Mini Round Robin:
An Enhanced Frame-Based Scheduling Algorithm for Multimedia Networks

T. Al-Khasib, H. Alnuweiri, H. Fattah, V.C.M. Leung
Department of Electrical & Computer Engineering
University of British Columbia
Vancouver, B.C. V6T 1Z4, CANADA
Email: tariqa@ece.ubc.ca

Abstract- The broad spread of packet data networks and the emergence of applications in multimedia communications, created a driving force towards an improved Quality of Service (QoS) model for today’s Internet. A primary component of this model is packet schedulers. We introduce a new frame-based scheduling technique called Mini Round Robin (MRR) that is primarily designed for providing lower latency bounds, and lower start-up latency bound for low-rate but high-priority flows. This enables applications such as Voice-over-IP to demand low delay despite the low reserved bit rate of the voice sessions.

Keywords- Mini Round Robin, Deficit Round Robin, Elastic Round Robin, Quality of Service, frame-based scheduling.

1 INTRODUCTION

Packet schedulers play an essential role in providing per-flow or per-class QoS guarantees in packet data networks. The QoS guarantees include bounded delay, guaranteed bit rate and fair service allocation to contending flows. This is achieved by solving the contention problem over shared network resources and by deciding the sequence in which packets depart the network node.

Modern packet scheduling theory is based on few, but very powerful, concepts. One such concept is Generalized Processor Sharing (GPS) [1][2]. GPS is an ideal scheduler that serves an infinitesimal amount of data from each flow according to the flow’s reserved rate or relative bandwidth weight. GPS can provide every flow with its guaranteed bit rate, and at the same time, distribute excess bandwidth fairly among all contending flows according to their relative reserved rates. GPS is based on a fluid model that is, unfortunately, not implementable in packet data networks. In reality, a packet has to finish service completely before the server starts serving another packet.

Packet schedulers can generally be classified into two main categories: timestamp-based schedulers, and frame-based schedulers. In timestamp-based schedulers packets are given a time related stamp upon their arrival to the system. The time stamp is used by the scheduler to determine the sequence in which packets depart the system. This category of schedulers includes Weighted Fair Queuing (WFQ) [1][3], Worst-case Fair Weighted Fair Queuing (WFQ) [4], Virtual Clock (VC) [5], and Self-Clocked Fair Queuing (SCFQ) [6]. Timestamp-based schedulers can achieve a good approximation of the GPS fluid model by providing tight and low latency bounds, and providing good fairness. A major drawback of these schedulers is their unavoidable high work complexity, despite the significant improvements that have been proposed recently [7].

Frame-based schedulers, on the other hand, serve flows in rounds or frames. During each round a flow receives at least one transmission opportunity. Weighted Round Robin (WRR) [8], Deficit Round Robin (DRR) [9], and Elastic Round Robin (ERR) [10][11], are all examples of frame-based schedulers. This type of schedulers is generally known for its simplicity and low work complexity.

With link capacities reaching 100 Gigabits per second (Gbps), a typical “wire-speed” packet processor needs to handle millions of packets per second, and the number of flows served by a single node could grow up to tens of thousands of flows. Such network conditions give lead to simple and efficient schedulers over complex yet better ones. In this paper, and for the reasons above, we are interested in frame-based schedulers, for their well-known simplicity and efficiency.

The rest of this paper is organized as follows: Section 2 presents an overview of the most common scheduling algorithms. In particular, frame-based schedulers upon which our new algorithm is based. In sections 3 and 4, we introduce the Mini Round Robin scheduling algorithm, we explain the rationale behind it and provide analytical results on the work complexity, startup latency, fairness and latency bound. Section 5 is dedicated for results obtained from the OPNET simulation tool via network models written for each one of the algorithms under investigation and we conclude the paper in section 6.

2 PREVIOUS WORK

Round Robin (RR) is one of the earliest, simplest, and most widely used frame-based scheduling techniques, upon which all frame-based scheduling algorithms were later based. In RR, backlogged sessions are served in sequence, one packet at a time. All flows are treated similarly regardless of their reserved rates. The RR algorithm is fair as long as the same fixed packet size is used for all
contending flows, and all flows have the same reserved rate. RR is simple because it is stateless and has an $O(1)$ per packet work complexity. However, under more realistic assumptions, such as variable packet sizes and session rates, RR performs quite poorly both in terms of fairness and delay.

To support flows with different reserved rates, Weighted Round Robin (WRR) was introduced in [8]. A WRR scheduler serves multiple packets from a flow according to the flow’s normalized weight. The weight of a flow $i$ ($w_i$) is defined as its relative share of the total link bandwidth. As in RR schedulers, using different packet sizes by different flows may cause the WRR scheduler to be unfair.

Deficit Round Robin (DRR) [9] was the first frame-based scheduling algorithm to overcome the unfairness caused by different packet sizes used by different flows. DRR assigns each flow $i$ a Quantum ($Q_i$). The quantum of a flow is proportional to the flow’s weight and it represents the ideal amount of service the flow should receive in each round. Flows not consuming their entire quantum in a round get the chance to transmit more data in consecutive rounds as long as they have data to transmit. The quantum is added to the Deficit Counter ($DC_i$) of each flow at the beginning of each round. Packets are served from a flow as long as it maintains a positive $DC_i$. If the quantum assigned to each flow should be greater or equal to the maximum packet size that could potentially arrive to the system, for the DRR algorithm to operate with an $O(1)$ per-packet work complexity. If the quantum assigned to a flow is significantly higher than the maximum size of the packets that actually arrive to the system, this could increase the short-term unfairness of the system, leading to a higher latency bound. The DRR algorithm also requires knowledge of the packet size prior to scheduling it, this piece of information could be available in the packet header of IP packets, but may not be available for some networks such as wormhole networks [10].

Elastic Round Robin (ERR) [10][11] introduced the concept of a variable quantum that depends on the performance of flows in the previous round. ERR allows a flow to exceed its allowance by a maximum of one packet size, and a Surplus Counter ($SC_i(r)$) keeps track of the excess service the flow has received in round $r$. After each round $r$, the Maximum relative Surplus Counter ($MaxSC(r)$) is calculated and then used to calculate the new Allowance ($A_i(r)$) of each active flow to be served in the following round. The allowance represents the least amount of data a flow can send in a round as long as it has packets to transmit. Compared to DRR, ERR does not require knowledge of the maximum packet size, and at the same time can provide better short term fairness and lower latency bounds while maintaining an $O(1)$ per packet computational complexity.

Another frame-based scheduling algorithm is the Nested Deficit Round Robin (NDRR)[12]. NDRR splits each round in DRR into smaller inner rounds, and executes a version of DRR within the inner rounds. The flow receives its assigned quantum ($Q_{min}$ at a time) distributed over several inner rounds, where $Q_{min}$ is the quantum assigned to the flow with the lowest reserved rate. As in DRR, NDRR requires knowledge of the maximum packet size ($M$) that may eventually arrive to the system, for the NDRR scheduler to operate with an $O(1)$ per packet work complexity. If the actual packets arriving to the system have a size that is much less than $M$, NDRR will become unnecessarily less fair and flows will get higher latencies. NDRR also requires the knowledge of the packet size prior to scheduling it.

Two recently proposed frame-based schedulers are the Pre-order Deficit Round Robin (PDRR) [13] and the Prioritized Elastic Round Robin (PERR) [14][15]. Both techniques add a limited number of priority queues in which all active flows are sorted according to their quantum utilization. At the beginning of each round, PDRR classifies packets into priority queues according to the quantum availability of the corresponding flow. On the other hand, PERR has to sort only flow numbers into the priority queues, thus reducing the buffering requirements and processing time.

PDRR is based on DRR, while PERR is based on ERR and thus inherits ERR’s advantages over DRR. It was shown in [14][15] that PERR provides lower latency bounds and better fairness than PDRR, while maintaining the same per-packet work complexity of $O(p)$, $p$ being the number of priority queues.

PDRR and PERR show some improvement over DRR and ERR with respect to the latency bound and fairness, but at the expense of an increased per-packet processing complexity. As was said earlier, both PDRR and PERR have an $O(p)$ per-packet work complexity, where $p$ is the number of priority queues. As the number of priority queues increases, the work complexity of the system increases, to reach a maximum of an $O(n)$ per-packet work complexity, which is the work complexity of the PGPS algorithm. From this perspective we see that it is unfair to compare these two algorithms to other $O(1)$ algorithms like DRR and ERR.

As was discussed earlier, ERR shows some improvements over DRR and PERR shows some improvements over PDRR. NDRR tries to provide flows that have a low reserved rate with a lower latency bound. We will use these three algorithms, ERR, PERR and NDRR, as a basis for evaluating the performance of our new algorithm.

Unfortunately, these three algorithms were designed under one misconceived assumption that sessions were always backlogged. In reality, session traffic varies widely over time and sessions are not always backlogged. Thus, the performance of a scheduling algorithm should be studied and tested under such a condition. A deeper look into the description of ERR, PERR and NDRR schedulers, reveals some serious problems that should be fixed before any of these algorithms could be adopted. In these three algorithms, a flow that continuously switches between active and idle states can get more or less service than its reserved rate. This kind of unfairness will severely affect the
latency and delay experienced by that flow and, worse, affect the performance of other flows.

3 MINI ROUND ROBIN (MRR)

As in all frame-based scheduling algorithms, MRR serves flows in rounds, and a flow gets the opportunity to transmit packets at least once every round. Unlike ERR, MRR divides each round into multiple mini-rounds. Flows in ERR transmit their entire allowance once they receive their first transmission opportunity, while in MRR we allow a flow to transmit only a single packet during each mini-round as long as the flow has a positive balance, and it has packets to transmit.

MRR borrows the concept of mini-rounds from NDRR algorithm, which divides each outer round into multiple inner rounds. Unlike NDRR, which is based on DRR, MRR is based on ERR and thus it does not require knowledge of the maximum packet size that may potentially arrive to the system. It should be emphasized that MRR is not simply a Nested-ERR algorithm in the sense that it does not assign a fixed quantum or allowance that has to be consumed by a flow during an inner round. Rather, during a mini-round in MRR execution, each active flow having a positive balance is allowed to transmit a single packet.

In the MRR algorithm, we maintain two linked lists: The ActiveFlowsList and the MiniRoundList. The first list keeps track of all active flows having a balance less than or equal to zero and the second list, the MiniRoundList, holds all active flows having a positive balance.

At the beginning of each round, the contents of the ActiveFlowsList are moved to the MiniRoundList. The flows in the MiniRoundList are then served in order and one packet at a time. After each mini-round, flows with negative or zero balances are excluded and a new mini-round is started. When the balance for all the flows becomes negative or zero, a new major (outer) round is started with the flows’ balance updated according to equation (3), and a new series of mini-rounds is started. $DC_i(r)$ is the deficit counter and it corresponds to the deficit in service for flow $i$ is round $r$.

$$DC_i(r) = Sent_i(r) - Balance_i(r)$$ (1)

$$MaxD(r) = \max_{\forall i \text{ served in round } r} \left\{ \frac{DC_i(r)}{w_i} \right\}$$ (2)

$$Balance_i(r) = w_i(1 + MaxD(r-1)) - DC_i(r-1)$$ (3)

After scheduling a packet for transmission, the corresponding session’s reference number is appended back to the tail of one of the two linked lists as long as the flow has more packets waiting in its queue. The flow’s balance determines the list to which the flow is added. If the Balance was positive, the flow is added to the tail of the MiniRoundList. Otherwise it is added to the tail of the ActiveFlowsList.

Figure 1 shows a sample trace of a server running the MRR scheduler. The three flows (1, 2, and 3) are always backlogged and equally share the output link. We assume that the three flows became active at the beginning of the first round. Upon the arrival of the first packet of each flow, the flow is initialized with a zero balance, and directly inserted into the MiniRoundList. When a flow receives its first transmission opportunity in a round, its balance is calculated according to equation (3). The three flows get a unity balance in the first round, and transmit a single packet that will cause there balance to drop below zero. The first round ends given that all flows have a negative balance, with flow 1 having the maximum deficit (MaxD) in that round. The second round consists of three mini-rounds. In the first mini-round, the MaxD of the previous round is used to calculate the balance of each flow according to (8), and each flow is given the chance to transmit exactly one packet. After transmitting a packet of length 10 units, flow 1 balance becomes negative, and thus it is moved back to the ActiveList while the other two flows continue to the second mini-round. In mini-round 2, flow 2 overdraws its balance and joins flow 1 in the ActiveList, and flow 3 moves on to the third and last mini-round (of round 2).

**Figure 1: MRR sample trace**

<table>
<thead>
<tr>
<th>Round Number</th>
<th>Mini-Round Number</th>
<th>Balance BEFORE</th>
<th>Balance AFTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>16</td>
<td>1 -15</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>1 -7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>1 -3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>10</td>
<td>1 -9</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>9 -3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>13 -9</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>12</td>
<td>1 -11</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>9 -1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>3 -1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>5 -1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>1 -7</td>
<td></td>
</tr>
</tbody>
</table>

4 ANYLATIC RESULTS

In this section we apply theoretical analysis to derive the MRR work complexity, start-up latency, fairness, and latency bounds. Detailed proof will not be shown, and it can be found in [16].
4.1 Computational Complexity
The per-packet computational complexity of an MRR scheduler is \( O(1) \). This is achieved through having a dequeue and an enqueue processes each of which has an \( O(1) \) per packet work complexity.

4.2 Start-up Latency
The lifetime of any flow that is not always backlogged consists of a series of busy and idle periods. The start-up latency can be defined as the maximum time difference between the arrival time of the first packet in a busy period and the time the last bit of that packet is transmitted. The least the start-up latency the better the scheduling algorithm is.

In MRR, when a flow \( i \) becomes active, the upper bound on the start-up latency, \( S_{MRR} \), experienced by the first packet of flow \( i \) is

\[
S_{MRR} \leq \frac{(n-1)m}{r} + \frac{m}{r},
\]

where \( r \) is the service rate of the server, \( m \) is the size in bits of the largest packet actually served during the execution of a scheduling algorithm, and \( n \) is the total number of flows.

A lower start-up latency bound is an important feature a scheduling algorithm could have, especially if control messages or low rate and real time flows were involved. MRR provides the lowest start-up latency bounds among ERR, PERR, NDRR, and other frame-based scheduling algorithms.

4.3 Fairness
Let \( \text{Sent}_i(t_1,t_2) \) be defined as the service received by flow \( i \) during the period \((t_1,t_2)\), and define the Relative Fairness (RF) over the interval \((t_1,t_2)\), \( RF(t_1,t_2) \) as:

\[
RF(t_1,t_2) = \max \left( \frac{\text{Sent}_i(t_1,t_2)}{w_i}, \frac{\text{Sent}_j(t_1,t_2)}{w_j} \right)
\]

\( \forall \ i \ and \ j \ in \ Active \ flows \ during \ (t_1,t_2) \)

If we define the Relative Fairness Bound (RFB) as the maximum RF over all intervals of time, then for any execution of the MRR scheduling discipline, \( RFB \leq 4m - 2 \)

4.4 Latency Bound
In LR servers [17], the latency of a flow is defined as the minimum non-negative constant \( \Theta_i \) that satisfies the following:

\[
\text{Sent}_i(t_j,t) \geq \max \left( 0, \rho_i(t - \alpha_i - \Theta_j) \right)
\]

for all busy periods of flow \( i \)

For any execution of the MRR scheduling discipline, the upper bound on the latency \( \Theta_i^{MRR} \) experienced by flow \( i \) is:

\[
\Theta_i^{MRR} = \left\{ (n-1)(m-1) + \sum_{j \in F} w_j m + \max (w_j - 1) \left[ xm - \left( \frac{W}{w_j} - 1 \right) s \right] + (y - z)m \right\}
\]

The quantities \( x, y, \) and \( z \) are quantities calculated using a worst case scenario that is detailed in [16].

As an example, assume having 10 continuously backlogged flows with weights varying between 1 and 100. A server running MRR, ERR, or PERR is serving these flows with an output link of 21.44 Mbps. The packet size distribution is uniform between 64 and 576 bytes. Figure 2 shows a comparison between the latency bound provided by each one of the three schedulers. As we can see the MRR scheduler provides the lowest latency bound for flows with small reserved rates. We should also keep in mind that the PERR algorithm has a higher work complexity than ERR and MRR.

![Figure 2: Analytical latency bound of MRR, ERR, and PERR](image)

Unlike ERR or PERR, MRR provides a latency bound that depends on the ratio between the largest packet size \( m \) and the smallest packet size \( s \). Using the same example above, figure 3 shows how changing the size \( s \) of the smallest packet can affect the latency bound.

5 SIMULATION RESULTS
In this section we present performance and validation tests obtained from simulations conducted using OPNET [18] network models built for each one of the scheduling algorithms under consideration.

5.1 Simulation Set-up
Table 1 provides a description of the multimedia traffic sources used to evaluating the MRR performance. The first 20 flows are constant bit-rate (CBR) voice-over-IP flows
with a bit rate of 32 kbps each. These flows are mixed with 5 high bit rate video flows, 1 Mbps each, and an aggregate of best effort flows that are given the excess bandwidth present on the link.

<table>
<thead>
<tr>
<th>Flow number</th>
<th>Flow description</th>
<th>Bit rate (kbps)</th>
<th>Packet size (Bytes)</th>
<th>Burst size (Packets)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-20</td>
<td>Voice</td>
<td>32</td>
<td>64-128</td>
<td>1</td>
</tr>
<tr>
<td>21-25</td>
<td>Video</td>
<td>1000</td>
<td>512-1500</td>
<td>10</td>
</tr>
<tr>
<td>26</td>
<td>Best Effort</td>
<td>3360</td>
<td>64-1500</td>
<td>100</td>
</tr>
</tbody>
</table>

**Table 1: Flows’ description**

5.2 Results

Figures 4 and 5 illustrate the maximum latency experienced by voice, video and best effort flows. ERR-F, PERR-F and NDRR-F are fixed versions of the original scheduling algorithms described in section 2. In figure 4 the average link utilization was approximately 70%, and that was achieved by reducing the best effort traffic flooded to the link. In figure 5 the average link utilization was increased to about 90%. Both of the figures 4 and 5 suggest that MRR offers the least latency bound for voice and video flows.

In [19], Shi et al suggested the use of the Gini Index [20] as an instantaneous measure of fairness. The index evaluates the area between the service curve of a scheduler and the corresponding service curve of GPS. It is believed that this measure is more accurate than RFB because it gives an indication on the frequency at which a scheduler reaches its upper bound.

Figures 6 and 7 show the Gini fairness index measurements for the 70% and 90% link utilization consecutively. Figure 6 suggests that MRR is fairer than the other three schedulers. This fairness advantage is much more obvious in figure 7 where the link utilization is 90%. The Gini index figures for the modified versions of ERR, NDRR and PERR have not been shown because they are almost equal to those of their older versions.
6 CONCLUSION

In this paper, we have presented the Mini Round Robin (MRR) scheduler, which is fair, efficient and provides low-rate, high-priority flows with a lower latency bound. MRR also provides a lower start-up latency bound than any other frame-based scheduler. Analytical and simulation results demonstrated these facts, and illustrated the advantages of the MRR algorithm over other closely related frame-based schedulers.

REFERENCES