Full Exploitation of the Deficit Round Robin Capabilities by Efficient Implementation and Parameter Tuning

Technical Report, October 2003

L. Lenzini, E. Mingozzi, G. Stea
Dipartimento di Ingegneria della Informazione, University of Pisa
Via Diotisalvi 2, 56126 Pisa – Italy
{l.lenzini, e.mingozzi, g.stea}@iet.unipi.it

***Abstract*—Deficit Round Robin (DRR) is a scheduling algorithm devised for providing fair queueing in the presence of variable length packets. The main attractive feature of DRR is its simplicity of implementation: in fact, it can exhibit $O(1)$ complexity, provided that specific allocation constraints are met. However, according to the DRR implementation proposed in [7], meeting such constraints often implies tolerating high latency and poor fairness. In this paper, we first derive new and exact bounds for DRR latency and fairness. On the basis of these results, we then propose a novel implementation technique, called Active List Queue Method (Aliquem), which allows a remarkable gain in latency and fairness to be achieved, while still preserving $O(1)$ complexity. We show that DRR achieves better performance metrics than those of other round robin algorithms such as Pre-Order Deficit Round Robin and Smoothed Round Robin. We also show by simulation that the proposed implementation allows the average delay and the jitter to be reduced.**

Keywords—DRR; scheduling algorithms; Quality of Service

I. INTRODUCTION

Multi-service packet networks are required to carry traffic pertaining to different applications, such as e-mail or file transfer, which do not require pre-specified service guarantees, and real-time video or telephony, which require performance guarantees. The best-effort service model, though suitable for the first type of applications, is not so for applications of the other type. Therefore, multi-service packet networks need to enable Quality of Service (QoS) provisioning. A key component for QoS enabling networks is the scheduling algorithm, which selects the next packet to transmit, and when it should be transmitted, on the basis of some expected performance metrics. During the last decade, this research area has been widely investigated, as proved by the abundance of literature on the subject (see [2-18]). Various scheduling algorithms have been devised, which exhibit different fairness and latency properties at different worst-case per-packet complexities. An algorithm is said to have $O(1)$ *worst-case per packet complexity* (hereafter *complexity*) if the number of operations needed to select the next packet to be transmitted is *constant* with respect to the number of active flows.

The existing work-conserving scheduling algorithms are commonly classified as *sorted-priority* or *frame-based*. Sorted-priority algorithms associate a timestamp with each queued packet and transmit packets by increasing timestamp order. On the other hand, frame-based algorithms divide time into frames, and select which packet to transmit on a per-frame basis. Within the frame-based class, round-robin algorithms service the various flows cyclically, and within a round each flow is serviced for up to a given *quantum*, so that the frame length can vary up to a given maximum. Sorted-priority algorithms generally exhibit better latency and fairness properties compared to round-robin ones, but have a higher complexity, due to the calculation of the timestamp and to the sorting process [3]. Specifically, the task of sorting packets from N different flows has an intrinsic complexity of $O(\log N)$ [1]. At high link speeds, such a complexity may limit the scalability. Simpler algorithms requiring a constant number of operations per packet transmission would then be desirable. Round-robin scheduling algorithms, instead, can exhibit $O(1)$ complexity.

Deficit Round Robin (DRR) [7] is a scheduling algorithm devised for providing fair queueing in the presence of variable length packets. Recent research in the DiffServ area [19] proposes it as a feasible solution for implementing the Expedited Forwarding Per-hop Behavior [20,21]. According to the implementation proposed in [7], DRR exhibits $O(1)$ complexity *provided that* each flow is allocated a quantum no smaller than its maximum packet size. As observed in [16], removing this hypothesis would entail operating at a complexity which can be as large as $O(N)$. On the other hand, in this paper we show that meeting such a constraint may lead to poor performances if flows with very different rate requirements are scheduled. This is because a frame can be very large under the above constraint, and this in turn implies poor fairness and longer delays.

The purpose of this paper is twofold. Firstly, we propose a novel implementation technique, called Active List Queue Method (*Aliquem*), which allows DRR to operate at $O(1)$ complexity even if quanta are *smaller* than the maximum packet size. More specifically, the Aliquem implementation allows DRR to operate at $O(1)$ complexity, provided that each flow is allocated a quantum no smaller than its maximum packet size *scaled by a tunable parameter* q. By appropriately selecting the q value, it is then possible to make a trade-off between operational overhead and performance. We propose different solutions for implementing Aliquem, employing

[1] It has been shown that this task can be performed at $O(\log \log n)$ complexity if coarsened timestamps are used [8].

different data structures and requiring different space occupancy and operational overhead. In addition, we present a variant of the Aliquem technique, called Smooth Aliquem, which further reduces the output burstiness at the same complexity.

However, in order to fully understand the performance gains which are achieved by means of the Aliquem implementation, a careful analysis of the DRR performance – specifically of its latency and fairness – is required. In fact, the results related to DRR reported in the literature only apply to the case in which each flow is allocated a quantum no smaller than its maximum packet size [3-4], [7]. Therefore, as a second issue, we also provide a comprehensive analysis of the latency and fairness bounds of DRR, presenting new and exact results. We show that DRR achieves better performance metrics than those of other round robin algorithms such as Pre-Order Deficit Round Robin (PDRR) [13] and Smoothed Round Robin (SRR) [14]; we also compare our implementation with Self-Clocked Fair Queuing (SCFQ) [6], which is a sorted-priority algorithm. Finally, we report simulation results showing that the Aliquem and Smooth Aliquem techniques allow the average delay and the jitter to be reduced.

The rest of the paper is organized as follows. Section II gives the results related to the DRR latency and fairness. The Aliquem implementation is described in Section III and analyzed in Section IV, whilst in Section V we describe the Smooth Aliquem DRR. We compare Aliquem DRR with previous work in Section VI. We show simulation results in Section VII, and draw conclusions in Section VIII.

II. DRR Latency and Fairness Analysis

In this section we briefly recall the DRR scheduling algorithm and the proposed implementation. We then give generalized results related to the latency and fairness properties of DRR, and propose guidelines for selecting parameters.

A. DRR operation and implementation complexity

Deficit Round Robin is a variation of Weighted Round-Robin (WRR) that allows flows with variable packet lengths to share the link bandwidth fairly. Each flow i is characterized by a *quantum* of ϕ_i bits, which measures the quantity of packets that flow i should ideally transmit during a round, and by a *deficit* variable Δ_i. When a backlogged flow is serviced, a burst of packets is allowed to be transmitted of an overall length not exceeding $\phi_i + \Delta_i$. When a flow is not able to send a packet in a round because the packet is too large, the number of bits which could not be used for transmission in the *current* round is saved into the flow's deficit variable, and are therefore made available to the same flow in the *next* round.

More specifically, the deficit variable is managed as follows:

- reset to zero when the flow is not backlogged;
- increased by ϕ_i when the flow is selected for service during a round;
- decreased by the packet length when a packet is transmitted.

Let $\overline{L_i}$ be the maximum length of a packet for flow i (measured in bits). In [7] the following inequality has been proved to hold right after a flow has been serviced during a round:

$$0 \le \Delta_i < \overline{L_i} \tag{1}$$

which means that a flow's deficit never reaches the maximum packet length for that flow.

DRR has $O(1)$ complexity under certain specific conditions. Let us briefly recall the DRR implementation proposed in [7] and discuss those conditions. A FIFO list, called the *active list*, stores the references to the backlogged flows. When an idle flow becomes backlogged, its reference is added to the tail of the list. Cyclically, if the list is not empty, the flow which is at the head of the list is dequeued and serviced. After it has been serviced, if still backlogged, the flow is added to the tail of the list. It is worth noting that:

- dequeueing and enqueueing flows in a FIFO list are $O(1)$ operations;
- since backlogged flows are demoted to the tail of the list after reaching the head of the list and receiving service, the relative service order of any two backlogged flows is preserved through the various rounds.

It has been proved in [7] that the following inequality must hold for DRR to exhibit $O(1)$ complexity:

$$\forall i,\ \phi_i \ge \overline{L_i}\ . \tag{2}$$

Inequality (2) states that each flow's quantum must be large enough to contain the maximum length packet for that flow. If some flows violate (2), a flow which is selected for transmission (i.e. dequeued from the head of the list), though backlogged, may not be able to transmit a packet during the current round. Nevertheless, its deficit variable must be updated and the flow must be queued back at the tail of the list. In a worst case, this can happen as many consecutive times as the number of flows violating (2). Therefore, if (2) does not hold, the number of operations needed to transmit a packet in the worst case can be as large as $O(N)$ [16].

B. Latency Analysis

Let us assume that DRR is used to schedule packets from N flows on a link whose capacity is C. Let

$$F = \sum_{i=1}^{N} \phi_i \tag{3}$$

denote the *frame length*, i.e. the number of bits that should ideally be transmitted during a round if all flows were backlogged.

DRR has been proved to be a *latency-rate server* (LR server) [3]. An LR server is characterized by its *latency*, which may be regarded as the worst-case delay experienced by the first packet of a flow busy period, and by its *rate*, i.e. the guaranteed service rate of a flow.

The following expression has been derived as the DRR latency in [4]:

$$\Theta_i^* = \frac{3F - 2\phi_i}{C}. \quad (4)$$

However, (4) was obtained under the assumption that quanta are selected equal to the maximum packet length, i.e. $\phi_j = \overline{L_j} \ \ \forall j$. As a consequence, it does not apply to cases in which the above hypothesis does not hold, as in the following example:

Example – Suppose that N flows with maximum packet lengths $L_1 = L_2 = ... = L_N = L$ are scheduled, and that $\phi_j = L/100$, $j = 1...N$. According to (4) the latency should be:

$$\Theta_i^* = \frac{3F - 2\phi_i}{C} = \frac{(3N-2)\overline{L}/100}{C}.$$

However, a packet can actually arrive at a (previously idle) flow i when every other flow in the active list has a maximum length packet ready for transmission and a deficit value large enough to transmit it. Therefore the delay bound for a packet that starts a busy period for flow i, is no less than $(N-1)\cdot \overline{L}/C$, i.e., much higher than (4).

To the best of our knowledge, no general latency expression for DRR has yet been derived. The following result is proved in the Appendix.

Theorem 1

"The latency of DRR is

$$\Theta_i = \frac{1}{C}\left[(F - \phi_i)\left(1 + \overline{L_i}/\phi_i\right) + \sum_{j=1}^{N} \overline{L_j} \right]." \quad (5)$$

Note that, when $\phi_j = \overline{L_j} \ \ \forall j$, it is $\Theta_i = \Theta_i^*$.

Another common figure of merit for a scheduling algorithm is the *start-up latency*, defined as the upper bound on the time it takes for the last bit of a flow's head-of-line packet to leave the scheduler. Such a figure of merit is less meaningful than latency, since it does not take into account the correlation among the delays of a sequence of packets in a flow, but it is generally easier to compute. Moreover, computing the start-up latency allows DRR to be compared with scheduling algorithms not included in the LR servers category, such as SRR. We prove the following result:

Theorem 2

"The start-up latency of DRR is

$$S_i = \frac{1}{C}\left[(F - \phi_i)\cdot \left\lceil \frac{\overline{L_i}}{\phi_i} \right\rceil + \sum_{j=1}^{N} \overline{L_j} \right]." \quad (6)$$

Proof

A head-of-line packet of length L_i will be serviced after flow i receives exactly $ceiling(L_i/\phi_i)$ service opportunities. Meanwhile, all other flows are serviced for $ceiling(L_i/\phi_i)$ times as well. It has been proved in [7] that the following inequality bounds the service received by a backlogged flow which is serviced m times in $(t_1, t_2]$:

$$m\phi_i - \overline{L_i} < W_i(t_1, t_2) < m\phi_i + \overline{L_i}. \quad (7)$$

Thus, the maximum service that each flow $j \neq i$ can receive in $ceiling(L_i/\phi_i)$ visits is upper bounded by $ceiling(L_i/\phi_i)\phi_j + L_j$. Therefore, a packet of length L_i will leave the scheduler after at most:

$$\frac{1}{C}\left[\left\lceil \frac{\overline{L_i}}{\phi_i} \right\rceil \cdot \sum_{j=1..N, j\neq i} \phi_j + \sum_{j=1..N, j\neq i} \overline{L_j} + L_i \right] \quad (8)$$

have elapsed since it became the head-of-line packet for flow i. By considering that $\overline{L_i}$ bounds the packet length for flow i and substituting the definition of frame (3) into (8), we straightforwardly obtain (6).

■

C. Fairness Analysis

DRR has been devised as a scheduling algorithm for providing fair queueing. A scheduling algorithm is fair if its *fairness measure* is bounded. The fairness measure was introduced by Golestani [6], and may be seen as the maximum absolute difference between the normalized service received by two flows over any time interval in which both are backlogged.

Let us assume that DRR schedules N flows requiring rates $\rho_1, \rho_2, ..., \rho_N$ on a link whose capacity is C, with $\sum_i \rho_i \leq C$. Let us denote with $W_i(t_1, t_2)$ the service received by flow i during the time interval $(t_1, t_2]$. The *fairness measure* is given by the following expression

$$FM_{i,j} = \frac{1}{C} \max_{t_2 > t_1} \left| \frac{W_i(t_1, t_2)}{\rho_i / \sum_h \rho_h} - \frac{W_j(t_1, t_2)}{\rho_j / \sum_h \rho_h} \right|.$$

As a consequence of (7), if we want flows to share the link bandwidth fairly, the ratio of any two flows' quanta must then match their required rate ratio. Therefore, the following set of equalities constrains the choice of the quanta:

$$\phi_i / \phi_j = \rho_i / \rho_j \ \ \forall i, j, \ i \neq j. \quad (9)$$

A fair system would thus be one for which the fraction of the frame length for which a flow is serviced is equal to the fraction of the link bandwidth the flow requires. Let us define

$$f_i \triangleq \phi_i / \sum_{j=1..N} \phi_j = \phi_i / F. \quad (10)$$

Therefore, the following equality follows from (9) and (10):

$$f_i = \rho_i \Big/ \sum_{j=1..N} \rho_j \,. \tag{11}$$

It can be easily shown that the fairness measure for DRR is

$$FM_{i,j} = \frac{1}{C}\left(F + \frac{\overline{L_i}}{f_i} + \frac{\overline{L_j}}{f_j} \right). \tag{12}$$

The proof is obtained by considering the following worst-case scenario (see Lemma 2 in [7]):

- in $(t_1, t_2]$, flow i is serviced *one more time* than flow j, say m times against $m-1$;
- $W_i(t_1,t_2) = m\phi_i + \overline{L_i}$, i.e. flow i has the maximum deficit at time t_1 and no deficit at time t_2;
- $W_j(t_1,t_2) = (m-1)\phi_j - \overline{L_j}$, i.e. flow j has no deficit at time t_1 and the maximum deficit at time t_2.

By substituting the above expressions for $W_i(t_1,t_2)$ and $W_j(t_1,t_2)$ in the fairness measure, and keeping into account (10) and (11), expression (12) follows straightforwardly.

D. Parameter Selection

A known DRR problem (which is also common to other round-robin schedulers) is that the latency and fairness depend on the frame length. The longer the frame is, the higher the latency and fairness are. In order for DRR to exhibit lower latency and better fairness, the frame length should therefore be kept as small as possible. Unfortunately, given a set of flows, it is not possible to select the frame length arbitrarily if we want DRR to operate at $O(1)$ complexity. In fact, inequality (2) establishes a lower bound for each quantum in order for DRR to operate in the $O(1)$ region. Assuming that fairness is a key requirement, and therefore (11) holds, in order for (2) to hold, a frame cannot be smaller than

$$F^S = \max_{j=1..N}\left(\overline{L_j} \Big/ f_j \right). \tag{13}$$

It is straightforward to see that if the frame length is F^S, there is at least one flow for which equality holds in (2). Therefore, operating with a frame smaller than F^S implies not operating at $O(1)$ complexity.

Even if the frame length is selected according to (13), it can be very large in practical cases, as shown by the following numerical example. Suppose that 20 data flows requiring 50 Kbps each share a 10 Mbps link with 9 video flows requiring 1 Mbps each. The maximum packet length is $\overline{L} = 1500$ bytes for all flows. According to our scheme, we obtain $F^S = 300$ Kbytes (240 ms at 10 Mbps), and each video flow has a quantum of 30 Kbytes, i.e. is allowed to transmit a burst of 20 maximum length packets in a round. The latency for a video flow is 262 ms, and the latency for a data flow is 512.4 ms.

As the above example shows, when DRR is used to schedule flows with very different rate requirements, quanta and frame length can be very large, thus leading to high latencies for all flows and bursty transmissions for the flows with high rates. In the next section, we present an implementation that allows the frame length to be reduced without the drawback of operating at $O(N)$ complexity.

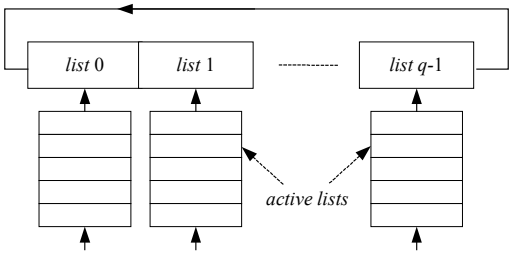


Figure 1. Active List Queue.

III. The Aliquem Implementation

The Active List Queue Method (Aliquem) is an implementation technique that allows DRR to obtain better delay and fairness properties preserving the $O(1)$ complexity. In order to distinguish between the Aliquem-based implementation of DRR and the implementation proposed in [7] and described in the previous section, we will call the former Aliquem DRR and the latter Standard DRR. The rationale behind Aliquem DRR is the following: suppose that the frame length is selected as $F = \alpha F^S$, $0 < \alpha < 1$. This implies that there are flows violating (2). Let flow i be one of those. As already stated, if a *single* FIFO active list is used to store references to the backlogged flows, it is possible that i is dequeued and enqueued several times before its deficit variable becomes greater than the head packet length. However, by only looking at flow i's quantum, deficit and head-packet length, it is possible to state in advance how many rounds will elapse before flow i actually transmits its head packet, and we can defer updating flow i's deficit variable until that time. Specifically, let L_i be the length of the head packet which is at the head of flow i's queue right after the flow has been serviced in a round, and let Δ_i be the deficit value at that time. We can assert that the head packet will be transmitted in the R-th subsequent round, where R is computed as follows

$$R = \left\lceil \frac{L_i - \Delta_i}{\phi_i} \right\rceil \tag{14}$$

and the deficit right before the head packet transmission is started will be

$$R \cdot \phi_i + \Delta_i \,. \tag{15}$$

Note that (14) and (15) can also be applied when the packet arrives at a previously idle flow; in this case the flow's deficit when the arriving packet reaches the head of the flow's queue is null.

Aliquem DRR avoids unnecessary enqueueing, dequeueing and deficit updating by taking (14) and (15) into account. Instead of considering a single FIFO active list, we consider the data structure shown in Figure 1. A circular queue of $q > 1$ elements contains the head and tail references to q different FIFO active lists. Each active list contains references to

```
PacketArrival (packet p, flow I) {
   if ( NotBacklogged(I) ) {
      Δ_I= 0;
      R=ceiling (Length(p)/φ_I);
      g = (CurrentList+R) mod q;
      EnqueueFlow(I, ActiveList(g));
   }
   EnqueuePacket(p, FlowQueue(I));
}
AliquemDRRScheduler () {
   while (SystemBusy) {
      while ( NotEmpty(CurrentList) ) {
         I = DequeueFlow(CurrentList);
         L = Length(HeadPacket(I));
         Δ_I = ceiling((L-Δ_I)/φ_I)*φ_I + Δ_I;
         loop {
            L=Length(HeadPacket(I));
            if (L==0) {
               Backlogged=false;
               exit loop;
            }
            if (L>Δ_I) {
               Backlogged=true;
               exit loop;
            }
            p=DequeueHeadPacket(I);
            TransmitPacket(p);
            Δ_I = Δ_I - L;
         }
         if (Backlogged) {
            R = ceiling((L-Δ_I)/φ_I);
            g = (CurrentList+R) mod q;
            EnqueueFlow(I, ActiveList(g));
         }
      }
      CurrentList=NextNonEmptyList();
   }
}
```

Figure 2. Pseudo-code for Aliquem DRR.

backlogged flows. During round k, only the flows whose reference is stored in the $(k \bmod q)$-th active list (the *current* active list) are considered for transmission, and they transmit their packets according to the DRR pseudo-code.

When a flow has completed service, if still backlogged, it is queued back in *another* active list, specifically the one that will become the current list in the round in which the flow can actually transmit its head packet. The correct active list g where flow i must be enqueued is located as follows:

$$g = (k + R) \bmod q \,. \tag{16}$$

In order for this implementation to work, it is mandatory that, for any possible values of L_i, Δ_i and ϕ_i, $R < q$. By taking (1) into account, this requirement translates to the following constraints on the quanta length:

$$\forall i,\ \phi_i \geq \frac{\overline{L_i}}{q-1}. \tag{17}$$

The pseudo-code for Aliquem DRR is reported in Figure 2. On a packet arrival to an empty flow, the flow – whose initial deficit is null – must be inserted into the Aliquem data structure: this is done by applying (14) and (16). If the flow was already backlogged, no action needs to be taken (apart from enqueueing the incoming packet in the flow's packet queue, of course). While there are packets in the system, the following steps are performed cyclically:

- as long as the current active list is not empty, the head flow is dequeued: its deficit is updated according to (15), and its packets are transmitted until either the flow is empty or its deficit is not sufficient to transmit the next packet;
- if the flow which has just been serviced is still backlogged, it is inserted into another active list located by applying (14) and (16); otherwise its reference is simply discarded;
- when the current list has been emptied (i.e. all packets which were to leave during the current round have been transmitted), the next non-empty active list from which to dequeue flows, i.e. the active list to be selected as the next current list, has to be located. The function `NextNonEmptyList()` is intentionally left unspecified for the moment. In the following section we will propose two different implementations for it, which yield different worst-case and average complexity and require different space occupancy.

IV. ALIQUEM DRR ANALYSIS

A. Operational Overhead and Space Occupancy

The space occupancy introduced by Aliquem DRR is quite modest. The active lists pool can contain at most N flow references (the same as Standard DRR). The circular queue can be implemented by a vector of $2 \cdot q$ flow references (head and tail of an active list for each element). Thus, the only space overhead introduced by Aliquem is $2 \cdot q \cdot$ `sizeof(reference)`.

Suppose that Aliquem DRR is servicing flows from the current active list k ($0 \leq k < q$). It is straightforward to see that dequeueing a flow from the current list and updating its deficit variable are $O(1)$ operations. All flows referenced in the current list are backlogged and, when dequeued, their deficit is large enough to transmit (at least) the head packet. Locating the correct active list in which to enqueue a flow which has just been serviced (or has just become backlogged) and enqueueing it are $O(1)$ operations as well.

When the flows in the current active list have all been serviced, the next non-empty active list must be located, and this task may require some overhead. However, it is easy to see that the overhead involved in it only depends on the number of elements in the circular queue q, and *does not depend* on the number of flows N. Therefore, the complexity of Aliquem DRR does not depend on the number of flows. If no specific data structure for locating the next non-empty list is employed, Aliquem DRR might have to perform $q-1$ (i.e., $O(q)$) operations per packet transmission in the worst case, since a forward linear search of the circular queue until a non-null active list head reference is found must be accomplished. However, we observe that the *average* number of operations per packet transmission is expected to be considerably lower, since

```
int NextNonEmptyList() {
   Scanned = 0;
   loop  {
      CurrentList = (CurrentList+1) mod q;
      if NotEmpty(CurrentList)
         return CurrentList;
      Scanned++;
      if (Scanned==q) {
         SystemBusy = false;
         return 0;
      }
   }
}
```

Figure 3. Pseudo-code for the `NextNonEmptyList()` function, implementation by linear searching.

```
int NextNonEmptyList() {
   x = successor(CurrentList,VEBqueue);
   if (x == ∞)
      x = successor(-1,VEBqueue);
   delete(x,VEBqueue);
   return x;
}
```

Figure 4. Pseudo-code for the `NextNonEmptyList()` function, implementation by a VEB-PQ.

`NextNonEmptyList()` is called once per round, and a round may include several packet transmissions. The pseudo-code for this implementation of the `NextNonEmptyList()` function is reported in Figure 3.

If q is large, $O(q)$ operations in a packet transmission time could represent a burden at high link speeds. Below, we describe two different additional data structures which allow us to reduce the complexity of locating the next non-empty list in Aliquem DRR.

1) Van Emde Boas Priority Queue

The Van Emde Boas priority queue (VEB-PQ) [22] is used to sort a finite set of integers U. Let $y \in U$ and let $X \subset U$ be the subset currently maintained by the priority queue. Three operations are defined on the VEB-PQ:

- `insert(y,X)`: inserts the element y into the VEB-PQ;
- `delete(y,X)`: extracts the element y from the VEB-PQ;
- `successor(y,X)`: returns the smallest element larger than y in the VEB-PQ; if y is the largest element, returns a special symbol ∞.

The following Lemma holds [22]:

VEB-PQ Lemma

"Let $U = \{0,1,2,...,L\}$, let $y \in U$ and let $X \subset U$ be a subset. The operations `insert(y,X)`, `delete(y,X)` and `successor(y,X)` can be implemented in time $O(\log\log L)$ each. The priority queue can be initialized in time $O(L\log\log L)$ using $O(L\log\log L)$ space."

More recent research on the VEB-PQ shows that:

- It is possible to reduce the space occupancy to $O(L)$ without increasing the operational overhead [23].
- By employing a non-standard (but practically implementable) memory model, the VEB-PQ operations `insert(y,X)`, `delete(y,X)` and `successor(y,X)` can be implemented in *constant* time [24].

In Aliquem DRR, a VEB-PQ can be used to store the indexes $\{0,1,2,...,q-1\}$ of the non-empty active lists. Thus, finding the next non-empty active list implies executing the pseudo-code reported in Figure 4. The two calls to `successor()` in the above procedure are due to the use of modular arithmetic in the active lists indexing. According to the VEB-PQ Lemma, this implementation of `NextNonEmptyList()` takes $O(\log\log q)$ operations. However, if we use a VEB-PQ, the `EnqueueFlow()` procedure may require $O(\log\log q)$ operations too (instead of $O(1)$), since it has to perform an `insert()` if the active list that the flow is enqueued in was empty. Moreover, a `delete()` operation must be performed whenever the current list is emptied out, before invoking the `NextNonEmptyList()` function.

2) Bit Vector Tree

Many off-the-shelf processors (including the Intel Pentium family) have machine instructions which allows one to locate the least (most) significant bit set to 1 in a word. This feature can be exploited in order to achieve a fast and lightweight implementation of the `NextNonEmptyList()` function.

Let X be the word dimension in bits, and let us assume $q \geq \mathrm{X}$ for ease of reading. We associate one bit in a set of $M = \lceil q/\mathrm{X} \rceil$ words with each list in the Aliquem structure, and assume that the bit is set if the corresponding list is non-empty. Thus, the M words can be regarded as a bit vector

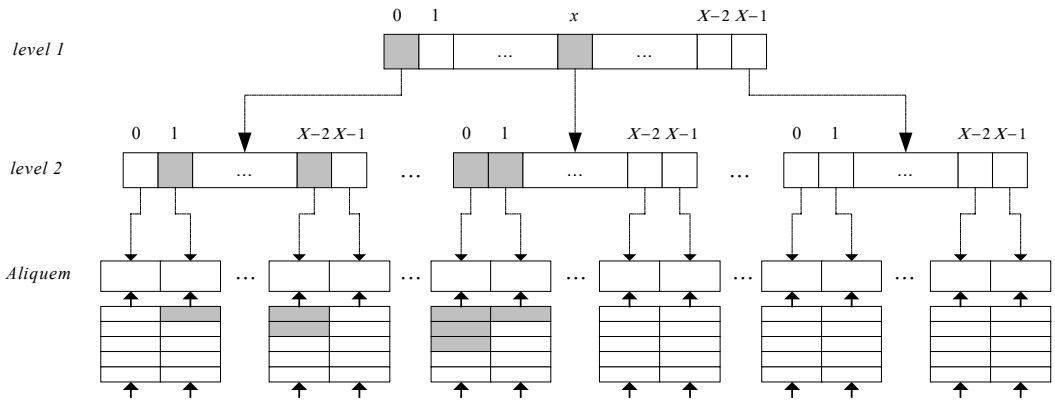


Figure 5 – Example of a 2-level Bit Vector Tree for accessing an Aliquem structure

containing information about the state of the q lists. Locating the least significant bit set to 1 in a given word takes *one* machine instruction; on the other hand, locating the correct word in a set of M can be efficiently accomplished by considering them as leaf nodes of a X -ary tree structure (as shown in Figure 5). In the X -ary tree, each non-leaf node is itself a word, whose x^{th} bit, $x = 0...\text{X}-1$, is set if there is at least one non-empty list in the node's x^{th} sub-tree.

Given an Aliquem structure with q active list, the X -ary tree structure has a $\lceil \log_{\text{X}} q \rceil$ depth, and therefore it is possible to locate the next non-empty list in $\lceil \log_{\text{X}} q \rceil$ time. As X ranges from 32 to 128 in today's processors, the operational overhead of such a data structure is negligible. The same can be said about the space occupancy of a X -ary tree, in which only a single *bit* is associated with each list at the leaf nodes.

As an example, consider a system with $\text{X} = 32$. It is possible to manage $q = 1K$ active lists with a 2-level tree, or $q = 32K$ active lists with a 3-level tree. The tree root can be stored directly in a register and the overall tree occupancy (128 bytes for a 2-level tree, ~4Kbytes for a 3-level tree) is negligible.

B. Latency

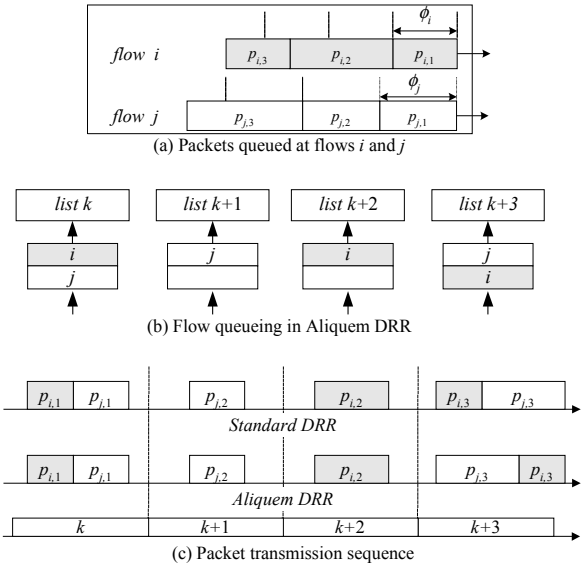
A side result of the proof process of Theorem 1 is that the latency of DRR does not change if it is implemented as Aliquem DRR. This is because Aliquem DRR just avoids polling flows that cannot transmit a packet during a round. This means that packets that are supposed to leave during a round in Standard DRR will leave during the same round in Aliquem DRR (though not necessarily in the same order, as we explain later on). Therefore, the scenario under which the Standard DRR latency is computed is also feasible for Aliquem DRR. Note that, in that particular scenario, packets from the tagged flow are the last to leave on each round. However, although the latency for Standard and Aliquem DRR is the same, it is not obtained at the same complexity. The lowest latency for Standard DRR operating at $O(1)$ complexity, achieved by selecting the frame length according to (13), is

$$\Theta_i^S = \frac{1}{C}\left[(1-f_i)\left(F^S + \overline{L_i}/f_i\right) + \sum_{j=1}^{N}\overline{L_j}\right]. \qquad (18)$$

Aliquem DRR allows the frame size to be reduced by a factor of $q-1$ with respect to the minimum frame length allowed by Standard DRR at $O(1)$ complexity. In order for Aliquem DRR to work, inequality (17) (rather than (2)) must hold. Inequality (17) allows quanta (and therefore frames) to be smaller by a factor of $q-1$ with respect to (2) The minimum frame length for Aliquem DRR is thus the following

$$F^A = \frac{F^S}{q-1} = \frac{1}{q-1}\ \max_{j=1..N}\left(\overline{L_j}/f_j\right). \qquad (19)$$

The lowest latency for Aliquem DRR with q active lists, obtained when the frame length is selected according to (19), is therefore:



(a) Packets queued at flows *i* and *j*

(b) Flow queueing in Aliquem DRR

(c) Packet transmission sequence

Figure 6. Inversion in the service order.

$$\Theta_i^A = \frac{1}{C}\left[(1-f_i)\left(\frac{F^S}{q-1} + \overline{L_i}/f_i\right) + \sum_{j=1}^{N}\overline{L_j}\right].$$

Whereas in Standard DRR latency Θ_i^A is only achievable when operating at $O(N)$ complexity, in Aliquem DRR latency Θ_i^A can be obtained at the cost of a much smaller complexity.

As far as the start-up latency is concerned, by manipulating (6) we obtain the following lower bound for Standard DRR at $O(1)$ complexity:

$$S_i^S = \frac{1}{C}\left[\left\lceil \frac{\overline{L_i}}{f_i \cdot F^S}\right\rceil \cdot F^S \cdot (1-f_i) + \sum_{j=1}^{N}\overline{L_j}\right] \qquad (20)$$

The lowest start-up latency for Aliquem DRR with q active lists, obtained when the frame length is selected according to (19), is thus:

$$S_i^A = \frac{1}{C}\left[\left\lceil \frac{\overline{L_i}\cdot(q-1)}{f_i \cdot F^S}\right\rceil \cdot \frac{F^S}{q-1} \cdot (1-f_i) + \sum_{j=1}^{N}\overline{L_j}\right] \qquad (21)$$

It can be easily shown that, when $q \geq 2$, $S_i^A \leq S_i^S$, and that the reduction in the start-up latency is non-decreasing with q. However, the amount of reduction in the start-up latency also depends on the value of $\overline{L_i}$ and f_i: specifically, the smaller $\overline{L_i}/f_i$ is with respect to F^S, the higher the start-up latency reduction is for flow i. In the case in which $\overline{L_i}/f_i = F^S$, we have $S_i^A = S_i^S$ for any value of q.

C. Fairness

In Standard DRR, since flows are inserted in and extracted from a unique FIFO active list, the service sequence of any

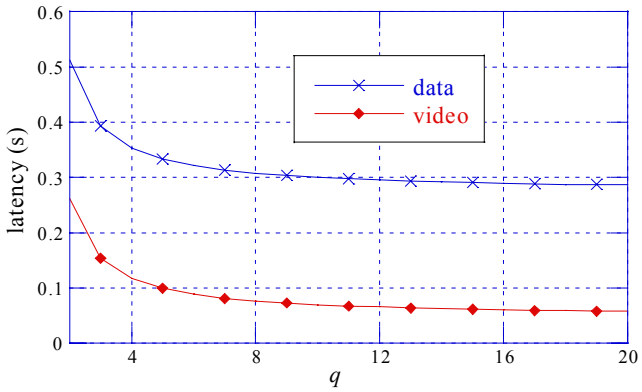


Figure 7. Latency gain against q.

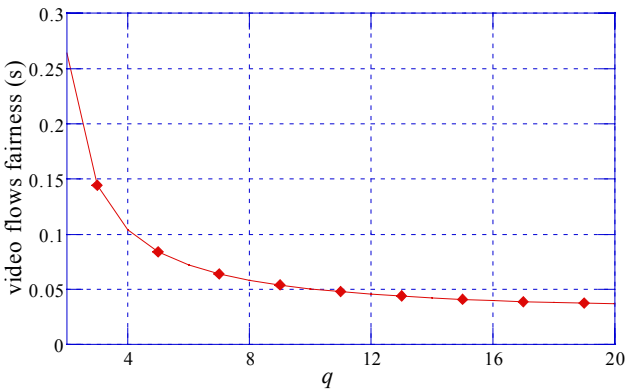


Figure 8. Fairness gain among video flows against q.

two backlogged flows is preserved during the rounds: if flow i is serviced *before* flow j during round k and both flows are still backlogged at round $k+1$, flow i will be serviced *before* flow j at round $k+1$. Thus, if packets from both flows are supposed to leave during a given round, flow i packets will be transmitted before flow j's. In Aliquem DRR, due to the presence of multiple FIFO active lists, it is possible that the service sequence of two backlogged flows can be altered from one round to another.

Let us show this with a simple example. Let us consider flows i and j, which are backlogged at a given round k. Let us assume that the sequence of packets queued at both flows is the one reported in Figure 6(a). Both flows transmit a packet during round k, and the order of transmission of the packets is $p_{i,1}$, $p_{j,1}$. In Standard DRR, this implies that, as long as both flows are backlogged, flow i is serviced before flow j; thus, the packet transmission sequence during round $k+3$ (i.e. in a round when packets from both flows are transmitted) is $p_{i,3}$, $p_{j,3}$ (see Figure 6(c)). Consider the same scenario in Aliquem DRR: both flows are initially queued in the k-th active list (assume for simplicity that $k < q-3$), and flow i is queued *before* flow j (see Figure 6(b)). However, even though both flows are constantly backlogged from round k to round $k+3$, the packet transmission sequence during round $k+3$ is the opposite, i.e., $p_{j,3}$, $p_{i,3}$.

Since in Aliquem DRR backlogged flows do not necessarily share the same active list on every round, it is not possible to preserve their service sequence by employing FIFO active lists. Doing so would require using *sorted lists*, which have an intrinsic complexity of $O(\log N)$. This would lead to a higher complexity.

Note that in Aliquem DRR the service sequence inversion does not affect couples of flows whose quantum contains a maximum length packet. In fact, if $\phi_i \geq \overline{L_i}$ and $\phi_j \geq \overline{L_j}$, the two flows are always dequeued from the head of the same list (the current list) and enqueued at the tail of the same list (the subsequent list); therefore, no service inversion can take place.

Let us refer to the worst-case scenario under which (12) was derived. Due to the flow sequence inversion, during $(t_1, t_2]$, flow i can receive service *two more times* than flow j; in the above example, this happens if t_1 and t_2 are selected as the time instants in which packets $p_{j,1}$ and $p_{i,3}$ start being transmitted. By following the same procedure used for computing the standard DRR fairness measure, it is straightforward to prove that the fairness measure of Aliquem DRR is:

$$FM_{i,j}^{A} = \begin{cases} \dfrac{1}{C}\left(F + \dfrac{\overline{L_i}}{f_i} + \dfrac{\overline{L_j}}{f_j}\right) & \text{if } \phi_i \geq \overline{L_i} \text{ and } \phi_j \geq \overline{L_j} \\ \dfrac{1}{C}\left(2F + \dfrac{\overline{L_i}}{f_i} + \dfrac{\overline{L_j}}{f_j}\right) & \text{otherwise} \end{cases}.$$

Given a frame length F, Aliquem DRR can have a worse fairness measure than Standard DRR. However, as already stated in the previous subsection, Aliquem DRR allows a frame length to be selected which is less than the minimum frame length of Standard DRR operating at $O(1)$ complexity by a factor of $q-1$. Therefore, by employing Aliquem DRR with $q > 2$, we obtain a better fairness measure than that of Standard DRR operating at $O(1)$ complexity. We also observe that, if $q = 2$, flow sequence inversion cannot take place, and therefore the fairness measure of an Aliquem DRR with two active lists is the same as that of Standard DRR operating at $O(1)$ complexity.

D. *Trade-off between Performance and Overhead*

As seen earlier, Aliquem DRR allows us to obtain much better performance bounds (both for latency and for fairness) by selecting smaller frames. Smaller frames are obtained by employing a large number of active lists, i.e., a large q value, and the q value affects the operational overhead. However, with a modest increase in the operational overhead it is possible to achieve a significant improvement in both latency and fairness. Let us recall the example of Section II.D: if Aliquem DRR with $q = 10$ is employed, the latency of a data flow is reduced by 41% and the latency of a video flow is reduced by 73%, as shown in Figure 7 and 8. In addition, the fairness measure between any two video flows is reduced by 80%.

V. SMOOTH ALIQUEM DRR

According to DRR, during a round a flow is serviced as long as its deficit is larger than the head packet. Aliquem DRR isolates the subset of flows that can actually transmit packets during a round, but, according to the pseudo-code in Figure 2, it

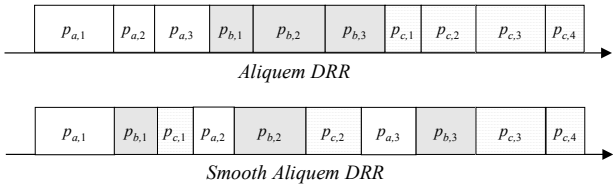


Figure 9. Aliquem DRR and Smooth Aliquem DRR Output

```
PacketArrival (packet p, flow I) {
   if ( NotBacklogged(I) ) {
      Δ_I= 0;
      YetServiced[I]=False;
      R=ceiling (Length(p)/ϕ_I);
      g = (CurrentList+R) mod q;
      EnqueueFlow (I, ActiveList(g);
      }
   EnqueuePacket (p, FlowQueue(I));
}

SmoothAliquemDRRScheduler () {
   while (SystemBusy) {
      while ( NotEmpty(CurrentList) ) {
         I = DequeueFlow(CurrentList);
         L = Length(HeadPacket(I));
         if(not(YetServiced[I]))
            Δ_I = ceiling((L-Δ_I)/ϕ_I)*ϕ_I + Δ_I;
         p=DequeueHeadPacket(I);
         TransmitPacket(p);
         Δ_I-= L;
         L = Length(HeadPacket(I));
         if (L>Δ_I) {
            R = ceiling((L-Δ_I)/ϕ_I);
            g = (CurrentList+R) mod q;
            YetServiced[I]=False;
            EnqueueFlow(I, ActiveList(g));
            }
         else if (L>0) {
            YetServiced[I]=True;
            EnqueueFlow(I, CurrentList);
            }
      }
   CurrentList=NextNonEmptyList();
   }
}
```

Figure 10. Pseudo-code for Smooth Aliquem DRR

services each flow exhaustively before dequeuing the next one. Therefore, a burst of packets of overall length at most equal to $L_i + \phi_i$ can leave the server when a flow is serviced. Clearly, reducing the frame length also implies reducing the flow burstiness as a side effect, since flows are polled more frequently and for smaller quanta. However, we can further reduce it by forcing each flow to transmit only *one* packet when selected for transmission. The flow is then queued back (if still backlogged) either in the current active list (if the deficit is still larger than the new head packet) or in a new active list, located by applying (16). Reducing the output burstiness has several well-known advantages, such as reducing the buffer requirements on the nodes of a multi-node path, and improving the average fairness and average delay. We define a slightly modified version of Aliquem DRR, called Smooth Aliquem DRR, in which flows transmit one packet at a time. Figure 9 shows an example of the difference in the output packet sequence between Aliquem DRR and Smooth Aliquem DRR, assuming that three flows are queued in the current list. The pseudo-code for Smooth Aliquem DRR is reported in Figure 10.

Since a flow might receive service more than once per round, a flag `YetServiced[i]` is needed in order to distinguish whether it is the first time that flow i is going to be serviced during the round (and in that case its deficit must be updated) or not. This flag is set to *false* for a newly backlogged flow and for a flow which is enqueued in a different active list after being serviced, and is set to *true* when the flow is queued back in the same active list.

Smooth Aliquem DRR has a similar operational overhead as Aliquem DRR, which only depends on the q value and on the chosen implementation of the `NextNonEmptyList()` function. The only space overhead added by Smooth Aliquem DRR is the vector of N flags `YetServiced[]`. However, it is possible to show that the latency, start-up latency and fairness measure of Smooth Aliquem DRR are the same as Aliquem DRR's.

VI. Comparison with Previous Work

A. Comparison with Other Implementation Techniques

In this subsection we discuss the differences between our proposal and the one described in [15] for reducing the output burstiness of a DRR scheduler. In [15], it is observed that the output burstiness of a DRR scheduler could be reduced by allowing a flow to be serviced several times within a round, one packet at a time. In order to do so, a DRR round is divided into *sub-frames*; each sub-frame is associated with a FIFO queue, in which references to backlogged flows are stored. Sub-frame queues are visited orderly, and each time a flow is dequeued it is only allowed to transmit *one* packet; after that, it will be enqueued in another sub-frame queue if still backlogged. The correct sub-frame queue a backlogged flow has to be enqueued into is located by considering the finishing timestamp of the flow's head-of-line packet. The data structure proposed in [15] consists of *two* arrays of q sub-frames each, associated with the *current* and *subsequent* rounds respectively, which are cyclically swapped as rounds elapse. Obviously, the number of operations needed to select the next non-empty sub-frame queue increases linearly with the array dimension. It can be observed that – though this aspect is not dealt with in [15] – it could be possible to reduce the operational overhead of the sub-frames array by employing two VEB-PQs or bit vector trees (one for the array related to the current round and another for the array related to the subsequent round). Thus, assuming that the dimension of a sub-frames array and the number of active lists in Aliquem are comparable, both implementations have the same operational overhead; however, the sub-frames array data structure takes twice as much space as an active list queue of the same length (either with or without employing additional data structures). It is observed in [15] that, although the proposed implementation reduces the typical DRR burstiness, it does not reduce its performance bounds such as the latency or fairness measure; this is probably due to the fact that the proposed implementation cannot reduce the frame length, which is in fact bounded by (2). On the other hand, the Aliquem implementation allows the frame length to be reduced, which actually reduces latency and fairness measure.

B. Comparison with Other Scheduling Algorithms

In this subsection we compare Aliquem DRR with some existing scheduling algorithms for packet networks. Specifically, we compare Aliquem DRR with Self-Clocked Fair Queueing (SCFQ) [6], Pre-Order DRR (PDRR) [13] and Smoothed Round Robin (SRR) [14].

We have already observed that the DRR latency and fairness measure are related to the frame length. If we let the frame length go to zero, from (5) and (12) we obtain the following DRR *limit latency* and *limit fairness measure* respectively:

$$\Theta_i^0 = \frac{1}{C}\left(\overline{L_i}/f_i + \sum_{j=1, j\neq i}^{N} \overline{L_j}\right), \tag{22}$$

$$FM_{i,j}^0 = \frac{1}{C}\left(\frac{\overline{L_i}}{f_i} + \frac{\overline{L_j}}{f_j}\right). \tag{23}$$

The limit latency and fairness measure of DRR are equal to the latency and fairness measure of SCFQ. The latter is a sorted-priority scheduling algorithm that has $O(\log N)$ complexity, and it has been analyzed as an LR server in [3], [4]. Thus, Aliquem DRR performance bounds are bounded from below by those of SCFQ, to which they tend as $q \to \infty$. Recalling the example in Section IV.D, we obtain that, when $q = 20$, latency bounds are 4.1% above the limit latency for data flows and 22% above the limit latency for the video flows.

PDRR is aimed at emulating the Packet Generalized Processor Sharing (PGPS) [5] by coupling a DRR module with a priority module that manages Z FIFO priority queues. Each packet which should be transmitted in the current DRR round is instead sent to one of such priority queues, selected according to the expected packet leaving time under PGPS. Packets are dequeued from the priority queues and transmitted. The higher the number of priority queues Z is, the closer PGPS is emulated. In order to locate the non empty priority queue with the highest priority, a min heap structure is employed. It is said in [13] that PDRR exhibits $O(\log Z)$ worst-case per packet complexity, the latter being the cost of a single insertion in the min heap, provided that its DRR module operates at $O(1)$ complexity. However, a careful analysis of the PDRR pseudo code shows that this is not the case. In fact, at the beginning of a new round, the min heap is empty, and it is filled up by dequeueing packets from *all* backlogged flows and sending each of them to the relevant priority queue. This process has to be completed *before* packet transmission is started, otherwise the PDRR ordering would be thwarted. Looping through all backlogged flows requires $O(N)$ iterations, Z of which might trigger a min heap insertion. Thus, transmitting the first packet in a round requires $O(N + Z\log Z)$ operations. Clearly, the complexity of Aliquem DRR is instead much lower. As far as latency is concerned, the expression computed in [13] applied to the example of Section II.D yields a latency for a video flow which is 480ms when $Z = 10$, i.e., higher than that of Standard DRR itself computed by applying (5). This probably implies that PDRR latency bound is not tight, which makes a well-based comparison impossible.

SRR smoothens the round-robin output burstiness by allowing a flow to send one maximum length packet worth of bytes every time it is serviced. The relative differentiation of the flows is achieved by visiting them a different number of times during a round, according to their rate requirements, and two visits to the same flow are spread apart as far as possible. The space occupancy required by SRR – which is mainly due to the data structure needed to store the sequence of visits – grows exponentially with the number of bits k employed to memorize the flows rate requirements. If $k = 32$, the space occupancy is about 200 Kbytes. Aliquem DRR does not require such space occupancy. Although the complexity of SRR does not depend on the number of active flows, $O(k)$ operations must be performed whenever a flow becomes idle or backlogged, and this may happen many times in a round. It has been observed in [14] that such a burden may be comparable to the $O(\log N)$ complexity of sorted-priority algorithms. On the other hand, the operational overhead arising from searching for the next non-empty list in Aliquem DRR (which has to be done at most *once* per round) is expected to be much lower, especially if either of the two additional data structures proposed in Section IV.A is employed.

In [14], it is claimed that the start-up latency of SRR (referred to therein as the *scheduling delay*) is much lower that that of DRR. We then compare the start-up latency of both algorithms. Let us suppose that a set of N flows requiring rates $\rho_1, \rho_2, ..., \rho_N$ are scheduled on a link whose capacity is C, with $\sum_i \rho_i = C$. In that case, after some straightforward manipulation, we obtain the following result for SRR:

$$S_i^{SRR} < \frac{2L_{\max}}{C}\left(\frac{1}{f_i} + N - 1\right) = \overline{S_i^{SRR}} \tag{24}$$

where $L_{\max} = \max_{i=1..N}\left(\overline{L_i}\right)$

For Aliquem DRR, assuming as a worst case that $\overline{L_i} = L_{\max}$ and $F^S = \overline{L_i}/f_i$, we obtain from (21):

$$S_i^A = \frac{L_{\max}}{C}\left[\frac{1}{f_i} + \sum_{j=1}^{N}\frac{\overline{L_j}}{L_{\max}} - 1\right] \le \frac{\overline{S_i^{SRR}}}{2} \tag{25}$$

This result, which counters the claim made in [14], can be partially explained by considering that SRR performs a large number of visits (up to $2^k - 1$) in a round, and on each visit a flow's deficit is increased by $L_{\max}$, regardless of the flow's actual maximum packet size L_i.

VII. Simulations

In this section we show some of the Aliquem DRR and Smooth Aliquem DRR properties by simulation. We have implemented both schedulers in the *ns* simulator [25].

A. Operational Overhead

We consider a scenario consisting of a single node with a 1 Mbps output link shared by 40 traffic sources. All sources transmit UDP packets with lengths uniformly distributed between 500 and 1500 bytes and are kept constantly backlogged.

Twenty traffic sources require 10 Kbps, and the remaining twenty require 40 Kbps. The scenario is simulated for 10 seconds with both Standard DRR and Aliquem DRR, and the number of operations that are required for each packet transmission is traced.

By following the guidelines for DRR quanta allocation outlined in Section II.D, we would obtain $\phi_i = 1500$ bytes for the 10 Kbps sources and $\phi_i = 6000$ bytes for the 40 Kbps sources, which yield a frame length $F^S = 150$ Kbytes. We want to compare the number of operations per-packet transmission of Standard DRR and Aliquem DRR if the frame length is selected as $F^S/100$, and quanta are allocated consequently, so as to obtain a latency which is close to the limit latency. In Aliquem DRR, this requires $q = 101$ active lists. In this simulation, we use the linear search `NextNonEmptyList()` implementation for Aliquem DRR and we consider as an operation unit a flow enqueueing/dequeueing or an iteration of the loop in `NextNonEmptyList()`. Clearly, this is the most unfavorable scenario for assessing the Aliquem DRR operational overhead reduction, since no additional data structure (as a VEB-PQ or a bit vector tree) is employed. Nevertheless, as Figure 11 clearly shows, the number of operations per packet transmission is much lower in Aliquem DRR. Moreover, the average number of operations per packet transmission in Aliquem DRR (which is 3.2) is very far from the upper bound, which is $q+2 = 103$. A thorough evaluation of the operational overhead would also entail taking into account protocol-related and architectural issues (such as header processing, memory transfers and so on), which cannot be easily covered through *ns* simulation.

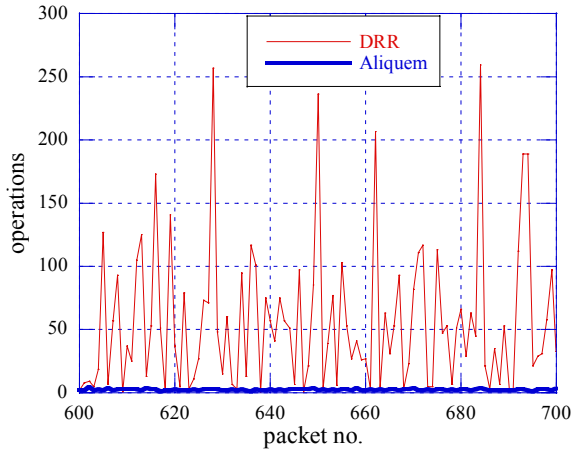


Figure 11. Per-packet operations comparison.

TABLE I. AVERAGE AND MAXIMUM DELAY COMPARISON.

	Aliquem DRR				Smooth Aliquem DRR			
	q=2	*q=5*	*q=11*	*q=101*	*q=2*	*q=5*	*q=11*	*q=101*
avg delay (ms)	*28.953*	19.142	18.669	18.201	24.768	19.263	18.633	18.197
avg delay gain %	-	33.9%	35.5%	37.1%	14.5%	33.5%	35.6%	37.2%
max delay (ms)	*61.444*	39.627	38.870	37.968	61.689	39.942	38.870	37.968
max delay gain %	-	35.5%	36.7%	38.2%	-0.4%	35.0%	36.7%	38.2%

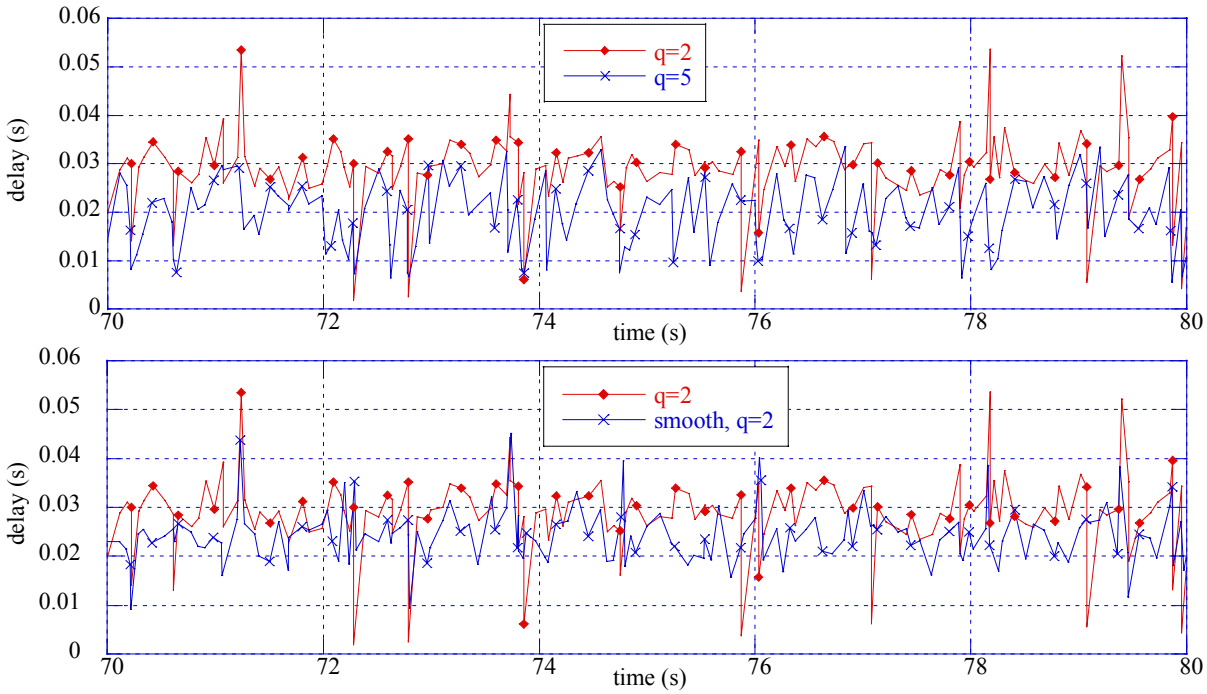


Figure 12. Delay trace comparison.

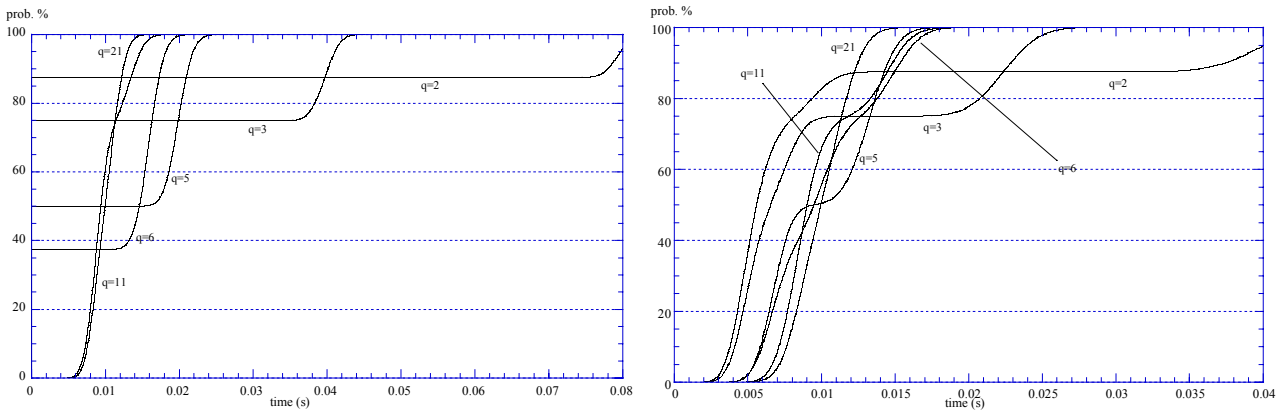


Figure 14. Distribution of the inter-packet spacing for the CBR flows for various values of q in Aliquem DRR (left) and Smooth Aliquem DRR (right)

B. *Delay*

We have shown in the previous sections how it is possible to achieve lower latencies in Aliquem DRR. In this simulation we show how it is possible to reduce a flow's *average* delay by increasing the number of active lists q and using the Smooth Aliquem version. We consider a scenario consisting of a single node with a 10 Mbps output link shared by 40 traffic sources. The flow we monitor in the simulation sends 500 bytes long UDP packets at a constant rate of 100 Kbps. The other 39 sources transmit UDP packets with lengths uniformly distributed between 100 and 1500 bytes. Nineteen traffic sources require 100 Kbps, and the remaining twenty require 400 Kbps. The scenario is simulated for 100 seconds with various values of q with Aliquem DRR and Smooth Aliquem DRR, and the sum of the queueing and transmission delay for each packet of the tagged flow is traced.

We report the average and maximum delay experienced by the tagged flow's packets in the various experiments in Table 1. In order to show the delay gain, we take Aliquem DRR with $q = 2$ (which exhibits the same behavior as Standard DRR operating at $O(1)$ complexity) as a reference. The table shows that an average and maximum delay reduction of about 33% can be achieved even with q as small as 5; furthermore, employing Smooth Aliquem reduces the average delay by about 15% even when $q = 2$. We observe that, as q gets higher, the performance metrics of Smooth Aliquem DRR and Aliquem DRR tend to coincide. This is because quanta get smaller, and therefore the probability that a flow is able to send more than one packet per round decreases. Figure 12 shows two delay traces, from which the benefits introduced by Aliquem and Smooth Aliquem are also evident.

C. *Delay Variation in a Multi-node Path*

In this experiment we show that reducing the frame length also helps to preserve the traffic profile of a CBR flow in a multi node path, thus reducing the need for buffering both in the nodes and at the receiving host. We consider a scenario consisting of three scheduling nodes connected by 10 Mbps links. A CBR source sends 125 bytes packets with 10ms period across the path, thus occupying 100Kbps. Other traffic sources (kept in asymptotic conditions) are added as background traffic along the path, so that the capacity of each link is fully utilized. On each link, the background traffic consists of 21 sources sending packets uniformly distributed between 50 and 1500 bytes. Six sources require 0.9 Mbps each, 5 sources require 0.6 Mbps each, the remaining 10 require 0.15 Mbps each. To avoid correlation on the downstream nodes, background traffic sent from a node to its downstream neighbor is not re-scheduled. Instead, each node removes the incoming background traffic and generates new outgoing background traffic, as shown in Figure 13.

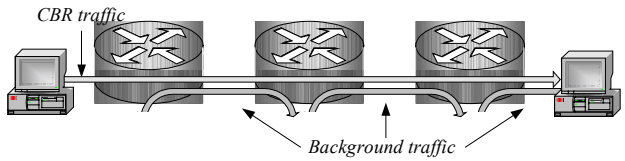


Figure 13. Simulation scenario

We simulate the network for 350 seconds employing both Aliquem DRR and Smooth Aliquem DRR as schedulers. We select the frame length according to (19), which yields $F = 100/(q-1)$ Kbytes, and perform experiments with various values of q, tracing the inter-arrival spacing of the CBR source packets on the destination host. In an ideal fluid-flow fair queueing system, the inter-arrival spacing would be constant (and obviously equal to 10ms). In a packet-wise fair-queueing system, we can expect the inter-arrival spacing to be distributed in some way around an average value of 10ms: in this case, the narrower the distribution is, the closer the ideal case is approximated. Figure 14 shows the probability distribution of the inter-packet spacing (confidence intervals are not reported, since they are too small to be visible). For small q values, the distribution tends to be bimodal: this means that the scheduling on the various nodes has turned the original CBR traffic injected by the source into bursty on/off traffic at the destination host. The latter will then need buffering in order to counterbalance the jittering introduced. As q grows, the curves become narrower around the average value, thus denoting a smoother traffic profile at the destination. The same behavior with respect to a variation in the q parameter can be observed both in Aliquem DRR and in Smooth Aliquem DRR. However, in the latter the distributions for a

given q value are generally narrower[2], due the smoothing effect.

VIII. CONCLUSIONS

In this paper we have analyzed the Deficit Round-Robin scheduling algorithm, and we have derived new and exact bounds on its latency and fairness. Based on these results, we have proposed an implementation technique, called the Active Lists Queue Method (Aliquem), which allows DRR to work with smaller frames while still preserving the $O(1)$ complexity. As a consequence, Aliquem DRR achieves better latency and fairness. We have proposed several solutions for implementing Aliquem DRR, employing different data structures and requiring different space occupancy and operational overhead. We have also presented a variation of Aliquem DRR, called Smooth Aliquem DRR, which further reduces the output burstiness at the same complexity. We have compared Aliquem DRR to PDRR and SRR, showing that it either achieves better performances with the same operational overhead or provides comparable performances at a lower operational overhead than either of the other two. Our simulations showed that Aliquem DRR allows the average delay to be reduced, and it also lessens the likelihood of bursty transmission in a multi-hop environment.

REFERENCES

[1] L. Lenzini, E. Mingozzi, and G. Stea, "Aliquem: a Novel DRR Implementation to Achieve Better Latency and Fairness at *O*(1) Complexity", in *Proc. of the 10th International Workshop on Quality of Service (IWQOS), Miami Beach,USA, May 2002, pp. 77-86*.

[2] H. Zhang "Service Disciplines for Guaranteed Performance Service in Packet-Switching Networks", Proceedings of the IEEE, Vol. 83, No. 10, pp. 1374-1396, October 1995.

[3] D. Stiliadis and A. Varma, "Latency-Rate Servers: A General Model for Analysis of Traffic Scheduling Algorithms," *IEEE/ACM Trans. on Networking*, Vol. 6, pp. 675-689, October 1998.

[4] D. Stiliadis and A. Varma, "Latency-Rate Servers: A General Model for Analysis of Traffic Scheduling Algorithms," *Tech. Rep. CRL-95-38*, University of California at Santa Cruz, USA, July 1995.

[5] 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. on Networking*, Vol. 1, pp. 344-357, June 1993.

[6] S. J. Golestani, "A Self-Clocked Fair Queueing Scheme for Broadband Applications," in *Proc. of IEEE INFOCOM'94*, pp. 636-646, Jun. 14-16, 1994, Toronto, Canada.

[7] M. Shreedhar and G. Varghese, "Efficient Fair Queueing Using Deficit Round Robin," *IEEE/ACM Trans. on Networking*, Vol. 4, pp. 375-385, June 1996.

[8] S. Suri, G. Varghese, and G. Chandranmenon "Leap Forward Virtual Clock: A New Fair Queueing Scheme With Guaranteed Delay and Throughput Fairness," in *Proc. of IEEE INFOCOM'97*, Apr. 7-11, 1997, Kobe, Japan.

[9] P. Goyal, H.M. Vin and H. Cheng, "Start-Time Fair Queueing: A Scheduling Algorithm for Integrated Services Packet Switching Networks", *IEEE/ACM Trans. on Networking*, Vol. 5, No. 5, pp. 690-704, October 1997.

[10] J. Bennett and H. Zhang: "Hierarchical Packet Fair Queueing Algorithms", *IEEE/ACM Trans. on Networking*, Vol. 5, No. 5, pp. 675-689, October 1997.

[11] F. Toutain, "Decoupled Generalized Processor Sharing: A Fair Queuing Principle for Adaptive Multimedia Applications", *Proc. 17th IEEE Infocom '98*, San Francisco, CA, USA, March – April 1998

[12] D. Saha, S. Mukherjee, K. Tripathi: "Carry-Over Round Robin: A Simple Cell Scheduling Mechanism for ATM Networks". *IEEE/ACM Trans. on Networking,* Vol. 6, No. 6, pp. 779-796, December 1998.

[13] S.-C. Tsao and Y.-D. Lin, "Pre-Order Deficit Round Robin: a new scheduling algorithm for packet switched networks," *Computer Networks*, Vol. 35, pp. 287-305, February 2001.

[14] G. Chuanxiong, "SRR: An O(1) Time Complexity Packet Scheduler for Flows in Multi-Service Packet Networks," in *Proc. of ACM SIGCOMM'01*, pp. 211-222, Aug. 27-31, 2001, San Diego, CA, USA.

[15] A. Francini, F. M. Chiussi, R. T. Clancy, K. D. Drucker, and N. E. Idirene, "Enhanced Weighted Round Robin Schedulers For Accurate Bandwidth Distribution in Packet Networks," *Computer Networks*, Vol. 37, pp. 561-578, Nov. 2001.

[16] S. S. Kanhere, H. Sethu, A. B. Parekh, "Fair and Efficient Packet Scheduling Using Elastic Round-Robin", *IEEE Trans. on Paral. and Dist. Systems*, Vol. 13, No. 3, March 2002

[17] L. Lenzini, E. Mingozzi, G. Stea, "A unifying service discipline for providing rate-based guaranteed and fair queueing services based on the Timed Token Protocol", *IEEE Trans. on Computers, Vol. 51, No. 9 Sept. 2002, pp. 1011-1025.*

[18] L. Lenzini, E. Mingozzi, G. Stea, "Packet Timed Token Service Discipline: a scheduling algorithm based on the dual-class paradigm for providing QoS in integrated services networks", *Computer Networks 39/4, July 2002,* pp.363-384.

[19] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Services", RFC 2475, The Internet Society, December 1998.

[20] B. Davie et al. "An Expedited Forwarding PHB (Per-Hop Behavior)", RFC 3246, The Internet Society, March 2002.

[21] A. Charny et al. "Supplemental Information for the New Definition of the EF PHB (Expedited Forwarding Per-Hop Behavior)", RFC 3247, The Internet Society, March 2002.

[22] P. Van Emde Boas, R. Kaas, and E. Zijlstra, "Design and Implementation of an Efficient Priority Queue," *Math. Syst. Theory*, Vol. 10, pp. 99-127, 1977.

[23] K. Mehlhorn, "Data Structures and Algorithms 1: Sorting and Searching", *EATCS Monographs on Theoretical Computer Science,* Springer-Verlag.

[24] A. Brodnik, S. Carlsson, J. Karlsson, J. Ian Munro, "Worst case Constant Time Priority Queue", *ACM/SIAM 12th Symposium on Discrete Algorithms 2001*, Washington, DC, USA, Jan. 2001, pp. 523-528.

[25] The Network Simulator - ns-2, <http://www.isi.edu/nsnam/ns/>.

IX. APPENDIX

A. Proof of the Theorem 1

Before proving the Theorem 1, let us report the following lemmas:

Lemma 1

"Let ϕ_i be flow i quantum, and let $\overline{L_i}$ be its maximum packet length; the service received by flow i in the time interval $[t_1, t_2)$ in which it is always backlogged is lower bounded by:

$$v \cdot \phi_i - \overline{L_i} < W_i(t_1, t_2)$$

where v is the number of service opportunities in $[t_1, t_2)$ ".

[2] Note that the horizontal scale of the graph related to Smooth Aliquem DRR is one half of the other

Proof: see [7].

Lemma 2

"*Let flow i be always backlogged since the beginning of round k. If at least one packet is serviced from flow i during round $k+v$, $v \geq 1$, then the deficit value right after the $k+v^{th}$ service opportunity is bounded by:*

$$\Delta_i^{k+v} < \min\left\{\phi_i, \overline{L_i}\right\}.$$

Proof:

If $\phi_i \geq \overline{L_i}$ then the thesis follows straightforwardly from inequality (1), which proves that $\Delta_i < \overline{L_i}$ after each service opportunity. Let us then assume that $\phi_i < \overline{L_i}$; we can express Δ_i^{k+v} as:

$$\Delta_i^{k+v} = \Delta_i^{k+v-1} + \phi_i - S_i(k+v) \tag{26}$$

where $S_i(k+v) > 0$ denotes the service received during the $k+v^{th}$ service opportunity. It is easy to prove that $S_i(k+v) > \Delta_i^{k+v-1}$. Suppose that $\Delta_i^{k+v-1} > 0$ (otherwise $\Delta_i^{k+v} = \Delta_i^{k+v-1} + \phi_i - S_i(k+v) = \phi_i - S_i(k+v)$ and the thesis obviously holds). This implies that a packet which is *longer* than Δ_i^{k+v-1} could not be serviced during round $k+v-1$; let $L > \Delta_i^{k+v-1}$ be its length. Clearly, $S_i(k+v) \geq L$ (otherwise no packet transmission takes place at round $k+v$), and therefore $S_i(k+v) \geq L > \Delta_i^{k+v-1}$. By substituting into (26) we obtain:

$$\Delta_i^{k+v} = \Delta_i^{k+v-1} + \phi_i - S_i(k+v) < \phi_i$$

■

We are now ready to compute the latency of a DRR system. We observe that the following proof process can be easily adapted in order to compute the latency of other round-robin LR servers, such as the Packet Timed Token Service Discipline [17], [18].

Theorem 1

"The latency of DRR is:

$$\Theta_i = \frac{1}{C}\left[\left(F - \phi_i\right)\left(1 + \overline{L_i}/\phi_i\right) + \sum_{j=1}^{N} \overline{L_j}\right]. \tag{5}$$

Proof

Let us denote with $W_i(t_1, t)$ the service received by flow i in the time interval $[t_1, t)$ (*service curve* of flow i). We assume that $W_i(t_1, t)$ increases by a quantity equal to the packet length when the *last bit* of the packet has been serviced. In order to derive the DRR latency, we apply the methodology described in [3], Lemma 7. Specifically, we assume that flow i is continuously backlogged starting from time t_1 and we prove that, in a generic time interval $[t_1, t)$, it is:

$$W_i(t_1, t) \geq \max\left(0, f_i \cdot C\left(t - t_1 - \Theta_i\right)\right) \tag{27}$$

the latter representing the *latency rate curve* of flow i. We then show that the bound is tight by presenting a case in which the service received is exactly equal to the bound.

As (27) is required to hold in any possible scenario, we can limit ourselves to proving it when $W_i(t_1, t)$ is the lowest possible. Clearly, this happens in a scenario (hereafter *worst-case* scenario) in which:

a) flow i has each service opportunity *as late as possible*;

b) the service received by flow i after each service opportunity is *the lowest possible*.

Without loss of generality, we will assume t_1=0. Let us number the rounds from 1 onward starting with the first one in which flow i has a service opportunity. We will denote with T_v the time instant in which the v^{th} service opportunity for flow i ends.

Let us distinguish two cases, $\phi_i \geq \overline{L_i}$ and $\phi_i < \overline{L_i}$.

a) Case 1: $\phi_i \geq \overline{L_i}$

When $\phi_i \geq \overline{L_i}$, a backlogged flow can transmit at least one packet on each service opportunity. For mathematical convenience, we need to define two functions which bound the service curve $W_i(0,t)$:

- $\overline{W}_i(0,t)$, which increases with a C slope whenever flow i is being serviced. More formally:
  - for any instant s in a time interval $[s_1, s_2) \subseteq [0,t)$ in which flow i is not being serviced, $\overline{W}_i(0,s) = \overline{W}_i(0,s_1)$, i.e. $\overline{W}_i(0,s)$ remains constant;
  - for any instant s in a time interval $[s_1, s_2) \subseteq [0,t)$ in which flow i is being serviced, $\overline{W}_i(0,s) = \overline{W}_i(0,s_1) + C \cdot (s - s_1)$, i.e. $\overline{W}_i(0,s)$ increases with a C slope;
- $w_i(0,t)$, which increases stepwise with a step equal to the received service at the end of a time interval in which flow i is being serviced and remains constant elsewhere.

Clearly, for any time instant t, the following relationship holds:

- $w_i(0,t) \leq W_i(0,t) \leq \overline{W}_i(0,t)$ whenever i is being serviced
- $w_i(0,t) = W_i(0,t) = \overline{W}_i(0,t)$ whenever i is not being serviced.

Figure 15 shows the three curves. For the sake of readability, the channel capacity is assumed to be equal to 1 in the figures, so that both the horizontal and vertical axis have the same scale.

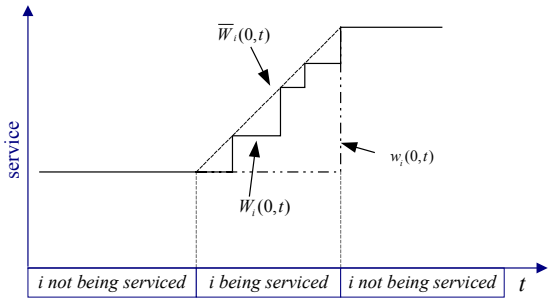


Figure 15. Bounds on the service curve for $\phi_i \geq \overline{L_i}$

In this case, according to assumption b) and Lemma 1, we can assert that, after v service opportunities, the worst-case service curve reaches a height:

$$W_i(0,t) = v \cdot \phi_i - \overline{L_i} \tag{28}$$

This implies that $W_i(0,t)$ increases by a step equal to ϕ_i at each time instant T_v, $v > 1$[3]. In a finite capacity system, the service curve cannot increase by a step greater than the maximum length packet; therefore we can then obtain a tighter lower bound on the service curve than $w_i(0,t)$ by considering:

$$w_i^{'}(0,t) = \max\left\{w_i(0,t), \overline{W}_i(0,t) - \overline{L_i}\right\}$$

The curve $w_i^{'}(0,t)$ is shown in Figure 16.

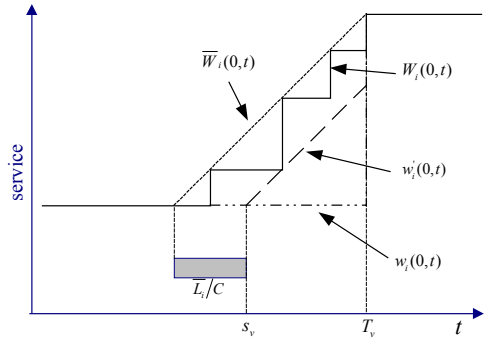


Figure 16. Improved bounds on the service curve for $\phi_i \geq \overline{L_i}$

We will denote by s_v the time instant that lies $\left(\phi_i - \overline{L_i}\right)/C$ units of time before T_v (note that $T_v = s_v$ if $\phi_i = \overline{L_i}$).

Each time instant T_v comes "as late as possible" (i.e. condition a) holds) under the following conditions:

- each flow is always backlogged;
- each flow $h \neq i$ starts with the maximum deficit $\Delta_h \cong \overline{L_h}$[4];
- each flow $h \neq i$ always transmits packets in such a way that its quantum is always fully exploited (i.e. no deficit is carried over onto subsequent rounds);
- every flow $h \neq i$ is serviced before flow i gets its first service opportunity.

Under the above conditions, we have:

$$\begin{aligned} T_1 &= \frac{1}{C}\left[\sum_{h \neq i}\phi_h + \sum_{h \neq i}\overline{L_h} + (\phi_i - \overline{L_i})\right] \\ &= \frac{1}{C}\left[F + \sum_{h=1}^{N}\overline{L_h} - 2\overline{L_i}\right] \end{aligned} \tag{29}$$

$$T_2 = T_1 + \frac{1}{C}\left[\sum_{h \neq i}\phi_h + \phi_i\right] = \frac{1}{C}\left[2F + \sum_{h=1}^{N}\overline{L_h} - 2\overline{L_i}\right] \tag{30}$$

$$T_k = T_{k-1} + \frac{1}{C}\left[\sum_{h \neq i}\phi_h + \phi_i\right] = \frac{1}{C}\left[k \cdot F + \sum_{h=1}^{N}\overline{L_h} - 2\overline{L_i}\right] \tag{31}$$

We observe that, for $k > 1$,

$$\frac{w_i^{'}(0,s_{k+1}) - w_i^{'}(0,s_k)}{s_{k+1} - s_k} = \frac{C \cdot \phi_i}{F} = f_i \cdot C \tag{32}$$

which confirms that the normalized rate allocated to flow i is in fact $f_i \cdot C \leq C$. This also implies that the line that joins points $\left(s_v, w_i^{'}(0,s_v)\right)$, $v > 1$, has a slope $f_i \cdot C$, (as shown in Figure 17). We will prove that this line is the *latency-rate curve* for flow i.

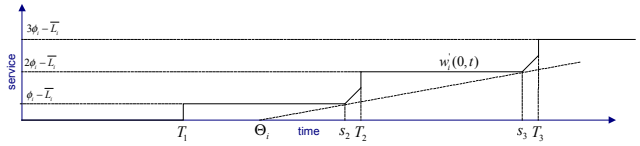


Figure 17. Latency computation for $\phi_i \geq \overline{L_i}$

Let us define:

$$\Theta_i = s_2 - \frac{\phi_i - \overline{L_i}}{f_i \cdot C} = T_2 - \frac{\phi_i - \overline{L_i}}{C} - \frac{\phi_i - \overline{L_i}}{f_i \cdot C} \tag{33}$$

[3] When $v = 1$, $w_i(0,t)$ increases by a step equal to $\phi_i - \overline{L_i}$.

[4] We observe that it is not possible for the deficit variable to assume the value of the maximum length packet, since inequality holds into Lemma 1. However, the deficit can be arbitrarily close to the maximum packet length.

Θ_i marks the intersection of the latency-rate curve with the time axis. By substituting (30) into (33), after a few algebraic manipulations we obtain expression (5). According to the methodology described in [3-4], we need to prove that $w_i^{'}(0,t) \geq f_i \cdot C(t-\Theta_i)$.

Let us first prove that $w_i^{'}(0,s_k) = f_i \cdot C(s_k - \Theta_i)$, $\forall k > 1$. By definition we have $w_i^{'}(0,s_k) = (k-1)\phi_i - L_i$, whilst:

$$f_i \cdot C(s_k - \Theta_i) = f_i \cdot C(s_k - s_2) + \phi_i - \overline{L_i} = (k-1)\phi_i - \overline{L_i};$$

this proves the equality.

We then prove that $w_i^{'}(0,t) \geq f_i \cdot C(t-\Theta_i)$ when $t \neq s_k$, $k > 1$. Suppose that $t \in (s_k, T_k]$, $k > 1$. In these regions, the curve $w_i^{'}(0,t)$ increases with a slope which is at least C [5], whilst the latency-rate curve increases with a constant slope $f_i \cdot C \leq C$. This implies that $w_i^{'}(0,t) \geq f_i \cdot C(t-\Theta_i)$ for any $t \in (s_k, T_k]$, $k > 1$, since $w_i^{'}(0,s_k) = f_i \cdot C(s_k - \Theta_i)$.

Suppose then that $t \in (T_k, s_{k+1})$, $k > 1$. Since the curve $w_i^{'}(0,t)$ has a step in T_k, we can assert that $w_i^{'}(0,T_k) > f_i \cdot C(T_k - \Theta_i)$. We also know that $w_i^{'}(0,s_{k+1}) = f_i \cdot C(s_{k+1} - \Theta_i)$. Since in $t \in (T_k, s_{k+1})$, $k > 1$, both curves have a constant slope (which is null for $w_i^{'}(0,t)$ and $f_i > 0$ for the latency-rate curve), we conclude that $w_i^{'}(0,t) > f_i \cdot C(t-\Theta_i)$ when $t \in (T_k, s_{k+1})$, $k > 1$.

Suppose now that $t \in [0, s_2)$. When $t \in [0, \Theta_i)$ the latency-rate curve is null, and therefore the assertion holds; when $t \in [\Theta_i, s_2)$ two sub-cases are given:

1. $\Theta_i \geq T_1$;

   in this case we have $w_i^{'}(0,\Theta_i) \geq f_i \cdot C(\Theta_i - \Theta_i) = 0$ and $w_i^{'}(0,s_2) = f_i \cdot C(s_2 - \Theta_i)$; in the time interval $[\Theta_i, s_2)$ both curves have a constant slope and therefore $w_i^{'}(0,t) \geq f_i \cdot C(t-\Theta_i)$ when $t \in [\Theta_i, s_2)$;

2. $\Theta_i < T_1$;

   by using (29), (30) and (33), after a few algebraic manipulations it is possible to prove that this condition is equivalent to:

$$\phi_i > \frac{1+f_i}{f_i}\overline{L_i} \geq 2\overline{L_i} \qquad (34)$$

   This implies that the curve $w_i^{'}(0,t)$ has the profile shown in Figure 18. By using (29), (30) and (33), it is also easy to prove that (34) implies $\Theta_i \geq t^* = T_1 - (\phi_i - 2L_i)/C$. Therefore the same considerations used for the former cases also apply to this sub-case.

[5] Note that at time T_k the curve has a step.

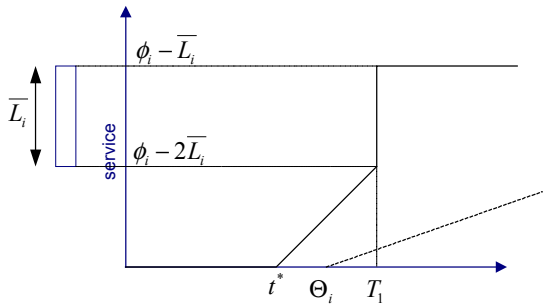


Figure 18. Latency for sub-case 2

This proves that, when $\phi_i \geq \overline{L_i}$, $w_i^{'}(0,t) \geq f_i \cdot C(t-\Theta_i)$, and therefore that Θ_i is an *upper bound* to the latency of DRR. We will now show that Θ_i is a *tight* bound.

In order to prove that the latency bound (5) is tight, we need to prove that there exists a scenario in which the service curve is arbitrarily close to the sloped segment of the latency-rate curve. We prove that it is possible to find a packet sequence for flow i, such that point $(s_2, w_i^{'}(0,s_2))$ (which belongs to the latency-rate curve) is arbitrarily close to point $(s_2, W_i(0,s_2))$.

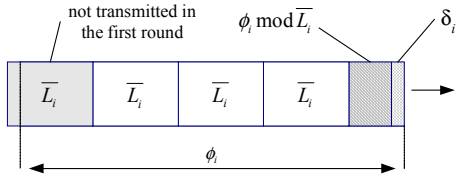


Figure 19. Packets queued at flow i

Let us assume that flow i becomes backlogged at time 0, and at that time $\Delta_j^0 = \overline{L_j} - \delta_j$ with an arbitrarily small δ_j for any $j \neq i$. Since packets can be of arbitrary length, it is possible to assume that each flow $j \neq i$ fully exploits its quantum on each round, carrying a null deficit onto the subsequent round. Assume that flow i has the following packet length sequence: one small packet of length δ_i, one packet of length $\phi_i \bmod \overline{L_i}$ [6], and then $\lfloor \phi_i / \overline{L_i} \rfloor$ packets of length $\overline{L_i}$ (as shown in Figure 19). In this case, flow i will not be able to transmit the last packet of length $\overline{L_i}$ at the first service opportunity, and will therefore accumulate a deficit equal to $\Delta_i^1 = \overline{L_i} - \delta_i$. Let us compute time T_1:

$$T_1 = \frac{1}{C}\left[\sum_{h \neq i}(\phi_h + \Delta_h^0) + (\phi_i - \Delta_i^1)\right]$$
$$= \frac{1}{C}\left[F + \sum_{h=1}^{N}\overline{L_h} - 2\overline{L_i} - (\sum_{h=1}^{N}\delta_h - 2\delta_i)\right]$$

At time T_1 the service curve of flow i reaches a height of $W_i(0,T_1) = \phi_i - \overline{L_i} + \delta_i$. At the second service opportunity, flow i will be able to transmit the remaining maximum length

[6] This can be null if the quantum contains an integer number of maximum length packets.

packet. Since we have assumed that the service curve only increases when the last bit of a packet has been serviced, $W_i(0,t) = W_i(0,T_1) = \phi_i - \overline{L_i} + \delta_i$ until the last bit of the grayed packet is transmitted. Let us compute the time instant t' at which the gray packet is transmitted.

$$t' = T_1 + \frac{1}{C}\left[\sum_{h \neq i} \phi_h + \overline{L_i}\right] = s_2 - \frac{\sum_{h=1}^{N} \delta_h - 2\delta_i}{C}$$

Therefore, point

$$\left(s_2 - \frac{\sum_{h=1}^{N} \delta_h - 2\delta_i}{C}, \phi_i - \overline{L_i} + \delta_i \right)$$

belongs to the worst-case service curve. This point can be made arbitrarily close to $\left(s_2, \phi_i - \overline{L_i}\right)$ (which belongs to the latency-rate curve) by choosing appropriately small δ_j values. This proves that the latency bound derived for $\phi_i \geq \overline{L_i}$ is tight.

b) Case 2: $\phi_i < \overline{L_i}$

In this case, flow i is not guaranteed (as it was for case 1) to transmit packets on each service opportunity.

According to Lemma 1, we can bound the service received by flow i after v service opportunities as follows:

$$W_i(0,t) \geq \max\left\{0, v \cdot \phi_i - \overline{L_i}\right\} \tag{35}$$

Let $v^* = \left\lceil \overline{L_i} / \phi_i \right\rceil$: expression (35) ensures that, after the v^* th service opportunity, flow i has transmitted at least one packet.

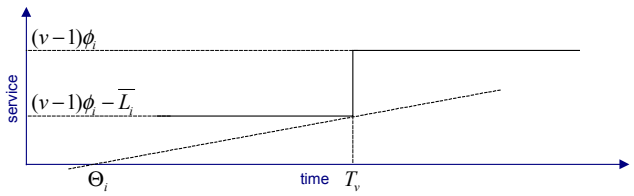


Figure 20. Latency computation for $\phi_i < \overline{L_i}$

Let us assume that flow i transmits a packet during service opportunity $v > v^*$. Then, by Lemma 2, right after time T_v, the curve $W_i(0,t)$ reaches a height which is greater than $(v-1)\phi_i$, as shown in Figure 20. Since packet lengths are upper bounded by $\overline{L_i}$, at time T_v we have:

$$W_i(0,T_v) > (v-1)\phi_i - \overline{L_i} = w_i(0,T_v) \tag{36}$$

The time instant T_v comes "as late as possible" under the same conditions expressed for case 1. Under such conditions, each flow $h \neq i$ receives a service equal $v\phi_h + \overline{L_h}$ in $[0,T_v)$, whilst flow i is serviced for a time $(v-1)\phi_i$ in the same interval. Therefore we have:

$$\begin{aligned} T_v &= \frac{1}{C}\left[\sum_{h \neq i}\left(v\phi_h + \overline{L_h}\right) + (v-1)\phi_i\right] \\ &= \frac{1}{C}\left[v \cdot F + \sum_{h=1}^{N} \overline{L_h} - \left(\overline{L_i} + \phi_i\right)\right] \end{aligned} \tag{37}$$

For any $v > v^*$ in which flow i transmits a packet, the following relationship holds:

$$\frac{w_i(0,T_{v+1}) - w_i(0,T_v)}{T_{v+1} - T_v} = \frac{\phi_i}{F} \cdot C = f_i \cdot C$$

which implies that the points $\left(T_v, w_i(0,T_v)\right)$ are joined by a line of slope $f_i \cdot C$, which we will prove to be the latency-rate curve. Let us denote with Θ_i the intersection of the above-mentioned line with the time axis. We have:

$$\Theta_i = T_v - \frac{(v-1)\phi_i - \overline{L_i}}{f_i \cdot C}$$

By substituting (37) into the above expression, after a few algebraic manipulations we obtain expression (5). According to the methodology described in [3-4], we need to prove that:

$$W_i(0,t) \geq f_i \cdot C(t - \Theta_i) \tag{38}$$

In a worst-case scenario, flow i transmits no packet until time T_{v^*}. Let us then prove that $\Theta_i > T_{v^*}$ as a first step.

$$\begin{aligned} T_{v^*} &\leq \frac{1}{C}\left[\sum_{h \neq i}\left(\phi_h + \overline{L_h}\right) + \left(v^* - 1\right)\sum_{h \neq i}\phi_h + \overline{L_i}\right] \\ &= \frac{1}{C}\left[\sum_{h=1}^{N} \overline{L_h} + v^* F\left(1 - f_i\right)\right] \end{aligned}$$

We also know that:

$$v^* = \left\lceil \overline{L_i} / \phi_i \right\rceil < \overline{L_i} / \phi_i + 1$$

Therefore:

$$T_{v^*} < \frac{1}{C}\left[\sum_{h=1}^{N} \overline{L_h} + \left(\overline{L_i} / \phi_i + 1\right) F\left(1 - f_i\right)\right] = \Theta_i$$

This implies that, for any $t \in [0, \Theta_i]$, (38) holds. We now prove that (38) also holds when $t > \Theta_i$.

Since $W_i(0,t)$ may increase only during a service opportunity, we can assume as a worst case that it remains constant until the end of the *subsequent* service opportunity (say, the

n^{th}) following time t, i.e. at time $T_n \geq t$. Therefore, we only need to prove that (39) holds right before the end of each service opportunity[7], i.e.:

$$W_i(0,T_n) \geq f_i \cdot C(T_n - \Theta_i) \tag{40}$$

Note that $t > \Theta_i$ also implies $n > v^*$.

If flow i has transmitted a packet at the n^{th} service opportunity, we can apply Lemma 2, which ensures that (40) holds. Let us then assume that flow i has transmitted no packets at the n^{th} service opportunity, and let us denote with m the *last* round before round n at which flow i has transmitted a packet. Let Δ_i^m be the deficit value of flow i right after the m^{th} service opportunity. We have $v^* \leq m < n$ by hypothesis, and Lemma 1 ensures that $\Delta_i^n = \Delta_i^m + (n-m)\phi_i < \overline{L_i}$. Therefore we can assert the following:

$$n - m < \left\lceil \frac{\overline{L_i} - \Delta_i^m}{\phi_i} \right\rceil < \frac{\overline{L_i} - \Delta_i^m}{\phi_i} + 1 \tag{41}$$

Since no transmission takes place after round m, we have:

$$W_i(0,T_n) = m \cdot \phi_i - \Delta_i^m . \tag{42}$$

If every flow $h \neq i$ always transmits for the maximum time allowed by DRR, the time instant T_n is upper bounded by:

$$\begin{aligned} T_n &= \frac{1}{C}\left[\sum_{h \neq i} \overline{L_h} + n \sum_{h \neq i} \phi_h + m \cdot \phi_i - \Delta_i^m \right] \\ &= \frac{1}{C}\left[\sum_{h=1}^{N} \overline{L_h} - \overline{L_i} + nF - (n-m)\phi_i - \Delta_i^m \right] \end{aligned} \tag{43}$$

Let us compute the difference $T_n - \Theta_i$:

$$T_n - \Theta_i = \frac{1}{C}\left[(n-m)(F - \phi_i) + mF - \Delta_i^m - F + \phi_i - \overline{L_i}/f_i \right]$$

By substituting (41) into the above expression, after some algebraic manipulations we obtain:

$$T_n - \Theta_i < \frac{1}{C}\left[mF - \frac{\Delta_i^m}{f_i} - \overline{L_i} \right] \tag{44}$$

and therefore:

$$f_i \cdot C(T_n - \Theta_i) < m\phi_i - \Delta_i^m - f_i \overline{L_i} \tag{45}$$

By putting (45) and (42) together we obtain:

$$W_i(0,T_n) = m \cdot \phi_i - \Delta_i^m > m\phi_i - \Delta_i^m - f_i \overline{L_i} > f_i \cdot C(T_n - \Theta_i)$$

and therefore (40) holds. This proves that expression (5) is an *upper bound* to the latency when $\phi_i < \overline{L_i}$. In order to prove that (5) is a *tight* bound, we need to build a scenario in which a point of the service curve can be arbitrarily close to a point of the sloped segment of the latency-rate curve.

Let us assume that flow i becomes backlogged at time 0, and at that time $\Delta_j = \overline{L_j} - \delta_j$ with an arbitrarily small δ_j for any $j \neq i$. Since packets can be arbitrarily small, it is possible to assume that each flow $j \neq i$ fully exploits its quantum on each round, carrying a null deficit onto the subsequent round. Let us express L_i as $L_i = k \cdot \phi_i + r$, with $k \geq 1$, $0 \leq r < \phi_i$, and let us assume that flow i has the following packet length sequence:

- $k+1$ packets of length ϕ_i;
- one packet of length $\phi_i - (r - \delta_i)$;
- one packet of length $\overline{L_i}$;
- one packet of length $l > \phi_i$.

It is easy to see that, after $k+2$ service opportunities (i.e. after the packet of length $\phi_i - (r - \delta_i)$ has been transmitted), flow i has a deficit equal to $\Delta_i^{k+2} = r - \delta_i$. The subsequent packet (of length $\overline{L_i}$) is transmitted during the $2k+3^{\text{th}}$ service opportunity, after which flow i has a deficit equal to $\Delta_i^{2k+3} = \phi_i - \delta_i$, which is in accord with Lemma 2. Therefore, we can assert that:

$$W_i(0,T_{2k+3}) = (k+2)\phi_i - (r - \delta_i) \tag{46}$$

We will show that point $(T_{2k+3}, W_i(0,T_{2k+3}))$ can be arbitrarily close to the latency-rate curve. Let us compute the time instant T_{2k+3}:

$$T_{2k+3} = \frac{1}{C}\left[\sum_{h=1}^{N} (\overline{L_h} - \delta_h) + 2\delta_i + (2k+3)F - (k+1)\phi_i - r \right] \tag{47}$$

We can then compute the latency-rate curve value at time $t = T_{2k+3}$, i.e. expression $f_i \cdot C(T_{2k+3} - \Theta_i)$:

$$f_i \cdot C(T_{2k+3} - \Theta_i) = (k+2)\phi_i - r + f_i\left(-\sum_{h=1}^{N} \delta_h + 2\delta_i \right) \tag{48}$$

By comparing (46) and (48) it clearly emerges that, by choosing arbitrarily small δ_h, points $(T_{2k+3}, W_i(0,T_{2k+3}))$ and $(T_{2k+3}, f_i \cdot C(T_{2k+3} - \Theta_i))$ can be made arbitrarily close. This proves that the latency bound (5) is tight when $\phi_i < \overline{L_i}$.

■

7 Note that we have assumed that the service curve increases right after the end of a packet transmission, and therefore right after the end of a service opportunity.