Design and Analysis of Fair, Efficient and Low-Latency Schedulers for High-Speed Packet-Switched Networks

A Thesis

Submitted to the Faculty

of

Drexel University

by

Salil S. Kanhere

in partial fulfillment of the

requirements for the degree

of

Doctor of Philosophy

May 2003

Dedications

This thesis is dedicated to my wife Alpa and my family.

Acknowledgements

Several people supported my efforts during my graduate years and I would like to thank them. First of all, I must point out the invaluable help of my advisor, Dr. Harish Sethu. This dissertation would not have been possible without his encouragement and valuable guidance. I feel fortunate that I had a chance to work with such an excellent mentor, and I would like to express my gratitude to him. Special thanks to him for giving so much time and attention to my dissertation and helping me with the documentation and presentation.

I would also like to thank Dr. Constantine Katsinis, Dr. Oleh Tretiak, Dr. Lazar Trachtenberg and Dr. Lloyd Greenwald for serving on my committee.

Special thanks to Yunkai Zhou for helping me with the formatting of the dissertation. I would also like to thank Hongyuan Shi for her help in the simulation efforts using the Gini index. And of course all the others in the Computer Communications Lab : Madhu, Hapreet and Adam for their help and advice.

Last but not least, I would like to thank my family for their constant support and guidance and Alpa for her love and support.

Once again, thanks Dr. Sethu for all the help and time that you have given us and for making these five years at Drexel memorable.

Table of Contents

List of Tables

List of Figures

Abstract

Design and Analysis of Fair, Efficient and Low-Latency Schedulers for
High-Speed Packet-Switched Networks

Salil S. Kanhere
Harish Sethu, Ph.D.

A variety of emerging applications in education, medicine, business, and entertainment rely heavily on high-quality transmission of multimedia data over high speed networks. Packet scheduling algorithms in switches and routers play a critical role in the overall Quality of Service (QoS) strategy to ensure the performance required by such applications. Fair allocation of the link bandwidth among the traffic flows that share the link is an intuitively desirable property of packet schedulers. In addition, strict fairness can improve the isolation between users, help in countering certain kinds of denial-of-service attacks and offer a more predictable performance. Besides fairness, efficiency of implementation and low latency are among the most desirable properties of packet schedulers.

The first part of this dissertation presents a novel scheduling discipline called *Elastic Round Robin (ERR)* which is simple, fair and efficient with a low latency bound. The per-packet work complexity of ERR is O(1). Our analysis also shows that, in comparison to all previously proposed scheduling disciplines of equivalent complexity, ERR has significantly better fairness properties as well as a lower latency bound. However, all frame-based schedulers including ERR suffer from high start-up latencies, burstiness in the output and delayed correction of fairness.

In the second part of this dissertation we propose a new scheduling discipline called *Prioritized Elastic Round Robin (PERR)* which overcomes the limitations associated with the round robin service order of ERR. The PERR scheduler achieves this by rearranging the sequence in which packets are transmitted in each round of the ERR scheduler. Our analysis reveals that PERR has a low work complexity which is independent of the number of flows. We also prove that PERR has better fairness and latency characteristics than other known schedulers of equivalent complexity. In addition to their obvious applications in Internet routers and switches, both the ERR and PERR schedulers also satisfy the unique requirements of wormhole switching, popular in interconnection networks of parallel systems.

Finally, using real gateway traces and based on a new measure of instantaneous fairness borrowed from the field of economics, we present simulation results that demonstrate the improved fairness characteristics and latency bounds of the ERR and and PERR schedulers in comparison with other scheduling disciplines of equivalent efficiency.

Chapter 1. Introduction

Over the last decade, high-speed packet switched networks have become the standard mode of communication, replacing the earlier telephone-based circuit-switched networks. In circuit-switched networks, a dedicated path known as a *circuit* is established between the communication end points. In addition, the resources needed along this path (buffers, link bandwidth) are reserved for the entire duration of the communication session. The network is responsible for allocating sufficient resources to allow the sender to transmit data as a continuous data flow at its peak transmission rate. A circuit in a link is implemented with either *time-division multiplexing (TDM)* or *frequency-division multiplexing (FDM)*. The public switched telephone network (PSTN) is an example of a circuit switched network.

It is well-known that circuit switching is not very efficient because the dedicated circuits are idle during *silent periods*, i.e., when the sender and receiver are not exchanging any messages. This results in poor utilization of the link bandwidths, especially given the fact that Internet traffic is extremely assymetric and bursty. In addition, establishing end-to-end circuits and reserving end-to-end resources is complicated and requires complex signaling mechanisms. To address these shortcomings, *packet switching* networks [1] were introduced as an alternative to circuit-switching. With packet switching, the sender breaks down messages into smaller chunks of data called *packets* and each packet is transmitted over a series of communication links just as in circuit switching. However, an important difference is that in packet switching no resource reservation is required in the network. If one of the links is congested because multiple packets need to be transmitted over the link at the same time, then one of the packets is chosen for transmission and the rest have to be stored in a buffer at the sending end of the link. As a result, buffering is required in the intermediate nodes of the network to absorb traffic bursts and avoid packet losses. The main advantage of packet-switching is that it permits *statistical multiplexing* [2] on the communication links. This allows packets from different sources to share the links resulting in an efficient utilization of the link capacities. Today's Internet is a quintessential packet-switched network. The Internet makes its *best-effort* to deliver packets in a timely manner but it does not make any guarantees.

Until recent years, the best-effort Internet architecture supplemented by the reliable transport protocol, TCP, has provided good service for the vast majority of the *traditional* applications such as file transfer, e-mail, web-browsing and remote login. For example, even though a file transfer application would ideally like to have a very high bandwidth and low end-to-end delay it will still work correctly if the available link bandwidth decreases and the end-to-end delay increases. In other words, the performance requirement of such applications are *elastic* since they can adapt to the resources that are available. These applications are called *best effort applications* since the network only guarantees to attempt to deliver their packets without providing for any performance guarantees.

In the last few years we have witnessed a rapid growth in the development and deployment of new networked applications in education, business, entertainment and medicine. Examples of such applications are many and are rapidly rising. Popular services such as video and audio on-demand, online radio and TV stations and interactive games are transforming the entertainment industry. Multimedia teleconferencing and banking are just a couple of other applications that are impacting businesses. Storing multimedia information in servers and allowing users access to it through networks is an economical solution to the problem of large-scale distribution of massive amounts of multimedia content. For example, applications such as distance learning enable universities to offer courses using video servers and high-performance networks. In medicine, many hospitals are already experimenting with high-resolution video and imaging applications that transfer data from various hospital facilities to the diagnostic center. All of these applications and many others, will have a profound impact on our day-to-day lives in ways that we have least imagined.

The utility value of the applications described above will be determined largely by the quality of the multimedia transmission and the ability of networks to carry heavy volumes of such traffic in a scalable, reliable and predictable fashion. In order to support multimedia traffic, the network must be able to guarantee performance bounds or meet the Quality-of-Service (QoS) requirements. The best-effort service model is highly ineffective in meeting their QoS demands. The task of designing network architectures, suitable for these applications is both critical and challenging. Providing the necessary bandwidth and keeping delay within known and manageable bounds are some of the concerns that must be addressed. For example, to support an application that carries voice as a 64 Kbps stream, the network must provide 64 Kbps bandwidth on the entire path from end-to-end. Moreover, if the application is two-way interactive, besides guaranteeing a bandwidth of 64 Kbps, the network must also guarantee a round trip delay of around 150 ms. Thus, this application and others of its kind, demand a guarantee of service quality from the network. As a result, they are known as *guaranteed-service applications*. Such applications are not elastic or adaptive and variations in delay or loss of packets can have a detrimental effect on their functionality.

Another motivation behind enabling QoS guarantees is to achieve service differentiation [3]. Providers may want to offer an eclectic mix of services to their customers. The service to be carried on the Internet may consist of real-time traffic for applications such as Internet telephony. A second class of service may be for applications such as transaction processing that require reliable, low-delay delivery of data. Yet another class of service to be carried on the Internet is the best-effort traffic, constituting traffic from applications such as file transfer and e-mail. Real-time applications, such as Internet telephony, demand a guarantee on the performance bounds and require the network to reserve resources on their behalf. Services for other applications, such as queries and responses in transaction processing, may not require stringent performance guarantees but may still require lower delays than best-effort applications. A network should thus be capable of providing a variety of services with different QoS requirements to be carried on the same switching nodes or links. These different applications have varying traffic characteristics with different requirements, and rely on the ability of the network to provide QoS guarantees with respect to several quality measures, such as end-to-end delay, bandwidth allocation, delay jitter and packet loss. However, network resources such as link bandwidth and buffers are shared by multiple users or services, some or all of which may try to access a resource simultaneously. Resource contention arises because of this sharing. A QoS mechanism is needed to efficiently apportion, allocate and manage limited network resources among competing users. QoS deals with engineering and managing network resources to deliver the performance levels that satisfy user's expectations. A QoS mechanism is necessary when there are not enough resources to prevent queues from becoming congested and when congestion degrades performance guarantees. So, effective queue management is fundamental to many proposed QoS schemes.

An essential component of a queue manager is a *scheduler* which employs a scheduling mechanism to decide which packet to serve next. In other words, given a set of resource requests in the service queue, a switch uses a scheduling algorithm to decide which request to serve next. A scheduler has to ensure that the network resources are scheduled fairly among its contending users. Scheduling disciplines are important because they are responsible for protecting one user's traffic from another and hence are a key to fair sharing of the network resources. The scheduling algorithms also affect the performance received by a certain traffic flow. To understand this, let us assume that each traffic flow is assigned a separate queue and the switch queues the packets that are ready for transmission from different traffic flows in their respective queues. By the choice of its service order, the scheduler can allocate different mean delays to the packets belonging to different queues. Also it can allocate different bandwidths by serving a certain minimum number of packets from a particular flow in a given time interval. We thus need a scheduling algorithm that supports fair resource allocation and also supports these performance bounds in order to serve the performance critical applications such as telephony and other interactive audio and video applications. Fair scheduling becomes especially critical in access networks, within metropolitan area networks and in wireless networks where the resource capacity constraints tend to be significantly limiting to high-bandwidth multimedia applications today. Even with the over-provisioning of resources such as is typical in the Internet core, fairness in scheduling is essential to protect flows from other misbehaving flows triggered by deliberate misuse or malfunctioning software on routers or end-systems. Fairness in the management of resources is also helpful in countering certain kinds of denial-of-service attacks. Fair schedulers have now found widespread implementation in switches and Internet routers [4, 5]. Some of the most desirable properties of a scheduling discipline include:

- *Fairness:* The available link bandwidth must be distributed among the flows sharing the link in a fair manner. We use the classic notion of fairness given by the max-min fair share policy [6] which is explained in greater detail in the following section. In general, it is desirable that the scheduler serves the connections proportional to their reservations and distributes the unused bandwidth left behind by the idle sessions proportionally among the active ones. In addition, flows should not be penalized for the excess bandwidth they received while other flows were inactive. Fairness is also desirable for good performance, since unfair treatment of some traffic flows in the network can easily lead to unnecessary bottlenecks.

- *Isolation:* The scheduling algorithm must isolate a flow from the undesirable effects of other (possibly misbehaving) flows. This will ensure that the QoS guarantees for a flow will be maintained even in the presence of other misbehaving flows. Note that, isolation among flows is necessary even when traffic policing strategies are used for traffic shaping at the edge of the network, since the flows may become increasingly bursty as they traverse through the network [7]. Isolation among flows also results in a more predictable performance for end user applications.

- *Latency:* It is desirable that the scheduling discipline provides an end-to-end delay guarantee to individual flows. For guaranteed-rate services, the latency should be measured as the length of time it takes a new flow to begin receiving service at the guaranteed rate. Low delay bounds imply low buffer requirements for guaranteeing no packet loss. Thus, the latency of a scheduler has a direct effect on the cost of implementation in terms of the required memory. The latency is also directly related to the amount of playback buffering required at the receiver for real-time communication applications.

- *Efficiency:* In addition to providing performance bounds and being fair, it is also important that a scheduler be easily implementable. A scheduler should require as few simple operations as possible to make a scheduling decision. In particular, the number of operations should be as independent of the number of flows that are to be scheduled as possible. Thus, if n is the total number of queues or traffic flows to be scheduled by a scheduler, then a scheduler that has $O(1)$ time complexity is preferred in comparison to the one that has $O(n)$ time complexity. This property is especially desired in high-speed networks and in routers where the number of flows can be in thousands as in the Internet core.

Even though a network must ideally schedule every multiplexed resource, in this dissertation, we have concentrated on the most commonly scheduled resource: the bandwidth on a link.

Besides the Internet, fair scheduling algorithms also find their place in the interconnection networks for parallel systems, wherein packets belonging to different traffic flows often share links in their respective paths toward their destinations. Fairness is an intuitively desirable property in the allocation of bandwidth available on a link among multiple traffic flows that share the link. Fairness in packet scheduling becomes especially desirable with the increasing use of parallel systems in multi-user environments with the interconnection network shared by several users at the same time. Fair allocation of bandwidth at links within a network is a necessary requirement for providing protection to flows, i.e., for ensuring that the performance is not affected when another possibly misbehaving flow tries to send packets at a rate faster than its fair share. In multi-user environments, the protection guaranteed by fair scheduling of packets improves the isolation between users, a quality strongly desired by customers of parallel systems [8]. Isolation offers a more predictable performance to user applications, which also facilitates repeatability of performance results necessary for reliable benchmarking of systems and applications without taking all the users off the system. Fairness is also essential for good performance with compilers that take into account the predicted communication delays in the network. Strict fairness is also desirable for good performance, since unfair treatment of some traffic flows in the network can easily lead to unnecessary bottlenecks.

Most switch architectures designed for interconnection networks of parallel systems, however, eliminate only the worst kinds of unfairness such as starvation, where packets belonging to one traffic flow may not be scheduled for an indefinite period of time. The need to design a fair scheduler in these networks is an important motivation for the work presented in this dissertation. In addition, even though over the last decade, a number of advances have been made in the architecture of these switches for improved delay and throughput characteristics, very few researchers have focused on possible performance gains with scheduling strategies that may be used at the output link or for access to output queues from the input queues. However, implementing fair scheduling algorithms in the interconnection networks of parallel systems, poses certain unique challenges which are described in detail in the following section.

1.1 Overview of Wormhole Switching

Almost all interconnection networks, both direct and indirect, are constructed out of switches connected together in a certain topology. Wormhole switching is a popular switching technique used in the implementation of switches in interconnection networks for parallel systems [8, 9]. Wormhole switching is distinguished by the fact that the granularity of flow control in a wormhole network can be smaller than a packet [10, 11]. A message packet is broken up into *flits*, a flit being the unit of flow control. Thus, a message is pipelined through the network at the flit level and at any instant of time, a blocked message occupies queues in several routers. We use the terms router and switch interchangeably in this dissertation. In order to not add to the per-flit overhead, only the head flit (the first flit) of the packet contains information necessary to route the packet through the network. A switch in the network reads the information in the head flit and directs it to the next switch or end-system device on its path. The rest of the data flits of a packet are simply forwarded to the same output link as the head flit. Consequently, the transmission of distinct packets cannot be multiplexed over a physical link. A packet must be transmitted in its entirety before a link can be used to transmit another packet.

Implementing fair scheduling disciplines in such a switch poses certain unique challenges which we describe in the following paragraphs. Consider a wormhole switch with several input and output queues, with packets in the input queues ready for transmission to one of the output queues. We define a *queue* as a logical entity containing a sequence of flits that have to be served in a FIFO order. Note that, depending on the buffering architecture of the switch, a queue may not be the same as a buffer since a single buffer can implement multiple logical queues [12, 13]. Consider a certain packet *A*, with a head flit that arrives at an input queue of a wormhole switch and is ready to be routed to an output queue. Once the head flit has been routed to the output queue, as already explained, no flits from any other packet can be routed to this output queue, until all of the remaining flits of packet *A* are routed. Thus, in Figure 1.1, packet *B* that also wants to use the same output link as *A* is blocked and cannot make progress until all the flits from packet *A* have been transmitted.

In wormhole switches with virtual channels [14], one typically has as many output

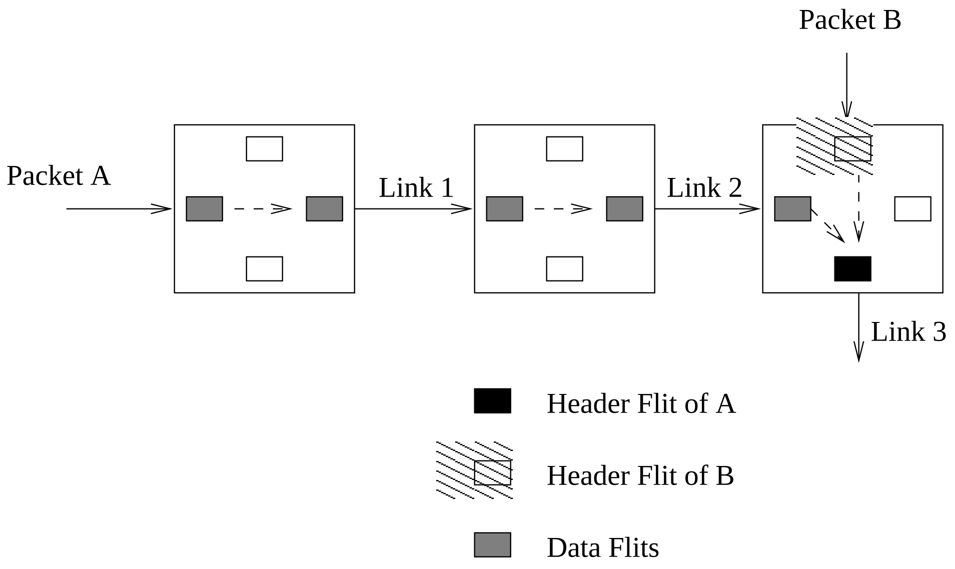


Figure 1.1: Example of downstream congestion in wormhole networks

queues as there are virtual channels associated with each of the output links. Since each flit is marked by the virtual channel it belongs to, in scheduling flits to the output link from these output queues, it is not necessary to schedule all flits belonging to a packet before a flit from another virtual channel is scheduled. If each flow can be assigned a separate virtual channel, one may use the Flit-Based Round Robin (FBRR) scheduler which visits the flow queues in round-robin fashion, and transmits one flit from each queue. This scheme is very fair among the flows in terms of the number of flits scheduled from each flow over any interval of time. However, this scheme for achieving fairness is prohibitively expensive since it can only be used if there are as many virtual channels implemented as there are flows (which can be in hundreds or thousands). In addition, by serving packets flit-by-flit, the FBRR scheme uniformly increases the delay of packets in all the flows [15]. Further, multiplexing of flits belonging to different packets is not always feasible. For example, even in switches with virtual channels, while scheduling entry into the output queues from the various input queues, all flits of a packet have to be scheduled before a flit from another packet enters the same output queue. A packet-by-packet scheduler, therefore, is a more suitable and all-encompassing solution. Such a scheduler can also achieve a better average delay.

Wormhole switches typically use a flit-by-flit credit-based flow control protocol, and therefore, downstream congestion can thwart the progress of the packet currently being served for an unpredictable length of time. Since, as explained earlier, it may not always be possible to time-multiplex the transmission of packets from different flows, one cannot always begin forwarding packets from another flow until all flits belonging to the packet currently being served are forwarded. Thus, during the time that a packet is in the middle of its transmission, packets from other flows may be blocked without access to the output link even while there are no flits being transmitted over the link. A packet of length L bytes, scheduled for forwarding to an output queue feeding a link of capacity C bytes/second, may take more than L/C seconds for transmission. In other words, the length of a packet cannot singly determine the length of time it takes to dequeue a packet while it blocks other flows from access to the output queue.

Therefore, the relevant measure of the use of a resource, in this case the output link, is the length of time a flow occupies the link. In wormhole networks, therefore, fairness should be based on the length of time each flow occupies a link, and not on the number of flits sent by each flow over the link. This length of time depends on the downstream congestion which can be hard to predict without complex feedback mechanisms. In wormhole networks, unlike in Internet routers and many other networks, this length of time cannot be accurately estimated from knowledge of the length of the packet being transmitted. The actual length of time that a packet takes to be dequeued, thus, may not be known until the last flit of the packet is dequeued. A scheduling discipline for wormhole networks, therefore, should be able to make a decision on starting the transmission of a packet without knowledge of the length of time it will take to transmit the entire packet. In addition, the algorithm also cannot assume an upper bound on this length of time. In other words, the unique requirements of wormhole switching require that a scheduler perform its operations without *any* assumptions on how long it will take to transmit a packet. Such a requirement on the scheduling discipline is also essential in networks where packet delimiters are the only indication of the beginning and the end of packets, with no length fields in the packet headers. For example, an ATM network transmitting IP packets over AAL5, where the end of the packet is not known until the arrival of the last ATM cell corresponding to the packet.

1.2 What is Fair Resource Allocation ?

Traditionally, a *flow* is defined as a sequence of packets generated by the same source and headed toward the same destination via the same path in the network. It is assumed that packets belonging to different flows are queued separately while they await transmission. A *scheduler* dequeues packets from these queues, and forwards them for transmission. A flow is said to be *active* during a period if its queue is non-empty throughout this period. A flow is *inactive* when its queue is empty. Note that, even though we use the above traditional definition of a flow to present our results in this paper, a flow can also be more broadly defined as any distinct sequence of packets queued separately at the scheduler and competing with other sequences of packets for service by the scheduler. For example, in parallel systems, a flow may also be defined as the set of all packets belonging to the same user, with packets of these flows queued at the scheduler accordingly.

A precise definition of fairness is essential before further discussion of fair scheduling of flows. The classic notion of fairness in the allocation of a resource among multiple requesting entities with equal rights to the resource but unequal demands, is the *max-min fair share* policy [6]. It can be formally defined as follows,

- The resource is allocated in order of increasing demand.

- No requesting entity gets a share of the resource larger than its demand.

- Requesting entities with unsatisfied demands get equal shares of the resource.

Generalized Processor Sharing (GPS) [16] is an unimplementable but ideal scheduling discipline that satisfies the above notion of max-min fair allocation. GPS is unimplementable because it does not transmit packets as entities and assumes that the traffic is infinitely divisible. The GPS scheduler visits each non-empty queue in a round-robin order, and serves an infinitesimally small amount of data from each queue, such that in any finite time interval, it can visit every queue at least once. Consider a set of n flows denoted by $1, 2, \ldots, n$ demanding bandwidths $b_1, b_2, \ldots, b_n$ on a link of total bandwidth B. All of the n flows have an equal right to the link bandwidth. Without loss of generality, assume $b_1 \leq b_2 \leq \cdots \leq b_n$. The GPS scheduler first allocates B/n of the bandwidth to each of the active flows. If this is more than the bandwidth demanded by flow 1, the unused bandwidth, $B/n - b_1$, is divided equally among the remaining $n-1$ flows. If the total bandwidth allocated thus far to flow 2 is more than b_2, the unused excess bandwidth is again divided equally, this time among the remaining $n - 2$ flows. The allocation process of the GPS scheduler continues in this fashion until each flow has received no more than its demand, and if the demand was not satisfied, no less than any other flow with higher demand. GPS thus satisfies the max-min fair share allocation.

Note that, thus far, we had assumed that all the flows had the same right to the resource. However, since guaranteed-rate flows require the network to guarantee a certain minimum average rate, a weight must be associated with each such flow. The weight allocated to a flow should be proportional to its reserved rate. The concept of max-min share can be extended to *max-min weighted fair share allocation* to account for the flow weights. The only change is that the resource allocated to each flow is now normalized by its weight. The GPS scheduler can be similarly modified to account for the flow weights. Let r represent the transmission rate of the output link and let ϕ_i represent the weight assigned to flow i. If $B(t)$ denotes the set of flows that are active at time $t \geq 0$, the flow i is guaranteed to receive a minimum service rate of $r_i(t)$ given by,

$$r_i(t) = \begin{cases} \frac{\phi_i}{\sum_{j \in B(t)} \phi_j} r & : \quad i \in B(t) \\ 0 & : \quad \text{otherwise} \end{cases} \qquad (1.1)$$

The effectiveness of a fair scheduler is measured by how closely it approximates GPS. Over the last decade, a number of scheduling disciplines have been proposed which try to emulate the GPS scheduler.

1.3 Classification of Scheduling Disciplines

The scheduling disciplines in general can be classified broadly as:

1. Work-conserving: A work-conserving scheduler is never idle while there are packets waiting to be transmitted in service queues.

2. Non-work-conserving: A non-work-conserving scheduler may be idle even if there are packets waiting to be served. A scheduler may, for example, postpone the transmission of a packet when it expects a higher priority packet to arrive soon, even though it is currently idle.

   There are arguments both in favor of and against the above two classes of scheduling disciplines. For example, an argument against non-work conserving scheduling disciplines is that they waste link bandwidth. However, by idling away the link bandwidth, the non-work conserving schedulers can make the traffic arriving at the downstream switches more predictable, reducing the delay jitter experienced by a certain traffic flow [17]. However, work-conserving servers always have lower average delays than non-work-conserving servers. Examples of work-conserving schedulers include Weighted Fair Queuing (WFQ) [16, 18], Virtual Clock Queuing [19, 20] and Deficit Round Robin (DRR) [21]. On the other hand, Hierarchical Round Robin (HRR) [22] and Stop-and-Go Queuing [23] are non-work-conserving schedulers. The interested reader may refer to [6] for more details.

The scheduling disciplines can also be classified into one of the following two categories based on their internal architecture:

1. Sorted-Priority Schedulers: These schedulers maintain a global variable known as the *virtual time* or the *system potential* function. A timestamp computed as a function of this variable is associated with each packet in the system. Packets are sorted based on their timestamps, and are transmitted in that order.

2. Frame-based or Round Robin Policies: In these schemes, on the other hand, the scheduler provides service opportunities to the backlogged flows in a particular order (usually round-robin) and, during each service opportunity, the intent is to provide the flow an amount of service proportional to its fair share of the bandwidth.

Weighted Fair Queuing, Virtual Clock Queuing, Self-Clocked Fair Queuing (SCFQ) [24], Time-Shift Scheduling [25] and Frame-Based Fair Queuing [1] (FFQ) [26] and Worst-Case Fair Weighted Fair Queuing (WF^{2}Q) [27] are some of the popular sorted-priority queuing mechanisms. The basic idea is depicted in Figure 1.2. These schedulers differ in the manner in which they calculate the global virtual time function. They generally provide good fairness and delay properties but are not very efficient. There are two major costs associated with the implementation of sorted-priority schedulers:

1. The complexity of computing the system virtual time: For WFQ, the worst-case complexity is $O(n)$ where n is the number of flows sharing the same output link. However, in a number of schedulers such as SCFQ, SFQ and FFQ proposed in recent years, the complexity of computing the virtual time is $O(1)$.

2. The complexity of maintaining a sorted list of packets based on their timestamps, and

[1]Note that, Frame-based Fair Queuing, in spite of its name, is actually a sorted-priority scheduling discipline. The algorithm uses a framing approach similar to that used in frame-based schedulers to update the state of the system. However, as in sorted-priority schedulers, packets are transmitted based on their timestamps.

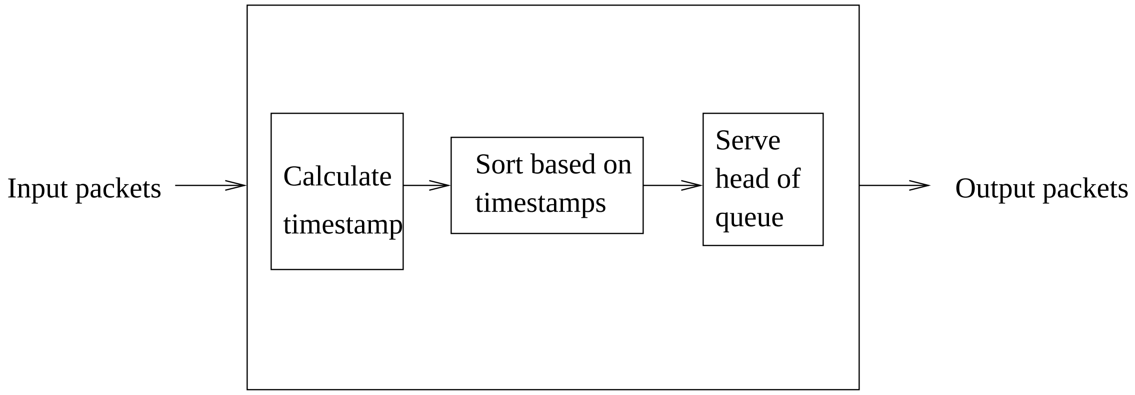


Figure 1.2: Sorted-priority schedulers

the complexity of computing the maximum or the minimum in this list prior to each packet transmission. For n flows the work complexity of the scheduler prior to each packet transmission is $O(\log n)$.

Thus, with a large number of flows, the sorted-priority schedulers becomes expensive to implement at high speeds. Attempts have been made to improve the efficiency of sorted-priority schedulers; however, such attempts either do not avoid the implementation bottleneck or compromise on fairness.

On the other hand, the frame-based schedulers do not maintain a global virtual time function and also do not require any sorting among the packets available for transmission. This reduces the implementation complexity of frame-based scheduling disciplines to $O(1)$, making them attractive for implementation in routers, and especially so, in hardware switches. Examples of frame-based schedulers are Packet-by-Packet Round Robin (PBRR) [28] and Deficit Round Robin (DRR) [21, 29].

1.4 Representative Schedulers

A large number of scheduling algorithms have been proposed in the literature (refer to [30] for a survey). In this section we will review some of these scheduling algorithms and outline their properties.

1.4.1 First Come First Serve (FCFS)

FCFS is one of the most common queuing algorithms employed in switches. As the name implies, in FCFS, the order of arrival completely determines the bandwidth allocation. FCFS service is trivial to implement, requiring a router or a switch to store only a single head and tail pointer per output link. However, FCFS fails to provide adequate protection from a bursty source that may suddenly send packets at a rate higher than its fair share for brief periods of time. It is easy to see that this rogue flow will capture an arbitrary part of the outgoing bandwidth. Also such a source can significantly increase the upper bound on the queuing delay of packets belonging to a flow from another source. Fairness, however, requires that as long as a source is demanding bandwidth within its rightful share, the delay experienced by packets from this source should not be affected by other traffic in the network.

1.4.2 Round Robin Service Policy

Consider several flows, with flits belonging to packets waiting in the respective queues to be forwarded to another queue or an output link. Two implementation techniques for round-robin policy are possible.

1. Flit-Based Round Robin (FBRR): One scheduling technique would be to use a pure *Flit-Based Round Robin* scheme, in which the scheduler visits each flow's queue in a round-robin fashion, and transmits one flit from each queue. This scheme is only possible in wormhole networks when each flit is tagged with a flow id, such as when

each flow represents a virtual channel [14]. This scheme is very fair among the flows in terms of the number of flits scheduled from each of the queues during any time interval. However, it cannot be used in other contexts such as for scheduling packets from input queues to output queues in a wormhole switch for the reasons described in Section 1.1. In addition, by serving packets flit-by-flit, the FBRR scheme uniformly increases the delay of packets in all the flows [15].

2. Packet-Based Round Robin (PBRR): An alternate technique would be *Packet-Based Round Robin* proposed by Nagle [28], in which the scheduler visits each of the queues in a round-robin fashion, and transmits an entire packet from a queue before beginning transmission from another queue. This reduces the average latency experienced by a packet as compared to FBRR. These techniques and a number of their variations have been analyzed in [31] for their performance characteristics, but not for their fairness properties. The PBRR scheduling discipline, for example, ignores the packet lengths and would be fair if the average packet size over the interval of a connection were the same for all the traffic flows, in which case each flow would get an equal share of the outbound bandwidth. It is, however, not fair among the flows when the packet sizes in the different flows are not equal. Consequently, flows sending longer packets use up an unfairly high fraction of the available transmission bandwidth. In the worst case, a flow can get Max/Min times the bandwidth of another flow, where Max is the maximum size of the packet and Min is the minimum packet size.

1.4.3 Weighted Fair Queuing (WFQ)

Scheduling mechanisms such as Weighted Fair Queuing also known as Packet-by-Packet Generalized Processor Sharing (PGPS) [16, 18, 32], try to emulate the ideal GPS scheme by time-stamping each arriving packet with the *finish number*, which is the expected completion time that a packet would have had if it were scheduled by the GPS scheduler. The

WFQ then serves the packets in the increasing order of the finish numbers. Hence, this requires computation of the finish number for every packet and then sorting among these time-stamps to determine the relative order in which the packets are to be served. In other words WFQ simulates GPS *on the side* and uses the result of the simulation to determine the service order.

Let $R(t)$ be the number of rounds of service made by the hypothetical GPS scheduler up to time t known as the *round number*. Depending on the number of flows served, each round of service takes a variable amount of time: the greater the number of flows served, the longer a round takes. The time taken to serve one bit from each active flow is the length of a round and it increases in direct proportion to the number of active flows. Hence, in order to calculate round number, the WFQ scheduler keeps track of the number of active flows, since the round number grows at the rate inversely proportional to n, where n is the total number of active flows. The finish number of a packet arriving to an inactive connection, i.e, a flow whose queue is empty when the packet arrives, is the sum of the current round number and the size of the packet in bits. If a packet arrives at an active flow, then its finish number is the sum of the largest finish number of a packet in its queue and the packet size in bits. Let $L(i, k, t)$ be the size of the k-th packet that arrives on flow i at time t and $F(i, k, t)$ be the finish number for the kth packet on flow i. Also let $\phi(i)$ be the weight associated with flow i. Note that, this weight is a function of the reserved rate requested by flow i. Then,

$$F(i, k, t) = max\{F(i, k-1, t), R(t)\} + \frac{L(i, k, t)}{\phi(i)} \tag{1.2}$$

Note that, the finish number is only a service tag that indicates the relative order in which the packet is to be served, and has nothing to do with the actual time at which the packet is served.

WFQ conforms to the definition of fairness, in the sense that a flow is not punished if it temporarily exceeds its reserved rate to take advantage of the unused bandwidth. This can be understood with a simple example given below:

Example 1: Let the packet sizes be constant, and let the output channel have a rate of 1 bit/s. The scheduler has two flows i and j, each with the same weight. Assume that the scheduler is initialized at time 0, so that $R(0) = 0$ and the finish number of the flows i and j are also 0 at time 0. Suppose that a packet of size 100 bits arrives at time 0 on flow i and no packet from flow j is received up to time 100. Since this is the first packet to arrive on flow i, its finish number at time 0, $F(i, 1, 0) = 0 + L(i, 1, 0) = 100$ as given by Equation (1.2). At time 100, 100 bits from i have been forwarded to the output channel. Since $\partial R/\partial t = 1/n$, $R(100) = 100$. Hence, from Equation (1.2), assuming that the flow i is still active, the finish number of flow i at time 100 is equal to 200. At time 100, a packet arrives on flow j and it is assigned the finish number, $F(j, 1, 100) = max\{0, 100\} + 100 = 200$. Since both the flows have same finish number, the scheduler alternates between the flows and hence the bandwidth is equally shared. Thus flow i is not punished for using the bandwidth unused by flow j for the first 100 time units.

The above property may not be satisfied by other scheduling disciplines as will be illustrated in a subsequent example. While this scheme guarantees absolute fairness, the packet processing cost makes it hard to implement economically at high speeds. Since it needs to simulate a GPS scheduler in parallel for updating the round number, the work complexity of a WFQ scheduler is $O(\log n)$ [32]. In general, higher the number of flows going through the switch, the more expensive it is to implement WFQ.

1.4.4 Self-Clocked Fair Queuing (SCFQ)

To improve the implementation complexity of WFQ, an approximate implementation called Self-Clocked Fair Queuing was proposed in [24]. In SCFQ, the finish number of a packet is computed based on the packet currently in service at the scheduler. In other words, when a packet arrives to an empty queue, instead of using the round number to compute its finish number, it uses the finish number of the packet currently in service. Thus, if *CF* represents the finish number of the packet currently being served by the SCFQ scheduler the finish number is now calculated as,

$$F(i,k,t) = max\{F(i,k-1,t), CF\} + \frac{L(i,k,t)}{\phi(i)} \tag{1.3}$$

This approach reduces the complexity of the algorithm greatly. However, the price paid is the unfairness over short time scales which also results in larger delay bounds as compared with WFQ.

1.4.5 Virtual Clock Queuing (VCQ)

The basic idea of virtual clock queuing [19, 20] is inspired by Time Division Multiplexing (TDM). The way WFQ emulates GPS, VCQ emulates time-division multiplexing (TDM). In the virtual clock method, the scheduler time-stamps the arriving packets with the completion times under time-division multiplexing, and then serves packets in order of these completion times. Using the same notation as above, the finish numbers under VCQ is computed as:

$$F(i,k,t) = max\{F(i,k-1,t), VC\} + \frac{L(i,k,t)}{\phi(i)} \tag{1.4}$$

where VC is the time at which the packet is received by the scheduler. This scheme, however, like WFQ suffers from the cost associated with sorting among the time-stamps. Also this scheme may not always be fair as explained by the following example:

Example 2: The scheduler has two flows i and j, each with a reserved rate of 1/2 packet/s. Assume that the packet sizes are constant (1 bit) and the output channel has a rate of 1 packet/s. The scheduler is initialized at time 0. From time 0 up to time 100, packets from flow i arrive at a rate higher than 1 packet/s, and no packet is received from flow j. At time 0, when the first packet arrives on flow i, from Equation (1.4), its finish number, $F(i,1,0) = 2$. The finish number of the packet that arrives at time 1, $F(i,2,1) =$

$max\{2,1\}+2=4$. Proceeding in the similar manner, at time 100, the finish number or the time-stamp of flow i is 202. At time 100, packets from j arrive at the rate of at least 1 packet/s, and packets from i also continue to arrive. However, when the first packet from flow j is received, its time-stamp, $F(j,1,100) = max\{0,100\}+2 = 102$. Since the time-stamp of flow j is less than the time-stamp of flow i, no packet from i will be forwarded to the output channel until 50 packets from j are forwarded, that is until time 150. In effect, i is denied service for 50 time units because it earlier took advantage of bandwidth unused by j. It was seen in the earlier example that this unfairness did not occur in WFQ.

1.4.6 Deficit Round Robin (DRR)

All of the schedulers described above, are sorted-priority schedulers and hence do not avoid the O($\log n$) complexity associated with sorting among the timestamps. Deficit Round Robin (DRR) [21], a less fair but more efficient scheduling discipline with an O(1) per-packet work complexity, was proposed by Shreedhar and Varghese in 1996. DRR is not a timestamp-based algorithm, and therefore, avoids the associated computational complexity. DRR achieves O(1) time-complexity because it serves the active flows in a strict round-robin order [21,33]. It succeeds in eliminating the unfairness due to different packet lengths observed in pure PBRR. This is done by keeping a state, associated with each queue called a *deficit count (DC)* to measure the past unfairness. A *Quantum* is assigned to each of the queues and when a flow is picked for service, its DC is incremented by the quantum value for that flow. A packet is served from a queue only if the packet size at the head is less or equal to the sum of the quantum and the deficit counter value; otherwise, the scheduler begins serving the next flow in the round robin sequence. When a packet is transmitted, the DC corresponding to that flow is decremented by the size of the transmitted packet.

In DRR, in order that the per-packet work complexity is O(1), one has to make sure that the quantum value chosen is no smaller than the size of the largest packet that may potentially arrive at the scheduler [21]. Otherwise the per-packet work complexity increases to O(n) since one may encounter a situation, in which, even after visiting each of the n flows and examining the respective DC values, no packet is eligible for transmission. A per-packet work complexity of O(1) is ensured if we make sure that at least one packet is transmitted from each active flow during each round. This is ensured if the quantum is no smaller than the size of the largest possible packet, since this guarantees that the packet size at the head of each queue at the start of its service opportunity will always be less than the sum of the DC value and the quantum value of the flow. In order to achieve a per-packet work-complexity of O(1), therefore, the DRR scheduler requires knowledge of the upper bound on the size of a packet. DRR, thus, is not ideally suitable for wormhole networks since it requires the knowledge of the size of a packet before making a decision on transmitting it, and in addition requires an upper bound on the size of a packet.

1.4.7 Surplus Round Robin (SRR)

In [34, 35], a fair scheduler similar to DRR was proposed. This algorithm, later known as *Surplus Round Robin (SRR)*, has also been used in other contexts such as in [36]. SRR is a modified version of DRR, in which the scheduler continues serving a flow as long as the DC value of the flow is positive. When the DC becomes negative, the scheduler begins serving the next flow in the round robin sequence. Thus, while DRR never allows a flow to overdraw its account but rewards an under-served flow in the next round, SRR allows a flow to overdraw its account but penalizes the flow accordingly in the next round. DRR keeps an account of each flow's deficit in service, while SRR keeps an account of the surplus service received by each flow. SRR does not require the scheduler to know the length of a packet before scheduling it. However, it does require the use of a fixed quantum assigned to each flow per round. As in DRR, in order to ensure an O(1) per-packet work complexity, the quantum value has to be no smaller than the size of the largest packet that may potentially arrive at the scheduler. SRR, like DRR, cannot be readily adapted for use in wormhole switching since it also requires knowledge of the upper bound on packet sizes.

1.5 Contributions

In this section we will outline the important contributions of this dissertation. As explained in Section 1.1, wormhole switching, popular in interconnection networks of parallel systems, imposes certain unique restrictions on the scheduling algorithms. These constraints can be summarized as follows:

- The scheduler should be able to make a decision on dequeueing a packet without knowing the length of time it takes to transmit transmission of a packet without knowledge of how long it will take to transmit the packet.

- The scheduler also cannot assume an upper bound on the length of time it takes to transmit the packets.

Over the last decade, a variety of scheduling algorithms that seek to achieve fairness in bandwidth allocation have been proposed and implemented in Internet routers [16, 18, 21, 24, 27, 32, 34–37]. A number of these scheduling disciplines were discussed in Section 1.4. Unfortunately, most fair scheduling disciplines proposed for Internet routers are either too expensive to implement in high-speed hardware switches because of the work complexity of per-packet processing, or cannot be easily adapted to the unique requirements of wormhole networks described above. For example, most timestamp-based schedulers such as Weighted Fair Queuing [6] have a work complexity of O($\log n$) with respect to the number of flows. On the other hand, more efficient schedulers such as Deficit Round Robin (DRR) [21] and Surplus Round Robin (SRR) [34–36] require knowledge of the upper bound on packet lengths to achieve a work complexity of O(1), rendering them difficult to adapt to wormhole networks.

In the first part of this dissertation we present a novel and simple scheduling discipline called *Elastic Round Robin (ERR)*, which is designed to address the unique requirements of wormhole networks. In traditional scheduling literature, it is typically assumed that the length of time it takes to transmit a packet is directly proportional to the size of the packet. Therefore, the problem of designing a fair scheduler for wormhole networks is equivalent to the problem of designing a fair scheduler in the traditional sense but without the scheduler making *any* assumptions on the size of a packet before beginning the transmission. Based on this equivalence, we present the ERR scheduler as a solution to the latter problem. It should be noted that in many real interconnection networks as well as in Internet routers, packet headers do carry a field with the packet length in it, and therefore, the problem in such cases is not a lack of knowledge of the packet length. However, when a scheduler uses the size of a packet to make its decisions, it cannot be readily adapted to the unique requirements of wormhole switching.

In spite of the constraints of wormhole switching imposed on the design, ERR is also suitable for use in Internet routers for scheduling best-effort connections. The emerging high-speed packet-switched networks are expected to support a variety of services beyond the best-effort service available in the Internet today. The fair packet scheduling algorithms in switches and routers play a critical role in providing the Quality-of-Service (QoS) guarantees required by the new multimedia applications. We also present a modified version of ERR which can be used for scheduling these guaranteed-rate application flows. Further we present analytical results on the efficiency, fairness and performance characteristics of ERR. A scheduler is considered to be efficient if the order of the work complexity of enqueuing and dequeuing a packet, with respect to the number of flows, is $O(1)$. We prove that the work complexity of ERR is $O(1)$, equal to or better than other scheduling disciplines. The fairness of a scheduling discipline is measured using a well-known and widely used metric, known as the *Relative Fairness Bound (RFB)* [24]. We prove that the relative fairness bound of ERR is $3m$, where m is the size of the largest packet that *actually* arrives during the execution of ERR. For guaranteed-rate flows another important performance characteristic is the latency, which is measured as the cumulative length of time that a newly active flow has to wait until it can start receiving service at its reserved rate. We also evaluate the latency bound of ERR and prove that it belongs to the class of *Latency-Rate ($\mathcal{LR}$) Servers*, a general class of guaranteed-rate schedulers [38]. Our analysis proves that the ERR algorithm has better fairness properties as well as better performance characteristics than other scheduling disciplines of comparable efficiency such as Deficit Round Robin and Surplus Round Robin. As a result, ERR is an attractive scheduling discipline for both best-effort and guaranteed-rate traffic.

As explained earlier, frame-based schedulers are extremely efficient with an $O(1)$ complexity making them attractive for implementation in routers and, especially so, in hardware switches. However, these frame-based schedulers suffer from a number of disadvantages. Let us first re-examine the behavior of a frame-based scheduler such as ERR. The ERR scheduler works in *rounds*, where a round is defined as one round-robin iteration over all flows that are active at the start of the round. The scheduler selects the flow at the head of this list and serves it for a continuous period of time in proportion to its weight. This results in a highly bursty packet stream at the output of the scheduler. A high degree of burstiness in the traffic increases the delay jitter which can in turn have an adverse effect on the performance of real-time applications. In addition, when a new flow becomes active it has to wait until all the other previously active flows are served by the scheduler before it can receive any service. As a result, the latency bound of these schedulers can be considerably greater than the sorted-priority schedulers. In addition, due to the round robin order of service, a flow that is lagging in service in comparison to other flows has to wait for its compensation for this service lag in the subsequent round. Further, there is no means for such a lagging flow to receive precedence over all the other flows which have already received more than their fair share of the service in the previous round. These weaknesses stem from the round robin nature of the service order and from the fact that each flow receives its entire share of service in the round at once in one service opportunity.

In the second part of this dissertation, we address the above problems associated with the round robin service order of the ERR scheduler and present a a scheduling discipline called Prioritized Elastic Round Robin (PERR) [39] as a solution. The total service received by a flow in a round in PERR is identical to the service received by the flow in the corresponding round in ERR. However, in PERR, this service received by a flow is split into several parts over the course of the round. The PERR scheduler eliminates the strict round-robin nature of service order and re-orders the transmission sequence within each round of the ERR scheduler. This reordering of packets allows the flows that have received less service in the previous round to gain precedence over the other flows in the current round. The exact manner in which the transmission sequence is re-ordered depends on a certain per-flow state that indicates how far ahead or behind a flow is in consuming its share of bandwidth. This re-ordering of the transmission sequence in PERR is accomplished by adding a limited number, p of priority queues to the original architecture of the ERR scheduler. We conclude this part of our work with a detailed analysis of the fairness and performance characteristics of the PERR. To simplify the analysis, we use a novel approach based on interpreting the PERR scheduler as an instance of the Nested Deficit Round Robin (Nested-DRR) discipline discussed in [40]. Our analysis shows that both the relative fairness bound and the latency bound of PERR are lower than those of other schedulers of comparable efficiency. We also show that the worst-case work complexity of the PERR scheduler is $O(\log p)$, where p denotes the number of priority queues. It is important to note that $p \ll n$, where n is the total number of flows being serviced by the scheduler. As a result, the work complexity of the PERR scheduler is much lower than those of sorted-priority schedulers such as WFQ which have a work complexity of $O(\log n)$. The low work complexity of PERR also results in an efficient and simple software implementation. In addition, since PERR is based on the ERR scheduler it too satisfies the constraints of wormhole routing and hence can also be used for scheduling flows in interconnection networks of parallel computer systems.

The fairness and latency measures used in this dissertation and in other literature on scheduling algorithms, however, are only bounds and do not accurately capture the behavior of the scheduler most of the time under normal circumstances. Recently a new instantaneous measure of fairness known as the *Gini index* [41] has been proposed. This index is adapted from the measures of inequalities used in the field of economics. A complete evaluation of a scheduler for real-time multimedia traffic is possible with this measure. In the final part of this work, we present extensive simulation results to comparitively judge the instantaneous fairness achieved by ERR and PERR in comparison with other scheduling disciplines of equivalent complexity such as DRR, SRR and Pre-order DRR [42]. We also include the WFQ scheduler as a representative sorted-priority scheduler in our comparisons. For our simulations we make use of both synthetic traffic and real gateway traffic traces. Besides the fairness, we also present a comparison of the latency bounds of these schedulers.

1.6 Dissertation Organization

The rest of the dissertation is organized as follows. Chapter 2 presents a detailed description of the Elastic Round Robin (ERR) scheduling algorithm along with the rationale behind it. In Chapter 3 we presents analytical results on the efficiency, fairness and performance characteristics of ERR. Chapter 4 first highlights the problems associated with the frame-based schedulers. We then present the Prioritized Elastic Round Robin (PERR) as a solution to eliminating these limitations. In Chapter 5 we evalulate the performance and fairness bounds of the PERR scheduler. Chapter 6 presents a simulation-based evaluation of the instantaneous fairness and latency bounds of ERR and PERR as compared to other fair efficient schedulers. Finally, Chapter 7 gives a summary of this dissertation, together with some conclusions and directions for future research.

Chapter 2. Elastic Round Robin

In this chapter we propose a novel fair, efficient and low-latency scheduling discipline called Elastic Round Robin (ERR) which addresses the constraints imposed by wormhole switching as described in Section 1.1. Even though ERR is designed for wormhole networks, it can be used in a wide variety of contexts whenever there is a shared resource that needs to be allocated fairly among multiple requesting entities. In some of these contexts its unique properties relevant to wormhole switching are critical, and in some others, its advantages derive from its simplicity, better fairness and better performance characteristics. For e.g., ERR can be implemented in the Internet routers for fair scheduling of various flows of traffic corresponding to a source-destination pair. Also, despite the constraints of wormhole switching imposed on the design, ERR can be used to schedule packets in such a network, since it does not use the knowledge of the packet length to make a scheduling decision. Besides scheduling packets from input queues to output queues in wormhole switches, the ERR algorithm can actually also be used for achieving low average delay in the fair scheduling of packets to the output link from output queues belonging to various virtual channels. ERR can also be easily adapted for scheduling guaranteed-rate flows.

Because of the wide applicability of our solution, and so that this work may be readily understood and used in a variety of contexts, we present this algorithm as a solution to the following abstraction of the problem. Consider n flows, each with an associated queue with packets in it. All the n flows have a continuous stream of arbitrary sized packets arriving to the switch and all these flows wish to leave the switch via the same outgoing link as shown in Figure 2.1. The scheduler dequeues packets from these queues according to a scheduling discipline and forwards them for transmission on an output link or to another queue. As in traditional scheduling problems, we allow that the length of time it takes to dequeue a packet is proportional to the size of the packet. However, to apply this work to wormhole

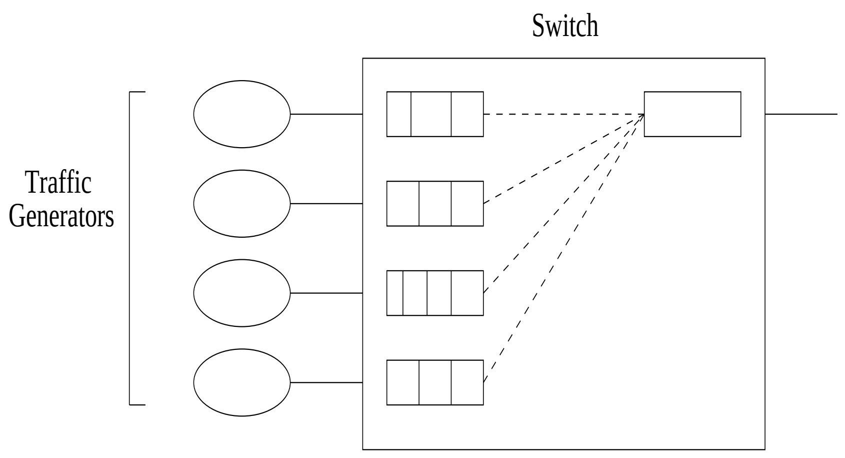


Figure 2.1: The switch model

networks, we require that the scheduling algorithm not know the length of a packet until it has completely dequeued the packet. For wormhole networks, references to the length of the packet in the algorithm may be replaced by length of time it takes to dequeue the packet. In all of this chapter, we use a flit as the smallest piece of a packet that can be independently scheduled, and we measure the length of a packet in terms of flits.

The aim of the ERR scheduler is to give each flow a fair share of the output bandwidth and hence provide protection from flows that may send large sized packets or the flows that may send packets at a high rate. The main idea is as follows: in ERR, each flow is assigned an *allowance,* measured in terms of the number of flits, that a flow can transmit in any given round. The queues are serviced in round robin order. When a queue is selected for service, ERR scheduler calculates its *allowance*. ERR serves a queue as long as the number of flits transmitted from that queue in any round is less than its allowance in that round. The ERR scheduler, however, allows a flow to exceed its allowance. Associated with each queue is a state, *surplus count* which keeps track of the extra bandwidth that a flow used in any given round. The *surplus count* is used to calculate the allowance for a flow in the following round, so that if a flow overdraws its allowance by some amount, it is penalized by this amount in the next round. A flow that tries to seize a large fraction of bandwidth beyond its fair share will have a large value of surplus count. In the following round, the ERR scheduler allows the other flows to transmit at least as many flits as this flow did in addition to its fair share in the previous round. Thus, the flows that received little service in a round are compensated for in the next round.

The rest of the chapter is organized as follows. Section 2.1 presents a detailed description of the ERR scheduler along with the rationale behind it. ERR was originally designed for use in wormhole networks such as the interconnection networks of parallel computer systems. In addition, ERR can also be used for scheduling best-effort traffic in the Internet. In Section 2.2, we present a weighted version of ERR which can be used for scheduling guaranteed-rate flows.

2.1 Algorithm Description

A pseudo-code implementation of the ERR scheduling algorithm is shown in Figure 2.2, consisting of *Initialize*, *Enqueue* and *Dequeue* routines.

We define a flow as *active* when a packet belonging to this flow is in the middle of being dequeued by the scheduler, or when the queue corresponding to the flow is not empty. In ERR, we maintain a linked list, called the *ActiveList*, of flows which are active. The ERR scheduler moves through this list in a round robin manner and serves packets from the queue it points to. A flow whose queue was previously empty and therefore not in the *ActiveList*, is added to the tail of the list whenever a new packet belonging to the flow arrives. The ERR scheduler serves the flow i at the head of this list. After serving flow i, if the queue of flow i becomes empty, it is removed from the list. On the other hand if the queue of flow i is not empty after it has received its round robin service opportunity, flow i is added back to the tail end of the list.

```
Initialize: (Invoked when the scheduler is initialized)
    RoundRobinVisitCount $= 0$;
    PreviousMaxSC $= 0$;
    MaxSC $= 0$;
    for ($i = 0$; $i < n$; $i = i + 1$)
        $SC_i = 0$;

Enqueue: (Invoked when a packet arrives)
    $i =$ QueueInWhichPacketArrives;
    if (ExistsInActiveList(i) == FALSE) then
        AddToActiveList(i);
        Increment SizeOfActiveList;
        $SC_i = 0$;
    end if;

Dequeue:
    while (TRUE) do
        if (RoundRobinVisitCount $== 0$) then
            PreviousMaxSC = MaxSC;
            RoundRobinVisitCount = SizeOfActiveList;
            MaxSC $= 0$;
        end if;
        $i =$ HeadOfActiveList;
        RemoveHeadOfActiveList;
        $A_i = 1 + PreviousMaxSC - SC_i$;
        $Sent_i = 0$;
        do
            TransmitPacketFromQueue($i$);
            Increase $Sent_i$ by LengthInFlitsOfTransmittedPacket;
        while ($Sent_i < A_i$);
        $SC_i = Sent_i - A_i$;
        if ($SC_i > MaxSC$) then
            $MaxSC = SC_i$;
        end if;
        if (QueueIsEmpty == FALSE) then
            AddQueueToActiveList($i$);
        else
            $SC_i = 0$;
            Decrement SizeOfActiveList;
        end if;
        Decrement RoundRobinVisitCount;
    end while;
```

Figure 2.2: Pseudo-code for ERR

The *Enqueue* routine is invoked whenever a new packet arrives at a flow. The *Enqueue* routine queues a packet for an output link of a switch. This is done by looking at the flow identifier or flow id in the packet header. Then, if the flow identifier does not already exist in the *ActiveList*, it is added to the list and its *Surplus Count* is reset to zero and the *SizeOfActiveList* is incremented by one. The *Dequeue* routine is the heart of the algorithm which schedules packets from the queues corresponding to different flows. As long as there are packets queued for an output link, this process is active. Thus, the work to process a packet involves two parts: enqueuing and dequeuing.

Consider the instant of time, t_1, when the scheduler is first initialized. We define *Round 1* as one round robin iteration starting at time t_1 and consisting of visits to all the flows that were in the *ActiveList* at time t_1. We illustrate this definition of a round using Figure 2.3. Assume that flows A, B and C are the only flows active at the beginning of *Round 1*. The visits of the scheduler to the flows A, B and C, comprise *Round 1*. Let flow D become active after the time instant t_1, but before the completion of Round 1 at time instant t_2. The scheduler does not visit flow D in *Round 1* since D was not in the *ActiveList* at the start of *Round 1*. *Round 2* is now defined as consisting of the visits to all of the flows that are in the *ActiveList* at time t_2. Assuming that flows A, B and C are still active at time t_2, *Round 2* will consist of visits to the flows A, B, C and D. In general, we define round i recursively as the set of visits to all the flows in the *ActiveList* at the instant round $(i-1)$ is completed. In order that the scheduler knows the number of flows it has to visit in any given round, we introduce the quantity *RoundRobinVisitCount* which denotes the number of flows that are in the *ActiveList* at the start of a round. This is done by setting *RoundRobinVisitCount* to the *SizeOfActiveList* at the start of each round. After a visit to each flow, *RoundRobinVisitCount* is decremented by one. This means that at any given time, *RoundRobinVisitCount* indicates the number of flows that are yet to be visited by the scheduler in the current round. When *RoundRobinVisitCount* eventually equals zero, it implies the end of a round.

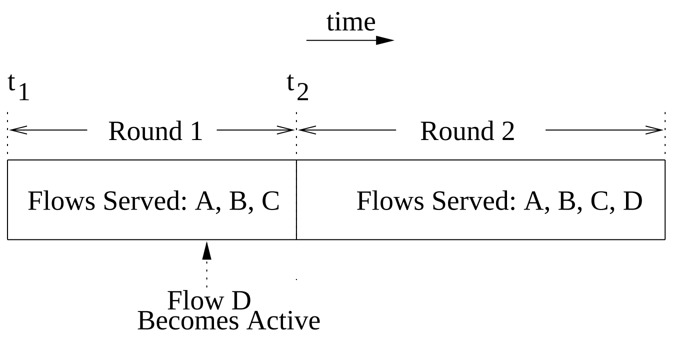


Figure 2.3: Definition of a round

In each round, the scheduling algorithm determines the number of flits that a flow is allowed to send. We call this quantity the *allowance* for the flow during that round. The allowance assigned to flow i during round s is denoted by $A_i(s)$. This allowance, however, is not a rigid one and is actually *elastic*, in that a flow may be allowed to send more flits in a round than its allowance. We allow a flow to exceed its *allowance* because, as already explained, in a wormhole switch, once the scheduler starts serving a packet, it will have to serve that packet in its entirety before it schedules packets from other flows for the output link. Let $Sent_i(s)$ be the number of flits that are transmitted from the queue of flow i in round s. Each time a packet is successfully transmitted, this quantity in incremented by the length of the packet in flits. The ERR scheduler will keep serving the packets from the queue, if the total number of flits transmitted by the flow so far in the current round is less than its allowance. The ERR scheduler, thus, makes the scheduling decision without any knowledge about the packet length.

Note that the last packet transmitted by a flow may cause it to exceed its allowance, as can happen when the allowance is smaller than the size of the packet at the head of the corresponding queue. The following example will help make this point clear. Consider a flow i, with allowance equal to 11 flits. Let the queue of flow i have two packets, and assume that the length of the packet at the head and the one following it be 10 and 20 flits, respectively. Now, the ERR scheduler will transmit the packet at the head. Since the total number of flits transmitted by the flow i so far in the current round is less than its allowance, ERR scheduler will schedule the 20-flit packet (now at the head of the queue), letting flow i to exceed its allowance by 19 flits. Note that every time a scheduler visits a certain flow, it transmits at least as many flits as its allowance from the queue of that flow. When a flow ends up sending more than its allowance, it is interpreted as having obtained more than its fair share of the bandwidth. The scheduler records this unfairness in the *Surplus Count* (SC) associated with each flow. The surplus count, during any round, is the number of flits the flow sent in addition to its allowance.

Let $SC_i(s)$ denote the surplus count of flow i in round r. Then after serving flow i in round s, the scheduler computes $SC_i(s)$ as

$$SC_i(s) = Sent_i(s) - A_i(s) \tag{2.1}$$

We introduce a quantity $MaxSC(s)$ which denotes the largest surplus count among all the flows served during round s. In other words,

$$MaxSC(s) = max\left\{SC_j(s)\right\}, \forall j \text{ served in round } s \tag{2.2}$$

When the ERR scheduler is first initialized this quantity is set to zero. In any given round, after a flow has been visited by the scheduler, its surplus count is computed using Equation (2.1). The ERR scheduler then checks if this surplus count is greater then the current value of $MaxSC(s)$ and if it is, it sets the value of $MaxSC(s)$ to this new surplus count. In a given round, after the scheduler has visited all the flows, let flow i be the one that has the largest value of the surplus count. In the following round, the scheduler ensures that each flow is allowed to send at least as many flits as flow i actually sent in excess of its allowance in the previous round. $MaxSC(s)$ is used, as follows, to recursively compute the allowances for each of the flows in the next round.

$$A_i(s) = 1 + MaxSC(s-1) - SC_i(s-1) \tag{2.3}$$

Note that, for flow i, which had the largest surplus count in the previous round, the new allowance is 1. This is ensured by the addition of 1 in (2.3) so that the scheduler will transmit at least one packet from this flow during the next round.

The allowance given to each of the flows in a given round is not fixed and is computed depending on the behavior of the flows in the previous round. After the ERR scheduler serves flow i, if the queue of flow i is empty, its surplus count is reset to zero and it is removed from the *ActiveList* and the *SizeOfActiveList* is decremented by one. Otherwise if flow i has packets in its queue that are ready for transmission, it is added back at the tail end of the list and we store its surplus count for use in the next round.

Figure 2.4 illustrates the first three rounds in an execution of the ERR scheduling discipline. In this figure, at the beginning of the first of these rounds, the surplus counts for all the three flows and the $MaxSC$ are all initialized to 0. Thus, from Equation (2.3), the allowance during round 1 is equal to 1 for all the flows. The sizes of the packets actually sent by the flow during this round are shown by the vertical bars, and the new allowances for the next round are again computed using Equations (2.1) and (2.3). The value of $MaxSC$ used in each round to compute the allowance is also shown. In round 1, flow A transmits a packet of length of 32 flits, thus exceeding its allowance by 31 flits. Flow B transmits a packet of length 16 flits and $SC_B(1)$ is computed as 15. The scheduler schedules a packet of length 24 flits from flow C and $SC_C(1)$ is computed as 23 flits. Thus, at the end of round 1, $MaxSC(1)$ equals $SC_A(1)$ and this is used to calculate the allowances for all the flows in the next round in accordance with Equation (2.3). In round 2, the allowance for flow A is minimum, while the allowances for the other flows have been increased, thus compensating for the past unfairness. It is easily observed from Figure 2.4 that, in general, flows which receive very little service in a round are given an opportunity to receive proportionately

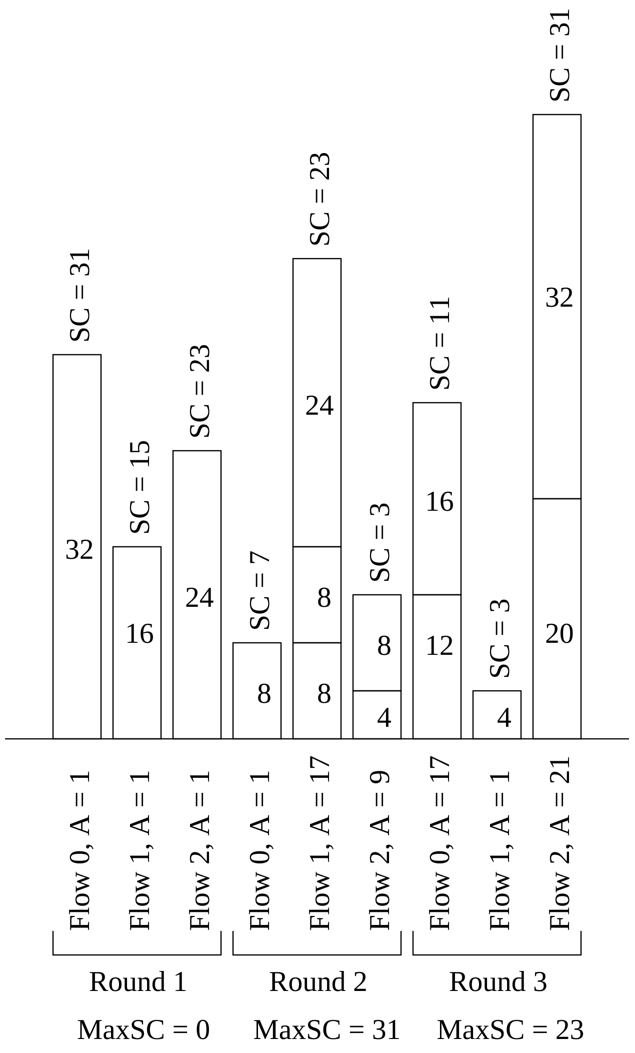


Figure 2.4: An illustration of 3 rounds in an ERR execution

more service in the next round.

Figure 2.5 shows a block diagram of a portion of ERR to illustrate the various operations used to determine when the scheduler should stop service of one flow and begin service for another. In the diagram, *PreviousMaxSC* and *RoundRobinVisitCount* are abbreviated as *PMaxSC* and *RRVC*. The thin lines in the figure indicate single-bit signals, while the thick

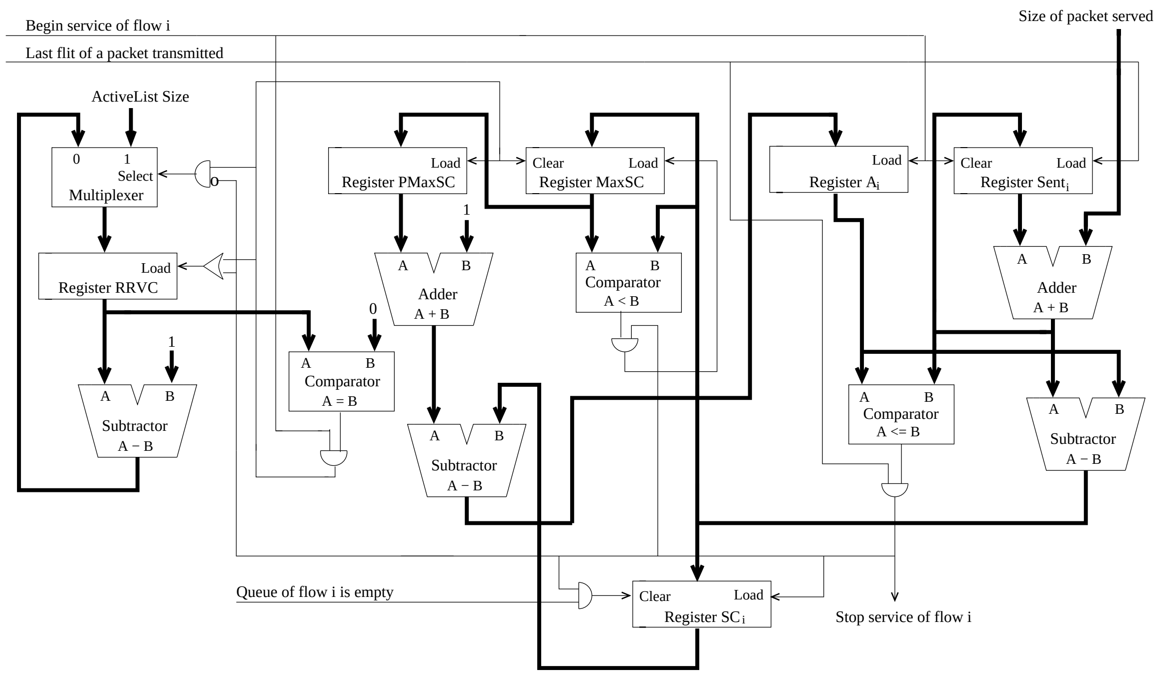


Figure 2.5: A block diagram illustration of the ERR scheduler

dark lines indicate multi-bit buses carrying quantities such as packet sizes and values of various counters.

2.2 Guaranteed-Rate Scheduling using ERR

In this section, we show that ERR can be easily adapted for scheduling guaranteed rate connections. We present a weighted version of ERR for guaranteed-rate services. Consider an output link of transmission rate r, access to which is controlled by the ERR scheduler. Let n be the total number of flows and let ρ_i be the reserved rate for flow i. Let ρ_{min} be the smallest of the reserved rates. Note that since all the flows share the same output link, a necessary constraint is that the sum of the reserved rates be no more than the transmission rate of the output link. In other words,

$$\sum_{i=1}^{n} \rho_i \leq r$$

In order that each flow receives service proportional to its guaranteed rate, the ERR scheduler assigns a weight to each flow. The weight assigned to flow i, w_i, is given by,

$$w_i = \frac{\rho_i}{\rho_{min}} \tag{2.4}$$

Note that for any flow i, $w_i \geq 1$.

The weighted version of ERR is exactly similar to the ERR algorithm described in the preceding section. The only difference is in the calculation of the *Allowance* and the $MaxSC(r)$. $MaxSC(s)$ is defined as the largest weighted surplus count among all the flows served in round s. In other words,

$$MaxSC(s) = max\left\{\frac{SC_j(s)}{w_j}\right\}, \forall j \text{ served in round } s \tag{2.5}$$

The allowance for each flow is calculated using the $MaxSC$ value in the previous round, as follows:

$$A_i(s) = w_i(1 + MaxSC(s-1)) - SC_i(s-1) \tag{2.6}$$

Chapter 3. Performance Analysis of ERR

In this chapter we present a detailed analysis of the performance characteristics of the ERR scheduling discipline. We evaluate the performance of ERR based on the following important properties:

- *Efficiency*: The efficiency of a scheduling discipline is measured in terms of the order of work complexity associated with the enqueuing and dequeuing operations, with respect to n, the number of active flows. In high-speed networks with large numbers of active flows, the time available for a scheduler to make its scheduling decision is very small. Hence, it is desirable that the time to enqueue a received packet or to dequeue a packet for transmission is as independent as possible of the number of flows sharing the output link. A per-packet work complexity of $O(1)$ is most desirable.

- *Fairness:* The available link bandwidth must be distributed among the flows sharing the link in a fair manner. This ensures that the performance achieved by a flow is not affected when a possibly misbehaving flow tries to transmit packets at a rate faster than its fair share. We measure fairness using a well-known and widely used metric, known as the relative fairness bound [24].

- *Latency:* An appropriate measure of packet schedulers in this regard, especially for schedulers seeking to provide guaranteed services is the upper bound on the length of time it takes a new flow to begin receiving service at the guaranteed rate [38]. The latency bound is directly related to the amount of playback buffering required at the receiver.

The results in this chapter present an analytical proof that the ERR algorithm has better fairness properties as well as better performance characteristics than other fair scheduling disciplines of comparable efficiency such as DRR and SRR.

The rest of this chapter is organized as follows. In Section 3.1, we prove that the work complexity of ERR is $O(1)$, equal or better than other scheduling disciplines. Section 3.2 presents the fairness analysis of ERR. We prove that the relative fairness bound of ERR is $3m$, where m is the size of the largest packet that *actually* arrives during the execution of the ERR scheduler. Section 3.3 briefly describes the concept of latency-rate ($\mathcal{LR}$) servers, a general class of schedulers proposed by Stiliadis and Verma in [38]. In Section 3.4, we evaluate the latency bound of ERR and prove that it belongs to the class of $\mathcal{LR}$ servers. In addition, we also show that the latency bound derived in this section is a tight one.

3.1 Work Complexity

The work complexity of a scheduling discipline is defined as follows,

Definition 3.1.1 *Consider an execution of a scheduling discipline over n flows. We define the work complexity of the scheduler as the order of the time complexity, with respect to n, of enqueuing and then dequeuing a packet for transmission.*

Note that, this definition of work complexity does not include the transmission time of the packet.

Theorem 3.1.1 *The work complexity of an ERR scheduler is O(1).*

Proof: We prove the theorem by showing that enqueuing and dequeuing a packet are each of time complexity O(1).

The time complexity of enqueuing a packet is the same as the time complexity of the *Enqueue* procedure in Figure 2.2, which is executed whenever a new packet arrives at a flow. Determining the flow at which the packet arrives is an O(1) operation. Once, we figure out the queue to enqueue, the packet is appended to the end of the queue. The flow at which the new packet arrives is added to the *ActiveList*, if it is not already in the list. This addition of an item to the tail of a linked list data structure is also an O(1) operation.

We now consider the time complexity of dequeuing a packet. During each service opportunity, the ERR scheduler transmits at least one packet. Thus, the time complexity of dequeuing a packet is equal to or less than the time complexity of all the operations performed during each service opportunity. Each execution of the set of operations inside the while loop of the *Dequeue* procedure in Figure 3.2, represents all operations performed during each service opportunity given to a flow. These operations include determining the next flow to be served, removing this flow from the head of the *ActiveList* and possibly adding it back at the tail. All of these operations on a linked list data structure can be executed in O(1) time. Additionally, each service opportunity includes updating the values of surplus count and allowance corresponding to the flow being served, and also updating the values of *MaxSC*, *PreviousMaxSC*, *SizeOfActiveList* and *RoundRobinVisitCount*. All of these can be done in constant time, as represented by the constant number of operations in the dequeue procedure in Figure 2.2. ■

3.2 Fairness Analysis

The fairness of a scheduling discipline is best measured in comparison to the GPS scheduling algorithm. The quantity, known as the *Absolute Fairness Bound (AFB)* of a scheduler *S*, is defined as the upper bound on the difference between the service received by a flow under *S* and that under GPS over all possible intervals of time. This bound is often difficult to derive analytically. Also it has been shown in [43] that the *AFB* is related by a simple equation to another popular fairness measure known as the *Relative Fairness Bound (RFB)* first proposed in [24]. The *RFB* is also much easier to evaluate as compared to the *AFB*. In our fairness analysis, we therefore make use of the *RFB*. Our metric is identical to the one used in [21]. . The *RFB* is defined as the maximum difference in the service received by any two flows over all possible intervals of time. The following provides a more rigorous definition. In the following, a flow is considered *active* during an interval of time, if, during this interval, its queue is never empty of packets awaiting transmission. We consider only the active flows to measure the fairness because it makes no sense in comparing a flow that is not active with the one that is, since the former does not receive any service when it is not active.

Definition 3.2.1 *Let $Sent_i(t_1, t_2)$ be the number of flits transmitted by flow i during the time interval between t_1 and t_2. Given an interval (t_1, t_2), we define the Relative Fairness, RF(t_1, t_2) for this interval as the maximum value of $|Sent_i(t_1, t_2) - Sent_j(t_1, t_2)|$ over all pairs of flows i and j that are active during this interval. Define the relative fairness bound (*RFB*) as the maximum of RF(t_1, t_2) for all possible time intervals (t_1, t_2).*

It is desirable that *RFB* be a small constant. The smaller the *RFB*, the closer the scheduler emulates the GPS scheduler which is considered an ideal fair scheduling algorithm.

Definition 3.2.2 *Define m as the size in flits of the largest packet that is actually served during the execution of a scheduling algorithm.*

Definition 3.2.3 *Define M as the size in flits of the largest packet that may potentially arrive during the execution of a scheduling algorithm. Note that, $M \geq m$.*

Lemma 3.2.1 *For any flow i and round r in the execution of an ERR scheduling discipline,*

$$0 \leq SC_i(r) \leq m - 1 \tag{3.1}$$

Proof: The lower bound on $SC_i(r)$ in the expression of the lemma is obvious since the ERR algorithm always schedules at least as many flits as $A_i(r)$ during round r. The only exception is when the queue for flow i becomes empty in round r, in which case the surplus count of the flow is reset to 0.

The ERR algorithm never begins dequeuing a new packet in a flow after the number of flits sent in a round r is equal to or more than the allowance $A_i(r)$. Thus, the lowest value of the allowance at which the ERR scheduler may select a new packet for transmission is 1, and this will be the last packet transmitted by the flow during this round. Since the size of this packet can be no greater than m, from Equation (2.1), the upper bound in the expression of the lemma is proved. ■

The following corollary follows directly from Lemma 3.2.1.

Corollary 3.2.1 *In any round r, $MaxSC(r)$ is bounded as follows,*

$$0 \leq MaxSC(r) \leq m - 1 \tag{3.2}$$

The next theorem gives the upper and the lower bounds on the number of flits that any flow can transmit in n consecutive rounds during which it is active.

Theorem 3.2.1 *Given n consecutive rounds starting from round k during which flow i is active, the bounds on the total number of flits, N, transmitted by flow i are given by,*

$$n + \sum_{r=k-1}^{k+n-2} MaxSC(r) - (m-1) \leq N \leq n + \sum_{r=k-1}^{k+n-2} MaxSC(r) + (m-1)$$

Proof: Substituting for $A_i(r)$ using Equation (2.3) into Equation (2.1), we get,

$$Sent_i(r) = 1 + MaxSC(r-1) - SC_i(r-1) + SC_i(r) \tag{3.3}$$

Now, since N is the total number of flits transmitted by flow i from round $r = k$ to $r = k + n - 1$,

$$N = \sum_{r=k}^{k+n-1} Sent_i(r)$$

Over n rounds of servicing of flow i, starting form round k, we get the following values of $Sent_i(r)$:

$$Sent_i(k) = 1 + MaxSC(k-1) - SC_i(k-1) + SC_i(k)$$

$$Sent_i(k+1) = 1 + MaxSC(k) - SC_i(k) + SC_i(k+1)$$

$$\vdots$$

$$Sent_i(k+n-2) = 1 + MaxSC(k+n-3) - SC_i(k+n-3) + SC_i(k+n-2)$$

$$Sent_i(k+n-1) = 1 + MaxSC(k+n-2) - SC_i(k+n-2) + SC_i(k+n-1)$$

Summing up the above n Equations, we get:

$$\begin{gathered} Sent_i(k) + Sent_i(k+1) + \cdots + Sent_i(k+n-1) = \\ n + \sum_{r=k-1}^{k+n-2} MaxSC(r) + SC_i(k+n-1) - SC_i(k-1) \end{gathered}$$

Recall that:

$$N = Sent_i(k) + Sent_i(k+1) + \cdots + Sent_i(k+n-1)$$

Therefore, we get:

$$N = n + \sum_{r=k-1}^{k+n-2} MaxSC(r) + SC_i(k+n-1) - SC_i(k-1) \tag{3.4}$$

Using Lemma 3.2.1, $0 \leq SC_i(k+n-1) \leq m-1$, and $0 \leq SC_i(k-1) \leq m-1$. The result of the theorem is readily obtained by substituting for these bounds on $SC_i(k-1)$ and $SC_i(k+n-1)$ in Equation (3.4). ■

We now proceed to prove the bound on the fairness measure of the ERR scheduling discipline. Note that, the relative fairness bound, RFB, is defined taking into consideration all possible intervals of time (t_1, t_2). In the following, we prove that a tight upper bound can be obtained considering only a subset of all possible time intervals. This subset is the set of all time intervals bounded by time instants that coincide with the beginning or the end of the service opportunity of flows.

Definition 3.2.4 *Let* $\mathbf{T}$ *be the set of all time instants during an execution of the ERR algorithm. Define* $\mathbf{T}_s$ *as the set of all time instants at which the scheduler ends serving one flow and begins serving another. Define by* $F(t)$*, for* $t \notin \mathbf{T}_s$*, the flow which is being served at time instant* t*. For* $t \in \mathbf{T}_s$*, we define* $F(t)$ *as the flow just about to begin service.*

The following lemma allows us to prove an upper bound on the fairness measure, stated in Theorem 3.2.2, considering only the time intervals (t_1, t_2), where $t_1, t_2 \in \mathbf{T}_s$.

Lemma 3.2.2 $\mathrm{RFB} = \max\limits_{t_1, t_2 \in T_s} \mathrm{RF}(t_1, t_2)$.

Proof: This lemma is proved if for any $t_1, t_2 \in \mathbf{T}$, we can find $t_1', t_2' \in \mathbf{T_s}$, such that RF$(t_1', t_2') \geq$ RF(t_1, t_2).

Consider any two active flows i and j during the interval between t_1 and t_2, where $t_1, t_2 \in \mathbf{T}$. Without loss of generality, assume that during this interval, more flits have been scheduled from flow i than from flow j. By appropriately choosing t_1' as the time instant at either the beginning or the end of the service opportunity given to $F(t_1)$ at time t_1, one may verify that RF$(t_1', t_2) \geq$ RF(t_1, t_2). Similarly, an appropriate choice of t_2' as either the beginning or the ending instant of the service opportunity given to $F(t_2)$ at t_2, can lead to RF$(t_1', t_2') \geq$ RF(t_1, t_2). ■

Theorem 3.2.2 *For any execution of the ERR scheduling discipline, FM $< 3m$.*

Proof: By the statement of Lemma 3.2.2, we need to only consider all time intervals bounded by time instants that coincide with the starting or ending of service to a flow. We therefore prove the statement of the theorem using the time interval between instants t_1 and t_2, where both t_1 and t_2 belong to $\mathbf{T}_s$.

Consider any two flows i and j that are active in the time interval between t_1 and t_2. From the algorithm in Figure 2.2, it follows that after flow i receives service, it is added to the tail end of the *ActiveList*. Since flow j is in the *ActiveList*, the ERR scheduler will visit flow j, before flow i receives service again. Thus, in between any two consecutive service opportunities given to flow i, flow j receives exactly one service opportunity. Hence, if n_i and n_j denote the total round robin opportunities received by flows i and j respectively in the time interval (t_1, t_2) then, $|n_i - n_j| \leq 1$.

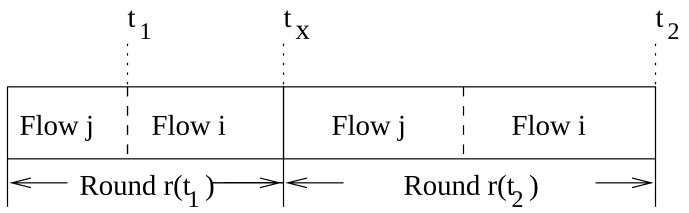


Figure 3.1: Explanation for $|r_i - r_j| \leq 1$

Let $r(t)$ denote the round in progress at time instant t. Also note that the time instant t_1 may be such that the service opportunity received by one of the two flows in round $r(t_1)$ may not be a part of interval (t_1, t_2). Thus, the first time that the scheduler visits this flow in the interval under consideration would be in the round following $r(t_1)$. Consequently, if r_i and r_j, denote the rounds in which flows i and j receive service for the first time in the interval (t_1, t_2) respectively, then $|r_i - r_j| \leq 1$. This is illustrated in Figure 3.1

Since t_1 and t_2 both belong to set $\mathbf{T}_s$, assume that the time instant t_1 coincides with the time when the scheduler has finished serving flow j in round $r(t_1)$. Hence, the scheduler now visits flow i. Let the time instant t_X mark the end of the service opportunity of flow i and the round $r(t_1)$. Thus, in the time interval under consideration, the ERR scheduler visits flow i for the first time in the round $r(t_1)$ and flow j in the round $r(t_2)$.

Without loss of generality, we can assume that in the interval (t_1, t_2), flow i starts receiving service before flow j. Thus,

$$\begin{aligned} & r_j \leq r_i + 1, \\ \text{and} \quad & n_i \leq n_j + 1 \end{aligned} \tag{3.5}$$

From Theorem 3.2.1, for flow i

$$Sent_i(t_1, t_2) \leq n_i + \sum_{k=r_i-1}^{r_i+n_i-2} MaxSC(k) + (m-1) \tag{3.6}$$

For flow j,

$$n_j + \sum_{k=r_j-1}^{r_j+n_j-2} MaxSC(k) - (m-1) \leq Sent_j(t_1, t_2) \tag{3.7}$$

Combining Equations (3.6) and (3.7), and using (3.5), we get,

$$\begin{aligned} Sent_i(t_1, t_2) - Sent_j(t_1, t_2) \quad \leq \quad & 1 + \sum_{k=r_i-1}^{r_i+n_i-2} MaxSC(k) \\ & - \sum_{k=r_j-1}^{r_j+n_j-2} MaxSC(k) + 2(m-1) \end{aligned} \tag{3.8}$$

Let us now consider the quantity D given by,

$$D = \sum_{k=r_i-1}^{r_i+n_i-2} MaxSC(k) - \sum_{k=r_j-1}^{r_j+n_j-2} MaxSC(k)$$

We now compute D for each of the four possible cases.

Case 1 ($r_i = r_j$, $n_i = n_j$):

$$D = 0$$

Case 2 ($r_i = r_j$, $n_i = n_j + 1$):

$$D = MaxSC(r_i + n_i - 2)$$

Case 3 ($r_i = r_j - 1$, $n_i = n_j$):

$$D = MaxSC(r_i - 1) - MaxSC(r_i + n_i - 1)$$

Case 4 ($r_i = r_j - 1$, $n_i = n_j + 1$):

$$D = MaxSC(r_i - 1)$$

Using Corollary 3.2.1, in each of the above four cases, $D < m$. Substituting in (3.8), the statement of the theorem is proved. ■

It can be easily verified that Theorem 3.2.2 can also be proved for the weighted ERR scheduler using Equations (2.6) and (2.5) in place of Equation (2.3) and (2.2) in the proof above.

In comparison to a relative fairness bound of $3m$ for ERR, both DRR and SRR have a relative fairness bound of $M + 2m$, where M is the size of the largest packet that may *potentially* arrive during the lifetime of the execution of the scheduling discipline. Recall that m is the size of the largest packet that *actually* arrives during the execution of the scheduler. In most networks, including the Internet, the vast majority of the packets in the traffic are of much smaller size than the maximum possible size of a packet [44, 45]. The value of m in the expression for the relative fairness, especially over short intervals of time, is likely to be much smaller than M. The fairness achieved by ERR, thus, is always equal to or better than that achieved by DRR or SRR.

3.3 Latency Rate Servers

In this section we present a brief overview of the concept of *Latency Rate* servers, first introduced in [38]. The theory of *Latency Rate* ($\mathcal{LR}$) servers provides a means to describe the worst-case behavior of a broad range of scheduling algorithms in a simple manner. The two key parameters that determine the behavior of a $\mathcal{LR}$ server are the *latency* and the *reserved rate* of each flow. The latency of a $\mathcal{LR}$ server is a measure of the cumulative time that a flow has to wait until it begins receiving service at its guaranteed rate. The latency of a particular scheduling algorithm may depend on a number of factors such as internal parameters of the scheduling discipline, the reserved rates of the other flows multiplexed on the same output link and the transmission rate of the flow on the output link. It has been shown in [38] that several well-known scheduling disciplines such as Weighted Fair Queuing (WFQ), Self-Clocked Fair Queuing (SCFQ), Virtual Clock and Deficit Round Robin (DRR) belong to the class of $\mathcal{LR}$ servers.

We first present some definitions and notations which will be useful in understanding the concept of $\mathcal{LR}$ servers.

Definition 3.3.1 *A* system busy period *is defined as the maximal time interval during which the server is continuously transmitting packets.*

Definition 3.3.2 *We define a flow as* active *during an interval of time, if at all instants of time during this interval, it has at least one packet awaiting service or being served.*

We now define define the notion a *busy period*, an essential component of the concept of $\mathcal{LR}$ servers.

Definition 3.3.3 *A* busy period *of a flow is defined as the maximal time interval during which the flow is active if it served at exactly its reserved rate.*

Let ρ_i be the reserved rate for flow i. Also let $Arrived_i(t_1, t_2)$ denote the total number of bits of flow i that arrive at the scheduler during the time interval (t_1, t_2). Consider an interval of time (τ_1, τ_2) which represents a busy period for flow i. Then for any time interval (τ_1, t) such that $t \in (\tau_1, \tau_2)$, the number of bits that arrive during this interval is greater than or equal to the number of bits that would exit the scheduler if the flow received service at its reserved rate, ρ_i. In other words, for all $t \in (\tau_1, \tau_2)$,

$$Arrived_i(\tau_1, t) \geq \rho_i(t - \tau_1)$$

A graph of $Arrived_i(\tau, t)$ against time is plotted in Figure 3.2. Figure 3.2 illustrates two busy periods, (t_1, t_2) and (t_3, t_4) for flow i. It is important to understand the basic difference between a session busy period and a session active period. The definition of the busy period supposes that flow i is served at the constant reserved rate, and therefore, depends only on the reserved rate of the flow and the packet arrival pattern of the flow. An active period of a flow, however, reflects the actual behavior of the scheduler where the

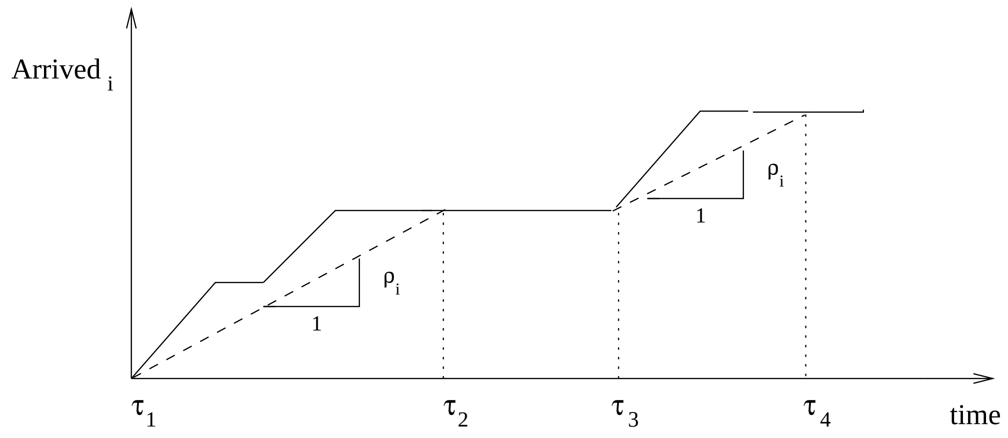


Figure 3.2: Two busy periods for flow i

instantaneous service offered to flow i varies according to the number of active flows. If during a busy period of flow i, the instantaneous service rate offered to flow i is greater than the allocated rate, then the flow may cease to be active. Thus, a busy period of a flow may include multiple active periods for that flow. The start of a busy period of a flow is always caused by the arrival of a packet belonging to the flow.

Note that, when the same traffic distribution is applied to two different schedulers with identical reserved rates, the ensuing active periods of the flows can be quite different. This makes it difficult to make use of active periods to analyze a broad class of schedulers. On the other hand, the busy period of a flow depends only on the arrival rate of the flow and its reserved rate. Therefore, the busy period can be used as an invariant in the analysis of different schedulers. It is because of this important property that the definition of an $\mathcal{LR}$ server is based on the service received by a flow during a busy period.

The following definitions lead to a formal notion of latency in the case of guaranteed-rate servers. The reader is referred to [38] for a more detailed discussion.

Definition 3.3.4 *Define $Sent_i(t_1, t_2)$ as the amount of service received by flow i during the* *interval* (t_1, t_2).

Definition 3.3.5 *Let time instant* α_i *represent the start of a certain busy period for flow* i. *Let* $t > \alpha_i$ *be such that the flow is continuously busy during the time interval* (α_i, t). *Define* $S_i(\alpha_i, t)$ *as the number of bits belonging to packets in flow* i *that arrive after time* α_i *and are scheduled during the time interval* (α_i, t).

Note that, $Sent_i(\alpha_i, t)$ is not necessarily equal to $S_i(\alpha_i, t)$. This is because, during this interval of time, the scheduler may still be serving packets that arrived during a previous busy period. $S_i(\alpha_i, t)$, therefore, is not necessarily the same as the total number of bits scheduled from flow i in this interval. We are now prepared to present the definition of latency in $\mathcal{L}R$ servers.

Definition 3.3.6 *The latency of a flow is defined as the minimum non-negative constant* Θ_i *that satisfies the following for all possible busy periods of the flow,*

$$S_i(\alpha_i, t) \geq max\{0, \rho_i(t - \alpha_i - \Theta_i)\} \tag{3.9}$$

As defined in [38], a scheduler which satisfies Equation (3.9) for some non-negative constant value of Θ_i is said to belong to the class of *Latency Rate* ($\mathcal{LR}$) servers. The above definition captures the fact that the latency of a guaranteed-rate scheduler should not merely be the time it takes for the first packet of a flow to get scheduled, but should be a measure of the cumulative time that a flow has to wait until it begins receiving service at its guaranteed rate. A graph of $S_i(\alpha_i, t)$ against time is plotted in Figure 3.3. The RHS of the above equation defines an envelop which bounds the minimum service received by a flow i during the busy period (α_i, t). The dashed line in Figure 3.3 corresponds to this envelop. For a particular scheduling algorithm several parameters such as its transmission rate on the output link, the number of the other flows sharing the link and their reserved rate may influence the latency. It has been proved in [38] that the maximum end-to-end delay

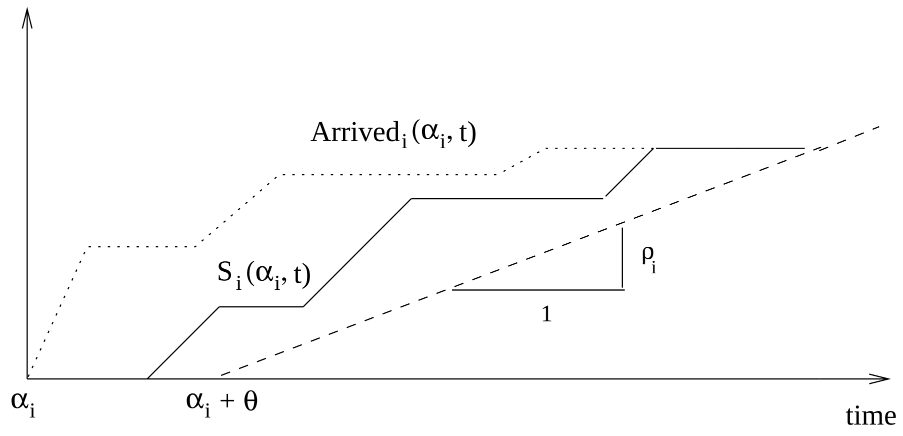


Figure 3.3: An example of the behavior of an $\mathcal{LR}$ server

experienced by a packet in a network of schedulers can be calculated from only the latencies of the individual schedulers on the path of the connection and the traffic parameters on the connection that generated the packet. Since the end-to-end delay increases directly in proportion to the latency of the schedulers, the model highlights the significance of using low-latency schedulers for achieving low end-to-end delays.

3.4 Latency Analysis

The analysis of the latency of ERR is facilitated by the following result, stated below as Lemma 3.4.1 and proved in [38]. This result allows one to obtain a bound on the latency achieved by a flow as given by Definition 3.3.6 by considering only the active periods of a flow.

Lemma 3.4.1 *Let τ_i be an instant of time when flow i becomes active. Let $t > \tau_i$ be some instant of time such that the flow is continuously active during the time interval (τ_i, t). Let*

Θ'_i be the smallest non-negative number such that the following is satisfied for all t.

$$Sent_i(\tau_i, t) \geq max\{0, \rho_i(t - \tau_i - \Theta'_i)\} \tag{3.10}$$

Even though (τ_i, t) may not be a continuously busy period for flow i, the latency as defined by Definition 3.3.6 is bounded by Θ'_i.

We now proceed to derive a bound on the latency of the ERR scheduler, using the lemma above. Note that, the instant of time when a flow i becomes active, τ_i, may or may not coincide with the start of the round robin service opportunity of some other flow. In the following, we prove that a tight upper bound on the latency of the ERR scheduler can be obtained by considering τ_i belonging to only a subset of all possible time instants. This subset is the set of all time instants that coincide with the beginning or the end of the service opportunity of flows.

Definition 3.4.1 *Define **T** as the set of all time instants, during an execution of the ERR algorithm, at which the scheduler ends serving one flow and begins serving another. Define $\mathbf{T}_i$ as the set of all time instants at which the scheduler begins serving flow i. Note that, the set **T** is the union of $\mathbf{T}_i$ for all flows i, that are served during the execution of the scheduler.*

Lemma 3.4.2 *The latency experienced by flow i in an ERR scheduler will reach its upper bound, Θ'_i, only if the time instant, τ_i, at which flow i becomes active, belongs to the set **T**.*

Proof: Assume that flow i becomes active at time instant τ_i. Let (t_1, t_2), $t_1 \leq \tau_i < t_2$ be the time interval during which some flow $j \neq i$ receives its round robin service opportunity. Consider the case when τ_i does not coincide with the start of the service opportunity of flow i, i.e., $\tau_i > t_1$. Now, the time interval (t_1, τ_i), which is a part of the round robin service opportunity of flow j, will not contribute to the latency experienced by flow i. On the other hand, consider the case when τ_i coincides with t_1, the start of the service opportunity of flow j. Now, the time for which flow i has to wait before receiving any service will include the entire time interval (t_1, t_2) during which flow j receives its round robin service opportunity. Clearly, the latency experienced by flow i is always greater when τ_i coincides with the start of the service opportunity of some other flow. The statement of the lemma follows from this observation. ■

From Lemma 3.4.2, in deriving an upper bound on the latency experienced by flow i, therefore, one needs to only consider τ_i such that $\tau_i \in \mathbf{T}$. The following lemma further limits the cases we need to consider in deriving the upper bound. Note that, in proving the upper bound, we need to only consider the intervals of time over which Equation (3.10) is an equality. In fact, the upper bound on the latency is achieved at time instants t when $Sent_i(\tau_i, t) = \rho_i(t - \tau_i - \Theta'_i)$. The following lemma provides a simple condition on t in order to achieve the equality in Equation (3.10).

Lemma 3.4.3 *If flow i becomes active at time instant τ_i, then there exists some $t \in \mathbf{T}_i$ such that the flow remains active during the interval (τ_i, t), and*

$$Sent_i(\tau_i, t) = \rho_i(t - \tau_i - \Theta'_i)$$

Proof: Note that, if $Sent_i(\tau_i, t) = \rho_i(t - \tau_i - \Theta'_i)$, then the flow experiences the worst-case latency at time instant t. Consider any two consecutive time instants t_1 and t_2 which both belong to $\mathbf{T}_i$. Consider an instant of time t, $t_1 < t < t_2$.

Case 1: Let t be such that flow i is receiving service at time instant t.

During the interval (t_1, t), the amount of service received by the flow is $r(t - t_1)$, where r is the rate of the link. Clearly, during this time, the flow is receiving service at the guaranteed rate or higher. Therefore, the worst-case latency experienced by the flow until any time in the interval (t_1, t) is no worse than that experienced by it until time $t_1 \in \mathbf{T}_i$.

Case 2: Let t be such that some flow other than i is receiving service at time instant t.

During the interval (t, t_2), flow i receives no service at all. Therefore, the latency experienced by the flow increases after time t but only until time t_2. Thus, the worst-case latency experienced by the flow until any time in the interval (t, t_2) is no worse than that experienced by it until time $t_2 \in \mathbf{T}_i$.

The above two cases illustrate that the worst-case latency experienced by a flow during an interval (t_1, t_2) is equal to the latency experienced by the flow until either time t_1 or time t_2. Extrapolating to all intervals between consecutive time instants that belong to $\mathbf{T}_i$, the statement of the lemma is proved. ■

Theorem 3.4.1 *The ERR scheduler belongs to the class of $\mathcal{LR}$ servers, with an upper bound on the latency Θ_i for flow i given by,*

$$\Theta_i \leq \frac{(W - w_i)m + (n-1)(m-1)}{r} \tag{3.11}$$

where n is the total number of active flows, W is the sum of the weights of all the flows and r is the transmission rate of the output link.

Proof: From Lemma 3.4.1, we know that the latency of the ERR scheduler as defined by Equation 3.3.6 is bounded by Θ'_i. Hence, we will prove the theorem by showing that,

$$\Theta'_i \leq \frac{((W - w_i)m + (n-1)(m-1))}{r}$$

Let time instant τ_i represent the time instant at which flow i becomes active. To prove the statement of the theorem we must consider a time interval (τ_i, t) where $t > \tau_i$, during which flow i is continuously active. We first obtain the lower bound on the total service received by flow i during the time interval under consideration. Then we express the lower bound in the form of Equation (3.10) to derive the latency bound. By the statement of Lemma 3.4.2, this upper bound on the latency is reached only if the time instant at which flow i becomes active, τ_i, belongs to $\mathbf{T}$. Therefore, in seeking the upper bound on the latency, we may assume that flow i becomes active at exactly the instant that some other flow begins receiving service. Let τ_i^k be the time instant marking the start of the k-th service opportunity of flow i after it becomes active at time instant τ_i. Note that, τ_i^k belongs to

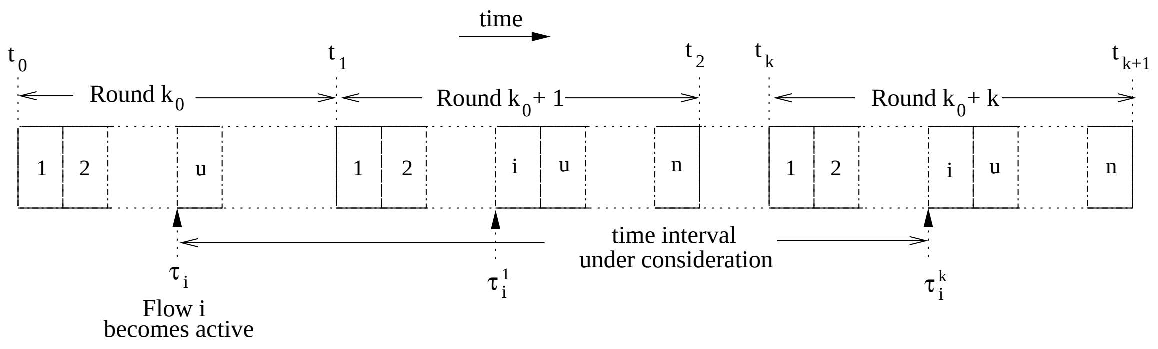


Figure 3.4: An illustration of the time interval under consideration for the analysis of the latency bound of ERR

$\mathbf{T}_i$. Therefore, from Lemma 3.4.3, in order to determine the latency bound as defined by Equation 3.10, one needs to only consider intervals of time (τ_i, τ_i^k) for all k. Figure 3.4 illustrates the time interval under consideration for a given k.

The first step toward analyzing the latency bound involves determining the upper bound on the time interval under consideration. Note that, the time instant τ_i may or may not coincide with the end of a round and the start of the subsequent round. Let k_0 be the round which is in progress at time instant τ_i or which ends exactly at time instant τ_i. Let the time instant t_h mark the end of round $(k_0 + h - 1)$ and the start of the subsequent round. For any flow j during a round s in the interval under consideration, using (2.1) and (2.6), we have,

$$Sent_j(s) = w_j(1 + MaxSC(s-1)) + SC_j(s) - SC_j(s-1) \tag{3.12}$$

As illustrated in Figure 3.4, assume that the time instant when flow i becomes active coincides with the time instant when some flow j is about to start its service opportunity during the k_0-th round. Let G_a denote the set of flows which receive service during the time interval (τ_i, t_1), i.e., *after* flow i becomes active and during round k_0. Similarly, let

G_b denote the set of flows which are served by the ERR scheduler during the time interval (t_0, τ_i), i.e., *before* flow i becomes active and during round k_0. Note that, flow i is not included in either of these two sets since flow i will receive its first service opportunity only in the (k_0+1)-th round. If the time instant τ_i coincides with the end of a round, then the set G_a will be empty and all the $(n-1)$ flows will belong to the set G_b. Note that, the union of the sets G_a, G_b and flow i results in the set of n active flows. In other words,

$$G_a \cup G_b \cup \{i\} = \{n\} \tag{3.13}$$

Consider the time interval (τ_i, τ_i^k) during which flow i receives k round robin service opportunities. This time interval can be split into three sub-intervals:

1. (τ_i, t_1): This sub-interval includes the part of the k_0-th round during which all the flows belonging to the set G_a will be served by the ERR scheduler. Summing Equation (3.12) over all these flows,

$$t_1 - \tau_i \leq \frac{1}{r} \sum_{j \in G_a} \{w_j(1 + MaxSC(k_0 - 1)) + SC_j(k_0) - SC_j(k_0 - 1)\} \tag{3.14}$$

2. (t_1, t_k): This sub-interval includes $k-1$ rounds of execution of the ERR scheduler starting at round (k_0+1). Consider the time interval (t_h, t_{h+1}) when round (k_0+h) is in progress. Summing Equation (3.12) over all n flows,

$$\begin{aligned} t_{h+1} - t_h \leq & \frac{W}{r}(1 + MaxSC(k_0 + h - 1)) \\ & + \frac{1}{r}\sum_{j=1}^{n}\{SC_j(k_0+h) - SC_j(k_0+h-1)\} \end{aligned}$$

   Summing the above over $(k-1)$ rounds beginning with round (k_0+1),

$$\begin{aligned} t_k - t_1 \quad \leq \quad & \frac{W}{r}(k-1) + \frac{W}{r}\sum_{h=1}^{k-1} MaxSC(k_0 + h - 1) \\ & + \frac{1}{r}\sum_{j=1}^{n}\{SC_j(k_0+k-1) - SC_j(k_0)\} \end{aligned} \tag{3.15}$$

3. (t_k, τ_i^k): This sub-interval includes the part of the $(k_0 + k)$-th round during which all the flows belonging to the set G_b will be served by the ERR scheduler. Summing Equation (3.12) over all these flows,

$$\begin{aligned}\tau_i^k - t_k &\leq \frac{1}{r}\sum_{j \in G_b} w_j(1 + MaxSC(k_0 + k - 1)) \\ &\quad + \frac{1}{r}\sum_{j \in G_b}\{SC_j(k_0 + k) - SC_j(k_0 + k - 1)\} \qquad (3.16)\end{aligned}$$

Combining Equations (3.14), (3.15) and (3.16) we have,

$$\begin{aligned}\tau_i^k - \tau_i &\leq \frac{1}{r}\sum_{j \in G_a} w_j(1 + MaxSC(k_0 - 1)) + \frac{1}{r}\sum_{j \in G_b} w_j(1 + MaxSC(k_0 + k - 1)) \\ &\quad + \frac{W}{r}(k - 1) + \frac{W}{r}\sum_{h=1}^{k-1} MaxSC(k_0 + h - 1) \\ &\quad + \frac{1}{r}\sum_{j \in G_a}\{SC_j(k_0) - SC_j(k_0 - 1)\} \\ &\quad + \frac{1}{r}\sum_{j \in G_b}\{SC_j(k_0 + k) - SC_j(k_0 + k - 1)\} \\ &\quad + \frac{1}{r}\sum_{j=1}^{n}\{SC_j(k_0 + k - 1) - SC_j(k_0)\} \qquad (3.17)\end{aligned}$$

Simplifying Equation (3.17) using Equation (3.13) results in,

$$\begin{aligned}\tau_i^k - \tau_i &\leq \frac{1}{r}\sum_{j \in G_a} w_j(1 + MaxSC(k_0 - 1)) + \frac{1}{r}\sum_{j \in G_b} w_j(1 + MaxSC(k_0 + k - 1)) \\ &\quad + \frac{W}{r}(k - 1) + \frac{W}{r}\sum_{h=1}^{k-1} MaxSC(k_0 + h - 1) \\ &\quad + \frac{1}{r}\sum_{j \in G_a}\{SC_j(k_0 + k - 1) - SC_j(k_0 - 1)\} \\ &\quad + \frac{1}{r}\sum_{j \in G_b}\{SC_j(k_0 + k) - SC_j(k_0)\} \\ &\quad + \frac{1}{r}\{SC_i(k_0 + k - 1) - SC_i(k_0)\}\end{aligned}$$

Using the bounds on the surplus count from Equation (2.1) in the above equation, we have,

$$\begin{aligned}\tau_i^k - \tau_i \;\; \leq \;\; & \frac{W}{r}(k-1) + \frac{W-w_i}{r} + \frac{1}{r}\sum_{j\in G_a} w_j MaxSC(k_0-1) \\ & + \frac{W}{r}\sum_{h=1}^{k-1} MaxSC(k_0+h-1) + \frac{1}{r}\sum_{j\in G_b} w_j MaxSC(k_0+k-1) \\ & + \frac{(n-1)(m-1)}{r} + \frac{1}{r}SC_i(k_0+k-1) \end{aligned} \tag{3.18}$$

Solving for $(k-1)$,

$$\begin{aligned}(k-1) \;\; \geq \;\; & (\tau_i^k - \tau_i)\frac{r}{W} - \frac{W-w_i}{W} - \frac{1}{W}\sum_{j\in G_a} w_j MaxSC(k_0-1) \\ & - \sum_{h=1}^{k-1} MaxSC(k_0+h-1) - \frac{1}{W}\sum_{j\in G_b} w_j (MaxSC(k_0+k-1) \\ & - \frac{(n-1)}{W}(m-1) - \frac{1}{W}SC_i(k_0+k-1) \end{aligned} \tag{3.19}$$

Note that, the total data transmitted by flow i during the time interval under consideration, (τ_i, τ_i^k) can be expressed as the following summation,

$$Sent_i(\tau_i, \tau_i^k) = \sum_{z=k_0+1}^{k_0+k-1} Sent_i(z)$$

Substituting Equation (3.12) in the above equation,

$$Sent_i(\tau_i, \tau_i^k) = w_i(k-1) + w_i\sum_{h=1}^{k} MaxSC(k_0+h-1) + SC_i(k_0+k-1) - SC_i(k_0)$$

Note that, in ERR the surplus count of a newly active flow is initialized to zero. As a result, since flow i becomes active at time instant τ_i, $SC_i(k_0)$ is equal to zero. Substituting this in the above equation, we get,

$$Sent_i(\tau_i, \tau_i^k) = w_i(k-1) + w_i\sum_{h=1}^{k} MaxSC(k_0+h-1) + SC_i(k_0+k-1) \tag{3.20}$$

Using Equation (3.19) to substitute for $(k-1)$ in Equation (3.20), we have,

$$\begin{aligned}
Sent_i(\tau_i,\tau_i^k) \;\geq\;& \frac{w_i r}{W}(\tau_i^k-\tau_i) - \frac{w_i}{W}(W-w_i) - \frac{wi}{W}\sum_{j\in G_a} w_j MaxSC(k_0-1) \\
&- w_i\sum_{h=1}^{k} MaxSC(k_0+h-1) - \frac{w_i}{W}\sum_{j\in G_b} w_j MaxSC(k_0+k-1) \\
&- \frac{w_i}{W}(n-1)(m-1) - \frac{w_i}{W}SC_i(k_0+k-1) \\
&+ w_i\sum_{h=1}^{k} MaxSC(k_0+h-1) + SC_i(k_0+k-1)
\end{aligned}$$

Simplifying the above equation,

$$\begin{aligned}
Sent_i(\tau_i,\tau_i^k) \;\geq\;& \frac{w_i r}{W}\Bigg((\tau_i^k-\tau_i) - \frac{1}{r}(W-w_i) - \frac{1}{r}\sum_{j\in G_a} w_j MaxSC(k_0-1) \\
&- \frac{1}{r}\sum_{j\in G_b} w_j MaxSC(k_0+k-1) - \frac{1}{r}(n-1)(m-1)\Bigg) \\
&- SC_i(k_0+k-1)\left(\frac{w_i}{W}-1\right)
\end{aligned} \tag{3.21}$$

Using Equation (3.18), it can be easily verified that,

$$\begin{aligned}
&(\tau_i^k-\tau_i) - \frac{1}{r}(W-w_i) - \frac{1}{r}\sum_{j\in G_a} w_j MaxSC(k_0-1) \\
&- \frac{1}{r}\sum_{j\in G_b} w_j MaxSC(k_0+k-1) - \frac{1}{r}(n-1)(m-1) > 0
\end{aligned} \tag{3.22}$$

Now, since the reserved rates are proportional to the weights assigned to the flows as given by Equation (2.4), and since the sum of the reserved rates is no more than the link rate r, we have,

$$\rho_i \leq \frac{w_i}{W} r \tag{3.23}$$

Using Equations (3.22) and (3.23) in Equation (3.21), we get,

$$\begin{aligned} Sent_i(\tau_i, \tau_i^k) \quad \geq \quad & max\Bigg\{0, \rho_i\Bigg((\tau_i^k - \tau_i) - \frac{1}{r}(W - w_i) - \frac{1}{r}\sum_{j \in G_a} w_j MaxSC(k_0 - 1) \\ & - \frac{1}{r}\sum_{j \in G_b} w_j MaxSC(k_0 + k - 1) - \frac{1}{r}(n-1)(m-1)\Bigg) \\ & - SC_i(k_0 + k - 1)\left(\frac{w_i}{W} - 1\right)\Bigg\} \end{aligned} \tag{3.24}$$

Comparing the Equation (3.24) with Equation (3.10) and using Lemma 3.4.1, the latency bound is given by,

$$\begin{aligned} \Theta_i \quad \leq \quad & \frac{1}{r}(W - w_i) + \frac{1}{r}\sum_{j \in G_a} w_j MaxSC(k_0 - 1) \\ & + \frac{1}{r}\sum_{j \in G_b} w_j MaxSC(k_0 + k - 1) + \frac{1}{r}(n-1)(m-1) \\ & + SC_i(k_0 + k - 1)\left(\frac{w_i}{W} - 1\right) \end{aligned} \tag{3.25}$$

From the above equation it is readily seen that the upper bound on the latency will be reached under the following conditions:

- $MaxSC(k_0 - 1)$ for each flow j in the set G_a is equal to it upper bound, $(m - 1)$. (From Corollary 3.2.1)

- $MaxSC(k_0 + k - 1)$ for each flow j in the set G_b is equal to it upper bound, $(m - 1)$. (From Corollary 3.2.1)

- $SC_i(k_0 + k - 1)$ is equal to its lowest possible value, 0. (From Equation (2.1))

Substituting these bounds in Equation (3.25), we get,

$$\Theta_i \leq \frac{(W - w_i)m + (n-1)(m-1)}{r}$$

As discussed earlier, based on Lemmas 3.4.2 and 3.4.3, flow i will experience its worst latency during an interval (τ_i, τ_i^k) for some k. Therefore, from (3.20) and Lemma 3.4.1, the statement of the theorem is proved. ■

We now proceed to show that the latency bound given by Theorem 3.4.1 is tight by illustrating a case where the bound is actually met. Assume that a flow i becomes active at time instant τ_i, which also coincides with the end of a certain round k_0 and the start of the round $(k_0 + 1)$. Since other flows in the *ActiveList* will be served first, flow i becomes backlogged instantly. Assume that for any time instant t, $t \geq \tau_i$, a total of n flows, including flow i, are active. Also, assume that the summation of the reserved rates of all the n flows equals the output link transmission rate, r. Hence, $\rho_i = \frac{w_i}{W} r$. Since flow i became active at time τ_i, its surplus count at the start of round $(k_0 + 1)$ is 0. Let the surplus count of all the other flows at the start of round $(k_0 + 1)$ be equal to 0. Assume that, a flow p which is not active after time α_i and hence is not included in the n flows, was active during the k_0-th round. Also assume that flow p exceeded its allowance by $(m - 1)$ during its service opportunity in round k_0, leading to a value of $MaxSC(k_0)$ equal to $(m - 1)$. From Equations (2.1) and (3.12) and Corollary 3.2.1, any given flow j can transmit a maximum of $w_j m + (m - 1)$ bits during a round robin service opportunity. In the worst case, before flow i is served by the ERR scheduler, each of the other $(n - 1)$ flows will receive this maximum service. Hence, the cumulative delay until flow i receives service is given by,

$$\begin{aligned} D &= \frac{(\sum_{j \neq i} w_j)m + (n-1)(m-1)}{r} \\ &= \frac{(W - w_i)m + (n-1)(m-1)}{r} \end{aligned}$$

Noting that $S_i(\alpha_i, \alpha_i + D)$ equals zero, it is readily verified that the bound is exactly met at time $t = \alpha_i + D$.

Chapter 4. Prioritized Elastic Round Robin

As explained in Chapter 1, in the frame-based schemes, the scheduler provides service opportunities to the backlogged flows in a particular order and, during each service opportunity, the intent is to provide the flow an amount of service proportional to its fair share of the bandwidth. Examples of such schedulers are Deficit Round Robin (DRR) [21], Surplus Round Robin (SRR) [34–36] and Elastic Round Robin (ERR) [46]. The frame-based schedulers do not maintain a global virtual time function and also do not require any sorting among the packets available for transmission. This reduces the implementation complexity of frame-based scheduling disciplines to $O(1)$, making them attractive for implementation in routers and, especially so, in hardware switches. However, these frame-based schedulers suffer from the following disadvantages:

- *High Start-Up Latency:* The above frame-based schedulers operate in a round-robin fashion, with each active flow receiving exactly one opportunity to transmit in each round. When a new flow becomes active, it has to wait until all other previously active flows receive their service opportunity before it can receive service from the scheduler. With large numbers of flows, this time period can be very large, especially in comparison to sorted-priority schedulers such as WFQ and WF^{2}Q.

- *Bursty output:* Each flow is served over a continuous time interval during its round robin service opportunity leading to a bursty packet stream at the output of the scheduler for any given flow. This is not an ideal situation for real-time multimedia traffic since even smooth flows are rendered bursty as they exit the scheduler.

- *Delayed correction of unfairness:* If a flow receives very little service in a particular round, it is compensated with proportionately more service in the next round. While this disadvantaged flow waits for its compensation in the next round, other flows which have already received more service than their fair share in the previous round continue to receive yet more service before the disadvantaged flow receives its opportunity.

- *Compounded Jitter:* When a flow's arrival pattern at the scheduler has high jitter, it can frequently happen that the flow runs out of packets even before it has received its fair share of service during its service opportunity. At this point, the scheduler moves on to serve other currently active flows in a round-robin fashion. Our flow with high jitter will receive its next opportunity only in the next round after all the other active flows have completed their transmissions. This further increases the jitter in the output of the scheduler since a delayed packet that just misses its service opportunity in a certain round ends up experiencing significant additional delay because of having to wait for all the other active flows to complete their transmissions.

These weaknesses of frame-based schedulers discussed above are caused by the same features that are common to these schedulers:

1. The round robin nature of the service order.

2. Each flow receives its entire share of service in the round at once in one service opportunity.

Overcoming these weaknesses while preserving the low complexity of frame-based schedulers forms the primary motivation behind this chapter.

At least a few proposals have been made in the last few years to overcome the limitations of frame-based schedulers discussed above. The Nested-DRR scheduler proposed in [40] tries to eliminate some of these limitations of the DRR scheduler [21]. For each flow i, the DRR scheduler maintains a *quantum*, Q_i, which represents the ideal service that the flow should receive in each round of service. If the entire quantum is not used in a given round, the deficit is recorded and used to compensate the flow in the next round.

Nested-DRR splits each DRR round, referred to as an *outer round*, into one or more smaller *inner rounds* and executes a modified version of the DRR algorithm within each of these inner rounds. If Q_{min} is the quantum assigned to the flow with the lowest reserved rate, the Nested-DRR scheduler tries to serve Q_{min} worth of data from each flow during each inner round. During an outer round, a flow is considered to be eligible for service in as many inner rounds as are required by the scheduler to exhaust its quantum. The Nested-DRR scheduler, just like the DRR scheduler, has a per-packet work complexity of $O(1)$ as long as the largest packet that may potentially arrive in flow i is smaller than Q_i.

The technique of nesting smaller rounds within a round as in the Nested-DRR scheduler may be adapted for use with other $O(1)$ schedulers such as ERR and SRR. The Nested-DRR scheduler results in a significant improvement in the latency in comparison to DRR, but only in those cases in which there is a significant difference between the quanta assigned to the flows. If all flows are of the same weight, the behavior of the Nested-DRR scheduler is identical to that of DRR. Further, the improvement gained in fairness or latency is again limited by the fact that, within each inner round, the nested scheduler still serves the active flows in a round robin manner.

The Pre-order DRR algorithm proposed in [42] combines the nesting technique explained above with a scaled down version of the sorting of packets used in the sorted-priority schedulers and thus, succeeds in overcoming some of the drawbacks of the DRR scheduler. The Pre-order DRR scheduler adds a limited number of priority queues in which packets wait before being actually scheduled for transmission. The packets that are transmitted in a DRR round from each flow are now classified into these queues depending on the percentage of the flow's quantum that each packet will utilize following its transmission. Thus, the transmission sequence of the packets in a round in DRR is *reordered* allowing certain packets to receive priority over others, resulting in an improvement in the latency and fairness properties. The rest of the chapter is organized as follows. Section 4.1 highlights the important contributions of the our solution. In Section 4.2 we present a detailed description of our new scheme, Prioritized Elastic Round Robin (PERR) which aims at overcoming the drawbacks of ERR.

4.1 Contributions

In this chapter, we propose a novel packet scheduler, *Prioritized Elastic Round Robin (PERR)*, which exhibits improved fairness and latency characteristics in comparison to other known schedulers of equivalent complexity, including Pre-order DRR discussed earlier. The total service received by a flow in a round in PERR is identical to the service received by the flow in the corresponding round in ERR. However, in PERR, this service received by a flow is split into several parts over the course of the round. The PERR scheduler, borrowing the principle used in Pre-order DRR, re-orders the transmission sequence within each round of the ERR scheduler. The transmission sequence of the packets in a round is reordered to allow the flows that have received less service in the previous round to gain precedence over the other flows in the current round. The exact manner in which the transmission sequence is re-ordered depends on a certain per-flow state that indicates how far ahead or behind a flow is in consuming its share of bandwidth. As in the Pre-order DRR, the scheduler maintains a limited number, p, of priority queues which serve to implement the re-ordered transmission sequence.

The PERR scheduler achieves a significant improvement in the fairness, latency, and delay jitter characteristics and addresses several of the weaknesses (such as burstiness of output traffic) of round-robin schedulers. In addition to its superior fairness and latency characteristics, the PERR scheduler holds several advantages over other schedulers, such as Pre-order DRR, that have attempted to address these weakness of round-robin schedulers. For example, at the start of a round, the Pre-order DRR scheduler has to classify all the packets that will be transmitted by the active flows in that round into the priority queues prior to the beginning of the transmission of the packets. On the contrary, the PERR scheduler simply has to classify the flows (as opposed to packets) present in the *ActiveList* into its priority queues before the start of the round. This reduces the buffering requirements and the delay through the finite state machines managing the transmission scheduling, since classifying all the packets in the round into priority queues requires considerably more time than simply sorting the flow identifiers. In addition, in comparison to Pre-order DRR, this allows a more dynamic re-ordering of the transmission sequence based on the latest state of the flows, leading to improved fairness at all instants of time.

As shown in [46, 47], the ERR scheduler has a couple of important advantages in comparison to DRR. Since PERR is based on the ERR scheduler, PERR inherits some of these advantages as well. For example, unlike DRR or Pre-order DRR, the PERR scheduler does not require the knowledge of the transmission time of each packet prior to the scheduling operation. As a result, the scheduler can be used in other networks such as wormhole networks, where the transmission time of a packet depends not only on the size of the packet but also the downstream congestion. For the same reasons, PERR—but not DRR or Pre-order DRR—may be used in ATM networks transmitting IP packets over AAL5, where the end of the packet is not known until the arrival of the last ATM cell corresponding to the packet.

4.2 Algorithm Description

The basic principle of the PERR scheduler involves modifying the transmission sequence of the packets that are scheduled within each round in ERR. This re-ordering is performed upon the transmission of each packet, and is carried out based on the amount of each active flow's allowance for the round that is actually consumed until the instant that the re-ordering is executed. This allows each flow to utilize its allowance in pieces over the duration of each round. The reordering is implemented through the use of priority queues, which are nothing but linked lists of flow identifiers. The scheduler transmits packets from the flows in the highest priority queue first, and begins serving a flow in another priority queue only after all higher priority queues are empty. The core aspect of the PERR algorithm is how it manages these priority queues and rearranges flows amongst these priority queues.

In this section, we present a detailed description of the PERR algorithm. We begin our discussion by introducing certain important definitions that are essential to understanding the rationale behind the design of the PERR scheduler.

As in ERR, let $Sent_i(s)$ represent the total service received by flow i in the s-th round of service. Assume that a total of y packets are transmitted from flow i in round s. The packets are labeled as 1, 2, $\ldots$, y, indicating their position in the transmission sequence of flow i. Let $Sent_i^k(s)$ represent the total data transmitted by flow i after completion of the transmission of the first k packets of the flow during the s-th round. Note that, the service received by flow i in round s prior to the transmission of its first packet in that round is equal to zero, i.e., $Sent_i^0(s) = 0$. Also, note that $Sent_i^y(s) = Sent_i(s)$, since both represent the total service received by flow i in round s. In general, of course,

$$0 \leq Sent_i^k(s) \leq Sent_i(s), \;\; 0 \leq k \leq y$$

The following defines a quantity that tracks the unused portion of a flow's allowance, and thus serves to help in determining the priority queue into which the flow should be placed.

Definition 4.2.1 *Define the* Unserved Allowance *of a flow at any given instant of time during a certain round as the flow's allowance for the round minus the amount of traffic transmitted by the flow during the round until that instant.*

Let $UA_i^k(s)$ represent the unserved allowance of flow i after the transmission of its k-th packet during the s-th round. In general, $UA_i^k(s)$ is computed as follows:

$$UA_i^k(s) = A_i(s) - Sent_i^k(s) \tag{4.1}$$

At the start of round s, before service for flow i begins, $UA_i^0(s)$ is exactly equal to the flow's allowance for the round, $A_i(s)$. Note that, the last packet transmitted from flow i in round s may cause the flow to exceed its allowance. This may result in a negative value of $UA_i^k(s)$.

Definition 4.2.2 *Define $UA_i^{max}(s)$ as the maximum possible value of the unserved allowance of flow i in round s.*

At the start of each round, the unserved allowance of a flow is initialized to its allowance for the round, as defined in Equation (2.6). Therefore, $UA_i^{max}(s)$ equals the maximum possible value of the right-hand side of Equation (2.6). Using Equation (3.1), we have,

$$UA_i^{max}(s) = w_i(1 + MaxSC(s-1)) \tag{4.2}$$

The ratio of the unserved allowance of a flow at a given instant and the allowance of the flow for the entire round represents the fraction of the allowance of a flow that is not yet consumed until the given instant. This ratio accurately captures how far ahead or behind a flow is in comparison to other flows in obtaining its fair share of service, and may therefore be used in placing flows in specific priority queues. However, an approximation to this quantity is necessary to ensure a per-packet work complexity of $O(1)$ for PERR.

Normalizing the unserved allowance of a flow with respect to its maximum possible value (instead of the actual allowance of the flow for the round) represents one measure, though not necessarily the most accurate measure, of the fraction of its allowance that is not yet consumed. The PERR scheduler uses this approximation which is necessary for the efficient implementation of the scheduler. It will be shown in later sections of this chapter that, in spite of this approximation, the PERR scheduler achieves better fairness than other known $O(1)$ schedulers.

Definition 4.2.3 *The* Unserved Allowance Quotient *of a flow at any given instant is defined as the ratio of the unserved allowance of the flow at that instant and its maximum possible* *unserved allowance, $UA_i^{max}(s)$. Let $Q_i^k(s)$ represent the unserved allowance quotient of flow i after the transmission of the k-th packet of flow i during the s-th round.*

$Q_i^k(s)$ is given by,

$$Q_i^k(s) = \frac{UA_i^k(s)}{UA_i^{max}(s)} = \frac{UA_i^k(s)}{w_i(1 + MaxSC(s-1))} \tag{4.3}$$

For purposes of brevity, in the rest of this chapter, the unserved allowance quotient will be simply referred to as the *quotient*.

The quotient of a flow at any given instant during a round represents the approximate fraction of its unserved allowance that can be used by the flow in the remainder of the round. The ERR scheduler never begins dequeuing the new packet from a flow i if $Sent_i^k(s)$ is equal to or more than the allowance, $A_i(s)$. Thus, the next packet of a flow is transmitted in the same round as the previous packet if and only if $UA_i^k(s)$ is positive. This in turn implies that a flow i, after the transmission of its k-th packet in round s, is eligible for more service in the same round if and only if,

$$0 < Q_i^k(s), \;\; 0 \leq k \leq y \tag{4.4}$$

where y is the number of packets of flow i served during round s.

The quotient for flow i at the start of round s is equal to $Q_i^0(s)$. Using Equations (2.6), (4.1), (4.2) and (4.3) we have,

$$Q_i^0(s) = \frac{UA_i^{max}(s) - SC_i(s-1)}{UA_i^{max}(s)}$$

Simplifying further, we get,

$$SC_i(s-1) = (1 - Q_i^0(s))\, UA_i^{max}(s) \tag{4.5}$$

This indicates that flow i has already used up $(1 - Q_i^0(s))$-th fraction of its $UA_i^{max}(s)$ in the excess service that it received in the previous round $(s-1)$. If the quotient for a flow at the start of a round is equal to unity, it implies that the surplus count of the flow following its service in the previous round is zero, i.e., the flow did not receive any excess service in the previous round.

In general, the larger the quotient of a flow, the lesser the proportion of its *UnservedAllowance* that has been expended in the current round.

Definition 4.2.4 *Define $Q^{max}(s)$ as the maximum of the quotients among all active flows at the start of round s.*

Since the *Unserved Allowance* for each flow at the start of a round is equal to its allowance, using Equations (2.6), (4.2) and (4.3), we have,

$$Q_i^0(s) = \frac{w_i(1 + MaxSC(s-1)) - SC_i(s-1)}{w_i(1 + MaxSC(s-1))}$$

This implies that the flow i with the least value of $\frac{SC_i(s-1)}{w_i}$, which is the normalized surplus count at the end of the previous round $(s-1)$, will be the one with the maximum value of the quotient among all active flows at the start of round s.

Ideally the scheduler should serve a packet from the flow with the largest quotient among all the active flows since it has received the least service in the current round. However, the complexity of maintaining a sorted list of active flows based on their quotients, and the complexity of computing the maximum in this list prior to each packet transmission is high. Given n flows, the work complexity of the scheduler prior to each packet transmission would be $O(\log n)$. The PERR scheduler avoids this by grouping the flows into a limited number of priority queues.

Figure 4.1(a) illustrates a block diagram of a generic scheduler. The *Scheduling Decision Module* which determines the order in which packets are served from the flow queues is the heart of the scheduler. Figure 4.1(b) details the architecture of the *Scheduling Decision Module* of the PERR scheduler which is responsible for selecting the next flow for service. As can be seen from Figure 4.1(b), an *Organizer*, p priority queues and a *Selector* are appended to the original *Scheduling Decision Module* of ERR. Let PQ_1, PQ_2, $\ldots$ and

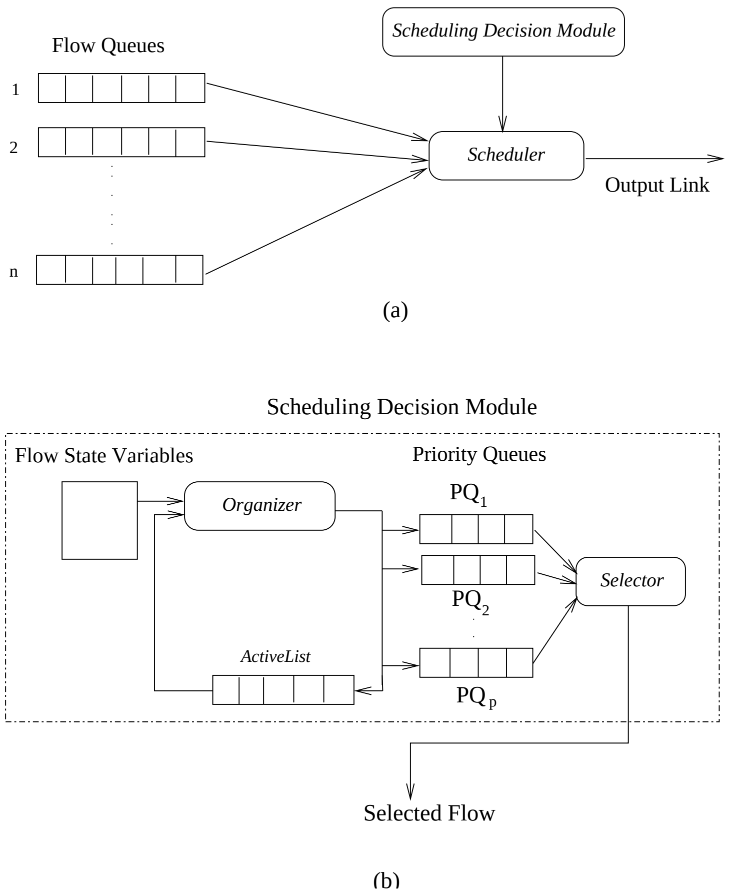


Figure 4.1: Block diagram of (a) PERR scheduler and (b) scheduling decision module of PERR

PQ_p denote the priority queues in the descending order of priority with PQ_1 representing the queue with the highest priority. Unlike the priority queues in Pre-order DRR which have to buffer the packets that will be transmitted in the round in progress, these queues in PERR simply contain the flow identifiers. As in ERR, the PERR scheduler maintains a linked list, called the *ActiveList*, of flows which are active. However, the flows in the *ActiveList* are not served in a round robin manner as in ERR. This is a list of the active flows that have exhausted their allowance in the current round but, will be eligible for receiving service in the subsequent round. It is the task of the *Organizer* to determine the order in which the flows will receive service in a round. At the start of the round, the *Organizer* module classifies the active flows present in the *ActiveList* into several classes according to their *Unserved Allowance Quotient* and places them into the corresponding priority queue. Since there are p priority queues, the *Organizer* can classify the flows into p classes. In general, class z of flow i after serving the k-th packet of the s-th round is derived as,

$$z = (p+1) - \left\lceil p \times \frac{Q_i^k(s)}{Q^{max}(s)} \right\rceil \tag{4.6}$$

Note that, at the start of the s-th round, $k = 0$ for all the flows in the above equation.

Note that, the quotient of a flow is a monotonically decreasing function of time over the duration of a round. Using this fact and the definition of $Q^{max}(s)$, we can conclude that during round s, the quotients of all the active flows will always be less than or equal to $Q^{max}(s)$, and thus z is always a non-negative quantity. By the above method, the flow with the maximum quotient at the start of the round s is initially added into the highest priority queue, PQ_1.

Note that, $Q^{max}(s)$ is computed only at the start of the round and need not be updated as the round progresses. The computation of $Q^{max}(s)$ simply requires the scheduler to record the least value of the normalized surplus count amongst all the flows in the *ActiveList*. This can be easily accomplished in $O(1)$ time by carrying out a simple comparison operation as the flows are added into the *ActiveList* after exhausting their allowance in the previous round.

Consider a situation where a flow i becomes active for the first time while some round s is in progress. It may be possible that the the initial value of the quotient of flow i, $Q_i^0(s)$ is greater than $Q^{max}(s)$. Using Equation (4.6), it is seen that the priority class for flow i is less than 1, the highest priority class. One possible solution would be to delay the service of flow i until the next round as is done in ERR. However, this would result in an increased latency for flow i since it would then have to wait until all the active flows have exhausted their allowance before receiving any service. The PERR scheduler instead simply adds flow i into the highest priority queue PQ_1. This eliminates the increased latency that would be otherwise experienced by flow i.

When the scheduler is ready to transmit, the *Selector* module selects the highest non-empty priority queue, say PQ_e and chooses the flow at the head of PQ_e, say flow i, for service. The scheduler serves the packet at the head of the queue corresponding to flow i and following the service of this packet recalculates the priority class, z, to which flow i belongs using Equation(4.6). The scheduler will continue to serve the next packet from the queue of flow i until the occurrence of at least one of the following events:

1. *A newly active flow is added to a higher priority queue :* In this case, the scheduler stops the service of flow i and begins serving the newly active flow since it belongs to a higher priority class than flow i.

2. *Queue of flow i is empty :* In this case, flow i is removed from the head of priority queue PQ_e. Also, the $Served$ flag for flow i is set to indicate that it has received service in the current round.

3. *The newly computed priority class of flow i, f, does not match its current priority class, e :* In this case flow i is removed from the head of priority queue PQ_e and is added to the tail of priority queue PQ_f. Note that, the only exception is when $f > p$. In this case no further packets are to be scheduled from flow i during the current round since it has exhausted its allowance. Flow i is instead added to the tail of the *ActiveList*.

At any given instant of time each active flow can either be present in at most one of the p priority queues or in the *ActiveList*. Over the course of the round, as the flow consumes more and more of its allowance, it will gradually move down from the higher priority queues into lower priority queues until it completely exhausts its allowance following which it is added into the *ActiveList*. However, during a round it is not necessary that a flow will pass through each of the p priority queues. In fact it may be possible that the only priority queue which a flow visits is the initial queue into which it is classified at the start of the round. As a result of the categorization brought about by the priority queue module in PERR, each flow uses its allowance in pieces over the course of the round. The *Organizer* reorders the sequence of transmissions to enable the flows that have not utilized a large portion of their *UnservedAllowance* to get precedence over the other flows.

The PERR scheduler also maintains two flags, *Served* and *Active* for each flow. The *Active* flag indicates whether a flow is active or not. The *Served* flag is set when the scheduler serves the first packet from a flow in the current round and remains set for the entire duration of the round indicating that the flow has been served at least once during the current round. The *Served* flags for all the flows are reset at the start of a new round. The *Served* flag prevents a flow which frequently oscillates between active and inactive periods to receive excessive service. Consider a flow which runs out of packets in the middle of a round before utilizing its entire allowance in that round. Since this flow is no longer active its *Active* flag will be reset. Assume that a new packet arrives at the flow at some time before the end of the current round. In the absence of the *Served* flag this flow would be treated as a newly active flow and its per-flow states would be reset. This would allow the flow to receive service in excess of what it would have received if it were active during the entire duration of the current round. However, in PERR, the *Served* flag will be set for the flow under consideration indicating that the per-flow states for that flow are still valid for the current round. When the flow becomes active it will be added into the appropriate priority queue using Equation (4.6) depending on how much of its *UnservedAllowance* was utilized when it was last active in the current round, and thus, the flow will not receive any excess service. Note that, since the *Served* flags for all the flows are reset at the end of a round, accumulation of service credits from a previous round is prevented.

A psuedo-code implementation of the PERR scheduling algorithm is shown in Figure 4.2, consisting of the *Initialize*, *Enqueue* and *Dequeue* routines. The *Enqueue* routine is called when a new packet arrives at a flow. The *Dequeue* routine is the heart of the algorithm which schedules packets from the queues corresponding to different flows. Figure 4.3–4.6 illustrate the pseudo-code of four routines that are used in the execution of the *Enqueue* and *Dequeue* routines. All of these routines can be easily implemented as simple hardware modules.

```
Initialize: (Invoked when the scheduler is initialized)
    $MaxSC = 0$;
    $CurrentPriority = 0$;
    $Q^{max} = 1$;
    $MinNormalizedSC = MAX$;
    for ($i = 0; i < n; i = i + 1$)
        $Active_i$ = FALSE;
        $Served_i$ = FALSE;

Enqueue: (Invoked when a packet arrives)
    $i$ = QueueInWhichPacketArrives;
    if ( $Active_i$ == FALSE) then
        if ($Served_i$ ==FALSE) then
            $SC_i = 0$;
            $Sent_i = 0$;
        end if;
        InitializeFlow($i$);
        $NewPriority$ = ComputeNewPriority($i$);
        AddToPriorityQueue($NewPriority, i$);
        if ($NewPriority > CurrentPriority$) then
            ObtainHighestActivePriority == TRUE;
        end if
    end if;

Dequeue:
    while (TRUE) do
        if (AllPriorityQueuesEmpty == TRUE) then
            InitializeRound();
        end if;
        if(ObtainHighestActivePriority == TRUE) then
            $CurrentPriority$ = GetHighestActivePriorityQueue;
        end if;
        $i$ = HeadOfPriorityQueue($CurrentPriority$);
        do
            TransmitPacketFromQueue($i$);
            Increase $Sent_i$ by LengthInFlitsOfTransmittedPacket;
            $Served_i$ == TRUE;
            $NewPriority$ == ComputeNewPriority($i$);
        while ( ($NewPriority == CurrentPriority$) and
                (IsEmpty($Queue_i$) == FALSE) and
                (ObtainHighestActivePriority == FALSE) )
        if ( ($NewPriority > CurrentPriority$) or
            (IsEmpty($Queue_i$) == TRUE) ) then
            RemoveHeadOfPriorityQueue($CurrentPriority$);
            if (IsEmpty($Queue_i$) == FALSE) then
                AddToPriorityQueue($NewPriority, i$);
            end if
            if (IsEmpty($CurrentPriority$) == TRUE) then
                ObtainHighestActivePriority = TRUE;
            end if
        end if
    end while
```

Figure 4.2: Pseudo-code for PERR

```
InitializeRound()
    for ($i = 0; i < n; i = i + 1$)
        $Served_i$ = FALSE;
    PreviousMaxSC = MaxSC;
    MaxSC $= 0$;
    MinNormalizedSC $= MAX$;
    InitializeFlow($min$);
    $Q^{max} = \frac{A_{min}}{UA^{max}_{min}}$;
    ObtainHighestActivePriority == TRUE;
    while (IsEmpty(ActiveList) == FALSE) do
        $flow$ = HeadOfActiveList;
        RemoveHeadOfActiveList;
        $Sent_{flow} = 0$;
        InitializeFlow($flow$);
        $NewPriority$ = ComputeNewPriority($flow$);
        AddToPriorityQueue($NewPriority$, $flow$);
        $SC_{flow} = 0$;
    end while;
```

Figure 4.3: *InitializeRound*() routine

```
InitializeFlow($i$)
    $Active_i$ = TRUE;
    $UA_i^{max} = w_i(1+$ PreviousMaxSC) ;
    $A_i = UA_i^{max} - SC_i$;
```

Figure 4.4: *InitializeFlow*() routine

```
AddToPriorityQueue($z, i$);
    if ($z > p$) then
        AddFlowToActiveList($i$);
        $SC_i = Sent_i - A_i$;
        if ($\frac{SC_i}{w_i} > MaxSC$) then
            $MaxSC = \frac{SC_i}{w_i}$;
        end if
        if ($\frac{SC_i}{w_i} < MinNormalizedSC$ ) then
            $min = i$;
            $MinNormalizedSC = \frac{SC_i}{w_i}$;
        end if
    else
        AddFlowToPriorityQueue($Pq_z$ , $i$);
    end if
```

Figure 4.5: *AddToPriorityQueue*() routine

```
ComputeNewPriority($i$)
    $Q_i = \frac{A_i - Sent_i}{A_i^{max}}$ ;
    $z = (p+1) - \left\lceil \frac{Q_i \times p}{Q^{max}} \right\rceil$ ;
    return $z$;
```

Figure 4.6: *ComputeNewPriority*() routine

Chapter 5. Performance Analysis of PERR

This chapter focuses on the performance analysis of the PERR scheduler. We analytically prove the fairness and latency properties of PERR, using a novel approach based on interpreting the PERR scheduler as an instance of the Nested Deficit Round Robin (Nested-DRR) discipline discussed in [40]. We prove that the latency bound obtained in this paper using this approach is tight. We also show that the per-packet work complexity of the PERR scheduler is $O(1)$ with respect to the number of flows and $O(\log p)$ with respect to the number of priority queues, p. It is important to note that $p << n$, where n are the total number of flows being serviced by the scheduler. As a result, the work complexity of the PERR scheduler is much lower than the sorted-priority schedulers such as WFQ which have a work complexity of $O(\log n)$. This low work complexity of PERR makes it attractive for implementation in high speed switches and routers.

The rest of this chapter is organized as follows. Section 5.1 discusses the interpretation of PERR bandwidth allocations as an instance of allocations in a nested version of ERR. In Section 5.2, we evaluate the latency bound of ERR and prove that it belongs to the general class of Latency Rate ($\mathcal{LR}$) servers [38]. In addition we also show that the latency bound derived in this section is tight. Section 5.3 analyzes the relative fairness bound of PERR. In Section 5.4, we prove that the worst-case work complexity of PERR is $O(\log p)$, where p denotes the number of priority queues in the PERR scheduler.

5.1 Nested Round Robin Interpretation

The primary goal of the PERR scheduler is to distribute the *UnservedAllowance* of a flow in an ERR round into several parts, so that it can be utilized in pieces over the course of the round. The Nested-DRR algorithm proposed in [40], modifies the DRR scheduler by creating a nested set of multiple rounds inside each DRR round. The Nested-DRR scheduler serves the active flows in a round robin order in these nested rounds by executing a modified version of the DRR algorithm.

We can hypothetically interpret the operation of the PERR scheduler as a *nested* version of ERR which is similar to Nested-DRR. This interpretation proves useful in the analysis of the latency bound of the PERR scheduler. Each round in ERR can be referred to as an *outer round*. The time interval during which the PERR scheduler serves the flows present in priority queue PQ_u during the s-th outer round is referred to as *inner round* (s, u). In effect, each outer round is split into as many inner rounds as the number of priority queues, p. Since the PERR scheduler serves the priority queues in a descending order starting at the higher priority queue PQ_p, the first inner round during outer round s will be $(s, 1)$, while (s, p) will denote the last inner round.

From Equation (4.5), we know that the excess service, if any, received by each flow i in the previous outer round $(s-1)$ is equal to $(1 - Q_i^0(s))\,UA_i^{max}(s)$. Since $Q^{max}(s)$ represents the maximum quotient among all the flows at the start of round s, it is guaranteed that each active flow i has already utilized at least $(1 - Q^{max}(s))$-th fraction of its $(UA_i^{max}(s))$ during its last service opportunity in the previous round $(s-1)$. The goal of the PERR scheduler is to distribute the remaining portion of each flow's maximum possible *UnservedAllowance*, $Q^{max}(s)(UA_i^{max}(s))$, equally among the p inner rounds. Let $IdealServed_i(s)$ represent the ideal service received by flow i during each inner round of the s-th outer round. $IdealServed_i(s)$ is computed as follows,

$$IdealServed_i(s) = \frac{Q^{max}(s)(UA_i^{max}(s))}{p} \tag{5.1}$$

Ideally, therefore, each flow i will receive exactly $IdealServed_i(s)$ amount of service in each of the p inner rounds of outer round s. In reality, however, the last packet served in an inner round from a flow may cause it to exceed its ideal service in that inner round. Just as in ERR, a *Surplus Count (SC)* is maintained for each flow which records any excess service received by the flow. The flow is penalized for this excess transmission in the subsequent inner round. When the scheduler selects a flow i for service in an inner round (s, u), its SC is incremented by $IdealServed_i(s)$. The scheduler will serve the packet at the head of flow i as long as its SC value is positive. Following the transmission of a packet, the SC corresponding to that flow is decremented by the size of the transmitted packet. Let $SC_i(s, u)$ represent the surplus count of flow i at the end of inner round (s, u). Further, let $Served_i(s, u)$ denote the actual service received by flow i in inner round (s, u). $SC_i(s, u)$ is calculated as follows,

$$SC_i(s, u) = Served_i(s, u) - (IdealServed_i(s) + SC_i(s, u - 1)) \tag{5.2}$$

Note that, if $IdealServed_i(s)$ is less than or equal to the $SC_i(s, u - 1)$, then flow i will not receive any service in inner round (s, u). Thus, a flow does not necessarily receive service in each inner round. However, the surplus count for flow i is updated at the end of each inner round using Equation (5.2), irrespective of whether the flow receives service in that inner round or not. In fact, it may be possible that none of the active flows receive service in an inner round. Hence, if the PERR scheduler followed a round robin service order as in Nested-DRR, then the scheduler would have a prohibitively large work complexity. However, the *Organizer* module of the PERR scheduler decides which priority queues each active flow is added into over the course of each outer round. This, in turn, determines the inner rounds in which each flow will be served. The PERR scheduler does not need to query all the active flows in a round robin order, thus leading to a low implementation complexity.

Note that, the surplus count of a flow at the end of the last inner round of an outer round is the same as its surplus count at the end of the corresponding round in ERR. In other words, $SC_i(s, p)$ is the same as $SC_i(s)$. Also, note that $SC_i(s, 0)$ represents the surplus count of flow i at the start of the first inner round, $(s, 1)$, in outer round s. As explained earlier, we know that flow i should ideally transmit $Q^{max}(s)(UA_i^{max}(s))$ worth of data in outer round s. The remaining fraction, $(1 - Q^{max}(s))$-th of the quantity $UA_i^{max}(s)$, has already been utilized in the excess service received by flow i in outer round $(s-1)$ and, therefore, is a part of $SC_i(s-1)$. To account for this already utilized portion of $UA_i^{max}(s)$, $SC_i(s,0)$ is computed as:

$$SC_i(s,0) = SC_i(s-1) - (1 - Q^{max}(s))(UA_i^{max}(s)) \tag{5.3}$$

It can be easily proved that Equation (3.1), which expresses the bounds on the surplus count, $SC_i(s)$, also holds true for $SC_i(s,u)$. Therefore, for any flow i and inner round (s,u),

$$0 \leq SC_i(s,u) \leq m-1 \tag{5.4}$$

Definition 5.1.1 *Let $Sent_i(s,u)$ represent the total service received by flow i since the start of the s-th outer round until the PERR scheduler has finished serving the flows in the priority queue, PQ_u.*

Note that, it is not necessary that flow i was present in priority queue PQ_u during outer round s. From Equation (5.2), the total data served from flow i in inner round (s,u) is,

$$Served_i(s,u) = IdealServed_i(s) + SC_i(s,u) - SC_i(s,u-1) \tag{5.5}$$

$Sent_i(s,u)$ is calculated as follows:

$$Sent_i(s,u) = \sum_{w=1}^{w=u} Served_i(s,w) \tag{5.6}$$

Substituting for $Served_i(s,w)$ from Equation (5.5) in Equation (5.6), we have,

$$Sent_i(s,u) = u(IdealServed_i(s)) + SC_i(s,u) - SC_i(s,0) \tag{5.7}$$

$Sent_i(s,u)$ will be positive only if $u(IdealServed_i(s))$ is greater than $SC_i(s,0)$. Otherwise it indicates that flow i has not received any service until the end of the (s,u)-th inner round.

However, flow i is guaranteed to receive service in at least one inner round during the s-th outer round. Using Equations (5.1), (5.3) and (5.7) we have,

$$Sent_i(s, u) = \left(\frac{u}{p}\right) Q^{max}(s)\, UA_i^{max}(s) + SC_i(s, u) - SC_i(s-1) \tag{5.8}$$

Definition 5.1.2 *Define $Sent_i(s)$ as the total service received by flow i in outer round s.*

Note that, $Sent_i(s, p)$ represents the service received by flow i when the scheduler has finished serving the flows in priority queue PQ_p which in fact equals $Sent_i(s)$. Substituting $u = p$ in Equation (5.8) and using Equation(4.2), we get,

$$Sent_i(s) = w_i(1 + MaxSC(s-1)) + SC_i(s) - SC_i(s-1) \tag{5.9}$$

Hence, the total service received by flow i in an outer round in PERR is identical to the service received by flow i in the corresponding round in ERR.

Ideally, during the normal operation of the PERR scheduler, the p inner rounds in outer round s follow a strictly sequential order starting at inner round $(s, 1)$ and ending at round (s, p). However, in certain situations, it is possible to interrupt the sequential ordering. Let us assume that a flow j becomes active for the first time in outer round s while the PERR scheduler is serving a flow k at the head of priority queue PQ_d. Since the quotient for flow j is equal to 1, it will be added into priority queue PQ_1 which has the highest priority among all the priority queues. Upon finishing the transmission of the current packet from flow k, the PERR scheduler temporarily suspends the service of flow k and starts serving flow j which is at the head of queue PQ_p. The PERR scheduler will keep serving flow j until it is added either into priority queue PQ_d or some other queue with lower priority than PQ_d. The scheduler will then resume service of the k-th flow. Note that, the service received by flow i while it is present in queue PQ_p is part of the inner round (s, p) even though it is not contiguous with the time interval during which the PERR scheduler served the flows present in queue PQ_p at the start of the outer round s. Also, the inner round (s, d) will not extend over a continuous time interval because it will be interposed by the entire

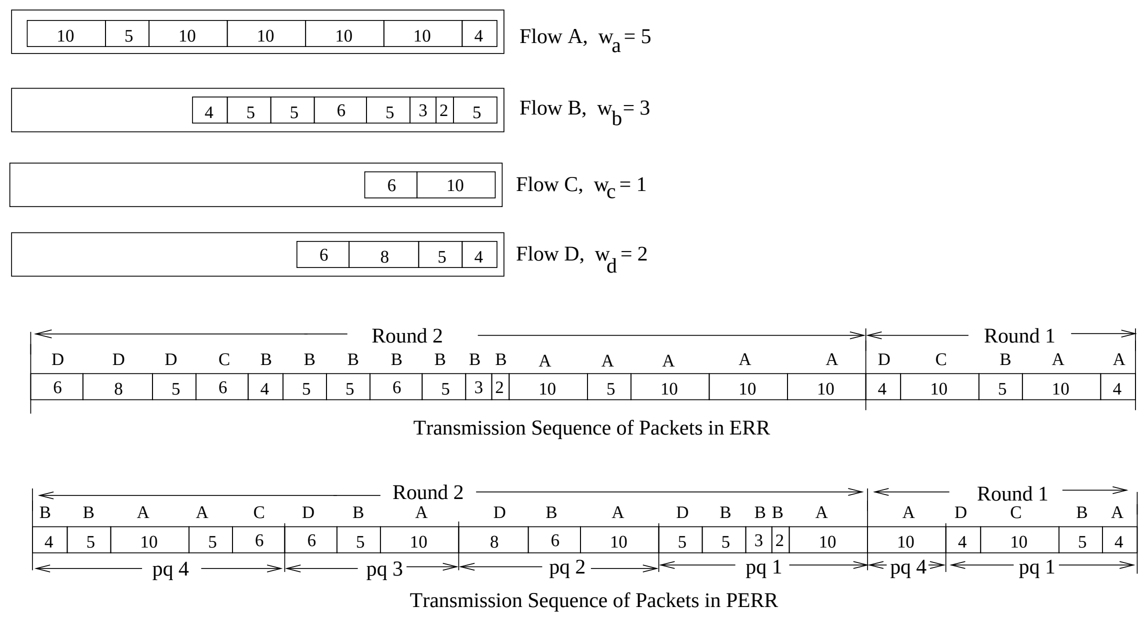


Figure 5.1: Comparison of the transmission sequence of packets in ERR and PERR over two rounds of execution

service received by flow i since the time it became active until its addition into priority queue PQ_d or a lower priority queue. It is, therefore, not necessary that the inner rounds in an outer round should sequentially follow one another and that the flows which receive service in an inner round should be served in succession. However, note that this disruption of the otherwise sequential service can only be caused due to a new flow becoming active during the execution of that outer round.

Figure 5.1 compares the transmission sequence in the first two rounds of execution of the ERR and PERR schedulers for the given input pattern and flow weights. In the PERR scheduler, the flows are classified into 4 classes corresponding to the 4 priority queues. At the start of inner round $(1, 1)$ all the flows are present in priority queue PQ_1. After receiving service in this inner round, flow A is added into PQ_4. However, the other 3 flows exceed their allowance in this inner round and are therefore added into the *ActiveList*. Note that, in the second outer round, unlike the ERR scheduler where the flows B and D have to wait for their turn in the round robin order to receive service, flows B and D start receiving service in the inner round $(2, 1)$. However, note that flow C is served for the first time only in inner round $(2, 4)$. This is because its surplus count in the previous outer round was very large resulting in a very low value of the quotient.

In the following sections we present analytical results on the fairness, latency properties and the work complexity of PERR.

5.2 Latency Analysis

A detailed description on the concept of the Latency-Rate ($\mathcal{LR}$) servers was presented in Section 3.3. In this section we derive an upper bound on the latency of the PERR scheduler and prove that it belongs to the general class of $\mathcal{LR}$-servers. In addition, we also prove that this bound is tight by illustrating a case where the bound is actually met. Our approach here is similar to the one used in Section 3.4 while analyzing the latency of the ERR scheduler.

Theorem 5.2.1 *The PERR scheduler belongs to the class of $\mathcal{LR}$ servers, with an upper bound on the latency Θ_i for flow i given by,*

$$\Theta_i \leq \frac{\frac{(W-w_i)m}{p} + (n-1)(m-1)}{r} \tag{5.10}$$

where n is the total number of active flows, p is the number of priority queues, r is the transmission rate of the output link and W is the sum of the weights of all the flows.

Proof: Since the latency of a $\mathcal{LR}$ server can be estimated based on its behavior in the flow active periods, we will prove the theorem by showing that,

$$\Theta_i' \leq \frac{\frac{(W-w_i)m}{p} + (n-1)(m-1)}{r}.$$

Let τ_i be the time instant when flow i becomes active. To prove the statement of the theorem we consider a time interval (τ_i, t), where $t > \tau_i$, during which flow i is continuously active. We first obtain the lower bound on the total service received by flow i during the time interval under consideration. Then we express the lower bound in the form of Equation (3.10) to derive the latency bound.

In Section 3.4 and [47] it has been proved that to obtain a tight upper bound on the latency of the ERR scheduler, we must consider an active period (τ_i, t) such that τ_i coincides with the beginning of the service opportunity of a flow and t belongs to the set of time instants at which the scheduler begins serving flow i. We can easily prove that the same conditions apply for proving the upper bound on the latency of the PERR scheduler. Let $\tau_i^{(e,f)}$ be the time instant marking the start of the service of flow i when flow i is at the head of priority queue PQ_f in round e. In other words, this time instant represents the start of the service opportunity of flow i in inner round (e, f). Therefore, in trying to determine the latency bound of the PERR, we need to only consider time interval $(\tau_i, \tau_i^{(e,f)})$ for all (e, f).

The first step in proving the latency bound involves determining the upper bound on the size of the time interval under consideration. Note that, the time instant τ_i may or may not coincide with the start of a new round. Let k_0 be the round which is in progress at time instant τ_i or which starts exactly at time instant τ_i. Let t_h mark the start of the round $(k_0 + h)$. In either case, flow i will be able to transmit at least $A_i(k_0)$ worth of data over the course of the k_0-th round. If flow i becomes active when the round k_0 is in progress, i.e. when $\tau_i < t_0$ then the service received during the interval (t_0, τ_i) will be excluded from the time interval under consideration. The time interval $(\tau_i, \tau_i^{(e,f)})$ will be maximal only if the time instant τ_i coincides with t_0, the start of the k_0-th round. Hence, we assume that τ_i coincides with the start of the the k_0-th round. Figure 5.2 illustrates the interval under consideration assuming that (e, f) is equal to $(k_0 + k, v)$. Note that, in Figure 5.2, $OR(e)$ represents the e-th outer round in the execution of the PERR scheduler and $IR(e, f)$ denotes the inner round (e, f).

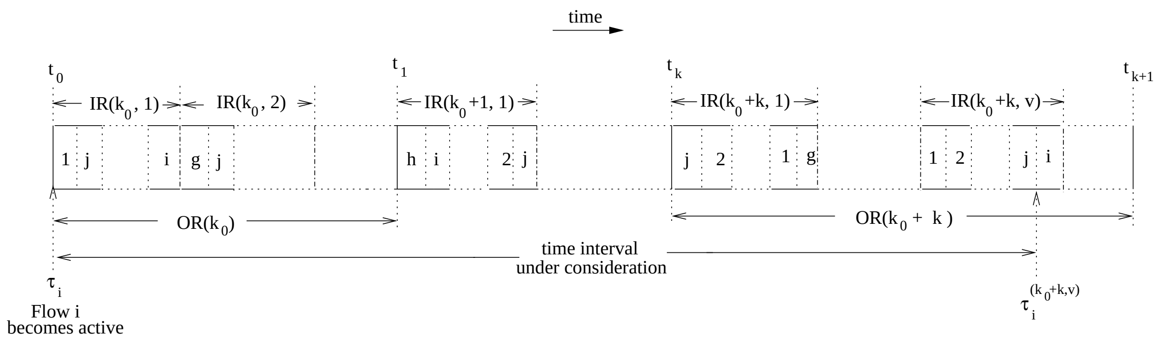


Figure 5.2: An illustration of the time interval under consideration for the analysis of the latency bound of PERR

The time interval under consideration, $(\tau_i, \tau_i^{(k_0+k,v)})$, can be split into two sub-intervals:

1. (τ_i, t_k): This sub-interval includes k rounds of execution of the PERR scheduler starting at round k_0. Consider the time interval (t_h, t_{h+1}) when round (k_0+h) is in progress. Summing Equation (5.9) over all n flows,

$$\begin{aligned} t_{h+1} - t_h = &\frac{W}{r}(1 + MaxSC(k_0 + h - 1)) \\ &+ \frac{1}{r}\sum_{j=1}^{n}\{SC_j(k_0+h) - SC_j(k_0+h-1)\} \end{aligned}$$

Summing the above over k rounds beginning with round k_0,

$$\begin{aligned} t_k - \tau_i = &\frac{W}{r}(k) + \frac{W}{r}\sum_{h=0}^{k-1} MaxSC(k_0 + h - 1) \\ &+ \frac{1}{r}\sum_{j=1}^{n}\{SC_j(k_0+k-1) - SC_j(k_0-1)\} \end{aligned} \tag{5.11}$$

2. $(t_k, \tau_i^{(k_0+k,v)})$: This sub-interval includes the part of the $(k_0 + k)$-th round prior to the start of the service of flow i when it is at the head of priority queue PQ_v. In the worst-case flow i will be the the last flow to receive service among all other flows

which are present in priority queue PQ_v. In this case, during the sub-interval under consideration, the service received by flow i equals $Sent_i(k_0+k, v-1)$ whereas the service received by each flow j among the other $n-1$ flows equals $Sent_i(k_0+k, v)$. Note that, if v equals 1 then flow i does not receive service in this sub-interval. Hence, summing $Sent_i(k_0+k, v-1)$ and $Sent_j(k_0+k, v)$ for each flow j such that $1 \leq j \leq n, j \neq i$, we have,

$$\tau_i^{(k_0+k,v)} - t_k = \frac{1}{r}\left(Sent_i(k_0+k, v-1) + \sum_{\substack{j=1 \\ j\neq i}}^{n} Sent_j(k_0+k, v) \right)$$

Using Equation (5.8) in the above, we have,

$$\begin{aligned}
\tau_i^{(k_0+k,v)} - t_k = & \frac{1}{r}\sum_{\substack{j=1 \\ j\neq i}}^{n} \left(\frac{v}{p}\right) Q^{max}(k_0+k)\,UA_j(k_0+k) \\
& + \frac{1}{r}\left(\frac{v-1}{p}\right) Q^{max}(k_0+k)\,UA_i(k_0+k) \\
& + \frac{1}{r}\sum_{\substack{j=1 \\ j\neq i}}^{n}(SC_j(k_0+k, v) - SC_j(k_0+k-1)) \\
& + \frac{1}{r}(SC_i(k_0+k, v-1) - SC_i(k_0+k-1))
\end{aligned} \tag{5.12}$$

To simplify the analysis we introduce a new variable Ω, such that,

$$\Omega = Q^{max}(k_0+k)(1 + MaxSC(k_0+k-1)) \tag{5.13}$$

Using Equation (3.2) in Equation (5.13), we get,

$$0 < \Omega \leq m \tag{5.14}$$

Combining Equations (5.11) and Equations (5.12) and using Equations (4.2) and (5.13), we have,

$$\begin{aligned}\tau_i^{(k_0+k,v)} - \tau_i \leq \frac{W}{r}k + \frac{W}{r}\sum_{h=0}^{k-1} MaxSC(k_0+h-1) + \frac{1}{r}\sum_{\substack{j=1\\j\neq i}}^{n}\left(\frac{v}{p}\right)w_j\Omega \\ + \frac{1}{r}\left(\frac{v-1}{p}\right)w_i\Omega + \frac{1}{r}\sum_{\substack{j=1\\j\neq i}}^{n}(SC_j(k_0+k,v) - SC_j(k_0-1)) \\ + \frac{1}{r}(SC_i(k_0+k,v-1) - SC_i(k_0+k-1))\end{aligned}$$

Using the bounds on the surplus count from Equation (3.1) in the above equation, we have,

$$\begin{aligned}\tau_i^{(k_0+k,v)} - \tau_i \leq \frac{W}{r}k + \frac{W}{r}\sum_{h=0}^{k-1} MaxSC(k_0+h-1) + \frac{1}{r}\sum_{\substack{j=1\\j\neq i}}^{n}\left(\frac{v}{p}\right)w_j\Omega \\ + \frac{1}{r}\left(\frac{v-1}{p}\right)w_i\Omega + \frac{1}{r}(n-1)(m-1) + \frac{1}{r}SC_i(k_0+k,v-1)\end{aligned} \tag{5.15}$$

Solving for k and using the fact that W is the sum of the weights of all the n flows,

$$\begin{aligned}k \geq (\tau_i^{(k_0+k,v)} - \tau_i)\frac{r}{W} - \sum_{h=0}^{k-1} MaxSC(k_0+h-1) - \frac{1}{W}\left(\frac{v}{p}\right)(W-w_i)\Omega \\ - \frac{1}{W}\left(\frac{v-1}{p}\right)w_i\Omega - \frac{1}{W}(n-1)(m-1) - \frac{1}{W}SC_i(k_0+k,v-1)\end{aligned} \tag{5.16}$$

Note that, the total data transmitted by flow i during the time interval under consideration can be expressed as the following summation,

$$Sent_i(\tau_i, \tau_i^{(k_0+k,v)}) = Sent_i(\tau_i, t_k) + Sent_i(t_k, \tau_i^{(k_0+k,v)}) \tag{5.17}$$

As explained, earlier $Sent_i(t_k, \tau_i^{(k_0+k,v)})$ is the same as $Sent_i(k, v-1)$. $Sent_i(\tau_i, t_k)$ can be obtained by summing Equation (5.9) over k rounds starting at round k_0. Substituting the result and Equation (5.8) in Equation (5.17), we get,

$$\begin{aligned}Sent_i(\tau_i, \tau_i^{(k_0+k,v)}) = w_i k + w_i\sum_{h=0}^{k-1} MaxSC(k_0+h-1) \\ + \left(\frac{v-1}{p}\right)UA_i(k_0+k) + SC_i(k_0+k,v-1) - SC_i(k_0)\end{aligned}$$

Note that, in PERR, the surplus count of a newly active flow is initialized to zero. As a result, since flow i becomes active at time instant τ_i, $SC_i(k_0)$ is equal to zero. Substituting this and using Equations (4.2) and (5.13) in the above equation, we get,

$$\begin{aligned} Sent_i(\tau_i, \tau_i^{(k_0+k,v)}) = w_i k + w_i \sum_{h=0}^{k-1} MaxSC(k_0+h-1) \\ + \left(\frac{v-1}{p}\right) w_i \Omega + SC_i(k_0+k, v-1) \end{aligned} \tag{5.18}$$

Using Equation (5.16) to substitute for k in Equation (5.18), we get,

$$\begin{aligned} Sent_i(\tau_i, \tau_i^{(k_0+k,v)}) \geq \frac{w_i r}{W}(\tau_i^{(k_0+k,v)} - \tau_i) + \left(\frac{v-1}{p}\right) w_i \Omega \left(\frac{W-w_i}{W}\right) \\ - \frac{w_i}{W}\left(\frac{v}{p}\right)(W-w_i)\Omega - \frac{w_i}{W}(n-1)(m-1) \\ - SC_i(k_0+k, v-1)\left(\frac{w_i}{W} - 1\right) \end{aligned}$$

Simplifying further we get,

$$\begin{aligned} Sent_i(\tau_i, \tau_i^{(k_0+k,v)}) \geq \frac{w_i r}{W}\Bigg((\tau_i^{(k_0+k,v)} - \tau_i) - \frac{1}{r}\left(\frac{W-w_i}{p}\right)\Omega \\ - \frac{1}{r}(n-1)(m-1)\Bigg) - SC_i(k_0+k, v-1)\left(\frac{w_i}{W} - 1\right) \end{aligned} \tag{5.19}$$

Using Equation (5.15), it can be easily verified that,

$$\tau_i^{(k_0+k,v)} - \tau_i > \frac{1}{r}\left(\frac{W-w_i}{p}\right)\Omega - \frac{1}{r}(n-1)(m-1) \tag{5.20}$$

Now, since the reserved rates are proportional to the weights assigned to the flows as given by Equation (2.4), and since the sum of the reserved rates is no more than the link rate r, we have,

$$\rho_i \leq \frac{w_i}{W} r \tag{5.21}$$

Substituting for $\frac{w_i r}{W}$ from Equation (5.21) in Equation (5.19) and using Equation (5.20) we have,

$$\begin{aligned} Sent_i(\tau_i, \tau_i^{(k_0+k,v)}) \geq \rho_i \Bigg((\tau_i^{(k_0+k,v)} - \tau_i) - \frac{1}{r}\left(\frac{W - w_i}{p}\right)\Omega \\ - \frac{1}{r}(n-1)(m-1)\Bigg) - SC_i(k_0+k, v-1)\left(\frac{w_i}{W} - 1\right) \end{aligned} \quad (5.22)$$

Comparing the above equation with Equation (3.10), the latency bound is given by,

$$\begin{aligned} \Theta_i \leq \frac{1}{r}\left(\frac{W - w_i}{p}\right)\Omega + \frac{1}{r}(n-1)(m-1) \\ + SC_i(k_0+k, v-1)\left(\frac{w_i}{W} - 1\right) \end{aligned} \quad (5.23)$$

From the above equation it is readily seen that the latency reaches the upper bound under the following conditions:

- Ω is equal to its upper bound, m. (From Equation (5.14))
- $SC_i(k_0 + k, v - 1)$ is equal to its lower bound, 0. (From Equation (3.1))

Substituting these bounds in Equation (5.23), we get,

$$\Theta_i \leq \frac{1}{r}\left(\left(\frac{W - w_i}{p}\right) m + (n-1)(m-1)\right) \quad (5.24)$$

As discussed earlier, flow i will experience its worst latency during an interval $(\tau_i, \tau_i^{(e,f)})$ for some inner round (e, f). Therefore, from Equation (5.24), the statement of the theorem is proved. ■

We now proceed to show that the latency bound given by Theorem (5.2.1) is tight by illustrating a case where the bound is actually met. Assume that a flow i becomes active at time instant τ_i, which also coincides with the start of a certain round k_0. Assume that for any time instant t, $t \geq \tau_i$, a total of n flows, including flow i, are active. Also, assume that the summation of the reserved rates of all the n flows equals the output link transmission rate, r. Hence, $\rho_i = \frac{w_i}{W} r$. Since flow i became active at time τ_i, its surplus count at the start of round k_0 is 0. Let the surplus count of all the other flows at the start of round k_0 be equal to 0. Assume that, a flow l which is not active after time τ_i and hence is not included in the n flows, was active during the k_0-th round. Assume that flow l exceeded its allowance by $(m-1)$ in its last service opportunity in round $(k_0 - 1)$, leading to a value of $MaxSC(k_0-1)$ equal to $(m-1)$. Since the surplus counts of all the n active flows are equal to 0, the *Unserved Allowance Quotient* for all the flows at the start of the k_0-th round will be equal to unity. Hence, $Q^{max}(k_0)$ will be equal to 1 and all the n flows will be added into the priority queue PQ_p at the start of the round. Assume that flow i is the last flow to be added into this queue. From Equations (5.4), (3.2) and (5.5), any given flow j can transmit a maximum of $w_j(\frac{m}{p}) + (m-1)$ bits during its service opportunity in an inner round. In the worst case, before flow i is served by the PERR scheduler, each of the other $(n-1)$ flows will receive this maximum service. Hence, the cumulative delay until flow i receives service is given by,

$$\begin{aligned} D &= \frac{(\sum_{j \neq i} w_j)(\dfrac{m}{p}) + (n-1)(m-1)}{r} \\ &= \frac{(\frac{W-w_i}{p})m + (n-1)(m-1)}{r} \end{aligned}$$

Noting that $S_i(\tau_i, \tau_i + D)$ equals zero, it is readily verified that the bound is exactly met at time $t = \tau_i + D$.

5.3 Fairness Analysis

In our fairness analysis, we use the popular metric, *Relative Fairness Bound (RFB)* first proposed in [24] and defined in Section 3.2. The RFB is defined as the maximum difference in the normalized service received by any two flows over all possible intervals of time.

Theorem 5.3.1 *For any execution of the PERR scheduling discipline,* $RFB < 2m + \frac{2m}{p}$

Proof: In [46] and Section 3.2, while analyzing the fairness properties of ERR, we have proved that a tight upper bound on the the RFB of ERR can be obtained by considering only a subset of all possible time intervals. This subset is the set of all time intervals bounded by time instants that coincide with the start of the service opportunities of flows. It can be easily verified that to prove the RFB of the PERR scheduler we need to consider a time interval (t_1, t_2) such that both the time instants t_1 and t_2 coincide with the start of a service opportunity of a flow in an inner round.

Consider any two flows i and j that are active in the time interval between the time instants t_1 and t_2. Let (k_0, f) and $(k_0 + k, g)$ be the inner rounds which are in progress at time instants t_1 and t_2 respectively. Let time instant $t_{(h,v)}$ mark the start of inner round $(k_0 + h, v)$. In other words, $t_{(k_0,f)} < t_1 < t_{(k_0,f+1)}$ and $t_{(k_0,g)} < t_2 < t_{(k_0,g+1)}$. It may be possible that, if flow j receives service in the inner round (k_0, f), then it does so in the time interval $(t_{(k_0,f)}, t_1)$. Unlike the ERR scheduler, in PERR if flow j is served before flow i in a certain inner round then, it is not necessary that the same order of service is followed in each of the following inner rounds. Hence, on a similar note, it may be possible that if flow j receives service in the inner round $(k_0 + k, g)$, then that service is also not included in the time interval under consideration. Hence, $Sent_i(t_1, t_2)$ and $Sent_j(t_1, t_2)$ can be evaluated as follows:

$$\begin{aligned} Sent_i(t_1, t_2) &= Sent_i(k_0) - Sent_i(k_0, f-1) \\ &\quad + \sum_{h=1}^{h=k-1} Sent_i(k_0 + h) + Sent_i(k_0 + k, g) \\ Sent_j(t_1, t_2) &= Sent_j(k_0) - Sent_j(k_0, f) \\ &\quad + \sum_{h=1}^{h=k-1} Sent_j(k_0 + h) + Sent_j(k_0 + k, g-1) \end{aligned}$$

Using Equations (4.2), (5.8) and (5.9) in the above and simplifying, we get,

$$\begin{aligned} Sent_i(t_1, t_2) = &\left(\frac{f}{p} Q^{max}(k_0)\right) UA_i^{max}(k_0) + k \sum_{h=1}^{h=k-1} UA_i^{max}(k_0 + h) \\ &+ \left(\frac{g}{p} Q^{max}(k_0 + k)\right) UA_i^{max}(k_0 + k) \\ &+ SC_i(k_0 + k, g) - SC_i(k_0, f-1) \end{aligned} \tag{5.25}$$

$$\begin{aligned}Sent_j(t_1,t_2) = {} & \left(\frac{f-1}{p}Q^{max}(k_0)\right) UA_j^{max}(k_0) + k\sum_{h=1}^{h=k-1} UA_j^{max}(k_0+h) \\ & + \left(\frac{g-1}{p}Q^{max}(k_0+k)\right) UA_j^{max}(k_0+k) \\ & + SC_j(k_0+k, g-1) - SC_j(k_0, f)\end{aligned} \tag{5.26}$$

Without loss of generality, we can assume that in the interval (t_1, t_2) flow i receives more service as compared to flow j. The normalized service for each of the flows can be obtained by dividing the above two equations by their respective weights. Subtracting the normalized service of flow j from that of flow i using Equations (5.25) and (5.26) we have,

$$\begin{aligned}\frac{Sent_i(t_1,t_2)}{w_i} - \frac{Sent_j(t_1,t_2)}{w_j} = {} & \frac{UA_i^{max}(k_0)Q^{max}(k_0)}{w_i\, p} \\ & + \frac{UA_j^{max}(k_0+k)Q^{max}(k_0+k)}{w_j\, p} + \frac{SC_i(k_0+k, g)}{w_i} \\ & - \frac{SC_i(k_0, f-1)}{w_i} + \frac{SC_j(k_0+k, g-1)}{w_j} - \frac{SC_j(k_0, f)}{w_j}\end{aligned}$$

Simplifying the above using Equations (3.2), (4.2), (5.4) and Corollary 1, the statement of the theorem is proved. ■

5.4 Work Complexity

Consider an execution of the PERR scheduler over n flows. The work involved in processing each packet at the scheduler involves two parts: enqueuing and dequeuing. Hence, the work complexity of a scheduler is defined as the order of time complexity, with respect to n of enqueuing and then dequeuing a packet for transmission [21,46]. Note that, n, the number of flows competing for a link can be of the order of tens of thousands of flows in backbone routers. Hence, it is desirable that the work complexity should be as independent as possible of n.

Theorem 5.4.1 *The worst-case work complexity of the PERR scheduler is* $O(\log p)$.

Proof: The time complexity of enqueuing a packet is the same as the time complexity of the *Enqueue* routine in Figure 4.2, which is executed whenever a new packet arrives at a flow. Identifying the flow at which the packet arrives is an $O(1)$ operation. If the *Active* flag is not set for the flow i, the *Ideal Allowance Utilization* for the flow is calculated which in turn determines the priority queue into which the flow should be added. Also, if this priority queue is of a higher priority than the priority queue which the PERR scheduler is serving, a flag is set to indicate that after completing the transmission of the current packet, the scheduler should start serving packets from the newly active flow. The addition of an item to the tail of a linked-list data structure and conditionally setting a flag are both $O(1)$ operations.

Let us now consider the time complexity of dequeuing a packet. Assume that the PERR scheduler is serving flows from the priority queue PQ_g. Note that, at least one packet is transmitted from each of the flows that are present in PQ_g. The operations involved in serving flows in this priority queue include determining the next flow to be served, removing this flow from the head of the priority queue and possibly adding it into some other priority queue or the ActiveList. All these operations can be executed in $O(1)$ time. Additionally the PERR scheduler may need to update certain per-flow variables which can be easily done in constant time. However, once the queue PQ_g is empty the PERR scheduler needs to determine the highest non-empty priority queue. To aid in this, the PERR scheduler maintains a linked list of the identifiers of all the non-empty priority queues. This linked list is sorted in descending order of priority with the head of the list pointing to the highest non-empty priority queue. The complexity of inserting a new identifier into this sorted linked list is $O(\log p)$ where p represents the total number of priority queues. To select the highest non-empty priority queue, the PERR scheduler simply has to read the identifier at the head of this sorted list which can be done in $O(1)$ time. Hence, the overall time complexity of this operation is $O(\log p)$. A similar situation arises when a flow is added into a priority queue which has a higher priority than the current priority queue being served by the PERR scheduler. However, if all the priority queues are empty it is an indication that the current round has come to an end. In this case, prior to the start of the subsequent round, the *Organizer* module has to classify all the flows present in the *ActiveList* into the p priority queues which requires $O(n)$ time. However, since each of the n flows is guaranteed to transmit at least one packet, the overall complexity of this operation is $O(1)$.

Note that, the PERR scheduler needs to sort the non-empty priority queues only in the two special cases discussed above, unlike the sorted-priority algorithms like WFQ and SCFQ where these sorting operations need to be executed prior to each packet transmission resulting in a $O(\log n)$ work complexity where n is the number of active flows. Also, since $n \gg p$, the work complexity of the PERR scheduler is always lower than that of the sorted-priority schedulers. Hence, the worst-case work complexity of the PERR scheduler is $O(\log p)$ resulting in an efficient hardware implementation. ■

Chapter 6. Simulation Analysis

In this chapter we present a detailed simulation-based evaluation of both the ERR and PERR schedulers in comparison with other schedulers of comparable efficiency. We first present a brief discussion on a recently proposed new measure of fairness which captures the *instantaneous* behavior of a scheduler. A more detailed presentation of this measure may be found in [41]. We then present some simulation results using real gateway traffic traces which compare the fairness characteristics of ERR and PERR based on this new metric with other efficient schedulers. In addition, we also compare the latency bounds of our schemes with these schedulers.

The rest of this chapter is organized as follows. Section 6.1 presents arguments for the need of a new measure of fairness which captures the instantaneous fairness of a scheduler. In Section 6.2 we describe the new fairness measure known as *Gini Index*. A more detailed description can be found in [41]. Section 6.3 presents a simulation-based evaluation of the Elastic Round Robin (ERR) and Prioritized Elastic Round Robin (PERR) schedulers with other guaranteed-rate schedulers of comparable efficiency. This section also presents a qualitative analysis of the observed results.

6.1 The Need for a New Measure of Fairness

As explained in earlier chapters of this dissertation, the fairness of scheduling algorithms is most commonly judged by the *relative fairness bound (RFB)* [24]. The RFB captures the maximum possible difference between the normalized service received by any two backlogged flows and therefore serves as a measure of fairness. The RFB of the ideally fair GPS scheduler, of course is zero. A related measure of fairness called the *absolute fairness bound (AFB)*, captures the upper bound on the difference between the normalized service received by a flow under the current scheduler being analyzed and that it would receive with the ideal GPS scheduler [6]. It has been shown in [43], that the AFB and RFB are related to each other by a simple relationship. Another fairness measure known as the *worst-case fair index*, is defined in [48]. It is used to analyze the Worst-Case Fair Weighted Fair Queuing (WF^{2}Q) scheduler in [27]. However, all these fairness measures including the RFB simply capture the worst-case behavior of any scheduler. They do not provide any insight into the actual quality of a fair scheduler. This is because a scheduler that rarely reaches the upper bound of relative fairness will achieve the same measure of fairness as another scheduler that frequently or almost always operates at the same upper bound. Hence, we also need an *instantaneous* measure of fairness that captures the fairness achieved by the scheduler at any given instant of time.

Note that, the RFB also fails to inform us of how a scheduler treats packets of one flow in comparison to those of another. Fairness, after all, is expected to be based on a comparison among the levels of service received by all the flows and not merely on the maximum difference in the normalized service received by flows. Figure 6.1(a) and (b) illustrate an example where the bars represent the service received by flows under two different schedulers A and B, during a certain interval of time in which all flows are backlogged. Assuming the weights associated with the flows are identical, the service received by each of the flows under the ideally fair GPS scheduler is illustrated as a separate bar. One may observe from the figures that scheduler B leads to a greater disparity in the levels of service received by the flows since scheduler A allows more flows to achieve service close to the ideally fair level. If the absolute and relative fairness bounds are exactly reached in this interval of time, note that both schedulers would lead to the same values for the RFB and the AFB, even though, the levels of service received by flows under scheduler A are closer to each other than with scheduler B.

Thus, measures of fairness based on an upper bound serve the excellent purpose of capturing the fairness characteristics of a scheduler in a single number. However, they do

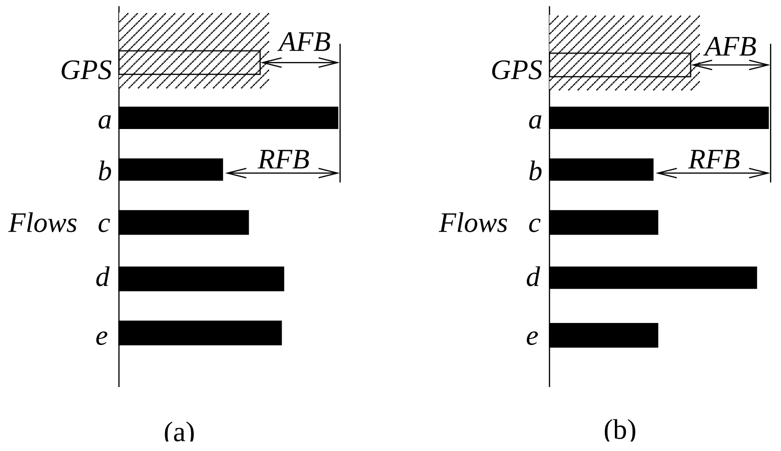


Figure 6.1: An illustration of the difference in the disparity in service received while the upper bounds of the relative fairness and absolute fairness measures are identical

not capture the overall behavior of the scheduler at all instants of time and also do not quite capture the characteristics of the distribution of the service among all the flows (the AFB only reports the maximum deviation from GPS for any flow while the RFB reports the maximum difference in service received by two different flows, but neither capture the overall fairness of the allocation among all the flows). This is addressed by the measure of *instantaneous* fairness described below based on measures of inequality used in the field of economics.

To measure the fairness at any instant of time, we also need to consider situations in real applications. The RFB is defined under the assumption that queues are continuously backlogged in the interval of interest. Such an assumption is rarely true in real networks. In networks with real traffic, flow states can change frequently from active state to idle state, or vice versa. However, existing measures of fairness have not taken this factor into account. To effectively guide the design of a fair scheduler, a fairness measure should also be able to capture the performance under the situation where flows change their states unpredictably.

6.2 Gini Index: A New Measure of Fairness

A recently proposed measure of fairness described in [41] addresses all of the above issues. There are two components to this measure: one of how to handle real traffic where flows are not always backlogged and the other of how to measure inequality in service received by the flows. We review these components briefly in this section; a detailed treatment and a more extensive rationale behind the approach may be found in [41].

6.2.1 Handling a Newly Backlogged Flow

In order to evaluate the fairness of a scheduler in its treatment of a newly backlogged flow, we need to first define the ideally fair way of doing this. We begin with an examination of the ideal but unimplementable GPS scheduling discipline.

Let $B(t)$ represent the set of backlogged flows at time t. Assume that the system starts at time $t = 0$.

Definition 6.2.1 *Let $V(t)$ represent the virtual time function [18, 49] (also known as the system potential) at time t. The virtual time is used to track the progress of the GPS scheduler and is computed as follows:*

$$V'(t) = \frac{dV(t)}{dt} = \left(\sum_{i \in B(t)} w_i \right)^{-1} \tag{6.1}$$

Hence, the service received by a backlogged flow under the GPS server in the time interval $(0, t)$ is given by $w_i V'(t)$. Intuitively, the virtual time represents the ideal fair normalized service that each flow should have received by time t.

Let us now consider a set of n flows served by a scheduling policy P. Consider a case in which the n flows have been backlogged since time t_1. One of these flows, flow i, changes its status from being backlogged to idle at time $t_2 > t_1$ and later becomes backlogged again at time $t_3 > t_2$. In order to accurately and meaningfully compare the service received by all the flows at time instants after t_3, it is necessary to assign an appropriate value of the normalized service received by flow i until t_3 so that the comparison is over the entire time interval (t_1, t_3). As mentioned earlier, a newly backlogged flow should neither be favored nor be punished for its idle period in the interval (t_2, t_3). Therefore, based on the discussion of the virtual time, the service received by flow i at time t should be $w_i V(t)$. However, if flow i has already received more service than the above amount of service before time t_2 while it was backlogged, then the total assumed service should be $Sent_i(0, t_2)$. This is because a flow that receives excess service should not be able to become idle and then immediately become backlogged again without being disadvantaged later for the excess service it received earlier. These concepts and similar arguments have also been made in [24, 26, 38, 41].

In evaluating the fairness of a specific scheduler it is necessary to keep track of the amount of service allocated by the above method. We borrow the method used in [41] where a per-flow state known as the *session utility* is defined for this purpose. This variable is independent of the scheduling discipline used by the scheduler and is defined as a function of time. Let $u_i(t)$ represent the session utility for flow i at time instant t. Assume that the system starts at time 0. During the period (t_1, t_2) that a flow is continuously backlogged, its session utility is updated as follows:

$$u_i(t_2) = u_i(t_1) + \frac{Sent_i(t_1, t_2)}{w_i} \tag{6.2}$$

We now discuss how to update the session utility of a flow that just becomes backlogged. Let flow i become newly backlogged or backlogged again at time t. Let $B(t-)$ represent the set of flows that are backlogged just prior to the time that flow i becomes backlogged. Our goal in assigning a session utility value to flow i at time t is to ensure that the comparison between session utilities of all the flows is being made as though the flows have all been backlogged for the same length of time. Accordingly flow i is assigned the following session utility value:

$$u_i(t) = \max\{u_i(t-), V(t)\} \tag{6.3}$$

With the above definition of the session utility, a newly backlogged flow can be treated as if it had been backlogged for the same length of time as all other flows. Therefore, with a measure which is based on session utility, it is possible to capture the fairness of a scheduler in its treatment of flows that are not always continuously backlogged.

6.2.2 The Gini Index

Various measures of inequality have been used in the field of economics for several decades in the study of social wealth distribution and many other economic issues of interest [50]. Some of these methods are related to the theory of majorization used in mathematics as a measure of inequality [51]. This theory has occasionally found use in research in computer networks in the fairness analysis of protocols [52]. The fairness measure proposed in [41] and adopted in this paper borrows from a related measure of inequality developed in the field of economics based on the concept of the *Lorenz curve* and *Gini index* [50].

Consider the problem of measuring the inequality among k quantities, $g_1 \leq g_2 \leq \cdots \leq g_k$. 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* [53], as shown in Figure 6.2(a). 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 measures the inequality amongst the k quantities [50].

In our case, we wish to measure the inequality in the session utilities of the backlogged flows at any given instant of time. The Gini index in our case, therefore, is the area between the Lorenz curve of the actual normalized service received and the Lorenz curve corresponding to the ideally fair GPS scheduler.

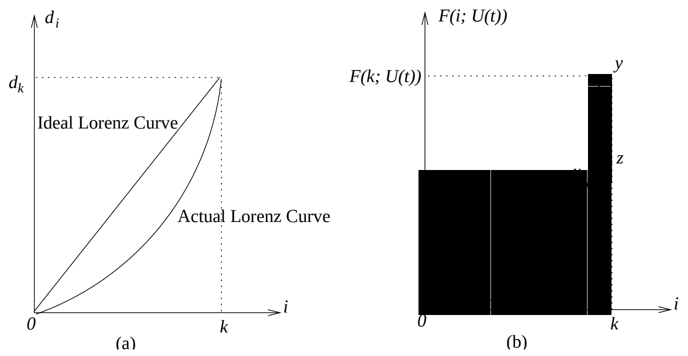


Figure 6.2: An illustration of the lorenz curve and Gini index in the measure of inequalities among (a) income distribution (b) session utilities in a packet scheduler

When the sum of the k quantities is the same as the sum in the case of perfect equality, the Lorenz curve always lies below the straight line corresponding to the Lorenz curve of the ideal equal case, as shown in Figure 6.2(a). However, the sum of the session utilities with a real scheduler is almost never exactly identical to the sum of the session utilities with the ideally fair GPS scheduler. Note that, in a work-conserving scheduler, only the sum of the total service delivered is identical to that in the ideally fair GPS scheduler; the sum of the session utilities is not identical to that in the GPS system. In the Lorenz curve for a work-conserving scheduler, when the sum of the k quantities is not the same as the sum in the case of perfect equality as with the GPS scheduler, a portion of the curve for the actual scheduler will lie below and another portion will lie above the straight line Lorenz curve for the GPS scheduler. This is illustrated in Figure 6.2(b). The sum of the shaded areas in the figure is the Gini index.

The computation of the Gini index is described formally as follows:

Definition 6.2.2 *Let $\mathbf{U}(t)$ represent the set of the session utilities of the flows at time instant t when served by a real scheduler and let $\mathbf{G}(t)$ denote a similar set which is obtained when* *the flows are served with the ideal GPS scheduler. Let* $u_{c_1}, u_{c_2}, \ldots, u_{c_k}$ *be the elements of the set* $\mathbf{U}(t)$*, such that* $u_{c_1} \leq u_{c_2} \leq \cdots \leq u_{c_k}$*. The* Lorenz Curve *of the set of session utilities* $\mathbf{U}(t)$ *is the function* $F(i; \mathbf{U}(t))$*, given by,*

$$F(i; \mathbf{U}(t)) = \sum_{j=1}^{i} u_{c_j}, 0 \leq i \leq k$$

The Gini index over the k *elements in* $\mathbf{U}(t)$ *is computed as:*

$$\sum_{i=1}^{k} \big| F(i; \mathbf{U}(t)) - F(i; \mathbf{G}(t)) \big| \tag{6.4}$$

As defined above, the closer the Lorenz curve of $\mathbf{U}(t)$ is to the curve of GPS, the smaller the Gini index is, and thus, the fairer the distribution of the session utilities. From the definition of the virtual time function, we know that the normalized service received by each flow at time t in the GPS system is equal to the virtual time, $V(t)$. Hence, the Gini index can be computed as :

$$\sum_{i=1}^{k} \big| F(i; \mathbf{U}(t)) - F(i; V(t)) \big| \tag{6.5}$$

With the Gini index, the fairness of a scheduler can be evaluated at each instant during the execution of the scheduler. A comparison of schedulers based on their Gini indices allows us to determine which scheduler achieves better fairness than others at each instant of time.

6.3 Simulation Results

In our simulation experiments, we compare the Gini index and latency bound of PERR with those of other efficient and fair schedulers such as DRR, ERR, Nested-DRR and Pre-order DRR. We also include the WFQ as a representative sorted-priority scheduler in our comparisons. We shall first present our simulation model.

6.3.1 Simulation Model

Figure 6.3 illustrates our simulation model. We model each flow as a *traffic generator* module. Each of these modules generates a packet stream with the packet length distribu-

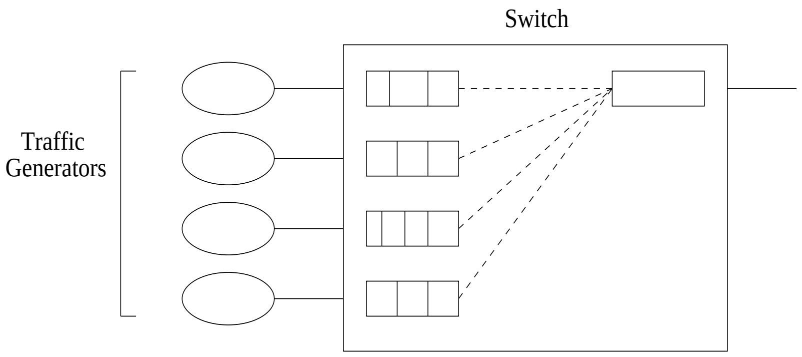


Figure 6.3: Simulation model

Table 6.1: Settings for traffic sources from router traces

Source	1	2	3	4	5
Router Abbr. [2]	ANL	APN	BUF	MEM	TXS
Interface	OC3	OC3	OC3	OC3	OC3
$L_{\min}$ (bytes)	28	29	28	32	32
$L_{\max}$ (bytes)	9,180	1,500	1,560	4,470	9,180
r_{avg} (10^6Bps)	0.63	1.4	1.45	0.39	2.1
Weight (w_i)	1.6	3.5	3.7	1	5.5
Link Capacity	6×10^6 Bps				
Total Time	50 seconds				

tion obtained from backbone router traces.

We used the traces provided by the National Laboratory for Applied Network Research

[2]The long names of routers are: Argonne National Laboratory(ANL), APAN(APN), University of Buffalo(BUF), University of Memphis(MEM) and Rice University(TXS)

[54]. Each generator is fed by a router trace with a random starting time. Table 6.1 shows the settings for this set of input traffic. The flow weights are set based on the average rate of each flow. Here we the set the weight of the slowest flow as 1, and weights of other flows are equal to the ratios of their average rate to the smallest rate. The switch maintains separate input queues for packets arriving from different sources. We assume that the packets arriving at all the input queues wish to leave the switch through the same output port, say port 0, thus creating a contention for the bandwidth on that link. The scheduling discipline employed by the switch for serving these input queues is varied. In our simulation we compare the instantaneous fairness of ERR and PERR with other efficient schedulers such as DRR, SRR, Nested-DRR and Pre-order DRR. We also include the popular WFQ scheduler, as a representative sorted-priority scheduler. The number of prioriy queues, p for the Pre-order DRR and PERR schedulers have both been set to 10.

6.3.2 Results with Backlogged Traffic

In our first set of experiments we extract the length of each packet from the router traces and simulate a scheduling system with continuously backlogged queues. Figure 6.4 shows the Gini index at periodic instants of time for the schedulers under consideration. Recall that the lower the Gini index, the more fair the algorithm. As is readily seen from the graph, ERR outperforms SRR and DRR by a huge margin. In fact ERR has substantially better fairness properties as compared to the Nested DRR scheduler. This illustrates that ERR has the best fairness properties among all existing $O(1)$ scheduling disciplines. PERR improves upon the fairness properties of ERR as illustrated by the extremely low value of its Gini index. Figure 6.4 shows that PERR displays better fairness than all other schedulers of equivalent complexity.

Figure 6.5 compares the Gini indices of the WFQ and PERR schedulers. The WFQ scheduler achieves slightly better fairness than PERR. However, note that WFQ is a sorted-

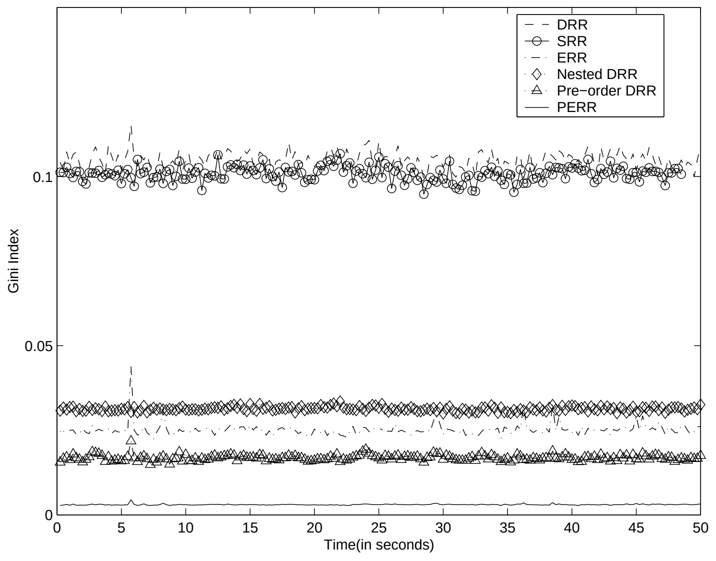


Figure 6.4: Gini indices of various efficient schedulers with backlogged traffic

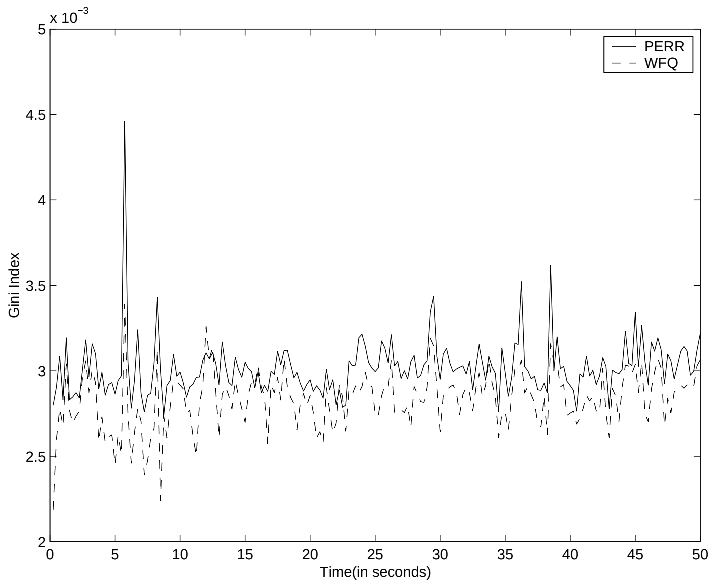


Figure 6.5: Gini indices of PERR and WFQ with backlogged traffic

priority scheduler and has a large work complexity which is proportional to the number of flows being served.

6.3.3 Results with Real Router Traces

In our next set of experiments, we allow that the flows are not always backlogged, while still using real router traces. Since we are more interested in the performance when the link is close to fully utilized, we set the link capacity such that the sum of average rates of all the flows is 98% of the link capacity. Figure 6.6 shows the average

length of arriving packets among all sessions during the simulation interval.

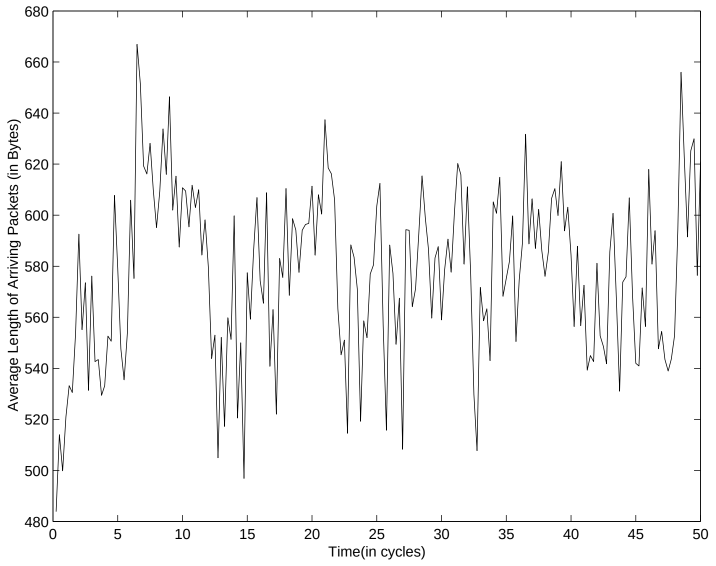


Figure 6.6: Average length of arriving packets

In Figures 6.7 to 6.10 we compare the Gini index of ERR with those of DRR, SRR, Nested-DRR and Pre-order DRR. For the sake of clarity, we plot each scheduler's Gini index on a separate graph, with that of the ERR scheduler plotted on each of the graphs. Once again, we find that ERR exhibits better fairness than DRR, SRR and Nested-DRR.

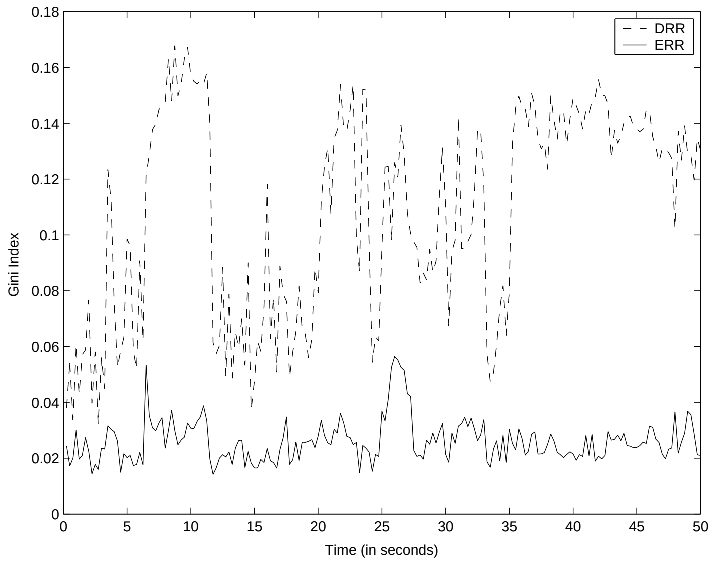


Figure 6.7: Comparison of Gini indices of DRR and ERR

Note that, in both SRR and DRR the quantum for each flow i, Q_i is w_iQ_{min} where Q_{min} represents the quantum for the flow with the lowest reserved rate. Note that, Q_{min} has to be greater than or equal to the size of the largest packet that may potentially arrive, M. As a result, irrespective of the actual size of the arriving packets both these schedulers on an average serve a quantum's worth of data from each flow in its round robin service opportunity. However, in reality a large percentage of the arriving packets are much smaller than M. As a result, these schedulers are unfair over short periods of time . On the contrary the ERR scheduler calculates the allowance of a flow based on the maximum normalized surplus count in the previous round. As a result, for the most part the allowance of a flow in ERR is much lower than the flow's quantum in DRR and SRR resulting in better fairness for the ERR scheduler.

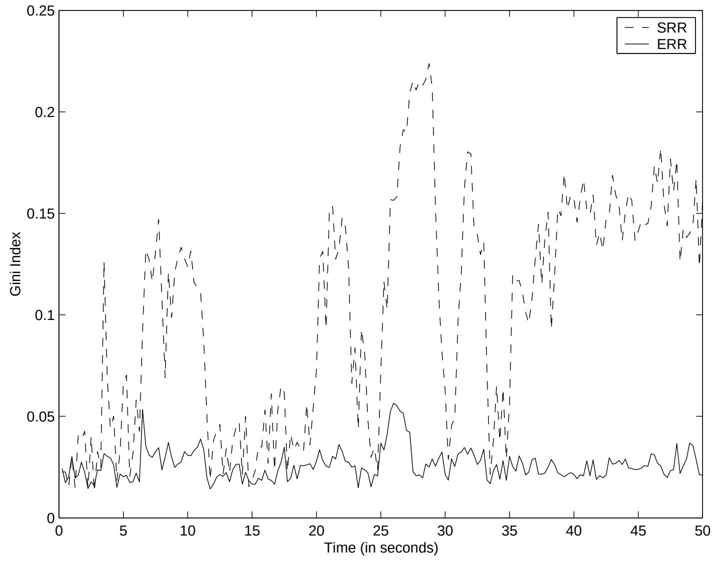


Figure 6.8: Comparison of Gini indices of SRR and ERR

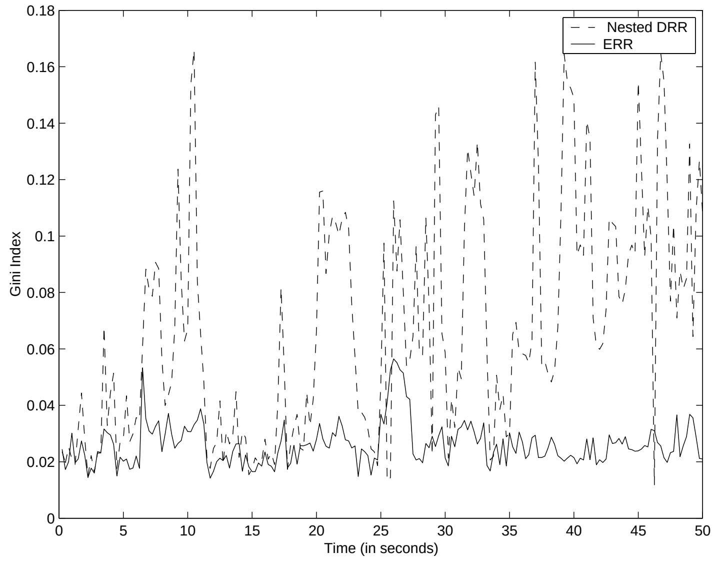


Figure 6.9: Comparison of Gini indices of Nested DRR and ERR

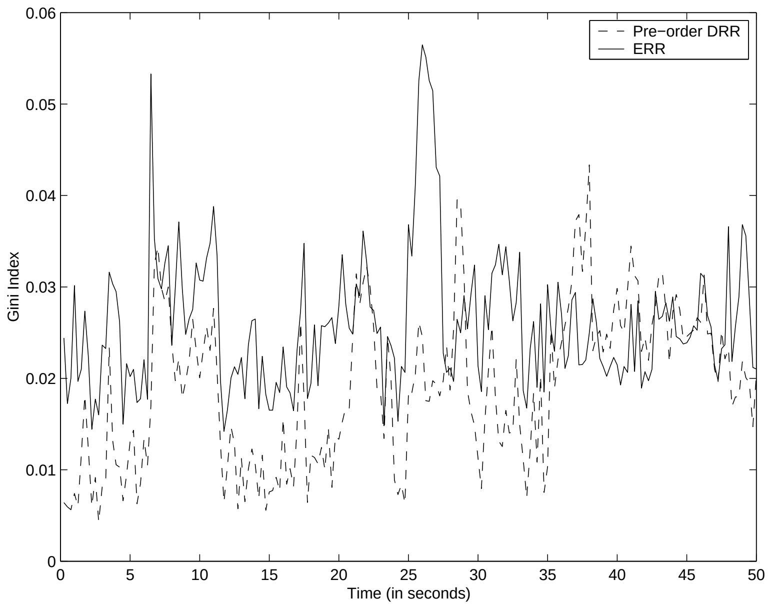


Figure 6.10: Comparison of Gini indices of Pre-order DRR and ERR

Since the quantum assigned to each flow in DRR is proportional to its reserved rate, flows with large reserved rates tend to have a very large quantum. As a result, DRR can be extremely unfair to flows with lower reserved rates over short durations of time. The Nested DRR eliminates some of the drawbacks of DRR by spliting each DRR round into multiple inner rounds and then scheduling the minimum quantum, Q_{min} from each eligible flow in the inner rounds. Even though this succeeds in alleviating the unfairness experienced by the low-rate flows it has an adverse effect on the fairness properties of the flows with higher reserved rates. Hence, the Gini index for Nested DRR is widely varying as seen in Figure 6.9. Finally, in Figure 6.10 we compare the Gini indices of ERR and Pre-order DRR. As seen from the plot, for a small percentage of the time Gini index of ERR is lower than that of Pre-order DRR. Also for the other time instants Pre-order DRR is only marginally better than ERR.

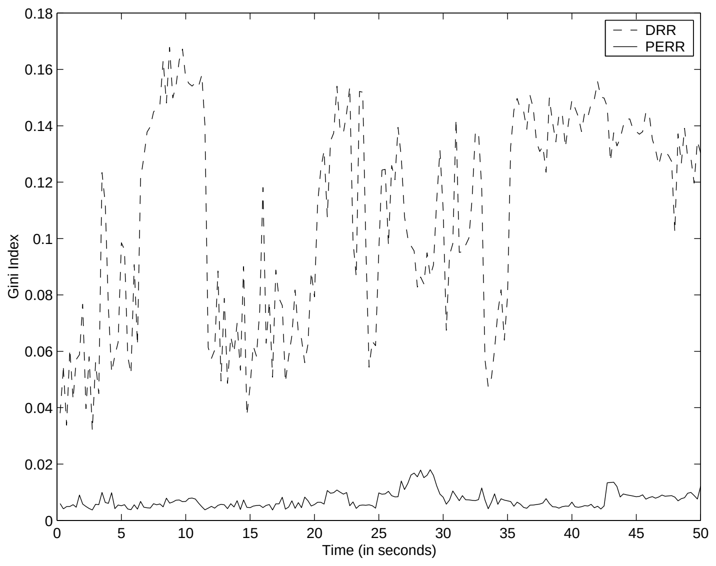


Figure 6.11: Comparison of Gini indices of DRR and PERR

Figures 6.11 to 6.15 illustrate the comparison of the Gini index of PERR with those of DRR, SRR, Nested-DRR and Pre-order DRR. As before, we plot each scheduler's Gini index on a separate graph, with that of the ERR scheduler plotted on each of the graphs. Once again, we find that the PERR scheduler displays a lower Gini index than any of the other schedulers of equivalent complexity at almost all instants of time. Finally, Figure 6.16 compares the Gini indices of the PERR and WFQ schedulers. It is seen PERR is almost as fair as the WFQ scheduler. However, WFQ is a sorted-priority scheduler and hence suffers from a large implementation complexity. PERR on the higher hand has a low work complexity which is independent of the number of flows.

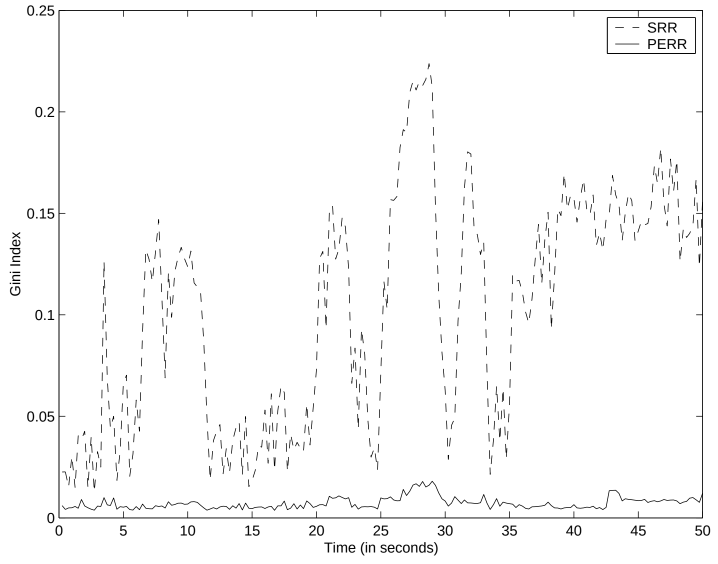


Figure 6.12: Comparison of Gini indices of SRR and PERR

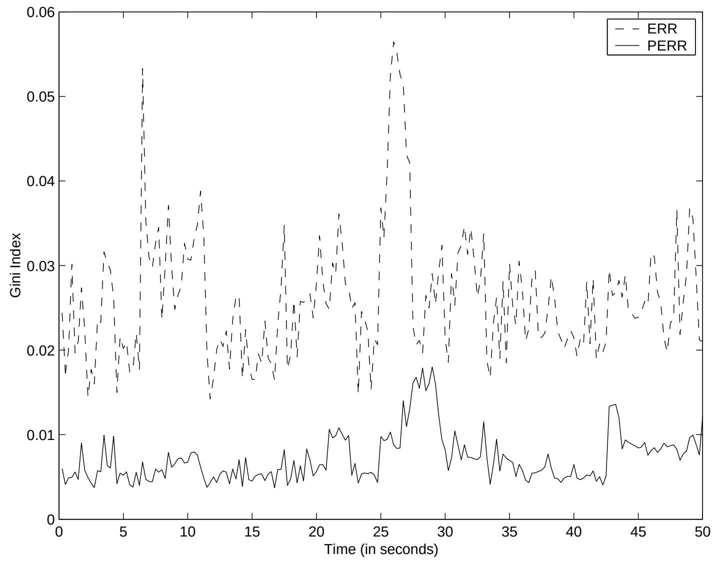


Figure 6.13: Comparison of Gini indices of DRR and PERR

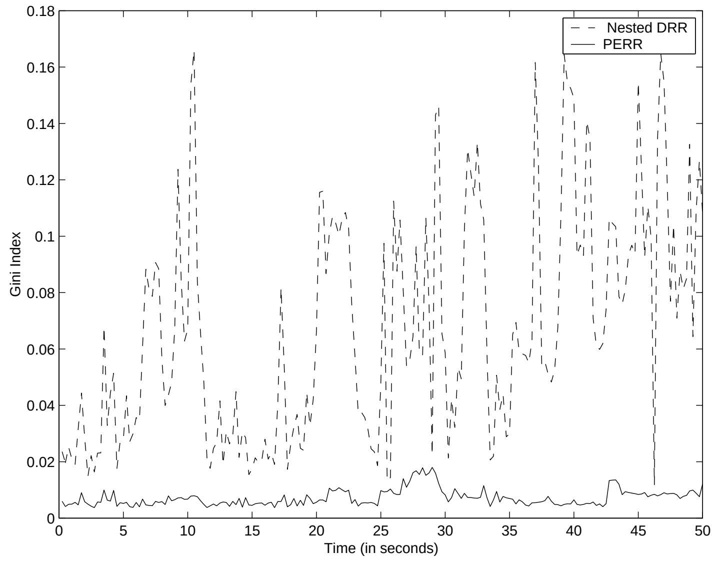


Figure 6.14: Comparison of Gini indices of SRR and PERR

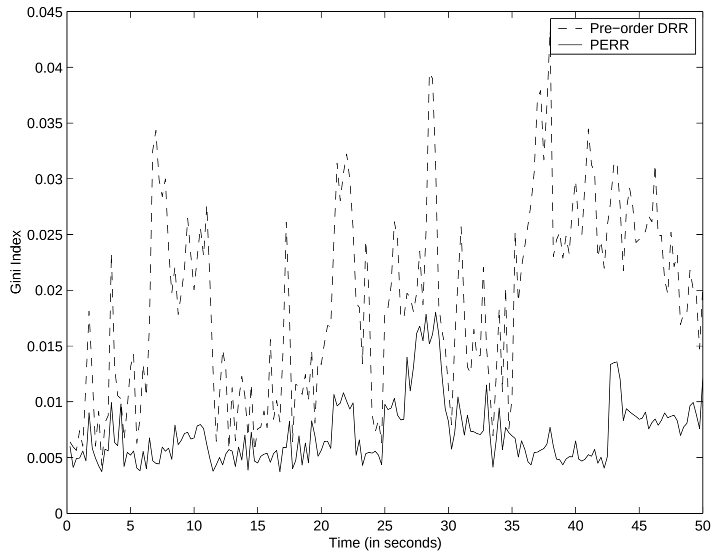


Figure 6.15: Comparison of Gini indices of Pre-order DRR and PERR

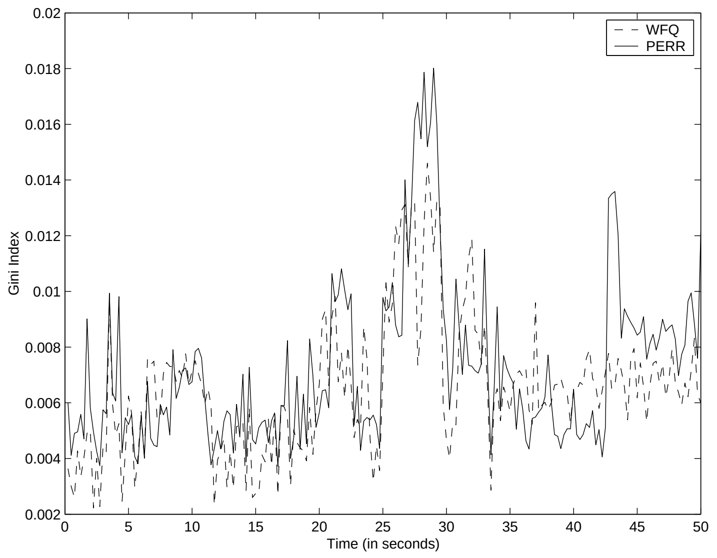


Figure 6.16: Comparison of Gini indices of WFQ and PERR

6.3.4 Comparison of Latency Bounds

We now compare the latency bounds of the router traces in Table 6.1 for the PERR and ERR schedulers with that of the other efficient schedulers such as DRR, SRR, Nested-DRR and Pre-order DRR. As before we include the WFQ scheduler as a representative sorted-priority scheduler. Note that the latency bounds of all these schedulers are summarized in Table 7.1.

Figure 6.17 illustrates the latency bounds of these schedulers for the five router traces that we use in our simulations. As is readily seen from the graph, the latency bound of ERR is much lower that that of the DRR and SRR schedulers. In addition, ERR even outperforms the Nested-DRR and Pre-order DRR schedulers. It is also evident that the latency bound of the PERR scheduler is significantly lower that all the other efficient schedulers. In fact, the bound for PERR is almost equal to that of the $O(\log n)$ WFQ scheduler.

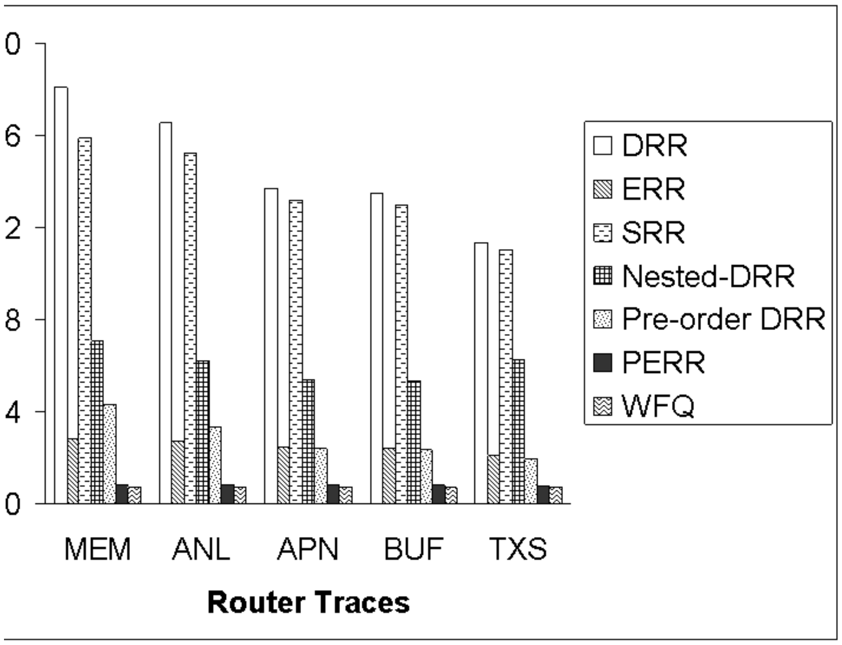


Figure 6.17: Comparison of latency bounds of various schedulers

Chapter 7. Concluding Remarks

In the emerging integrated-services packet-switched networks, fair packet scheduling algorithms in switches and routers play a critical role in providing the Quality-of-Service (QoS) guarantees required by real-time applications. In this dissertation, we have considered the problem of fair and efficient scheduling of packets to meet these QoS objectives. We have also addressed the requirements and the constraints imposed on a scheduling discipline used in wormhole switches, popular in the interconnection networks for parallel systems. In this dissertation, we first presented a new fair, efficient, simple and low-latency scheduling discipline called Elastic Round Robin (ERR). ERR was originally designed to satisfy the unique requirements of wormhole switching. We have shown that ERR can be easily adapted for scheduling best-effort and guaranteed-rate traffic in the Internet. We have also provided a detailed analysis of the performance and fairness properties of the ERR scheduler. We have shown that the work complexity of ERR is O(1) and therefore, ERR can be easily implemented in networks such as the Internet, with large number of flows. We have proved analytically the fairness properties of ERR. In particular, we show that ERR satisfies Golestani's [24] definition of throughput fairness, i.e., the difference in the bandwidths allocated to any two backlogged flows in any time interval is bounded by a small constant. The relative fairness bound of ERR has an upper bound of $3m$, where m is the size of the largest packet that actually arrives during the execution of ERR. While fairness is an intuitively desirable goal, its practical relevance is in the bound on the latency that fair schedulers are able to provide. This latency, as defined for Latency Rate $\mathcal{L}R$ servers in [38], has a direct bearing on the size of the playback buffers needed at the receivers for real-time communications. We have shown that ERR belongs to the the class of $\mathcal{LR}$-servers and have also evaluated the upper bound on the latency experienced by a flow served by the ERR scheduler. Our analysis reveals that ERR has better fairness characteristics and a significantly better latency bound in comparison to other scheduling disciplines of equivalent complexity such as DRR and SRR.

We also identified the problems associated with the frame-based schedulers. Since ERR is inherently a frame-based scheduling algorithm, it suffers from the limitations of all round robin schedulers such as (i) bursty transmission and (ii) the inability of the flows lagging in service to receive precedence over the flows that have received excess service. In the latter part of this dissertation, we have presented a novel scheduling discipline called Prioritized Elastic Round Robin (PERR) which rearranges the sequence in which packets are transmitted in each round of the ERR scheduler. This is achieved through the addition of a priority queue module consisting of p priority queues. An *Organizer* module dynamically classifies the active flows into these priority queues. We have analytically shown that PERR has a low work complexity of $O(\log p)$ which is independent of the total number of flows, resulting in a simple and efficient implementation. We also prove that PERR belongs to the class of $\mathcal{LR}$ servers and also evaluate an upper bound on its latency using a novel technique based on interpreting the PERR scheduler as an instance of the Nested Deficit Round Robin algorithm. Our analysis also reveals that PERR has better fairness characteristics and a significantly lower latency bound in comparison to other scheduling disciplines of equivalent work complexity such as DRR, SRR, ERR, Nested-DRR and Pre-order DRR.

We further study the fairness properties of ERR and PERR using a recently proposed measure of instantaneous fairness known as the Gini index that seeks to more accurately capture the fairness of a scheduler. Using real router traffic traces, we present simulation results that demonstrate that ERR achieves better fairness characteristics than other scheduling disciplines of equivalent complexity such as DRR and SRR. In fact, the Gini index of ERR is lower than that of Nested-DRR and almost equal to that of the Pre-order DRR scheduler. PERR improves upon ERR and has better fairness properties than all other efficient scheduling algorithms. In fact, our results show that PERR is almost as fair as WFQ, a popular sorted-priority scheduler.

The rest of the section is organized as follows. Section 7.1 presents a detailed comparison of the fairness and performance characteristics of the ERR and PERR scheduler in comparison with other popular guaranteed-rate scheduling disciplines. A tabulated summary of the properties is provided. Section 7.2 discusses various other situations where the ERR and PERR scheduling disciplines can be utilized.

7.1 Comparison of ERR and PERR with Other Schedulers

In this section we present a detailed comparison of the the fairness and performance characteristics of the ERR and PERR scheduler which were evaluated in Chapters 3 and 5 with other well-known guaranteed-rate schedulers.

Table 7.1 summarizes the work complexity, fairness and latency bounds of several guaranteed-rate scheduling disciplines that belong to the class of ($\mathcal{LR}$) servers. It is important to note that not all of these schedulers satisfy the unique requirement imposed by wormhole switching. Table 7.1 makes this distinction by providing a column which indicates the applicability of a scheduling discipline to workhole networks. In this table, n represents the number of active flows and p represents the number of priority queues in Pre-order DRR and PERR. The peak rate of the output link is denoted by r. M is the size of the largest packet that may potentially arrive during the execution of a scheduling algorithm. Recall that m is the size of the largest packet that *actually* arrives during the execution of the scheduler. Usually, in most networks including the Internet, $M \gg m$ since the majority of packets are much smaller than the largest possible packet [44, 45]. The properties of all the frame-based scheduling disciplines are derived in [49]. The latency bounds of DRR and Pre-order DRR have been analyzed in [55] and [56, 57]. The appendix at the end of the dissertation includes the latency analysis of both these scheduling disciplines. Note that, the latency bounds evaluated in the appendix are tighter than the previously known bounds of DRR and Pre-order DRR. The expression for the latency bound of Nested-DRR, as stated in [40], is extremely complex and hence does not allow for an easy comparison with the other schedulers under consideration. Hence, in order to gain a quick understanding of the differences in the latency bounds of Nested-DRR and the other schedulers, in our comparison we include the latency bound of Nested-DRR at two boundary conditions. In the first case, we consider the latency bound of a flow whose reserved rate is much lower than that of the other flows sharing the same output link ($\rho_i \ll \rho_j, \forall j \in n, j \neq i$). In the second case, we consider the opposite end of the spectrum, i.e., the latency bound of a flow whose reserved rate is much greater than the other flows multiplexed on the same link ($\rho_i \gg \rho_j, \forall j \in n, j \neq i$). In [40], an expression for the latency of a low-rate flow has been derived and it has also been shown that the latency of a high-rate flow is marginally lower than that of the DRR scheduler. For simplicity, we assume the latter to be equal to DRR.

Table 7.1: Comparison of ERR and PERR with other guaranteed-rate scheduling disciplines

Scheduling Algorithms	Complexity	Fairness	Latency Bound for flow i	Applicable to Wormhole Networks
GPS [18]	—	0	0	—
Weighted Fair Queuing [16]	$O(n)$	$3m$	$\frac{m}{r} + \frac{m}{\rho_i}$	—
Self Clocked Fair Queuing [24]	$O(\log n)$	$2m$	$\frac{(n-1)m}{r} + \frac{m}{\rho_i}$	—
Virtual Clock [19]	$O(\log n)$	∞	$\frac{m}{r} + \frac{m}{\rho_i}$	—
Frame-based Fair Queuing [26]	$O(\log n)$	$2M + m$	$\frac{m}{r} + \frac{m}{\rho_i}$	—
Deficit Round Robin [21]	$O(1)$	$M + 2m$	$\frac{1}{r} \quad (W - w_i)Q_{min} + (m-1) \quad \frac{W}{w_i} + n - 2$	—
Surplus Round Robin [34–36]	$O(1)$	$M + 2m$	$\frac{1}{r}\{(W - w_i)Q_{min} + (m-1)(n-1)\}$	—
Elastic Round Robin [46, 47]	$O(1)$	$3m$	$\frac{1}{r}\{(W - w_i)m + (m-1)(n-1)\}$	√
Nested-DRR [40]	$O(1)$	$M + 2m$	*Low-rate:* $\frac{1}{r} \quad (n-1)Q_{min} + (m-1) \quad \frac{W}{w_i} + n - 2$ *High-rate:* $\frac{1}{r} \quad (W - w_i)Q_{min} + (m-1) \quad \frac{W}{w_i} + n - 2$	—
Pre-order DRR [42]	$O(\log p)$	$\frac{2M}{p} + 2m$	$\frac{1}{r} \quad \frac{(W - w_i)Q_{min}}{p} + (m-1) \quad \frac{W}{w_i} + n - 2$	—
Prioritized ERR [39]	$O(\log p)$	$\frac{2m}{p} + 2m$	$\frac{1}{r} \quad \frac{(W - w_i)m}{p} + (m-1)(n-1)$	√

The GPS scheduler visits each active flow in a round-robin fashion, and serves an infinitesimally small amount of data proportional to the reserved rate of the flow [18]. Using this fluid model, the GPS scheduler is able to ensure that over any interval of time however small, the normalized difference between the service received by any two active flows is exactly zero. The RFB of GPS, therefore, is zero. The latency of GPS is also zero since a newly active flow begins receiving service instantaneously at the guaranteed rate. Recall that GPS is an ideal but not an implementable scheduler. The sorted priority schedulers such as Weighted Fair Queuing (WFQ) [16], Self-clocked Fair Queuing (SCFQ) [24], Frame-based Fair Queuing (FBFQ) [26] and Virtual Clock [19] all have a low value of the latency bound. Virtual Clock has an RFB of infinity and therefore, cannot be considered to be a fair scheduler. The RFB of the WFQ scheduler is $3m$ and thus, has better fairness However, WFQ requires the simulation of the ideal GPS scheduler on the side and hence has a very large work complexity of $O(n)$. SCFQ does not require simulation of GPS in parallel and also has a lower value of the RFB. However, the latency bound of SCFQ is much greater than that of WFQ. FBFQ achieves the same latency bound as that of WFQ and is also more efficient. The price paid is in the slightly higher RFB. In addition, FBFQ also requires periodic re-calibration of the virtual time, and also has a work complexity of $O(\log n)$ rendering it less efficient than ERR or DRR. In fact, the work complexity of all these sorted priority schedulers is a function of the number of the active flows, n. As a result, these schedulers are inefficient to implement in high-speed hardware switches.

In fact, only ERR, DRR, SRR and Nested-DRR have a work complexity of $O(1)$. However, as explained earlier, excluding ERR, none of these schedulers are ideally suitable for use in wormhole networks, where the length of time a packet occupies the link is not known before a decision to transmit the packet is made. On the other hand ERR can be readily used in wormhole networks, in addition to being perfectly suitable for achieving fair scheduling in Internet routers. In addition, Table 7.1 shows that ERR has better fairness properties and a lower latency bound as compared to other scheduling disciplines of equivalent complexity, especially considering that M is typically much greater than m.

As discussed earlier in Chapter 4, ERR, DRR, SRR and Nested-DRR are all round-robin schedulers and hence suffer from the characteristic limitations of all round robin schedulers. Both Pre-order DRR and PERR overcome these drawbacks in the DRR and ERR schedulers respectively and also have a low work complexity of $O(\log p)$ which is independent of the number of flows. As explained earlier, the basic principle of the PERR algorithm is similar to the Pre-order DRR algorithm [42]. The Pre-order DRR scheduler alters the transmission sequence of the packets in each DRR round based on the quantum utilization of each flow. The PERR scheduler similarly changes the sequence in which packets are transmitted in an ERR round depending on the utilization of the maximum possible allowance of each flow in that round.

There is however an important difference between these two schedulers. At the start of a round, the Pre-order DRR scheduler has to classify all the packets that will be transmitted by the active flows in that round into the priority queues prior to starting the transmission of the packets. On the contrary, the PERR scheduler simply has to classify the flows present in the *ActiveList* into its priority queues before the start of the round. It has been shown in [47] that ERR has a couple of important advantages over DRR. Now since the PERR and Pre-order DRR algorithms are modifications of ERR and DRR, respectively, PERR inherits those advantages over Pre-order DRR. The following lists these advantages:

- *Lower Buffer Requirement :* Since the priority queues in PERR simply need to contain the flow identifiers they can be much smaller in size as compared to those in Pre-order DRR which have to buffer all the packets that will be transmitted in the current round.

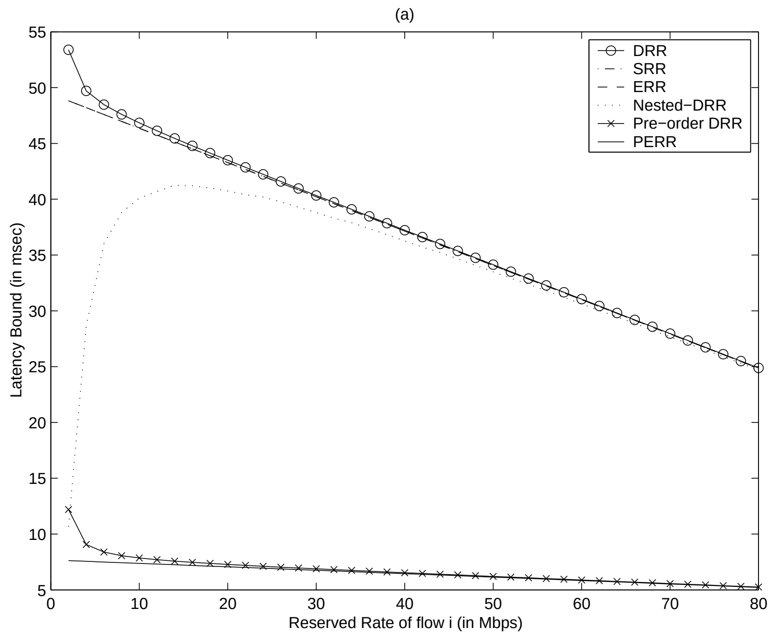
- *Reduced Round Start Delay :* Classifying all the packets that are to be transmitted in the round into the priority queues requires considerably more time than simply sorting the flow identifiers. Thus, the delay incurred by the PERR scheduler at the start of the round is much less in comparison to that incurred by the Pre-order DRR scheduler.

- *Improved Latency and Fairness Characteristics:* Table 7.1 illustrates that PERR has better fairness properties and a lower latency bound than Pre-order DRR, especially considering that M is typically much greater than m.

- *Adaptability in other contexts:* Unlike DRR and Pre-order DRR, the PERR scheduler does not require the knowledge of the transmission time of each packet. As a result, the scheduler can be used in other networks such as wormhole networks, where the transmission time of a packet depends not only on the size of the packet but also the downstream congestion.

Note that, the latency bound of a $\mathcal{LR}$ server is an extremely important QoS parameter since it has a direct influence of the the size of the playback buffers needed at the receiver for real-time communication applications. In order that the reader can fully appreciate the improvement in the latency characteristics of PERR, we compare the latency bound of PERR with other efficient scheduling disciplines of equivalent work complexity such as DRR, ERR, SRR, Nested-DRR and Pre-order DRR within the context of an example. Note that, for this comparison we use the actual latency bound for Nested-DRR as proved in [40].

Let us assume that a total of 100 flows are multiplexed on an output link with a transmission rate, r of 150 Mbps. Assume that M is equal to 576 bytes, equal to the minimum value of the Maximum Transmission Unit (MTU) required of all networks. Assume that ρ_{min} is equal to 0.1Mbps and that the output link is completely utilized, i.e. $\sum_{i=1}^{n} \rho_i = r$. Note that, this implies that the sum of all the weights, W, is equal to $150/0.1 = 1500$. Let the number of priority queues in the priority queue module of Pre-order DRR and PERR be equal to 10, i.e., $p = 10$. We compare the latency bounds of the afore-mentioned schedulers for flow i as a function of its reserved rate, ρ_i, for two values of m : (a) $m = M$, (b) $m = M/2$. Figure 7.1 illustrates a plot of these latency bounds of flow i for both the values of m. Note that, latency bounds of all the schedulers under consideration depend



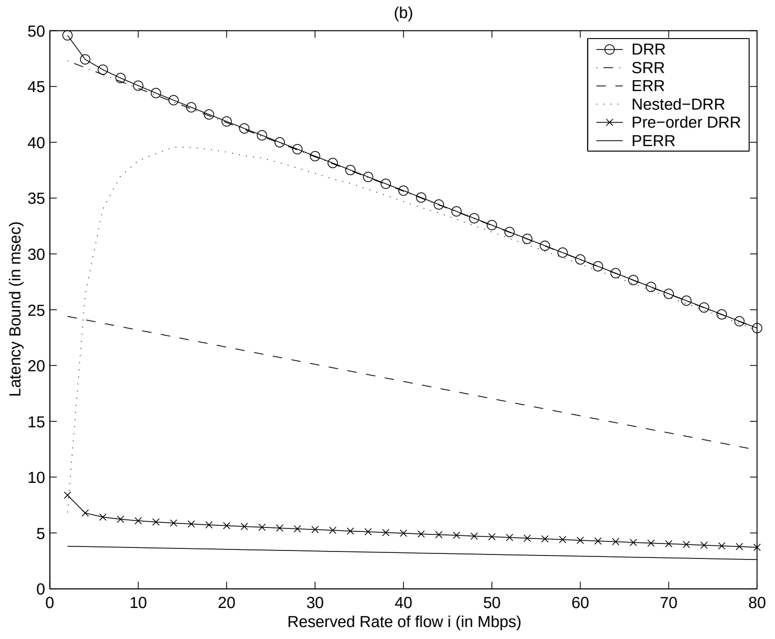


Figure 7.1: Comparison of the latency bound of PERR and ERR with other schedulers of equivalent complexity

on the sum of the weights of all the flows but not on the distribution of the weights among all the flows other than flow i. Therefore, the weights of the flows other than flow i are not discussed in the context of this illustration. From Figure 7.1, it can be easily seen that PERR has the lowest latency bound among all the scheduler under consideration. The improvement in the latency of PERR is even more apparent when $m < M$.

7.2 Other Applications of ERR and PERR

Even though ERR was originally designed for wormhole networks, it can be used in a wide variety of contexts whenever there is a shared resource that needs to be allocated fairly among multiple requesting entities. In some of these contexts its unique properties relevant to wormhole switching are critical, and in some others, its advantages derive from its simplicity, better fairness and better performance characteristics. Note that, since PERR is based on the ERR scheduling discipline it too inherits these advantages of ERR. In this section we present a few scenarios where the ERR and PERR schedulers can be put to use to achieve improved performance. Note that, in the rest of this section we list several possible applications of ERR. However, the ERR scheduler can be readily replaced by PERR in all of these instances.

For example, in wormhole networks one may define a flow as the stream of packets belonging to the same virtual channel, in which case, ERR can be used to achieve fairness among virtual channels in the forwarding of flits to the output link. ERR can also be used in the forwarding of packets from the input buffers to the output buffers of switching elements in networks. Further, the fact that ERR does not require the knowledge of the length of a packet before making a scheduling decision makes it a suitable candidate in an ATM network transmitting IP packets over AAL5, where the end of the packet is not known until the last ATM cell of the packet arrives.

Fair scheduling deals with the problem of partitioning the bandwidth on a single output

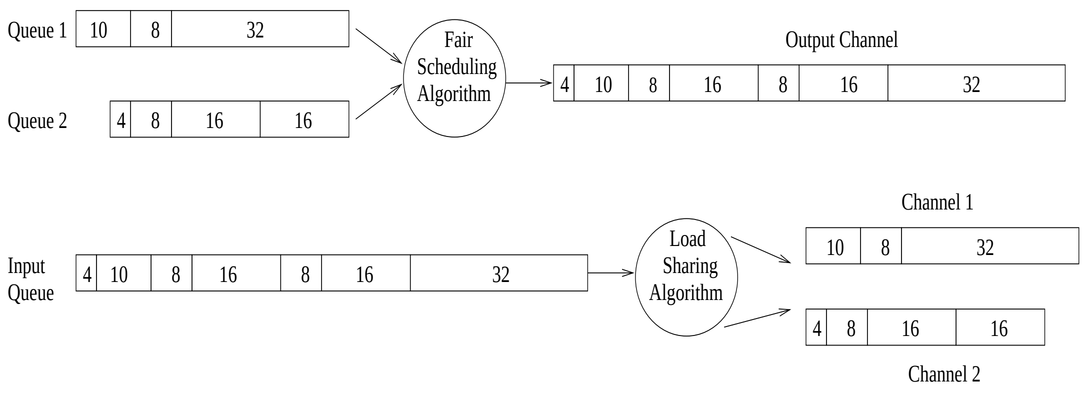


Figure 7.2: Example of load sharing using ERR

channel equally from a set of input queues which feed that channel. On the other hand, load sharing deals with the problem of partitioning the traffic arriving on a single input queue equally among a set of output channels [58]. Intuitively, one can see that the above mentioned two problems are complementary in nature. Figure 7.2 helps make this idea clear. The ERR algorithm works in the same way in the load sharing context as in the case of fair scheduling. This is explained with the help of Figure 7.2. Initially the surplus counts of the channels 1 and 2 are both initialized to zero. Channel 1 is first selected for service, i.e. for transmitting packets and its allowance is calculated using Equation (2.3). The ERR load sharing algorithm will keep transmitting packets on this channel, if the total number of flits transmitted so far in the current round is less than its allowance. A packet of length 32 flits is transmitted on channel 1, at the end of which the surplus count is calculated as 31 flits using Equation (2.1). Channel 2 is then selected for service. The operation continues to proceed in a similar manner. It is easy to see that the ERR allocates the bandwidth fairly among the output channels.

ERR can also be a solution in token ring networks where the bandwidth of the ring has to be shared among multiple sources. ERR, similarly, can be used to efficiently arbitrate access to a busy shared bus. The lower start-up latency of ERR among similarly efficient algorithms is especially useful here in improving the latency of short control messages. Finally, ERR is particularly relevant to the problem of job scheduling in operating systems, where multiple processes are competing for limited CPU cycles.

Bibliography

[1] P. Baran, “On distributed communication networks,” *IEEE Transactions on Communication Systems*, March 1964.

[2] D. Bertsekas and R. Gallager, *Data Networks*, Prentice Hall, Upper Saddle River, NJ, 2nd edition, 1991.

[3] V. P. Kumar, T. V. Lakshman, and D. Stiliads, “Beyond best effort: Router architectures for the differentiated services of tomorrow’s internet,” *IEEE Communications Magazine*, vol. 36, no. 5, pp. 152–164, May 1998.

[4] D. C. Stephens, J. C. R. Bennett, and H. Zhang, “Implementing scheduling algorithms in high-speed networks,” *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 6, pp. 1145–1158, June 1999.

[5] Cisco Systems Inc., “Cisco 12016 Gigabit Switch Router: Application Note,” 1999.

[6] S. Keshav, *An Engineering Approach to Computer Networks*, Addison-Wesley Publishing Company, Reading, MA, 1997.

[7] R. Cruz, “A calculus for network delay. i. network elements in isolation,” *IEEE Transactions on Information Theory*, vol. 37, pp. 114–131, January 1991.

[8] H. Sethu, C. B. Stunkel, and R. F. Stucke, “IBM RS/6000 SP large system interconnection network topologies,” in *Proceedings of the International Conference on Parallel Processing*, Minnesota, MN, August 1998.

[9] C. B. Stunkel, “The SP2 high-performance switch,” *IBM Systems Journal*, vol. 34, no. 2, pp. 185–204, February 1995.

[10] W. J. Dally and C. L. Seitz, “The torus routing chip,” *Journal of Distributed Computing*, vol. 1, no. 3, pp. 187–196, October 1986.

[11] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks: An Engineering Approach*, IEEE Computer Society Press, Los Alamitos, CA, 1997.

[12] J. Ding and L. N. Bhuyan, “Evaluation of multi-queue buffered multistage interconnection networks under uniform and non-uniform traffic patterns,” *International Journal of Systems Science*, vol. 28, no. 11, 1997.

[13] M. Katevenis, P. Vatsolaki, and A. Efthymiou, “Pipelined memory shared buffer for VLSI switches,” in *Proceedings of ACM SIGCOMM*, Cambridge, MA, August 1995, pp. 39–48.

[14] W. J. Dally, "Virtual channel flow control," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, no. 3, pp. 194–205, March 1992.

[15] H. Sethu, H. Shi, S. S. Kanhere, and A. B. Parekh, "A round-robin scheduling strategy for reduced delays in wormhole switches with virtual lanes," in *Proceedings of the International Conference on Communications in Computing*, Las Vegas, NV, June 2000, pp. 275–278.

[16] A. Demers, S. Keshav, and S. Shenker, "Design and analysis of a fair queuing algorithm," in *Proceedings of ACM SIGCOMM*, Austin, September 1989, pp. 1–12.

[17] D. Verma, D. Ferrari, and H. Zhang, "Guaranteeing delay jitter bounds in packet switched networks," in *Proceedings of Tricomm*, April 1991, pp. 35–43.

[18] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control—the single node case," in *Proceedings of IEEE INFOCOM*, Florence, Italy, May 1992, pp. 915–924.

[19] L. Zhang, "Virtual clock: A new traffic control algorithm for packet switching networks," in *Proceedings of ACM SIGCOMM*, Philadelphia, PA, September 1990, pp. 19–29.

[20] N. Figueira and J. Pasquale, "An upper bound on delay for the virtual clock service discipline," *IEEE/ACM Transactions on Networking*, vol. 3, no. 4, pp. 399–408, August 1995.

[21] M. Shreedhar and G. Varghese, "Efficient fair queuing using deficit round-robin," *IEEE Transactions on Networking*, vol. 4, no. 3, pp. 375–385, June 1996.

[22] A. Kumar and J. Kleinberg, "Fairness measures for resource allocation," in *Proceedings of the Symposium on Foundations of Computer Science*, November 2000, pp. 75–78.

[23] S. Golestani, "A framing straregy for congestion management," *IEEE Journal on Selected Areas in Communications*, vol. 9, pp. 1064–1077, September 1991.

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

[25] J. A. Cobb, M. G. Gouda, and A. El-Nahas, "Time-shift scheduling—fair scheduling of flows in high-speed networks," *IEEE Transactions on Networking*, vol. 6, no. 3, pp. 274–285, June 1998.

[26] D. Stiliadis and A. Varma, "Efficient fair queuing algorithms for packet-switched networks," *IEEE Transactions on Networking*, vol. 6, no. 2, pp. 175–185, April 1998.

[27] J. C. R. Bennett and H. Zhang, "WF^{2}Q : Worst-case fair weighted fair queueing," in *Proceedings of IEEE INFOCOM*, San Francisco, CA, March 1996, pp. 120–128.

[28] J. Nagle, “On packet switches with infinite storage,” *IEEE Transactions on Communications*, vol. 35, no. 4, April 1987.

[29] M. Shreedhar, “Efficient fair queuing using deficit round-robin,” M.S. thesis, Department of Computer Science, Washington University, St. Louis, 1996.

[30] Hui Zhang, “Service disciplines for guaranteed performance service in packet-switched networks,” in *Proceedings of the IEEE*, October 1995, vol. 8, pp. 1374–1396.

[31] M. Pirvu, L. Bhuyan, and N. Ni, “The impact of link arbitration on switch performance,” in *Proceedings of the Fifth Symposium on High-Performance Computer Architecture*, Orlando, FL, January 1999.

[32] S. Keshav, “On the efficient implementation of fair queuing,” *Journal of Internetworking Research and Experience*, vol. 2, no. 3, pp. 3–26, September 1990.

[33] M. Shreedhar and G. Varghese, “Efficient fair queueing using deficit round robin,” in *Proceedings of ACM SIGCOMM*, Boston, MA, September 1995.

[34] S. Floyd and V. Jacobson, “Link-sharing and resource management models for packet networks,” *IEEE Transactions on Networking*, vol. 3, no. 4, pp. 365–386, August 1995.

[35] S. Floyd, “Notes on class-based-queueing and guaranteed service,” *Unpublished Notes: http://www.aciri.org/floyd/cbq.html*, July 1995.

[36] H. Adiseshu, G. Parulkar, and G. Varghese, “A reliable and scalable striping protocol,” in *Proceedings of ACM SIGCOMM*, Palo Alto, CA, August 1996, pp. 131–141.

[37] P. Goyal, H. M. Vin, and H. Cheng, “Start-time fair queueing: A scheduling algorithm for integrated services packet switching networks,” *IEEE Transactions on Networking*, vol. 5, no. 5, pp. 690–704, October 1997.

[38] D. 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.

[39] S. S. Kanhere and H. Sethu, “Prioritized elastic round robin: An efficient and low-latency packet scheduler with improved fairness,” *submitted to Computer Networks*, March 2003.

[40] S. S. Kanhere and H. Sethu, “Fair, efficient and low-latency packet scheduling using nested deficit round robin,” in *Proceedings of the IEEE Workshop on High Performance Switching and Routing*, Dallas, TX, May 2001, pp. 6–10.

[41] 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.

[42] S. Tsao and Y. Lin, "Pre-order deficit round robin: a new scheduling algorithm for packet-switched networks," *Computer Networks*, vol. 35, no. 2-3, pp. 287–305, February 2001.

[43] Y. Zhou and H. Sethu, "On the relationship between absolute and relative fairness bounds," *IEEE Communication Letters*, vol. 6, no. 1, pp. 37–39, January 2002.

[44] K. Thompson, G. J. Miller, and R. Wilder, "Wide-area internet traffic patterns and characteristics," *IEEE Network*, vol. 11, no. 6, pp. 10–23, November/December 1997.

[45] I. Widjaja and A. I. Elwalid, "Performance issues in vc-merge capable switches for multiprotocol label switching," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 6, pp. 1178–1189, June 1999.

[46] S. S. Kanhere, H. Sethu, and A. B. Parekh, "Fair and efficient packet scheduling using elastic round robin," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 324–326, March 2002.

[47] S. S. Kanhere and H. Sethu, "Low-latency guaranteed-rate scheduling using elastic round robin," *Computer Communication*, vol. 25, no. 14, pp. 1315–1322, September 2002.

[48] A. K. Parekh, *A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks*, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, February 1992.

[49] D. Stiliadis, *Traffic Scheduling in Packet-Switched Networks: Analysis, Design and Implementation*, Ph.D. thesis, University of California, Santa Cruz, 1996.

[50] F. A. Cowell, *Measuring Inequalities: Techniques for the Social Sciences*, John Wiley and Sons, New York, NY, 1977.

[51] A. W. Marshall and I. Olkin, *Inequalities: Theory of Majorization and its Applications*, Academic Press, New York, NY, 1979.

[52] A. Kumar and J. Kleinberg, "Fairness measures for resource allocation," in *Proceedings of the Symposium on Foundations of Computer Science*, November 2000, pp. 75–78.

[53] J. E. Stiglitz, *Economics*, W. W. Norton and Co, 1993.

[54] National Laboratory for Applied Network Research, ""Passive Measurement and Analysis"," `http://pma.nlanr.net/PMA/`.

[55] S. S. Kanhere and H. Sethu, "On the latency bound of deficit round robin," in *Proceedings of the International Conference on Computer Communications and Networks*, Miami, FL, October 2002.

[56] S. S. Kanhere and H. Sethu, "On the latency bound of pre-order deficit round robin," in *Proceedings of the IEEE Conference on Local Computer Networks*, Tampa, FL, November 2002.

[57] S. S. Kanhere and H. Sethu, "On the latency and fairness characteristics of pre-order drr," *submitted to Computer Communications*, January 2003.

[58] H. Adisheshu, G. Parulkar, and G. Varghese, "Reliable fifo load balancing over multiple fifo channels," Tech. Rep., Washington University, 1995.

[59] S. S. Kanhere, A. B. Parekh, and H. Sethu, "Fair and efficient packet scheduling in wormhole networks," in *Proceedings of the International Parallel and Distributed Processing Symposium*, Cancun, Mexico, May 2000, pp. 623–632.

Appendix A. Latency Analysis of DRR and Pre-order DRR

In this appendix we present the latency analysis of the Deficit Round Robin (DRR) and Pre-order DRR schedulers. DRR is one of the most popular frame-based fair scheduling disciplines that is now employed in a number of real environments involving fair scheduling, including Cisco routers and Microsoft's Windows NT. It is shown in [49] that the DRR scheduler belongs to the class of $\mathcal{LR}$ servers. Stiliadis and Varma report an upper bound on the latency of DRR [38], and its derivation is detailed in [49]. In this appendix, however, we obtain a significantly lower value of the upper bound on the latency of DRR and show that the DRR scheduler has better performance characteristics than previously believed. We also show that our upper bound on the DRR latency is tight.

In [42], Tsao and Lin have proposed a new scheduling discipline called Pre-order Deficit Round Robin (Pre-order DRR) which aims at overcoming the aforementioned drawbacks. In Pre-order DRR a limited number of priority queues, p, are added to the DRR scheduler. These queues reorder the transmission sequence of the packets in each DRR round and thus eliminate the strict round-robin nature of service order. It is shown in [42] that Pre-order DRR belongs to the general class of Latency-Rate ($\mathcal{LR}$) servers [38] and the authors derive an upper bound on its latency. In this appendix, we use a different, unique, and novel approach to analytically re-derive the latency bound of Pre-order DRR and we prove that our bound is a tight one. Our approach is based on interpreting the Pre-order DRR bandwidth allocations as an instance of the Nested Deficit Round Robin (Nested DRR) discipline discussed in [40]. Note that, a similar interpretation was used for while evaluating the latency bound of the PERR scheduler in Section 5.1. The latency bound of Pre-order DRR derived in this appendix is significantly lower than the bound derived by Tsao and Lin, demonstrating that Pre-order DRR has even better performance characteristics than previously argued by its own authors.

The rest of the appendix is organized as follows. Section A.1 presents a brief overview of the DRR scheduler. In Section A.2 we present our analysis of the latency bound of Pre-order DRR. Section A.3 presents a brief overview of the Pre-order DRR. Section A.4 discusses the interpretation of Pre-order DRR bandwidth allocations as an instance of allocations in Nested DRR. Finally, in Section A.5, we present our analysis of the latency bound of Pre-order DRR.

A.1 Overview of DRR

In this section, we present a brief overview of the DRR scheduler, a detailed description of which can be found in [21].

Let r be the transmission rate of the output link, the access to which is controlled by a DRR scheduler. Assume that there are a total of n flows multiplexed on this link. Let ρ_i be the reserved rate for flow i and let ρ_{min} be the minimum reserved rate among all the n flows. Since all these n flows share the same output link, $\sum_{i=1}^{n} \rho_i \leq r$. In order that the flows receive service proportional to their reserved rates, each flow i is assigned a weight, w_i, given by,

$$w_i = \frac{\rho_i}{\rho_{min}} \tag{A.1}$$

Note that, for any flow i, $w_i \geq 1$.

A flow is said to be *active* during a certain time interval, if it always has packets awaiting service during this interval. The DRR scheduler maintains a linked list of the active flows, called the *ActiveList*. At the start of an active period of a flow, the flow is added to the tail of the *ActiveList*. A *round* is defined as one round robin iteration during which the DRR scheduler serves all the flows that are present in the *ActiveList* at the outset of the round. Each active flow is assigned a *quantum* by the DRR scheduler. The quantum allocated to a flow is defined as the service that the flow should receive during each round robin service opportunity. Let Q_i represent the quantum assigned to flow i and let Q_{min} be the quantum assigned to the flow with the lowest reserved rate. The quantum assigned to flow i, Q_i is given by $w_i Q_{min}$. Thus, the quanta assigned to the flows are in proportion of their reserved rates. In order that the work complexity of the DRR scheduler is O(1), it is necessary that Q_{min} should be greater than or equal to the size of the largest packet that may potentially arrive during the execution of the scheduler. Note that, during some service opportunity, a flow may not be able to transmit a packet because doing so would cause the flow to exceed its allocated quantum. The scheduler maintains a per-flow state, the *deficit count*, which records the difference between the amount of data actually sent thus far, and the amount that should have been sent. This deficit is added to the value of the quantum in the next round, as the amount of data the scheduler should try to schedule in the next round. Thus, a flow that received very little service in a certain round is given an opportunity to receive more service in the next round.

A frame is defined as the sum of the quanta allocated to all the active flows in a DRR round. Let F denote the size in bits of a DRR frame. The upper bound of the latency of DRR is derived in [49] as $(3F - 2\phi_i)/r$, where r represents the transmission rate of the output link. This, however, is a loose bound borne of the incorrect assumption that the upper bound on the deficit count of a flow is equal to its quantum.

A.2 Latency Analysis of DRR

In this section we analyze the latency analysis of the DRR scheduler. Our approach is similar to the approach employed in Section 3.4 in deriving the latency bound of ERR.

Theorem A.2.1 *The DRR scheduler belongs to the class of $\mathcal{LR}$ servers, with an upper bound on the latency Θ_i for flow i given by,*

$$\Theta_i \leq \frac{1}{r}\left((W - w_i)Q_{min} + (m-1)\left(\frac{W}{w_i} + n - 2\right)\right) \tag{A.2}$$

where n is the total number of active flows, W is the sum of the weights of all the flows and r is the transmission rate of the output link.

Proof: Since the latency of an $\mathcal{LR}$ server can be estimated based on its behavior in the flow active period, we will prove the theorem by showing that,

$$\Theta_i' \leq \frac{1}{r}\left((W-w_i)Q_{min} + (m-1)\left(\frac{W}{w_i} + n - 2\right)\right)$$

Let flow i become active at time instant τ_i. For deriving an upper bound on the latency of DRR we consider a time interval (τ_i, t) during which flow i is continuously active. Then we obtain the lower bound on the total service received by flow i during the time interval under consideration. Lastly we express the lower bound in the form of Equation (3.10) to derive the latency bound. In Section 3.4, it has been proved that for deriving a tight upper bound on the latency of the ERR scheduling discipline, we must consider an active period (τ_i, t) such that τ_i coincides with the beginning of the service opportunity of a flow and t belongs to the set of time instants at which the scheduler begins serving flow i. It can be

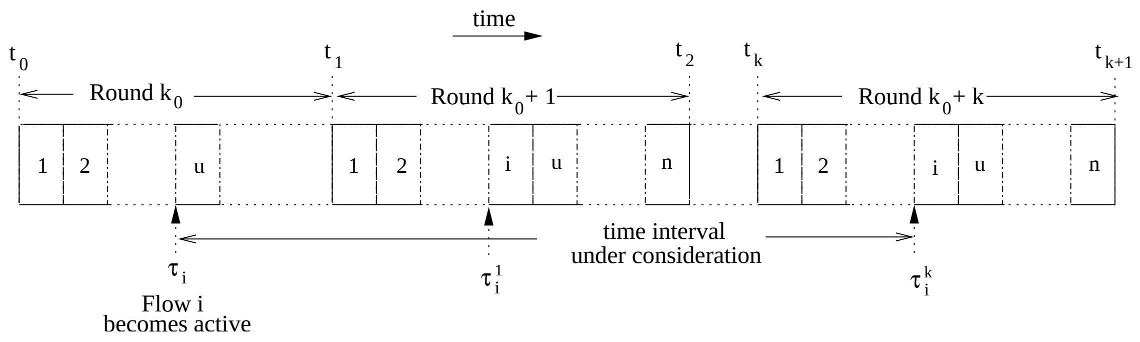


Figure A.1: An illustration of the time interval under consideration for the analysis of the latency bound of DRR

easily verified that these conditions are applicable in the analysis of the latency bound of all round robin schedulers including DRR. Let τ_i^k be the time instant marking the start of the k-th service opportunity of flow i. From the above, to determine a tight upper bound on the latency of the DRR scheduler we need to only consider time intervals (τ_i, τ_i^k) for all k. Figure A.1 illustrates the time interval under consideration for a given k. Note that, the time instant τ_i may or may not coincide with the end of a round and the start of the subsequent round. Let k_0 be the round which is in progress at time instant τ_i or which ends exactly at time instant τ_i. Let the time instant t_h mark the end of round $(k_0 + h - 1)$ and the start of the subsequent round.

Let $Sent_i(s)$ represent the total data transmitted from flow i in the s-th round of service of the DRR scheduler. Also let $DC_i(s)$ represent the deficit count of flow i following its service in round s. It has been proved in [21] that for any flow i in any round s,

$$0 \leq DC_i(s) \leq m - 1 \tag{A.3}$$

$$Sent_i(s) = w_i Q_{min} + DC_i(s-1) - DC_i(s) \tag{A.4}$$

Note that, while analyzing the latency bound of the DRR scheduler in [49], it has been assumed that the upper bound on $DC_i(s)$ is Q_i. It is easily verified that this assumption will be true only if $Q_i = M = m$. In all other situations, the upper bound on the deficit count as specified by Equation (A.3) is a much tighter bound.

As illustrated in Figure A.1, assume that the time instant when flow i becomes active coincides with the time instant when some flow u is about to start its service opportunity during the k_0-th round. Let G_a denote the set of flows which receive service during the time interval (τ_i, t_1), i.e., *after* flow i becomes active. Similarly, let G_b denote the set of flows which are served by the DRR scheduler during the time interval (t_0, τ_i), i.e., *before* flow i becomes active. Note that, flow i is not included in either of these two sets since flow i will receive its first service opportunity only in the $(k_0 + 1)$-th round. If the time instant τ_i coincides with the time instant t_1, which marks the end of the k_0-th round and the start of the $(k_0 + 1)$-th round, then the set G_a will be empty and all the $n - 1$ flows will be included in the set G_b. Note that, in this case, flow i will be the last to receive service in the $(k_0 + 1)$-th round and all subsequent rounds during the time interval under consideration.

The first step towards analyzing the latency bound involves obtaining an upper bound on the size of the time interval (τ_i, τ_i^k). This time interval can be split into the following three sub-intervals:

1. (τ_i, t_1): This sub-interval includes the part of the k_0-th round during which all the flows belonging to the set G_a will be served by the DRR scheduler. Summing Equation (A.4) over all these flows,

$$t_1 - \tau_i = \frac{1}{r} \sum_{j \in G_a} \{w_j Q_{min} + DC_j(k_0 - 1) - DC_j(k_0)\} \tag{A.5}$$

2. (t_1, t_k): This sub-interval includes $k - 1$ rounds of execution of the DRR scheduler starting at round $(k_0 + 1)$. Consider the time interval (t_h, t_{h+1}) when round $(k_0 + h)$ is in progress. Summing Equation (A.4) over all n flows and since W is the sum of the weights of all the n flows, we have,

$$t_{h+1} - t_h = \frac{W}{r} Q_{min} + \frac{1}{r} \sum_{j=1}^{n} \{DC_j(k_0 + h - 1) - DC_j(k_0 + h)\}$$

   Summing the above over $(k - 1)$ rounds beginning with round $k_0 + 1$,

$$t_k - t_1 = \frac{W}{r}(k-1) Q_{min} + \frac{1}{r} \sum_{j=1}^{n} \{DC_j(k_0) - DC_j(k_0 + k - 1)\} \tag{A.6}$$

3. (t_k, τ_i^k): This sub-interval includes the part of the $(k_0 + k)$-th round during which all the flows belonging to the set G_b will be served by the DRR scheduler. Summing Equation (A.4) over all these flows,

$$\tau_i^k - t_k = \frac{1}{r} \sum_{j \in G_b} \{w_j Q_{min} + DC_j(k_0 + k - 1) - DC_j(k_0 + k)\} \tag{A.7}$$

Combining Equations (A.5), (A.6) and (A.7) and since W is the sum of the weights of all the n flows, we have,

$$\begin{aligned}\tau_i^k - \tau_i &= \frac{W}{r}(k-1)Q_{min} + \left(\frac{W - w_i}{r}\right)Q_{min} \\ &+ \frac{1}{r}\sum_{j \in G_a}(DC_j(k_0 - 1) - DC_j(k_0 + k - 1)) \\ &+ \frac{1}{r}\sum_{j \in G_b}(DC_j(k_0) - DC_j(k_0 + k)) \\ &+ \frac{1}{r}(DC_i(k_0) - DC_i(k_0 + k - 1)) \qquad \text{(A.8)}\end{aligned}$$

Now since flow i becomes active during round k_0, its deficit count at the end of the k_0-th round, $DC_i(k_0)$ is equal to zero. Using this fact and the bounds on the deficit count from Equation (A.3) in Equation (A.8), we have,

$$\begin{aligned}\tau_i^k - \tau_i &\leq \frac{W}{r}(k-1)Q_{min} + \left(\frac{W - w_i}{r}\right)Q_{min} \\ &+ \frac{(n-1)(m-1)}{r} - \frac{1}{r}DC_i(k_0 + k - 1)\end{aligned}$$

Solving for $(k-1)$,

$$\begin{aligned}(k-1) &\geq \frac{r}{WQ_{min}}(\tau_i^k - \tau_i) - \frac{W - w_i}{W} - \frac{1}{WQ_{min}}(n-1)(m-1) \\ &+ \frac{1}{WQ_{min}}DC_i(k_0 + k - 1) \qquad \text{(A.9)}\end{aligned}$$

Note that, during the time interval under consideration, (τ_i, τ_i^k), flow i receives service in $(k-1)$ rounds starting at round (k_0+1). Hence, using Equation (A.4) over these $(k-1)$ rounds of service for flow i, and since the deficit count of a newly active flow is 0, we get,

$$Sent_i(\tau_i, \tau_i^k) = w_i(k-1)Q_{min} - DC_i(k_0 + k - 1) \qquad \text{(A.10)}$$

Using Equation (A.9) to substitute for $(k-1)$ in Equation (A.10), we get,

$$\begin{aligned}Sent_i(\tau_i, \tau_i^k) &\geq \frac{w_i r}{W}(\tau_i^k - \tau_i) - \frac{w_i}{W}(W - w_i)Q_{min} - \frac{wi}{W}(n-1)(m-1) \\ &+ \frac{w_i}{W}DC_i(k_0 + k - 1) - DC_i(k_0 + k - 1) \qquad \text{(A.11)}\end{aligned}$$

Simplifying the above, we have,

$$\begin{aligned} Sent_i(\tau_i, \tau_i^k) \quad &\geq \quad \frac{w_i r}{W}\left(\tau_i^k - \tau_i - \frac{1}{r}(W - w_i)Q_{min} - \frac{1}{r}(n-1)(m-1)\right. \\ &\left. - \quad \frac{DC_i(k_0 + k - 1)}{r}\left(\frac{W - w_i}{w_i}\right)\right) \end{aligned} \tag{A.12}$$

Using Equation (A.8) it can be easily verified that,

$$\begin{aligned} \tau_i^k - \tau_i \quad &> \quad \frac{1}{r}(W - w_i)Q_{min} + \frac{1}{r}(n-1)(m-1) \\ &+ \frac{DC_i(k_0 + k - 1)}{r}\left(\frac{W - w_i}{w_i}\right) \end{aligned} \tag{A.13}$$

Now, since the reserved rates are proportional to the weights assigned to the flows as given by Equation (A.1), and since the sum of the reserved rates is no more than the link rate r, we have,

$$\rho_i \leq \frac{w_i}{W} r \tag{A.14}$$

Substituting for $\frac{w_i r}{W}$ from Equation (A.14) in Equation (A.12) and using Equation (A.13) we have,

$$\begin{aligned} Sent_i(\tau_i, \tau_i^k) \quad &\geq \quad \rho_i(\tau_i^k - \tau_i) - \frac{\rho_i}{r}(W - w_i)(Q_{min}) \\ &- \frac{\rho_i}{r}(n-1)(m-1) \\ &- \frac{\rho_i}{r}\left(\frac{W - w_i}{w_i}\right) DC_i(k_0 + k - 1) \end{aligned} \tag{A.15}$$

Comparing the above equation with Equation (3.10), the latency bound of the DRR scheduler is given by,

$$\begin{aligned} \Theta_i \quad &\leq \quad \left(\frac{W - w_i}{r}\right) Q_{min} + \frac{1}{r}(n-1)(m-1) \\ &+ \frac{1}{r}\left(\frac{W - w_i}{w_i}\right) DC_i(k_0 + k - 1) \end{aligned} \tag{A.16}$$

From the above equation it is readily seen that the latency reaches the upper bound if the deficit count, $DC_i(k_0+k-1)$ is equal to its upper bound $(m-1)$ as given by Equation (A.3).

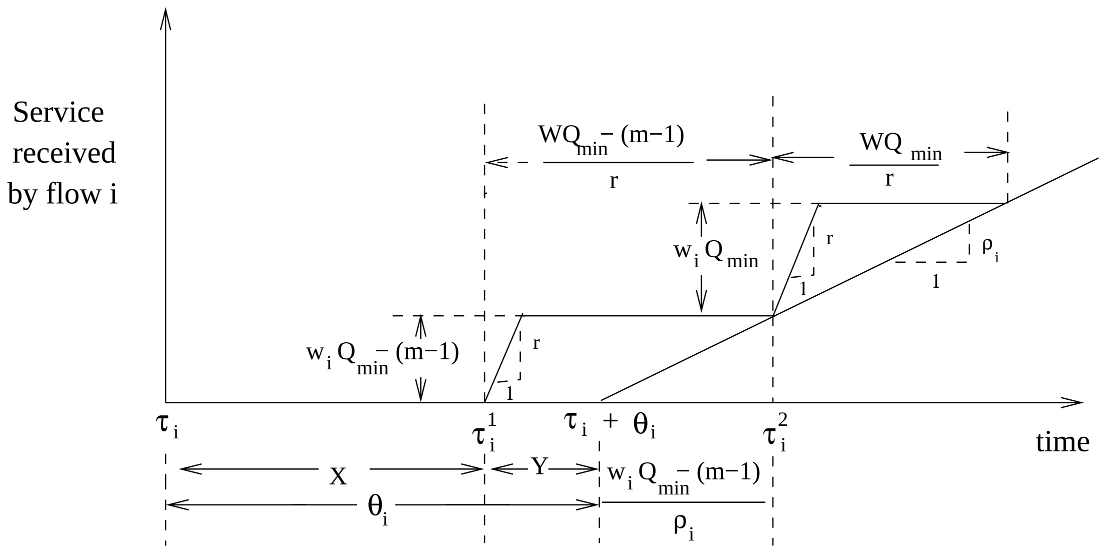


Figure A.2: Plot of the service received by flow i with DRR

Substituting this in Equation (A.16), we get,

$$\Theta_i \leq \frac{1}{r}\left\{(W - w_i)Q_{min} + (m-1)\left(\frac{W}{w_i} + n - 2\right)\right\} \tag{A.17}$$

As discussed earlier, flow i will experience its worst latency during an interval (τ_i, τ_i^k) for some k. Therefore, from Equation (A.17), the statement of the theorem is proved. ■

We now proceed to show that the latency bound given by the above is tight by illustrating a case when the bound is actually achieved. Assume that flow i becomes busy at a certain time instant τ_i, which also coincides with the start of a certain round (k_0+1). Since the other flows in the *ActiveList* will be served first, flow i becomes backlogged instantly and τ_i is also the start of its active period. Assume that for any time instant t, $t \geq \tau_i$, a total of n flows, including flow i, are active. Let $\mathcal{F}$ represent the set of all n flows. Also, assume that the summation of the reserved rates of all the n flows equals the output link transmission rate, r. Hence, $\rho_i = \frac{w_i}{W}r$. Since flow i became active at time τ_i, its deficit count at the start of round (k_0+1) is 0. Let the deficit count of all the other flows at the start of round $(k_0 + 1)$ be equal to (m-1). From Equations (A.3) and (A.4), a flow j can transmit a maximum of $w_j Q_{min} + (m-1)$ bits during a round robin service opportunity. In the worst case, before flow i is served by the DRR scheduler, each of the other $(n-1)$ flows will receive this maximum service. Hence, the cumulative delay until flow i receives service, X, is given by,

$$\begin{aligned} X &= \frac{(\sum_{\substack{j\in\mathcal{F}\\ j\neq i}} w_j)(Q_{min}) + (n-1)(m-1)}{r} \\ &= \frac{(W - w_i)(Q_{min}) + (n-1)(m-1)}{r} \end{aligned} \tag{A.18}$$

Even though X represents the time for which flow i has to wait until its first packet is scheduled, Equation (3.10) does not hold true when X is substituted as Θ_i. This is because in the time interval $(\tau_i, \tau_i + X)$ flow i has not yet started receiving service at its guaranteed rate. We assume that the latency, Θ_i is given by,

$$\Theta_i = X + Y \tag{A.19}$$

A plot of the service received by flow i against time is illustrated in Figure A.2. Assume that the total service received by flow i during its first service opportunity is $w_i Q_{min} - (m-1)$. Note that, from Equations (A.3) and (A.4), this equals the minimum service that flow i can receive during any service opportunity. At the end of the $(k_0 + 1)$-th round, the deficit count for flow i is $(m-1)$ whereas the deficit count for all the other flows is zero. In the worst case, during the $(k_0 + 2)$-th round, each flow j from amongst the other $(n-1)$ flows will transmit a maximum of $w_j Q_{min}$ bits before flow i receives its second service opportunity. During this service opportunity, flow i will be able to transmit at-least a minimum of $w_i Q_{min}$ bits, and will thus start receiving service at its guaranteed rate. Referring to Figure A.2, we have,

$$Y + \frac{w_i Q_{min}}{\rho_i} - \frac{m-1}{\rho_i} = \frac{W Q_{min}}{r} - \frac{m-1}{r}$$

Now since $\rho_i = \frac{w_i}{W} r$, simplifying further, we have,

$$Y = \frac{(m-1)}{r} \left(\frac{W - w_i}{w_i} \right) \tag{A.20}$$

Substituting for X and Y from Equations (A.18) and (A.20) in Equation (A.19), it can be easily verified that the latency bound is exactly met.

A.3 Overview of Pre-Order DRR

However, note that DRR is a frame-based scheduler and hence suffers from all of its associated problems as discussed in Chapter 4. In [42], Tsao and Lin have proposed a new scheduling discipline called Pre-order DRR which aims to eliminate the above weaknesses of the DRR scheduler while trying to preserve its good properties such as its low work complexity. The assignment of the weights and the quanta in Pre-order DRR are identical to those in DRR. In fact, the Pre-order DRR scheduler also works in rounds. However, unlike the DRR scheduler which serves the active flows in a round robin fashion, the Pre-order DRR scheduler reorders the transmission sequence of the packets within each DRR round. In this section, we present a brief overview of the Pre-order DRR scheduling discipline. A significantly more detailed treatment may be found in [42].

Let us assume that a total of y packets are transmitted from flow i in the s-th round of service. The packets are labeled as 1, 2, $\ldots$ y indicating their position in the stream of packets that are scheduled from flow i in round s. Note that, y represents the last packet that is served in round s from flow i. As in DRR, the deficit count serves as a measure of past unfairness. Let $DC_i^m(s)$ represent the deficit count of flow i following the transmission of the m-th packet of the s-th round.

Definition A.3.1 *Define the* Quantum Availability, *denoted by $QA_i^m(s)$, of flow i after the transmission of the m-th packet from flow i in round s as follows:*

$$QA_i^m(s) = \frac{DC_i^m(s)}{Q_i} \tag{A.21}$$

The *Quantum Availability* of a flow keeps track of the unused quantum of the flow in the current round.

In Pre-order DRR a priority queue module consisting of p queues and a *classifier* module are appended to the original DRR architecture. Let PQ_1, PQ_2, ... , PQ_p represent the priority queues in the descending order of priority with PQ_1 as the highest priority queue and PQ_p denoting the lowest priority queue. Just as in DRR, the Pre-order DRR maintains a linked list of active flows called the *ActiveList*. However, the flows in the *ActiveList* are not served in a round robin manner as in DRR. This is a list of the active flows that have already received their fair share of service in the current round. These flows are, however, eligible for receiving service in the subsequent round. At the start of a round, the *Classifier* module classifies the packets that will be served in the current round from each flow present in the *ActiveList* according to its *Quantum Availability* into the p priority queues. In general, the priority queue, $z_i^m(s)$ into which the m-th packet served from flow i in the s-th round is added is calculated as follows,

$$z_i^m(s) = p - \lfloor QA_i^m(s) \times p \rfloor \tag{A.22}$$

Once all the packets that can be scheduled in the current round from flow i have been transferred from the flow buffers into the priority queues, if flow i is still active, it is added to the tail of the *ActiveList*.

When the scheduler is ready to transmit, it begins serving the packet at the head of the highest non-empty priority queue. Note that, if a packet is added to a priority queue that has a higher priority than the queue from which the scheduler is currently serving a packet, then following the current transmission, the scheduler will first serve the packet added into the higher priority queue. The round in progress ends when all the priority queues are empty. It has been proved in [42] that Pre-order DRR has a low worst-case work complexity of $O(1)$ with respect to the number of flows and $O(\log p)$ with respect to the number of priority queues.

A.4 The Nested DRR Interpretation

In this section we present the interpretation of PERR bandwidth allocations as an instance of allocations in the Nested-DRR scheduler. Note that, a similar approach was adopted in Section 5.1 while analyzing the latency bound on the PERR scheduling discipline.

The primary goal of the Pre-order DRR scheduler is to break the quantum allocated to a flow in a DRR round into several pieces so that it can be utilized in pieces over the course of the round. The Nested DRR scheduler proposed in [40] tries to eliminate the drawbacks of the DRR scheduler by creating a set of multiple rounds inside each DRR round and executes a modified version of the DRR algorithm within each of these inner rounds. The Nested DRR scheduler tries to serve Q_{min} worth of data from each flow during each inner round. During an outer round, a flow is considered to be eligible for service in as many inner rounds as are required by the scheduler to exhaust its quantum. This results in a significantly lower latency bound, while preserving the $O(1)$ work complexity and the fairness characteristics of DRR. We can hypothetically interpret the operation of the Pre-order DRR scheduler as a *nested* version of DRR similar to Nested DRR. This interpretation is useful in analyzing the latency bound of the Pre-order DRR scheduler. Each round in DRR can be referred to as an *outer round*. The time period during which the Pre-order DRR scheduler serves the flows present in the priority queue PQ_u during the s-th outer round is referred to as *inner round* (s, u). Thus, each outer round can be split into as many inner rounds as the number of priority queues, p. Since the Pre-order DRR scheduler visits the priority queues in a descending order of priority starting at priority queue PQ_1 and ending with queue PQ_p, the first and last inner rounds in outer round s are $(s, 1)$ and (s, p) respectively.

The quantum assigned to each flow is divided equally among the p priority queues. Thus, the quantum allocated to flow i in each of its inner rounds is equal to $\frac{Q_i}{p}$. Let

$Served_i(s,u)$ represent the total data scheduled from flow i in inner round (s,u). Also let $DC_i(s,u)$ denote the deficit round of flow i at the end of the (s,u)-th inner round. Note that, the deficit count of a flow at the end of the last inner round of an outer round is the same as its deficit count at the end of the corresponding round in DRR. Also, this deficit count is carried over to the first inner round of the subsequent outer round. Hence,

$$DC_i(s,p) = DC_i(s) = DC_i(s+1,0)$$

Note that, $DC_i(s+1,0)$ is used to represent the deficit count of flow i at the start of the inner round $(s+1,1)$. As in DRR, the deficit count is calculated as follows,

$$DC_i(s,u) = \frac{Q_i}{p} + DC_i(s,u-1) - Served_i(s,u) \tag{A.23}$$

It can be easily proved that Equation (A.3) which represents the bounds on the deficit count, $DC_i(s)$, also holds true for $DC_i(s,u)$. Hence, for any flow i and inner round (s,u),

$$0 \le DC_i(s,u) \le m-1 \tag{A.24}$$

In DRR, since the quantum of each flow is greater than or equal to the size of the largest packet that may potentially arrive during its execution, the scheduler is guaranteed to serve at least one packet from each of the active flows in each round. However, in Pre-order DRR, it may be possible that the sum of $\frac{Q_i}{p}$ and $DC_i(s,u-1)$ is less than the size of the packet at the head of flow i. In this case, flow i will not receive any service in inner round (s,u). Thus, a flow need not necessarily receive service in each inner round. If the Pre-order DRR scheduler was serving flows in an exact round robin manner as in Nested DRR then, in the worst-case, it may be possible that none of the active flows will be able to transmit a packet in an inner round resulting in a work complexity of $O(n)$ or greater, where n represents the total number of active flows. The *Classifier* module in the Pre-order DRR scheduler avoids this large work complexity by classifying the packets into the p priority queues at the start of each outer round. This classification determines which inner rounds each flow will be served in and the scheduler does not need to query all the flows in a round robin order.

Note that, the deficit count of a flow is updated at the end of each inner round using Equation (A.23) irrespective of whether it receives service in that inner round. From Equation (A.23), the service received by flow i in inner round (s, u) is,

$$Served_i(s,u) = \frac{Q_i}{p} + DC_i(s,u-1) - DC_i(s,u) \tag{A.25}$$

Definition A.4.1 *Let $Sent_i(s,u)$ represent the total service received by flow i since the start of the s-th outer round until the time instant when the scheduler finishes serving the packets in the priority queue PQ_u.*

$Sent_i(s,u)$ is computed as follows:

$$Sent_i(s,u) = \sum_{w=1}^{w=u} Served_i(s,w)$$

Substituting for $Served_i(s,w)$ from Equation (A.25) in the above, we have,

$$Sent_i(s,u) = (\frac{u}{p})Q_i + DC_i(s-1) - DC_i(s,u) \tag{A.26}$$

$Sent_i(s,u)$ will be positive only if the the sum of $(\frac{u}{p})Q_i$ and $DC_i(s-1)$ is greater than or equal to the size of the packet at the head of flow i. If this condition is not satisfied then it implies that flow i has not received any service in the first u inner rounds. However, each flow is guaranteed to receive service during at least one inner round within each outer round.

Definition A.4.2 *Define $Sent_i(s)$ as the total service received by flow i in outer round s.*

Note that, $Sent_i(s)$ is equal to $Sent_i(s,p)$. Therefore, substituting $u = p$ in Equation (A.26), we get,

$$Sent_i(s) = Q_i + DC_i(s-1) - DC_i(s) \tag{A.27}$$

A.5 Latency Analysis of Pre-Order DRR

In this section we analyze the latency analysis of the DRR scheduler and also prove that it belongs to the general class of $\mathcal{LR}$-servers. Our approach is similar to the approach employed in Section 3.4 in deriving the latency bound of PERR.

Theorem A.5.1 *The Pre-order DRR scheduler belongs to the class of $\mathcal{LR}$ servers, with an upper bound on the latency Θ_i for flow i given by,*

$$\Theta_i \leq \frac{1}{r}\left\{\frac{(W-w_i)Q_{min}}{p} + (m-1)\left(\frac{W}{w_i} + n - 2\right)\right\} \tag{A.28}$$

where n is the total number of active flows, p represents the number of priority queues, W is the sum of the weights of all the flows and r denotes the transmission rate of the output link.

Proof: Since the latency of an $\mathcal{LR}$ server can be estimated based on its behavior in the flow active periods, we will prove the theorem by showing that,

$$\Theta_i' \leq \frac{1}{r}\left\{\frac{(W-w_i)Q_{min}}{p} + (m-1)\left(\frac{W}{w_i} + n - 2\right)\right\}$$

Let τ_i be the time instant when flow i becomes active. To prove the statement of the theorem we must consider an active period (τ_i, t) of flow i. We then obtain the lower bound on the total service received by flow i during the time interval under consideration. Lastly, we express the lower bound in the form of Equation (3.10) to derive the latency bound.

In [47] and Section 3.4 it has been proved that to obtain a tight upper bound on the latency of the Elastic Round Robin scheduler [46, 59], we need to consider only those active periods (τ_i, t) which satisfy the following two requirements:

1. τ_i coincides with the start of a service opportunity of some flow.

2. Time instant t belongs to a subset of all possible time instants at which the scheduler begins serving flow i.

It can be easily verified that these two conditions are applicable for proving the upper bound on the latency of the Pre-order DRR scheduler. Let $\tau_i^{(e,f)}$ be the time instant marking the start of the service of flow i when flow i is at the head of priority queue PQ_f in round e. In other words, this time instant represents the start of the service opportunity of flow i in inner round (e, f). Note that, $\tau_i^{(e,f)}$ belongs to the set of time instants when the scheduler begins serving flow i. Therefore, in order to determine the latency bound of the Pre-order DRR we need to only consider time intervals $(\tau_i, \tau_i^{(e,f)})$ for all (e, f) in which flow i receives service.

The first step towards analyzing the latency bound involves choosing a suitable time interval $(\tau_i, \tau_i^{(e,f)})$ such that the size of this time interval is the maximum possible. Note that, the time instant τ_i may or may not coincide with the start of a new outer round. Let k_0 be the outer round which is in progress at time instant τ_i or which starts exactly at time instant τ_i. In either case, flow i will receive an opportunity to transmit Q_i worth of data in the k_0-th round. Let the time instant t_h mark the start of the outer round $(k_0 + h)$. Consider the case when τ_i does not coincide with the time instant t_0, the start of outer round k_0, i.e., $\tau_i > t_0$. In this case, the time interval (t_0, τ_i) will be excluded from the time interval under consideration. On the other hand, when τ_i coincides with t_0, the size of the time interval $(\tau_i, \tau_i^{(e,f)})$ is maximal. We, therefore, assume that the τ_i coincides with the start of the k_0-th outer round. Figure A.3 illustrates the time interval under consideration assuming that (e, f) is equal to $(k_0 + k, v)$. Note that, in Figure A.3, $OR(a)$ represents the a-th outer round and $IR(a, b)$ denotes the inner round (a, b) in the execution of the Pre-order DRR scheduler.

The time interval under consideration, $(\tau_i, \tau_i^{(k_0+k,v)})$, can be split into two sub-intervals:

1. (τ_i, t_k): This sub-interval includes k outer rounds of execution of the Pre-order DRR scheduler starting at outer round k_0. Consider the time interval (t_h, t_{h+1}) when outer

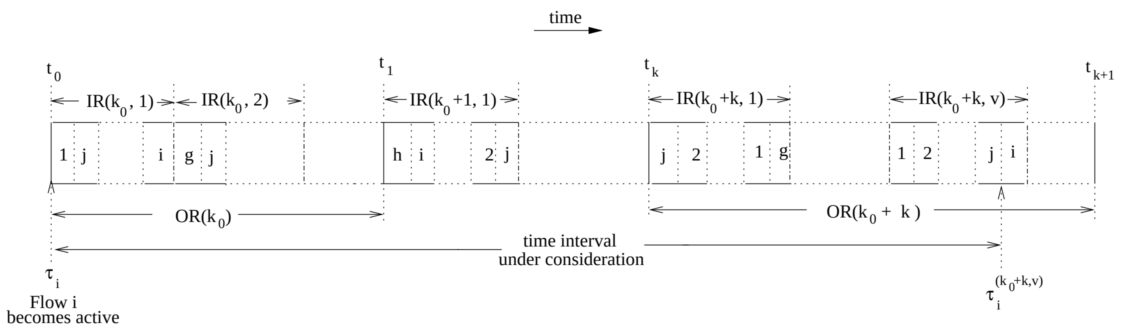


Figure A.3: An illustration of the time interval under consideration for the analysis of the latency bound of Pre-order DRR

round $(k_0 + h)$ is in progress. Summing Equation (A.27) over all n flows,

$$t_{h+1} - t_h = \frac{W}{r} Q_{min} + \frac{1}{r} \sum_{j=1}^{n} \{DC_j(k_0 + h - 1) - DC_j(k_0 + h)\} \tag{A.29}$$

Summing the above over k rounds beginning with round k_0,

$$t_k - \tau_i = \frac{W}{r}(kQ_{min}) + \frac{1}{r} \sum_{j=1}^{n} \{DC_j(k_0 - 1) - DC_j(k_0 + k - 1)\} \tag{A.30}$$

2. $(t_k, \tau_i^{(k_0+k,v)})$: This sub-interval includes the part of the $(k_0 + k)$-th round prior to the start of the service of flow i when it is at the head of priority queue PQ_v. In the worst-case, flow i will be the last flow to receive service among all the flows which may be present in priority queue PQ_v. In this case, during the sub-interval under consideration, the service received by flow i equals $Sent_i(k_0 + k, v - 1)$ whereas the service received by each flow j among the other $(n - 1)$ flows equals $Sent_j(k_0 + k, v)$. Note that, if v equals 1 then flow i does not receive service in this sub-interval. Summing $Sent_i(k_0 + k, v - 1)$ and $Sent_j(k_0 + k, v)$ for each flow j such that $1 \leq j \leq$

$n, j \neq i$ and using Equation (A.26), we have,

$$\begin{aligned}\tau_i^{(k_0+k,v)} - t_k &= \frac{1}{r}\sum_{\substack{j=1\\j\neq i}}^{n}(\frac{v}{p})w_j Q_{min} + \frac{1}{r}(\frac{v-1}{p})w_i Q_{min} \\ &+ \frac{1}{r}\sum_{\substack{j=1\\j\neq i}}^{n}(DC_j(k_0+k-1) - DC_j(k_0+k,v)) \\ &+ \frac{1}{r}(DC_i(k_0+k-1) - DC_i(k_0+k,v-1)) \end{aligned} \tag{A.31}$$

Combining Equations (A.30) and (A.31), we have,

$$\begin{aligned}\tau_i^{(k_0+k,v)} - \tau_i &= \frac{W}{r}(kQ_{min}) + \frac{1}{r}\sum_{\substack{j=1\\j\neq i}}^{n}(\frac{v}{p})w_j Q_{min} + \frac{1}{r}(\frac{v-1}{p})w_i Q_{min} \\ &+ \frac{1}{r}\sum_{\substack{j=1\\j\neq i}}^{n}(DC_j(k_0-1) - DC_j(k_0+k,v)) \\ &+ \frac{1}{r}(DC_i(k_0-1) - DC_i(k_0+k,v-1)) \end{aligned} \tag{A.32}$$

Now since flow i becomes active at the start of outer round k_0, its deficit count at the start of the k_0-th outer round, $DC_i(k_0-1)$ is equal to zero. Using this fact and the bounds on the deficit count from Equations (A.3) and (A.24) in Equation (A.32), we have,

$$\begin{aligned}\tau_i^{(k_0+k,v)} - \tau_i &\leq \frac{W}{r}(kQ_{min}) + \frac{1}{r}\sum_{\substack{j=1\\j\neq i}}^{n}(\frac{v}{p})w_j Q_{min} + \frac{1}{r}(\frac{v-1}{p})w_i Q_{min} \\ &+ \frac{(n-1)(m-1)}{r} - \frac{1}{r}DC_i(k_0+k,v-1) \end{aligned}$$

Solving for k,

$$\begin{aligned}k &\geq (\tau_i^{(k_0+k,v)} - \tau_i)\frac{r}{WQ_{min}} - \frac{r}{W}\sum_{\substack{j=1\\j\neq i}}^{n}(\frac{v}{p})w_j - \frac{r}{W}(\frac{v-1}{p})w_i \\ &- \frac{1}{WQ_{min}}(n-1)(m-1) + \frac{1}{WQ_{min}}DC_i(k_0+k,v-1) \end{aligned} \tag{A.33}$$

Note that, the total data transmitted by flow i during the time interval under consideration can be expressed as the following summation.

$$Sent_i(\tau_i, \tau_i^{(k,v)}) = Sent_i(\tau_i, t_k) + Sent_i(t_k, \tau_i^{(k,v)}) \tag{A.34}$$

As explained earlier, $Sent_i(t_k, \tau_i^{(k,v)})$ is the same as $Sent_i(k, v-1)$. $Sent_i(\tau_i, t_k)$ can be obtained by summing Equation (A.27) over k outer rounds starting at outer round k_0. Substituting the result of this summation and Equation (A.26) in Equation (A.34) and using the fact that the deficit count of a newly active flow is equal to zero, we have,

$$Sent_i(\tau_i, \tau_i^{(k_0+k,v)}) = w_i(kQ_{min}) + (\frac{v-1}{p})w_iQ_{min} - DC_i(k_0+k, v-1) \tag{A.35}$$

Using Equation (A.33) to substitute for k in Equation (A.35), we get,

$$\begin{aligned} Sent_i(\tau_i, \tau_i^{(k_0+k,v)}) \geq\ & \frac{w_i r}{W}(\tau_i^{(k_0+k,v)} - \tau_i) - \frac{w_i}{W}(\frac{v}{p})(W - w_i)Q_{min} \\ & - \frac{w_i}{W}(\frac{v-1}{p})w_iQ_{min} - \frac{w_i}{W}(n-1)(m-1) \\ & + \frac{w_i}{W}DC_i(k_0+k, v-1) + (\frac{v-1}{p})w_iQ_{min} \\ & - DC_i(k_0+k, v-1) \end{aligned}$$

Simplifying the above we get,

$$\begin{aligned} Sent_i(\tau_i, \tau_i^{(k_0+k,v)}) \geq\ & \frac{w_i r}{W}\Bigg(\tau_i^{(k_0+k,v)} - \tau_i - \frac{1}{r}(\frac{v}{p})(W - w_i)Q_{min} \\ & - \frac{1}{r}(\frac{v-1}{p})(W - w_i)Q_{min} - \frac{1}{r}(n-1)(m-1) \\ & - \frac{1}{r}DC_i(k_0+k, v-1)\left(\frac{W}{w_i} - 1\right)\Bigg) \end{aligned} \tag{A.36}$$

Using Equation (A.32) it can be easily verified that,

$$\begin{aligned} \tau_i^{(k_0+k,v)} - \tau_i >\ & \frac{1}{r}(\frac{v}{p})(W - w_i)Q_{min} \\ & + \frac{1}{r}(\frac{v-1}{p})(W - w_i)Q_{min} + \frac{1}{r}(n-1)(m-1) \\ & - \frac{1}{r}DC_i(k_0+k, v-1)\left(\frac{W}{w_i} - 1\right) \end{aligned} \tag{A.37}$$

Now, since the reserved rates are proportional to the weights assigned to the flows as given by Equation (A.1), and since the sum of the reserved rates is no more than the link rate r, we have,

$$\rho_i \leq \frac{w_i}{W}r \tag{A.38}$$

Substituting for $\frac{w_i r}{W}$ from Equation (A.38) in Equation (A.36) and using Equation (A.37), we have,

$$\begin{aligned} Sent_i(\tau_i, \tau_i^{(k_0+k,v)}) &\geq \rho_i(\tau_i^{(k_0+k,v)} - \tau_i) - \frac{\rho_i}{r}\left(\frac{W - w_i}{p}\right)Q_{min} - \frac{\rho_i}{r}(n-1)(m-1) \\ &\quad - \frac{\rho_i}{r}DC_i(k_0+k, v-1)\left(\frac{W}{w_i} - 1\right) \end{aligned} \tag{A.39}$$

Comparing the above equation with Equation (3.10), the latency bound of the Pre-order DRR scheduler is given by,

$$\begin{aligned} \Theta_i &\leq \frac{1}{r}\left(\frac{W - w_i}{p}\right)Q_{min} + \frac{1}{r}(n-1)(m-1) \\ &\quad + \frac{DC_i(k_0+k, v-1)}{r}\left(\frac{W}{w_i} - 1\right) \end{aligned} \tag{A.40}$$

From the above equation it is readily seen that the latency reaches the upper bound if the deficit count $DC_i(k_0 + k, v - 1)$ is equal to its upper bound $(m - 1)$ as given by Equation (A.24). Substituting this in Equation (A.40), we get,

$$\Theta_i \leq \frac{1}{r}\left\{\frac{W - w_i}{p}Q_{min} + (m-1)\left(\frac{W}{w_i} + n - 2\right)\right\} \tag{A.41}$$

As discussed earlier in Section 3.4, flow i will experience its worst latency during an interval $(\tau_i, \tau_i^{(k_0+k,v)})$ for some inner round $(k_0 + k, v)$. Therefore, from Equation (A.41), the statement of the theorem is proved. ■

We now proceed to show that the above latency bound is tight by illustrating a case when the bound is actually achieved. Let **F** represent the set of all n flows. Assume that flow i becomes active at a certain time instant τ_i which also coincides with the start of certain outer round k_0. Since the arrival of a packet into the empty buffer of a flow signals the start of a busy period of the flow, τ_i is also the start of its busy period. Assume that for any time instant t, $t > \tau_i$, a total of n flows, including flow i, are active. Also, assume that the summation of the reserved rates of all the n flows is equal to the transmission rate of the output link, r. Therefore, we have, $\rho_i = \frac{w_i}{W}r$. Since flow i became active at time τ_i, its deficit count at the start of outer round k_0 is 0. Let the deficit count of all the other $(n - 1)$

flows be equal to the maximum value of $(m-1)$. Using Equations (A.24) and (A.25), it is seen that the maximum service received by a flow j during an inner round, S_j^{max} is given by,

$$S_j^{max} = \frac{w_i Q_{min}}{p} + (m-1) \tag{A.42}$$

On a similar note, the minimum service received by flow j during an inner round, S_j^{min}, provided it is present in the priority queue being served, is given by,

$$S_j^{min} = \frac{w_i Q_{min}}{p} - (m-1) \tag{A.43}$$

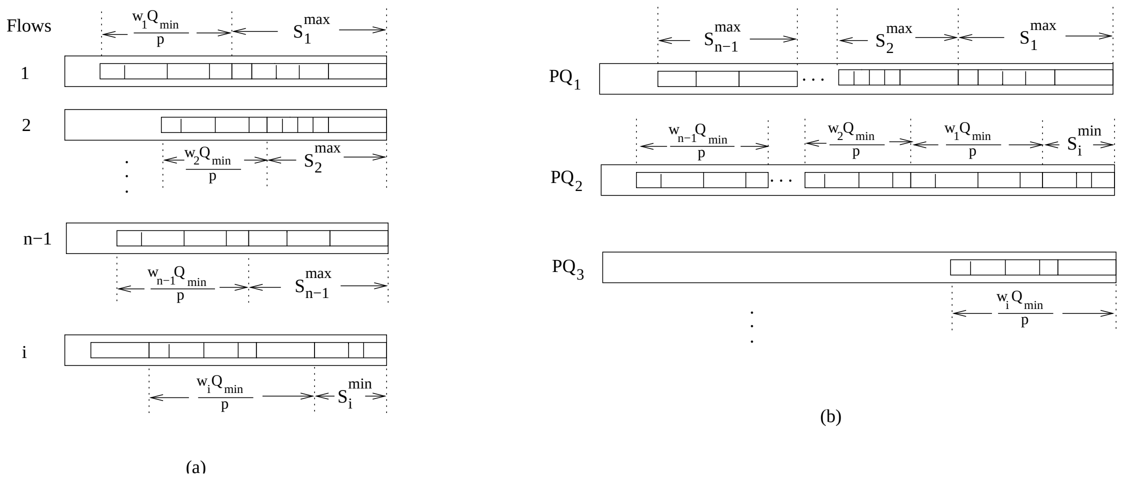


Figure A.4: (a) Input pattern (b) Packet classification in the priority queues

Figure A.4(a) illustrates a part of the input traffic present in the queues of the n flows at the start of outer round k_0. Figure A.4(b) shows how the *Classifier* module of the Pre-order DRR scheduler classifies these packets into the priority queues using Equation (A.22). From Figure A.4(b) it can be seen that, except for flow i, all the other $(n-1)$ flows have packets classified into the highest priority queue PQ_1. Prior to the service of the first packet of flow i, each flow j, $j \in \mathbf{F}$, $j \neq i$, transmits S_j^{max} worth of data. Hence, the cumulative delay until flow i receives service, X, is given by,

$$X = \sum_{\substack{j \in \mathbf{F} \\ j \neq i}} \frac{S_j^{max}}{r}$$

Substituting for S_j^{max} from Equation (A.42), we have,

$$X = \frac{1}{r} \left(\frac{(W - w_i) Q_{min}}{p} + (n-1)(m-1) \right) \tag{A.44}$$

Also the total flow i data that is served from PQ_2 equals S_i^{min}.

Even though X represents the time for which flow i has to wait until it starts receiving service, Equation (3.10) does not hold true if we substitute X as Θ_i. This is because in time interval $(\tau_i, \tau_i + X)$ flow i has not yet started receiving at its guaranteed rate. We assume that the latency, Θ_i is given by,

$$\Theta_i = X + Y \tag{A.45}$$

A plot of the service received by flow i against time is illustrated in Figure A.5. In order to determine the value of Y we shall consider the time interval $(\tau_i, \tau_i^{(k_0,2)})$ which satisfies the aforementioned requirements for deriving a tight upper bound on the latency. Referring to Figure A.5, we have,

$$Y + \frac{S_i^{min}}{\rho_i} = \frac{1}{r} \sum_{\substack{j \in \mathbf{F} \\ j \neq i}} w_j Q_{min} + \frac{S_i^{min}}{r}$$

Substituting for S_i^{min} from Equation (A.43), we have,

$$Y + \frac{\frac{w_i Q_{min}}{p}}{\rho_i} - \frac{m-1}{\rho_i} = \frac{\frac{W Q_{min}}{p}}{r} - \frac{m-1}{r}$$

Now, since $\rho_i = \frac{w_i}{W} r$, simplifying further, we have,

$$Y = \frac{(m-1)}{r} \left(\frac{W}{w_i} - 1 \right) \tag{A.46}$$

Substituting for X and Y from Equations (A.44) and (A.46) in Equation (A.45), it can be readily verified that the latency bound is exactly met.

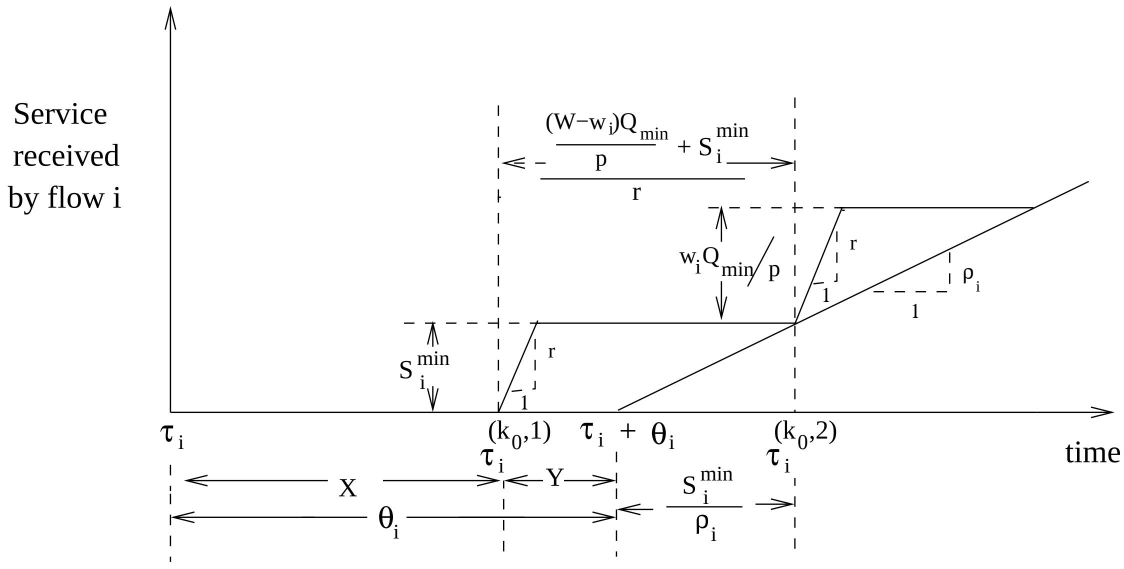

Service received by flow i
$\frac{\frac{(W-w_i)Q_{min}}{p} + S_i^{min}}{r}$
$w_i Q_{min}/p$
S_i^{min}
r
1
ρ_i
τ_i
$(k_0,1)$
$\tau_i + \theta_i$
$(k_0,2)$
time
X
Y
θ_i
$\frac{S_i^{min}}{\rho_i}$

Figure A.5: Plot of the service received by flow i with Pre-order DRR

Vita

Salil Subhash Kanhere was born in Bombay, India and is a citizen of India. He graduated from VJTI (Bombay, India) in 1998 with a bachelor's degree in electrical engineering. Subsequently, he joined the Department of ECE as a graduate student and in 1999, became a member of the Computer Communications Laboratory. He is currently pursuing a PhD in electrical engineering and expects to graduate in June of 2003. Salil's current research interests are in quality of service in computer networks, interconnection networks of parallel and distributed systems, switch and router design, mobile computing and systems, and computer architecture. His PhD dissertation is in the area of fair, efficient, and low-latency scheduling in high-speed networks with a particular focus on achieving low implementation complexity for practical use in switches and routers. During his years at Drexel, Salil has served as a teaching assistant in physics as well as several computer engineering courses. Salil was a recepient of the Teaching Assitance Excellence Award in 1999 and a Special Recognition in Teaching Assistance Award in 2001. He also received the Graduate Student Research Award from the College of Engineering and the Allen Rothworf Outstanding Graduate Student of the Year Award from the ECE Department, both in 2003. He has submitted six journal papers (of which two have already been published) and authored six conference papers.