Tradeoffs Between Low Complexity, Low Latency, and Fairness With Deficit Round-Robin Schedulers

Luciano Lenzini, Enzo Mingozzi, and Giovanni Stea

***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 original DRR implementation, 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.**

***Index Terms*—Deficit round-robin (DRR), quality of service (QoS), scheduling algorithms.**

I. Introduction

MULTISERVICE 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, multiservice 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.

Manuscript received April 29, 2002; revised April 3, 2003; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor N. Shroff. This work was supported by the Italian *Ministero dell'Istruzione, dell'Università e della Ricerca (MIUR)* under the framework of the Project "High Quality Web Systems."

The authors are with the Dipartimento di Ingegneria della Informazione, University of Pisa, 56122 Pisa, Italy (e-mail: l.lenzini@iet.unipi.it; e.mingozzi@iet.unipi.it; g.stea@iet.unipi.it).

Digital Object Identifier 10.1109/TNET.2004.833131

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. First, 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 tradeoff between operational overhead and performance. We propose different solutions for implementing Aliquem, employing different data structures and requiring different space occupancy and operational overhead. In addition, we present a variant of the

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

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, while 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:

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

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:

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

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]:

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

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 $\overline{L_1} = \overline{L_2} = \ldots = \overline{L_N} = \overline{L}$ are scheduled, and that $\phi_j = \overline{L}/100$, $j = 1 \ldots N$. According to (4) the latency should be

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

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 [26].

Theorem 1: The latency of DRR is

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

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

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

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 $(t_1, t_2]$:

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

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

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

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 $\rho_1, \rho_2, \ldots, \rho_N$ 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:

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

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:

$$\frac{\phi_i}{\phi_j} = \frac{\rho_i}{\rho_j} \qquad \forall i, j, \quad i \neq j. \tag{9}$$

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

$$f_i \triangleq \frac{\phi_i}{\sum_{j=1\ldots N}\phi_j} = \frac{\phi_i}{F}. \tag{10}$$

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

$$f_i = \frac{\rho_i}{\sum_{j=1\ldots N}\rho_j}. \tag{11}$$

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

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

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

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

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 kb/s each share a 10-Mb/s link with 9 video flows requiring 1 Mb/s each. The maximum packet length is $\overline{L} = 1500$ bytes for all flows. According to our scheme, we obtain $F^S = 300$ kB (240 ms at 10 Mb/s), and each video flow has a quantum of 30 kB, 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.

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 Rth subsequent round, where R is computed as follows:

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

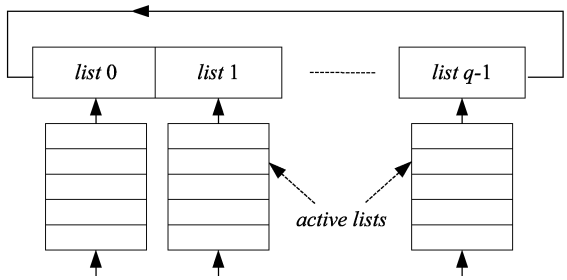

Fig. 1. Active list queue.

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 Fig. 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 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:

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

The pseudo-code for Aliquem DRR is reported in Fig. 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.

```
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();
   }
}
```

Fig. 2. Pseudo-code for Aliquem DRR.

- 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 nonempty 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 $2 \cdot q \cdot \texttt{sizeof(reference)}$.

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.

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

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

When the flows in the current active list have all been serviced, the next nonempty 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 nonempty 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 nonnull 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 `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 Fig. 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 nonempty 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 $U = \{0, 1, 2, \ldots, L\}$, 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

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

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

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 nonstandard (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 $\{0,1,2,\ldots,q-1\}$ of the nonempty active lists. Thus, finding the next nonempty active list implies executing the pseudo-code reported in Fig. 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 nonempty. Thus, the M words can be regarded as a bit vector 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 Fig. 5). In the X-ary tree, each nonleaf node is itself a word, whose x^{th} bit, $x = 0\ldots\mathrm{X}-1$, is set if there is at least one nonempty list in the node's xth sub-tree.

Given an Aliquem structure with q active list, the X-ary tree structure has a $\lceil \log_{\mathrm{X}} q \rceil$ depth, and therefore it is possible to locate the next nonempty list in $\lceil \log_{\mathrm{X}} q \rceil$ 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 $\mathrm{X} = 32$. It is possible to manage $q = 1$ K active lists with a 2-level tree, or $q = 32$ K 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, $\sim$4 kB 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

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

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:

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

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

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

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:

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

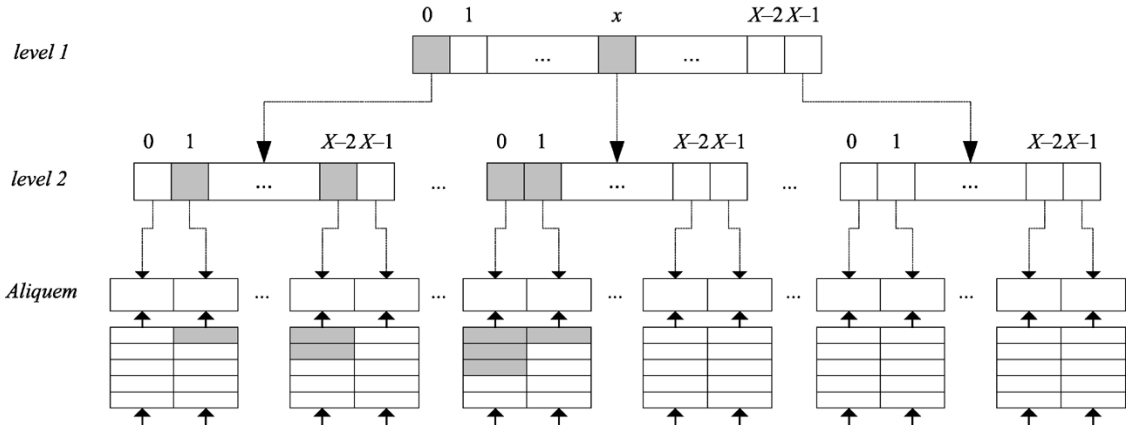

Fig. 5. Example of a 2-level bit vector tree for accessing an Aliquem structure.

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

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

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 nondecreasing 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 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 Fig. 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 Fig. 6(c)]. Consider the same scenario in Aliquem DRR: both flows are initially queued in the kth active list (assume for simplicity that $k < q-3$), and flow i is queued *before* flow j [see Fig. 6(b)]. However, even though both flows are constantly backlogged from round k to round $k+3$, the packet

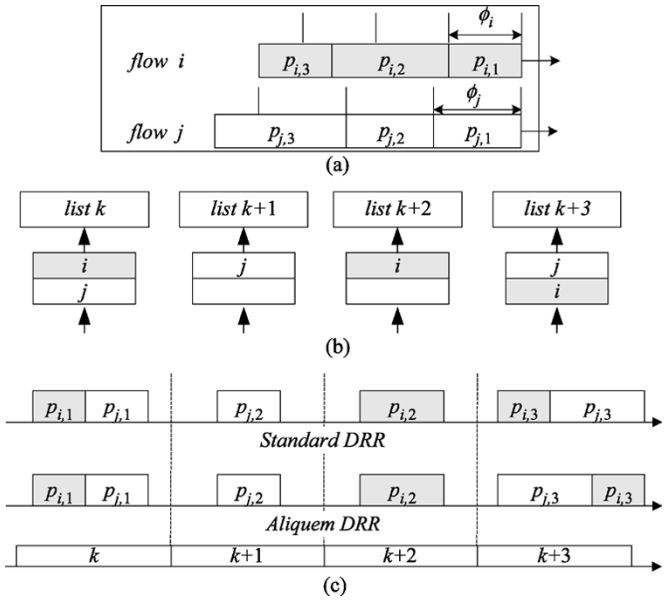

Fig. 6. Inversion in the service order. (a) Packets queued at flows i and j; (b) flow queueing in Aliquem DRR; (c) packet transmission sequence.

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

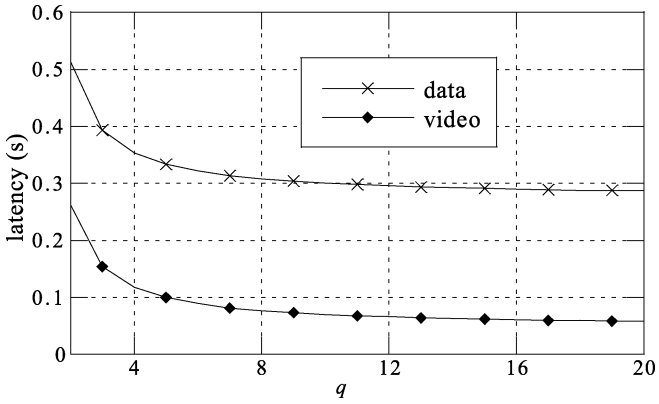

Fig. 7. Latency gain against q.

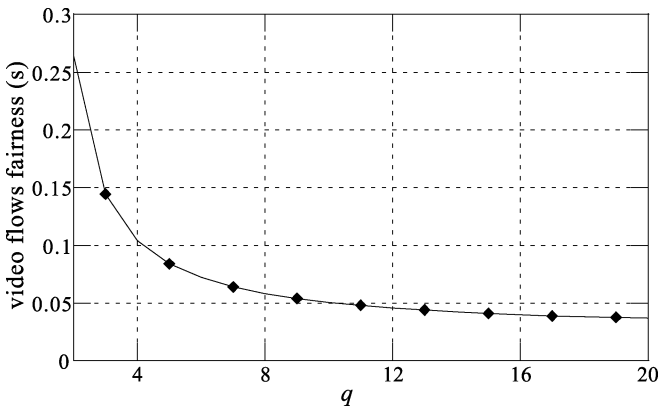

Fig. 8. Fairness gain among video flows against q.

DRR fairness measure, it is straightforward to prove that the fairness measure of Aliquem DRR is:

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

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. Tradeoff 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 Figs. 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 Fig. 2, it services each flow exhaustively before dequeuing the next one. Therefore, a burst of packets of overall length at most equal to $\overline{L_i}+\phi_i$ 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 multinode 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. Fig. 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 Fig. 10.

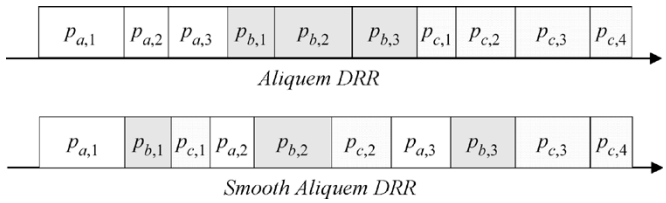

Fig. 9. Aliquem DRR and Smooth Aliquem DRR output.

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.

```
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();
   }
}
```

Fig. 10. Pseudo-code for Smooth Aliquem DRR.

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 nonempty 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:

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

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

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 may require as much as $O(N+Z\log Z)$ operations. Clearly, the complexity of Aliquem DRR is instead much lower. As far as latency is concerned, a tight bound for PDRR is computed in [27]. If $Z \approx q$, the bound obtained therein is comparable to that of Aliquem DRR.

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 kB. 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 nonempty 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 $\rho_1, \rho_2, \ldots, \rho_N$ 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:

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

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

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)

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

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 $\overline{L_i}$.

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].

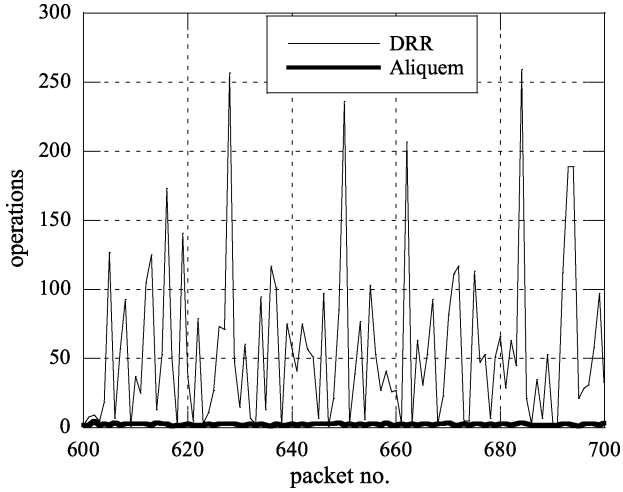

Fig. 11. Per-packet operations comparison.

A. Operational Overhead

We consider a scenario consisting of a single node with a 1-Mb/s 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 kb/s, and the remaining twenty require 40 kb/s. 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-kb/s sources and $\phi_i = 6000$ bytes for the 40-kb/s sources, which yield a frame length $F^S = 150$ kB. 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 Fig. 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.

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*

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

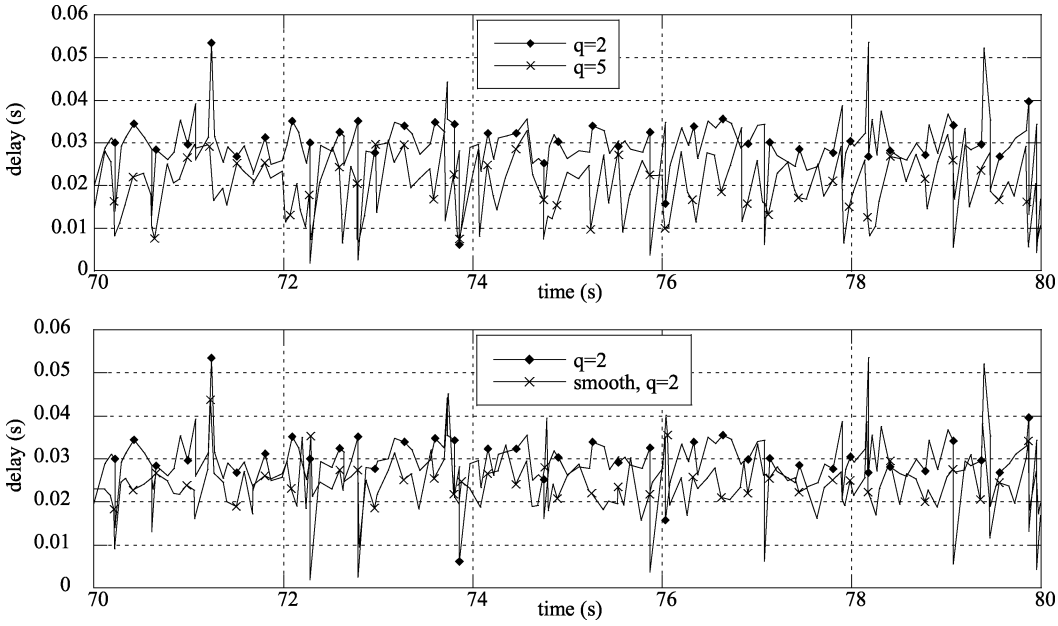

Fig. 12. Delay trace comparison.

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-Mb/s 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 kb/s. The other 39 sources transmit UDP packets with lengths uniformly distributed between 100 and 1500 bytes. Nineteen traffic sources require 100 kb/s, and the remaining 20 require 400 kb/s. 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 I. 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. Fig. 12 shows two delay traces, from which the benefits introduced by Aliquem and Smooth Aliquem are also evident.

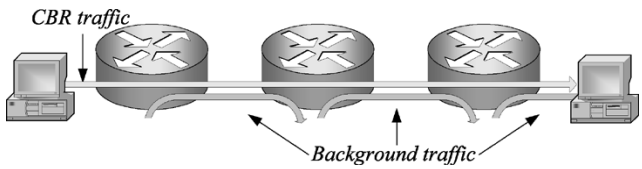

Fig. 13. Simulation scenario.

C. Delay Variation in a Multinode 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-Mb/s links. A CBR source sends 125-bytes packets with 10-ms period across the path, thus occupying 100 kb/s. 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 Mb/s each, 5 sources require 0.6 Mb/s each, the remaining 10 require 0.15 Mb/s each. To avoid correlation

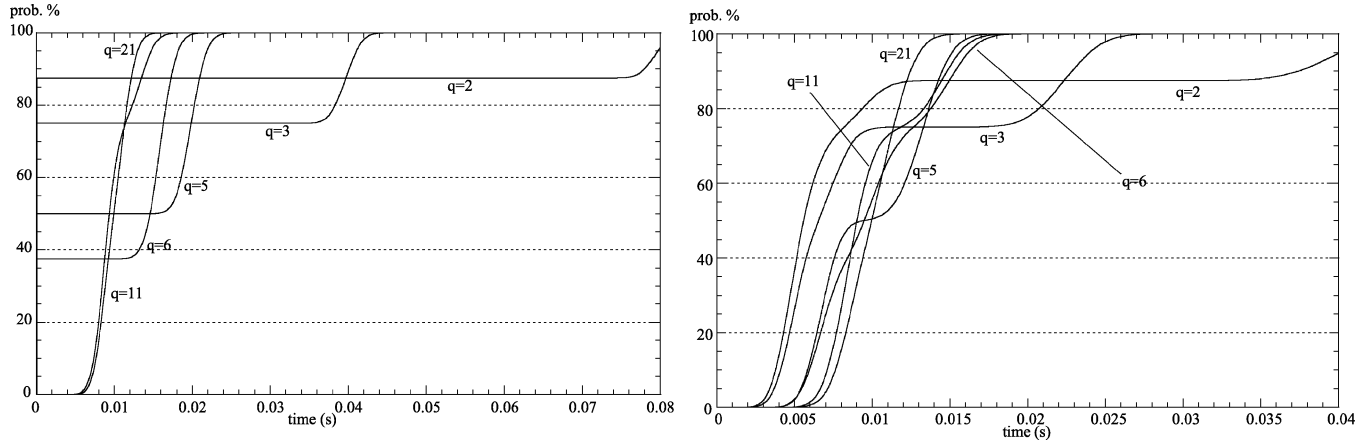

Fig. 14. Distribution of the inter-packet spacing for the CBR flows for various values of q in Aliquem DRR (left) and Smooth Aliquem DRR (right).

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 Fig. 13.

We simulate the network for 350 s employing both Aliquem DRR and Smooth Aliquem DRR as schedulers. We select the frame length according to (19), which yields $F = 100/(q-1)$ kB, 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 10 ms). 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 10 ms: in this case, the narrower the distribution is, the closer the ideal case is approximated. Fig. 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 to 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 multihop environment.

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

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. 10th Int. Workshop on Quality of Service (IWQoS)*, Miami Beach, FL, May 2002, pp. 77–86.

[2] H. Zhang, "Service disciplines for guaranteed performance service in packet-switching networks," *Proc. IEEE*, vol. 83, pp. 1374–1396, Oct. 1995.

[3] D. Stiliadis and A. Varma, "Latency-rate servers: A general model for analysis of traffic scheduling algorithms," *IEEE/ACM Trans. Networking*, vol. 6, pp. 675–689, Oct. 1998.

[4] ——, "Latency-rate servers: A general model for analysis of traffic scheduling algorithms," Univ. California, Santa Cruz, Tech. Rep. CRL-95-38, 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. Networking*, vol. 1, pp. 344–357, June 1993.

[6] S. J. Golestani, "A self-clocked fair queueing scheme for broadband applications," in *Proc. IEEE INFOCOM*, Toronto, Canada, June 1994, pp. 636–646.

[7] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round-robin," *IEEE/ACM Trans. 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. IEEE INFOCOM*, Kobe, Japan, Apr. 1997, pp. 557–565.

[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. Networking*, vol. 5, pp. 690–704, Oct. 1997.

[10] J. Bennett and H. Zhang, "Hierarchical packet fair queueing algorithms," *IEEE/ACM Trans. Networking*, vol. 5, pp. 675–689, Oct. 1997.

[11] F. Toutain, "Decoupled generalized processor sharing: A fair queuing principle for adaptive multimedia applications," in *Proc. IEEE INFOCOM*, San Francisco, CA, Mar.–Apr. 1998, pp. 291–298.

[12] D. Saha, S. Mukherjee, and K. Tripathi, "Carry-over round-robin: A simple cell scheduling mechanism for ATM networks," *IEEE/ACM Trans. Networking*, vol. 6, pp. 779–796, Dec. 1998.
[13] S.-C. Tsao and Y.-D. Lin, "Pre-order deficit round-robin: A new scheduling algorithm for packet switched networks," *Comput. Netw.*, vol. 35, pp. 287–305, Feb. 2001.
[14] G. Chuanxiong, "SRR: An $O(1)$ time complexity packet scheduler for flows in multi-service packet networks," in *Proc. ACM SIGCOMM*, San Diego, CA, Aug. 2001, pp. 211–222.
[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," *Comput. Netw.*, vol. 37, pp. 561–578, Nov. 2001.
[16] S. S. Kanhere, H. Sethu, and A. B. Parekh, "Fair and efficient packet scheduling using elastic round-robin," *IEEE Trans. Parallel Dist. Syst.*, vol. 13, pp. 324–336, Mar. 2002.
[17] L. Lenzini, E. Mingozzi, and G. Stea, "A unifying service discipline for providing rate-based guaranteed and fair queueing services based on the timed token protocol," *IEEE Trans. Computers*, vol. 51, pp. 1011–1025, Sept. 2002.
[18] ——, "Packet timed token service discipline: A scheduling algorithm based on the dual-class paradigm for providing QoS in integrated services networks," *Comput. Netw.*, vol. 39, pp. 363–384, July 2002.
[19] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated services," The Internet Society, RFC 2475, Dec. 1998.
[20] B. Davie *et al.*, "An expedited forwarding PHB (Per-Hop Behavior)," The Internet Society, RFC 3246, Mar. 2002.
[21] A. Charny *et al.*, "Supplemental information for the new definition of the EF PHB (Expedited Forwarding Per-Hop Behavior)," The Internet Society, RFC 3247, Mar. 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 I: Sorting and Searching (EATCS Monographs on Theoretical Computer Science)*. New York: Springer-Verlag, 1984.
[24] A. Brodnik, S. Carlsson, J. Karlsson, and J. I. Munro, "Worst case constant time priority queue," in *Proc. 12th ACM/SIAM Symp. Discrete Algorithms*, Washington, DC, Jan. 2001, pp. 523–528.
[25] *The Network Simulator—ns-2*, http://www.isi.edu/nsnam/ns/.
[26] L. Lenzini, E. Mingozzi, and G. Stea, "Full exploitation of the Deficit Round-Robin capabilities by efficient implementation and parameter tuning," Univ. of Pisa, Italy, Tech. Rep., Oct. 2003.
[27] S. S. Kanhere and H. Sethu, "On the latency bound of pre-order deficit round-robin," in *Proc. IEEE Conf. Local Computer Networks*, Tampa, FL, Nov. 2002, pp. 508–517.

Luciano Lenzini received the degree in physics from the University of Pisa, Italy.

Since 1994, he has been a Full Professor at the Department of Information Engineering at the University of Pisa. He has also worked extensively for CNUCE, an Institute of the Italian National Research Council (CNR). His current research interests include the design and performance evaluation of MAC protocols for wireless networks and the quality of service provision in integrated and differentiated services networks.

Enzo Mingozzi received the Laurea and Ph.D. degrees in computer systems engineering from the University of Pisa, Italy, in 1995 and 2000, respectively.

He is currently an Assistant Professor at the University of Pisa. Since 1995, he has been involved in several national and international projects including, among others, Eurescom P810, "Wireless ATM access and advanced software techniques for mobile networks architecture," and IST Wineglass, "Wireless IP NEtwork as a Generic platform for Location Aware Service Support." He also took part in the standardization process of the HIPERLAN/2 and HIPERACCESS protocols, in the framework of the ETSI project BRAN. He served as the tutorial chair on the European Wireless 2002 conference committee. His current research interests focus on the design and analysis of MAC protocols for wireless networks, and QoS provisioning and service integration in computer networks.

Giovanni Stea received the Laurea and the Ph.D. degrees in computer systems engineering from the University of Pisa, Italy, in 1999 and 2003, respectively.

Since December 2004, he has been a Assistant Professor (Ricercatore) at the Department of Information Engineering of the University of Pisa. He has been and is involved in national and European research projects. He has served as a Member of the Technical Program Committee and Local Arrangements Chairman for the European Wireless 2002 international conference. His current research interests include network scheduling, quality of service in multiservice networks, and network architectures.