A Simple Cell Scheduling Mechanism for ATM Networks *

Ming-Chi Lee, Tsung-Chun Chen, Hsin-Tai Yang, and Shie-Jue Lee
Department of Electrical Engineering
National Sun Yat-Sen University
Kaohsiung 804, Taiwan
E-mail: {mclee| nomad| styang| leesj}@water.ee.nsysu.edu.tw

ABSTRACT

Carry-Over Round Robin(CORR) is a simple mechanism for scheduling cells in asynchronous transfer mode networks. It is competitive with much more complex scheduling disciplines in terms of performance, especially in operational simplicity. However, CORR makes some extra design to maintain a maximum frame size, which leads to an unfair allocation of bandwidth and increases complexity of operations. In this paper, we propose a modified scheme to simplify CORR. We provide a mathematical analysis and show that this scheme achieves better performance than CORR.

1 INTRODUCTION

Integrated services networks are required to support a variety of applications with a wide range of Quality of Service(QoS) requirements. At a switch node in high-speed networks, hundreds of thousands of sessions belonging to different services interact with each other and contend for the same output link. Scheduling [1, 2] is a discipline that allocates the sequence in which the packets of different sessions are sent. The design of a traffic scheduling algorithm involves tradeoff among its delay, complexity of implementation, and fairness.

Concerning the delay and fairness, Packet by Packet Generalized Processor Sharing (PGPS) is considered an ideal scheduling algorithm and achieves nearly perfect performance [3]. However, PGPS requires $O(N)$ work per packet, where N is the number of flows that are currently active at the switch node. Several approaches were proposed to reduce the complexity of PGPS [4, 5, 6]. They exhibit desirable performance characteristics compared with PGPS and achieve high network utilization. All of the protocols mentioned above, called Flow Timestamps schedulers, have in common the requirement of transmitting packets in a priority order according to timestamps. Since maintaining a sorted priority queue [7, 8] introduces significant overhead, these algorithms are difficult to implement with a large number of active sessions at high speed.

*This work was supported by the National Science Council under the grant NSC-89-2213-E-110-010.

Another class of category based on frame-based [9] or round-robin [10] usually has $O(1)$ time complexity. Although they perform a little worse than Flow Timestamp schedulers in delay and fairness, these algorithms are simple and efficient in a high-speed network. In general, frame-based schedulers are nonwork-conserving and waste network bandwidth. Round-robin schedulers usually suffer from poor delay performance. Carry-Over Round Robin is essentially a frame-based scheduler. However, it is work-conserving and yields low delay independent of the number of connections.

2 CORR ALGORITHM

CORR is a variation of the frame-based scheduling discipline in ATM networks [11]. It divides the time line into cycles of maximum length T. At the time of admission, each connection C_i is allocated a rate R_i expressed in cells per cycle. Unlike simple round-robin schemes, where R_i has to be an integer, CORR allows R_i to be any real number. Although R_i can be real in CORR, the actual number of slots allocated to a connection in a cycle is still an integer. The debits and credit due to imperfect allocation are carried over into the next cycle.

The CORR scheduler (see Figure 1) consists of three asynchronous events — *Initialize, Enqueue*, and *Dispatch*. The event *Initialize* is invoked when a connection is admitted. If a connection is admissible, it simply adds the connection to the connection list $\{C\}$. Note that the connection list must be in the decreasing order of $R_i - \lfloor R_i \rfloor$, i.e., the fraction part of R_i. Intuitively, arranging connections in a decreasing order of the fractional part of R_i can make sure that the connections with higher fraction requirement are served earlier. Let us walk through an example(as shown in Figure 2) referred to [11]. Consider a system with cycle length T=4 cells serving three connections, C_1, C_2, and C_3, with $R_1 = 2, R_2$=1.5, and R_3=0.5, respectively. For ease of exposition, assume that all the three connections are backlogged starting from the beginning of the system busy period. In the major cycle of the first cycle, CORR allocates C_1, C_2 , and $C_3, \lfloor R_1 \rfloor = 2, \lfloor R_2 \rfloor = 1$, and $\lfloor R_3 \rfloor = 0$ slots, respectively. Hence, at the beginning of the first minor cycle, $r_1 = 0.0, r_2 = 0.5$, and $r_3 = 0.5$. The only slot left for the minor cycle

Constants

T:Cycle length.

R_i:Slots allocated to C_i

Variables

$\{C\}$:Set of all connections

t:Slots left in current cycle

n_i:Number of cells in C_i

r_i:Current slot allocation of C_i

Events

Initialize(C_i);/*Invoked at connection setup time*/
  add C_i to $\{C\}$;
  $n_i \leftarrow 0; r_i \leftarrow 0;$
Enqueue();/*Invoked at cell arrival time*/
  $n_i = n_i + 1;$
  add cell to connection queue;
Dispatch();/*Invoked at the beginning of a system busy period*/
  $\forall C_i : r_i \leftarrow 0;$
  while not end of busy period **do**
   $t \leftarrow T$
   1. Major Cycle:
    for all $C_i \in \{C\}$ **do** /* From head to tail*/
      $r_i \leftarrow \min(n_i, r_i + R_i); x_i \leftarrow \min(t, \lfloor r_i \rfloor);$
      $t \leftarrow t - x_i; r_i \leftarrow r_i - x_i; n_i \leftarrow n_i - x_i;$
      dispatch x_i cells from connection queue C_i;
    end for
   2. Minor cycle:
    for all $C_i \in \{C\}$ **do** /*From head to tail*/
      $x_i \leftarrow \min(t, \lceil r_i \rceil);$
      $t \leftarrow t - x_i; r_i \leftarrow r_i - x_i; n_i \leftarrow n_i - x_i;$
      dispatch x_i cells from connection queue C_i;
    end for
  end while

Figure 1: CORR algorithm.

goes to C_2. Consequently, at the end of the first cycle, $r_1 = 0.0, r_2 = -0.5$, and $r_3 = 0.5$. The adjusted requirements for the second cycle are

$$r_1 = r_1 + R_1 = 0.0 + 2.0 = 2.0$$

$$r_2 = r_2 + R_2 = -0.5 + 1.5 = 1.0$$

$$r_3 = r_3 + R_3 = 0.5 + 0.5 = 1.0$$

Since all the r_i's are integers, they are all satisfied in the major cycle of the second cycle.

The main attraction of CORR is its simplicity, which is regarded as the most important criterion for the selection of an algorithm for use in a real system. Nevertheless, we believe CORR should be simplified due to several reasons as follows.

1. The sorted connection list increases the complexity of implementation. Although CORR still maintains $O(1)$ work per cell, it will probably drive up the cost of routers.

2. A CORR scheduler makes at most two passes through the connection list in each cycle–one during the major cycle and the other during the minor cycle. Thus CORR performs twice the number of operations performed by a round-robin scheduler in the worst case. If we combine the two subcycles into one, or eliminate either of them successfully, the algorithm will be more efficient.

3. CORR allows r_i to be negative. It implies that some connections getting a slot in the minor cycle would exceed their allocated quotas. Hence, the allocation of bandwidth is a little unfair.

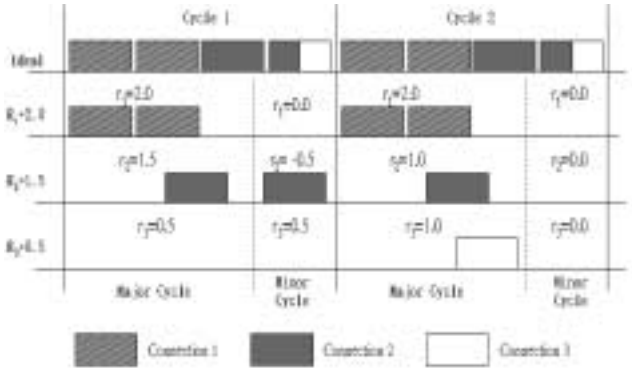

Figure 2: An example of CORR.

In fact, the drawbacks of CORR mentioned above come from maintaining the maximum frame size. In order to derive the worst-case end-to-end delay, a maximum frame length is helpful at the beginning. However, it doesn't mean such a step is necessary. In the next section, we will present a modified scheme to simplify CORR. Also, we will show that the scheme proposed has a similar delay bound and better fairness.

3 THE MODIFIED SCHEME

The minor cycles in CORR allow r_i to be negative. It has been proved that $\Sigma r_i \leq 0$ at the beginning of each cycle. After updating r_i to $r_i + R_i$, the number of cells dispatched during one cycle is smaller than T. Combined with the sorted connection list, CORR guarantees that the bandwidth allocated to each frame never exceeds T. Under this condition, the worst-case delay encountered by the ith cell in a CORR scheduler can be derived. A detailed proof can be found in [12] and [11].

The basic idea of our modifying CORR is to reduce operations. Now, assume we erase the minor cycle of the *Dispatch* event in CORR. Then a much larger frame with a probably infinite length may be generated. Since a larger frame leads to a larger delay, a similar bound after modifying CORR seems difficult to derive. Thus the key to simplify CORR is to derive a delay bound without a maximum frame size.

Let us consider the similar example presented in the previous section. However, at this time there are no minor cycles (as shown in Figure 2). Let cycle length

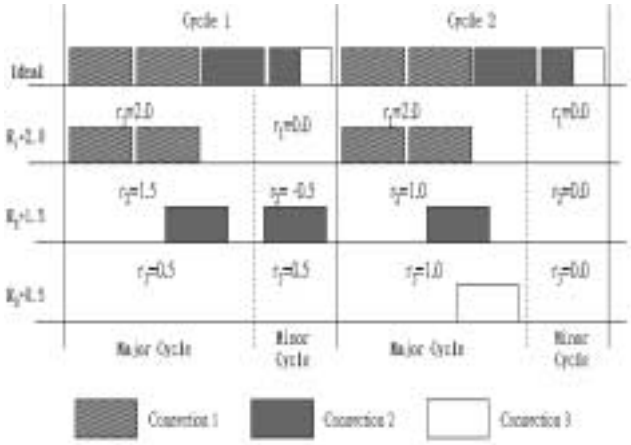

Figure 3: An example of our algorithm.

T=4 and $R_1 = 2, R_2 = 1.5$, and $R_3 = 0.5$. Observe that from time t_0 to t_2, the scheduler dispatches 8 cells during this interval. $r_1 = r_2 = r_3 = 0$ at the beginning of Cycle 1 and the end of Cycle 2. That is almost the same as CORR. Observe that in time interval (t_1, t_2), cells allocated in Cycle 2 are 5. $r_1 = 0.0, r_2 = 0.5$, and $r_3 = 0.5$ at the beginning of Cycle 2. $r_1 = r_2 = r_3 = 0$ at the end of Cycle 2. Cycle 2 is allowed to send one more cell than T. Note that $\Sigma r_i = 1$ at the beginning of Cycle 2. We generalize our observations as follows.

1. If we don't consider the initial values of r_i (or assume all r_i are set to zero), the following expression holds (the situation is shown in Figure 4).

$$\sum_{k=0}^{n-1} C_k \leq \sum_{k=0}^{n-1} T = nT$$

That is the case we observe at t_0.$C_1 = 3$ is smaller than the ideal length T and $C_1 + C_2 = 8$ is equal to the ideal length $2T$.

2. If we consider the initial values of r_i, the above inequality becomes

$$\sum_{k=0}^{n-1} C_k \leq nT + \sum_{i=0}^{N-1} r_i$$

This case is like the observation at t_0. $C_2 = 5 \leq T + \Sigma r_i = 5$.

Now we solve the problem of the maximum frame size. The delay without minor cycles is at most $N \times T$ plus Σr_i. So we can follow the steps used by CORR to analyze the delay performance. It is clear that the operational complexity of CORR can be reduced. The sorted connection list and the minor cycle are not necessary for deriving the delay bound. Thus we propose another scheduling algorithm as shown in Figure 5. The main difference between CORR mechanism and ours is that we erase the minor cycle in the new scheme. Hence each r_i becomes a non-negative number and connection list doesn't have to be sorted anymore. The

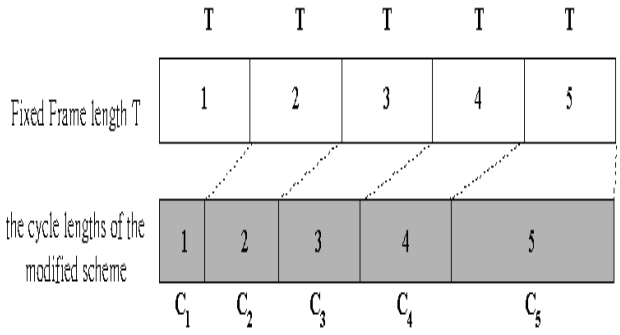

Figure 4: The unfixed cycle length of our scheme.

Constants
R_i:Slots allocated to C_i.
Variables
$\{C\}$:Set of all connections
n_i:Number of cells in C_i
r_i:Current slot allocation of C_i
Events
Initialize(C_i);/*Invoked at connection setup time*/
add C_i to $\{C\}$;
$n_i \leftarrow 0; r_i \leftarrow 0$;
Enqueue();/*Invoked at cell arrival time*/
$n_i = n_i + 1$;
add cell to connection queue;
Dispatch();/*Invoked at the beginning of a system busy period*/
$\forall C_i :: r_i \leftarrow 0$;
while not end of busy period **do**
for all $C_i \in \{C\}$ **do** /* From head to tail*/
$r_i \leftarrow \min(n_i, r_i + R_i)$;
$r_i \leftarrow r_i - \lfloor r_i \rfloor; n_i \leftarrow n_i - \lfloor r_i \rfloor$;
dispatch $\lfloor r_i \rfloor$ cells from connection queue C_i;
end for
end while

Figure 5: The modified algorithm.

number of the basic operations such as +, -, *, / is approximately reduced half as many as CORR. In the next section, we analyze the fairness properties and delay performance of the new scheme. Then we compare with CORR and typically Flow-Timestamp methods. The results show that, albeit its simplicity, this scheme outperforms CORR in terms of fairness. As long as the frame size T close to 1, the performance of delay and fairness is also similar to Flow-Timestamp methods.

4 MATHEMATICAL ANALYSIS

A.Fairness

In a work-conserving server, when the system is not fully loaded, the spare capacity can be used by the busy connections (a connection is busy if its queue is nonempty) to achieve better performance. One of the important performance metrics of a work-conserving server is the fairness. Here the fairess means how fair

the server is in distributing the excess capacity among the active connections. In this section, we analyze the fairness property of our scheduling algorithm and compare with CORR.

Formally, if R_i is the reserved rate of flow i and $send_i(t1,t2)$ is the aggregate service (in cells) received by it in the interval $[t1,t2]$, then an allocation is fair for any intervals $[t1,t2]$ in which both flows i and j are backlogged. That is,

$$\frac{send_i(t_1,t_2)}{R_i} = \frac{send_j(t_1,t_2)}{R_j}$$

This is an idealized definition of fairness as it assumes that flows can be served in infinitesimally divisible units. The objective of fair scheduling algorithms is to ensure that

$$\left|\frac{send_i(t_1,t_2)}{R_i} - \frac{send_j(t_1,t_2)}{R_j}\right|$$

is as close to zero as possible. However, it has been shown that if a scheduling algorithm guarantees that

$$\left|\frac{send_i(t_1,t_2)}{R_i} - \frac{send_j(t_1,t_2)}{R_j}\right| \leq H(i,j)$$

for all intervals $[t1,t2]$, then

$$H(i,j) \geq \frac{l}{2c}(\frac{1}{R_i} + \frac{1}{R_j})$$

where $H(i,j)$ is a function of the property of flows i and j, c denotes the capacity of the output channel, and l denotes the cell length in bits. The function $H(i,j)$ is referred to as fairness measure. Like CORR, we assume that our sampling points coincide with the beginning of the allocation cycles only. Now, when a connection is busy during the cycles $(c1,c2)$, the amount of services received is

$$send_i(c_1,c_2) = \lfloor (c_2-c_1)R_i + \delta_i \rfloor$$

where δ_i is the initial value of the counter at the beginning of cycles. Note that δ_i has to be nonnegative and smaller than 1. In the following, we let

$$W_i(c_1,c_2) = \frac{\lfloor (c_2-c_1)R_i + \delta_i \rfloor}{R_i}$$

Thus,

$$\begin{aligned}
&|W_i(c_2-c_1) - W_j(c_2-c1)| \\
&= \left|\tfrac{\lfloor (c_2-c_1)R_i+\delta_i \rfloor}{R_i} - \tfrac{\lfloor (c_2-c_1)R_j+\delta_j \rfloor}{R_j}\right| \\
&= \left|\tfrac{(c_2-c_1)R_i+\delta_i-x_i}{R_i} - \tfrac{(c_2-c_1)R_j+\delta_j-x_j}{R_j}\right| \\
&= \left|\tfrac{\delta_i-x_i}{R_i} + \tfrac{x_j-\delta_j}{R_j}\right| \quad 0 \leq x_i, x_j < 1
\end{aligned}$$

In the worst case,

$$|W_i(c_2-c_1) - W_j(c_2-c_1)| \leq \frac{1}{R_i} + \frac{1}{R_j}$$

Under the same scenario, it has been proved that in a CORR scheduler the following equation holds at all times.

$$|W_i(c_2-c_1) - W_j(c_2-c_1)| \leq \frac{2}{R_i} + \frac{2}{R_j}$$

Clearly, the fairness measure of CORR is within two times that of our scheme.

B. Delay in single node

The delay suffered by any cell in a scheduler is the difference between its arrival time at and departure time from the system. The arrival time of a cell can be obtained from the traffic envelope associated with the connection it belongs to and is determined by the shaping mechanism used. In the rest of the section, we derive delay bound for arrival function characterized by the composite leaky bucket.

Let's consider a composite leaky bucket consisting of n component leaky bucket $(b_i,\lambda_i), i = 1,2,3...,n$. It has been proven in [13] that the departure time of the ith cell from the composite shaper, denoted by $a(i)$, is

$$a(i) = \textstyle\sum_{k=1}^{n+1}(i-b_k+1)\lambda_k[U(i-B_k) - U(i-B_{k-1})], \quad i = 0,1,...,\infty$$

where

$$B_k = \begin{cases} \infty & k = 0 \\ \lfloor \frac{b_k\lambda_k - b_{k+1}\lambda_{k+1}}{\lambda_k - \lambda_{k+1}} \rfloor & k = 1,2,...,n \\ 0 & k = n+1 \end{cases}$$

and

$$U(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$$

We regard $a(i)$ as the arrival time of the ith cell at the network entry point. Now we have to derive the departure time of the ith cell. Assume a connection enters a busy period at $time = 0$. Let $d(i)$ be the latest time by which the ith cell departs the system. It's difficult to know the exact time when the cells exit. To capture the worst case, we assume that all cells served during a cycle leave at the end of cycle. Now, if cell i departs at the end of the Lth cycle from the beginning of the connection busy period, the number of slots allocated by the scheduler is $L \times R + \delta$, and the number of slots consumed is $i+1$(assuming packet number starts from 0). As a result,

$$L \times R + \delta - i \geq 1$$

Since δ is between 0 and 1, in the worst case, $\delta = 0$.Thus we get

$$L \geq \frac{1+i}{R}$$

From the above inequality and noting that L is an integer and $d(i) = L \times T$, we get

$$d(i) = \frac{l}{c}\left\lceil \frac{1+i}{R} \right\rceil T + \frac{l}{c}\sum_{k=0}^{N-1} r_k$$

Now we can compute the delay bound from the difference between $d(i)$ and $a(i)$. It has been proven in [11] that the maximum delay encountered by any cell is at most as large as the delay suffered by cell B_j (with composite leaky bucket) and $a(B_j)$ can be derived as follows.

$$a(B_j) = (B_j - b_j + 1)t_j \text{ when } \frac{1}{t_j} < \frac{R}{T} < \frac{1}{t_{j+1}}$$

Therefore, the delay bound is

$$Delay \le d(B_j) - a(B_j) = \left\lceil \frac{B_j+1}{R} \right\rceil T - (B_j - b_j + 1)t_j + \sum_{k=0}^{N-1} r_k \text{ when } \frac{1}{t_j} < \frac{R}{T} < \frac{1}{t_j+1}$$

C. Delay in multi-nodes

Consider a connection traversing n switching nodes between source and destination. Let's denote by $a_k(i)$ the arrival time of the ith cell of the connection at node k. The service time of the ith cell at node k is denoted by $s_k(i)$. The service time of cells $p-q$ at node k is $S_k(p,q)$. Assume the propagation delay between nodes is zero. Then the departure time of cell i from node k is $a_{k+1}(i)$. A theorem in [12] is descirbed as follows.

Theorem: For any node k and for any cell i, the following holds:

$$a_k(i) = \max_{1\le j\le i}\{a_1(j) + \max_{j=l_1\le l_2\le\ldots\le l_k=i}(\sum_{h=1}^{n} S_h(l_h, l_{h+1}))\}$$

where n is the number of switch nodes between the source and the destination.

The theorem presented above is very broad in the sense that it does not assume any specific arrival pattern of scheduling discipline. It means that if the ith cell never encounters any queuing in the system, its departure time can be computed as the sum of its arrival time into the system and its service time at different nodes. If the ith cell encounters queuing in the system, its departure time may potentially depend on the arrival time of any cell that entered the system before it. A detailed description of the theorem can be found in [11]. However, computing exact service time of different cells at each node is quite difficult. Fortunately, computing the worst case bound on departure time and the worst case delay for specific service discipline and arrival patterns is not that difficult. Thus we can replace the service time $S_h(.)$ at nodes $h = 1, 2, \ldots n$ with the worst case service time $S_w(.)$ in the expression for $a_k(i)$ in the above theorem. That becomes

$$a_k(i) = \max_{1\le j\le i}\{a_1(j) + \max_{j=l_1\le l_2\le\ldots\le l_k=i}(\sum_{h=1}^{n} S_w(l_h, l_{h+1}))\}$$

Recall that we have computed the worst case service time $d(i)$. Now we can derive the end-to-end delay bound by replacing

$$S_w(l_h, l_{h+1}) = \frac{l}{c}\left\lceil \frac{(l_{h+1} - l_h) + 1}{R} \right\rceil T + \frac{l}{c}\sum_{k=0}^{N-1} r_k$$

Also,

$$\begin{aligned}
&\textstyle\sum_{h=1}^{n} S_w(l_h, l_{h+1}) = \frac{l}{c}\sum_{h=1}^{n}\left\lceil \frac{(l_{h+1}-l_h)+1}{R} \right\rceil T \\
&\quad + \textstyle\frac{l}{c}\sum_{h=1}^{n}\sum_{k=1}^{N_h} r_k^h \\
&\le \textstyle\frac{l}{c}\sum_{h=1}^{n}\left[\frac{(l_{h+1}-l_h)+1}{R} + 1\right] T + \frac{l}{c}\sum_{h=1}^{n}\sum_{k=1}^{N_h} r_k^h \\
&\le \textstyle\frac{1}{c}\left[n + \frac{n-1}{R}\right] T + \frac{l}{c}\left[\frac{l_{n+1}-l_1+1}{R}\right] T + \frac{l}{c}\sum_{k=1}^{N_n} r_k^n \\
&\quad + \textstyle\frac{l}{c}\sum_{h=1}^{n-1}\sum_{k=1}^{N_h} r_k^h \\
&\le \textstyle\frac{1}{c}\left[n + \frac{n-1}{R}\right] T + S_w(l_1, l_{n+1}) + \frac{l}{c}\sum_{h=1}^{n-1}\sum_{k=1}^{N_h} r_k^h \\
&\le \textstyle\frac{1}{c}\left[n + \frac{n-1}{R}\right] T + S_w(j,i) + \frac{l}{c}\sum_{h=1}^{n-1}\sum_{k=1}^{N_h} r_k^h \\
&\quad (l_1 = j \text{ and } l_{n+1} = i)
\end{aligned}$$

Hence, the end to end delay bound is $ak(i) - a1(i)$.

$$Delay(i) \le \frac{l}{c}\left[n + \frac{n-1}{R_w}\right] T + \max_{1\le j\le i}\{a_1(j) + S_w(j,i)\} + \frac{l}{c}\sum_{h=1}^{n-1}\sum_{k=1}^{N_h} r_k^h - a_1(i)$$

D. Comparison with CORR

Now we compare the performance with CORR(as shown in Table 1). As we expect, the fairness of our scheme is better than that of CORR since the counter value r_i is limited to be nonnegative. The worst case complexity is still $O(1)$, the same as CORR. However, CORR has two subcycles in a frame. Thus CORR has nearly as twice the number of operations as our scheme. The worst case end to end delay of these two schedulers seems similar. So we assume all $r_k^h = 1$ and each node has the same number of connections N. Therefore, the difference in maximum delay that a cell may incur employing CORR and our scheme is

$$\begin{aligned}
&D^{CORR}(i) - D^{OURS}(i) \\
&= \textstyle\frac{l}{c}(\frac{n-1}{R_w}T - \sum_{h=1}^{n-1}\sum_{k=0}^{N-1} r_k^h) \\
&= \textstyle\frac{l}{c}(\frac{n-1}{R_w}T - N(n-1)) \\
&= \textstyle\frac{l}{c}(n-1)(\frac{T}{R_w} - N)
\end{aligned}$$

Observe that the maximum delay of cells of a connection in our scheme is smaller than in CORR if

	CORR	Our approach
Fair	$\frac{2}{R_i} + \frac{2}{R_j}$	$\frac{1}{R_i} + \frac{1}{R_j}$
End to End Delay	$\frac{l}{c}\left[n + 2\frac{n-1}{R_w}\right]T + \max_{1\le j\le i}\{a_1(j)+ S_w(j,i)\} - a_1(i)$	$\frac{l}{c}\left[n + \frac{n-1}{R_w}\right]T + \max_{1\le j\le i}\{a_1(j)+ S_w(j,i)\} + \frac{l}{c}\sum_{h=1}^{n-1}\sum_{k=1}^{N_h} r_k^h - a_1(i)$
Complexity	$O(1)$	$O(1)$

Table 1: Comparison with CORR.

$R_w < T/N$. In other words, a connection of CORR has a smaller delay if $R_w > T/N$. Since T/N is the average rate of all connections in full load, we can expect that half of connections would have reserved rate smaller than T/N. That is, half of them have a smaller delay than CORR if the system employs our scheme. Thus, the delay performance of our algorithm is not worse than CORR. However, this algorithm has a better fairness and lower operational complexity.

E. Comparison with Flow-Timestamps

As mentioned earlier, the schedulers based on computing timestamps are the best of breed. Although it is hard to implement in high-speed networks, they are still very attractive. The author of CORR has compared with SCFQ and PGPS, two typically flow-timestamp schedulers, in terms of fairness and delay. It seems CORR is competitive with them. However, this comparison is not suitable due to the different traffic models and definitions. For example, CORR uses the begging of allocation cycles $(c1, c2)$ instead of real time$(t1, t2)$ to derive fairness measure. Then CORR compared fairness with SCFQ directly. In fact, the fairness measure should have two properties—long-term fairness and short-term fairness. The long-term fairness implies fair allocation of bandwidth and the short-term fairness implies the size of traffic burst of a connection. Unfortunately, to derive the fairness measure using cycles instead of real time does not have the short-term fairness property. The scheduler may produce a large burst during any cycle even though the distribution of bandwidth is fair at the end of this cycle. This situation may occur in any frame-based but not in flow-timestamps schedulers. However, we can not see such a property in mathematical analysis as we derive fairness measure like CORR. Therefore, we provide another analysis similar to flow-timestamps and compare with them in Table 2.

As expected, flow-timestamps schedulers always perform better except complexity. The larger the frame size is, the worse ‘ the delay and fairness are. Fortunately, our scheduler is designed for asynchronous transfer mode networks. Since the cell length is fixed in ATM networks, the frame size can be reduced to a minimum as small as 48 bytes(one cell length). Then the performance is very similar to any flow-timestamps scheduler as shown in Table 3. However, it is not efficient if the frame size T is 1. The overhead of each pass through the connection list is high. It may lead to a bottleneck in the system as well as sorting in the flow-timestamps schedulers.

5 CONCLUSION

In this paper, we have proposed a modified scheme of CORR algorithm, which is a variation of frame-based scheduling discipline in ATM networks. We also have shown that our algorithm improves the fairness of CORR and reduces the operational complexity. In addition to CORR, we provided another analysis to compare with flow-timestamps schedulers. The result shows that our scheme performs as well as SCFQ and PGPS if the frame size is limited to one cell length. However, the implementation of ours is much easier. Like the scheduling mechanisms with high complexity of implementation, the frame size limited to 1 may cause a serious bottleneck in high-speed networks. We believe the problem can be solved if the frame size is dynamically adjusted. When the traffic load is light, the smaller frame size limits the burst to be small. When the traffic load is heavy, the larger frame size leads to a lower average delay than SCFQ and PGPS. Although a larger burst is produced in heavy traffic load, our scheduler is still simpler and more efficient than SCFQ and PGPS.

References

[1] S. Iatrou and S. I., “A dynamic regulation and scheduling scheme for real-time traffic management,” *IEEE/ACM Trans. Networking*, vol. 8, pp. 60–70, February 2000.

[2] S. D., Z. H., and J. Bennett, “Implementing scheduling algorithms in high-speed networks,” *IEEE J. Select. Areas Commun.*, vol. 17, pp. 1145 –1158, June 1999.

[3] K. Parekh and R. G. Gallager, “A generalized processor sharing approach to flow control in integrated services network: The single node case,” in *Proc. IEEE INFOCOM'92*, vol. 2, pp. 915–924, May 1992.

[4] P. Goyal, V. H. M., and C. H., “Start-time fair queuing:A scheduling algorithm for integrated services network,” *IEEE/ACM Trans. Networking*, vol. 5, pp. 690–704, 1997.

[5] J. A. Cobb, M. G. Gouda, and A. El-Nahas, “Time-shift scheduling-fair scheduling of flows in

	CORR	Our approach
Fair	$\frac{l}{c}\left(\frac{1}{R_i}+\frac{1}{R_j}\right)$	$\frac{l}{c}\left(2T+\frac{1}{R_i}+\frac{1}{R_j}\right)$
End to End Delay	$\frac{1}{R}\frac{l}{C}+\frac{l}{C}$ PGPS $\frac{1}{R}\frac{l}{C}+(N-1)\frac{l}{C}$ SCFQ	$\left(\frac{1}{R}+1\right)\frac{lT}{C}+(N-1)\frac{l}{C}$
Complexity	$O(\log N)$ or $O(N)$	$O(1)$

Table 2: Comparison with Flow-Timestamps.

	CORR	Our approach
Fair	$\frac{l}{c}\left(\frac{1}{R_i}+\frac{1}{R_j}\right)$	$\frac{l}{c}\left(2+\frac{1}{R_i}+\frac{1}{R_j}\right)$
End to End Delay	$\frac{1}{R}\frac{l}{C}+\frac{l}{C}$ PGPS $\frac{1}{R}\frac{l}{C}+(N-1)\frac{l}{C}$ SCFQ	$\left(\frac{1}{R}+1\right)\frac{l}{C}+(N-1)\frac{l}{C}$
Complexity	$O(\log N)$ or $O(N)$	$O(1)$

Table 3: Comparison with Flow-Timestamps.

high-speed networks," *IEEE/ACM Trans. Networking*, vol. 6, pp. 274 –285, June 1998.

[6] L. Zhang, "Virtualclock: A new traffic control algorithm for packet switching networks," *ACM trans. Comput. Syst.*, vol. 9, pp. 101–124, May 1991.

[7] D. Stiliadis and V. A., "Efficient fair queuing algorithms for packet-switched networks," *IEEE/ACM Trans. Networking*, vol. 6, pp. 175–185, April 1998.

[8] D. Stiliadis and V. A., "Rate-proportional Servers: A design methodology for fair queuing algorithms," *IEEE/ACM Trans. Networking*, vol. 6, April 1998.

[9] G. S. J., "A framing strategy for connection management," in *Proc. SIGCOMM'90*, 1990.

[10] M. Shreedhar and G. Vaghese, "Efficient fair queuing using deficit round robin," *IEEE/ACM Trans. Networking*, vol. 4, pp. 375–385, June 1996.

[11] D. Saha, S. Mukherjee, and S. Tripathi, "Carry-over round robin: A simple cell scheduling mechanism for ATM networks," *IEEE/ACM Trans. Networking*, vol. 6, pp. 779–796, December 1998.

[12] D. Saha, *Supporting distributed multimedia applications on ATM networks*. PhD thesis, Univ. Maryland, 1995.

[13] S. Raghavan and S. Tripathi, *Networked multimedia systems*. Prentice Hall, 1998.