^{n-1}, & 0 \leq i < 2^{n-1} \\ 2b_{i-2^{n-1}}^{n-1} + 1, & 2^{n-1} \leq i < 2^n \end{cases} \tag{16}$$

From Definition 4, we get TSS^1={0, 1}, TSS^2={0, 2, 1, 3}, TSS^3={0, 4, 2, 6, 1, 5, 3, 7}, and TSS^4 is shown in (14).

Definition 5: For a n bits binary number $i = b_{n-1}b_{n-2}\cdots b_0$, where $b_j \in \{0, 1\}$ and $0 \leq j < n$, its reversed binary number is defined as

$$RB(i, n) = b_0 b_1 \cdots b_{n-1}. \tag{17}$$

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

The following lemma holds for TSS.

Lemma 4: For a nth TSS sequence, the value of its ith term b_i^n is

$$b_i^n = RB(i, n) \tag{18}$$

where $n > 0$ and $0 \leq i < 2^n$.

Proof: We prove this lemma by induction.

1) It is obvious to verify that the lemma is correct when $n = 1$.

2) Suppose the lemma is correct for TSS^n, i.e., for $i = b_{n-1}b_{n-2}\cdots b_0$, $b_i^n = RB(i, n) = b_0 b_1 \cdots b_{n-1}$.

Then for TSS^{n+1}, when $0 \le i < 2^n$, $b_i^{n+1} = 2b_i^n = b_0b_1\cdots b_{n-1}0$, which is exactly $RB(i, n+1)$. When $2^n \le i+2^n < 2^{n+1}$, $b_{i+2^n}^{n+1} = 2b_i^n+1$, where $0 \le i < 2^n$. Therefore $b_{i+2^n}^{n+1} = b_0b_1\cdots b_{n-1}1 = RB(i+2^n, n+1)$.

Lemma 4 therefore follows by induction. ■

For a node $v_{l,i}$ of a PWBT tree of depth n ($0 \le l \le n$ and $0 \le i < 2^l$), the set of leaf nodes that owned by $v_{l,i}$ is then $\{v_{n,x} | i2^{n-l} \le x < (i+1)2^{n-l}\}$. We can express i in its binary form as $b_{l-1}b_{l-2}\cdots b_0$. The indexes of its leaf nodes can then be expressed as $b_{l-1}b_{l-2}\cdots b_000\cdots0$, $b_{l-1}b_{l-2}\cdots b_000\cdots1$, $\cdots$, $b_{l-1}b_{l-2}\cdots b_011\cdots1$. The corresponding reversed binary numbers of the indexes are therefore $00\cdots0b_0b_1\cdots b_{l-1}$, $10\cdots0b_0b_1\cdots b_{l-1}$, $\cdots$, $11\cdots1b_0b_1\cdots b_{l-1}$. The set of the reversed binary numbers can therefore be denoted as $RBS(v_{l,i}) = \{rb_x | 0 \le x < 2^{n-l}\}$, where $rb_x = x2^l + b_0b_1\cdots b_{l-1}$. We therefore get lemma as follows.

Lemma 5: The terms of $RBS(v_{l,i})$ are distributed evenly in the nth TSS ($n \ge l$), and the distance between two adjacent terms of $RBS(v_{l,i})$ is 2^l.

Based on the idea of TSS, we further introduce an array which we call Time-Slot Array (TArray). For a PWBT tree with level n, there is a corresponding time-slot array TArray^n of size 2^n. The value of TArray^n is set as follows.

for ($i = 0$; $i < 2^n$; $i++$)

$TArray^n[i]$ = id of the flow that occupies $v_{n,RB(i,n)}$;

For the PWBT tree in Fig. 3, its corresponding time-slot array is therefore

$$TArray^4 = \{f_1f_4f_3f_0f_2f_4f_3f_0f_0f_4f_3f_0f_2f_4f_3f_0\}.$$

This TArray exactly describes the service sequence of RRR. Hence, with the introducing of time-slot sequence and time-slot array, we can directly decide which flow to serve without traversing the binary tree from root to leaf.

C. An example to show how G-3 works

Before we formally present G-3, we use an example to illustrate how the newly introduced data structures are integrated together. Suppose we have 7 flows numbered from f_0 to f_6 with rate 1; 2 flows f_7 and f_8 with rate 2; 1 flow f_9 with rate 4. The bandwidth of the output link is $C = 15$.

The bandwidth can be expressed as $C = a_3 \times 2^3 + a_2 \times 2^2 + a_1 \times 2^1 + a_0 \times 2^0$, where the binary coefficients $a_3 = a_2 = a_1 = a_0 = 1$. The coefficients are then used to form a SWM matrix as below.

$$SWM = \begin{bmatrix} 0 & 0 & 0 & a_0 \\ 0 & 0 & a_1 & 0 \\ 0 & a_2 & 0 & 0 \\ a_3 & 0 & 0 & 0 \end{bmatrix}.$$

The WSS that corresponds to this SWM is

$$WSS^4 = \{1,2,1,3,1,2,1,4,1,2,1,3,1,2,1\}.$$

For each binary coefficient $a_i (0 \le i \le 3)$, there is a PWBT tree $Tree^i$. For each $Tree^i$, there is a corresponding $TArray^i$ as below.

$$TArray^3 = \{f_7, f_9, f_8, f_9, f_7, f_9, f_8, f_9\},$$
$$TArray^2 = \{f_3, f_5, f_4, f_6\},$$
$$TArray^1 = \{f_1, f_2\},$$
$$TArray^0 = \{f_0\}.$$

These 4 TArrays are formed from the 4 PWBT trees as illustrated in Fig. 4. The detailed procedure of how to generate the TArrays has been explained in Section III-B. There is a pointer p_w points to the current scanned position of the WSS^4 sequence. For each $TArray^i$, there is also a pointer p_i points to the current scanned position of $TArray^i$. p_w and the pointers (p_i) of the TArrays are initialized to point to their first terms, respectively.

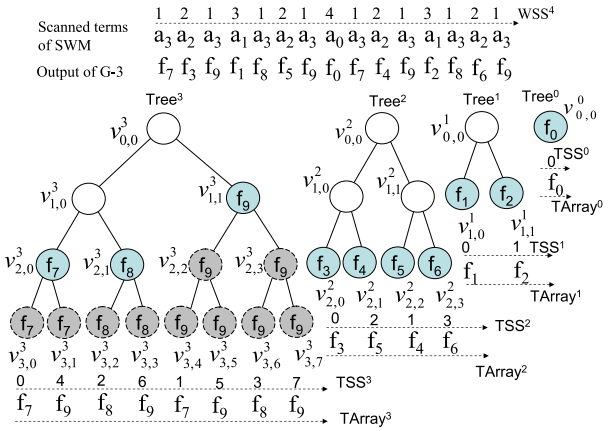

Fig. 4. An example to show how G-3 works.

G-3 then scans the WSS^4 sequence cyclically. When the current term of WSS^4 is i, column $j = 4 - i$ of the SWM is selected, and $TArray^j$ is selected consequently. Flow $f = TArray^j[p_j]$ is then served. After that, p_w and p_j are increased by one to point to their next terms, respectively. The output service sequence of G-3 in one round is therefore

$$f_7f_3f_9f_1f_8f_5f_9f_0f_7f_4f_9f_2f_8f_6f_9.$$

For the same set of flows, the service sequence of SRR is

$$f_9f_7f_8f_9f_0f_1f_2f_3f_4f_5f_6f_9f_7f_8f_9.$$

These two service sequences show that G-3 produces a much smoother output than SRR. For example, as to f_9, the distances between two successive appearances of f_9 are 3, 4, 4, and 4 in G-3, whereas the distances are 1, 3, 8, and 3 in SRR.

D. Description of G-3

In G-3, we assume that a flow is added into the scheduler by a call admission controller (CAC) and removed from the scheduler by a signalling protocol, and packets are classified to different flows by a packet classifier. A flow with packets in the scheduler is said to be *backlogged* and a flow with no packet is said to be *non-backlogged*. In G-3, we allocate the unallocated bandwidth to the best-effort flow f_0.

In G-3, the bandwidth of the output link C is denoted as $C = \sum_{i=1}^{h} 2^{n_i}$, where $0 \le n_1 < n_2, \cdots, n_h = \lfloor \log_2 C \rfloor$ and $h \le \lfloor \log_2 C \rfloor + 1$ is the number of non-zero binary coefficients of C. The weight of a flow f is its reserved rate, i.e., $w = r$.

There are four major data structures in G-3: A SWM matrix as defined in (12) to represent the binary coefficients of the output bandwidth C; a kth WSS sequence, where $k = \lfloor \log_2 C \rfloor + 1$; h binary trees $\{Tree^{n_1}, Tree^{n_2}, \cdots, Tree^{n_h}\}$ with $Tree^{n_i}$ represents 2^{n_i} of C (the root node $v_{0,0}^{n_i}$ of $Tree^{n_i}$ therefore has weight 2^{n_i}); and h time-slot arrays $\{TArray^{n_1}, TArray^{n_2}, \cdots, TArray^{n_h}\}$, to spread the corresponding h binary trees. There is a pointer p_w points to the current scanned term of WSS, and for each $TArray^i$, there is a pointer p_i points to the current scanned term of $TArray^i$. These points are initialized to point to their first terms, respectively.

There are k linked lists, denoted as $\{List_0, List_1, \cdots, List_{k-1}\}$. These lists are used to track the unallocated leaves of the h binary trees. For each $Tree^i$, the corresponding $List_i$ is initialized to have a node $v_{0,0}^i$, where $v_{0,0}^i$ is the root of $Tree^i$. We will show later that when the WBT trees are in good shape, each list has at most one node.

Fig. 5 gives the formal description of G-3. There are 3 subprocedures: A *Schedule* procedure to decide which flow to serve; an *Add_flow* to add an incoming flow into the scheduler; and a *Del_flow* to remove a flow when the flow leaves the system. Note that when a flow becomes non-backlogged, *Del_flow* will *not* be invoked. *Schedule* is performed per-packet level, whereas *Add_flow* and *Del_flow* are invoked only for flow arrivals and departures.

The *Schedule* procedure is quite simple: When there are packets in the system (line 4), *Schedule* scans the WSS sequence (line 5). If the corresponding term of the SWM is not 0 (line 6), it then selects the corresponding TArray and retrieves the flow id (line 7). If the selected flow is backlogged, it then uses *ServeFlow* to transmit a packet for that flow (line 8). *ServeFlow*(f) is to dequeue the first packet from f and then transmit the packet. If the selected flow is idle, it then executes *idle_sched*. In G-3, *idle_sched* simply assigns the idle time-slots to the best-effort flow f_0. Line 10 and line 12 are to update the pointers p_w of WSS and p_i of TArray^i, respectively.

Add_flow is invoked when a new flow is accepted by the CAC controller. For each of the m non-zero binary coefficient, *Add_flow* tries to allocate a node from the binary trees (line 14 and 15). Using 2^{n_j} as an example, *Add_flow* first tries to get a free node from the k linked lists (*get_free_node*, line 16). If it cannot get a free node, *Add_flow* fails and return (line 17). If the weight of the returned node $v_{l,i}^n$ is larger than 2^{n_j}, *split* is called (line 18 and 19). *split* first divides $v_{l,i}^n$ to $v_{l+1,2i}^n$ and $v_{l+1,2i+1}^n$. If the weight of $v_{l+1,2i}^n$ is still larger than 2^{n_j}, *split* then recursively divides $v_{l+1,2i}^n$, etc. The weight of the returned node is therefore 2^{n_j} and the node is then allocated to f. After that, the corresponding TArray is updated (line 20).

When a flow f leaves the scheduler, *Del_flow* is called. For each node of the binary trees that allocated to f, *Del_flow* updates the corresponding TArray (i.e., set the terms that occupied by f to f_0) (line 24). If the sibling node of the release node is also idle, *merge* operation is performed (line 26), else *Del_flow* just adds the node into the corresponding linked list (line 28). *merge* is the reverse operation of *split*.

```
Schedule:
1   integer: i, j;
2   p_w = 1;
3   for (i = 0; i < k; i++) p_i = 0;
4   while (the scheduler in busy-period) {
5     i = k − WSS^k[p_w];
6     if (a_i == 1) {
7       f = TArray^i[p_i];
8       if (f is backlogged) ServeFlow(f);
9       else idle_sched( );
10      p_i = (p_i + 1) mod 2^i;
11    }
12    p_w = 1 + p_w mod (2^k − 1);
13  }
Add_flow(f, w_f):
14  w_f = Σ_{j=1}^{m} 2^{n_j}, (0 ≤ n_1 < n_2 < ··· < n_m < k);
15  for (j = m; j ≥ 1; j − −){
16    v^n_{l,i} = get_free_node(n_j);
17    if (v^n_{l,i} == NULL) return FALSE;
18    if ( 2^{n−l} > 2^{n_j})
19      v^n_{l,i} = split(v^n_{l,i}, n_j);
20    update (TArray^n, v^n_{l,i}, f);
21  }
22  return TRUE;
Del_flow(f):
23  for (each node v^n_{l,i} assigned to f) {
24    update (TArray^n, v^n_{l,i}, f_0);
25    if ( its sibling v^n_{l,i'} is idle)
26      merge(v^n_{l,i}, v^n_{l,i'})
27    else
28      add v^n_{l,i} into List_{n−l};
29  }
```

Fig. 5. Description of G-3. G-3 has 3 major components: *Schedule* decides which flow to serve, *Add_flow* adds a new flow into the scheduler, and *Del_flow* removes a flow from the scheduler.

In both *Del_flow* and *Add_flow*, *update* ($TArray^n$, $v_{l,i}^n$, f) is used to update TArray^n when a flow arrives or departures. The number of terms in TArray^n that corresponding to node $v_{l,i}^n$ is 2^{n-l}. The number of terms that needs update in all the TArrays is therefore w_f. In G-3, we can further reduce the time-complexity of *update* by carrying out *update* and *Schedule* parallelly as follows. We first update the term of TArray that will be firstly visited by *Schedule*, then update the rest terms sequentially. *update* therefore only needs to update the terms before *Schedule* visits them. G-3 visits the leaf nodes of $v^n(l,i)$ as described by set RBS($v_{l,i}^n$) (which is defined in Section III-B). The first term of RBS($v_{l,i}^n$) that will be visited by *Schedule* is $x = \{RB(i,n) + y2^l\} \, mod \, 2^n$, where $y = \lceil \frac{p_n - RB(i,n)}{2^l} \rceil$. The *update* operation can therefore be carried out as follows.

update ($TArray^n$, $v_{l,i}^n$, f):

for ($j = 0$; $j < 2^{n-l}$; $j++$){
$tmp = (x + j2^l)\ mod\ 2^n$;
$TArray^n[tmp] = f$;
}

Note that G-3 may face the bandwidth fragmentation problem as described by the following example: After a period of time, the even-numbered leaf nodes may be allocated to flows with weight 1, whereas the odd-numbered leaf nodes are free. If a flow with rate 2 comes, the scheduler will not be able to accept the flow even if half of the leaf nodes are free. In order to overcome this bandwidth fragmentation problem, we introduce a background *Shaping* procedure to adjust the shape of the binary trees as illustrated in Fig. 6.

In Fig. 6, $v^n_{n-l,i}$ and $v^m_{m-l,j}$ are two nodes in $List_l$, which contains the unallocated nodes of the binary trees. The siblings of these two nodes are denoted as $v^n_{n-l,i'}$ and $v^m_{m-l,j'}$, respectively. (Note that $i' = i \pm 1$ and $j' = j \pm 1$). $v^n_{n-l,i'}$ is allocated to flow f, $v^m_{m-l,j'}$ is allocated to flow g.

The Shaping Procedure
marking:
while(1){
for (each linked list $List_l$)
if (exists two idle nodes $v^n_{n-l,i}$ and $v^m_{m-l,j}$)
add $swapping$ flags to $v^n_{n-l,i'}$ and $v^m_{m-l,j'}$;
}
swapping:
if ($v^n_{n-l,i'}$ is visited first) {
update ($TArray^m$, $v^m_{m-l,j}$, f);
update ($TArray^n$, $v^n_{n-l,i'}$, f_0);
merge ($v^n_{n-l,i}$, $v^n_{n-l,i'}$);
}else { /*$v^m_{m-l,j'}$ is visited first) */
update ($TArray^n$, $v^n_{n-l,i}$, g);
update ($TArray^m$, $v^m_{m-l,j'}$, f_0);
merge ($v^m_{m-l,j}$, $v^m_{m-l,j'}$);
}

Fig. 6. *Shaping* is a background procedure to keep the WBT trees of G-3 in good shape.

There are two operations in *Shaping*: a *marking* operation to add *swapping* flags to the siblings of the idle nodes, a *swapping* operation to be invoked after one of the marked sibling nodes is scheduled by G-3. Since the swapping operation takes place immediately after an allocated node has been visited, the swapped flow will be better off. Hence the *Shaping* procedure will not affect the delay property of the swapped flow. It is easy to observe that there is at most one node in each $List_l$ when no swapping operation can take place.

IV. Properties of G-3

A. Delay and Fairness

Lemma 6: For a flow f with rate 2^n $(0 \le n \le k-1)$ in G-3, its delay bound

$$d \le \theta(k-1) \times \frac{L}{C}. \tag{19}$$

Proof: Based on the description of G-3, flow f must be represented by one of the terms $a_m (m \ge n)$ of the SWM matrix.

From Lemma 5, flow f will be served once when a_m is served every 2^{m-n} times. We therefore have

$$TS_n(i) = TS_m(i2^{m-n} - \delta)$$

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

From (8), we have

$$TS_m(j) \le \frac{jC}{2^m} + \theta(m).$$

Thus

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

Therefore

$$d_i = t_0 + TS_n(i)\frac{L}{C} - (t_0 + i\frac{L}{2^n}) \le \theta(m) \times \frac{L}{C}$$

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

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

■

For flow with arbitrary rate, we have the following theorem.

Theorem 2: For a flow f with rate $r = w = \sum_{l=1}^{m} 2^{n_l}$ $(0 \le n_1 < n_2, \cdots, n_m \le k-1)$ in G-3, its delay bound

$$d^{g_3} \le \theta(k-1) \times \frac{L}{C} + \frac{mL}{r} - \frac{L}{C} \tag{20}$$

where m is the number of non-zero binary coefficients of r.

Proof: Suppose ps provides bounded delay d for flow f, d is defined in Definition 1. We first show the relationship between ideal serve curve and real service curve under packet scheduler ps.

For f with reserved rate r, its ideal service curve is $S^{id}(t-t_0) = r(t-t_0)\,(t \ge t_0)$. The service curve under ps is denoted as $S^{ps}(t-t_0)\,(t \ge t_0)$. Since ps provides bounded delay, we have

$$S^{ps}(t-t_0) \ge S^{id}(t-t_0-\tau) \tag{21}$$

for $t > t_0$. From Fig. 7, we can see that $S^{ps}(t-t_0)$ is bounded by right shifting $S^{id}(t-t_0)$ along the x-axis τ unit of time.

We have the following inequalities for d and τ.

$$d \le \tau \tag{22}$$

and

$$\tau \le d + \frac{L}{r} - \frac{L}{C}. \tag{23}$$

The correctness of (22) is apparent. The correctness of (23) can be proved by contradiction: Suppose $\tau > d + \frac{L}{r} - \frac{L}{C}$ for the ith packet of f. Then the finish time of the $(i+1)$th packet $t_{i+1} > \frac{iL}{r} + d + \frac{L}{r} = \frac{(i+1)L}{r} + d$. We therefore have $d_{i+1} > d$, which results in contradiction.

Since the rate of flow f is $r = w = \sum_{l=1}^{m} 2^{n_l}$ $(0 \le n_1 < n_2, \cdots, n_m \le k-1)$, flow f can be viewed as the composition of m separate flows $\{f_1, f_2, \cdots, f_m\}$ with rates $\{2^{n_1}, 2^{n_2}, \cdots, 2^{n_m}\}$.

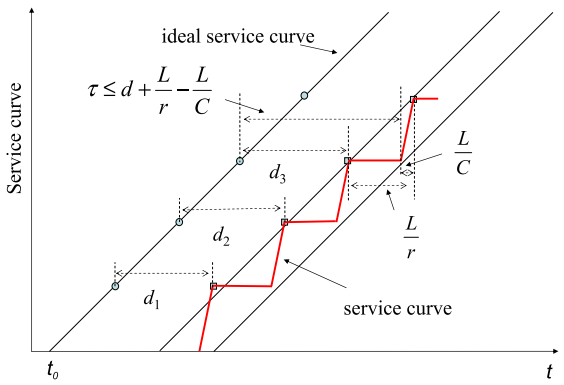

Fig. 7. Service curves and their relationship.

Based on Lemma 6, for flow f_i, its service curve $S_i(t-t_0)$ meets the following inequality:

$$S_i(t-t_0) \ge S_i^{id}(t-t_0-d_i-\frac{L}{2^{n_i}}+\frac{L}{C}).$$

The service curve of f is the sum of the service curves of the m flows. Therefore

$$\begin{aligned} S^{g_3}(t-t_0) &= \textstyle\sum_{i=1}^{m} S_i(t-t_0) \\ &\ge \textstyle\sum_{i=1}^{m}\{2^{n_i}(t-t_0-d_i-\frac{L}{2^{n_i}}+\frac{L}{C})\} \\ &= \textstyle\sum_{i=1}^{m}\{2^{n_i}(t-t_0)\} - \sum_{i=1}^{m}\{2^{n_i}\theta(k-1)\times\frac{L}{C}\} \\ &\quad -mL+\textstyle\sum_{i=1}^{m}\{2^{n_i}\frac{L}{C}\} \\ &= r(t-t_0-\theta(k-1)\tfrac{L}{C}-\tfrac{mL}{r}+\tfrac{L}{C}) \end{aligned}$$

Therefore, using (22), $d^{g_3} \le \tau \le \theta(k-1)\frac{L}{C}+\frac{mL}{r}-\frac{L}{C}$. ■

Though (20) is similar with (11) of RRR. (20) has one important property that (11) does not have: its m value is purely decided by r. Since the delay is mainly contributed by $\frac{mL}{r}$, users can therefore control their delay by choosing appropriate r in G-3. See Section V for simulation results to demonstrate the significant delay differences of RRR and G-3 due to this fact.

Theorem 2 indicates that $S^{g_3}(t-t_0) \ge S^{id}(t-t_0-\tau) = r(t-t_0-\tau)$, G-3 is therefore a latency-rate (LR) server [19]. From [19] and Theorem 2, we have the following corollary:

Corollary 1: When a flow with reserved rate r is constrained by a leaky bucket (σ, r) and served by a network of M cascading G-3 nodes, its end-to-end delay bound is

$$D \le \frac{\sigma}{r} + \sum_{i=1}^{M} d(i) \tag{24}$$

where $d(i)$ denotes the delay bound at node i and is calculated from Theorem 2.

Schedulers that have bounded delay may not have bounded Golestani Fairness index (also called Service Fairness Index, SFI [8]) and Worst-case Fairness Index (WFI) [1]. G-3, however, provides both bounded SFI and WFI indices. See [11] for details. G-3 therefore provides both good fairness and delay properties.

B. Space and Time Complexities and Space-Time Tradeoff

Theorem 3: G-3 needs $O(1)$ time to choose a packet for transmission.

Proof: Since the value of a_{k-1} of the SWM matrix is 1, from the property of WSS, lines 7-10 of the *Schedule* procedure will be executed once in at most 2 loops. Lines 5, 6, and 12 of the procedure can all be executed in one operation. By counting the number of lines, *Schedule* needs at most 9 operations to serve a flow. G-3 therefore provides strict $O(1)$ time complexity for packet scheduling. ■

The time complexity of *Add_flow* is $O(m+k+w_f)$, where m is the number of operations to allocate m nodes for f from the linked lists, k is the maximum number of split operations needed, since we need at most k split operations to generate m unallocated nodes of different weights, and w_f is the number of operations that *update* needs to update the TArrays. Similarly, the time complexity of *Del_flow* is also $O(m+k+w_f)$. Since we have shown in Section III-D that *update* can be performed parallelly with *Schedule*, the time complexities of *Add_flow* and *Del_flow* can therefore be considered as $O(m+k)$, and since $m \le k$, $O(m+k) = O(k)$.

The space needed by G-3 is composed of 5 parts: a 2^k space to store the WSS sequence, a k space for the SWM matrix, a space that is in proportion to the number of flows to store the binary trees, a space to store the k linked lists, and a 2^k space for the at most k TArrays. The space complexity of G-3 is therefore $O(N)$ + $O(2^k)$.

When k is small (e.g. $k \le 24$), the space needed by G-3 is affordable ($2^k \le 16M$ when $k \le 24$). But since 2^k increases exponentially as k increases, the space needed by G-3 may be unacceptable when k is large (e.g., when $k = 32$, 2^{32} will be 4G). In SRR [10], we have proposed a method to trade time for space. A 32th WSS sequence is constructed from a 17th WSS sequence. SRR takes one additional operation for scheduling, but reduces the space needed from 2^{32} to 2^{17}. Similar space-time tradeoff can be used to reduce the space of the TArrays. For example, for a WBT tree of depth 32, we only expand the first 24 levels of the tree into a perfect binary tree and keep the rest 8 low levels as it is. We therefore only need a TArray of size 2^{24} to represent the perfect binary tree, and *Schedule* will take at most 8 additional operations to traverse the rest 8-level binary sub-trees.

V. Simulation

We have implemented G-3, SRR, and RRR in *NS*$_2$ [14]. In the network as depicted in Fig. 8, 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 links. The MTU of the network is 200 bytes and the reserved rate of a flow is set to its sending rate.

We setup a CBR flow f_1 with rate 32 kb/s 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

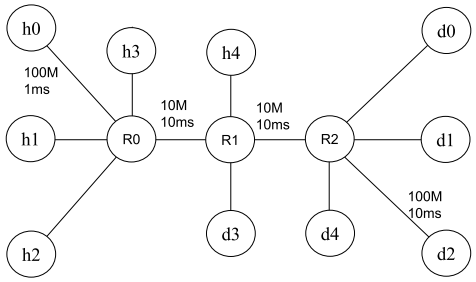

Fig. 8. Network topology of the simulation.

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 measure the end-to-end delays of f_1 and f_2 under G-3, RRR, and SRR, respectively. Fig. 9 shows the results.

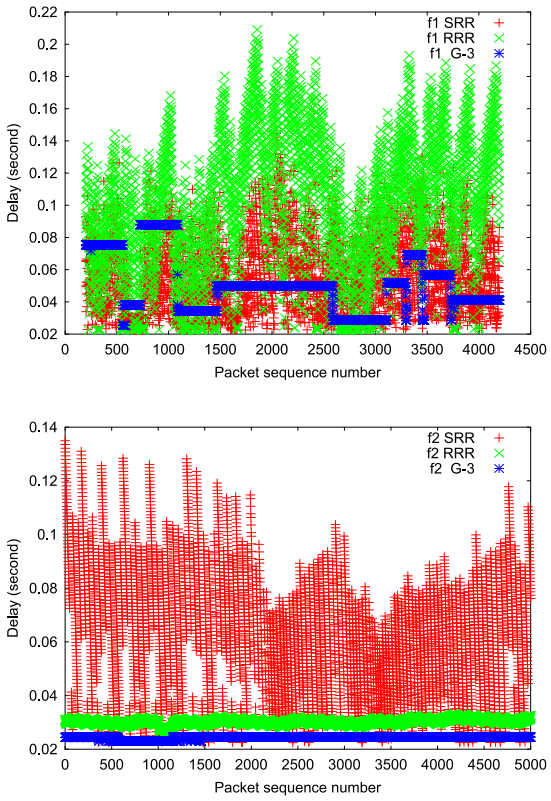

Fig. 9. End-to-end delays of f_1 (32 kb/s) and f_2 (1024 kb/s) under G-3, RRR, and SRR.

Fig. 9 demonstrates that G-3 outperforms RRR and SRR significantly. The maximum end-to-end delays of f_1 and f_2 are 88ms and 25.6ms, which are smaller than the corresponding G-3 upper bounds 122ms and 25.8ms, respectively. The worst-case delays for both f_1 and f_2 are large (both are 144ms) under SRR, and the delay of f_2 is rather large even if its reserved rate is 1024 kb/s. The delays of f_1 and f_2 under RRR are much larger than that under G-3. And the delay of f_1 under RRR is even much larger than that under SRR. This is because the m values of RRR are large ($m = 6$ and 11 for f_1 and f_2, respectively) and that m has large impact on flows with low reserved rate. See [11] for more simulations.

VI. Conclusion

We have presented a G-3 packet scheduling algorithm for fixed size packet networks. G-3 provides both strict $O(1)$ time complexity and bounded end-to-end delay. Both analysis and simulations demonstrate that G-3 provides fair bandwidth allocation and bounded end-to-end delays. G-3 demonstrates that *dynamic* circuits can be constructed efficiently in packet networks with minimal time complexity overhead. Due to its $O(1)$ time complexity, G-3 can be used in high-speed networks, where time complexity is one of the most critical factors for choosing scheduling algorithms, to provide guaranteed bandwidth and bounded delay.

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] J. Cobb, M. Gouda, and A. El-Nahas. Time-shift scheduling-fair scheduling of flows in high-speed networks. *IEEE/ACM trans. Networking*, 6(3), Jun. 1998.

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

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

[8] S.J. Golestani. A self-clocked fair queueing scheme for broadband applications. In *Proc. infocom*, pages 636–646, Apr. 1994.

[9] P. Goyal, H. M. Vin, and H Cheng. Start-time fair queueing: a scheduling algorithm for integrated services packet switching networks. *IEEE/ACM trans. Networking*, 5(4), Aug. 1997.

[10] 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.

[11] 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.

[12] 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.

[13] J. Nagle. On packet switches with infinite storage. *IEEE trans. Communications*, 35(4):435–438, Apr. 1987.

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

[15] 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.

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

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

[18] Dimitrios Stiliadis and Anujan Varma. Efficient fair queueing algorithms for packet-switched networks. *IEEE/ACM trans. Networking*, 6(2):175–185, Apr. 1998.

[19] 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.

[20] Jun Xu and Richard J. Lipton. On fundamental tradeoffs between delay bounds and computational complexity in packet scheduling algorithms. In *Proc. SIGCOMM*, 2002.

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