Gated-Scheduling Algorithms in Packet Switching Networks

Fu-Ming Tsou Zsehong Tsai
Rm. 543, Department of Electrical Engineering
National Taiwan University, Taipei, Taiwan.
E-mail: fmtsou@eagle.ee.ntu.edu.tw ztsai@cc.ee.ntu.edu.tw

Abstract— **In this paper, two novel scheduling algorithms with low implementation complexity are investigated. Most scheduling algorithms proposed so far usually involve a sorting operation with the complexity of $O(\log N)$ per packet, where N denotes the number of connections sharing the link. To solve this problem, we propose two new scheduling algorithms with the complexity $O(1)$, implemented with only a single FIFO queue in the output scheduler. The proposed scheduling algorithms make use of the concept of gated scheduling, and thus the schedulers are called *Gated-Scheduling Servers* (GSS). The key contribution of the GSS algorithms is the successful elimination of output sorter in their designs such that the scheduling mechanism can accommodate large number of flows. Both delay bounds and Fairness Index for flows scheduled under these two algorithms are validated with simulations.**

I. Introduction

It is well-known that providing real-time and multimedia communications over the Internet and high speed LAN is a widely accepted trend. Many future Internet applications are believed to be broadband and rely on the ability of the network to provide Quality-of-Service (QoS) guarantees. Conventionally, networks have used First-Come-First-Served (FCFS) queues to do packet scheduling at the switching nodes. However, FCFS can not make certain quantitative commitments regarding the QoS provision since it does not distinguish flows or classes of packets. Therefore, a network supporting multiple classes of services should employ a more advanced scheduling mechanism than FCFS.

Currently, scheduling algorithms may be classified as frame-based or sorted-priority-based[1]. In a frame-based scheduler, time is split into frames of either fixed or variable length. Reservations of flows are made in terms of the maximum amount of traffic at which the flow is allowed to transmit during a frame period. The most famous example is weighted round robin (WRR)[2]. On the other hand, within a sorted-priority scheduler there is always a global variable, usually referred to as the *virtual time*, associated with the outgoing link being scheduled. When a packet arrives, it is assigned a timestamp computed as a function of the virtual time. Packets are then sorted based on their time-stamps, and are transmitted in that order. The most representative examples of the sorted-priority scheduling algorithms are the General Processor Sharing (GPS)[3][4] and its extensions[5][6][7][8]. In general, the frame-based scheduling algorithms are with large delay bounds, but their implementation complexity can be relatively low. On the other hand, the sorted-priority scheduling algorithms are with smaller delay bounds but the implementation complexity is usually high.

In the following, we make a brief comparison of these scheduling algorithms in terms of complexity. In frame-based scheduling algorithms, DRR (deficit Round Robin)[9] and WRR are well-known to achieve fair scheduling of bandwidth with only $O(1)$ complexity. However, it is noted that DRR and WRR have a drawback that the end-to-end delay bound increases with N, where N denotes the number of connections or flows sharing the link[10]. Many variations of WRR try to reduce its scheduling delay. For example, URR (Urgency-based Round Robin)[11] maintains a *urgency-index table* to determine the serving sequence, but the complexity of maintaining the urgency-index table is $O(\log N)$.

In contrast to the frame-based scheduling algorithms, the sorted-priority scheduler usually has smaller scheduling delay but with higher implementation complexity. The complexity of the sorted-priority scheduler comes from two aspects: time-stamp calculation and output sorting operations. For example, the time required for selecting a packet to be transmitted under WFQ is $O(N)$. Recently many other scheduling disciplines have been proposed to try to reduce the computation complexity of the *virtual time function*. These scheduling algorithms include SCFQ (Self-Clocked Fair Queueing)[7], VirtualClock[12], FFQ (Frame-based Fair Queueing)[8], etc. These algorithms successfully reduce complexity to $O(\log N)$, which is the complexity of the output sorter. LFVC (leap forward virtual clock)[13] achieves $O(\log \log N)$ complexity with slight increase of the end-to-end delay and unfairness. QLWFQ (Queue Length Based Weighted Fair Queueing)[14] and BSW (Binary Scheduling Wheels) [15] successfully reduce the complexity to $O(1)$, but their delay performance are sacrified. A detailed comparison of various scheduling algorithms can be seen in [1].

In view of above mentioned research results, via either the frame-based or the sorted-priority approaches, to design a scheduling algorithm with both low complexity and low delay bound is not a trivial task. However, such scheduling mechanisms are especially desired in high-speed next generation networks where different Quality of Services are to be supported. Thus,

This work was supported by National Science Council of the Republic of China under Grant NSC88-2213-E-002-079.

0-7803-5794-9/99/$10.00 © 1999 IEEE

in this paper, we propose a new scheduling discipline, called *Gated-Scheduling Server* (GSS), which does not need output sorting circuits and hence becomes capable of supporting extreme large number of connections. Two versions of GSS algorithms are proposed and both provide tight delay bounds. The organization of the rest of the paper is as follows. In section 2, GSS algorithms are presented. Section 3 describes the simulation results. Finally, conclusions and future work are presented in section 4.

II. Gated Scheduling Server

In this section, two *gated-scheduling server* (GSS) algorithms, *Simple-GSS* (S-GSS) and *Enhanced-GSS* (E-GSS), are proposed.

For the description of S-GSS and E-GSS algorithms, we still make use of the framework established by the Generalized Processor Sharing (GPS)[3][4] and Weighted Fair Queueing (WFQ)[5]. GPS and WFQ are characterized by positive real numbers ϕ_1, ϕ_2, ..., ϕ_n. For any flow i backlogged in the interval $(\tau, t]$ and for another flow j under GPS or WFQ, the following relation holds[3]

$$\frac{S_i(\tau,t)}{S_j(\tau,t)} \geq \frac{\phi_i}{\phi_j}, \quad j = 1, 2, ..., n, \tag{1}$$

where n is the total number of backlogged flows, and $S_i(\tau,t)$ is the serviced workload for flow i during $[\tau,t)$.

In [10] and [16], Stiliadis explain the framework of scheduling algorithms from another point of view, called *potential*. If $P_i(t)$ denotes the potential of flow i at time t, then, during any interval $(\tau,t]$ within a backlogged period for flow i, its potential function is defined as

$$P_i(t) - P_i(\tau) = \frac{W_i(\tau,t)}{\lambda_i}, \tag{2}$$

where λ_i is the guaranteed service rate of flow i, and $W_i(\tau,t)$ is the workload of flow i during $[\tau,t)$. Under such definition, the potential of a flow then reflect its urgency for service capacity. In order to keep track of the progress of the total workload of the scheduler, the *system potential* is some function of $P_1(t-), ..., P_N(t-)$, and t. If the potential of a newly backlogged flow is estimated higher than the potential of the flows currently being served, the former may have to wait for one or more packets to be transmitted from each of the other flows before it can be served. Therefore, the complexity of a scheduling algorithm is determined by the computing procedures of the system potential. In the following, we adopt the concept of system potential in GSS algorithms for its simplicity in notation and easy characterization of the system behaviors.

A. Description of S-GSS Algorithm

Without loss of generality, one could assume flow i becomes active at time $t_i - \Delta$, $\Delta \geq 0$, and the whole system become busy at time 0. Denote $P_i(t_i)$ and $P(t_i)$ as the flow potential of flow i and the system potential at time t_i, respectively. One can easily derive the system potential at at time $t_i + \tau$, $P(t_i + \tau)$, under fluid GPS system as

$$P(t_i + \tau) = P(t_i) + \frac{\tau}{\sum_{j \in \mathcal{B}(t_i)} \lambda_j}, \tag{3}$$

$$P(0) = 0, \tag{4}$$

where λ_j is the reserved rate of flow j. Note that, in (3), the system backlogged set $\mathcal{B}(t_i)$ has to be unchanged during the interval $[t_i, t_i + \tau)$.

The objective of any scheduling algorithm, including the proposed GSS algorithms, is to achieve properties of GPS. Ideally, the following scheduling goal should be achieved:

$$P_i(t_i + \tau) = P(t_i + \tau) = P(t_i) + \frac{\tau}{\sum_{j \in \mathcal{B}(t_i)} \lambda_j}, \tag{5}$$

for any $\tau > 0$.

A key feature of the GSS algorithms is that the system potential at time $t_i + \tau$ is "predicted" at time t_i, using backlogged set $\mathcal{B}(t_i)$, as

$$\hat{P}(t_i + \tau) \triangleq P(t_i) + \frac{\tau}{\sum_{j \in \mathcal{B}(t_i)} \lambda_j}. \tag{6}$$

Therefore, the workload during the interval $[t_i, t_i + \tau)$ can be determined via

$$W_i(t_i, t_i + \tau) = \lambda_i \left[P(t_i) + \frac{\tau}{\sum_{j \in \mathcal{B}(t_i)} \lambda_j} - P_i(t_i) \right]. \tag{7}$$

In order to reduce the implementation complexity, we re-calculate (7) at the starting epoch of fixed time intervals, which are called the *refreshing-periods*. The beginning time points of each refreshing-period are called *refreshing-points*. Therefore, the workload that can be transmitted during the interval $[t_i, t_i + T)$, $W_i(t_i, t_i + T)$, is calculated as follows:

$$W_i(t_i, t_i + T) = \min \left\{ T, \left\lfloor \lambda_i \left(P(t_i) - P_i(t_i) + \frac{T}{\sum_{j \in \mathcal{B}(t_i)} \lambda_j} \right)^+ \right\rfloor \right\}, \tag{8}$$

where $(x)^+$ represents x for $x > 0$ and 0 for $x \leq 0$. The pseudo code of S-GSS algorithm is shown in Fig. 1 and briefly described as follows.

According to the operation required within each time slot, the pseudo code of S-GSS algorithm is designed to include four major parts: the cell-receiving part, the cell-transmitting part, the cell-moving part, and the flow-refreshing part. These four procedures operate in parallel. When a cell arrives, it will be put

in the corresponding flow queue. If a new flow is accepted by the system or an existing flow becomes active from idle at time t, system potential $P(t)$ and the backlogged set $\mathcal{B}(t)$ will be recalculated and updated. The flow refreshing procedure is triggered at the same time. In the flow refreshing part, the system processor computes the workload of the target flow, e.g., flow i, during the interval $[t, t+T)$. The calculated workload W_i will determine the transmission rate that flow queue i adopts to move its cells to the output buffer. The resulting cell moving rate is W_i/T. The system processor will also arrange the next refreshing event which occurs at $t+T$. Each time a cell of flow queue i is moved to the output buffer, the condition that whether the number of cells, being moved to the output buffer, exceeds the workload W_i will be examined. If the condition is true, flow queue i will stop moving cells to the output buffer.

In the cell transmitting part, the output link transmits cells with its wired speed, as long as any cell resides in the output buffer. Once flow i becomes idle, i.e., no cell of flow i presented in the scheduler, system potential P and the backlogged set $\mathcal{B}$ will be recalculated and updated by the system processor.

B. Fairness Index of S-GSS Algorithm

The original definition of *Fairness Index* can be seen in [7]. In this paper, the definition of Fairness Index is extended from that in [7] and re-defined as follows.

Definition 1: Assume two flows i, j are always backlogged after time τ. Denote T as the refreshing period, $W_r(t_1, t_2)$ as the workload of flow r during the interval $[t_1, t_2)$, and λ_r as the normalized service offered to flow r. Then the Fairness Index of the scheduling algorithm is defined as

$$\mathcal{FRI} \triangleq \max_{t_i, t_j} \left| \frac{W_i(t_i, t_i+T)}{\lambda_i T} - \frac{W_j(t_j, t_j+T)}{\lambda_j T} \right|, \quad (9)$$

where t_i and t_j satisfy $\tau \le t_i \le t_j < t_i + \frac{T}{2}$, and, without loss of generality, $\lambda_i \le \lambda_j$.

According to Definition 1, the fairness index of the S-GSS algorithm is derived in the following theorem.

Theorem 1: The Fairness Index of S-GSS algorithm is

$$\mathcal{FRI} = \min\left\{ \frac{1}{\lambda_i}, \frac{2}{\lambda_i+\lambda_j} - 1, \frac{1}{\lambda_i} + \frac{1}{\lambda_i+\lambda_j} + \frac{1}{\lambda_i T} - 2, \frac{3}{\lambda_i+\lambda_j} - 3 + \frac{1}{\lambda_i T} \right\}. \quad (10)$$

The proof of Theorem 1 is described in the Appendix.

C. Enhanced Gated-Scheduling Algorithm

Although the above scheduling algorithm simplifies the implementation complexity, but its worst case delay bound is quite large. In this section, we propose a modified scheduling algorithm which is a little more complex than the original scheduling algorithm but whose delay bound is small.

In the enhanced algorithm, when a flow i is accepted or becomes busy at time t_i, the flow processor not only calculates the workload offered to flow i but also recalculates the workload of the flow which has been granted maximum workload before t_i. Assume that the flow i is accepted or becomes busy at time t_i and that flow m is with maximum workload at t_i^-. Then at time t_i, the workload of flow i, $W_i(t_i, t_i+T)$, is calculated via (8). At the same time, the normalized service rate and the workload of flow m is updated. Although a sorter is needed to choose the flow with maximum workload, but the sorting procedures are required only when a new connection is admitted, rather than performed in each time slot. Hence, the operation speed of the sorter is no longer the bottleneck of the scheduler.

Because the service amount updated via E-GSS algorithm is more accurate than the original algorithm, hence the fairness and the delay bound will be better. However, if we consider the worst case performance, the flow may be not updated by other flows. Therefore, the worst case Fairness Index remains unchanged.

III. Simulation Analysis

In this section, simulation results are presented to illustrate delay and fairness performance of the GSS algorithms. The performance metrics adopted in the experiments include average delay, maximum delay and Fairness Index experienced by individual traffic flows. In this simulation, we assume an ATM network environment, although the GSS algorithms can also be applied to variable-length packet, such as IP.

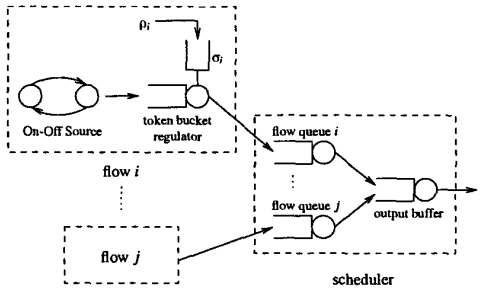

Fig. 2. The token-bucket shaped source model of the simulation experiments.

Fig. 2 shows the token-bucket shaped source model throughout the simulation experiments, where each flow is an ON/OFF source that continuously generates cells at the link rate during an ON-period and generates no cell during an OFF-period. Both the ON and OFF periods of the traffic source follow the Poisson distribution; the mean duration of the ON period of each flow was set to $100\rho_i$ cell times, and the mean OFF duration is $100(1-\rho_i)$ cell times, where ρ_i is the arrival rate of flow i. Before each flow enters the ATM switch, it is shaped through a token-bucket regulator,

```
variable definition:
  P: system potential;
  P_i: flow potential of flow i;
  P_prev: previous updated system potential;
  t_prev: previous system potential updating time;
  B: flow backlogged set;
  T: refreshing period;
  W_i: workload of flow i;
  queue_i: total queue length of flow i in the scheduler;
  t_i: last refreshing-time of flow i;
  r_i: cell moving rate of flow i to the output buffer;
calculate_system_potential(t)
{
  P ← P_prev + (t − t_prev) / Σ_{r∈B} λ_r;
  P_prev ← P;
  t_prev ← t;
  keep P_prev and t_prev;
}
main()
{
  while (1){
    t = system time;
    if (cell_arrived==TRUE){
      p = arrived cell;
      receive_cell(p,t);
    }
    if (output_buffer!=EMPTY){
      p = HOL cell of output buffer;
      transmit_cell(p,t);
    }
    while (cell_moving_event_occurs==TRUE){
      i = operated flow;
      move_cell_output_buffer(i,t);
    }
    while (flow_refreshing_event_occurs==TRUE){
      i = refreshed flow;
      refresh_flow(i,t);
    }
  }
}
receive_cell(p,t)
{
  i ← classify(p);
  /* determine to which flow cell p belongs */
  if (queue_i == 0){ /* if flow i is idle */
    calculate_system_potential(t);
    P_i ← P;
    update backlog set B;
    refresh_flow(i,t);
  }
  queue_i ← queue_i + 1;
  put cell p into flow_queue i;
}
move_cell_to_output_buffer(i,t)
{
  p = HOL cell of flow_queue i;
  extract p from flow_queue i;
  put p into output buffer;
  W_i ← W_i − 1;
  if (W_i ≠ 0)
    schedule_next_moving_event(i, ⌈t + 1/r_i⌉);
    /* schedule next moving event of flow i */
}
transmit_cell(p,t)
{
  i ← classify(p);
  /* determine to which flow cell p belongs */
  queue_i ← queue_i − 1;
  extract cell p from output buffer and transmit it;
  if (queue_i == 0){ /* if flow i is idle */
    calculate_system_potential(t);
    W_i ← 0; /* reset workload of flow i */
    cancel_next_refreshing_event(i);
    /* cancel the next refreshing event of flow i */
    update backlog set B;
  }
}
refresh_flow(i,t)
{
  update backlog set B;
  calculate_system_potential(t);
  W_i ← min{ T, ⌊λ_i · (P + T/Σ_{r∈B} λ_r − P_i)^+ ⌋ };
  P_i ← P_i + W_i/λ_i;
  r_i = W_i/T;
  keep r_i, P and P_i;
  schedule_next_cell_moving_event(i, t + 1);
  /* schedule the next cell moving event of flow i
     which occurs at time t + 1 */
  schedule_next_refreshing_event(i, t + T);
  /* schedule the next refreshing event of flow i */
}
```

Fig. 1. The pseudo code of S-GSS algorithm.

where the token generation rate is the average cell arrival rate of the flow. Since our interest is in evaluating the delay in the scheduler rather than the effect of the input burstiness, the bucket depth, σ_i, is selected as 2 for each flow.

In the simulation experiment, 25 flows share the same outgoing link. In this simulation experiment, the refreshing periods of two GSS algorithms are set as 20 slot times, and the total simulation time is 4×10^6 slot times. In order to simplify the presentation of simulation results, 25 flows are classified into five groups, where two of them, 4 flows, are misbehaving groups, attempting to transmit more than their reservations. Detailed information of each group is shown in Table I. Fig. 3 shows the average delay with 99% confidence intervals of four scheduling algorithms. In this scenario, it is observed that WFQ still has the smallest scheduling delay. This is achieved by the cost of high implementation complexity. We also can observe that average delays of two GSS algorithms are similar to that of SCFQ. The maximum scheduling delays of four scheduling algorithms experienced by each flow are shown in Fig. 4. From Fig. 4, we can observe again that SCFQ and two GSS algorithms all have experienced almost similar maximum delays.

In order to examine the carried workload in details, we present the average and 99% confidence intervals of the normalized workloads of four misbehaving flows in Table II, where the normalized workload is defined as $\frac{W_i(t_i, t_i+T)}{\lambda_i T}$. We also list the Fairness Index for WFQ, SCFQ, S-GSS and E-GSS. It is easy to verify that the normalized workload of fluid GPS[3] corresponding to this scenario is 1.0167, which means each misbehaving flow can obtain 1.0167 times its reservation in practice. From Tab. II, we could observe that the fairness achieved by our two GSS algorithms is close to those

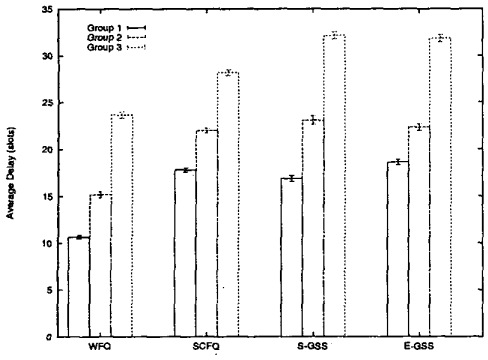

Fig. 3. Average scheduling delay with 99% confidence intervals of four scheduling algorithms.

of WFQ and SCFQ.

The simulation results of delay and fairness verify the point we make in the beginning of the paper that GSS algorithms achieve similar, even better, performance to SCFQ with much lower implementation complexity. This property makes the GSS algorithms capable of accommodating large number of flows in very high speed networks.

Group ID	Flow ID	Reserved Bandwidth	Arrival Rate
0	0 – 1	0.05000	0.1000
1	2 – 6	0.05000	0.0498
2	7 – 8	0.04000	0.1000
3	9 – 16	0.04000	0.0398
4	17 – 24	0.03125	0.0312

TABLE I
DETAILED FLOW INFORMATION.

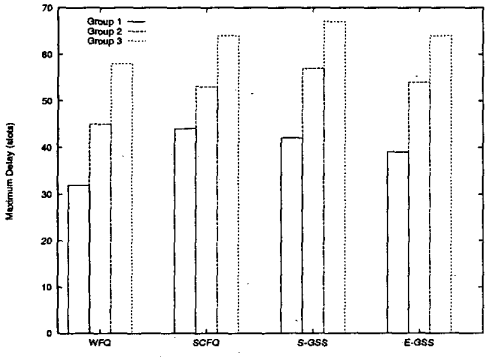

Fig. 4. Maximum scheduling delay experienced by three groups of flows.

IV. Conclusions

In this paper we first propose the S-GSS algorithm. The S-GSS algorithm does not need any sorter, therefore, the output buffer is sufficient to be implemented with a simple FIFO queue. Hence, different QoS requirements, even with large number of connections, can be supported by the S-GSS algorithm. Then, we show that the fairness is achieved and delay is bounded. Because, under some special arrival pattern, the maximal delay of S-GSS is very large, we modify the S-GSS algorithm and propose E-GSS algorithm to provide smaller maximum delay. Although a sorter is needed to choose the flow with maximum workload, but the sorting procedures are required only when a new connection is admitted, rather than performed in each time slot. Hence, the operation speed of the sorter is no longer the bottleneck of the scheduler. In the simulation, the performance of two GSS algorithms are examined by applying the token-bucket shaped ON-OFF traffic sources. The results are compared with WFQ and SCFQ. The simulation results show that though two GSS algorithms perform slightly worse than WFQ, but they are comparable to SCFQ. Hence, GSS algorithms are able to achieve similar performances to SCFQ via much lower implementation complexity.

In summary, the key contribution of the S-GSS and E-GSS algorithms is the elimination of output sorter in their designs while the scheduling mechanism can still accommodate large number of flows. Although the current model in the simulation assumes ATM as the transport technology, it is still possible to achieve similar performance and large switching capacity with other protocols, such as IP. Only a small part of these GSS algorithms, namely the *move* procedure in their pseudo codes, needs to be modified.

We believe there are other scheduling approaches that are capable of handling large number of flows, without involving sorting procedures. But factors such as the FIFO constraint, Fairness Index, and delay performance requirements will still affect the design direction of scheduling algorithms. And this should open an interesting area for further investigation.

References

[1] A. Varma and D. Stiliadis, "Hardware Implementation of Fair Queueing Algorithms for Asynchronous Transfer Mode Networks," *IEEE Commun. Magazine*, vol. 35, no. 12, pp. 54-68, Dec. 1997.

[2] M. Katevenis, S. Sidiropoulos, and C. Courcoubetis, "Weighted Round-Robin Cell Multiplexing in a General-Purpose ATM Switch Chip," *IEEE J. Select. Areas Commun.*, vol. 9, no. 8, pp. 1265-1279, Oct. 1991.

[3] A. K. Parekh and R. G. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single Node Case," *IEEE/ACM Trans. Networking*, vol. 1, no. 3, pp. 344-357, Jun. 1993.

[4] A. K. Parekh and R. G. Gallager, "A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Multiple Node Case," *IEEE/ACM Trans. Networking*, vol. 2, no. 2, pp. 137-150, Apr. 1994.

[5] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm," *Proc. ACM SIGCOMM'89*, pp. 1-12, 1989.

[6] J.C.R.Bennett and H. Zhang, "WF^{2}Q: Worst-Case Fair

Flow ID	WFQ		SCFQ		S-GSS		E-GSS	
	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
0	1.064 ± 0.008	3.00	1.082 ± 0.009	2.00	1.096 ± 0.023	3.00	1.096 ± 0.015	3.00
1	0.994 ± 0.007	2.00	0.978 ± 0.010	2.00	1.096 ± 0.016	3.00	1.096 ± 0.016	3.00
7	1.064 ± 0.012	5.00	1.082 ± 0.012	2.50	1.097 ± 0.024	3.75	1.097 ± 0.019	3.75
8	1.064 ± 0.012	3.75	1.082 ± 0.012	2.50	1.096 ± 0.026	3.75	1.096 ± 0.019	2.50
$\mathcal{FRI}$	5.00		2.50		3.75		3.75	

TABLE II

NORMALIZED WORKLOAD OF FOUR MISBEHAVING FLOWS.

weighted Fair Queueing," *Proc. IEEE INFOCOM'96*, San Francisco, USA, pp. 120-128, Apr. 1996.

[7] S.J.Golestani, "A Self-Clocked Fair Queueing Scheme for Broadband Applications," *Proc. IEEE INFOCOM'94*, Toronto, Canada, pp. 636-646, Jun. 1994.

[8] D.Stiliadis and A. Varma, "Efficient Fair Queueing Algorithms for Packet-Switched Networks," *IEEE/ACM Trans. Networking*, vol. 6, no. 2, pp. 175-185, April 1998.

[9] M. Shreedhar and G. Varghese, "Efficient Fair Queueing Using Deficit Round Robin," *IEEE/ACM Trans. Networking*, vol. 4, no. 3, pp. 375-385, Jun. 1996.

[10] D.Stiliadis, "Traffic Scheduling in Packet-Switched Networks: Analysis, Design, and Implementation," Ph.D dissertation of Computer Engineering, University of California, Santa Cruz, ftp://ftp.cse.ucsc.edu/pub/hsnlab/dimitrios_dissertation.ps.Z, June 1996.

[11] O. Altintas, Y. Atsumi, and T. Yoshida, "Urgency-Based Round Robin: A New Scheduling Discipline for Multiservice Packet Switching Networks," *IEICE Trans. Commun.*, vol. E81-B, no. 11, pp. 2013-2021, Nov. 1998.

[12] L. Zhang, "VirtualClock: A New Traffic Control Algorithm for Packet Switching Networks," *Proc. ACM SIGCOMM'90*, pp. 19-29, 1990.

[13] S. Suri, G. Varghese, and G. Chandranmenon, "Leap Forward Virtual Clock: A New Fair Queueing Scheme with Guaranteed Delays and Throughput Fairness," *Proc. IEEE INFOCOM'97*, 1997.

[14] Y. Ohba, "QLWFQ: A Queue Length Based Weighted Fair Queueing Algorithm in ATM Networks," *Proc. IEEE INFOCOM'97*, Kobe, Japan, pp. 567-576, Apr. 1997.

[15] Y. Chen and J. S. Turner, "Design of a Weighted Fair Queueing Cell Scheduler for ATM Networks," *Proc. IEEE GLOBECOM'98*, 1998.

[16] D.Stiliadis and A. Varma, "Rate-Proportional Servers: A Design Methodology for Fair Queueing Algorithms," *IEEE/ACM Trans. Networking*, vol. 6, no. 2, pp. 164-174, April 1998.

APPENDIX

I. PROOF OF THEOREM 1

According to the definition, the flow potential of flow i at time t_i is

$$P_i(t_i) = \hat{P}(t_i) = P(t_i - T) + \frac{T}{\sum_{r \in \mathcal{B}(t_i - T)} \lambda_r}. \quad (11)$$

But the real system potential is

$$P(t_i) = P(t_i - T) + P(T|t_i - T), \quad (12)$$

where $P(T|t_i - T)$ represents the increment of the system potential during the interval $[t_i - T, t_i)$. Hence, we have

$$P(t_i) - P_i(t_i) = P(T|t_i - T) - \frac{T}{\sum_{r \in \mathcal{B}(t_i - T)} \lambda_r}. \quad (13)$$

Because flow i and flow j are both always backlogged, the following relation holds:

$$\lambda_i + \lambda_j \le \sum_{k \in \mathcal{B}(t)} \lambda_k \le 1, \quad t \ge \tau, \quad (14)$$

where the whole system become busy at time τ. Therefore, we have

$$T \le P(T|t_i - T) \le \frac{T}{\lambda_i + \lambda_j}, \quad (15)$$

and

$$T \le \frac{T}{\sum_{r \in \mathcal{B}(t_i - T)} \lambda_r} \le \frac{T}{\lambda_i + \lambda_j}. \quad (16)$$

Hence, the following inequality can be obtained

$$2T - \frac{T}{\lambda_i + \lambda_j} \le P(t_i + T) - P_i(t_i + T) \le \frac{T}{\lambda_i + \lambda_j} - T. \quad (17)$$

Combining with (8), we have

$$\begin{aligned} &\max\left\{0, 2\lambda_i T - \frac{\lambda_i T}{\lambda_i + \lambda_j} - 1\right\} \\ \le\ & W_i(t_i, t_i + T) \le \min\left\{T, \frac{2\lambda_i T}{\lambda_i + \lambda_j} - \lambda_i T\right\} \end{aligned} \quad (18)$$

Then, one could derive

$$\begin{aligned} &\max\left\{\frac{-1}{\lambda_j}, 1 - \frac{2}{\lambda_i + \lambda_j}, 2 - \frac{1}{\lambda_j} - \frac{1}{\lambda_i + \lambda_j} - \frac{1}{\lambda_i T}, \right. \\ &\left. 3 - \frac{3}{\lambda_i + \lambda_j} - \frac{1}{\lambda_i T}\right\} \le \frac{W_i(t_i, t_i + T)}{\lambda_i T} - \frac{W_j(t_j, t_j + T)}{\lambda_j T} \le \\ &\min\left\{\frac{1}{\lambda_i}, \frac{2}{\lambda_i + \lambda_j} - 1, \frac{1}{\lambda_i} + \frac{1}{\lambda_i + \lambda_j} + \frac{1}{\lambda_j T} - 2, \right. \\ &\left. \frac{3}{\lambda_i + \lambda_j} - 3 + \frac{1}{\lambda_j T}\right\} \end{aligned}$$

Because $r_i \le r_j$, one can easily simplify (19) as

$$\begin{aligned} &\left|\frac{W_i(t_i, t_i + T)}{\lambda_i T} - \frac{W_j(t_j, t_j + T)}{\lambda_j T}\right| \\ \le\ & \min\left\{\frac{1}{\lambda_i}, \frac{2}{\lambda_i + \lambda_j} - 1, \frac{1}{\lambda_i} + \frac{1}{\lambda_i + \lambda_j} + \frac{1}{\lambda_i T} - 2, \right. \\ &\left. \frac{3}{\lambda_i + \lambda_j} - 3 + \frac{1}{\lambda_i T}\right\} \end{aligned} \quad (19)$$

which leads to eq. (10). **Q.E.D.**