**MINI ROUND ROBIN:
AN ENHANCED FRAME-BASED SCHEDULING ALGORITHM FOR
MULTIMEDIA NETWORKS**

By

TARIQ JAMAL AL-KHASIB

B.Sc (Electrical Engineering), JUST, 2001

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

In

THE FACULTY OF GRADUATE STUDIES

Department of Electrical and Computer Engineering

We accept this thesis as conforming to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

June 2004

© Tariq Jamal Al-Khasib, 2004

Library Authorization

In presenting this thesis in partial fulfillment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

AL-KHASIB, TARIQ

Name of Author (please print)

15th June 2004

Date (dd/mm/yyyy)

Title of Thesis: Mini Round Robin: An Enhanced
Frame-Based Scheduling Algorithm for Multimedia
Networks.

Degree: Master of Applied Science Year: 2004

Department of Electrical and Computer Engineering
The University of British Columbia
Vancouver, BC Canada

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. Most scheduling methods are based on the concept of providing a delay bound based on the bandwidth reserved by the session. The main advantage of MRR is its ability to decouple the delay from session bandwidth. This enables applications such as Voice-over-IP to demand low delay despite the low reserved bit rate of the voice sessions. MRR is computationally very efficient with an $O(1)$ per packet work complexity, and it provides better fairness to multimedia sessions compared to other frame-based scheduling algorithms. We show that some of the most recently introduced schedulers fail to provide fairness or bounded latency under certain network conditions, and we propose a solution to overcome this problem. Finally, we present an extensive set of performance and validation test results obtained from simulations conducted using network models built for each one of the scheduling algorithms under consideration.

Contents

ABSTRACT.....	ii
Contents	iii
List of Tables	v
List of Figures	vi
List of Acronyms and Symbols	vii
ACKNOWLEDGEMENTS	ix
1. Introduction and Motivation	1
1.1 Introduction.....	1
1.2 QoS in IP Networks	2
1.3 Multiprotocol Label Switching (MPLS).....	5
1.4 Packet scheduling.....	5
1.5 Main Contributions	7
1.6 Thesis Organization	8
2. Previous Work	9
2.1 Introduction.....	9
2.2 Frame-Based Schedulers.....	9
2.3 Timestamp-Based Schedulers	15
2.4 Summary	19
3. Deficiencies in ERR, PERR, and NDRR Schedulers	20
3.1 Introduction.....	20
3.2 Deficiencies in ERR.....	21

3.3	Deficiencies in PERR	23
3.4	Deficiencies in NDRR	25
4.	Mini Round Robin (MRR)	27
4.1	Algorithm Description	27
4.2	Performance Analysis of MRR.....	32
4.2.1	Computational Complexity.....	32
4.2.2	Start-up Latency.....	33
4.2.3	Fairness	36
4.2.4	Latency Bound.....	39
5.	MRR Performance	48
5.1	MRR performance compared to other schedulers	48
5.2.1	Simulation Set-up.....	48
5.2.2	Simulation Results	50
5.2	MRR performance for multimedia traffic.....	58
5.2.1	Simulation Set-up.....	58
5.2.2	Simulation Results	58
6.	Conclusion and Future Work	62
	Bibliography	64
	Appendix A: $\mathcal{LR}$ servers.....	69
	Appendix B: The Gini Index.....	71

List of Tables

Table 1: Start-up latency comparison 35

Table 2: Flows’ weights and reserved rates 49

Table 3: Flows’ description for scenario 2 58

List of Figures

Figure 1: QoS in IP Networks.....	4
Figure 2: DRR sample Trace	11
Figure 3: ERR sample trace	13
Figure 4: ERR scheduler deficiency	22
Figure 5: PERR scheduler deficiency	24
Figure 6: MRR sample trace	30
Figure 7: MRR algorithm pseudo-code	32
Figure 8: Worst-case Scenario for relative fairness bound calculation.	39
Figure 9: Worst-case Scenario for latency bound calculation.	45
Figure 10: Analytical latency bound of MRR, ERR, and PERR	46
Figure 11: Analytical latency bound of MRR for different values of s	47
Figure 12: Simulation topology	49
Figure 13: Average Gini Index whan flows are not always backlogged	53
Figure 14: Latency Bound	54
Figure 15: Latency Bound when flows are always backlogged	57
Figure 16: Average Gini Index	57
Figure 17: latency bound for different schedulers	60
Figure 18: Average Gini Index	61
Figure 19: Busy periods of session i	70
Figure 20: An example of the behaviour of an $\mathcal{LR}$ scheduler.	71
Figure 21: The Lorenz curve and Gini Index.....	72

List of Acronyms and Symbols

CAC	Connection Admission Control
DC	Deficit Counter
DRR	Deficit Round Robin
ERR	Elastic Round Robin
GPS	Generalized Processor Sharing
IP	Internet Protocol
ISP	Internet Service Provider
LR	Latency Rate
LSP	Label Switched Path
MPLS	Multiprotocol Label Switching
MRR	Mini Round Robin
NDRR	Nested Deficit Round Robin
OSPF-TE	Open Shortest Path First with Traffic Engineering extensions
PDRR	Pre-order Deficit Round Robin
PERR	Prioritized Elastic Round Robin
QoS	Quality of Service
RF	Relative Fairness
RFB	Relative Fairness Bound
RR	Round Robin
RSVP	resource ReSerVation Protocol

SC	Surplus Counter
SCFQ	Self-Clocked Fair Queuing
SFF	Smallest Finish-time First
SFQ	Start-time Fair Queuing
SRR	Surplus Round Robin
SSF	Smallest Start-time First
TE	Traffic Engineering
UA	Unserved Allowance
VC	Virtual Clock
VoIP	Voice over IP
WF ² Q	Worst-case Fair Weighted Fair Queuing
WFQ	Weighted Round Robin
WRR	Weighted Round Robin

ACKNOWLEDGEMENTS

All praise and thanks are due to almighty God, the source of all goodness and guidance.

My sincere gratitude and appreciation to The University of British Columbia and The Department of Electrical and Computer Engineering for providing me with the friendly environment for research and knowledge.

I would like to express my sincere thanks to my supervisor Dr. Hussein Alnuweiri for his endless support, expert guidance and great patience. Without him this thesis could not be produced in its present form. I would also like to express my appreciation and gratitude to Dr. Victor Leung for supporting and funding this research.

I would like to acknowledge Telus Mobility, Advanced Systems Institute of BC, and Natural Science and Engineering Research Council of Canada (grant CRDPJ 257855-01) for their financial support of this research.

I owe my deep regards and appreciation to my parents, my brothers, and my sisters. Their continuous love, support, and prayers made this work possible.

I extend my thanks to the great friends I made through my stay in Vancouver. I especially would like to thank Raef, Moh. Bdair, Tamer, Ayman, Haitham, Hosam, Khaled, and Abu Elainain, for the wonderful times we spent together

Dedicated with love

To

My Parents JAMAL and SAMEERA

Chapter 1

Introduction and Motivation

1.1 Introduction

The high cost of network equipment assets and the commercial competitive nature of providing Internet services have been pressing large Internet service providers (ISPs) to look for ways to generate more revenue without substantially increasing their investments in upgrading network infrastructure. A key to achieving such goal is the ISPs' ability to provide "high-value" IP services to customers, while lowering their network costs by achieving maximal operational efficiency.

The current Internet is based on a best-effort service model that offers no service guarantees to flows and applications. This lack of Quality of Service (QoS) assurance has a serious impact on certain types of applications that have strict end-to-end QoS requirements. Multimedia streaming, Voice-over-IP (VoIP), and interactive

applications are all examples of applications that require special treatment while traversing network nodes.

1.2 QoS in IP Networks

QoS refers to the capability of a network to provide differentiated service levels to selected network traffic. The primary goal of QoS is to provide deterministic and predictable performance guarantees including dedicated bandwidth, bounded jitter and latency, and improved loss characteristics.

In a QoS-capable network model, means of resource reservation and admission control should be provided. Connection Admission Control (CAC) [1] entities throughout the network can have an up to date information about the resource availability in the network. This information can be distributed using one of the routing protocols with Traffic Engineering (TE) extensions, such as Open Shortest Path First with Traffic Engineering extensions (OSPF-TE) [2][3][4]. The CAC entities will direct users, upon their request, to paths through the network that most likely would be able to provide them with the QoS requirements they need. Users will then use a resource reservation protocol such as resource ReSerVation Protocol (RSVP) [5] to reserve a guaranteed QoS path from source to destination.

Figure 1 shows a simple QoS-enabled network configuration. When packets arrive to a network node, they are switched to the appropriate output line card through the router's switching core. Packets are switched based on either a routing decision by a separate routing entity as in IP networks, or a label carried by the packet itself as in

Multiprotocol Label Switched (MPLS) [6] networks as explained later. Upon their arrival to an output line card, packets are classified and inserted into queues that are associated with their assigned flows. A flow can be viewed as a sequence of packets having a set of common characteristics. These characteristics vary depending on the way flows are classified and where in the network they are being classified. In IP networks, classification is usually performed using one or a combination of the following: source IP address, destination IP address, protocol type, source port, and destination port. In this thesis we will use the terms *flow* and *session* interchangeably.

Network resources need to be protected against congestion and conflicts, caused by ill behaving traffic sources. Traffic shaping and policing are two important mechanisms to ensure that traffic conforms to QoS contract and does not overload the network. Shaping is usually performed at the hosts or access (edge) routers, and it modifies traffic timing so that it conforms to the session's QoS profile. Policing, on the other hand, is usually done in switches and routers along the network path, and it checks if the traffic conforms to the session's QoS profile. During policing, non-conforming packets are dropped, buffered, or marked to be dropped later if network congestion occurs.

One of the most essential components of a QoS-supporting network model is packet schedulers. By determining the sequence in which packets depart the node, schedulers can provide strict end-to-end delay as well as bandwidth and fairness guarantees to flows or sessions.

Figure 1: QoS in IP Networks

1.3 Multiprotocol Label Switching (MPLS)

MPLS [6][7][8] was originally developed to provide faster packet forwarding than traditional IP routing. However, the flexibility of MPLS has made it the main mechanism for modern networks to achieve scalable QoS.

In the framework of MPLS, resource reservations by users are used by a separate routing and label distribution entity to establish Label Switched Paths (LSPs). Individual flows or aggregates of multiple flows can be then mapped to one of these LSPs. Per-flow or per-aggregate scheduling can correlate different LSPs with different QoS profiles, and assist in the process of choosing the end-to-end paths during LSP set-up.

1.4 Packet scheduling

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 a network node.

Modern packet scheduling theory is based on few, but very powerful, concepts. One such concept is Generalized Processor Sharing (GPS) [9][10][11]. 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) [9][12], Worst-case Fair Weighted Fair Queuing (WF²Q) [13], Virtual Clock (VC) [14], and Self-Clocked Fair Queuing (SCFQ) [15]. 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 [16][17].

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) [18], Deficit Round Robin (DRR) [19], and Elastic Round Robin (ERR) [20][21], 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 thesis, and for the reasons above, we are interested in frame-based schedulers, for their well-known simplicity and efficiency.

Timestamp-based and Frame-based schedulers, suffer from the fundamental problem of coupling latency experienced by a flow to the flow’s reserved rate compared to the reserved rates of other flows. In other words, flows with lower reserved rate will get higher latency. Therefore, increasing a flow’s reserved rate is viewed as the primary and straightforward solution to providing lower latency bounds. We show that our proposed scheduler, Mini Round Robin (MRR), enhances the fairness, latency bounds, and startup latency of existing frame-based schedulers. Moreover, it maintains an $O(1)$ per-packet work complexity, and is able to decouple the latency bound from the flow’s reserved rate.

1.5 Main Contributions

In this thesis, we introduce the new frame-based Mini Round Robin scheduling technique. It provides multimedia flows with lower latency bounds, and lower start-up latency bound. MRR is computationally very efficient with an $O(1)$ per packet work complexity, and it provides better fairness compared to other frame-based scheduling algorithms. We provide a comprehensive worst-case scenario analysis,

and we derive analytical bounds for startup latency, fairness, and latency. Another contribution of this thesis is the uncovering of serious deficiencies present in several recently proposed schedulers. In particular [20][22][23]. We show that these schedulers fail to provide fairness or bounded latency under certain network conditions and we propose solutions to overcome these problems. Finally we present an extensive set of performance and validation test results obtained from simulations conducted using network models built for each one of the scheduling algorithms under consideration.

1.6 Thesis Organization

Chapter 2 presents a survey of the most common scheduling algorithms. In particular, frame-based schedulers upon which the new algorithm is based. In chapter 3, we discuss major deficiencies present in some recent scheduling algorithms, and provide solutions for these problems. In chapter 4 we introduce the Mini Round Robin scheduling algorithm, we explain the rational behind it, and provide analytical results on the work complexity, startup latency, fairness and latency bound. Chapter 5 is dedicated for results obtained from the OPNET simulation tool via network models written for each one of the algorithms under investigation. We conclude the thesis in chapter 6, and provide some guidelines for future work.

Chapter 2

Previous Work

2.1 Introduction

Packet schedulers can be broadly divided into two main categories: frame based schedulers and timestamp-based schedulers. In this chapter we review some of the well-known schedulers under each one of these categories.

2.2 Frame-Based Schedulers

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 [18]. 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. This unfairness problem could be overcome if the mean packet size for all the flows is known prior to scheduling packets. WRR is also stateless and has an $O(1)$ computational complexity per packet.

Deficit Round Robin (DRR) [19] 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 quantity called the Quantum (Q_i). The quantum for 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 . 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 [20].

Figure 2 demonstrates a sample trace of a server running a DRR scheduler. Three flows equally share the output link and they are always backlogged. The maximum packet size that could potentially arrive to the system is 16 units. Thus, the quantum assigned to the flows is equal to 16 units.

Figure 2: DRR sample Trace

Surplus Round Robin (SRR) [24] is a modified version of DRR, in which a flow is allowed to overdraw its balance and be penalized for that in the following round. SRR does not require knowledge of the packet size prior to scheduling it, but it does require knowledge of the maximum packet size that could potentially arrive to the system to ensure an $O(1)$ per-packet work complexity.

Elastic Round Robin (ERR) [20][21] introduced the concept of a variable quantum that depends on the performance of flows in the previous round. Like SRR, 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 using equation (2) below, and then used in equation (3) to calculate the new Allowance ($A_i(r)$) of each active flow to be served in the next round. The allowance represents the least amount of data a flow can send in a round as long as it has packets to transmit. Equation (3) guarantees that the allowance of each flow will become greater than zero at the beginning of each round, and thus each active flow will receive at least one service opportunity in each round leading to $O(1)$ computations per packet work complexity. Equation (1) suggests that $SC_i(r)$ can be represented as the difference between the actual amount of data sent in round r ($Sent_i(r)$) and the allowance of the flow at the beginning of that round ($A_i(r)$).

$$SC_i(r) = Sent_i(r) - A_i(r) \quad (1)$$

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

$$A_i(r) = w_i[1 + \text{MaxSC}(r-1)] - \text{SC}_i(r-1) \quad (3)$$

Figure 3 shows how the scheduler behaves when fed by packets from three different flows that are always backlogged and equally share a link. The allowance of each flow in each round is updated according to the behaviour of the flows in the previous round. Note that in each round each flow transmits at least its allowance at the beginning of that round.

Figure 3: ERR sample trace

Compared to SRR and 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.

As stated in [22], a major drawback of DRR is that its latency bound is highly dependant on the reserved rate of the flow and the reserved rates of other flows. In other words, as the difference between the reserved rates of two flows increases, the difference in their latency bounds increases. Thus, flows with lower reserved rates may have a very high latency bounds compared to flows with higher reserved rates. The Nested Deficit Round Robin (NDRR) approach [22] 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) [25] and the Prioritized Elastic Round Robin (PERR) [23][26]. 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 [23][26] 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 prospective we see that it is unfair to compare these two algorithms to other $O(1)$ algorithms like DRR and ERR.

2.3 Timestamp-Based Schedulers

The GPS scheduler [9][10] is unrealizable in its fluid formulation since no packet can be partitioned into infinitesimal quantities for transmission. As a result, researchers decided to use the service order of packets in GPS to schedule packets in a packet based system. Two main GPS emulation policies are present; Smallest Finish-time First (SFF) and Smallest Start-time First (SSF). In the SFF techniques,

packets are serviced in the order in which they finish under GPS. An SSF technique, on the other hand, services packets according to their starting order under GPS.

Packet-by-packet GPS, or WFQ as it is known, is one of the GPS emulations that transmit packets according to their finish order under GPS. In WFQ, a GPS fluid model system is simulated in parallel with the actual packet-based system in order to identify the set of sessions backlogged at each instant of time and their service rates. Based on this information, a timestamp is calculated for each arriving packet, and the packets are inserted into a priority queue based on their timestamp values and transmitted in order of increasing timestamps. To calculate the finish number, WFQ keeps track of a virtual time function $V(t)$ which is a piecewise linear function of real time t , and whose slope changes depending on the number of busy sessions in GPS and their service rates. More precisely, if $B(t)$ represents the set of backlogged sessions in the GPS scheduler at time t , then the slope of the virtual time function at time t is given by:

$$\frac{\sum_{j=1}^N w_j}{\sum_{i \in B(t)} w_i}$$

The virtual time function keeps track of the "normalized" service provided by the system to all backlogged sessions. At the arrival of a new packet, the virtual time must first be calculated. Then, the timestamp TS_i^k associated with the k th packet of session i that arrives at time t is calculated as

$$TS_i^k = \max(TS_i^{k-1}, V(t)) + \frac{L_i^k}{r_i}$$

Where L_i^k is the length of the arrived packet and r_i is the guaranteed link share of session i .

One approximation of WFQ is a scheduling method called Self-Clocked Fair Queuing (SCFQ) [15]. In SCFQ, an approximation of the virtual time function $V(t)$ is calculated using the timestamp of the packet currently in service. Thus, if TS_{cur} denotes the timestamp of the packet currently in service, the virtual time $V(t)$ is taken as TS_{cur} . SCFQ calculates the timestamp of an arriving packet; say the k th packet of session i , as follows

$$TS_i^k = \max(TS_i^{k-1}, TS_{cur}) + \frac{L_i^k}{r_i}$$

This approach reduces the complexity of the algorithm greatly. However, the compromise is a reduced level of isolation among the sessions, causing the end-to-end delay bounds to grow linearly with the number of sessions that share the outgoing link.

The Virtual Clock (VC) scheduling algorithm [14] provides the same end-to-end delay bound as WFQ with a simple timestamp computation algorithm. The virtual time function in this algorithm is the real time and hence the timestamp becomes [14],

$$TS_i^k = \max(TS_i^{k-1}, t) + \frac{L_i^k}{r_i}$$

The disadvantage of this algorithm is that a backlogged session can be starved for an arbitrary period of time as a result of excess bandwidth it received from the server when other sessions were idle.

Another variant of WFQ is the Start-time Fair Queuing (SFQ) [27]. This technique tries to schedule packets according to their start time in GPS. The virtual time is approximated by the virtual start time of the packet currently in service. Packets are scheduled in order of start-time with ties broken by the toss of a coin. The virtual finish time of a packet is calculated as the sum of the virtual start time plus the ratio of length to session share. When a packet arrives to a backlogged session, its virtual start time is set equal to the virtual finish time of the previous packet of that session. Although SFQ is easier to implement than WFQ, it has a delay bound well above that of WFQ.

Finally, a more accurate version of WFQ is Worst-case Fair Weighted Fair Queuing (WF^2Q) scheduler [13]. In WF^2Q , packets are scheduled according to their finish time in GPS, but only if their virtual start time is less than or equal to the current virtual time. WF^2Q has better fairness properties than WFQ, and it could be shown that WF^2Q is the fairest packet-by-packet scheduling algorithm [13]. WF^2Q has the same high complexity as WFQ.

2.4 Summary

This chapter presented a brief review of the two major categories of packet schedulers: frame-based schedulers and timestamp-based schedulers. As mentioned in chapter 1, this thesis is mainly concerned with frame-based schedulers because of their efficiency and simplicity. The next chapter reveals deficiencies present in some recently proposed frame-based scheduling algorithm discussed above.

Chapter 3

Deficiencies in ERR, PERR, and NDRR Schedulers

3.1 Introduction

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.

3.2 Deficiencies in ERR

Referring to the ERR description given in section 2.2, the first problem surfaces when a flow becomes idle in a round in which it consumed more than its allowance, and then becomes active in the same round in which it became idle. The flow's SC_i counter will be reset, and the flow will get an allowance in the next round higher than that it would have obtained if the flow was continuously backlogged. Figures 4 illustrates this problem in ERR. Flows 1 and 2 have the same weight of 1, and have their SC counters set to 0 at the beginning of round k . Assume that $MaxSC(k-1)$ is equal to $(m-1)$ due to a third flow that is no longer active at round k .

In figure 4, after transmitting two packets of a total size of $2m-1$ bits in round k , flow 1 becomes idle, and flow 2 starts transmitting its packet. While the packet from flow 2 is in service, flow 1 becomes active again in round k and its SC is reset. When round $k+1$ starts, flow 1 will get an allowance of m , which is m bits higher than what it would have been allowed if it was continuously backlogged. The same scenario is repeated in round $k+1$, and can take place in future rounds. In the long run, this scenario results in flow 1 getting twice the service rate that flow 2 gets, even though

both of the flows have the same weight. This unfairness will cause an unbounded latency for flow 2.

Figure 4: ERR scheduler deficiency

This problem can be solved by keeping record of SC_i of idle flows for at least one round after the round in which the flow became idle. This way, the flow's SC_i will not be reset if the flow became idle in the current or previous round. This could be accomplished via the use of a counter that keeps track of the current running round, and a separate counter associated with each flow to record the last round in which the flow was served.

The second problem happens when a flow becomes idle in a round before consuming its entire allowance, and then switches back to active during the same round in which it became idle. In ERR, the flow will simply lose its opportunity to transmit the remaining portion of its allowance, and will not affect any of the other

flows. This could lead to a higher latency bound, which could be double the latency bound a continuously backlogged flow would face.

This problem cannot be solved for ERR without replacing the data structures used in the algorithm, a change that would entirely alter the way the algorithm operates. At the same time, the flow should not be given any advantage in the upcoming rounds to prevent allowance carryover.

3.3 Deficiencies in PERR

In PERR scheduling, the SC_i counter will be reset when a flow (i) becomes idle in a round in which it has consumed more than its allowance, followed immediately by a round in which flow i becomes active. This will lead to the flow getting an allowance in the next round higher than that it would have obtained had the flow been continuously backlogged. Figure 5 illustrates this problem in PERR. In this figure flows 1 and 2 have the same weight of 1, with SC set to 0 for both flows at the beginning of round k . Assume that $MaxSC(k-1)$ is equal to $(m-1)$ due to a third flow that is no longer active at round k .

Figure 5 demonstrates the same idea as figure 4 does, but for the PERR scheduler. The main difference is that flow 1 becomes active in the round immediately following the one in which it became idle. Flow 1 gets more than its reserved rate, leading to an unacceptable latency for flow 2.

Figure 5: PERR scheduler deficiency

As in ERR this problem can be solved by keeping record of the SC_i and $Sent_i$ quantities for idle flows for at least one round after the round in which the flow became idle. This way, the flow's SC_i and $Sent_i$ will not be reset if the flow became idle in the current or previous round.

The second problem in PERR, based on the formulation in [23], is similar to the second problem present of ERR, but it is more severe in consequences. Again when a flow becomes idle in a round before consuming its entire allowance, and then switches back to active during the same round in which it became idle, the new allowance the flow will get will be greater than its allowance before becoming idle. This causes that flow to receive an unfair advantage from the scheduler, and increases the latency experienced by other flows. This problem is caused by the loss of the last-round's SC_i , which was reset at the beginning of the current round, leading to a higher allowance when recalculated according to formula (4). Where the Maximum Unserved Allowance ($UA_i^{\max}(r)$), as defined in (5), is equal to the maximum possible Allowance for flow i at the beginning of round r .

$$A_i(r) = UA_i^{\max}(r) - SC_i(r-1) \quad (4)$$

$$UA_i^{\max}(r) = w_i(1 + MaxSC(r-1)) \quad (5)$$

Contrary to the ERR flaw, the PERR problem can be solved by initializing a newly active flow only if the flow was not served in the current round. Flows that were active during the current round will simply be switched from idle to active.

3.4 Deficiencies in NDRR

In NDRR, and according to the algorithm description presented in [22], if a flow becomes idle after consuming most of its quantum in a round, and becomes active again in the same round, then its deficit counter will be reset and its unserved quantum will be set back to its original value. This means that the flow will get its quantum twice in a round, which is unfair and may increase the latency of other flows. The worst-case scenario may happen if two flows switch between idle and active such that at least one of them is active at any time. This situation will cause starvation for other flows and these two flows will get the entire server bandwidth.

The solution for this problem is similar to the solutions we proposed for ERR and PERR. Essentially, newly active flows that were served in the current round should be appended to the tail of the NextActiveList instead of the ActiveFlowsList, unless the flow did not consume its entire Quantum. The flow's *DC* should not be reset unless a new round is started, or the flow was not served during the current or previous round.

Later in this thesis, we will refer to the modified versions of ERR, PERR and NDRR as ERR-FIXED, PERR-FIXED, and NDRR-FIXED consecutively. The performance of these algorithms will be compared to their modified versions, and the modified versions will be compared to our newly proposed MRR algorithm.

Chapter 4

Mini Round Robin (MRR)

4.1 Algorithm Description

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 (8), and a new series of mini-rounds is started. Equations (6), (7) and (8) are similar to equations (1), (2) and (3) consecutively. $DC_i(r)$ is the deficit counter and it corresponds to the deficit in service for flow i is round r . In the algorithm description, as we will see later, the deficit counter is implicitly contained in the balance itself. Having a positive balance indicates a zero deficit, while having a negative balance implies a positive deficit equal to the absolute value of the balance.

$$DC_i(r) = Sent_i(r) - Balance_i(r) \quad (6)$$

$$MaxD(r) = \underset{\forall i \text{ served in round } r}{Max} \left\{ \frac{DC_i(r)}{w_i} \right\} \quad (7)$$

$$Balance_i(r) = w_i(1 + MaxD(r-1)) - DC_i(r-1) \quad (8)$$

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.

Flow 1
 Flow 2
 Flow 3

Round Number	Mini-Round Number		Balance BEFORE	Balance AFTER
1	1	16	1	-15
		8	1	-7
		4	1	-3
2	1	10	1	-9
		6	9	3
		4	13	9
	2	10	3	-7
		8	9	1
	3	12	1	-11
3	1	4	3	-1
		4	5	1
		8	1	-7
	2	8	1	-7

Figure 6: MRR sample trace

Figure 6 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 (8). The three flows get a unity balance in the first round, and transmit a single packet that will cause their 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).

1	Initialize:
2	CurrentRound = 0;
3	OldHighestDeficit = 0;
4	HighestDeficit = 0;
5	For(i=0 ; i<n; i++)
6	{
7	LastActiveRound(i) = -1;
8	Balance(i) = 0;
9	FirstMiniRound(i) = TRUE;

```

10     }
11
12   Enqueue: (invoked when a packet arrives)
13     i = FlowToWhichPacketArrives;
14     if(QueueIsEmpty(i) == TRUE) Then
15         if(LastActiveRound(i) != CurrentRound) then
16             AddToTailOfMiniRoundList(i);
17             FirstMiniRound(i) = TRUE;
18         Else
19             If(Balance(i) > 0) then
20                 AddToTailOfMiniRoundList(i);
21             Else
22                 AddToTailOfActiveList(i);
23                 FirstMiniRound(i) = TRUE;
24             Endif
25         Endif
26         If ((LastActiveRound(i) < (CurrentRound - 1)) ||
27            ((LastActiveRound(i) == CurrentRound - 1) && (Balance(i) > 0))) Then
28             Balance(i) = 0;
29         Endif
30         LastActiveRound(i) = CurrentRound;
31     Endif
32     AddPacketToQueueu(i);
33
34   Dequeue:
35     While (TRUE) do
36         If ( MiniRoundListIsEmpty() == FALSE) then
37             i = HeadOfMiniRoundList();
38             RemoveHeadOfMiniRoundList();
39             If ( FirstMiniRound(i) == TRUE) then
40                 Balance (i) = Weight (i) * ( 1 + OldHighestDeficit)) + Balance(i);
41                 FirstMiniRound(i) = FALSE;
42             End if
43
44             TransmitPacketFromQueue(i);
45             Balance(i) = Balance(i) - PacketSize;
46
47             If ( Balance(i) > 0 && QueueIsEmpty(i) == FALSE) then
48                 AddToTailOfMiniRoundList(i);
49             Elseif (Balance(i) <= 0 && QueueIsEmpty(i) == FALSE) then
50                 AddToTailOfActiveList(i);
51                 FirstMiniRound(i) = TRUE;
52                 If (-1 * Balance(i) / weight(i) > HighestDeficit) then
53                     HighestDeficit = -1 * Balance(i) / weight(i);
54                 Endif
55             Endif
56             LastActiveRound(i) = CurrentRound;
57         Else
58             MoveActiveListToMiniRoundList();
59             CurrentRound++;
60             OldHighestDeficit = HighestDeficit;
61             HighestDeficit = 0;
62             If ( MiniRoundListIsEmpty() == TRUE) then

```

```

63                                     Initialize();
64                                     WaitForArrivals();
65                                 Endif
66                             Endif
67                         End while;

```

Figure 7: MRR algorithm pseudo-code

Figure 7 shows the pseudo-code implementation of the algorithm, which consists of an initialization procedure performed at the beginning of each busy period of the server, a packet enqueue procedure that is called upon the arrival of a new packet, and a packet dequeue procedure to determine the next packet to leave the server.

4.2 Performance Analysis of MRR

In this section we apply theoretical analysis to derive the MRR work complexity, start-up latency bound, fairness, and latency bounds.

4.2.1 Computational Complexity

Theorem 1: The per-packet computational complexity of an MRR scheduler is $O(1)$.

Proof: To prove that the work complexity of the MRR scheduler is $O(1)$, we have to show that each of the enqueue and the dequeue processes has an $O(1)$ work complexity.

It is obvious from the pseudo-code in figure 7 that the time complexity of the enqueue process is $O(1)$. The enqueue process involves determining the flow to which the packet has arrived and that is an $O(1)$ operation. Determining the flow to which the packet belongs, adding the flow to the tail of one of the linked lists, and adding the actual packet to the packet queue, are all operations of $O(1)$ complexity.

The dequeue process also has an $O(1)$ work complexity. It involves adding/removing flow numbers to/from the head or the tail of linked lists. It also involves moving the content of the ActiveList to the MiniRoundList, which can be done by switching the pointers of the two lists. All of these operations have an $O(1)$ work complexity.

□

4.2.2 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.

Definition 1: m is the size in bits of the largest packet actually served during the execution of a scheduling algorithm.

Definition 2: M is the size in bits of the largest packet that may potentially arrive during the execution of a scheduling algorithm.

Definition 3: s is the size in bits of the smallest packet that may arrive during the execution of a scheduling algorithm.

Definition 4: (u, v) is a reference to the mini-round v of round u

Definition 5: $Served_i(u, v)$ is the service received by flow i in mini-round (u, v) .

Lemma 1: for any flow i and any mini-round (u, v) , we have:

$$0 \leq Served_i(u, v) \leq m$$

Proof: from the algorithm description, every flow is allowed to send at most one packet during any mini-round, as long as

- 1- The flow's Balance is positive, and
- 2- The flow is active and has packets in its queue.

If one of these conditions is not met then flow i will not send any data in mini-round (u, v) . Hence, $Served_i(u, v) \geq 0$. Otherwise flow i will be given the chance to transmit at most one packet, and from definition (1), the maximum packet size is m . Thus, $0 \leq Served_i(u, v) \leq m$.

□

Definition 6: W is the sum of the weights of all flows served by a scheduling algorithm such that:

$$W = \sum_{j=1}^n w_j$$

Theorem 2: if flow i becomes active, then the upper bound on the start-up latency, S_{MRR} , experienced by the first packet of flow i is

$$S_{MRR} \leq \frac{(n-1)m}{r}$$

Where r is the service rate of the server.

Proof: If flow i becomes active at the beginning of or during round r , the flow will be added to the tail of the MiniRoundList. In both cases there could be at most $(n-1)$ flows ahead of flow i in the MiniRoundList, each of which will get the chance to send

at most one packet before flow i gets its transmission opportunity. From lemma 1, the maximum amount of service a flow can receive in a mini-round is m . Thus, the maximum latency flow i will experience is the time required to transmit $(n-1)m$ worth of data.

□

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. Table 1 compares the start-up latency bound of different schedulers and MRR. For example, the ERR start-up latency is worse than the MRR start-up latency by a factor approximately equal to

$$\frac{(W - w_i)m}{r}.$$

Scheduler	Start-up Latency
MRR	$\frac{(n-1)m}{r}$
ERR	$\frac{(W - w_i)m + (n-1)(m-1)}{r}$
PERR	$\frac{(W - w_i)\frac{m}{p} + (n-1)(m-1)}{r}$
NDRR	$\frac{(n-1)M}{r}$

Table 1: Start-up latency comparison

4.2.3 Fairness

Let $Sent_i(t_1, t_2)$ 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:

$$MAX \left(\left| \frac{Sent_i(t_1, t_2)}{w_i} - \frac{Sent_j(t_1, t_2)}{w_j} \right| \right) \quad \forall i \text{ and } j \in \text{Active flows during } (t_1, t_2)$$

Now, if we define the Relative Fairness Bound (RFB) as the maximum RF over all intervals of time, then we have the following theorem.

Theorem 3: for any execution of the MRR scheduling discipline,

$$RFB \leq 4m - 2$$

Proof: To prove this theorem, we will consider the worst-case scenario shown in figure 8. Such a worst-case scenario is characterized as follows:

- 1) The weight of flow i (w_i) is greater than the weight of flow j (w_j)
- 2) At the beginning of round 1, flow j has its lowest allowance, while flow i has its highest allowance.
- 3) During round 1, flow i transmits packets of size s (each) until flow j completes transmitting its entire allowance using m -bit sized packets. In this round both flows exceed their allowance by $m-1$ bits.
- 4) In rounds 2 to k , flow i transmits the maximum possible amount of data, while flow j transmits the minimum possible amount of data. During these rounds, the sequence of transmission changes in a way that results in flow i transmitting the first packet in round $k+1$.

- 5) In round $k+1$ flow i transmits m -bit sized packets until it finishes transmitting the maximum amount of data possible. At the same time, flow j transmits s -bit sized packets to cause a maximum difference in service.

It is obvious from figure 8 that the maximum difference in service between flows i and j occurs between times t_1 and t_2 , where t_1 marks the time at which flow j finished service in round 1 and t_2 marks the time at which flow i finished service in round $k+1$.

The period (t_1, t_2) can be divided into the following three different periods:

- 1) The period $(t_1, \text{End of round 1})$:

$$\begin{aligned} \text{Sent}_i(t_1, \text{End of round 1}) &= w_i m + m - 1 - \left(\frac{w_j m}{m} - 1\right)s \\ &= w_i m + m - 1 - (w_j - 1)s \end{aligned}$$

$$\text{Sent}_j(t_1, \text{End of round 1}) = 0.$$

- 2) The period $(\text{Start of round 2}, \text{End of round } k)$:

$$\text{Sent}_i(\text{Start of round 2}, \text{End of round } k) = \sum_2^k w_i m = (k-1)w_i m$$

$$\begin{aligned} \text{Sent}_j(\text{Start of round 2}, \text{End of round } k) &= w_j m - m + 1 + \sum_3^k w_j m \\ &= (k-1)w_j m - m + 1 \end{aligned}$$

3) The period (*Start of round $k+1, t_2$*):

$$Sent_i(\text{Start of round } k+1, t_2) = w_i m$$

$$\begin{aligned} Sent_j(\text{Start of round } k+1, t_2) &= MIN \left[\left(\frac{w_i m}{m} - 1 \right) s, w_j m \right] \\ &= MIN \left[(w_i - 1) s, w_j m \right] \end{aligned}$$

$$\begin{aligned} Sent_i(t_1, t_2) &= Sent_i(t_1, \text{End of round 1}) + Sent_i(\text{Start of round 2, End of round } k) + \\ &\quad Sent_i(\text{Start of round } k+1, t_2) \\ &= (k+1)w_i m + m - 1 - (w_j - 1)s \end{aligned}$$

$$\begin{aligned} Sent_j(t_1, t_2) &= Sent_j(t_1, \text{End of round 1}) + Sent_j(\text{Start of round 2, End of round } k) + \\ &\quad Sent_j(\text{Start of round } k+1, t_2) \\ &= (k-1)w_j m - m + 1 + MIN \left[(w_i - 1) s, w_j m \right] \end{aligned}$$

$$\begin{aligned} RFB &= \frac{Sent_i(t_1, t_2)}{w_i} - \frac{Sent_j(t_1, t_2)}{w_j} \\ &= (k+1)m + \frac{m-1}{w_i} - \left(\frac{w_j - 1}{w_i} \right) s - (k-1)m + \frac{m-1}{w_j} - MIN \left[\left(\frac{w_i - 1}{w_j} \right) s, m \right] \\ &= 2m + \frac{m-1}{w_i} + \frac{m-1}{w_j} - \left(\frac{w_j - 1}{w_i} \right) s - MIN \left[\left(\frac{w_i - 1}{w_j} \right) s, m \right] \end{aligned}$$

RFB is maximum if $w_i = w_j = 1$, therefore $RFB \leq 4m - 2$

□

The RFB for MRR is higher than it is for ERR and PERR which is $3m$ and $2m + \frac{2m}{p}$ consecutively. But, a closer look at the proof of theorem 3 above reveals

that MRR has a special characteristic. The relative fairness bound of MRR depends

on the weights of the two flows under consideration, and the RFB of $4m-2$ is reached only if the two flows have a weight of 1. On the other hand, the relative fairness of both ERR and PERR is independent of the weights of the flows under consideration. Therefore, the relative fairness of MRR is generally less than that of ERR and PERR if the weights of the flows were other than unity weights.

Figure 8: Worst-case Scenario for relative fairness bound calculation.

4.2.4 Latency Bound

In $\mathcal{LR}$ servers [28], the latency of a flow is defined as the minimum non-negative constant Θ_i that satisfies the following:

$$Sent_i(\tau_i, t) \geq \max\{0, \rho_i(t - \alpha_i - \Theta_i)\} \quad \text{for all busy periods of flow } i$$

Appendix A provides a more detailed description of the $\mathcal{LR}$ servers.

To find a latency bound for a flow i served by the MRR scheduler, we define the following worst-case scenario:

Definition 7: Define the worst-case scenario for a flow i as the scenario in which:

- 1) All flows have a zero balance at the beginning of round 1, and the maximum deficit of the previous round is $(m-1)$ due to a flow that is no longer active.
- 2) All flows other than flow i transmit the maximum amount of data possible in all rounds, and flow i just transmits its allowance.
- 3) Flow i finishes transmission in all rounds the latest it could, and that is accomplished if:
 - a. Flow i transmits packets of size s .
 - b. All other flows transmit packets of size m .

Figure 9 illustrates the worst-case scenario according to definition 7. The detailed operation of the algorithm is not shown for the rounds 1 to k for simplicity. All discussions below are based on this worst-case scenario.

Lemma 2: According to the worst-case scenario defined in 7, the sequence in which flows finish service in round 1 will be the same sequence in all consecutive rounds.

Proof: to prove this lemma, we will consider the sequence in which flows finish service in round 1 and other rounds:

Round 1:

In round 1 a flow j could finish before flow i if it was successful in transmitting $w_j m + m - 1$ bits using packets of size m before flow i transmits its $w_i m$ bits using packets of size s . This can be expressed as:

$$\left\lceil \frac{w_j m + m - 1}{m} \right\rceil \leq \left\lceil \frac{w_i m}{s} \right\rceil$$

$$\left\lceil w_j + 1 - \frac{1}{m} \right\rceil \leq \left\lceil w_i \frac{m}{s} \right\rceil$$

$$w_j + 1 \leq \left\lceil w_i \frac{m}{s} \right\rceil$$

$$w_j < \left\lceil w_i \frac{m}{s} \right\rceil$$

Rounds 2 to $k+1$:

In rounds 2 to $k+1$, a flow j can finish before flow i if it was successful in transmitting $w_j m$ bits using packets of size m before flow i transmits its $w_i m$ bits using packets of size s . There are two cases:

Case 1: Flow j finished before flow i in the previous round.

$$\left\lceil \frac{w_j m}{m} \right\rceil \leq \left\lceil \frac{w_i m}{s} \right\rceil$$

$$w_j \leq \left\lceil w_i \frac{m}{s} \right\rceil$$

But because these flows finished before flow i in the previous round they have to fulfill the condition ($w_j < \left\lceil w_i \frac{m}{s} \right\rceil$). By combining these two conditions, flow j has to fulfill the following condition:

$$w_j < \left\lceil w_i \frac{m}{s} \right\rceil$$

Case 2: Flow j finished after flow i in the previous round.

$$\left\lceil \frac{w_j m}{m} \right\rceil < \left\lceil \frac{w_i m}{s} \right\rceil$$

$$w_j < \left\lceil w_i \frac{m}{s} \right\rceil$$

We conclude that, in any round, for a flow j to finish service before flow i , flow j has to satisfy the following condition:

$$w_j < \left\lceil w_i \frac{m}{s} \right\rceil$$

□

In order to find the latency bound of the MRR scheduler we need to define the following two sets.

Definition 8: Set Y as the set of flows that finish service before flow i in round 1, and define y as the number of flows in the set Y .

Definition 9: Set F as the set of flows that must finish transmitting packets in mini-round k so that flow i starts to get a service equal to or greater than its reserved rate in mini-round $k+1$ while transmitting packets of size s bits.

Lemma 3: the set F is a subset of the set Y .

Proof: If y flows finish service in a round, then flow i has to be the next to finish according to lemma 2. Flow i would not be able to finish before the rest of the flows unless it gets more than its reserved rate.

Definition 10: Let z be the size of the set F , and define x as the size of the set $\overline{F} - \{i\}$. In other words, x is the maximum number of flows still in service while flow i is getting its share of the bandwidth:

$$x = (n - 1) - z$$

Lemma 4: the number of flows x that are still in service while flow i is getting service equal or greater than its reserved rate is the maximum number of flows g that satisfy the following:

$$\frac{s}{gm + s} \geq \frac{w_i}{W}$$

Proof: The relative reserved rate flow i has is $\frac{w_i}{W}$, and the actual relative service flow

i is getting in a mini-round is $\frac{s}{gm + s}$, where g is the number of flows still in service

other than i . From definition 10, x is the largest g that satisfies:

$$\frac{s}{gm + s} \geq \frac{w_i}{W}$$

□

Theorem 4: for any execution of the MRR scheduling discipline, the upper bound on the latency of Θ_i^{MRR} experienced by flow i is:

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

Proof: From figure 9, the latency bound is equal to the latency experienced by the packet of size s in mini-round $(k+1, u)$. We will refer to this packet as the *boundary* packet. The latency can be divided into several components:

Component 1: latency caused by rounds 1 to k .

$$\Theta_1 = \frac{(n-1)(m-1) + \sum_1^k (W - w_i)m + \sum_1^k w_i m - \frac{W}{w_i} \sum_1^k w_i m}{r}$$

$$\Theta_1 = \frac{(n-1)(m-1)}{r}$$

Component 2: latency from the beginning of round $k+1$ caused by flows in the set F , that finished before the boundary packet.

$$\Theta_2 = \frac{\sum_{j \in F} w_j m}{r}$$

Component 3: latency from the beginning of round $k+1$ till the beginning of mini-round $(k+1, u)$, caused by flows not in F including flow i :

$$\Theta_3 = \frac{[\text{Number of mini - rounds before } (k+1, u)] \times \left[(xm + s) - \frac{W}{w_i} s \right]}{r}$$

$$\Theta_3 = \frac{\left[\max_{j \in F} (w_j) - 1 \right] \times \left[xm - \left(\frac{W}{w_i} - 1 \right) s \right]}{r}$$

Component 4: latency from the beginning of mini-round $(k+1, u)$.

$$\Theta_4 = \frac{(y - z)m}{r}$$

Note that one of the packets in this region is already covered in component 2.

The total latency is then equal to

$$\Theta_i^{MRR} \leq \Theta_1 + \Theta_2 + \Theta_3 + \Theta_4$$

$$\Theta_i^{MRR} \leq \frac{(n-1)(m-1) + \sum_{j \in F} w_j m + \left[\max_{j \in F} (w_j) - 1 \right] \cdot \left[xm - \left(\frac{W}{w_i} - 1 \right) s \right] + (y - z)m}{r}$$

□

Figure 9: Worst-case Scenario for latency bound calculation.

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 10 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.

Figure 10: Analytical latency bound of MRR, ERR, and PERR

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 11 shows how changing the size (s) of the smallest packet can affect the latency bound.

Figure 11: Analytical latency bound of MRR for different values of s , $m=576$.

Chapter 5

MRR Performance

In this chapter we present performance and validation tests obtained from simulations conducted using OPNET [29] network models built for each one of the scheduling algorithms under consideration.

5.1 MRR performance compared to other schedulers

5.2.1 Simulation Set-up

Figure 12 shows the topology used to collect performance measurements under different service disciplines. The server output link is shared by 10 flows having different weights and reserved rates. Table 2 shows the flows' reserved rates and their corresponding weights.

All the sources generate packets with a uniformly distributed size between 64 and 1500 bytes, with an exponential mean inter-arrival times set according to the reserved

rates of the flows. Simulations lasted 40 minutes each, and were repeated several times to reach a confidence interval above 95 percent.

Figure 12: Simulation topology

Flow#	Flow Rate (Bytes / sec)	Flow Weight
1	10,000	1
2	20,000	2
3	30,000	3
4	50,000	5
5	70,000	7
6	100,000	10
7	200,000	20
8	500,000	50
9	700,000	70
10	1,000,000	100

Table 2: Flows' weights and reserved rates

5.2.2 Simulation Results

The tests were conducted in two phases. In the first set of experiments, we allow flows to switch back and forth between active and idle states. The shared link capacity is set to be 1-5 % higher than the sum of the average arrival rates of all inputs. This way we ensure that the link is highly utilized but not over subscribed. In the second set of experiments we ensure that flows are always backlogged, this is accomplished via over reserving the shared link and installing an adaptive source that will generate packets for flows suspected to become idle in the future.

5.1.2.1 Flows not always backlogged

In general, flow queues are not always backlogged. Thus, while designing a scheduling algorithm this fact should be taken into consideration. As we have seen earlier, ERR, PERR, and NDRR, all originally failed to provide fair treatment to flows continuously switching back and forth between active and idle states.

In [30], Shi et al suggested the use of the *Gini Index* [31] 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. Appendix B provides detailed information about this fairness measure.

Figure 13 shows the Average Gini Index for the ERR, NDRR, and PERR schedulers before and after being modified compared to the MRR scheduler. Figures 13a, 13b and a3c clearly demonstrate that the original ERR, NDRR and PERR

algorithms are less fair than their corrected versions. The lack of fairness can be clearly observed in the PERR and NDRR algorithms, due to the fact that both algorithms allow newly backlogged flows to be directly inserted into the currently running round, while in ERR, newly active flows have to wait until the start of the next round.

Figure 13a: Average Gini Index of ERR, ERR-FIXED and MRR

Figure 13b: Average Gini Index of NDRR, NDRR-FIXED and MRR

Figure 13c: Average Gini Index of PERR, PERR-FIXED and MRR

Figure 13c: Average Gini Index of ERR-FIXED, NDRR-FIXED, PERR-FIXED and MRR

Figure 13: Average Gini Index when flows are not always backlogged

According to figures 13a and 13b, MRR offers better fairness than the ERR, NDRR and their modified versions. The MRR scheduler also offers better fairness over PERR, but it is less fair compared to the modified version of PERR. We should keep in mind that the PERR algorithm has a higher per packet work complexity than MRR, and the lower fairness of MRR could be affordable if we seek a lower latency bound for flows with low reserved rates and a lower per packet work complexity.

From the results discussed above, and from the explanation presented in chapter 3, we will use the modified versions of ERR, PERR, and NDRR in further comparisons with the newly proposed MRR algorithm.

Figure 14 shows the maximum latency experienced by flows when served using different scheduling techniques. It can be clearly seen that MRR provides the minimum latency bound for flows with low reserved rates, while at the same time provides a moderate latency bound for flows with high reserved rates.

This graph proves the point for which MRR was originally designed, which is to provide low-rate but high-priority flows with a lower latency bound. MRR reduces the high dependency between a flow's reserved rate and its latency bound. The other frame-based schedulers, namely ERR, PERR, and NDRR, exhibits the usual coupling between reserved bandwidth and latency.

Figure 14: Latency Bound

5.1.2.2 Flows with persistent backlogged queues

When the flows become always backlogged, the ERR, PERR, and the NDRR algorithms behaves in the same manner and produce the same results as their corresponding modified versions. In this set of experiments, the same sources used in the previous part were used, but with a lower mean inter-arrival time to guarantee that flows are continuously backlogged.

Figure 15 suggest that the latency provided by the MRR scheduler for flows with a low reserved rate is still better or comparable to it in the ERR, PERR, and NDRR. MRR offers lower latency bound for all flows when compared to the ERR scheduler in figure 15a. In figure 15b, MRR shows a substantial improvement over NDRR for the first 6, low rate flows and shares comparable latency bounds for the remaining 4 high rate flows. Compared to PERR, MRR still performs better when low rate flows are concerned.

PERR keeps its good fairness, while the other algorithms share a comparable Gini Index figures as shown in figure 16. As we said earlier, it is unrealistic to have flows always backlogged. The exceptional performance of the PERR algorithm under these conditions is due to its higher computational complexity.

Figure 15a: Latency Bound of ERR, and MRR

Figure 15b: Latency Bound of NDRR, and MRR

Figure 15c: Latency Bound of PERR, and MRR

Figure 15: Latency Bound when flows are always backlogged

Figure 16: Average Gini Index

5.2 MRR performance for multimedia traffic

5.2.1 Simulation Set-up

Table 3 provides a description of multimedia traffic sources that will be used for evaluating 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.

Flow number	Flow Description	Bit rate (kbps)	Packet size (Bytes)	Burst size (Packets)
1-20	Voice	32	64-128	1
21-25	Video	1000	512-1500	10
26	Best Effort	3360	64-1500	100

Table 3: Flows' description for scenario 2

5.2.2 Simulation Results

Figure 17 illustrates the maximum latency experienced by voice, video and best effort flows. In figure 17a the average link utilization was approximately 70%, and that was achieved by reducing the best effort traffic flooded to the link. In figure 17b the average link utilization was increased to about 90%. Both of the figures 17a and 17b suggest that MRR offers the least latency bound for voice and video flows.

Figure 18 shows the Gini fairness index measurements for the 70% and 90% link utilization consecutively. Figure 18a suggests that MRR is fairer than the other three schedulers. This fairness advantage is much more obvious in figure 18b 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.

Figure 17a: Latency bounds with 70% link utilization

Figure 17b: Latency bounds with 90% link utilization

Figure 17: latency bound for different schedulers

Figure 18a: Average Gini Index with 70% link utilization

Figure 18b: Average Gini Index with 90% link utilization

Figure 18: Average Gini Index

Chapter 6

Conclusion and Future Work

In this thesis, 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. We have also shown some of the deficiencies present in some of the recently introduced frame-based scheduling algorithms such as ERR, PERR, and NDRR, and provided directions for solving these problems. MRR overcomes the unfairness problem present in these algorithms when flows are not always backlogged which is the case in real life networks.

Research in packet scheduling algorithms is pretty mature and a lot of work has been already accomplished. Future work related to this thesis could include the following:

- Study the end-to-end performance of a network of MRR schedulers.
- Study the end-to-end performance of a network of heterogeneous schedulers and derive the bandwidth and latency guarantees such a network could provide for multimedia sessions.
- Investigating the effect of aggregation on multimedia and real time sessions.

Bibliography

- [1] H. G. Perros and K. M. Elsayed, "Call admission control schemes: a review," IEEE Commun. Mag, vol. 34, no. 11, pp. 8291, Nov. 1996.
- [2] Hussein M. Alnuweiri, Lai-Yat Kelvin Wong, Tariq Al-Khasib, "Performance of New Link State Advertisement Mechanisms in Routing Protocols with Traffic Engineering Extensions", IEEE Communications Magazine, no. 5, May 2004.
- [3] Narvaez, P.; Kai-Yeung Siu; Hong-Yi Tzeng, "Traffic Engineering with Traditional Routing Protocols," IEEE/ACM Transactions on Networking, Volume: 9 Issue: 6, December 2001 pp. 706 -718.
- [4] D. Katz, K. Kompella, D. Yeung, IETF Draft: "Traffic Engineering Extensions to OSPF Version 2," draft-katz-yeung-ospf-traffic-10.txt, work in progress.
- [5] RFC 2205, "Resource ReSerVation Protocol (RSVP) - Version 1 Functional Specification".
- [6] E. Rosen, A. Viswanathan, R. Callon, RFC 3031, "Multiprotocol Label Switching Architecture," January 2001.
- [7] F. Le Faucheur, L. Wu, B. Davie, S. Davari, P. Vaananen, R. Krishnan, P. Cheval, J. Heinanen, RFC 3270, "Multi-Protocol Label Switching (MPLS) Support of Differentiated Services," May 2002.

- [8] Data Connection Portable MPLS Products, www.dataconnection.com.
- [9] A.K. Parekh, R. G. Gallagher, "A *generalized processor sharing approach to flow control in integrated services networks: the single-node case*," IEEE/ACM Transaction on Networking, vol. 1, pp. 344-357, 1993.
- [10] Abhay K. Parekh, Robert G. Gallagher, "A *generalized processor sharing approach to flow control in integrated services networks: the multiple node case*," IEEE/ACM Transactions on Networking 2(2): 137-150 (1994)
- [11] S. Keshav , "An *Engineering Approach to Computer Networking*," AddisonWesley Publishing Company, 1997.
- [12] A. Demers, S. Keshav and S. Shenker, "Analysis and Simulation of a *Fair Queuing Algorithm*," Computer Communication Review SIGCOMM '89 Symposium Communications Architectures and Protocols', vol. 19 no. 4 1989 p.1-12.
- [13] J.C.R. Bennett and H. Zhang, "WF2Q: *Worst-Case Fair Weighted Fair Queueing*," Proc. IEEE INFOCOM, pp. 120-128, Mar. 1996.
- [14] L. Zhang, "VirtualClock: a new traffic control algorithm for packet switching networks," ACM Transactions on Computer Systems, vol. 9, pp. 101--124, May 1991.
- [15] S. Golestani, "A *self-clocked fair queuing scheme for broadband applications*," Proc. IEEE INFOCOM, pp. 636-646, 1994.
- [16] H. Tayyar, H. Alnuweiri, "The Complexity of Computing Virtual-Time in Weighted Fair Queuing Schedulers," IEEE ICC, Paris, 2004.

- [17] H. Tayyar, "A Minimum-Work Weighted Fair Queuing Algorithm for Guaranteed End-to-End Quality-of-Service in Packet Networks," Ph.D. thesis, University of British Columbia, October 2002.
- [18] M. Katevenis, S. Sidiropoulos, C. Courcoubetis, "Weighted Round-Robin Cell Multiplexing in a General-Purpose ATM Switch Chip," *IEEE Journal on Selected Areas in Communications*, vol. 9, no. 8, October 1991, pp. 1265-1279.
- [19] M. Shreedhar, G. Varghese, "Efficient Fair Queuing using Deficit Round Robin," *IEEE/ACM Transactions on Networking*, vol. 4, no. 3, June 1996, p 375-385.
- [20] Salil S. Kanhere , Harish Sethu , Apha B. Parekh, "Fair and Efficient Packet Scheduling Using Elastic Round Robin," *IEEE Transactions on Parallel and Distributed Systems*, v.13 n.3, p.324-336, March 2002 .
- [21] S. Kanhere and H. Sethu, "Low-latency guaranteed-rate scheduling using elastic round robin," *Computer Communications*, vol. 25, no. 14, pp. 1315--1322, September 2002.
- [22] S. S. Kanhere, H. Sethu, "Fair, efficient and low-latency packet scheduling using nested deficit round robin," *Proc. IEEE Workshop on High Performance Switching and Routing*, Dallas, TX, 2001, pp. 6–10.
- [23] S. S. Kanhere, H. Sethu, "Prioritized Elastic Round Robin: An Efficient and Low-Latency Packet Scheduler with Improved Fairness," Technical Report DU-CS-03-03, July 2003.

- [24] S. Floyd and V. Jacobson, "*Link-sharing and resource management models for packet networks*," IEEE/ACM Transactions on Networking, vol. 3, no. 4, pp. 365--386, Aug. 1995.
- [25] S. Tsao, Y. Lin, "*Pre-order Deficit Round Robin: a new scheduling algorithm for packet-switched networks*," Computer Networks, February 2001, Vol. 35.
- [26] S. Kanhere, "*Design and Analysis of Fair, Efficient and Low-Latency Schedulers for High-Speed Packet-Switched Networks*," Ph.D. thesis, Drexel University, May 2003.
- [27] P. Goyal, H. M. Vin, and H. Haichen, "*Start-Time Fair Queuing: A Scheduling Algorithm for Integrated Services Packet Switching Networks*," IEEE/ACM Trans. Networking, 5(5): 690-704, Oct. 1997.
- [28] Stiliadis and A. Verma, "*Latency-rate servers: A general model for analysis of traffic scheduling algorithms*," IEEE Transactions on Networking, vol. 6, no. 3, pp. 611--624, October 1996.
- [29] OPNET Technologies Inc., www.opnet.com.
- [30] H. Shi and H. Sethu, "*An evaluation of timestamp-based packet schedulers using a novel measure of instantaneous fairness*," in Proceedings of IEEE International Performance, Computing and Communications, Phoenix, AZ, April 2003.
- [31] F. A. Cowell, "*Measuring Inequalities: Techniques for the Social Sciences*," John Wiley and Sons, New York, NY, 1977.
- [32] J. E. Stiglitz, "*Economics*," W. W. Norton and Co, 1993.

Appendix A: $\mathcal{LR}$ servers

The theory of $\mathcal{LR}$ servers provides a means to describe the worst-case behaviour of a broad range of scheduling algorithms in a simple and elegant manner [28]. For a scheduler to belong to this class, it is only required that the average rate of service offered by the scheduler to a busy session, over every interval starting at time Θ from the beginning of the busy period, is at least equal to its reserved rate. The parameter Θ is called the latency of the scheduler.

Let $A_i(\tau, t)$ be the arrivals from session i during the interval (τ, t) and let $W_i(\tau, t)$ denote the amount of service received by session i during the same interval. We can define a busy period of session i as the maximal interval of time $(\tau_1, \tau_2]$ such that for any time $t \in (\tau_1, \tau_2]$, packets of session i arrive with rate greater than or equal to ρ_i , or,

$$A_i(\tau, t) \geq \rho_i(t - \tau_1)$$

Where ρ_i is the reserved rate of session i .

This definition is best illustrated by figure 19 where flow I has two busy periods.

Figure 19: Busy periods of session i .

The distribution of the session busy periods depends only on the arrival pattern of the session and its reserved rate and it does not depend on the service discipline. In other words, the distribution of the session busy periods will be the same regardless the scheduling algorithm used if the same arrival pattern and reserved rate were used. Thus, the definition of the $\mathcal{LR}$ servers was based on the service received by a session over a busy period [28].

Definition A.1: A server S belongs in the class $\mathcal{LR}$ if and only if for all times t after time τ that the j th busy period started and until the packets that arrived during this busy period are serviced,

$$W_{i,j}^S(\tau, t) \geq \max(0, \rho_i(t - \tau - \Theta_i^S))$$

Where $W_{i,j}^S(\tau, t)$ denotes the total service provided to the packets of the session that arrived after time τ and until time t by server S .

Θ_i^S is the minimum non-negative number that satisfies the above inequality.

Figure 20 provides an example of the behaviour of an $\mathcal{LR}$ scheduler. The latency Θ_i^s represents the worst-case delay seen by the first packet of a busy period of a session i .

Figure 20: An example of the behaviour of an $\mathcal{LR}$ scheduler.

Appendix B: The Gini Index

In the field of economics, several measures of inequalities were used to study the social wealth distribution and many other economics issues of interest. The use of the *Lorenz curve* and *Gini index* [31] was borrowed by [30] to be used as a fairness measure in packet schedulers.

Figure 21: The Lorenz curve and Gini Index

Consider the problem of measuring the inequality among k quantities, $g_1 \leq g_2 \leq \dots \leq g_k$. Define $d_0 = 0$, and $d_i = d_{i-1} + g_i$, for $1 \leq i \leq k$. Now, a plot of d_i against i is a concave curve, known as the *Lorenz curve* [32], as shown in figure 21. Note that if there is perfect equality in these k quantities, the Lorenz curve will be a straight line starting from the origin. The Gini index measures the area between the Lorenz curve and this straight line, and thus it measures the inequality amongst the k quantities [31].

In the case of packet schedulers, the Gini index is the area between the Lorenz curve of the actual normalized service received and the Lorenz curve corresponding to the ideally fair GPS scheduler.