Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information Processing Letters 97 (2006) 238–243
Information Processing Letters
www.elsevier.com/locate/ipl

Bandwidth allocation algorithms for weighted maximum rate constrained link sharing policy

Jeng Farn Lee [a,b,*], Meng Chang Chen [a], Ming Tat Ko [a], Wanjiun Liao [b]

[a] *Institute of Information Science, Academia Sinica, Taiwan*
[b] *Department of Electrical Engineering, National Taiwan University, Taiwan*

Received 3 May 2005; received in revised form 28 September 2005; accepted 9 November 2005

Available online 15 December 2005

Communicated by A.A. Bertossi

Abstract

This paper addresses the problem of bandwidth allocation under the weighted maximum rate constrained link sharing policy and proves a key theory in the condition of allocation termination. We propose several algorithms with various worst-case and average-case time complexities, and evaluate their computation elapse times.

Keywords: Scheduling; Bandwidth allocation; Weighted scheduling; Maximum rate constraint; Network management

1. Introduction

In a resource sharing environment, when a resource is limited or scarce, resource managers distribute the resource to meet the requirements or business models, according to a given policy. It is common and useful to distribute resource to users proportionally to their assigned weights. Accordingly, we can determine the shared resources, or provide guaranteed resources to users by adjusting their assigned weights. In this paper, the resource sharing environment is a computer network, and the resource is the shared link capacity. Note that this resource sharing policy can be applied to other domains, such as transactional scheduling, process scheduling, and multimedia processing.

Many critical Internet applications have urgent performance requirements in terms of throughput, delay, delay jitter and loss rate, or a combination of the four. A number of service disciplines have endeavored to provide per-session (or per-user) performance guarantees by using the weighted fair queuing approach [2,4,5,7]. The allocated bandwidths of sessions in weighted fair queueing are calculated according to their weights.

Maximum rate constraint (MRC) is an important management policy for service providers and many applications in which the maximum resource allocated to particular users are limited. Here are examples: (1) when resource is scarce, e.g., a wireless medium, or a shared link from an office, building, or campus to the Internet; (2) when there is a specific management policy, e.g., a restriction on throughput of certain traffics types, such as Web browsing, to a maximum rate; (3) to support a business model, e.g., rate constraints to support a pricing model; and (4) in multimedia stream-

* Corresponding author.
E-mail address: kunimi@iis.sinica.edu.tw (J.F. Lee).

0020-0190/$ – see front matter
doi:10.1016/j.ipl.2005.11.006

ing, especially when there are no feedback mechanisms from receivers. MRC is used in edge routers to shape outgoing and ingoing traffic into the desired traffic patterns and to provide QoS according to the Service Level Agreements (SLAs). Additionally, it can be used by local carrier in their Internet Data Center (IDC). An IDC accommodates hundreds of servers attached to the carrier's backbone network via a number of high speed LAN. Each server is owned by one customer, who chooses the connection speed. For instance, a big server may need a 100 Mbps connection, while a small one only needs 64 kbps. But now as they are connected via high speed LANs, it is critical for the carrier to regulate their traffic so that the servers cannot use more bandwidth than what they subscribe. Another situation is also popular that users in a building/campus sharing a link to the Internet require the MRC mechanism to protect normal usage from excessive and/or abusive usage of greedy users. Moreover, MRC can be used in both Diffserv and Inserv networks, depending on the definition of session to be per service class or per flow. When weighted MRC is applied, the allocated bandwidth of a session by a weighted fair queuing approach cannot exceed its assigned maximum rate; therefore, the excess bandwidth is redistributed to other sessions in proportion to their weights. However, during the allocation process, the allocated bandwidths of sessions may exceed their assigned maximum rates after receiving the excess bandwidths such that the excess bandwidth must be redistributed, and the process repeated.

In this paper, we define the weighted MRC resource sharing policy, and also define a variable, called saturation index, to calculate the allocated bandwidths more efficiently. We then provide several algorithms with different time complexities and conduct simulations to find the best-practice algorithm.

The rest of this paper is organized as follows. In Section 2, we review the max–min fairness, and the rate controller for maximum rate constraint. In Section 3, we define the models of weighted MRC fair queueing. In Section 4 we define an important variable, called saturation index, and a useful theorem. In Section 5, we describe the proposed algorithms. In Section 6, we show the implementation results of the proposed algorithms. Finally, in Section 7, we present our conclusions.

2. Related works

In this section, we review two classic bandwidth sharing principles for data networks, namely, max–min and weighted fairness.

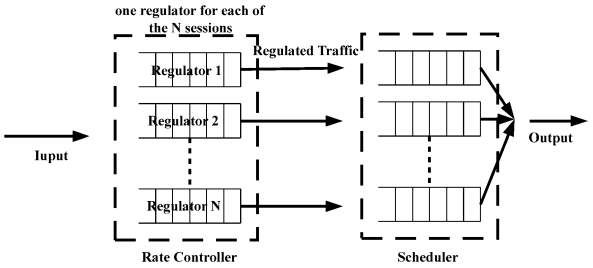

Fig. 1. Two-stage rate-control service discipline.

2.1. Max–min fairness

Max–min fair allocation [1] maximizes the minimum share of a session whose demand is not fully satisfied. It maximizes the bandwidth allocation of each session i subject to the constraint that an incremental increase in allocation of the bandwidth cannot lose feasibility or reduce the bandwidth of another session with a bandwidth equal to or smaller than session i. For example, if four sessions each require bandwidths 2, 2.6, 4, and 5, respectively, but the total capacity is 10, the allocated bandwidths of the four sessions would be 2, 2.6, 2.7, and 2.7, respectively, according to the max–min fairness.

2.2. Rate-controlled service discipline

To achieve the maximum rate control, Zhang and Ferrari proposed a general rate-controlled service discipline [9]. As shown in Fig. 1, the general rate-controlled server has two stages: rate controller and scheduler. The rate controller is responsible for shaping input traffic into desired traffic patterns, assigning an eligible time for each packet, and moving packets to the scheduler when eligible. The scheduler multiplexes eligible packets from all connections and determines the service sequence of the packets.

An important operation issue of the rate-controller is deciding when to move packets to the scheduler. A simplest method is to set a timer for the head packet of each queue. However, this introduces overhead, which is not acceptable for high-speed routers. Hardware implementation may help to reduce overhead; even so, this method would limit the number of timers, and therefore the number of classes. Most solutions in the literature are based on time-framing, event driven strategies or both. There is a tradeoff between system accuracy and time granularity in the framing strategy. A smaller frame period provides more accurate bandwidth allocation and higher operation cost, and a larger frame period results in the opposite. The event-driven strategy is based on the occurrence of driving events, e.g., packet enqueue or

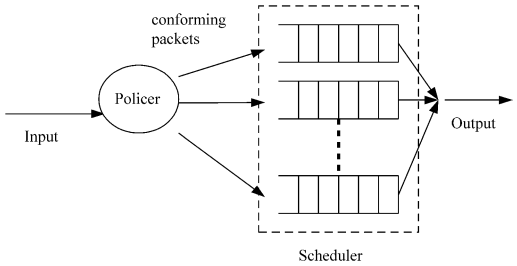

Fig. 2. Policer-based rate-control service discipline.

dequeues. As the timing of event occurrence is not predictable, high uncertainty is intrinsic in this approach. Our proposed service discipline alleviates the problem by eliminating the eligible time calculation and combining the rate controller and scheduler which greatly reduces overhead.

Another popular rate-controlled model is the policer-based rate-control service model, as shown in Fig. 2, in which a token bucket or a leaky bucket [8] is used as the policer. When an incoming packet obtains enough tokens, it proceeds directly to the scheduler; otherwise, it is dropped. Note that if a policer implemented with the packet buffer can be considered as a special case of the two-stage rate-control service discipline. While a token bucket policer can maintain the session's average rate, burst traffic with a rate exceeding the designated maximum rate may be transmitted. If the policer uses a leaky bucket, the maximum rate constraint can be strictly enforced, however bursty packets may be dropped, resulting in less than expected throughput. Although they do not have the complexity problems in a two-stage rate-control service, their drawbacks prevent them from providing effective maximum rate control.

The proposed cascade systems use either a concatenation of rate controller and scheduler, or a policer in front of scheduler. The concatenation method uses two sets of queues and management apparatus that incur overhead or inaccuracy. The latter method allows bursty traffic to pass through the controller, which, however, violates the maximum rate constraint and results in packet losses. Another approach, WF^{2}Q-M [6], calculates the sharing rate of each session, finds the saturated sessions, determines the system virtual clock ticking rate, and then uses the virtual clock adjustment method to enforce the maximum rate control by distributing the excess bandwidths of saturated sessions to other sessions without recalculating the virtual starting and finishing times of regular sessions. WF^{2}Q-M can simultaneously support the maximum rate control and minimum service rate guarantee, and thus is more efficient than conventional two-stage rate-control service disciplines (which uses only one set of queues), and produces more accurate output than the policer-based rate-control service discipline. However, WF^{2}Q-M needs to calculate the sharing rates of sessions when the system backlog is changed. Thus we need an efficient bandwidth allocation algorithm.

In the next section, we define the bandwidth sharing principle in weighted MRC fair management policy.

3. Model of weighted MRC fair queueing

The fluid model GPS-M (Generalized Processor Sharing with Maximum Rate Control) is used to illustrate the bandwidth allocation policy in this paper. GPS-M is an extension of GPS [7], and sessions that may be constrained by their assigned maximum rates P_i are called *maximum rate constrained* (MRC) sessions. If an MRC session receives a higher rate in the corresponding GPS than its assigned maximum rate, we call the session *saturated*, and the set of saturated sessions is denoted as $B_p(t)$. We also denote the set of backlogged sessions not in $B_p(t)$ as $\overline{B_p}(t)$. In GPS-M, the sessions in $B_p(t)$ receive their assigned maximum rates, and sessions in $\overline{B_p}(t)$ share the remaining bandwidth in proportion to their weights, as in GPS. The allocated bandwidth of S_i is defined as $r_i(t)$; therefore,

$$r_i(t) = \begin{cases} P_i & \text{if } S_i \in B_p(t), \\ \phi_i * norm(t) & \text{otherwise,} \end{cases}$$

$$\text{where } norm(t) = \frac{(C - \sum_{k \in B_p(t)} P_k)}{\phi_{B(t)} - \phi_{B_P(t)}}, \tag{1}$$

where $\phi_{B(t)}$ is the sum of the assigned weights of backlogged sessions, and $\phi_{B_p(t)}$ is the sum of the assigned weights of sessions in $B_p(t)$ at time t. Note that although $norm(t)$ would be infinite when $\phi_{B(t)}$ equals $\phi_{B_p(t)}$, we will not use $norm(t)$ in this situation because no session is in $\overline{B_p}(t)$. Additionally, $\sum_{i \in B_p(t)} P_i$ will not exceed the link capacity C because $\sum_{i \in B(t)} (\phi_i / \sum_{j \in B(t)} \phi_j) * C = C$ and S_i is saturated only when $(\phi_i / \sum_{j \in B(t)} \phi_j) C > P_i$. Therefore, the sum of the maximum rates of saturated sessions is always less than the link capacity.

Eq. (1) of bandwidth allocation is a declarative definition. The algorithms proposed in this paper calculate the allocated bandwidths of sessions using different ideas to utilize the proposed theorem in Section 4.

4. Saturation index

In order to minimize the time required to recalculate the allocated bandwidths of sessions, we must first

find all the eventually saturated sessions and redistribute their excess bandwidths. Once the eventually saturated sessions are identified, the allocated bandwidths of non-saturated sessions need only to be calculated once that highly reduces the computation cost. We find whether a session is eventually saturated is related to its weight and the assigned maximum rate. The saturation index SI_i, defined as the weight divided by the difference between the assigned maximum rate and the allocated bandwidth in the corresponding GPS, indicates the likelihood of S_i becoming saturated. We can find the saturated sessions more efficiently by using the saturation index.

Theorem 1. *For two sessions S_i and $S_j \in MRC$, and $SI_i > SI_j$, if $S_i \notin B_p(t)$, then $S_j \notin B_p(t)$.*

Proof. Assume there exists S_o, $j > o > i$ and $SI_i > SI_o > SI_j$ such that S_o is the first session after S_i and

$$\left(C - \sum_{k \in B_p(t)} P_k\right) * \frac{\phi_o}{\phi_{B(t)} - \phi_{B_P(t)}} > P_o.$$

Since $SI_i > SI_o$, by definition of the saturation index, we have:

$$\frac{\phi_i}{P_i - r\_GPS_i(t)} > \frac{\phi_o}{P_o - r\_GPS_o(t)}. \tag{2}$$

Note that the allocated bandwidth of S_i,

$$\left(C - \sum_{k \in B_p(t)} P_k\right) * \frac{\phi_i}{\phi_{B(t)} - \phi_{B_P(t)}},$$

can be rewritten as

$$r\_GPS_i(t) + \frac{\phi_i}{\phi_{B(t)} - \phi_{B_p(t)}} * \sum_{k \in B_p(t)} \left(r\_GPS_k(t) - P_k\right).$$

Let $C' = \sum_{k \in B_p(t)} (r\_GPS_k(t) - P_k)$. From $S_i \notin B_p(t)$, we get

$$\frac{\phi_i}{\phi_{B(t)} - \phi_{B_P(t)}} * C' + r\_GPS_i(t) < P_i. \tag{3}$$

Since S_o is the first session after S_i such that $(C - \sum_{k \in B_p(t)} P_k) * \phi_o / (\phi_{B(t)} - \phi_{B_P(t)}) > P_o$, $B_P(t)$ maintains the same and the following condition holds.

$$\frac{\phi_o}{\phi_{B(t)} - \phi_{B_P(t)}} * C' + r\_GPS_o(t) > P_o. \tag{4}$$

From (3), we have

$$\frac{C'}{\phi_{B(t)} - \phi_{B_P(t)}} < \frac{P_i - r\_GPS_i(t)}{\phi_i}. \tag{5}$$

Similarly, from (4), we have

$$\frac{C'}{\phi_{B(t)} - \phi_{B_P(t)}} > \frac{P_o - r\_GPS_o(t)}{\phi_o}. \tag{6}$$

From (5) and (6), we get

$$\frac{P_i - r\_GPS_i(t)}{\phi_i} > \frac{P_o - r\_GPS_o(t)}{\phi_o},$$

which can be rewritten as

$$\frac{\phi_i}{P_i - r\_GPS_i(t)} < \frac{\phi_o}{P_o - r\_GPS_o(t)}. \tag{7}$$

Inequality (7) violates inequality (2). Therefore, such S_o cannot exist. Since no S_o exists, $j > o > i$ such that $(C - \sum_{k \in B_p(t)} P_k) * \phi_o / (\phi_{B(t)} - \phi_{B_P(t)}) > P_o$, S_i and S_j have the same $B_p(t)$. The previous proof shows that $(C - \sum_{k \in B_p(t)} P_k) * \phi_j / (\phi_{B(t)} - \phi_{B_P(t)})$ must be less than P_j. Therefore, S_j cannot be in $B_p(t)$. □

Theorem 1 shows that if a session with SI_i is not saturated, sessions with saturation indexes less than SI_i will not become saturated. In other words, sessions with saturation indexes larger than SI_i must be saturated if S_i is saturated. Therefore, we only need to identify the saturated session with the lowest saturation index using sorting or partitioning algorithms, and then the sessions with higher saturation indexes are saturated sessions and allocated with their assigned maximum rate, while the sessions with lower saturation indexes are allocated with bandwidths proportional to their weights.

5. Algorithms of weighted MRC fair queueing

In this section, we will introduce several algorithms with various theoretical computation complexities to calculate the allocated bandwidth of each MRC session. Note that the bandwidth allocation of non-MRC sessions is trivial and thus not be considered in the following algorithms.

5.1. An intuitive algorithm

The first algorithm *WMRC_intuitive* is shown in Fig. 3. The algorithm puts the session in $B_p(t)$ and deducts its maximum rate from the link capacity, and restarts the bandwidth allocation process. Assume there are n MRC sessions. In the worst case, the last session in each run of the *for* loop in lines 2–7 becomes saturated such that the complexity of the algorithm is $n + (n-1) + (n-2) + \cdots + 2 + 1 = \frac{n(n+1)}{2} = O(n^2)$.

5.2. Algorithm using sorting

According to Theorem 1, we can sort the MRC sessions in decreasing order of the saturation index, and calculate the allocated bandwidths in that order. Fig. 4

```
Algorithm WMRC_intuitive() {
01.  Set B_p(t) = null
02.  for S_i ∈ {MRC − B_P(t)} {
03.    r_i(t) = (C − Σ_{k∈B_p(t)} P_k) * φ_i/(φ_B(t) − φ_B_P(t))
04.    if (r_i(t) > P_i) {
05.      B_p(t) = B_p(t) ∪ S_i
06.      r_i(t) = P_i
07.      go to line 2.}}}
```

Fig. 3. The algorithm *WMRC_intuitive*.

```
Algorithm WMRC_mergesort() {
01.  Set B_p(t) = null
02.  for S_i ∈ MRC {
03.    r_GPS_i = (φ_i/φ_B(t)) * C
04.    if (r_GPS_i > P_i) {
05.      B_p(t) = B_p(t) ∪ S_i
06.      r_i(t) = P_i }
07.    else SI_i = φ_i/(P_i − r_GPS_i)}
08.  mergesort(MRC − B_p(t), 1, sizeof(MRC − B_p(t))
09.  for (int i = 1; i ⩽ sizeof(MRC − B_p(t); i++) {
10.    r_i(t) = (C − Σ_{k∈B_p(t)} P_k) * φ_i/(φ_B(t) − φ_B_P(t))
11.    if (r_i(t) > P_i) {
12.      B_p(t) = B_p(t) ∪ S_i
13.      r_i(t) = P_i }}
```

Fig. 4. The algorithm *WMRC_mergesort*.

presents another algorithm, *WMRC_mergesort*, to determine the allocated bandwidths of MRC sessions. The statements in lines 2–7 perform the first scan to find the saturated sessions, and calculate SI_i. Before executing statement line 9, sessions not in $B_p(t)$ are sorted in the descending order of their saturation indexes using *mergesort* [3] algorithm. The *for* loop in lines 9–13 will find saturated sessions, and according to Theorem 1, if a non-saturated session appears, then all other non-processed sessions are not saturated.

The time complexity of the algorithm in Fig. 4 is $O(n \log n)$ [3], which is dominated by statement of line 8 when sessions are sorted in the descending order of SI_i.

5.3. Algorithm using partitioning

The algorithm *WMRC_partition* in Fig. 5 proposed in this section avoids the sorting process that dominates the complexity of the algorithm *WMRC_mergesort* in Fig. 4. The function *find_last_saturated_partition* (Fig. 6) finds all saturated sessions that uses the *partition* algorithm. The partition algorithm selects a session q with the saturation index SI as the pivot to rearrange the sessions in $A[p..r]$. During the execution, the sessions are partitioned into four (possibly empty) regions: sessions with saturation indexes that are larger than or

```
Algorithm WMRC_partition {
01.  Set B_p(t) = null
02.  for S_i ∈ MRC{
03.    r_GPS_i = (φ_i/φ_B(t)) * C
04.    if (r_GPS_i > P_i) {
05.      B_p(t) = B_p(t) ∪ S_i
06.      r_i(t) = P_i }
07.    else SI_i = φ_i/(P_i − r_GPS_i)}
08.  find_last_saturated_partition((MRC − B_p(t)),
       1, sizeof(MRC − B_p(t)))
09.  for S_i ∈ {MRC − B_P(t)}
10.    r_i(t) = (C − Σ_{k∈B_p(t)} P_k) * φ_i/(φ_B(t) − φ_B_P(t))}
```

Fig. 5. The algorithm *WMRC_partition*.

```
Algorithm find_last_saturated_partition(A, p, r) {
01.  if (p == r) {
02.    r_p(t) = (C − Σ_{k∈B_p(t)} P_k) * φ_p/(φ_B(t) − φ_B_P(t))
03.    if (r_p(t) > P_p) {
04.      B_p(t)+ = S_p
05.      return p}
06.    else return p − 1}
07.  k = RANDOM(p, r)
08.  i = partition(A, p, r, SI_k)
09.  temp_sum_peakrate = sum_peakrate + Σ_{j=p}^{i−1} P_j
10.  temp_sum_weight = sum_weight + Σ_{j=p}^{i−1} φ_j
11.  r_i(t) = (C − Σ_{k∈B_p(t)} P_k − temp_sum_peakrate) * φ_i /
           (φ_B(t) − φ_B_P(t) − temp_sum_weight))
12.  if (r_i(t) > P_i) {
13.    for (h = p; h <= i; h ++) B_p(t) = B_p(t) ∪ S_h
14.    sum_peakrate = temp_sum_peakrate + P_i
15.    sum_weight = temp_sum_weight + φ_i
16.    return find_last_saturated_partition(A, i + 1, r)}
17.  else return find_last_saturated_partition(A, p, i − 1)
```

Fig. 6. The algorithm *find_last_saturated_partition*.

equal to SI, less than SI, pivot and sessions that are not traversed. At termination, the saturation indexes in $A[p..i]$ are all greater than or equal to SI and the saturation indexes in $A[i+1..j-1]$ are all less than SI.

The *find_last_saturated_partition* algorithm checks whether the random selected session is saturated. If it is saturated, the sessions before it must also be saturated, since—according to the *partitioning* algorithm—their saturation indexes are larger than SI_k; therefore, the '*last*' saturated session is after S_i. Otherwise, the sessions between S_p and S_i will be searched. If the input array has only one session, the algorithm converges and checks whether the session is saturated. If the session is saturated, it is the '*last*' saturated session; otherwise, the session before it is the '*last*' saturated session. After finding all the saturated sessions, we can calculate the allocated bandwidths of the non-saturated sessions.

The worst-case running time for *find_last_saturated_partition* is $O(n^2)$, because it may always partition the

Table 1
Time complexities of proposed algorithms

	Worst case	Average case
WMRC_intuitive	$O(n^2)$	$O(n^2)$
WMRC_mergesort	$O(n \lg(n))$	$O(n \lg(n))$
WMRC_partition	$O(n^2)$	$O(n)$

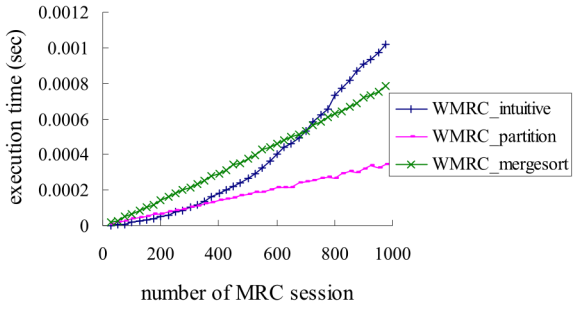

Fig. 7. The average running time for MRC sessions from 1 to 1000.

saturation indexes based on the smallest remaining element, and each partitioning takes $O(n)$ time. However, the algorithm generally works well. The operation of *find_last_saturated_partition* is similar to the problem of selecting the ith order statistic from a set of n distinct numbers such that the average running time complexity of *find_last_saturated_partition* and *WMRC_partition* are both $O(n)$ [3]. To sum up, Table 1 shows the time complexities of the algorithms proposed in this paper.

6. Performance evaluation

We implement the three algorithms and compare their performance to see how the algorithms behave under different number of MRC sessions on a PC-based platform with an Intel P3 550 CPU, a 256 M RAM and OS LINUX RedHat 7.2. We used RDTSC (Read Time Stamp Counter), which returns the number of clock cycles from the time the CPU is powered up or reset, instead of function *gettimeofday*, to calculate the running time more precisely. The weights and maximum rates of maximum rate constraint sessions were randomly selected. For each number of MRC sessions, we used ten random seeds to generate ten sets of data, and take the average running time.

Fig. 7 shows the evaluation results for MRC sessions from 1 to 1000. The result shows that *WMRC_intuitive* has the best performance when the number of MRC session is less than 300, but the execution time grows quickly when there are more than 500 MRC sessions. On the contrary, the execution time of *WMRC_mergesort* and *WMRC_partition* increase linearly as the number of MRC sessions increases. *WMRC_mergesort* and *WMRC_partition* perform better than *WMRC_intuitive* if the system has more than 700 and 300 MRC sessions, respectively. Moreover, *WMRC_partition* should be the best-practice algorithm as it performs well regardless of the number of MRC sessions.

7. Conclusion

In this paper, we address the problem of calculating the allocated bandwidths of sessions in the weighted maximum rate constrained link sharing policy, which is an important management feature for service providers and many applications. The intuitive solution has a complexity $O(n^2)$. We define the saturation index and prove that if S_i with saturation index SI_i is not saturated, sessions with saturation indexes less than SI_i will not become saturated. This property can be used as the termination condition to greatly reduce the running time. According to that property, we propose several algorithms with different running time complexities, and implement these algorithms on a PC-based platform to find the best-practice algorithm. In this work, we reduce the average case complexity from $O(n^2)$ to $O(n)$.

References

[1] D.P. Bertsekas, R. Gallager, Data Networks, Prentice-Hall, Englewood Cliffs, NJ, 1992.
[2] J.C.R. Bennett, H. Zhang, WF^2Q: worst-case fair weighted fair queueing, in: Proc. IEEE INFOCOM'96, San Francisco, CA, March 1996.
[3] H. Cormen, E. Leiserson, L. Rivest, C. Stein, Introduction to Algorithms, second ed., MIT Press, Cambridge, MA, 2001.
[4] S. Golestani, A self-clocked fair queueing scheme for broadband applications, in: Proc. IEEE INFOCOM'94, Toronto, CA, June 1994, pp. 636–646.
[5] P. Goyal, H.M. Vin, H. Chen. Start-time Fair Queuing: A scheduling algorithm for integrated services, in: Proc. ACM–SIGCOMM 96, Palo Alto, CA, August 1996, pp. 157–168.
[6] J.F. Lee, Y. Sun, M.C. Chen, On maximum rate control of weighted fair scheduling for transactional systems, RTSS, Cancun 2003.
[7] A.K. Parekh, R.G. Gallager, A generalized processor sharing approach to flow control in integrated services networks: The single-node case, IEEE/ACM Trans. Networking 1 (3) (1993) 344–357.
[8] A.S. Tanenbaum, Computer Networks, third ed., Prentice-Hall, Inc., Englewood Cliffs, NJ, 1996, pp. 380–381.
[9] H. Zhang, D. Ferrari, Rate-controlled service disciplines, J. High Speed Networks 3 (4) (1994) 389–412.