Guaranteeing service rates for cell-based schedulers with a grouping architecture

D. Wei, J. Yang, N. Ansari, S. Papavassiliou

Abstract: Packet fair queueing (PFQ) algorithms are packetised versions of generalised processor sharing (GPS), which is an idealised fluid scheduling model with desirable properties in terms of delay bound and fairness. To support a large number of sessions with diverse bandwidth requirements, the grouping architecture, which is scalable, has been proposed to approximate PFQ algorithms. The authors analyse the relationship between the guaranteed service rates and utilisation for cell-based schedulers with the grouping architecture. Based on this analysis, call admission control (CAC) schemes are proposed to provide the guaranteed service rate for each session, and their performance is evaluated and compared in terms of computational complexity.

List of symbols

g_m	the mth service rate group
r_{g_m}	the service rate of g_m
r_{min}	the minimum group service rate, i.e. r_{g_1}
e_i	the absolute error of the service rate of session i given $r_{g_{m-1}} < r_i \leq r_{g_m}$, then $e_i = r_{g_m} - r_i$
ε_i	the relative service rate error of session i, $\varepsilon_i = e_i/r_i$
$\hat{r}_i$	the guaranteed service rate of session i

1 Introduction

High-speed, service-integrated packet switches are required to support a large number of sessions with diverse service rate requirements. Statistical multiplexing is employed to improve the throughput of a switch. When multiplexed at the same output of a scheduler, different sessions interact with each other, and therefore scheduling algorithms are used to control the interactions among them.

Based on an idealised fluid model, Parekh [1] proposed the generalised processor sharing (GPS) algorithm and its packet-based version – packet-by-packet generalised processor sharing (PGPS), which can be implemented in integrated services networks. The GPS algorithm has been proven to have three desirable properties: (i) it can guarantee the latency bound to any leaky-bucket-constrained session; (ii) it can ensure fair allocation of bandwidth among all backlogged sessions; (iii) it has a certain capability of immunity, i.e. it can isolate well-behaving sessions from the disadvantageous effects of misbehaving sessions. However, GPS is an idealised model and cannot be implemented in real world. Some service disciplines generally called packet fair queueing (PFQ) algorithms, which differ in tradeoffs between implementation complexity and performance in terms of latency bound and service fairness, have been proposed to approximate GPS. In reality, owing to the complexity, it is difficult to implement these disciplines in a scheduler to support a large number of sessions with diverse service rate requirements while maintaining all desirable GPS properties.

Implementation complexity of PFQ algorithms is determined by the following factors [2]: (i) the calculation of the system virtual time, which indicates the amount of normalised fair service that should be received by each session; (ii) sorting the service order of all sessions; (iii) the management of another priority queue to regulate packets (only if those algorithms with the 'smallest eligible virtual finish time first', such as WF^{2}Q [3] or WF^{2}Q+ [4], are adopted as the service discipline). PGPS [1] and weighted fair queueing (WFQ) [5] use the virtual system time defined by the fluid GPS model. Both need to track all backlogged sessions, and hence the worst case complexity is $O(N)$, where N is the number of sessions. Some other PFQ algorithms, whose virtual system time complexity is $O(1)$ [6] and $O(\log N)$ [7, 4], have been developed. The sorting complexity of most algorithms is $O(\log N)$. Suri *et al.* [8] proposed use of the van Emde Boas data structure, which has a complexity of $O(\log\log N)$. Zhang *et al.* [3, 4] proposed a selection policy involving selecting packets among all eligible sessions. This selection policy can improve the worst-case delay for clearing the backlog of a session's queue, but it requires extra management of another priority queue.

A novel grouping architecture (a discrete-rate approach), which can dramatically reduce the overall complexity, has been proposed in [2]. All sessions with the same service rate stay in the same group when they are active. Chiussi and Francini [9] proposed a non-per-connection-timestamp scheduling algorithm, which can further simplify the computation and storage of the timestamp, reducing the number of timestamp from N to M, where M is the number of service groups.

To guarantee the service rate of each session, inequality $\sum_{i=1}^{N} r_i \leq R$ is used for admission control in [1], where r_i is the service rate of session i, and R is the output capacity of the scheduler. With this admission control policy, we demonstrate, in the following Section, that some sessions'

IEE Proceedings online no. 20030283

doi:10.1049/ip-com:20030283

Paper first received 21st January and in revised form 4th September 2002

The authors are with the Electrical and Computer Engineering Department, New Jersey Institute of Technology, University Heights, Newark, New Jersey, 07102, USA

service rates cannot be guaranteed because of the approximation of the required service rate in the grouping architecture, when all sessions are continuously backlogged.

So far, no work on implementing admission control schemes for schedulers with a grouping architecture has been reported. In this paper, we analyse the relationship between the guaranteed service rate and admission control policy for this type of schedulers. Based on this mathematical analysis, we propose two admission control schemes, which differ in terms of the computational complexity and system utilisation.

2 Background

The grouping architecture is a discrete approach to approximate packet fair queueing (PFQ) algorithms, which are also approximations of the fluid-based generalised processor sharing (GPS) model.

2.1 Grouping architecture – a discrete-rate approach

PFQ algorithms have a global variable, virtual system time $V(.)$, which is defined differently for different PFQ algorithms. They also maintain a virtual start time and a virtual finish time for each session. When p_i^k, the kth packet of session i, arrives, the virtual start time $S_i(.)$ and virtual finish time $F_i(.)$ of this packet are updated as follows:

$$S_i(t) = \begin{cases} \max(V(t), F_i(t-)) & \text{session } i \text{ becomes active} \\ F_i(t-) & p_i^{k-1} \text{ finished service} \end{cases}$$

$$F_i(t) = S_i(t) + \frac{L_i^k}{r_i}$$

where L_i^k is the packet size of the kth packet of session i, and r_i is the required service rate of session i. The virtual system time is updated when a packet starts to receive service or new sessions become active.

A grouping architecture for cell-based schedulers [2], as shown in Fig. 1, is presented to efficiently implement PFQ algorithms in high-speed cell-based switches. By employing the locally bounded timestamp (LBT) property [2], the priority relationship among sessions in the same service rate group can be maintained without sorting. All sessions with the same service rate requirements are placed in the same group. The operation of the system can be summarised as follows:

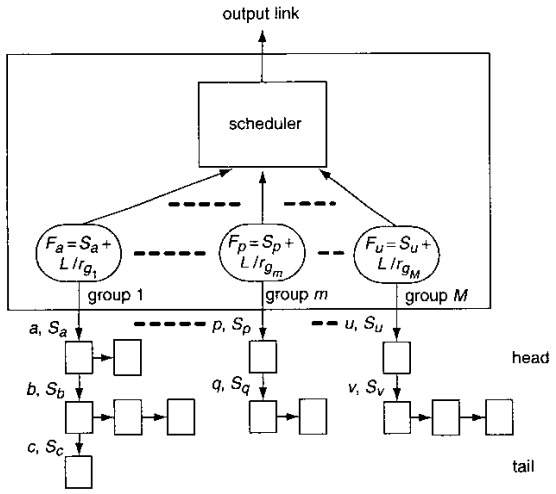

Fig. 1 *Cell-based scheduler with the grouping architecture*

(i) When a new session, for example session i, is set up, it is assigned to a service rate group according to its rate requirement. The service rate of the group must not be less than the requirement of session i. That is, session i is placed into the mth service rate group such that $r_{g_{m-1}} < r_i \leq r_{g_m}$. At this moment, the first packet of session i is placed at the tail of its service rate group.

(ii) The scheduler selects the packet with the smallest virtual finish time to transmit among all sessions in the heads of service rate groups.

(iii) After a session receives service, if it is still backlogged, it is placed at the tail of the service rate group; if the session is temporarily idle or finished, it is taken out of the service rate group. The next session in the same group is placed at the head.

(iv) When a session becomes active again, it can be treated as a new session and placed at the tail of the corresponding group.

In each group, each backlogged session is shifted one by one to the head of the group, and thus the session with the smallest virtual start time in each group is always at the head of the group. Scheduling is performed only among sessions at the head of each group. Therefore, with this grouping architecture, the worst-case algorithm complexity of scheduling and updating of the virtual system time is reduced. For example, if WF^{2}Q+ is employed, the implementation complexity is reduced from $O(\log N)$ to $O(\log M)$, where N is the number of sessions and M is the number of service rate groups. In other words, the complexity of scheduling and updating of the virtual system time is decoupled from the number of sessions. Another key advantage of the grouping architecture is that it is able to perform per session-based traffic regulation (by the virtual start time) and traffic scheduling (by the virtual finish time) in an integrated manner, hence reducing the overall worst-case complexity.

2.2 Call admission control

Clearly, it is not possible to provision quality-of-service (QoS) guarantees, which is very important in the design of integrated services networks, without call admission control. When a new session arrives the scheduler, the philosophy of call admission control can be put forth by the following:

If the new session is accepted by the scheduler, can the required QoS of all sessions (including the new session and those already existing sessions in the scheduler) be guaranteed?

In general, QoS parameters could include: service rate, delay bound, delay-jitter, loss rate, and so forth. In this paper, the QoS parameter under consideration is the service rate. That is, the scheduler can accept a new session only if the service rate requirement of each session is guaranteed.

3 Mathematical basis for admission control

Consider that N sessions are served by a scheduler with the grouping architecture. To meet the service rate requirement of each session, the following inequality must hold:

$$\hat{r}_i \geq r_i, \forall i = 1, 2, \ldots N \qquad (1)$$

This scheduler can be considered as a discrete version of the rate proportional processor sharing (RPPS) model [1], in which the service rates of all sessions are set according to the

session group service rate proportionally:

$$\hat{r}_i = \frac{r_i(1+\varepsilon_i)}{\sum_{j=1}^{N} r_j(1+\varepsilon_j)} R \tag{2}$$

Theorem 1: If $\sum_{i=1}^{N}(r_i + e_i) \le R$, then the service rate requirement of each session can be guaranteed.

Proof:

$$\because \sum_{i=1}^{N}(r_i+e_i) \le R, \ \therefore \frac{R}{\sum_{i=1}^{N}(r_i+e_i)} \ge 1$$

By using (2), $\hat{r}_i \ge r_i(1+\varepsilon_i)$.

$\because \varepsilon_i \ge 0, \ \therefore \hat{r}_i \ge r_i$. Thus, (1) can be met. □

Definition 1: Session i is said to be *least-in-favour* if

$$\varepsilon_i = \varepsilon_{LIF}^N \triangleq \min(\varepsilon_1, \varepsilon_2, \ldots \varepsilon_N) \tag{3}$$

Session i is said to be *most-in-favour* if

$$\varepsilon_i = \varepsilon_{MIF}^N \triangleq \max(\varepsilon_1, \varepsilon_2, \ldots \varepsilon_N) \tag{4}$$

Lemma 1: The service rate of each session can be satisfied if and only if the following inequality holds:

$$(1+\varepsilon_{LIF}^N) \ge \frac{\sum_{j=1}^{N} r_j(1+\varepsilon_j)}{R} \tag{5}$$

Proof:

(i) If all sessions' service requirements are satisfied, (1) must be met. Using (1) and (2),

$$\frac{(1+\varepsilon_i)}{\sum_{j=1}^{N} r_j(1+\varepsilon_j)} \cdot R \ge 1, \ \forall i = 1, 2, \ldots, N \tag{6}$$

$$(1+\varepsilon_i) \ge \frac{\sum_{j=1}^{N} r_j(1+\varepsilon_j)}{R}, \ \forall i = 1, 2, \ldots, N \tag{7}$$

Thus, (5) is obtained by (3) and (7).

(ii) From (3),

$$1+\varepsilon_i \ge 1+\varepsilon_{LIF}^N, \ \forall i = 1, 2, \ldots N \tag{8}$$

If (5) holds, (7) is true. Thus (6) is satisfied, and therefore (1) must be met. □

Lemma 1 states the necessary and sufficient condition to guarantee the service rate of each session. The right-hand side of (5) can be regarded as the 'overbooking' factor of the aggregated service rate of all sessions, and the left-hand side is the 'overbooking' factor of the least-in-favour session. Thus, lemma 1 can be interpreted as: the service rate of each session can be satisfied if and only if the service rate of the least-in-favour session is satisfied.

Definition 2: The scheduler with the grouping architecture is said to be *saturated* when the service rate of each current session in the scheduler can be satisfied and no more sessions can be admitted.

Theorem 2: The scheduler is *saturated* with N sessions if and only if (5) and the following inequality must hold simultaneously:

$$1+\varepsilon_{LIF}^N < \frac{\sum_{j=1}^{N} r_j(1+\varepsilon_j)}{R} + \frac{r_{\min}}{R} \tag{9}$$

Proof:

(i) When the scheduler is saturated with N sessions, inequality (5) must be met. If one more session $N+1$ is admitted, at least one session's service rate cannot be met. That is:

$$\exists i \in [1, N+1], s.t.(1+\varepsilon_i) < \frac{\sum_{j=1}^{N} r_j(1+\varepsilon_j)}{R} + \frac{r_{N+1}(1+\varepsilon_{N+1})}{R} \tag{10}$$

i.e. the following inequality must not hold:

$$1+\varepsilon_{LIF}^{N+1} \ge \frac{\sum_{j=1}^{N} r_j(1+\varepsilon_j)}{R} + \frac{r_{N+1}(1+\varepsilon_{N+1})}{R} \tag{11}$$

When the scheduler is saturated and (5) holds, assume (9) does not hold, i.e.

$$1+\varepsilon_{LIF}^{N} \ge \frac{\sum_{j=1}^{N} r_j(1+\varepsilon_j)}{R} + \frac{r_{\min}}{R} \tag{12}$$

Select session $N+1$ such that $r_{N+1}(1+\varepsilon_{N+1}) = r_{\min}$ and $\varepsilon_{N+1} = \varepsilon_{LIF}^N$, thus (11) is satisfied, i.e. session $N+1$ can be admitted and the scheduler is not saturated. Therefore, the assumption is contradictory. Hence (12) cannot hold, i.e. (9) must hold.

(ii) By contradiction. When (5) and (9) hold, assume the scheduler is not saturated, i.e. session $N+1$ can be admitted. By lemma 1, (11) must be met as well. Subtracting (9) from (11):

$$\varepsilon_{LIF}^{N+1} - \varepsilon_{LIF}^{N} > \frac{r_{N+1}(1+\varepsilon_{N+1})}{R} - \frac{r_{\min}}{R} \tag{13}$$

Since the right hand side

$$\frac{r_{N+1}(1+\varepsilon_{N+1})}{R} - \frac{r_{\min}}{R} \ge 0, \text{ and}$$

$$\varepsilon_{LIF}^{N+1} = \min(\varepsilon_{LIF}^N, \varepsilon_{N+1}) \le \varepsilon_{LIF}^N$$

the left-hand side of (13) ≤ 0. Therefore, (13) cannot hold. The assumption is contradictory. Therefore, when (5) and (9) hold simultaneously, no more sessions can be accepted, i.e. the scheduler is saturated. □

Theorem 2 provides the necessary and sufficient condition that a scheduler is saturated. We see that if a new session $N+1$, with $r_{N+1} = r_{min}$, cannot be accepted by the scheduler, then the scheduler is saturated.

4 Implementation and performance analysis

Let the condition for accepting a new session be such that every session (including the newly arrived one) service rate can be met if this session is accepted.

Based on the analysis presented in the previous section, we propose two call admission control schemes. Scheme 1 employs theorem 1, which provides a sufficient condition to guarantee the required service rate of each session. For the second scheme, we propose two different implements, referred to as schemes 2*a* and 2*b*. Scheme 2*a*, which employs (1) and (2), examines if the service rate of every session can be guaranteed. Scheme 2*b*, which is mathematically equivalent to scheme 2*a*, employs the results of lemma 1 and theorem 2. The algorithms of schemes 1, 2*a* and 2*b* are shown in Figs. 2–4, respectively.

Assume that N sessions are being served and $\sum_{i=1}^{N}(r_i + e_i)$ has been stored. When a new session $N+1$ arrives, scheme 1 is only required to check if $\sum_{i=1}^{N}(r_i+e_i)+(r_{N+1}+e_{N+1}) \le R$, and thus its computational complexity is $O(1)$. The computational complexity of scheme 2*a* is $O(N)$, because the service rate of each session should be

```
/* when new session N+1 comes */
1   if Ag_R(N) + r_{N+1}(1+ε_{N+1}) ≤ R   /* aggregate rate requirement
        Ag_R(N) = Σ_{j=1}^{N} r_j(1+ε_j)  */
2   then ACCEPT (session N+1)
3       Ag_R(N+1) ← Ag_R(N) + r_{N+1}(1+ε_{N+1})
4   else REJECT (session N+1)
5   endif

/* when session i ends, update Ag_R */
1   Ag_R(N-1) ← Ag_R(N) - r_i(1+ε_i)
```

Fig. 2 *Scheme 1: call admission control scheme which employs the sufficient condition*

```
/* when new session N+1 comes */
1   result = TRUE
2   for i = 1 to N+1
3       if Ag_R(N) + r_{N+1}(1+ε_{N+1}) > R·(1+ε_i)
4       then result = FALSE
5           break
6   if result = TRUE
7   then ACCEPT (session N+1)
8       Ag_R(N+1) ← Ag_R(N) + r_{N+1}(1+ε_{N+1})
9   else REJECT (session N+1)
10  endif

/* when session i ends, update Ag_R */
1   Ag_R(N-1) ← Ag_R(N) - r_i(1+ε_i)
```

Fig. 3 *Scheme 2a: call admission control scheme which employs (2)*

```
/* when new session N+1 comes */
1   if scheduler_is_saturated = TRUE
2   then REJECT (session N+1)
3   else
1       if Ag_R(N) + r_{N+1}(1+ε_{N+1}) ≤ R·(1+min(ε_LIF^N, ε_{N+1}))
5       then ACCEPT (session N+1)
6           Ag_R(N+1) ← Ag_R(N) + r_{N+1}(1+ε_{N+1})
7           ε_LIF^{N+1} ← min(ε_LIF^N, ε_{N+1})
8           if R·(1+ε_LIF^{N+1}) < Ag_R(N+1) + r_min
9           then scheduler_is_saturated = TRUE
10      endif
11  endif
12  endif

/* when session i ends, update Ag_R and ε_LIF^{N-1}*/
1   Ag_R(N-1) ← Ag_R(N) - r_i(1+ε_i)
2   ε_LIF^{N-1} ← min_{j=1,j≠1}^{N} (ε_j)
3   scheduler_is_saturated = FALSE
```

Fig. 4 *Scheme 2b: improved call admission control scheme which employs the necessary and sufficient condition*

Table 1: Performance comparison of admission control schemes

	Worst-case complexity	Average complexity
Scheme 1	$O(1)$	$O(1)$
Scheme 2a	$O(N)$	$O(N)$
Scheme 2b	$O(\log N)$	$O(1)$

checked when a new session arrives. If admission control scheme 2*b* is employed, when session i ends: if $\varepsilon_{LIF}^{N} = \varepsilon_i$, then the complexity to find ε_{LIF}^{N-1} is $\log_2(N-1)$, i.e. $O(\log N)$; if $\varepsilon_{LIF}^{N} \neq \varepsilon_i$, then $\varepsilon_{LIF}^{N-1} = \varepsilon_{LIF}^{N}$, and in this case, the complexity can be considered as $O(1)$. Assuming each session has the same probability to leave the scheduler, the average complexity to update ε_{LIF}^{N-1} is

$$\frac{1}{N} \cdot \log(N-1) + \frac{N-1}{N} \cdot 1$$

When N is big enough (so far, the schedulers which can support hundreds of thousands sessions have been widely reported), the average complexity can be considered as $O(1)$. Table 1 lists the computational complexity of the three schemes.

Although scheme 1 has desirable performance in terms of computational complexity, it presents an 'under-utilisation' issue. To illustrate this issue, we use the following scheduler as an example:

Let us assume a cell-based scheduler S with the grouping architecture, $R = 16$, $M = 5$, $r_{g_1} = 1$, $r_{g_2} = 2$, $r_{g_3} = 4$, $r_{g_4} = 8$ and $r_{g_5} = 16$. Assume that three sessions have been accepted, $r_1 = r_2 = 3$, $r_3 = 6$, thus, $\hat{r}_1 = \hat{r}_2 = 4$, $\hat{r}_3 = 8$, i.e. all sessions' service rates are guaranteed.

If $\sum_{i=1}^{N} r_i \leq R$ is used as the call admission control scheme [1], a new session with required service rate $r_4 = 2$ should be accepted; however, if session 4 is accepted, its guaranteed service rate $\hat{r}_4 = 1.78$, and thus, its service rate requirement cannot be satisfied. Therefore, it cannot be used as the call admission control scheme for the grouping architecture.

If we use scheme 1, $\sum_{i=1}^{3} (r_i + e_i) = 16$, and thus, no more sessions can be accepted. The long-term utilisation is 0.75. If scheme 2*a* or 2*b* is employed, a new session whose service rate requirement is 3 can be accepted. Thus, the long-term utilisation is 0.9375. The reason is that scheme 1 employs theorem 1, which is the necessary condition of (1), scheme 2*a* employs (2) to check if the service rate of each session can be satisfied, and 2*b* employs (5) and (9), which is the necessary and sufficient condition to guarantee the service rate of each session. Therefore, the utilisation of a scheduler using scheme 2*a* or 2*b* is equal or better than that of a scheduler using scheme 1, if both schedulers have the same grouping architecture.

5 Conclusions

In this paper, we have analysed the relationship between the guaranteed service rate and utilisation for cell-based schedulers with the grouping architecture. Based on this analysis, we have proposed two call admission control schemes to guarantee the service rate for each session; we have also evaluated and compared their performance in terms of computational complexity and utilisation.

We have achieved the primary objective of guaranteeing session service rates for cell-based schedulers with the

grouping architecture, and our future investigation will aim to also meet other QoS requirements such as delay bound and delay jitter.

6 Acknowledgments

This work was supported in part by the NJ Commission on Science and Technology via the New Jersey Center for Wireless Telecommunications, and the New Jersey Commission on Higher Education via the NJ I-TOWER project. We would also like to thank the anonymous reviewers for their insightful comments.

7 References

1 PAREKH, A.K.: 'A generalized processor sharing approach to flow control in integrated services networks', PhD thesis, MIT, 1992
2 STEPHENS, D.C., BENNETT, J.C.R., and ZHANG, H.: 'Implementing scheduling algorithms in high-speed networks', *IEEE J. Sel. Areas Commun.*, 1999, **17**, (6), pp. 1145–1158
3 BENNETT, J.C.R., and ZHANG, H.: 'WF^{2}Q: worst-case fair weighted fair queuing'. Proceedings of IEEE INFOCOM'96, March 1996, pp. 120–128
4 BENNETT, J.C.R., and ZHANG, H.: 'Hierarchical packet fair queuing algorithms'. Proceedings of ACM SIGCOMM'96, August 1996, pp. 143–156
5 DEMERS, A., KESHAV, S., and SHENKER, S.: 'Analysis and simulation of a fair queuing algorithm', *Internetw. Res. Exper.*, 1990, **1**, (1), pp. 3–26
6 GOLESTANI, S.J.: 'A self-clocked fair queuing scheme for broadband applications'. Proceedings of IEEE INFOCOM'94, April 1994, pp. 636–646
7 STILIADIS, D., and VARMA, V.: 'Design and analysis of frame-based fair queuing: a new traffic scheduling algorithm for packet-switched networks'. Proceedings of the ACM SIGMETRICS conference on Measurement & modeling of computer systems, 1996, pp. 104–115
8 SURI, S., VARGHESE, G., and CHANDRANMENON, G.: 'Leap forward virtual clock'. Proceedings of INFOCOM'97, April 1997, Vol. 2, pp. 557–565
9 CHIUSSI, F.M., and FRANCINI, A.: 'Implementing fair queuing in ATM switches: the discrete-rate approach'. Proceedings of IEEE INFOCOM '98, 1998, Vol. 1, pp. 272–281