Implicit Service Differentiation using Deficit Round Robin

A. Kortebi, S. Oueslati, J. Roberts

France Telecom, Division R&D
CORE/CPN
38, rue du Général Leclerc
92794 Issy les Moulineaux, France
{abdesselem.kortebi, sara.oueslati, james.roberts}@francetelecom.com

This paper describes and evaluates the performance of a modified deficit round robin scheduler called PDRR (for Priority DRR). In conjunction with per-flow admission control, PDRR allows the realization of implicit service differentiation where streaming and elastic flows are assured adequate performance without the need for explicit class of service marking or resource reservation. We show through simulation that PDRR is scalable and feasible since the number of flows to be scheduled remains less than a few hundred for any link speed. It is also shown to provide very low latency for the packets of streaming flows as long as their rate is a relatively small fraction of the link rate.

Key words: implicit service differentiation, fair queueing, deficit round robin, flow-aware networking.

1. Introduction

The practical difficulties of using mechanisms of standardized QoS models to cost-effectively realize performance guarantees have led us to define an alternative flow-aware networking architecture [8,7]. This architecture applies traffic controls to user defined flows identified on the fly by inspection of packet headers. The controls in question are per-flow scheduling and per-flow admission control invoked locally and independently on individual network links. The controls can be combined to provide necessary performance guarantees to streaming and elastic flows without any change to the current best effort user-network interface.

Implicit service differentiation is realized by three devices: max-min fair sharing is realizing per-flow fair queueing; packets of flows that emit packets at a rate less than the current fair rate are forwarded via a priority queue; admission control is applied to maintain the fair rate above a certain threshold. The latter threshold is chosen to ensure low latency forwarding for a useful range of streaming applications. A flow in this system is locally defined at any given link and consists of all packets with an identical flow identifier (derived from particular header fields such as IP addresses and the IPv6 flow label or IPv4 port numbers) occurring with some minimum inter-packet interval (a few seconds).

We introduced the notion of implicit service differentiation in [6] with fair queueing

realized using a self-clocked fair queueing algorithm called PFQ (for Priority Fair Queueing). The scalability of this algorithm was demonstrated by means of trace driven simulations and analytical modelling [4,5]. In the present paper we show that PFQ can be advantageously replaced by an adaptation of the well-known Deficit Round Robin (DRR) scheduler [10]. We retain the low complexity of DRR while again providing low latency for streaming flows.

The paper describes the PDRR (for Priority DRR) scheduler and presents the results of a performance evaluation. The latter is conducted by simulation using a mixture of real trace data and synthetically generated traffic. The results confirm the claimed scalability and latency properties. We also illustrate through an example how admission control preserves performance in overload. We first recall in the next section the essential principles of implicit service differentiation.

2. Implicit service differentiation

We consider the objective of service differentiation to be to ensure low packet latency for streaming applications while using residual bandwidth to provide maximum throughput to elastic flows. We have argued elsewhere that it is not useful, at least not in the backbone, to distinguish between more precisely defined classes [8]. Implicit service differentiation is possible on recognizing that streaming flows have a relatively low rate and that this characteristic can be used as the key to discrimination.

2.1. Priority fair queueing

Imposing max-min fair sharing between flows on individual links leads to overall max-min fair sharing in the network [3]. This is a reasonable resource allocation goal for elastic traffic [2]. Network imposed fairness makes performance less vulnerable to user misbehaviour and avoids the requirement for transport protocols to be "friendly" to legacy TCP [7].

With max-min fairness, any flow whose rate is less than the fair rate on all links of its path should not suffer packet loss. However, depending on the particular fair queueing scheduler employed, packet latency can be significant. To preserve the necessary low latency for streaming flows, we propose to give head of line priority to all packets of flows whose rate is less than the current fair rate. This is the principle of priority fair queueing.

Any flow whose rate exceeds the fair rate will suffer packet delay. If the rate excess is momentary, this delay simply smoothes the flow rate; any flow whose rate is persistently greater than the fair rate will loose packets and typically have to adapt the source rate in consequence.

2.2. Admission control

Priority fair queueing is an adequate traffic control in normal load conditions. To maintain good performance in overload, we propose to additionally employ implicit per-flow admission control.

If the fair rate should descend below a certain threshold or the load of the priority queue exceed another threshold, it is necessary to deny access to any new flows to protect the service quality of flows already in progress. The fair rate threshold must be chosen to ensure adequate throughput for elastic flows and priority treatment for the packets of

a useful range of streaming applications. The load threshold should ensure low enough packet latency. Note that per-flow fair queueing facilitates the measurement of these congestion indicators.

Admission control can be performed *implicitly* simply by discarding the packets of new flows whenever congestion occurs [1]. Higher layer protocols must be designed to recognize packet discard as a sign of flow level congestion and react accordingly.

2.3. Differentiation with respect to flow rates

To use flow rate as a key to service differentiation appears to be adequate whenever the link in question realizes a fairly high degree of sharing. This is because, to realize efficient statistical multiplexing, the peak rate of streaming flows must typically be a small fraction of the link rate. A value of around 1% is compatible with medium to high utilization (see [9, Sec 16.1], for example). In this case, an admission control threshold on fair rate of around 1% of link capacity is sufficient to ensure low latency for such flows. This threshold is also known to realize a satisfactory compromise between low blocking in normal load and adequate elastic flow throughput performance in overload [1].

The peak rate emerges as an essential traffic characteristic for both streaming and elastic flows. For streaming flows, the peak rate determines the efficiency of statistical multiplexing. For elastic flows, the peak rate, defined as the rate limit imposed by constraints outside a considered network, determines whether the flow can or cannot realize a fair share of available bandwidth. It is thus a natural as well as very convenient to choose as a criterion for differentiation.

3. Priority Deficit Round Robin

Priority Deficit Round Robin, or PDRR, is a fair queueing algorithm that inherits the O(1) complexity and fairness properties of DRR while improving latency by the use of a priority queue for low rate flows. The proposed use of fair queueing is somewhat unusual in that the population of flows to be scheduled is highly dynamic being determined only by the origin of packets currently in the buffer (like [11]).

3.1. Pseudo code

Table 3.2 presents the PDRR pseudo-code with additional instructions with respect to DRR highlighted using italics (see [10] for more information on DRR). We generally envisage use of PDRR for equitable fair sharing but present the pseudo-code for weighted sharing by means of variable quanta Q_i to facilitate the comparison with DRR.

A data structure called the active flow list (AFL) stores data for flows that have, or recently have had, packets in the queue. These data include the flow identity, the current deficit count DC_i, flow quantum Q_i and pointers realizing a FIFO linked list of queued packets for that flow. An additional parameter $ByteCount(i)$ is used to determine whether flow packets should or should not be sent to the priority queue (PQ). AFL entries are visited in a certain order in each scheduling round. This order is defined by a pointer in each AFL entry indicating the next flow to be visited.

3.1.1. Enqueue operation

If on the arrival of a packet p the buffer is full, a packet must be selected for dropping (lines 2-3)[1]. If packet p does not belong to an active flow, the flow is added to the current end of the AFL cycle and the packet is inserted at the tail of PQ (lines 5-9). $ByteCount(i)$ keeps track of the number of bytes already inserted in PQ for flow i. A packet arriving to an active flow will be given priority while $ByteCount(i) \leq Q_i$ (lines 11-13); otherwise it is placed at the end of the flow queue (line 15). These operations (and the removal of inactive flows from AFL at line 32) ensure that packets of flows emitting less than one quantum per round realize low packet latency.

3.1.2. Dequeue operation

The dequeue operation extracts the packet at the head of PQ whenever it is not empty (lines 17-20). When a packet is sent from PQ, the deficit counter of its flow is decreased by the packet size (line 21). This prevents more than one quantum being served during the same round. When PQ is empty and AFL contains at least one flow (line 22), the flow at the current head of the AFL cycle is selected for service and its deficit counter is incremented by one quantum (lines 23-24). The last instructions (lines 25-34) allow the flow to emit up to DC_i bytes. The flow is removed from the schedule if it empties (lines 32-34). Note that this will happen if the flow emitted less than its quantum in the current round or, in other words, if its rate is smaller than the current fair rate.

3.2. Measuring congestion

It is straightforward to detect congestion for elastic traffic by measuring the current value of the fair rate. This is generally only a small fraction of the link rate in situations of overload [1]. To do this we count the number of bits a fictitious permanently backlogged flow could emit in an interval of length t_f and divide by t_f. We also measure the load in the priority queue by averaging the emitted bit rate over a suitable interval t_p. Successive measurements are used to determine the admission control criteria to be applied to newly arriving flows (note that admission control requires an additional data structure recording the identities of flows currently in progress, independently of whether these are in AFL or not). Due to lack of space, we omit the details of these measurements.

3.3. Complexity

Unlike DRR, the active list in PDRR may contain flows with empty queues (these are flows whose packets are all forwarded via PQ). This implies that the dequeue operation can visit several empty queues before actually sending a packet, effectively invalidating the O(1) dequeue complexity. If necessary, this can be corrected by maintaining an AFL cycle as a linked list only for non-empty queues. This cycle is updated appropriately whenever a new flow receives more than its quantum in the initial round. It is necessary for this purpose to create additional pointers in the per-flow data of the AFL. We omit details.

The presence of a flow in the active list (line 5 in Table 3.2) can be detected in constant time (O(1) complexity) using a content addressable memory (CAM). This requires that size of AFL be small enough for hardware implementation. That this is the case is

[1] One effective policy is to drop the packet at the head of the flow with the longest backlog [11]

Table 1
Pseudo code for PDRR

Enqueuing module:	**Dequeuing module**:
on arrival of packet p	**While** (TRUE) do
If no free buffers left **then**	***While*** *(PQ not empty) do*
FreeBuffer();	*p = Dequeue(PQ);*
i = ExtractFlow(p);	*i = ExtractFlow(p);*
If ($i \notin AFL$)	*Send(p);*
InsertActiveList(i);	*DC_i -= Size(p);*
DC_i = 0;	**If** (AFL is not empty) **then**
ByteCount_i = size(p);	Get head of AFL, say flow i;
Enqueue(PQ,p);	DC_i += Q_i;
Else	**While** ((DC_i $\geq$ 0) and (Queue_i not empty)) do
ByteCount_i += size(p);	PacketSize = Size (Head(Queue_i));
If *(ByteCount_i $\leq$ Q_i)* ***then***	**If** (PacketSize $\leq$ DC_i) **then**
Enqueue(PQ,p);	Send(Dequeue(Queue_i));
Else	DC_i -= PacketSize;
Enqueue(Queue_i,p);	**Else**
	break; (*skip while loop*)
	RemoveActiveList(i);
	If *(Queue_i is not empty)* ***then***
	InsertActiveList(i);

demonstrated in the next section.

4. Performance evaluation

We have evaluated the performance of PDRR by means of *ns* simulations using both real Internet trace data and synthetic traffic. We report two sets of simulations, the first to evaluate the required size of AFL, the second to demonstrate the improved latency properties provided by the modified algorithm compared to DRR. In these simulations, all flows have the same quantum, equal to the value of the maximum size packet MTU.

4.1. Trace statistics

We have used three traces, as in [5]:
ADSL : an OC3 link concentrating the traffic outgoing to several thousand ADSL users; the trace represents 5 minutes of data recorded on August 25 2003;
Ab-I : an OC48 link on the Abilene research network between Indianapolis and Kansas City; the trace represents 5 minutes of data recorded on August 14 2002 (10:30 to 10:35 am);
Ab-III : an OC192 link on Abilene III between Indianapolis and Chicago; the trace represents 2 minutes of data recorded on June 1 2004 (7:31 to 7:33 pm).
The Abilene traces are publicly available on the NLANR website[2]. The ADSL data is from a commercial network. Summary statistics for these three traces are shown in Table 2. TCP connections contribute around 95% of bytes in all traces.

[2]http://pma.nlanr.net/Traces/Traces/long/ipls/1 and http://pma.nlanr.net/Special/ipls3.html

Table 2
Packet trace statistics summary

	ADSL	*Ab-I*	*Ab-III*
bandwidth	155Mbps	2.5Gbps	10Gbps
total packets	2.6M	19M	156M
total flows	850K	2.3M	683K
utilization	28%	13%	19%
flows in progress	24000	37000	62000
MTU	1500	1500	9000

4.2. Required AFL size

The active flow list must be dimensioned to ensure a small saturation probability. In case of saturation, packets belonging to newly active flows are handled with priority until the flow can be added to AFL. The only consequence is that such flows may momentarily emit at a rate higher than the current fair rate.

We performed simulations using the three traces described above with the link capacity reduced in order to attain loads of 0.6 and 0.9. The introduction of spurious effects due to the reaction of TCP congestion control, we provide sufficient buffering to avoid packet loss. As shown in Figure 1, the number of active flows is bounded with high probability to several hundred (550 in the worst case), much less than the number of flows in progress (see Table 2). These observations confirm results obtained using the PFQ scheduler [5].

We notice that the number of active flows is slightly greater with PDRR than with PFQ since the former removes non-backlogged flows from AFL somewhat later. However, the results confirm scalability in that the required AFL size does not increase with link speed. The size is also shown to be relatively small allowing efficient hardware implementation using a CAM.

An analytical explanation for these results is given in [**?**]. Succinctly, the number of simultaneous bottlenecked flows (those with peak rate greater than the fair rate) is small due to the stochastic behaviour of statistical bandwidth sharing [**?**] while the large number of non-bottlenecked flows only enter the AFL for the small fraction of time that they actually have a packet in the buffer.

4.3. Packet latency

We now show that PDRR ensures negligible packet delay for flows whose rate is less than the fair rate. We add five constant rate UDP connections to the Ab-I trace and adjust the link capacity (483 Mbps) to produce a load of 0.9. The rates of the UDP flows are 64 Kbps, 10, 20, 30 and 40 Mbps and they emit 1500 byte packets. The left plot of Figure 2 shows the distribution of packet delays for each of these flows. The lowest rate flows have very small delays since their packets are always given priority. The delay for the other flows increases with their rate. This is because the flows are sometimes backlogged due the random variations in the trace data.

The right hand figure shows the evolution of the average fair rate (exponentially smoothed 100 ms samples). Though this is consistently higher than 40 Mbps, the higher rate flows are occasionally backlogged due to very short term variations in the number of active flows. The crosses in the figure represent samples of the *instantaneous* fair rate derived by dividing one 1500 byte quantum by the duration of the current round. Backlog occurs

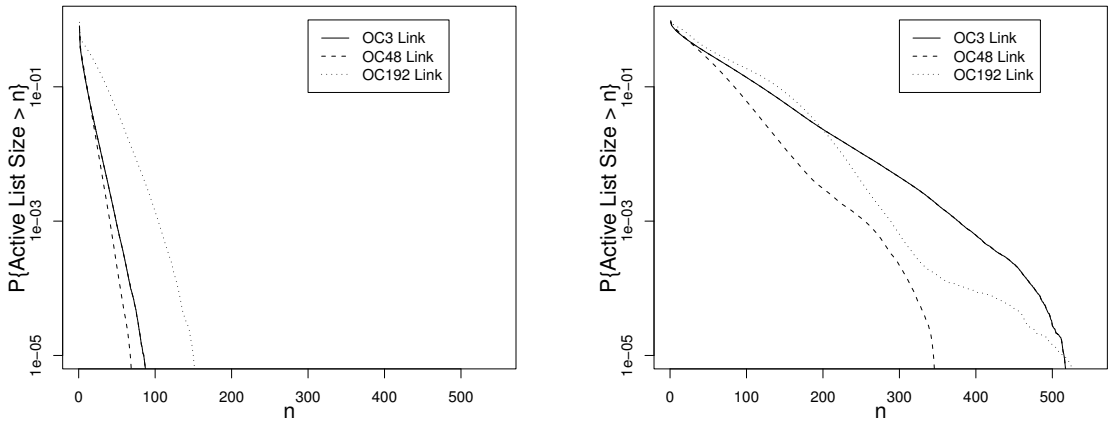

Figure 1. Complementary distribution of AFL size, loads 0.6 (left) and 0.9 (right)

whenever the round last longer than the inter-packet interval of a given flow.

Figure 3 compares packet latency of DRR and PDRR for low rate flows. The figure on the left relates to a mix of synthetic TCP and UDP traffic identical to that used in [6]. TCP connections emit 1000 byte packets and their size follows a truncated Pareto distribution with shape parameter 1.5, mean 25 packets, minimum 8 and maximum 1000. UDP connections of mean duration 1 minute emit on-off traffic with 64Kbps peak rate and 500 ms mean exponentially distributed burst and silence lengths. TCP flows count for 80% of total load. Link capacity is 10 Mbps and arrival rates are set to realize a load of 9 Mbps. The figure shows the delay distribution of UDP packets. The figure on the right relates to the Ab-I trace data augmented by a single 64 Kbps CBR flow and depicts the packet delay distribution of the latter. The link load is again 0.9 for a link capacity of 361 Mbps.

The relative difference between results for PDRR and DRR is similar in both cases. However, the absolute delays clearly depend on the link speed. The advantage of using PDRR is slight for the high capacity link where regular DRR would be sufficient for most streaming applications. Note that the delay with DRR depends on the number of flows in AFL. The delay is limited since this number is relatively small with high probability, as discussed in the previous section.

Packet delay in the priority queue can be predicted quite well if the priority load is known. Dotted lines in Figure represent the complementary delay distribution of an M/M/1 queue with mean packet size and priority load derived from the simulation results[3]. Latency can thus be controlled by ensuring the short term load due to priority packets is not greater than some threshold. This can be achieved using per-flow admission control, as envisaged in Section 2.2.

5. Service protection in overload

The previous results demonstrate that PDRR provides low latency and/or high throughput as required up to loads of 90%. In this section we illustrate the role of admission

[3]It would of course be possible to use a more refined model if precise latency guarantees are necessary

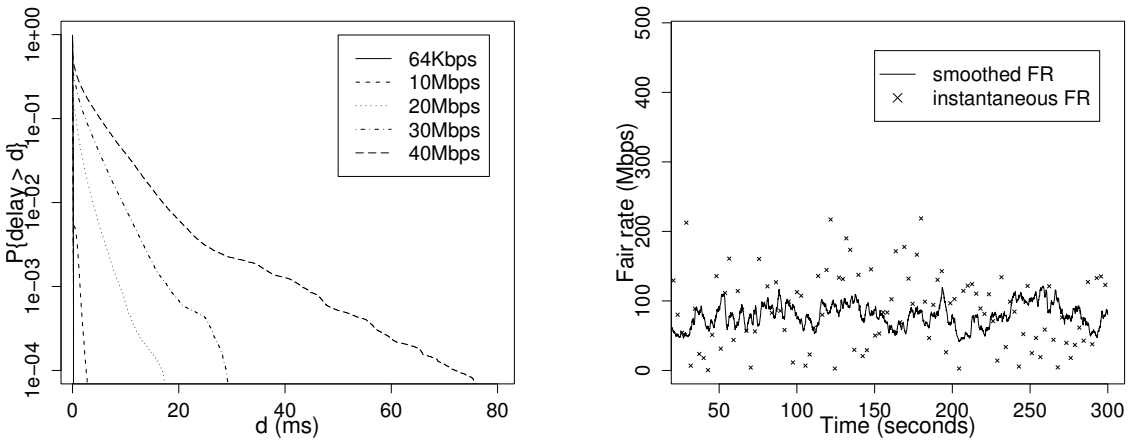

Figure 2. CBR flows in addition to Ab-I traffic, complementary distribution of packet delays and evolution of fair rate

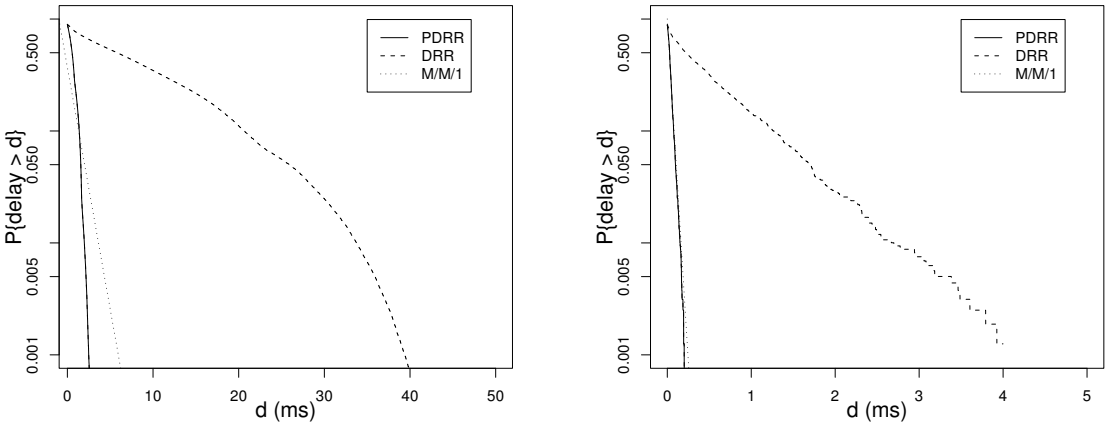

Figure 3. Comparison between complementary distribution of packet delays for 64Kbps CBR flows with PDRR and DRR, synthetic traffic (left) and Ab-I trace (right)

control in maintaining satisfactory performance in overload. Overload occurs whenever demand (flow arrival rate x average flow size) exceed link capacity[4].

We perform simulations with artificial traffic since, with trace data recorded on a congestion free link, it is impossible to account for the way TCP would react to inevitable packet loss. The simulation set-up is that of Section 4 with a mix of TCP and UDP flows. The arrival rate is increased to produce an offered load of 11 Mbps. The link rate is 10 Mbps, overall buffer capacity is 1000 packets and the AFL holds data for a maximum of 500 flows. The system starts empty. We have simulated the system with and without applying admission control.

When admission control is activated, flows are rejected (i.e., their packets are discarded) whenever the fair rate estimate is below 100 Kbps. The estimate is derived by exponen-

[4]Overload may also be said to occur at loads less than 100% but greater than some nominal threshold of 90%, say. The precise definition of overload in fact depends on the way admission control is implemented.

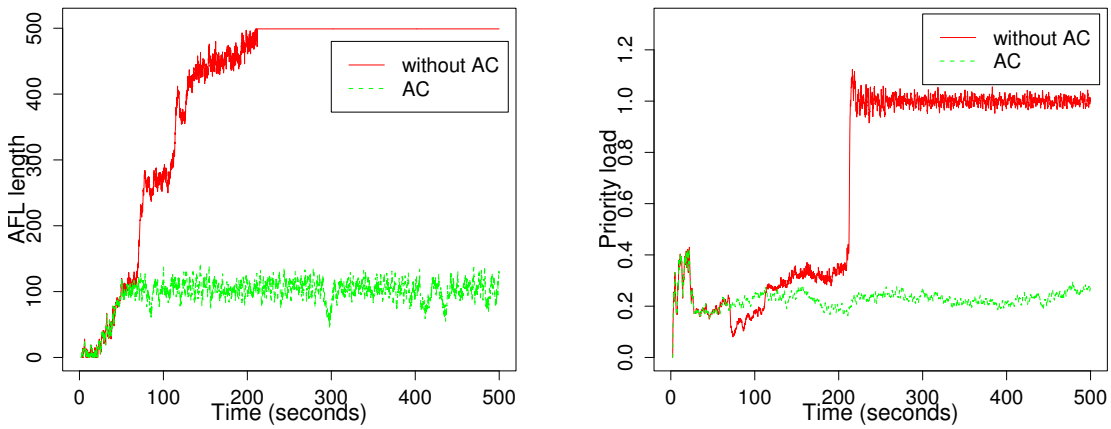

Figure 4. AFL length (left) and priority load (right) variations vs. time, synthetic traffic under overload

tially averaging fair rate samples evaluated every 100 ms using a smoothing parameter of 0.9^5. For the considered traffic mix, this is the only admission criterion necessary since the load in the priority queue is always quite low (around 0.25).

Figure 4 shows the variation in time of the number of AFL flows (left) and the load in the priority queue (right), with and without admission control. The simulation results are identical until 60 s. At this point the fair rate decreases to 100 Kbps for the first time. Without admission control the decrease continues as the number of flows competing for bandwidth grows. The AFL population increases until, at around 210 s, it saturates. From this point on, all new packets join the priority queue which rapidly saturates. This queue never empties so that no flow in the AFL is ever served and all "priority" packets suffer near maximum delay.

Admission control effectively protects the system from such drastic performance degradation. The left plots of Figure 4 show that the fair rate is maintained close to the admission threshold. The load of the priority queue is around 25%. Other results (not shown) confirm that the link is nearly fully utilized with a flow blocking probability of 10%. The delay of UDP flow packets is 0.69 ms on average while TCP flows realize an average throughput of 129 Kbps (without admission control, packet delay is more than 400 ms and TCP throughput tends to zero).

6. Conclusion

We have shown through the results of simulation that PDRR is scalable in that the number of flows to be scheduled does not increase with link rate. Implementation appears perfectly feasible since with high probability this number is relatively small (i.e., no more than a few hundreds) for loads up to 90% of link capacity.

In overload, it is necessary to apply per-flow admission control in order to preserve good performance for admitted flows. Note that no scheduler can avoid drastic performance degradation when offered traffic exceeds capacity. PDRR has the advantage of allowing

[5] i.e., new = 0.9 × old + 0.1 × measurement

simple measurement of the relevant congestion parameters. Our simulation results show that latency and throughput are effectively maintained by admission control.

PDRR discriminates between flows on the basis of their incoming rate. Bottlenecked flows are guaranteed the current max-min fair rate. Packets of flows whose rate is less than the fair rate are transmitted in a priority queue. This is a useful key for discrimination since streaming flows generally do have a relatively low peak rate. The absolute improvement in latency may be negligible in the case of a very high speed link. Latency with regular DRR is then low since the number of flows in any round is bounded and each quantum of service is measured in microseconds. On the other hand, PDRR is hardly more complex than DRR.

We believe the notion of implicit service differentiation and its realization using PDRR opens interesting possibilities for traffic control in IP networks. It remains to more fully evaluate the scope for implementation taking account notably of the significant differences between the edge and the core of the IP internet.

Since different variants of DRR are already deployed in some high speed routers, we believe that PDRR is simple to implement. It requires only a few additional instructions.

REFERENCES

1. N. Benameur, S. Ben Fredj, S. Oueslati, J. Roberts, Quality of service and flow level admission control in the Internet, Computer Networks, Vol 40, pp 57-71, 2002.
2. D. Bertsekas and R. Gallager. Data Networks. Prentice-Hall, 1987.
3. E. L. Hahne, Round-Robin Scheduling for Max-Min Fairness in Data Networks, IEEE Journal on Selected Areas in Communications 9(7): 1024-1039, 1991.
4. A. Kortebi, L. Muscariello, S. Oueslati, J. Roberts, On the Scalability of Fair Queuing,In Proc. of ACM HotNets III, San Diego, 2004.
5. A. Kortebi, L. Muscariello, S. Oueslati, J. Roberts, Evaluating the Number of Active Flows in a Scheduler Realizing Statistical fair Bandwidth Sharing, to appear in ACM SIGMETRICS, Banff, 2005.
6. A. Kortebi, S. Oueslati and J. Roberts. Cross-protect: Implicit Service Differentiation and Admission Control, In. Proc. of HPSR, Phoenix, 2004.
7. S. Oueslati, J. Roberts, A new direction for quality of service: Flow-aware networking, Proceedings of NGI 2005, Rome, June 2005.
8. J. Roberts, Internet Traffic, QoS and Pricing, Proceedings of the IEEE,Vol 92,No 9,pp 1389-1399, Sept. 2004.
9. J. Roberts, U. Mocci and J. Virtamo. Broadband Network Teletraffic (Final Report of COST 242). LNCS 1155 Springer Verlag, 1996.
10. M. Shreedhar, G. Varghese, Efficient Fair Queuing Using Deficit Round Robin, IEEE/ACM Transactions on Networking, Vol.4, No.3, June 1996.
11. B. Suter, T. Lankshman, D. Stiliadis, A. Choudhury, Buffer Management Schemes for Supporting TCP in Gigabit Routers with Per-Flow Queuing, IEEE Journal in Selected Areas un Communications, August 1999.