

## **INFORMATION TO USERS**

**This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.**

**The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.**

**In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.**

**Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.**

**Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.**

**ProQuest Information and Learning  
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA  
800-521-0600**

**UMI<sup>®</sup>**



**University of Alberta**

**A MEASUREMENT STUDY ON SCHEDULING LATENCY-CRITICAL TRAFFIC**

by

**Chong Zhang**



A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta  
Fall 2001



**National Library  
of Canada**

**Acquisitions and  
Bibliographic Services**

**395 Wellington Street  
Ottawa ON K1A 0N4  
Canada**

**Bibliothèque nationale  
du Canada**

**Acquisitions et  
services bibliographiques**

**395, rue Wellington  
Ottawa ON K1A 0N4  
Canada**

*Your file Votre référence*

*Our file Notre référence*

**The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.**

**The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.**

**L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.**

**L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.**

0-612-69625-1

**Canada**

**University of Alberta**

**Library Release Form**

**Name of Author:** Chong Zhang

**Title of Thesis:** A Measurement Study on Scheduling Latency-Critical Traffic

**Degree:** Master of Science

**Year this Degree Granted:** 2001

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as herein before provided, neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.



---

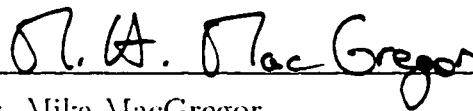
Chong Zhang  
Department of Computing Science  
232 Athabasca Hall  
University of Alberta  
Edmonton, AB  
Canada, T6G 2E8

**Date:** Sep. 11, 2001

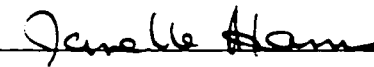
University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **A Measurement Study on Scheduling Latency-Critical Traffic** submitted by Chong Zhang in partial fulfillment of the requirements for the degree of **Master of Science**.



Dr. Mike MacGregor



Dr. Janelle Harms



Dr. Ivan Fair

Date: Aug 13, 2001

# Abstract

In recent years, many packet scheduling algorithms have been proposed to provide a low end-to-end delay bound. Simplicity is one significant issue in high speed networks. For example, a sorted list is used in some algorithms but it requires  $O(\log N)$  time to process a packet, where  $N$  is the number of the active flows at the router. Deficit Round Robin (DRR), proposed by Shreedhar and Varghese [MG96] is a low-complexity packet scheduler which requires only  $O(1)$  work to process a packet and is simple enough to be implemented in hardware. DRR was extended to DRR+ to accommodate latency-critical flows. This extension, however, constrains a latency-critical (LC) flow to generate very smooth arrivals. By using the original concept of deficit to enforce each flow's commitment to its contract, the scheduler DRR++ has much lower delay and delay jitter than DRR+. The resulting scheduler is still of low complexity, and is capable of handling bursty arrivals. This thesis presents the results of a measurement study that examines these issues. We also present a lower bound for the delay of DRR++.

# Acknowledgements

I would like to express my sincere gratitude to my supervisor, Professor Mike MacGregor, who has provided me with invaluable advice and generous help in many aspects since I started to take courses two years ago. Thanks to the excellent experimental environment and the inspiring talks from him, my work became much easier. I consider myself fortunate to have Mike as my supervisor. If it were not for him, my thesis would not have been possible.

I am very thankful for the committee members for their reviewing my thesis.

Special thanks go to my wife. Without her love, understanding, and encouragement, my life would not have been so brilliant.

I am grateful to my parents, who are always proud of me and offer me endless love and support.

Many thanks to my group mates for making such a warm and friendly environment, for raising those interesting and insightful questions, and for the badminton games we played together. Thanks are due to all my other friends. Their friendship supported me throughout the two years.

Finally I would like to thank the Department of Computing Science, for giving me the opportunity to enrich my knowledge and experience here.



# Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduction</b>   | <b>1</b>  |
| 1.1      | Router Architecture and Scheduling . . . . .  | 1         |
| 1.2      | Problem Statement . . . . .   | 7         |
| 1.3      | Contribution of the Thesis . . . . .  | 10        |
| 1.4      | Thesis Outline . . . . .  | 10        |
| <b>2</b> | <b>Related Work</b>   | <b>11</b> |
| 2.1      | Scheduler Classification . . . . .  | 11        |
| 2.1.1    | Work-Conserving vs. Non-Work-Conserving Scheduling  | 11        |
| 2.1.2    | Rate-based vs. Round Robin-based Scheduling . . . . .   | 12        |
| 2.2      | GPS and PGPS . . . . .  | 12        |
| 2.3      | Rate-Based Scheduling . . . . .   | 13        |
| 2.3.1    | The Latency-Rate server (LR server) . . . . .   | 13        |
| 2.3.2    | Virtual Clock (VC) . . . . .  | 14        |
| 2.3.3    | Self-Clocked Fair Queueing (SCFQ) . . . . .   | 14        |
| 2.3.4    | Rate-Proportional Servers (RPS) . . . . .   | 17        |
| 2.3.5    | Rate-Controlled Static-Priority Queueing (RCSP) . . . . .   | 18        |
| 2.4      | Round Robin-Based Scheduling . . . . .  | 19        |
| 2.4.1    | Weighted Round Robin (WRR) . . . . .  | 20        |
| 2.4.2    | Hierarchical Round Robin (HRR) . . . . .  | 20        |
| 2.4.3    | Carry-Over Round Robin (CORR) . . . . .   | 20        |
| 2.4.4    | Deficit Round Robin (DRR) . . . . .   | 21        |
| 2.5      | Scheduling Real-Time Traffic . . . . .  | 22        |
| 2.5.1    | Service Curve based Earliest Deadline first (SCED) and<br>Hierarchical Fair Service Curve (H-FSC) . . . . . | 22        |
| 2.5.2    | Multilayer Gated Frame Queueing (MGFQ) . . . . .  | 24        |

|          |  |           |
|----------|--|-----------|
| 2.5.3    | RCSP and Dynamic R&S . . . . .                                 | 25        |
| 2.5.4    | DRR+ and DRR++ . . . . .                                       | 25        |
| 2.6      | Summary . . . . .  | 26        |
| <b>3</b> | <b>Delay Analysis of DRR++</b>                                 | <b>28</b> |
| 3.1      | The Algorithms . . . . .                                       | 28        |
| 3.1.1    | DRR . . . . .  | 28        |
| 3.1.2    | DRR++ . . . . .  | 29        |
| 3.2      | Definitions . . . . .  | 35        |
| 3.3      | Mean Delay . . . . .   | 38        |
| 3.4      | Delay Bound of the LC flow . . . . .                           | 40        |
| 3.5      | Summary . . . . .  | 42        |
| <b>4</b> | <b>Measurement Study of DRR+ and DRR++</b>                     | <b>43</b> |
| 4.1      | System Implementation . . . . .                                | 43        |
| 4.2      | Measurement Topology . . . . .                                 | 44        |
| 4.3      | Methodology . . . . .  | 47        |
| 4.4      | Traffic Generation . . . . .                                   | 47        |
| 4.4.1    | Why UDP is more appropriate than TCP . . . . .                 | 48        |
| 4.4.2    | Smooth Traffic . . . . .                                       | 49        |
| 4.4.3    | Latency-Critical Traffic and Burstiness . . . . .              | 50        |
| 4.4.4    | Traffic Models . . . . .                                       | 50        |
| 4.5      | Measuring Delays Accurately . . . . .                          | 52        |
| 4.6      | Summary . . . . .  | 56        |
| <b>5</b> | <b>Measurement Results</b>                                     | <b>57</b> |
| 5.1      | Exponential LC Traffic . . . . .                               | 58        |
| 5.1.1    | Effect of background traffic intensity . . . . .               | 59        |
| 5.1.2    | Effect of quantum, $Q_{LC}$ , under low utilization . . . . .  | 61        |
| 5.1.3    | Effect of quantum, $Q_{LC}$ , under high utilization . . . . . | 63        |
| 5.2      | Bursty MPEG LC Traffic . . . . .                               | 64        |
| 5.2.1    | Effect of background traffic intensity . . . . .               | 65        |
| 5.2.2    | Effect of quantum, $Q_{LC}$ , under low utilization . . . . .  | 65        |
| 5.2.3    | Effect of quantum, $Q_{LC}$ , under high utilization . . . . . | 66        |
| 5.3      | Summary . . . . .  | 67        |

|          |                                    |           |
|----------|------------------------------------|-----------|
| <b>6</b> | <b>Conclusions and Future Work</b> | <b>75</b> |
| 6.1      | Conclusions . . . . .              | 75        |
| 6.2      | Future Work . . . . .              | 76        |
|          | <b>Bibliography</b>                | <b>77</b> |

# List of Figures

|     |  |    |
|-----|--|----|
| 1.1 | Delay components. . . . .  | 2  |
| 1.2 | Shared Memory Router Architecture. . . . .                       | 4  |
| 1.3 | Shared Bus Router Architecture. . . . .                          | 4  |
| 1.4 | Four Shared Fabric Technologies. . . . .                         | 6  |
| 1.5 | Shared Fabric Router Architecture. . . . .                       | 7  |
| 1.6 | Scheduler function on one output port. . . . .                   | 8  |
| 2.1 | Service Order of VC and SCFQ. . . . .                            | 15 |
| 2.2 | Delay-Jitter Control in RCSP. . . . .                            | 19 |
| 3.1 | Example of arrival behavior and service behavior. . . . .        | 36 |
| 3.2 | A state transmission diagram in DRR++. . . . .                   | 37 |
| 4.1 | DRR+/DRR++ Processes. . . . .                                    | 45 |
| 4.2 | Two Common Topologies: Single node and Multiple nodes. . .       | 46 |
| 4.3 | Measurement Topology. . . . .                                    | 47 |
| 4.4 | LC Seeds Offset Measurement. . . . .                             | 54 |
| 4.5 | BE Seeds Offset Measurement. . . . .                             | 55 |
| 4.6 | Self Clock Drift Measurement. . . . .                            | 55 |
| 5.1 | DRR++, varying background load. . . . .                          | 68 |
| 5.2 | DRR+, varying background load. . . . .                           | 68 |
| 5.3 | DRR++/DRR+, low bottleneck utilization, varying $Q_{LC}$ . . . . | 69 |
| 5.4 | DRR++, low bottleneck utilization, varying $Q_{LC}$ . . . . .    | 69 |
| 5.5 | DRR+, low bottleneck utilization, varying $Q_{LC}$ . . . . .     | 70 |
| 5.6 | DRR++, high bottleneck utilization, varying $Q_{LC}$ . . . . .   | 70 |
| 5.7 | DRR+, high bottleneck utilization, varying $Q_{LC}$ . . . . .    | 71 |
| 5.8 | DRR++, high bottleneck utilization, varying $Q_{LC}$ . . . . .   | 71 |

|      |   |    |
|------|---|----|
| 5.9  | DRR+. high bottleneck utilization, varying $Q_{LC}$ . . . . .                       | 72 |
| 5.10 | DRR++ carrying MPEG, varying background load. . . . .                               | 72 |
| 5.11 | DRR+ carrying MPEG, varying background load. . . . .                                | 73 |
| 5.12 | DRR++/DRR+ carrying MPEG. low bottleneck utilization,<br>varying $Q_{LC}$ . . . . . | 73 |
| 5.13 | DRR++ carrying MPEG, high bottleneck utilization, varying<br>$Q_{LC}$ . . . . .     | 74 |
| 5.14 | DRR+ carrying MPEG, high bottleneck utilization, varying<br>$Q_{LC}$ . . . . .      | 74 |

# List of Tables

|     |  |    |
|-----|--|----|
| 3.1 | Modified DRR++ Algorithm. . . . .  | 31 |
| 3.2 | Modified DRR++ Algorithm ( <i>Continued</i> ). . . . .   | 32 |
| 3.3 | DRR++ Algorithm in [MW00]. . . . .   | 33 |
| 3.4 | DRR++ Algorithm in [MW00] ( <i>Continued</i> ). . . . .  | 34 |
| 3.5 | Comparison of the execution sequence of the DRR++ in [MW00]<br>and the modified DRR++. . . . . | 34 |
| 5.1 | Traffic rate and burstiness for exponential traffic. . . . .                                   | 58 |
| 5.2 | LC delay in DRR+ and DRR++. . . . .  | 61 |

# Chapter 1

## Introduction

The work described in this thesis relates to packet scheduling, which is one of the main functions of a router. In this chapter, we will give an overview of router architecture and some techniques related to scheduling algorithms. The general functions of a router are briefly introduced for a better understanding of the importance of scheduling.

### 1.1 Router Architecture and Scheduling

Because the Internet is evolving rapidly, more and more applications with diverse requirements are driving the need for substantial changes in the Internet infrastructure. Currently, the majority of networks provide a uniform “best-effort” level of service where packets are processed as quickly as possible but there is no service guarantee. An alternative approach is to offer Quality of Service (QoS), to reliably provide a guaranteed level of service to the user. One of main attributes considered in QoS performance is *latency*, which is the *delay* that the data will experience through the network. End-to-end delay is composed of several factors. Figure 1.1 illustrates these factors in a simple two node network.

- *Propagation Delay* is the delay incurred in the transmission channel. It depends on the distance between the two nodes, so it is predictable.

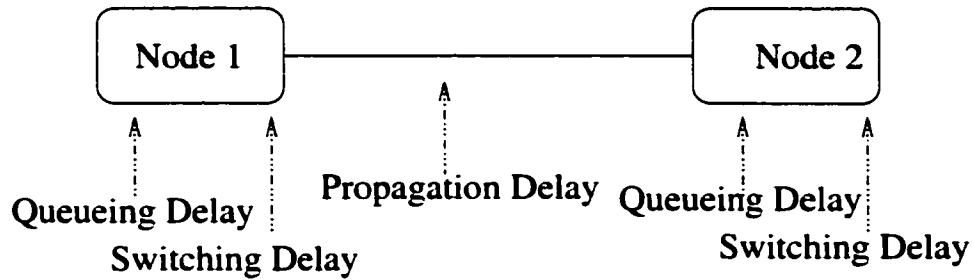


Figure 1.1: Delay components.

- *Switching Delay* is determined by the type of operations in the network node and is predictable too.
- *Queueing Delay* is the time period during which the data waits in the network node queue. It relates to the intensity of traffic and the data processing scheme in the node.

Although one method to decrease delay is to increase link bandwidth and processing speed of the intermediate nodes in the network, many researchers have concentrated on decreasing queueing delay in the nodes.

In the current Internet, the main device that we rely on to satisfy changing requirements is the *router*. The primary role of a router, which works in the network layer of the OSI model [AT96], is to forward packets from input links to output links. A router has two main functions:

- Determining the path or route that packets are to follow. This function, called *routing*, concerns the routing protocols or algorithms, such as RIP (Routing Information Protocol), OSPF (Open Shortest Path First) and BGP (Border Gateway Protocol). This requires inter-router communication in order to exchange routing data and determine the best path to a given destination.
- Actually transmitting packets from the incoming links to the appropri-



ate outgoing. This function is called *forwarding*, and concerns several processes inside the router.

Forwarding uses one of several technologies: shared memory, shared bus or shared fabric [SK96] [UB98].

Shared memory forwarding , as its name implies, uses a common memory for both input and output ports (Figure 1.2). The centralized processor reads the packets and forwards them based on the routing table in the shared memory.

In shared bus forwarding , the input and output links may have their own memories or caches but use a common bus to forward the packets between them (Figure 1.3). This scheme uses distributed control on the links. The forwarding function is implemented by the I/O card. The central CPU is only used to update the routing tables on the I/O cards.

In a shared fabric architecture, multiple parallel paths are provided between input and output links. Figure 1.4 gives some example switching fabrics: crossbar, knockout, multistage and Banyan [SK96] [UB98]. A combination of these technologies may be used to create more complex architectures.

Figure 1.5 briefly shows the architecture of a shared fabric router. There are three sub-functions: input, a general switching fabric and output.

- The input function reads packets from the physical layer and applies a routing table lookup algorithm to determine the output link. It is also responsible for classifying and marking user packets so that they are inserted into their intended service class or queue. The arbiter decides when to release packets from input queues in cases where packets from different input ports are destined to the same output port.
- The switching function moves the packets from an input port to an output port through a switch fabric [SK96].
- The output function transmits the packets that are queued in the buffers

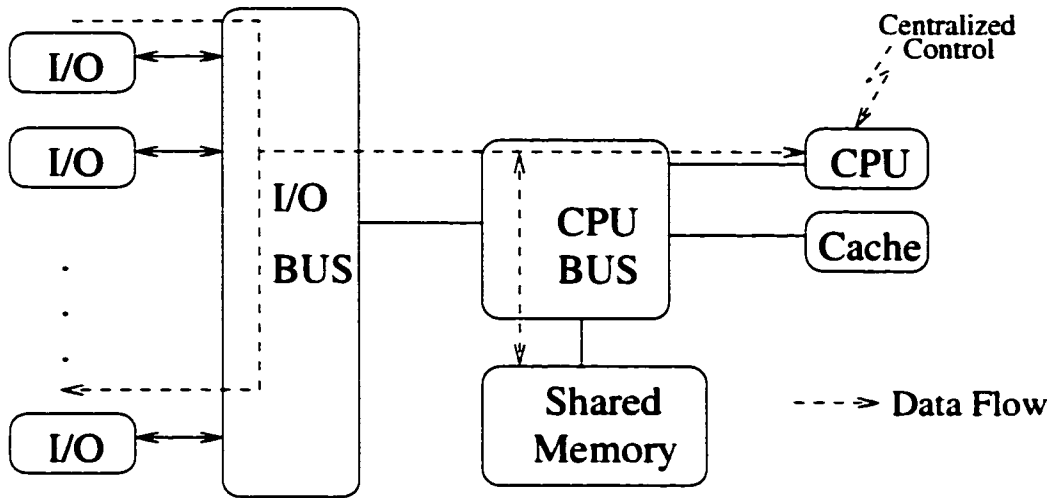


Figure 1.2: Shared Memory Router Architecture.

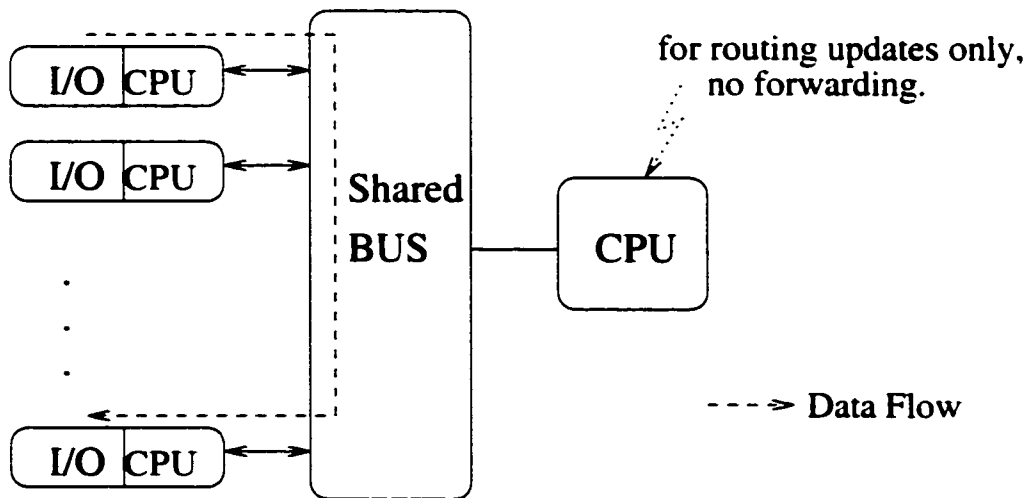


Figure 1.3: Shared Bus Router Architecture.

of an output link and controls the network resources that each service class or queue can consume.

Figure 1.5 shows only one queue for each input or output link, but in real routers there may be multiple queues managed according to different mechanisms, such as flow-based and class-based scheduling. Figure 1.6 depicts a more detailed view of one output link. The scheduler takes packets from multiple output queues and forwards them to the data link layer for transmission. Note that the sequence,  $(p1, p2, p3, p4, p5, \dots)$  in which packets are forwarded is decided by the scheduler.

All of the packets share resources, including buffers and bandwidth. Buffers at input ports are used to avoid dropping packets due to burst arrivals, which may happen even if sources agree to control their behavior. One of the causes of burstiness in a short time scale is that if a flow specifies an average rate, there will be periods during which the source generates data above the average rate, as well as periods below the average. Buffers at output ports store the packets which have been forwarded from the input ports.

A router needs to manage these resources in order to support different service requirements in the new Internet. The mechanisms required consist of buffer management algorithms and scheduling algorithms respectively. *Buffer management* algorithms decide whether packets can be accommodated for transmission and drop packets if congestion occurs. *Scheduling* algorithms control the actual transmission of packets stored in output buffers.

The topic of this thesis is the relationship between the scheduling algorithm and the *queueing* behavior on an output link of a router. The scheduling algorithm is also called a “service discipline” or “queueing algorithm” [HD93], [HD94], [DV98]. A scheduler distributes bandwidth for connections sharing the same port by determining the transmission order of the packets. Different scheduling algorithms propose different orders. For example, simple First-Come-First-Served (FCFS) uses only one queue on each output port and transmits packets in their order of arrival. More complex algorithms such as

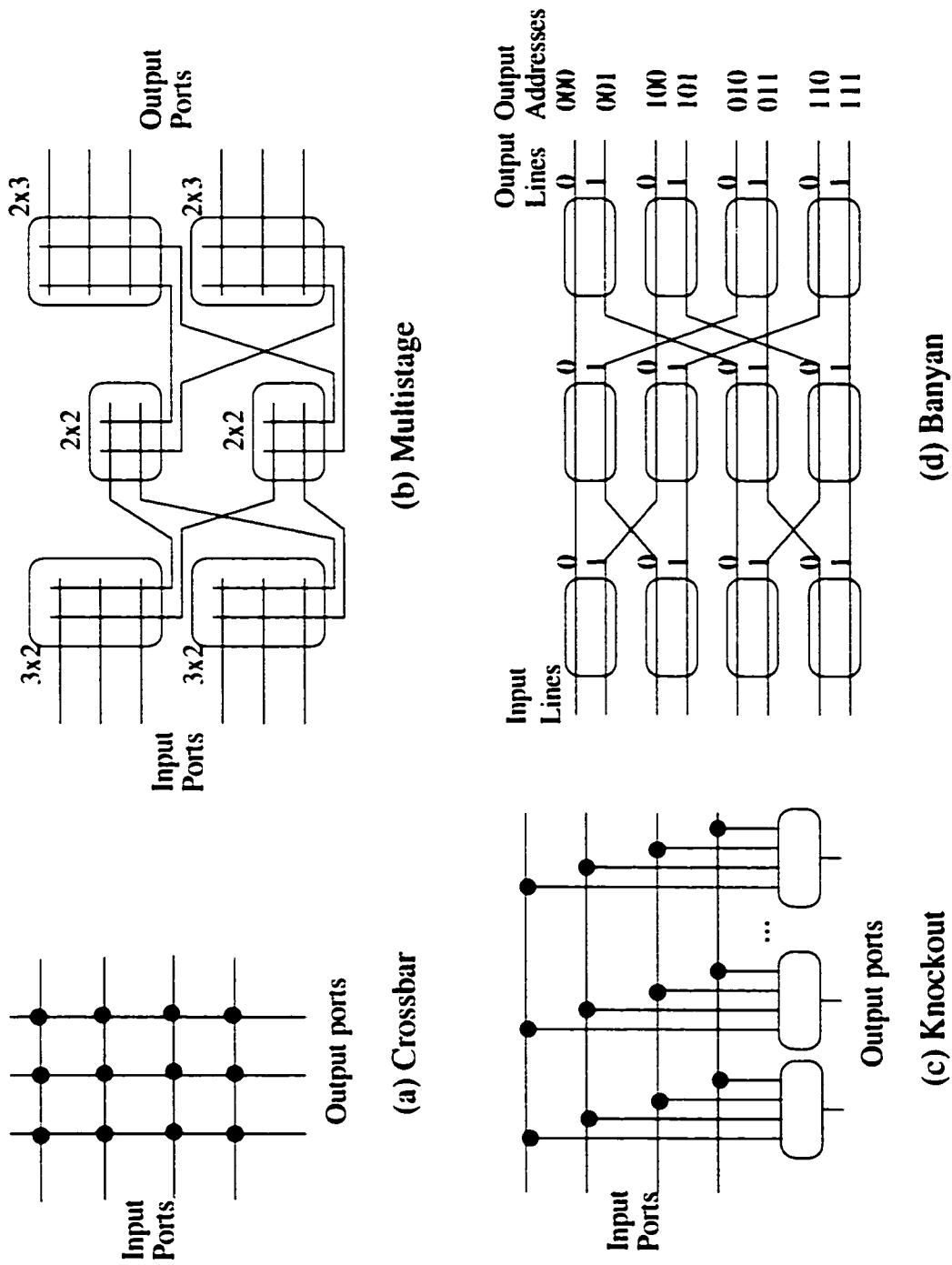


Figure 1.4: Four Shared Fabric Technologies.

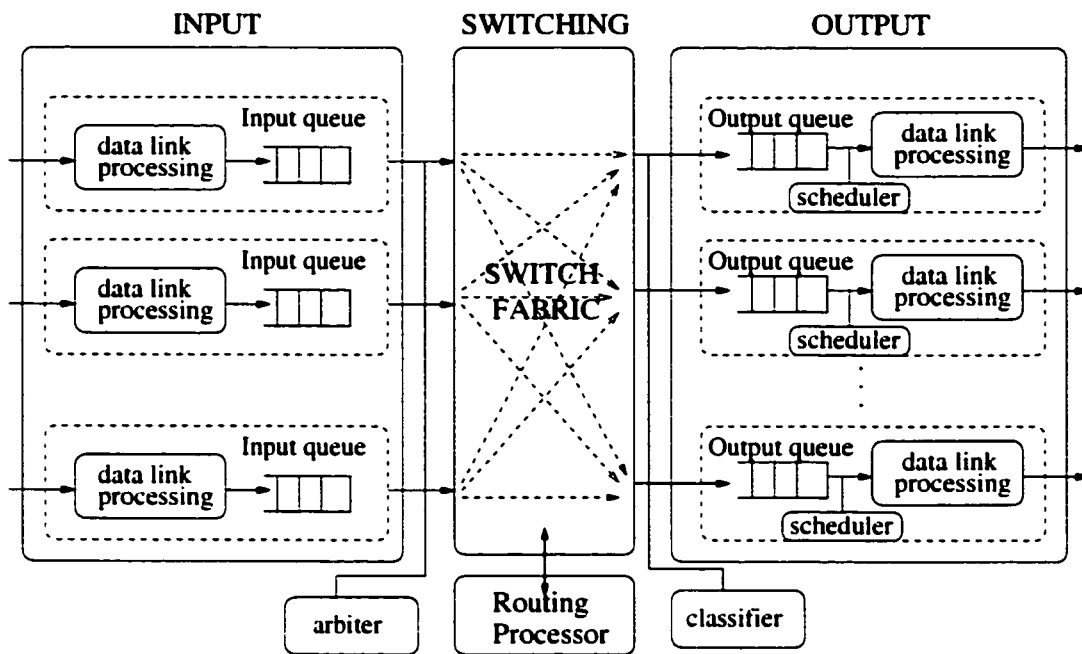


Figure 1.5: Shared Fabric Router Architecture.

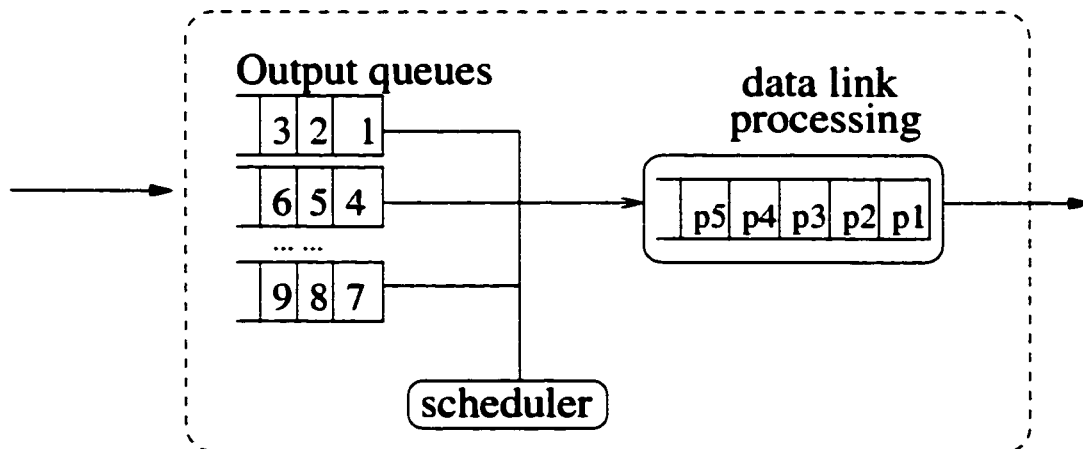


Figure 1.6: Scheduler function on one output port.

Weighted Fair Queueing (WFQ) use multiple output queues and may transmit packets in a different order (see Chapter 2).

## 1.2 Problem Statement

Network scheduling is an important part of router architecture. It decides the forwarding order of packets waiting in the output queue. Different orderings may contribute different queueing delays to the end-to-end delay of packets. It also determines how the output bandwidth is distributed over the active flows, and may or may not allow for the behavior of one flow to affect another. The metrics for evaluating a scheduling algorithm include delay, implementation complexity and fairness. In a real system, the primary objective is to provide a bound on delay with an implementation of minimum complexity per packet with the system control overhead.

- *Delay*: The main goal for a scheduler is to guarantee QoS performance for sessions in terms of queueing delay (refer to section 1.1).
- *Time Complexity*: The low time complexity is important because in a high speed network, the packet transmission time is smaller, so the time for the scheduler to choose the packets to forward should be inexpensive in terms of hardware implementation.
- *Fairness*: Fairness concerns how the scheduler allocates the link capacity to the sessions. Although it is a desirable feature of a scheduler serving best effort sessions, it is not an important concern when the high priority sessions are considered because they may pay an appropriate fee for the resource while the best effort sessions do not.

“Latency-critical” applications in QoS networks require more predictable and stringent performance in terms of delay. Some examples of latency-critical applications are: voice, video, interactive communication and virtual remote

terminal etc. In order to provide guarantees to the latency-critical flows, two other performance metrics must be added: these are *robustness* and *isolation*.

- *Robustness*: A robust scheduler ensures distribution of output link bandwidth, so that a stream is not altered dramatically on its way to the downstream nodes. Robustness is desirable because no source can guarantee the behavior of the network. First, the statistical characteristics of a stream can be altered during its transit through the network to such an extent that intermediate or terminal nodes view the stream as breaking any agreed contract. [HZ95] gives an example of how a smooth traffic becomes bursty after several intermediate nodes. Second, a bursty flow may also be seen by a node if a badly behaving client does not obey its contract.
- *Isolation*: Isolation between streams is a much stronger feature, and guarantees that changes in one stream cannot affect the behavior of other streams traversing the same interface. This allows latency-critical (LC) flows to obtain guaranteed service.

There have been many research studies of scheduling algorithms. Briefly, there are two main approaches: timestamp-based and round robin-based, which will be discussed in Chapter 2. One of the algorithms is Deficit Round Robin (DRR) [MG96], which provides low complexity and fairness while allowing variable packet length. DRR+ in [MG96] extends DRR to get better performance for latency-critical flows by allocating more service time to the LC flow. In DRR+, the LC flow may be punished if it breaks its contract. DRR++ [MW00] is more robust because it allows short bursts in the LC flow.

### 1.3 Contribution of the Thesis

The timestamp-based scheduling algorithms require a calculation of a timestamp or other similar parameters. Most of these algorithms need a sorted list, so they are more complex than the round robin-based algorithms. Much of the

existing work focuses only on analysis or simulation. Some studies offer only analytical results which are not verified by either simulation or measurement. The contributions of this thesis include:

- An analysis of DRR++ in terms of both mean delay and delay bound. The DRR++ algorithm in [MW00] is modified in order to simplify both the analysis and the implementation (See Chapter 3).
- Measurement of the DRR+ and DRR++ algorithms in a real network environment to determine whether the analyses match the measurement results. We created a test-bed, characterized the behavior of unsynchronized clocks, and found a practical solution to the problem (see Chapter 4).
- An investigation of the effects of background traffic intensity and the quantum value assigned to the LC flow. Two different types of burst traffic are used to test the performance of DRR+ and DRR++. We find that DRR++ offers a much better delay guarantee for MPEG traffic than DRR+ (see Chapter 5).

## 1.4 Thesis Outline

In Chapter 2 we provide an overview and discussion of some existing scheduling algorithms. In Chapter 3, we present an analysis of the delay of DRR++. The system architecture as well as the measurement environment and methodology are discussed in Chapter 4. In Chapter 5, we discuss the results of our measurements. Chapter 6 provides conclusions and ideas for future work.



# Chapter 2

## Related Work

Many scheduling algorithms have been developed to provide quality of service guarantees in high-speed networks. Naturally, each method has particular strengths and shortcomings. In this chapter, we first discuss the scheduler classification and then introduce some previous work on scheduler for both best effort and real time applications.

### 2.1 Scheduler Classification

Schedulers can be classified, firstly, on the basis of whether they are work-conserving or not, and secondly on the basis of whether they attempt to use a round robin policy or not.

#### 2.1.1 Work-Conserving vs. Non-Work-Conserving Scheduling

A work-conserving scheduler is never idle when there is a packet to serve, while a non-work-conserving scheduler may be idle even when there are packets waiting to be served. Though a non-work-conserving scheduler wastes bandwidth, “it makes the traffic arriving at downstream switches more predictable, thus reducing both the buffer size necessary at output queues and the delay jitter experienced by a connection” [SK96]. An example of the non-work-conserving scheduler, Rate-Controlled Static-Priority Queueing (RCSP), will be discussed in section 2.3.5.

### 2.1.2 Rate-based vs. Round Robin-based Scheduling

A rate-based scheduler keeps some state information about packets, sessions or the whole system. The state information can take a variety of forms: *virtual time* in Virtual Clock [LZ90], *eligibility time* in Rate-Controlled Static-Priority Queueing [HD93] or the *potential functions* in Self-Clocked Fair Queueing [SG94]. Based on the state information, a sorted list is used to decide the forwarding order of the packets. In this kind of policy, the insert and delete operations on the sorted list require  $O(\log(N))$  time, where  $N$  is the maximum number of sessions or packets according to the definition of the sorted list.

Round robin-based schedulers all have the same basic operation: they scan the active sessions one by one. This operation is much simpler than keeping a sorted list. Round robin algorithms differ according to how they scan the sessions and what information is kept.

## 2.2 GPS and PGPS

One ideal, though impractical, network scheduling algorithm is Generalized Processor Sharing (GPS). It is ideal because it is a bit-by-bit round robin method. GPS is characterized by  $N$  positive real numbers,  $\phi_1, \phi_2, \dots, \phi_n$ , each representing the share of bandwidth reserved by session  $i$ ,  $i = 1, 2, \dots, n$ . At any time  $\tau$ , session  $i$  receives the service rate  $W_i(\tau) = \frac{\phi_i}{\sum_{j \in B(\tau)} \phi_j} C$ , where  $B(\tau)$  is the set of connections for which there are packets waiting in the queue at time  $\tau$  and  $C$  is the link speed. "GPS is impractical as it assumes that the server can serve all connections with non-empty queues simultaneously and that the traffic is infinitely divisible. In a more realistic packet system, only one connection can receive service at a time and an entire packet must be served before another packet can be served." [HZ95]

A more practical algorithm is packet-by-packet GPS (PGPS) or Weighted Fair Queueing (WFQ) [AS89], which is widely used in current products. WFQ calculates the *finish time* of each packet in the corresponding ideal GPS system

and sends the packet with the smallest value of finish time. It is practical because it considers packet size when computing the finish time and does not consider traffic as infinitely divisible. If  $S_i, F_i$  denote the start and finish time of packet  $i$  in a GPS system, and  $P_i$  denotes the packet length, we have  $F_i = S_i + P_i$  and  $S_i = \max(F_{i-1}, R(t))$ , where  $R(t)$  is the number of bit rounds in the bit-by-bit round robin service discipline up to time  $t$  [WFQ].

“The GPS and PGPS schemes have drawn a lot of interest because they have a simple fluid-flow interpretation and are some of the first methods which have been rigorously analyzed and shown to have tight end-to-end performance bounds. However, this kind of method has one main drawback: the algorithm for calculating the finishing times is computationally intensive, rendering it rather difficult to implement for high speed networks” .[HR99]

## 2.3 Rate-Based Scheduling

### 2.3.1 The Latency-Rate server (LR server)

In [DA95] and [DA98], D. Stiliadis et al propose the LR server which is a general class of scheduling algorithms. An LR server has two parameters – the *latency* and the *allocated rate*. If we define a *busy period* for session  $i$  as any period of time during which session  $i$  always has packets to send, a scheduler  $S$  is an LR server if and only if

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

where  $\tau$  is the starting time of the  $j$ th busy period,  $W_{i,j}^S(\tau, t)$  is the amount of service received by session  $i$  during the interval  $(\tau, t)$  and  $\Theta^S$  and  $\rho_i$  are the latency and rate, respectively.

Given the latency and rate, the LR server gives an upper bound on both the session backlog and the delay. The authors give different latency values for different scheduling algorithms, such as GPS, PGPS, Self-Clocked Fair Queueing, Virtual Clock and Deficit Round Robin. We will use the results of the LR server to analyze the delay bound of our new algorithm.

### 2.3.2 Virtual Clock (VC)

The Virtual Clock [LZ90] algorithm uses a time-division multiplexing (TDM) approach to allocate a virtual transmission time for each newly arriving packet. It needs to know the arrival rate,  $AR_i$ , of flow  $i$  and computes the value  $Vtick_i = 1/AR_i$ . When a packet is received at time  $AT_i$ , the transmission time of this flow,  $auxVC_i$ , is updated based on its previous value:

$$auxVC_i = \max\{AT_i, auxVC_i\} + Vtick_i \quad (2.1)$$

The system forwards all packets according to their transmission time  $auxVC_i$ . One of the main features of Virtual Clock is that it creates a firewall between the flows: bad behavior of one flow cannot disturb network service to other flows because the offending behavior increases the  $auxVC$  of the offending flow very quickly [LZ90].

Virtual Clock is different from the traditional TDM because Virtual Clock is a work-conserving algorithm. It doesn't waste bandwidth if there are packets waiting in the queues. But since Virtual Clock only keeps track of the state on per session through  $Vtick_i$ , the status of system load is not considered. [HZ95, Fig4] gives an example showing that Virtual Clock punishes a bursty session even if there are no other sessions affected by the faster transmission during the bursty period.

### 2.3.3 Self-Clocked Fair Queueing (SCFQ)

GPS, PGPS and Virtual Clock all use a hypothetical queueing model to calculate the virtual time, which leads to considerable computational complexity. SCFQ [SG94] uses another approach in which the virtual time is obtained naturally from an actual packet-based queueing system. In the SCFQ algorithm, the packets in the queue are serviced in increasing order of the associated *service tag*,  $\hat{F}_k^i$ , for packet  $i$  in session  $k$ :

$$\hat{F}_k^i = \frac{1}{r_k} L_k^i + \max\left(\hat{F}_k^{i-1}, \hat{v}(a_k^i)\right)$$

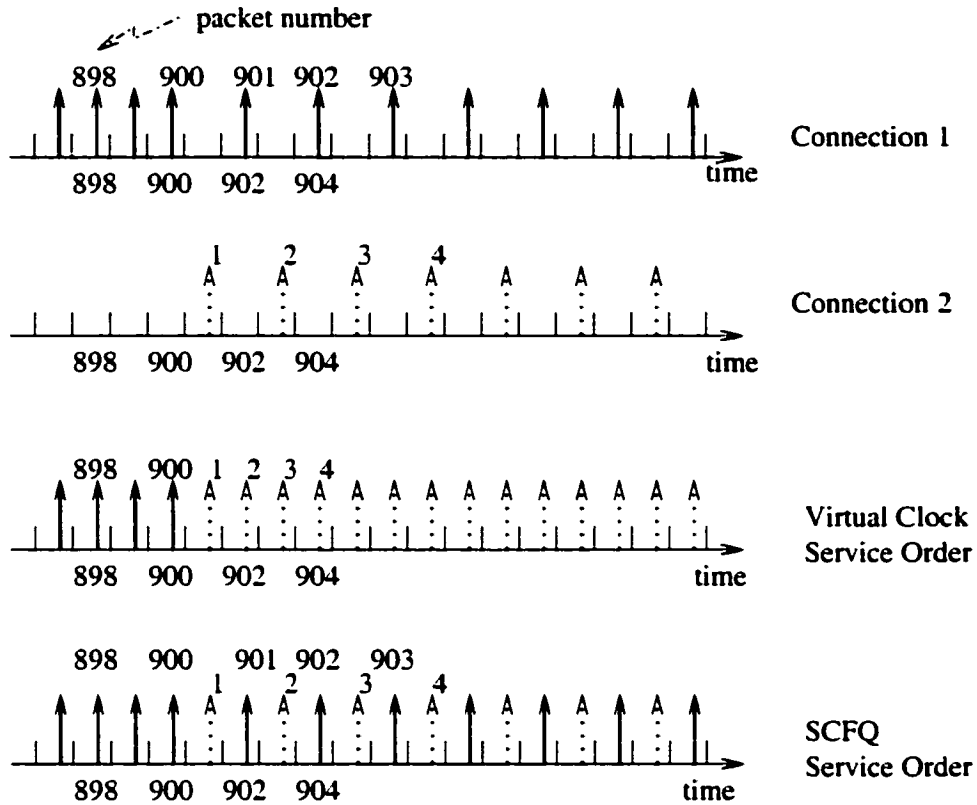


Figure 2.1: Service Order of VC and SCFQ.

where  $r_k$  is the service rate for session  $k$ .  $L_k^i$  is the length of the packet,  $a_k^i$  is the arrival time of the packet and  $\hat{v}(t)$  is the system's virtual time at time  $t$ .  $\hat{v}(t)$  is defined as being equal to the service tag of the packet receiving service at time  $t$  [SG94].

SCFQ overcomes the unfairness of Virtual Clock caused by sudden bursts. SCFQ uses  $\max(\hat{F}_k^{i-1}, \hat{v}(a_k^i))$  to circumvent this problem by replacing  $\hat{F}_k^{i-1}$  with the service tag of the packet in service, if the latter is larger. We use Figure 2.1, which is similar to [HZ95 Figure 6], to show that SCFQ has a fairer bandwidth distribution than Virtual Clock. Assume that two connections specify the same average rate of 0.5 packet/sec, and all packets have a constant size and can be served in exactly one second. Starting at time zero, there are 1000 packets from connection 1 that arrive at a rate of 1 packet/sec

and at a rate of 0.5 packets/sec from time 900. The packets from connection 2 arrive starting at time 901 at a rate of 0.5 packets/sec. If Virtual Clock is used and we define  $auxVC_i^k$  as the transmission time for packet  $k$  in flow  $i$ , we have

$$\begin{cases} auxVC_1^{900} = 1800, \\ auxVC_1^{900+k} = 1800 + 2k, k = 1, 2, \dots, \\ auxVC_2^1 = \max(900, 0) = 900, \\ auxVC_2^k = 900 + 2(k - 1), k = 1, 2, \dots, \end{cases}$$

So the 901st packet of connection 1 will be served after 450 packets of connection 2 are serviced.

If SCFQ is used, we have

$$\begin{cases} \hat{F}_1^{900} = 1800 = \hat{v}(900), \\ \hat{F}_2^1 = 2 + \hat{v}(900) = 1802 = \hat{v}(901), \\ \hat{F}_1^{901} = 2 + \max(\hat{F}_1^{900}, \hat{v}(901)) = 2 + 1802 = 1804 = \hat{v}(902), \\ \hat{F}_2^2 = 2 + \max(\hat{F}_2^1, \hat{v}(902)) = 2 + 1804 = 1806, \\ \dots \end{cases}$$

So we can get the service order shown in Figure 2.1. We also find that WFQ has the same order as SCFQ.

SCFQ reduces the complexity of the algorithm compared to PGPS from  $O(N)$  to  $O(\log N)$ , where  $N$  is the maximum number of sessions sharing the same output link. However, the delay bound of SCFQ is a linear function of  $N$ , so the delay bound grows linearly while  $N$  increases [DV98].

### 2.3.4 Rate-Proportional Servers (RPS)

RPS [DV98] uses an analytical model for the design of scheduling algorithms and the analysis of their properties. There are two kinds of RPS: one uses an ideal fluid model while the other is Packet-by-Packet RPS (PRPS). RPS uses two non-decreasing potential functions to reflect the state of the sessions

and the whole system. These are the session potential  $P_i(t)$  and the system potential  $P(t)$ , respectively.

According to [DV98], in the fluid RPS server, the session potential  $P_i(t)$  for session  $i$  is initialized as  $P_i(t') = \max(P_i(t'-), P(t'-))$  at the beginning of a busy period  $t'$ . At each timepoint  $t > t'$  of the busy period,

$$P_i(t) = P_i(t') + W_i(t', t) / \rho_i,$$

where  $W_i(t', t)$  is the amount of service received by session  $i$  during the period  $(t', t]$  and  $\rho_i$  is the service rate allocated to session  $i$ .

The calculation of a session's potential is similar to the calculation of the session service tags in SCFQ because both of them consider the status of whole system rather than an individual session. The system state is reflected by the system potential  $P(t)$  in RPS and the system virtual time  $\hat{v}(t)$  in SCFQ. In SCFQ,  $\hat{v}(t)$  is equal to the service tag of the packet receiving service at that time and is reset to zero when the busy period is over [SG94 pp25], while  $P(t)$  in RPS is always less than or equal to the potentials of all backlogged sessions [DV98 3.3].  $P(t)$  is a non-decreasing function that protects the sessions with lower session potential since those sessions will get service earlier. But  $\hat{v}(t)$  only reflects the state of the session receiving service, which may serve the session with a large service tag value.

[DV98] gives only the analysis of the RPS policy, without any experimental results. The authors do not make any suggestions about the frequency with which the system/session potentials are updated. There are two issues which need to be further considered: how to keep consistency between the session and system potentials and how to keep the system potential under the specified conditions specified in [DV98 section 3.1].

### 2.3.5 Rate-Controlled Static-Priority Queueing (RCSP)

RCSP [HD93] [HD94] uses a different model to do the scheduling work. First, it introduces a two-component model including a *Rate Controller* and a *Scheduler*. The *Scheduler* is a simple FIFO or static priority queue for forwarding

the packets. The Rate Controller calculates the *eligibility time*,  $ET_j^k$ , of packet  $k$  in switch  $j$ . There are two kinds of controllers – rate-jitter regulators and delay-jitter regulators – with different computation methods. The rate-jitter regulator is based on the arrival time,  $AT_j^k$ , and the eligibility time of the last packet in the same session  $ET_j^{k-1}$ , while the delay-jitter regulator is mainly based on the  $AT_j^{k-1}$  and the eligibility time of the same packet in the last switch  $ET_{j-1}^k$ . The main idea behind a delay-jitter regulator is to hold the packet in the regulator if the packet arrives earlier than expected. The period of time during which a packet stays in the regulator is the *holding time*, while the period of time during which a packet waits in the scheduler is the *waiting time*. In this way, the packets will not be serviced immediately and the bursty traffic is passed to the next switch. Thus, RCSP is a non-work-conserving scheduling algorithm. [HD94] defines  $d_j^k$  and  $h_j^k$  as the waiting time and holding time of the  $k$ th packet in switch  $j$ ,  $p_j^k$  is the link delay from the  $(j-1)$ th to  $j$ th switch, and  $DT_j^k$  is the departure time of the  $k$ th packet at switch  $j$ . A delay-jitter regulator has the following property:

$$\begin{cases} ET_0^k = AT_0^k \\ ET_j^k = ET_{j-1}^k + d_{j-1}^k + h_j^k + p_{j-1}, j > 0. \end{cases}$$

The summation of  $d_{j-1} + h_j^k$  is the delay bound of the packet. Figure 2.2 describes the parameters related to different time-points. Through the use of  $d_{j-1}$ , the delay variance caused at different nodes is transparent to the next switch. That is, if the packet is delayed more at a previous switch, it gets a smaller holding time in the regulator at the current switch and vice versa.

Although this method may increase the end-to-end delay of the packets, it is a common method to control delay jitter if the jitter is more important than delay, such as for real-time streaming applications.

In order to decrease the complexity of the regulator, RCSP uses a modified



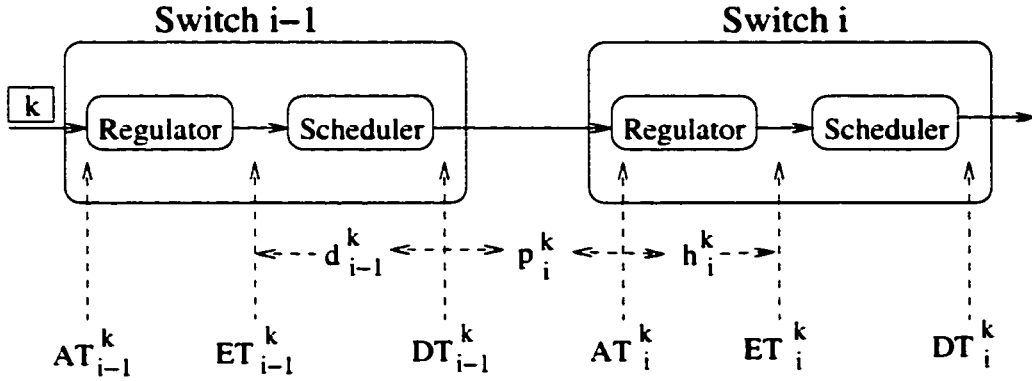


Figure 2.2: Delay-Jitter Control in RCSP.

calendar queue. But from the above formula, we can see that the switch needs to stamp the packet with the eligibility time for use at the downstream switch. and each switch needs to know the delay bound of the session.

## 2.4 Round Robin-Based Scheduling

One of the differences between rate-based scheduling and round robin-based scheduling is that the latter does not need a value to decide the transmission order of the packets. such as the timestamp of the packets or the potential of the sessions.

### 2.4.1 Weighted Round Robin (WRR)

In Weighted Round Robin (WRR) [BB94], packets receive service according to the priority queue to which they belong. Each queue has an associated weight. In every round of service, the number of packets served from a queue is proportional to its associated weight. One common method to find the queue number of a packet is to use the precedence bits in the IP packet header. Different *weights* are specified for the queues. WRR can assign higher weight to high priority sessions so they get more service. WRR also avoids starvation

for low priority sessions because they still get minimum service guarantees. However, WRR is only good for the networks with fixed packet size, such as ATM networks. The reason is that the weight assigned to a session is the number of packets, so if a low priority session has a large packet size, it still consumes more bandwidth. In this case, the weights are irrelevant to the session priorities. WRR also needs to know the mean packet size to achieve results fairly close to the ideal GPS scheduler.

### **2.4.2 Hierarchical Round Robin (HRR)**

Hierarchical Round Robin (HRR) [IH00] defines a hierarchy of service lists, each of which corresponds to different service rates. The topmost list has the highest service rate, while the bottom list has the lowest service rate and is used to forward the best effort traffic. HRR services packets from the top of the list to the bottom. Simulation results show that both the inter-packet spacing and end-to-end delay are independent of load, which means that HRR provides good delay and jitter bounds.

### **2.4.3 Carry-Over Round Robin (CORR)**

Carry-Over Round Robin (CORR) [DS96] allocates a rate  $R_i$  to each session  $i$ , expressed as the numbers of cells served per round. The key point in CORR is that  $R_i$  can be a real number rather than an integer as in simple round robin schemes. CORR has two sub-cycles in each cycle, a major cycle and a minor cycle. All sessions are allocated their integral requirement in the major cycle, and unfulfilled connections are allocated the slots left over from the major cycle. In each service cycle, the actual number of cells allocated to a session is an integer close to  $R_i$ , and the debits or credits are carried over into the next service cycle. The authors argue that each session receives exactly  $R_i$  services per cycle over a long time scale. However, CORR only works in ATM networks with a fixed cell length, which is reflected by the constant time unit used in the algorithm.

## 2.4.4 Deficit Round Robin (DRR)

WRR, HRR and CORR are designed for use in the ATM environment. WRR may fail to adjust its weight if the mean packet size changes. WRR works well in ATM networks because of the constant cell size. Both HRR and CORR use the cell unit to define the parameters in the algorithms.

Deficit Round Robin (DRR) [MG96] proposes a solution for packet-switched networks with variable packet length and requires only  $O(1)$  work to process a packet. Moreover, since there are only three lists used in the DRR algorithm: the active flow list, deficit flow list and quantum list, the algorithm requires small control overhead. The low complexity of DRR makes it possible to implement DRR in high speed networks<sup>1</sup>. DRR maintains the quantum value and the deficit value for each session. In each round, one session cannot forward packets with a length larger than the sum of the predefined quantum and the deficit left from the last round. A sorted list of the sessions is not required. Similar to CORR, each session in DRR receives the service with a quantum value during each round in a long time scale. Unlike CORR, which uses two cycles in each round in order to process the debit and credit, DRR only scans all the active sessions once in each round.

## 2.5 Scheduling Real-Time Traffic

Real-time or multimedia applications require much more stringent performance than best-effort applications. The network needs to provide this type of application a *guaranteed service* in terms of delay, delay jitter, throughput and loss rate. Much work has been done on scheduling real-time or multimedia traffic. There is also much work concerning CPU and operating system scheduling algorithms using similar approaches.

For example, MTRLS in [JE97] introduces a *process sharing* model in which any number of processes can be served simultaneously as long as the sum of

---

<sup>1</sup>Cisco 12000GSR, Microsoft Windows 2000 Server and Linux RedHat have implemented various modified DRR algorithms.

their service fractions does not exceed one. It uses a *virtual time quantum* to allocate the time slot of a process. The idea is similar to DRR but since the time tick is very small, there is no credit which can be used in the next slot. A process will consume all of its time quantum unless it is terminated or preempted by another process.

A rate-controlled scheduler is presented in [DS96]. It uses expected finishing time and other time-stamps to calculate the *RC value* (rate-controlled priority value) of different processes and then uses this value to decide the running order. In addition to the admission control, the algorithm in [DS96] adds another component, a *rate adaptation* mechanism, to adjust a flow's rate based on the feedback information provided by the use process during the process life cycle.

### 2.5.1 Service Curve based Earliest Deadline first (SCED) and Hierarchical Fair Service Curve (H-FSC)

Service Curve based Earliest Deadline first (SCED) [HR95] [HR99] tries to guarantee the service curve rather than a specific measure, such as a delay bound, for each session. This scheme introduces the notion of a *service curve* as a measure of service provided for a session. A service curve function  $S_i(\cdot)$  is a non-decreasing function which specifies the minimum number of packets served in a period. For each connection, if time  $t_1$  is in a backlogged period,  $t_0$  represents the beginning of one of the busy periods, such that  $S_i(t_1 - t_0) \leq R^{out}(t_0, t_1)$ , where  $R^{out}(t_0, t_1)$  is the number of packets served by session  $i$  during the interval  $[t_0, t_1]$  [HR95].

In SCED, if all the sessions have service curve guarantees, the packet with the earliest deadline is transformed first. [HR99] provides the algorithm for computing the deadline and proves that if each packet receives service before its deadline, then the session is guaranteed the service curve it requires. SCED defines a traffic as  $(\sigma, \rho)$  - *smooth* if during any interval  $x$ , the arrival curve,  $b(\cdot)$ , is less than  $\sigma + \rho x$ . One feature of SCED is that if the input traffic is

$(\sigma, \rho)$  – *smooth*, SCED gives the upper bound on delay and backlog [HR95].

SCED can be reduced to other schedulers, such as Virtual Clock or Earliest Deadline First [HR99]. For each generalized scheduler, SCED uses different service curve functions. For example, if service curve  $S_i(x)$  satisfies  $S_i(x) = \max\{0, (\alpha_i + \beta_i x)\}$ , where  $\alpha_i$  and  $\beta_i$  are two parameters, the deadline of a packet arriving at time slot  $u$  is calculated as follows:

$$\begin{aligned}\delta_i &= \max\{\delta_i, u - 1\} \\ \delta_i &= \delta_i + 1/\beta_i\end{aligned}\tag{2.2}$$

$$deadline = \max\{u, \delta_i\}$$

If we set  $S_i(x) = \rho_i x$ , i.e.  $\alpha_i = 0, \beta_i = \rho_i$ , so the equation (2.2) is reduced to  $deadline = \max\{u, \delta_i + 1/\rho_i\}$ , which is equivalent to the *auxVC<sub>i</sub>* computed in Virtual Clock.

The authors of Hierarchical Fair Service Curve (H-FSC) [IH00] state that SCED sometimes does not have the fairness property because SCED may punish a session for receiving excess service during a period by keeping it from receiving service during a later period. This feature also makes SCED unsuitable for a hierarchical scheme. H-FSC defines two criteria, the *real-time criterion* and the *link-sharing criterion*. The former is used to ensure the real-time requirement, while the latter is used to fairly distribute excess service. Several parameters are used in H-FSC.  $e_i$  and  $d_i$  denote the *eligible time* and the *deadline* of the first packet in session  $i$  respectively and  $v_i$  denotes the *virtual time* of session  $i$ . The virtual time can also be seen as the normalized amount of service that has been received by the class [H-FSC].

In order to achieve fairness, H-FSC tries to minimize the discrepancies between virtual times of sibling classes. H-FSC gives high priority to sessions with real-time requirements since doing so is more important. After serving real-time packets, H-FSC serves the sessions with the smallest virtual time in the link-sharing requirement class. The results show that H-FSC achieves much lower delays for real-time sessions and the link-sharing sessions get fairly distributed bandwidth. But since H-FSC is a deadline-based algorithm which

needs to sort the deadline time for all the packets, the complexity for processing a packet is  $O(\log(N))$  where  $N$  is the number of active sessions.

### 2.5.2 Multilayer Gated Frame Queueing (MGFQ)

Multilayer Gated Frame Queueing (MGFQ) [FH00] is also a time-based scheduler used to deal with real-time traffic in ATM networks. It uses the RTP/UDP/IP/AAL5 protocol stack and uses a *DD* (*due-date*) field in the AAL5 overhead. MGFQ uses two different schemes for voice traffic and video traffic. The *DD* field used in calculating the due date of the cell is carried in each voice cell, while for video traffic this field only shows up in the BOM (Beginning Of Message) cell. Both the cell format design and the due-date computing methods are different for voice and video traffic. Based on the due-date, a cell can be served without violating either the delay bound or the jitter bound. The complexity of MGFQ is also  $O(\log(N))$ , and it involves five (or seven) additions per cell for voice traffic (or video) [FH00].

The jitter bound in MGFQ is affected by a parameter  $T$ , which represents the refreshing period in which the operations are updated. The simulation results show that the traffic gets a pre-defined jitter bound, which can be tight or smooth. But the results also show that the delay doesn't decrease if a tight jitter bound is used, so the jitter bound in MGFQ is actually a jitter bound with maximum delay value. Another issue is that the authors do not give results on the effect of parameter  $T$ . So we do not know the effects on delay or jitter if other refreshing periods are used. MGFQ also has a compatibility problem since it requires one additional field in the ATM cell header.

### 2.5.3 RCSP and Dynamic R&S

We have discussed the RCSP proposed by H. Zhang et al in section 2.3.5. RCSP uses two separate components, a rate controller and a scheduler, to provide delay, jitter and throughput guarantees. One of the drawbacks of this scheme is that two separate functions may compromise the effectiveness

of regulation and scheduling (R&S) [SI00]. One consequence is that if there is more than one regulated traffic at a scheduler, the *conflicts* will result in a distorted stream at the output of the scheduler. In order to address this problem, [SI00] proposes dynamic R&S and tries to use some scheduler status information which is fed back from the scheduler to the regulator to accelerate packet releases from the system. It monitors the scheduler and allows the release of some packets before their eligibility time  $ET_k$ , which will impact positively on the cell delay and the available bandwidth.

#### 2.5.4 DRR+ and DRR++

DRR has several desirable features: it provides fairness amongst the connections, allows for varying packet sizes and has  $O(1)$  time complexity per packet. A modification called DRR+ is proposed in [MG96] to provide a latency bound for real time traffic because the delay bound for DRR increases as the number of flows increases. This is unacceptable for latency-critical flows. One possibility is to have both a best-effort transmit queue and a priority queue for latency-critical traffic. DRR+ uses the policy that the source should not send more than  $x$  bytes of data in some period  $T$ . This is policed by starting a timer whenever a packet from a particular flow arrives. If another packet from the same flow arrives before the timer expires, the flow is said to have violated its contract, and the flow is moved into the best-effort class. In DRR+, once a latency-critical flow has been moved into the best-effort class, there is no provision for it to graduate back to priority treatment.

DRR++ [MW00] uses the deficit approach of DRR for servicing latency-critical flows, along with priority transmit queueing. Doing so ensures that the available bandwidth is shared between flows, while giving priority to latency-critical flows. Most importantly, this approach allows latency-critical flows to burst occasionally without losing their preferred status.

A latency-critical flow scheduled by DRR++ is constrained to send less than one quantum between best-effort service intervals. If the latency-critical

flow exceeds this rate, the excess traffic simply remains backlogged. This method overcomes the weakness of DRR+ while still preserving fairness.

DRR++ is a non-work-conserving scheduler that can be used to provide robust service: in particular, it can provide robust service for latency-critical traffic. DRR++ is robust and is suitable for packet scheduling in the network core because, as long as a stream obeys its contract at the admission point, it will not be penalized for burst behavior impressed on the stream as it transmits the network. DRR++ is also suitable for use in the core because of its extremely low per-packet complexity inherited from DRR.

DRR++ is very effective in isolating latency-critical (LC) and best-effort (BE) traffic from each other so that misbehaving sources of one type cannot affect traffic from sources of the other type. DRR++ provides protection so that one flow is not able to steal bandwidth from others. DRR++ also exhibits fairness for BE traffic with variable length packets, while the normal round robin method provides fairness only with fixed-size packets.

## 2.6 Summary

In this chapter, we discussed several scheduling algorithms. We introduced algorithms in two categories: rate-based and round robin-based. Most of rate-based algorithms are required to keep a sorted list, so that they have  $O(\log N)$  complexity. We also described several scheduling algorithms for real-time applications. Our work on DRR++ is proposed to provide low delay for latency-critical traffic along with the low  $O(1)$  complexity inherited from DRR. The detailed algorithm and an analysis will be given in Chapter 3.



# Chapter 3

## Delay Analysis of DRR++

In this chapter, we discuss the performance analysis of DRR++ in terms of *mean delay* and *delay bound* for the LC flow. We provide a good approximation for the mean delay for the LC flow under conditions of high utilization and give the delay bound for the LC flow based on the DRR++ algorithm and the analysis of the LR server in [DA98]. These analyses will be compared with our measurement results in Chapter 5.

### 3.1 The Algorithms

#### 3.1.1 DRR

First, we briefly review the idea of DRR as proposed in [MG96]. Normal round robin algorithms service the queues in a constant time, which is an important attribute for a scheduling algorithm. The major problems, as stated in Chapter 2, are that most schedulers only deal with constant packet size and even if they allow different packet sizes, there is unfairness between different connections. DRR solves this problem while still keeping the low complexity feature. The main idea behind DRR is that a connection is allowed to transmit at an average rate. The scheduler works in rounds, with each queue being credited a new quantum in each round. The amount of service a queue can receive is limited by its current amount of credit. The credit is decremented for every byte transmitted. If a queue cannot be emptied in a given round because the

next packet requires more credit than available, the credit is preserved and incremented by the quantum at the beginning of the next round. Otherwise, if the queue is empty at the end of the round the credit is zeroed.

The quantum values play an important role in DRR. There are two observations on the quantum configuration in DRR:

- [MG96] points out that the quantum value for a flow can be at least as large as the maximum packet size within the flow so the system will serve at least one packet in the backlogged flow. However, since some traffic, such as those with an exponential distribution have large variance of packet size, it is difficult to decide the largest packet size in advance. In the case that the quantum value is less than the largest packet size, a flow is still not allowed to run a deficit, but can transmit large packets by accumulating credit.
- Because the scheduler will keep the states of multiple flows, the relationship of the quantum values among the flows is:

$$Q_i^m = \frac{r_i^m}{BNBW} \cdot \sum_{n \in B} Q_i^n,$$

$$\frac{Q_i^m}{Q_i^n} = \frac{r_i^m}{r_i^n}, \quad n \in B,$$

where  $Q_i^m$  and  $r_i^m$  are the quantum value and the service rate of flow  $m$  at node  $i$  respectively.  $B$  is the set of active flows, the  $BNBW$  is the bottleneck bandwidth of the outgoing link.

As a result, one flow with a large arrival rate and small packet size can also have a large quantum size compared to the maximum packet size. This flow will not be heavily backlogged because of the large quantum.

### 3.1.2 DRR++

The motivation and the difference between DRR+ and DRR++ have been discussed in Chapter 2. Tables 3.1 and 3.2 describe DRR++ in detail. This

version has some modifications from that in [MW00] in order to simplify its implementation and analysis. If we define  $n$  as the number of BE (Best Effort) flows, the DRR++ algorithm in [MW00] increases the quantum value,  $n \cdot Q_{LC}$ , of the LC (Latency Critical) flow before serving it and **all** the other BE flows. In the modified DRR++, the quantum value,  $Q_{LC}$ , is increased before serving it and **one** of the other BE flows.

For example, assume that all the flows are backlogged. Table 3.5 provides a comparison of the simplified execution sequence of the DRR++ in [MW00] and the modified DRR++. The effects of the modification include:

1. The service intervals for LC packets under the modified DRR++ are much more evenly distributed than those under the DRR++ in [MW00]. Consider a boundary condition such that there are more than  $n \cdot Q_{LC}$  packets waiting in the LC queue. Under the DRR++ in [MW00] at most  $n \cdot Q_{LC}$  packets may be served before the system serves the first BE flow, so the LC flow does not have enough credit to forward packets in the following service interval until all the BE flows are served when the deficit value is updated again (*step 1*). In the modified DRR++, the system serves  $n \cdot Q_{LC}$  LC packets in the  $n$  rounds.
2. It is much easier to analyze the modified DRR++ than the DRR++ in [MW00]. In the following section on the analysis of performance, we have a very simple definition of a *round* and the state transmission diagram is straightforward. It is much more complex to analyze the DRR++ in [MW00] because it is difficult to decide how many LC packets are served during different periods when serving the BE flows.

In the following discussion, we use the term DRR++ to refer to the modified DRR++ algorithm instead of the DRR++ in [MW00].

**Initialization:**

*ActiveList* = NULL  
*TxQueue* = NULL  
*PrioTxQueue* = NULL

**Enqueuing module:** on arrival of packet *p*

*i* = *ExtractFlow*( *p* )  
**If** ( *ExistsInActiveList*(*i*) == **FALSE**) **Then**  
   # Create scheduler queue for flow *i*  
   *AppendToActiveList*( *i* )  
    $DC_i = 0$   
**If** ( *IsLatencyCritical*( *i* ) == **TRUE**) **Then**  
   *AppendToLCList*( *p* )  
**Else**  
   *AppendToBestEffortList*( *i*, *p* )

**Dequeuing module:** manage deficit values and move LC and BE packets to transmit queue

**While** ( **TRUE** )  
   **If Not** *Empty*( *LCList* ) **Then**  
      $q = \text{Dequeue}(\text{Head}(\text{LCList}))$   
      $i = \text{ExtractFlow}(\text{Head}(q))$   
   **Else**  
      $q = \text{Dequeue}(\text{Head}(\text{BestEffortList}))$   
      $i = \text{ExtractFlow}(\text{Head}(q))$   
  
   # Add a quantum  
    $DC_i = \text{quantum}(i) + DC_i$   
  
   # Process a latency-critical queue  
   **If** ( *IsLatencyCritical*(*i*) == **TRUE** ) **Then**  
     **While** ( ( $DC_i > 0$ ) **And Not** *Empty*( *q* ) )  
        $\text{PacketSize} = \text{Size}(\text{Head}(q))$   
       **If** (  $\text{PacketSize} \leq DC_i$  ) **Then**  
          $p = \text{Dequeue}(q)$   
         *AppendToPrioTxQueue*(*p*)  
          $DC_i = DC_i - \text{PacketSize}$   
       **Else**  
         break  
     **If** ( *Empty*( *q* ) ) **Then**  
       # No Backlog - reset quantum  
        $DC_i = 0$

Table 3.1: Modified DRR++ Algorithm.

```

# Process a best-effort queue
If ( IsLatencyCritical(i) == FALSE ) Then
    While ((DCi > 0) And Not Empty( q ))
        PacketSize = Size( Head( q ))
        If (PacketSize ≤ DCi ) Then
            p = Dequeue( q )
            AppendToTxQueue(p)
            DCi = DCi - PacketSize
        Else
            break
    If ( Empty( q )) Then
        DCi = 0
    AppendToBestEffortList( i )

```

**Transmit module:** put the next packet into service

```

While ( TRUE )
    If Not Empty( PrioTxQueue ) Then
        p = Dequeue( Head( PrioTxQueue ))
    Else
        p = Dequeue( Head( TxQueue ))
    Send( p )

```

Table 3.2: Modified DRR++ Algorithm (Continued).

**Initialization:**

```
ActiveList = NULL
TxQueue = NULL
PrioTxQueue = NULL
```

**Enqueuing module:** on arrival of packet *p*

```
i = ExtractFlow( p )
If ( ExistsInActiveList(i) == FALSE) Then
    # Create scheduler queue for flow i
    AppendToActiveList( i )
     $DC_i = 0$ 
If no free buffers left Then
    FreeBuffer( )
If ( IsLatencyCritical( i ) == TRUE) Then
    # Move directly to priority Tx queue
    AppendToPrioTxQueue( p )
Else
    # Append to scheduler queue for flow
    AppendToBestEffortList( i, p )
```

**Dequeuing module:** manage deficit values and move best-effort packets to transmit queue

```
While ( TRUE )
    If Not Empty( ActiveList ) Then
        q = Dequeue( Head( ActiveList ))
        i = ExtractFlow( Head( q ))

        # Add a quantum
         $DC_i = \text{quantum}( i ) + DC_i$ 

        # Process a latency-critical queue
        If ( IsLatencyCritical(i) == TRUE ) Then
            # Is there a backlog?
            If ( NotInQueue( i, PrioTxQueue ) ) Then
                # No backlog - reset quantum
                 $DC_i = \text{quantum}( i )$ 
                AppendToActiveList( i )
```

Table 3.3: DRR++ Algorithm in [MW00].

```

# Process a best-effort queue
If ( IsLatencyCritical(i) == FALSE ) Then
    While ((DCi > 0) And Not Empty( q ))
        PacketSize = Size( Head( q ))
        If (PacketSize ≤ DCi ) Then
            p = Dequeue( q )
            AppendToTxQueue(p)
            DCi = DCi - PacketSize
        Else
            break
    If ( Empty( q )) Then
        DCi = 0
    AppendTo.ActiveList( i )

Transmit module: put the next packet into service
While ( TRUE )
    If Not Empty( PrioTxQueue ) Then
        p = Head( PrioTxQueue )
        i = ExtractFlow( p )
        If DCi > Size(p) Then
            p = Dequeue( Head( PrioTxQueue ))
            DCi = DCi - Size( p )
        Else
            p = Dequeue( Head( TxQueue ))
        Send( p )

```

Table 3.4: DRR++ Algorithm in [MW00](Continued).

| the DRR++ in [MW00]                       | modified DRR++                          |
|---|---|
| 1. $DC_{LC} = DC_{LC} + n \cdot Q_{LC}$ : | 1. <i>for</i> ( $i = 0; i < n; i++$ )   |
| 2. <i>for</i> ( $i = 0; i < n; i++$ )     | 2. $DC_{LC} = DC_{LC} + Q_{LC}$ ;       |
| 3. Serve the LC flow:                     | 3. Serve the LC flow;                   |
| 4. $DC_{BE}^i = DC_{BE}^i + Q_{BE}^i$ :   | 4. $DC_{BE}^i = DC_{BE}^i + Q_{BE}^i$ ; |
| 5. Serve the $BE_i$ flow:                 | 5. Serve the $BE_i$ flow;               |

Table 3.5: Comparison of the execution sequence of the DRR++ in [MW00] and the modified DRR++.

## 3.2 Definitions

In this section, we give some commonly used definitions in scheduling algorithms and specific definitions used in our work.

In order to provide advanced Quality of Service (QoS) architectures in high speed networks, per-flow queueing is commonly used in Integrated Service (IntServ) architectures and per-class queueing is used in Differentiated Service (DiffServ) architectures [RV99]. These two have different service granularities and classification methods. In our work, we use per-flow based network scheduling which keeps state information on each flow and applies a scheduling algorithm to decide the service order.

In TCP/IP networks, a *flow* is defined as a sequence of packets matching some criterion, which can be the same source/destination IP address and port number, the same route or the same class of service requirement. We use the following definition:

**Definition 1.** A *flow* is a sequence of packets with the same source/destination IP address and port number, both of which can be obtained from the TCP/IP packet header.

In order to analyze the performance of flows in different network scheduling algorithms, either the *busy period* or *backlogged period* are often used because the algorithm may take effect during a busy period but not during a non-busy period. The performance of the algorithm during the busy period is also much more realistic.

**Definition 2.** A *backlog period* for session  $i$  is any period of time during which session  $i$  is continually backlogged in the system, i.e., packets belonging to session  $i$  are continually queued in the system.



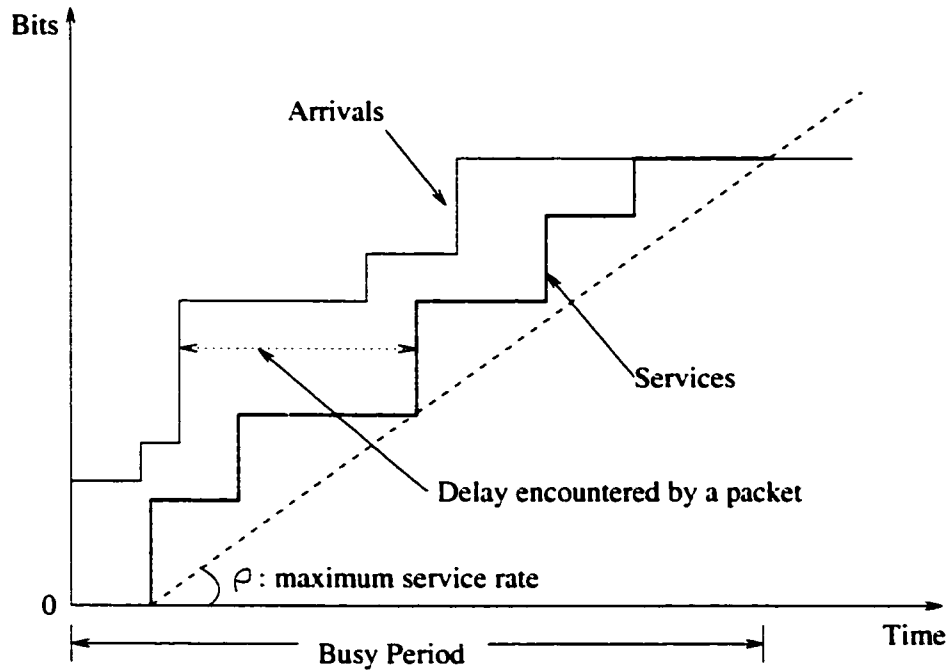


Figure 3.1: Example of arrival behavior and service behavior.

**Definition 3.** A *busy period* for session  $i$  is a any period of time during which session  $i$  always has packets to send.

A busy period can be longer than a related backlogged period. That is, it is possible that there are several backlogged periods during a busy period and also that a busy period may contain zero backlogged intervals. In addition, the start of a busy period always marks the start of a backlogged period, but the converse is not always true.

Figure 3.1 depicts an example of arrival and service behavior. The figure is similar to the graphs in [DA98] and [HZ95]. The upper curve shows the number of arriving bits and the lower curve shows the number of bits served. These two curves are not smooth because we consider packet-switched networks rather than the ideal fluid model. From the figure, the busy period ends when the service curve finishes all the arrival bits.

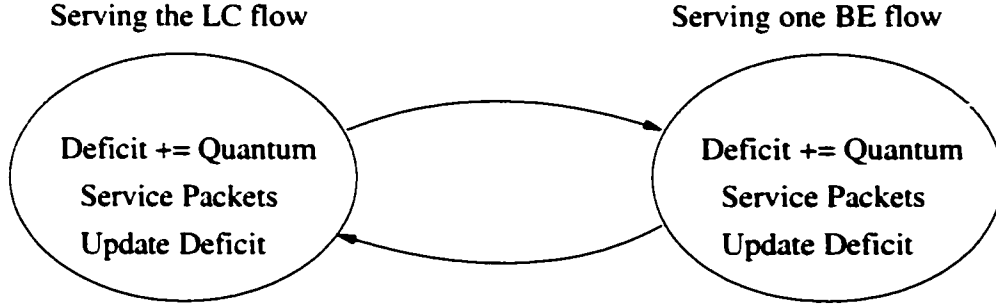


Figure 3.2: A state transmission diagram in DRR++.

**Definition 4.** *One round in DRR++ is any period of time during which the LC session and another BE session are serviced.*

Figure 3.2 shows that the DRR++ system alternates serving the LC session and one BE session. The definition of a round in DRR++ is different from that in DRR. One round in DRR is any period of time during which all the sessions are served once.

The following notation is used:

$\nu_{LC}$ : the average number of LC packets served in one interval.

$B.NBW$  = bottleneck bandwidth.

$D_{LC}$ : the mean delay for the LC flow.

$n_{BE}, n_{LC}$ : average packet length for the best-effort and latency-critical flows respectively.

$Q_{BE}, Q_{LC}$ : quantum for the best-effort and latency-critical flows respectively.

We assume that all the BE flows have the same quantum values.

$Q_i$ : total quantum to serve one BE flow / LC flow pair (one round) at node  $i$ ,

$Q_i = Q_{BE} + Q_{LC}$ .

$r_{LC}$ : the service rate of the LC flow:

$$r_{LC} = \frac{Q_{LC}}{Q_i} B.NBW$$

### 3.3 Mean Delay

In this section, we assume that the queue is always backlogged for both BE and LC traffic, a situation which will occur under conditions of high traffic intensity. We then analyze the mean delay of packets which arrive and are served during a backlogged period.

**Theorem 1** *In a high intensity situation, if  $D_{LC}$  is the mean delay of all the packets of the LC flow in DRR++, then*

$$D_{LC} \approx \left[ \frac{0.5Q_i}{BNBW - (Q_i \cdot \lambda_{LC})/\nu_{LC}} \right] \quad (3.1)$$

where  $\lambda_{LC}$  is the arrival rate of LC traffic.

**Proof.** The DRR++ system alternates between serving the latency-critical flow and one of the best-effort flows. The first packet arrival for the LC flow that begins a backlogged period is defined as packet number zero. Now consider the  $k^{th}$  packet arrival for the LC flow. Assuming that the server is sufficiently backlogged so that this packet is not served until the next LC service interval, the expected delay consists of two terms:

- $W_a$ : the residual service time for the current service interval, consisting of the LC service and one BE service.
- $W_b$ : the time before the beginning of the service interval in which the packet is served.

$W_a$  is simply the mean residual service time of the interval during which the packet arrives:

$$W_a = \frac{Q_i/2}{BNBW}$$

Since  $\nu_{LC}$  is the average number of LC packets served in one interval, then:

$$\nu_{LC} = \lfloor q_{LC}/n_{LC} \rfloor$$

From Little's law [RJ91]<sup>1</sup>, we know the mean length of the latency-critical queue is:

$$N_{LC} = \lambda_{LC} D_{LC}$$

so that the number of rounds before we serve the  $k^{th}$  arrival,  $\overline{s_{LC}}$ , will be

$$\overline{s_{LC}} = \lceil \frac{\lambda_{LC} D_{LC}}{\nu_{LC}} \rceil - 1$$

The total amount of time required for this is:

$$W_b = \frac{Q_i \cdot \overline{s_{LC}}}{B \cdot N \cdot BW}$$

---

<sup>1</sup>Little's law is one of the most commonly used theorems in queueing theory. If  $N$  is the mean number in the system,  $\lambda$  is the arrival rate and  $T$  is the mean reponse time, then Little's law can be presented as:

$$N = \lambda \cdot T$$

In this thesis, we apply Little's law to the output queue. If  $N_q$  is the mean number in the queue,  $\lambda$  is the arrival rate and  $W$  is the mean waiting time, then we have:

$$N_q = \lambda \cdot W$$

Thus, assuming that the arrival is served in a future round, the mean delay is:

$$\begin{aligned}
D_{LC} &= W_a + W_b \\
&= \frac{0.5Q_i + Q_i \cdot \overline{s_{LC}}}{B.NBW} \\
&= \frac{0.5Q_i + Q_i \cdot (\lceil \frac{\lambda_{LC} D_{LC}}{\nu_{LC}} \rceil - 1)}{B.NBW}
\end{aligned}$$

then we have:

$$\begin{aligned}
D_{LC} &\approx \frac{0.5Q_i + Q_i \cdot \frac{\lambda_{LC} D_{LC}}{\nu_{LC}}}{B.NBW} \\
D_{LC} \cdot B.NBW &\approx 0.5Q_i + Q_i \cdot \frac{\lambda_{LC} D_{LC}}{\nu_{LC}} \\
D_{LC} \cdot \left( B.NBW - \frac{Q_i \cdot \lambda_{LC}}{\nu_{LC}} \right) &\approx 0.5Q_i \\
D_{LC} &\approx \left[ \frac{0.5Q_i}{B.NBW - (Q_i \cdot \lambda_{LC})/\nu_{LC}} \right] \quad \square
\end{aligned}$$

This assumption should be a good approximation to the conditions at high LC arrival intensity. We will compare this analysis with our measurement results in Chapter 5.

### 3.4 Delay Bound of the LC flow

The state diagram of the DRR++ algorithm is displayed in Figure 3.2. In order to analyze the delay bound of the LC flow, we add the following definitions:

$t_0$ : the beginning time of a backlogged period, i.e. the beginning of the first round.

$t_k$ : the time of the end of the  $k$ th round,

$D_{LC}^k$ : the deficit value of the LC flow at the end of the  $k$ th round.

$W_{LC}(t_0, t)$ : the service offered to the LC flow during the interval  $(t_0, t)$ .

**Lemma 1.** *In DRR++, if the LC flow is continuously backlogged during the interval  $(t_0, t_k)$ , then at the end of the  $k$ th round,*

$$W_{LC}(t_0, t_k) \geq kQ_{LC} - D_{LC}^k$$

**Proof.** Though DRR and DRR++ have a different definition of *round*, there is no difference between the service assigned to a normal flow in DRR, or the LC flow in DRR++. Thus DRR++ inherits this property from DRR [MG96, Lemma2].  $\square$

**Lemma 2.** *In DRR++, at any point  $t$  such that  $(t_0, t)$  is a backlogged period,*

$$W_{LC}(t_0, t_k) \geq \max \left( 0, r_{LC} \left( t - t_0 - \frac{Q_{LC} + 3Q_{BE}}{B.NBW} \right) \right)$$

**Proof.** From Lemma 1 and the analysis of DRR based on the LR server [DA98, Lemma 11], we have:

$$W_{LC}(t_0, t_k) \geq \max \left( 0, r_{LC} \left( t - t_0 - \frac{3Q_i - 2Q_{LC}}{B.NBW} \right) \right)$$

so we can get Lemma 2 from the definition of  $Q_i$  in DRR++:  $Q_i = Q_{LC} + Q_{BE}$ .

$\square$

**Lemma 3.** *DRR++ is an LR server with latency  $\Theta_i^{DRR++}$  and*

$$\Theta_i^{DRR++} \leq \frac{Q_{LC} + 3Q_{BE}}{B.NBW} \quad (3.2)$$

**Proof.** The proof is straightforward from Lemma 2 and the analysis of the LR server [DA98].  $\square$

**Theorem 2** *In DRR++, if it is assumed that the arrivals of the LC flow satisfy the  $(\sigma_{LC}, \rho_{LC})$  model, where  $\sigma_{LC}$  and  $\rho_{LC}$  denote the burstiness and the average rate of the LC flow respectively, we can bound delay of any packet*

belonging to the LC flow,  $\hat{D}_{LC}$ , as

$$\hat{D}_{LC} \leq \frac{\sigma_{LC}}{\rho_{LC}} + \frac{Q_{LC} + 3Q_{BE}}{BNBW}.$$

**Proof.** The proof is straightforward from Lemma 2 and the analysis of the LR server [DA98].  $\square$

Theorem 2 shows that DRR++ provides a lower delay bound for the LC flow than DRR+, which has latency,

$$\Theta_i^{DRR+} \leq \frac{3F - 2Q_{LC}}{BNBW} \quad (3.3)$$

where

$$F = Q_{LC} + \sum_{j \in BE} Q_j$$

From (3.2) and (3.3), we have another observation on the improvement of DRR++: the delay bound of an LC flow in DRR++ is independent of the number of active flows,  $N$ , in the system. While in DRR+,  $\Theta_i^{DRR+}$  increases as  $N$  increases.

### 3.5 Summary

In this chapter, we have presented the DRR++ algorithm along with a modified version for straightforward implementation and analysis. We also have given the delay analysis of DRR++ in terms of average delay and delay bound.

# Chapter 4

## Measurement Study of DRR+ and DRR++

The goal of our measurement study is to apply the DRR++ and DRR+ algorithms in a practical network rather than in a simulation environment and find out whether the theoretical analysis matches the practical results. Although measurement is more difficult than simulation, it is more convincing since the results in such a study may be affected by factors which may not be accounted for in a simulation, such as operating system effects. In this chapter, we will discuss a scheduler implementation, measurement environment, traffic generation methodology, and issues concerning accuracy of measurements.

### 4.1 System Implementation

Both DRR+ and DRR++ have two modules – **Enqueueing** and **Dequeuing** ( [MG96], [MW00] ) – which deal with the input and output functions respectively. We implemented an application level scheduler on a Linux platform. That is, as opposed to adding the scheduling algorithms to the Linux kernel, an upper level application was implemented.

In our implementation of Enqueueing, we keep a list of active flows with operations such as *AddFlow*. When a new packet arrives, the system determines the flow of the packet based on the information in the packet header. If it is a new flow, the flow identification is added to the flow list by calling



*AddFlow*. After create a new identifier for the flow or if the flow identifier already exists in the flow list, the packet is added to the back of the queue.

In the Dequeueing implementation, we scan the active flow list and check if there are packets waiting in the input queues. If there are, we take the packets from the head of the queue if the credit is large enough and put them in the appropriate output queues.

Because both modules access the flow list, we designed two threads, each of which performs one function. The flow list is defined as a variable which can be accessed by both threads (See Figure 4.1). Through this method, the enqueue thread function can be executed as long as the flow list is not in use by the dequeue thread function and vice versa. Another implementation method involves two processes which share a block of memory and deal with the input and output separately. The active flow list and other shared variables are then shared by these two processes. However, in the thread implementation, since the data is inherently shared, context switches between the threads do not have to flush all the memory management buffers, so it requires less system overhead than the two process implementation.

## 4.2 Measurement Topology

There are two common topologies that have been used in previous research work (see [AS89], [LZ90], [MG96], [FH00] and [IH00] etc.). One is the single node or single congested link topology (Figure 4.2 (a)), and the other is a multiple node topology, which is also called the “parking lot topology” (Figure 4.2 (b)). In either case, the scheduling algorithm runs on the gateway (GW) nodes, and each source may generate multiple connections as background traffic or measured traffic. The first topology is often used to test a scheduling algorithm on the congested link. The performance of the algorithm in terms of delay and throughput are tested under the condition that multiple connections share the same outgoing link. The parking lot topology is often used to test the fairness of a scheduler because the traffic from different sources may have

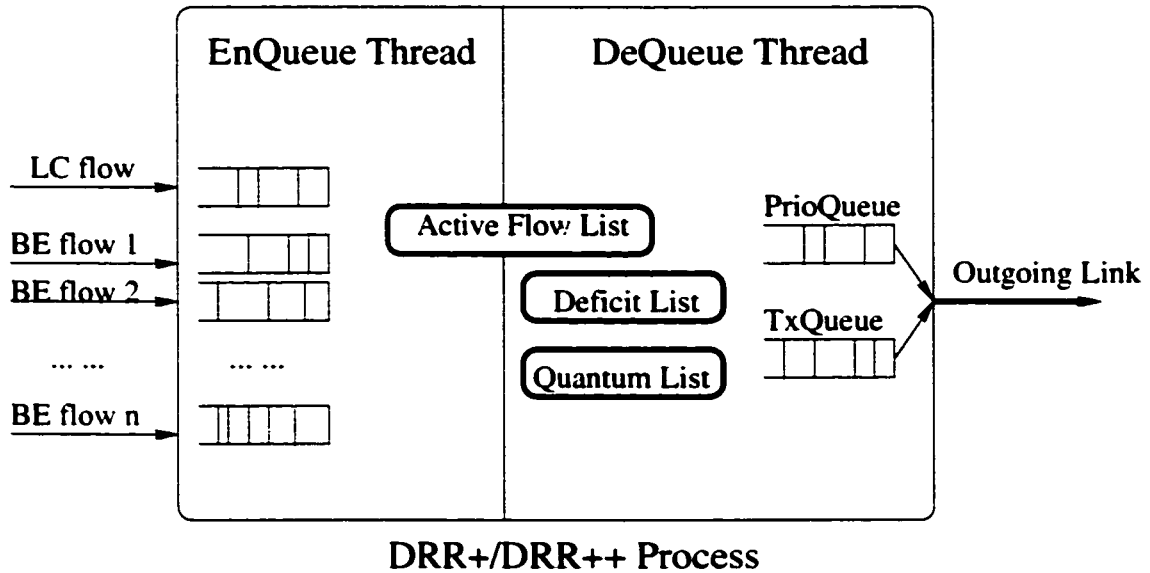
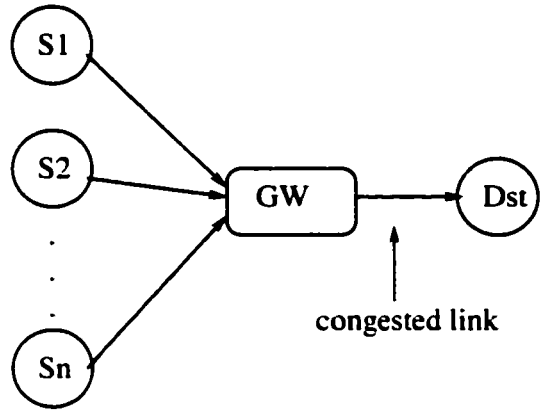


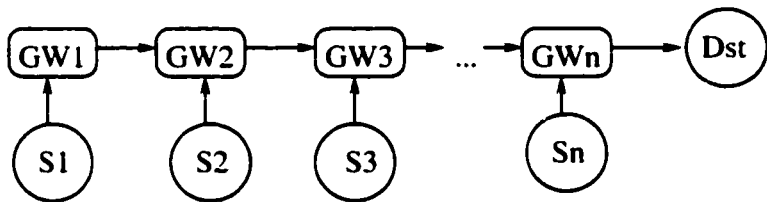
Figure 4.1: DRR+/DRR++ Processes.

different path lengths and these paths share links. The scheduler must decide how to distribute the bandwidth among the connections sharing the same link. For per-flow scheduling, the parking lot topology can be considered as a more complex version of the single congested link topology because of the finer granularity of the flow. The measurements in this study were made on the single congested link topology shown in Figure 4.3. The traffic route is indicated by the arrows from node S1 and S2 via the gateway G1 to node D1. The link between G1 and D1 was the congested link. Nodes S1 and S2 each hosted a DRRClient and generated a total of four flows. The nodes were both 400MHz Celeron CPUs with 64 MB of memory running Red Hat Linux 6.1. Both had multiple Ethernet NICs to generate the multiple flows to the DRRGateway. Node G1 acted as DRRGateway, and node D1 acted as DRRServer. The DRRGateway received packets from each DRRClient and forwarded them to the DRRServer after applying the scheduling algorithm under test.

We used two Cisco 2508 routers to connect the DRRGateway and the DRRServer. In order to keep the packets in the output queues before they are



(a)



(b)

Figure 4.2: Two Common Topologies: Single node and Multiple nodes.

transmitted by the scheduling algorithm in the DRRGateway, the bandwidth of the output link of DRRGateway is always lower than that of the input link. In our topology, the WAN link bandwidth was varied from 2400 bps to 4Mbps, much less than the 10Mbps Ethernet bandwidth between the DRRClient and DRRGateway.

In the topology, there are two NAS (Network Analysis Server) for the accurate measurement, which will be discussed in section 4.5.

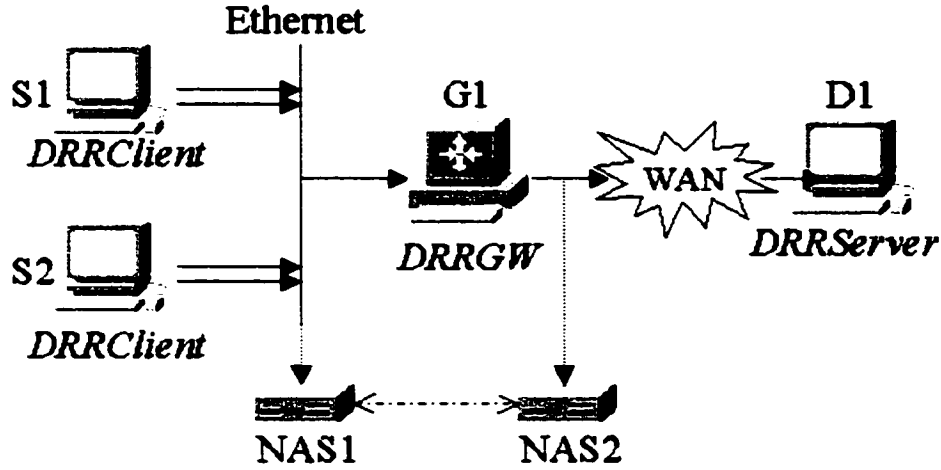


Figure 4.3: Measurement Topology.

### 4.3 Methodology

The scheduling algorithms reside at node G1, on the upstream side of the congested link, and take effect after the packets arrive at node G1 and are waiting to be forwarded to D1. We define  $T_s^p$  and  $T_a^p$  as the time-stamps before the last bit of a packet  $p$  reaches node G1 and after the last bit of it leaves node G1 respectively. The metric we use to evaluate the algorithms is the mean delay for each flow  $j$ :

$$D_j = \overline{(T_a^p - T_s^p)}, p \in j$$

The maximum delay for flow  $j$  is:

$$\hat{D}_j = \max\{(T_a^p - T_s^p), p \in j\}.$$

The parameters used in the study are the packet arrival rate,  $\lambda$ , for the best-effort flow and the quantum value,  $Q_{LC}$ , for the LC flow.

### 4.4 Traffic Generation

Both the protocol type and the burstiness of the traffic need to be resolved. For the former, both UDP and TCP were considered. For the latter, there are two kinds of burstiness that must be considered.

### 4.4.1 Why UDP is more appropriate than TCP

There are two different protocols available to us: TCP and UDP. In our initial tests, we found that it was difficult to obtain accurate timestamps for the calculation of the delay if TCP traffic was generated. First, TCP is much more complex than UDP because TCP uses error correction, flow control, sequencing and retransmission mechanisms. The TCP protocol dynamically adjusts the sending rate based on acknowledgements (ACK) information coming back from the receiver. If the delay between sender and receiver is large, the TCP protocol becomes insensitive to short term network load changes because a large delay increases the response time of the feedback loop. Second, TCP is a *stream protocol* rather than a *packet protocol*. Consider these two issues in an Ethernet environment with an MTU of value 1500 bytes:

- Consider a large packet with length of 15000 bytes, which may be an *I* frame in MPEG traffic (see section 4.4.3). It will be segmented into 10 frames in the DRRClient and each fragment will be sent out as one packet. They are re-assembled by the IP layer in the DRRServer. If one fragment is lost, it is retransmitted according to the TCP protocol, which results in an increased delay for the whole packet. There are several reasons for this increased delay, such as congestion or a transmission error. Because we only care about the performance of the scheduler, it is difficult to obtain correct measurements. In contrast, UDP does not retransmit a lost fragment because the UDP protocol does not use acknowledgements for each packet to confirm reception. In some real time applications, e.g. voice and video, a small degree of loss is much more tolerable than increased delay.
- Assume that two packets,  $p^1$  and  $p^2$ , with a length of 100 bytes and 200 bytes respectively are sent from the DRRClient to the DRRServer. If the interval ( $T_s$ ) between these two packets is small enough, when using TCP, the DRRClient sends them together as one packet to the

DRRServer. Consequently, the DRRServer records only one arrival time ( $T_a$ ) for the combined packet rather than times for two separate packets. Although we can calculate the arrival time for these two packets,  $T_a^1$  and  $T_a^2$ , as

$$T_a^1 = T_a - \frac{200\text{Bytes}}{BNBW}$$

$$T_a^2 = T_a,$$

doing so is not accurate because we do not know how much delay there has been for the first packet in the DRRClient. In our tests, we found that the TCP layer in the DRRClient may bind several packets together before sending them out. When using UDP, this will not happen because UDP sends packets separately.

By relying on UDP, the delay values in our tests reflect results from the scheduling algorithm rather than being a combined value of both the scheduling algorithm and the TCP protocol. Moreover, with the topology selected, we can always get packets belonging to one connection to arrive in order, which is otherwise not guaranteed by UDP.

#### 4.4.2 Smooth Traffic

We use exponentially distributed traffic as the source type for BE traffic under both DRR++/DRR+. The BE traffic is intended mostly to provide a background load for our study of the service received by the LC flow. The BE traffic had a packet length with a mean of 410 bytes, consisting of a 40 byte header plus a data portion with a length uniformly distributed in the range of [40, 700] bytes.

#### 4.4.3 Latency-Critical Traffic and Burstiness

The LC flow is meant to represent a bursty, real-time flow. In [KC94], two categories of *burstiness* are distinguished – one is the burstiness of inter-arrival times, and the other is the burstiness of packet length. For the former, in

this study we use an exponential distribution because it exhibits significant short term variance. For the latter, we use MPEG traffic because it exhibits significant variation in packet length.

MPEG video is encoded into three kinds of frames, designated  $I$  (intra-coded),  $P$  (predicted) and  $B$  (bidirectional). We use the MPEG traffic model from [SK98] with 10 frames per group of pictures (GOP). The sequence of frame types used to transmit one GOP is  $IPBBPBBPBB$ , and pictures are sent at a rate of 3 GOPs per second, so that there will be 3  $I$ -frames/sec, 9  $P$ -frames/sec and 18  $B$ -frames/sec. The different frame types also have different mean lengths and distributions.  $I$ -frames have a mean length of 16000 bytes, uniformly distributed between 15000 and 17000 bytes.  $P$ -frames have a truncated exponential distribution with a mean of 2100 bytes, and minimum and maximum values of 512 and 17000 bytes.  $B$ -frames also have a truncated exponential distribution, with a mean of only 550 bytes, and minimum and maximum values of 256 and 17000 bytes. The overall mean rate is 615Kbps, to which  $I$ -frames contribute about 50% of the bytes with only 10% of the total frames. Thus, while the frame inter-arrival times are predictable (at least at the source), the observed distribution of frame lengths for the aggregate traffic is quite bursty.

#### 4.4.4 Traffic Models

Different traffic models can be used to characterize traffic [AA97][MS96]. Some popular models for different kinds of traffics are:

- *Poisson*: Poisson traffic is used for normal data traffic. Given an average arrival rate  $\lambda$ , the average inter-arrival time between two packets is  $1/\lambda$ .
- *On-Off*: The On-Off model is mostly commonly used for single voice source. In this model, packets are generated during the On state (talk spurts) and there is no packet during the Off state (silence). The parameters in the On-Off model are: the mean length of time in the On

state, the mean length of time in the Off state and packet arrival rate in the On state. The first two parameters are specified as exponential distributions with mean  $1/\lambda$  and  $1/\beta$  respectively. The packet arrival rate in the On state is a fixed value, corresponding to the voice sampling rate.

- *Markov-Modulated Poisson Process (MMPP)*: MMPP is used to represent a variety of traffic types, such as voice, video or the mixture of several traffics [AA97] [MS96]. It defines  $n$  states,  $s_i, 1 \leq i \leq n$ , and in any state the arrivals occur according to a Poisson process with rate  $\lambda_i$ .
- *MPEG*: MPEG can be modeled as discussed in Section 4.4.3. Many studies have been done on the distribution of the  $I$ ,  $P$  and  $B$  frame sizes. For example, [SK98] uses a uniform distribution for  $I$  frames and exponential distributions for both  $P$  and  $B$  frames. [MH95] uses lognormal distributions for all the three frame types but with different parameters.

Another kind of traffic model is traffic shaping (or regulating), which controls the output of packets in a specified rate. The main reason for using traffic shaping is to ensure that traffic conforms to the contracts established for it, and to regulate the flow of traffic in order to avoid congestion that can occur when the transmitted traffic exceeds the access speed of remote nodes. Another reason is that it is easier to analyze the scheduling algorithm if the input traffic is shaped before entering the link. Note that traffic shaping is different from traffic policing because a policer typically drops traffic while a shaper typically delays excess traffic using a buffer to hold packets.

[HZ95] summarizes several traffic shaping models :  $(X_{min}, X_{ave}, I, S_{max})$ ,  $(\sigma, \rho)$  and  $(r, T)$ . “A traffic stream satisfies the  $(X_{min}, X_{ave}, I, S_{max})$  model if the inter-arrival time between any two packets in the stream is more than  $X_{min}$ , the average packet inter-arrival time during any interval of length  $I$  is more than  $X_{ave}$ , and the maximum packet size is less than  $S_{max}$ ”. “A traffic



stream satisfies the  $(\sigma, \rho)$  model if during any interval of length  $u$ , the number of bits in that interval is less than  $\sigma + \rho u$ . In the  $(\sigma, \rho)$  model,  $\sigma$  and  $\rho$  can be viewed as the maximum burst size and the long term bounding rate of the source respectively". "Similarly, a traffic satisfies  $(r, T)$  model if no more than  $r \cdot T$  bits are transmitted during any interval of length  $T$ " [HZ95].

We use an  $(r, T)$  model in our work because its definition is similar to the definition of the contract used in DRR+ and DRR++. The contract,  $C_j$ , is defined as the number of bytes that can be sent on connection  $j$  during an interval of length  $\Delta t$ . Formally, if  $N_j$  is defined as the total number of bytes that arrive at the DRRGateway on connection  $j$  during the interval  $(t, t + \Delta t)$ , then the flow is considered to obey its contract only if  $N_j \leq C_j, \forall t, \Delta t > 0$ . In the tests of DRR+, the contract,  $C_{lc}$ , was set to 135% of the average rate.

## 4.5 Measuring Delays Accurately

To quote RFC-2330 [VG98] "time lies at the heart of many Internet metrics." This is certainly an issue in this study, especially because we need to use the times at which events occur at various points in a spatially-distributed system.

A few definitions are useful when discussing the use of clock values in measurements [VG98], [VP98]. Consider two clocks generated from two different machines. The *offset* at a particular moment is the difference between the values of the two clocks. The *skew* at a particular moment is the first derivative of offset: this is the rate at which one clock value is approaching or moving away from the other. The *drift* at a particular moment is the second derivative of offset – the acceleration of one clock value toward or away from the other. The two clocks are *synchronized* if their relative offset is zero.

Calculating packet delays requires using the clock values from more than one machine, so synchronization between multiple clocks is as important as their accuracy. In [VG98], some drawbacks of the Network Time Protocol (NTP) as a long term solution are noted. One of its assumptions in NTP is that timestamps are monotone increasing, which is reflected as constant

skew and zero drift. But the results in [VP98] show that clock offsets can jump randomly at times. While conducting this study, we found nonzero drift between the clocks in our workstations, and we also found offset jumps at random times.

For example, Figure 4.4 and Figure 4.5 show the delay of one LC flow from DRRClient1 and one BE flow from DRRClient2 over five runs of approximately 25 minutes each: different runs show the same trend and have nearly constant drift over the course of the run. Figure 4.5 shows that the delay decreases when the arrival rate is between 10 and 20 packets/sec and increases when the arrival rate is larger than 20 packets/sec. This result is difficult to explain if we do not consider the clock synchronization. These two figures also depict the opposite offset behavior between the LC flow and the BE flow: given a BE traffic rate, the five seeds generate an *increased* offset value of about 130 millisc<sup>1</sup> for the LC flow, but a *decreased* offset value of about 160 millisc for the BE flow. This result shows that the two DRRClients have a different rate of change of offset compared with that of the DRRServer.

In another experimental run, we looked at the evolution of clock skew. At the DRRServer, we sampled the timestamps of arriving packets every 200 seconds. We then calculated the relative offset based on the first sample. If we define  $t_n$  as the timestamp of the  $n$ th sample, the relative offset of this sample is calculated as  $(t_n - t_0 - 200 * n)$ , which is the offset between the current clock value and the predicted value based on the clock time at the arrival of the first sample. In an ideal situation, the relative offset will not change for a clock with zero drift, which satisfies the adjustment assumption of NTP. But Figure 4.6 shows increasing clock offset, as well as offset jumps at some random times. Thus, the use of NTP is precluded as an accurate method for clock synchronization in these experiments.

As a result of these observations we decided to obtain delay measurements through the use of two accurately synchronized clocks. These were available as

---

<sup>1</sup>millisecond = 0.001 (1e-3) sec

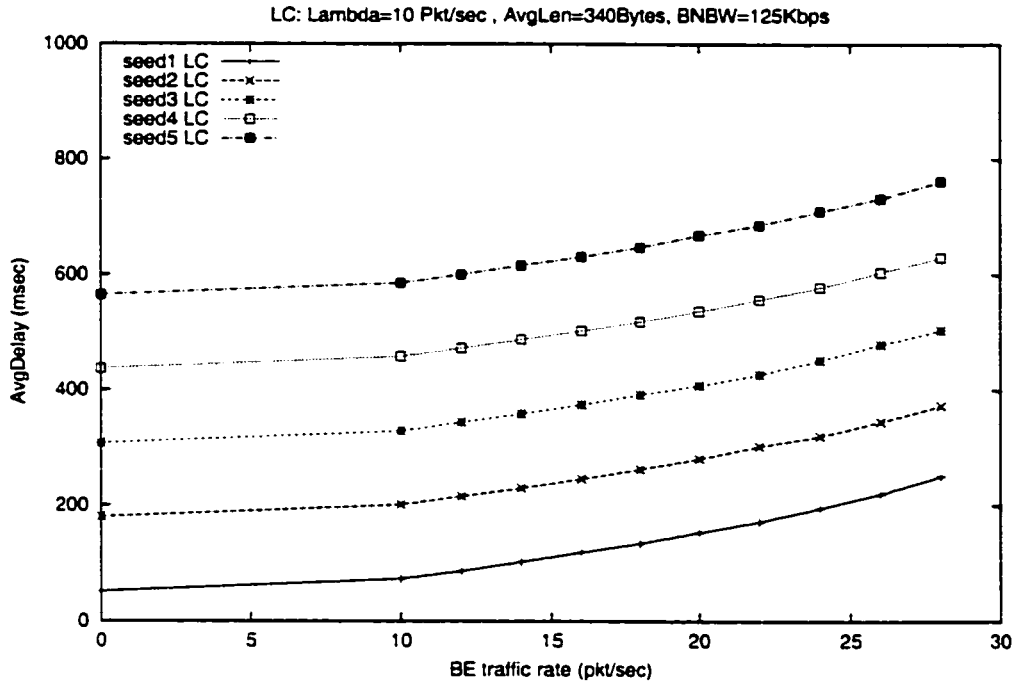


Figure 4.4: LC Seeds Offset Measurement.

two DominoPLUS.NAS (Network Analysis Server) from Wandel & Goltermann Technologies, Inc. A TTL pulse from one analyzer is used to trigger the clock of the other (See Figure 4.3). All of the following results are based on the timestamps captured from the two network analyzers.

## 4.6 Summary

This chapter begins by describing a thread-based application level implementation of the DRR+ and DRR++ algorithms. Then we discuss several issues about the measurement environment: *topology*, *methodology*, *traffic generation* of both smooth and bursty sources. Finally, we present the results of an investigation of the problem of measuring delays, and give a practical solution to this problem.

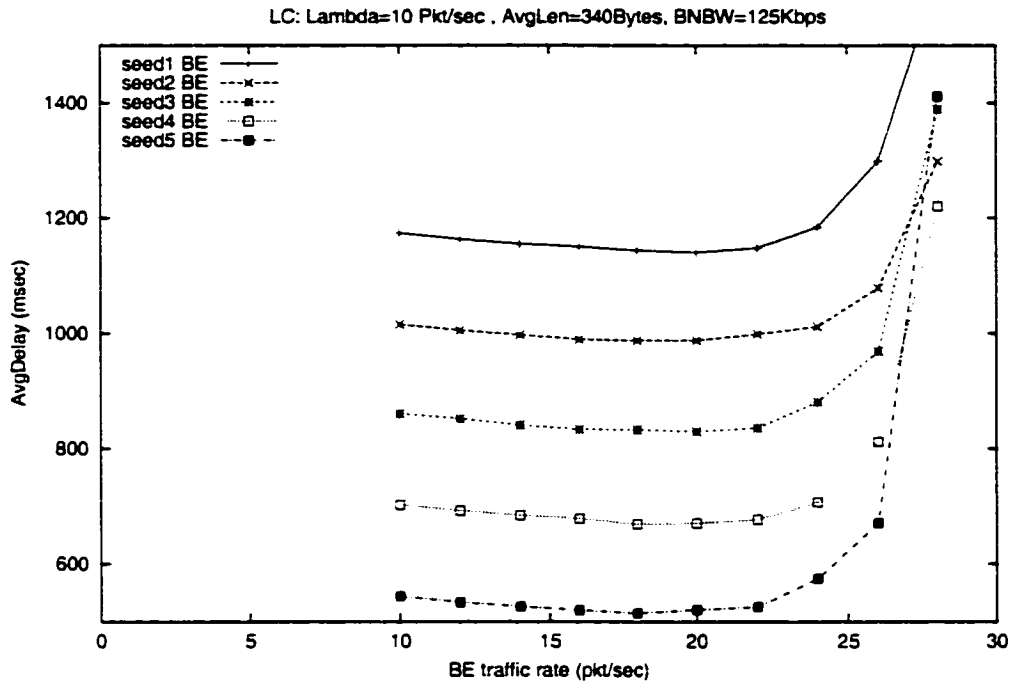


Figure 4.5: BE Seeds Offset Measurement.

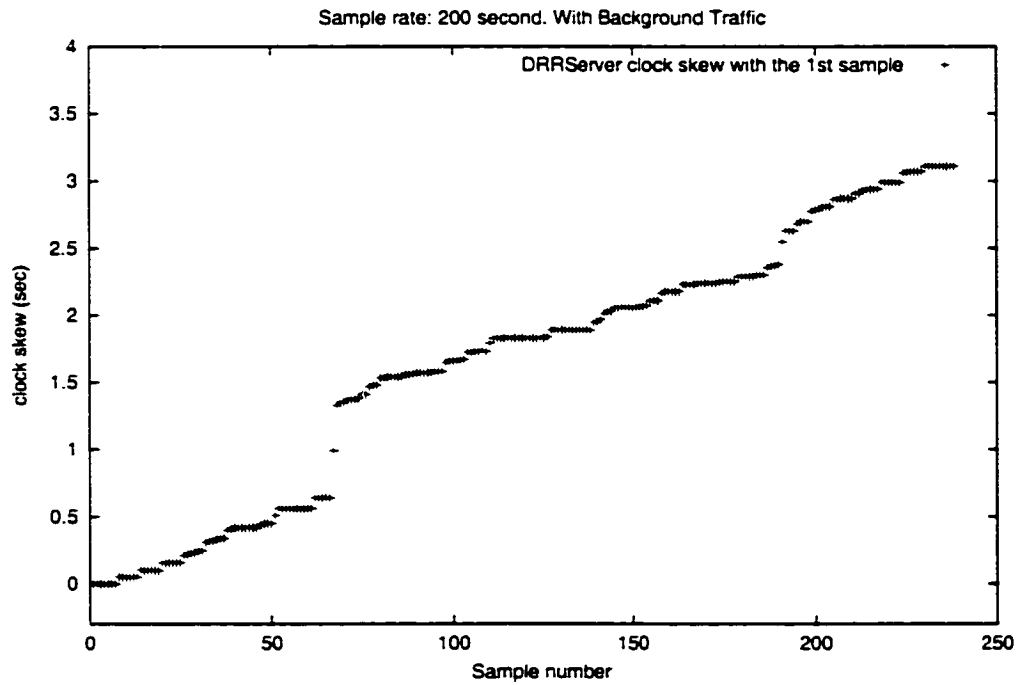


Figure 4.6: Self Clock Drift Measurement.

# Chapter 5

## Measurement Results

In this chapter, we report measurement results for both exponential and MPEG LC sources. Under these two different kinds of bursty traffic sources, we investigated the performance of DRR+ and DRR++ in terms of mean delay, isolation and fairness:

- We tested the mean delay of the LC flow under various environments and took it as the main metric to compare DRR+ and DRR++.
- In order to test the isolation feature, the behaviors of the background BE flows were changed to measure the effect on the performance of LC flow.
- For fairness, we did not give a comprehensive investigation but tested the mean delays for three BE flows with the same parameters.

For both types of bursty sources, we looked at the effects of varying the intensity of the background best-effort flows, and the effects of changing the quantum allocated to the LC traffic. In all the tests, we generated one LC flow and three BE flows and calculated the mean delay of each flow. We used one LC flow rather than multiple LC flows in order to eliminate the intra-class effect between several LC flows and to make sure that the performance of the LC flow is only affected by the BE flows. The number of BE flows was limited by the available memory in the Domino PLUS.

| $\lambda_{BE}$ | Traffic rate (bytes/sec) | Burstiness (bytes) |
|----------------|--------------------------|--------------------|
| 5              | 1850                     | 8983               |
| 7              | 2590                     | 7894               |
| 10             | 3700                     | 12141              |
| 11             | 4070                     | 10270              |
| 13             | 4810                     | 5810               |
| 15             | 5550                     | 4179               |
| 16             | 5920                     | 4146               |
| 18             | 6660                     | 3717               |
| 20             | 7400                     | 3576               |

Table 5.1: Traffic rate and burstiness for exponential traffic.

## 5.1 Exponential LC Traffic

During the tests, the capacity of the bottleneck link was 125Kbps. Each measurement test took 100 seconds, and five delay results,  $x_i, i = \{1, 2, 3, 4, 5\}$ , were gathered using five different groups of seeds for random generation in order to compute the 95% confidence interval of the mean delay as [RJ91]:

$$\bar{x} = \frac{1}{5} \sum_{i=1}^5 x_i$$

$$\sigma = \sqrt{\frac{1}{5-1} \sum_{i=1}^5 (x_i - \bar{x})^2}$$

$$95\%CI = 1.96 \cdot \frac{\sigma}{\sqrt{5}}$$

The behavior of exponential LC traffic was investigated because this represents a type of traffic whose interarrival times can be very bursty. Table 5.1 gives the arrival rate and the burstiness of the exponential traffic. The burstiness,  $\sigma$ , is calculated as:

$$\sigma \geq A(\tau, t) - \rho(t - \tau),$$

where  $A(\tau, t)$  is the total number of arrivals during  $(\tau, t)$  and  $\rho$  is the service rate allocated to the flow. This formula can be derived by rearranging inequality 3.1 in [DA95].

### 5.1.1 Effect of background traffic intensity

In the first series of measurements we observed the mean delay of the exponential LC flow as a function of the intensity of best effort traffic. Because of the isolation of DRR++, the behavior of the LC flow should not be affected by the behavior of the BE flows. The mean delay of the LC flow should be bounded when the intensity of the BE traffic increases.

In the tests, the value of  $\lambda_{BE}$  was varied while keeping  $\lambda_{LC}$  at 15 packets/sec and  $Q_{LC}$  at 2000 bytes. The three BE flows had the same arrival rate, ranging from 5 to 20 packets/sec. The quantum of the best effort flows,  $Q_{BE}$ , was increased in proportion to the changes in  $\lambda_{BE}$ . Specifically, we fixed the ratio of quantum to arrival rate to be the same as for the latency critical traffic:

$$\frac{Q_{BE}}{\lambda_{BE}} = \frac{Q_{LC}}{\lambda_{LC}}$$

The algorithms we are studying, DRR+ and DRR++, can be modeled as an  $M/G/1$  queue with vacations if we want to examine only the performance of the LC flow. In this model, when each busy period ends, the system spends some interval of "vacation" time. If a new packet arrives during the vacation period, it cannot be served until the end of the vacation period. If there is no new packet awaiting service when the vacation ends, another vacation starts again.

If we define  $V_{LC}$  as the average vacation time of the LC flow, then  $V_{LC}$  is the average service time of the BE flows before the next LC service. In DRR++,  $V_{LC} \leq Q_{BE}/B.NBW$  because the LC flow is served only after one BE traffic has been forwarded. In DRR+,  $V_{LC} \leq \sum_j Q_{BE}^j/B.NBW$  at any point in time after the LC flow breaks the contract. In DRR++, consider the points in Figure 5.1 at which  $\lambda_{BE} = 0$  and  $\lambda_{BE} = 20$  packets/sec. When

$\lambda_{BE} = 0$ ,  $V_{LC} = 0$ , and at  $\lambda_{BE} = 20$ :

$$\begin{aligned} \max(V_{LC}) &= \frac{Q_{BE}}{BNBW} \\ &= \frac{(2666 * 8)bits}{250Kbps} \\ &= 85.3 \text{ millisecond.} \end{aligned}$$

In our experiment,  $D_{LC}$  increases by  $106 - 15.5 = 90.5 \text{ millisecond}$ . This is a little bit larger than the  $\max(V_{LC})$  because the queuing delay of the DRRGateway is included in our measurements.

Under DRR++,  $D_{LC}$  increases up to an asymptote, while  $D_{BE}$  increases dramatically when the link nears saturation at a utilization of 98.4% for  $\lambda_{BE} = 20 \text{ packets/sec}$  (See Figure 5.1). The asymptote occurs where the vacation time of the LC flow is bounded by the quantum value of one BE flow. At high intensity the quantum of the BE flow can always be consumed due to the high arrival rate. As a result, the delay of the LC flow increases up to this value as the arrival rate of the BE flows increases.

When using DRR+,  $D_{LC}$  increases quickly in the same manner as  $D_{BE}$  (See Figure 5.2). This is because once the LC flow breaks its contract, it is put back into the BE flow list and gets the same service as the other BE flows. In addition, because the behavior of an exponential distribution depends on the value of seed, we observe that under different seeds the LC flow breaks its contract at different timepoints. The result is that the 95% CI of  $D_{LC}$  is much larger under DRR+ than when using DRR++ (Table 5.2).

The bounded delay value that the DRR++ offers to the LC flow is important because the behavior of other BE flows does not affect the performance of the LC flow. This isolation feature protects the LC flow if there are some badly behaved BE flows.

Figure 5.1 also shows the mean delay of the LC flow based on the analysis in Chapter 3. This analysis approximates the high utilization condition. When  $\lambda_{BE}$  is small, the BE flow usually has a much shorter busy period than when  $\lambda_{BE}$  is large, so the vacation time of the LC flow is decreased and the values



| $\lambda_{BE}$ | 95% <i>CI</i> in DRR+ | 95% <i>CI</i> in DRR++ |
|----------------|-----------------------|------------------------|
| 5              | 0.668                 | 0.910                  |
| 7              | 0.782                 | 0.772                  |
| 10             | 2.172                 | 1.152                  |
| 11             | 4.176                 | 2.053                  |
| 13             | 7.359                 | 2.402                  |
| 16             | 193.893               | 4.386                  |
| 18             | 970.229               | 2.942                  |
| 20             | 1743.618              | 2.689                  |

Table 5.2: LC delay in DRR+ and DRR++.

of both parts of the delay in the measurement,  $W_a$  and  $W_b$ , are lower than in the analysis.

Figures 5.1 and 5.2 show that the three BE flows have very similar delay values under different utilizations, which demonstrates that both DRR+ and DRR++ are fair to the BE flows.

### 5.1.2 Effect of quantum, $Q_{LC}$ , under low utilization

In DRR+ and DRR++, the quantum assigned to each flow plays an important role because it decides how many times a connection is served in each round. Moreover, the quantum is the only configurable parameter in the algorithm. There are two issues related to the quantum: how to choose the quantum value and the effect of the quantum on performance. We have discussed selecting the quantum value in Section 3.1.1. In this chapter we focus on the effect of the quantum on performance. There are two possibilities in a round: either the quantum is consumed partially or fully. These two cases occur under low and high utilization respectively. We investigate the performance of LC flow under low utilization in this section and high utilization in the next section.

Intuitively, under low utilization of the congested link, the flow has a short busy period because of the short queue length, so the waiting time of a packet should decrease according to Little's law. In order to test the effect of this parameter on the LC flow, we keep  $\lambda_{BE}$ ,  $\lambda_{LC}$  and  $Q_{BE}$  constant while increasing

only  $Q_{LC}$ . Figure 5.3 shows the behavior of DRR+ and DRR++ under low utilization (78.72%) when the arrival rates of all the flows are 15 packets/sec. When  $Q_{LC}$  is low, packets wait more rounds to be served in both DRR+ and DRR++ because they must accumulate enough of a balance to be forwarded. In addition,  $D_{LC}$  only changes a little in DRR++, while it changes significantly and quickly in DRR+. This is because with low utilization and a given  $Q_{LC}$ , DRR++ can finish servicing more LC packets in one round than DRR+, which results in lower delay.

Using Figures 5.4 and 5.5 we can compare the number of *TotalRounds*, *LCService* and *LCBacklog* of DRR+ and DRR++. In order to compare the number of rounds between DRR+ and DRR++, we define one *round* as a period in which there is at least one queue to be served when the scheduler scans all the queues. The *TotalRounds* increases in DRR+ as  $Q_{LC}$  decreases because DRR+ only serves the LC queue once per round after the LC flow breaks its contract. The total number of rounds is virtually constant in DRR++, which means that a change in  $Q_{LC}$  does not affect the *TotalRounds* under low utilization.

The number of *LCService* is how many times the LC queue is served during the experimental run. In DRR++, this value is larger than the *TotalRounds* because DRR++ may serve the LC queue more than once per round. The number of *LCService* decreases as  $Q_{LC}$  increases because with a larger  $Q_{LC}$ , DRR++ can send more packets in each service interval. The number of *LCService* is a little bit lower in DRR+ than in DRR++ because more packets arrive during two successive LC services.

The *LCBacklog* is how many times the LC queue is still backlogged after exhausting the quantum in the current round. In DRR++ this value decreases dramatically with increasing  $Q_{LC}$  because then most of the packets in the LC queue don't need to wait to be served. The number of *LCBacklog* in DRR++ is about half that in DRR+, probably because the increase in *LCService* helps keep the LC queue short. From Little's law, the shorter queue length generates

lower waiting time for the packets, so the difference in *LCBacklog* also explains why  $D_{LC}$  in DRR++ is lower than in DRR+.

In our experimental runs, we used three BE flows as the background traffic.  $Q_{LC}$  was varied from 11.8% to 40% of the total quantum. In the real world, there may be more BE flows passing through the router, so the delay  $D_{LC}$  with low  $Q_{LC}$  is perhaps of more interest than that with large  $Q_{LC}$ . Figure 5.3 shows that DRR++ has a much lower  $D_{LC}$  than DRR+, and it changes much more smoothly.

### 5.1.3 Effect of quantum, $Q_{LC}$ , under high utilization

If the congested link is heavily utilized, the performance of a flow is different than in low utilization because the queue may build up which results in a larger waiting time. The following tests show that DRR++ still provides a bounded delay for the LC flow while DRR+ fails to guarantee the delay bound of the LC flow once it breaks the contract.

Figures 5.6 and 5.7 present  $D_{LC}$  in DRR+ and DRR++ under high utilization of the bottleneck link (94.45%) when the arrival rates of all the flows are 18 packets/sec. Note that these two figures use very different scales for the LC delay: the values in the scale of Figure 5.7 for DRR+ are 100 times those in Figure 5.6 for DRR++. Under low values of  $Q_{LC}$ , the value of  $D_{LC}$  for DRR+ changes dramatically compared to DRR++.

Figures 5.8 and 5.9 present the number of *TotalRounds*, *LCService* and *LCBacklog* of DRR+ and DRR++ in high utilization.

Although when the  $Q_{LC}$  is small the backlog number in DRR++ is a little smaller than that in DRR+, it decreases more quickly in DRR+: it drops to about 100 when  $Q_{LC} = 1200$  in DRR++ while it reaches 100 when  $Q_{LC} = 2000$  in DRR+. As discussed above, less backlog decreases packet delay because of the short queue length.

We omit the 95% confidence intervals for  $D_{LC}$  in DRR+ in both the low and high utilization cases because the values are very large and similar to those

in Table 1.

## 5.2 Bursty MPEG LC Traffic

MPEG is a typical real time multimedia traffic requiring low delay. It is very challenging for the scheduler because of the large variations in frame length and has a nearly constant frame interarrival time, for example, 1/30 sec if the MPEG is coded as 3 GOP/sec and  $1GOP = \{IPBBPBBPBB\}$ . If a scheduler only considers the bursty interarrival time but does not consider the packet length burstiness, it may fail to guarantee the performance of the MPEG flow because the scheduler treats the MPEG stream as normal traffic. Our tests shows that DRR++ provides a delay bound for the MPEG LC flow by allowing burstiness of the packet length.

Using the same definition as previously, the burstiness,  $\sigma$ , of the MPEG traffic is 45560 bytes if we calculate the frame-burstiness by checking the burstiness frame by frame, and 31436 bytes if we calculate the group-burstiness by checking the burstiness group by group. The reason for the difference is because of the large variance in length of the different frame types.

In this section, we compare the behavior of DRR+ and DRR++ for MPEG traffic while changing  $\lambda_{BE}$  and  $Q_{LC}$ . In DRR+, the contract of the MPEG LC flow,  $C_{LC}$ , is set to  $1.35 \overline{MPEG}$ , where  $\overline{MPEG} = (\bar{I} + 3\bar{P} + 6\bar{B})/10 = 25600$  bytes.

In the following tests, we also discuss the performance effects of both the background traffic intensity and the quantum assigned to the MPEG LC flow. At the same time, we increase the capacity of the bottleneck link to 1.2Mbps because of the larger traffic rate of the MPEG source. Consequently, the arrival rates of the background flows are also increased in order to compete for the link with the MPEG flow.

### 5.2.1 Effect of background traffic intensity

Figures 5.10 and 5.11 show the effects of varying the rates of the three BE flows from 0 to 30 packets per second. Figure 5.10 shows that under DRR++ the delay of the MPEG LC flow approaches an asymptote, as it did with exponential traffic. At the same time, the delay of the BE flows increases dramatically once their arrival rates are larger than 13 packets per second.

This result can prove the robustness of DRR++: the delay of the LC flow is predictable given the quantum value of the BE flow. From the point of view of the receiver node, one of the results is to decrease the buffer spaces required for the stream playback. DRR++ provides better performance on the buffer storage requirement for the MPEG traffic because of the bounded delay value.

Figure 5.11 shows the results for DRR+, where the MPEG LC flow has a larger delay than the BE flows because it has been demoted to best-effort service, and is being served once in every round. At the same time, the large variance in packet length of the MPEG LC flow results in even larger delay than for the BE flows because the *I* frames must wait multiple rounds for transmission. The *P* and *B* frames in each group also have to wait until the *I* frame completes service. Although we can use multiple connections to transform the different types of frame, the receiver needs more space to buffer the arriving frames. The delay for all flows reaches more than one hundred times the delay under DRR++.

### 5.2.2 Effect of quantum, $Q_{LC}$ , under low utilization

We also tested the effect of  $Q_{LC}$  on the MPEG LC traffic for two different values of utilization of the bottleneck link. In these two tests, we only changed the arrival rates of background BE traffic and kept the MPEG traffic constant as above. Figure 5.12 presents the delay of the MPEG LC flow in both DRR++ and DRR+ with low utilization of the bottleneck link (56%) when the arrival rates of the BE flows were 12 packets/sec. Neither values changed much when the  $Q_{LC}$  changed because at low utilization not many BE packets arrive during

one round. So  $Q_{BE}$  isn't fully used, which results in a bounded value of service time for the LC flow. The reason that DRR++ has lower delay than DRR+ is that the MPEG LC packets have more chances to be served.

### 5.2.3 Effect of quantum, $Q_{LC}$ , under high utilization

Figures 5.13 and 5.14 show two totally different delay trends of MPEG LC and BE flows in DRR++ and DRR+ under a little higher utilization (60%) when the arrival rates of the BE flows are 18 packets/sec. In DRR++, the MPEG LC flow reaches very low delay when  $Q_{LC}$  equals 9000 bytes, which is 60% of the total quantum. In DRR+, the delay of the MPEG LC flow decreases from a very large value until  $Q_{LC}$  equals 21000 bytes, which is 78% of total quantum. When  $Q_{LC}$  is 27000 bytes (82% of the total quantum), the delay is still much larger than in DRR++. Although the MPEG LC flow has a large delay in DRR++ when  $Q_{LC}$  is less than 6000 bytes, this value is only about 35% of the maximum packet length of 17000 bytes, which is the maximum packet size of the MPEG traffic. Figure 5.13 shows that the delay of MPEG traffic decreases to a nearly constant low value if  $Q_{LC} = 9000$  bytes, where the scheduler takes a small number of rounds ( $\leq 2$ ) for the LC flow to accumulate enough credit to forward the  $I$  frame.

The behaviors of the BE flows are also different under DRR+ and DRR++. In DRR++, the BE flows exhibit small delays when  $Q_{LC} = 3000$  bytes. Their delay increases dramatically approaching  $Q_{LC} = 9000$  bytes. When the system services the LC packets, the BE packets which are queued have to wait a long time while the  $I$  frame is served. In DRR+, the delay of the BE flows decreases when  $Q_{LC} \geq 21000$  bytes. This is because after the LC flow breaks its contract and is demoted to BE service, all flows have the same opportunity to get served. Only when  $Q_{LC}$  is large enough can the MPEG flow be served quickly. This results in the long waiting time of the BE flows.

## 5.3 Summary

In this chapter, we presented measurement results for DRR+ and DRR++ under two different types of bursty traffic: exponentially distributed traffic and MPEG traffic. Under both types of traffic, we investigated the effects of the intensity traffic and the quantum value of LC flow respectively. The empirical results show that the DRR++ can significantly decrease the delay value and provide a bounded delay value for the LC flow under both kinds of bursty traffic.

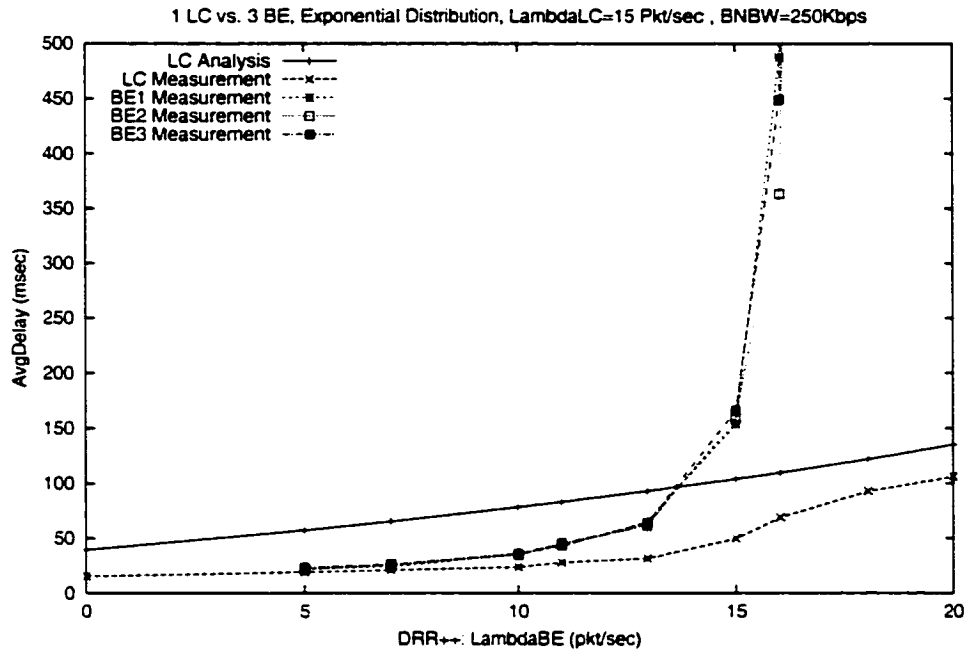


Figure 5.1: DRR++, varying background load.

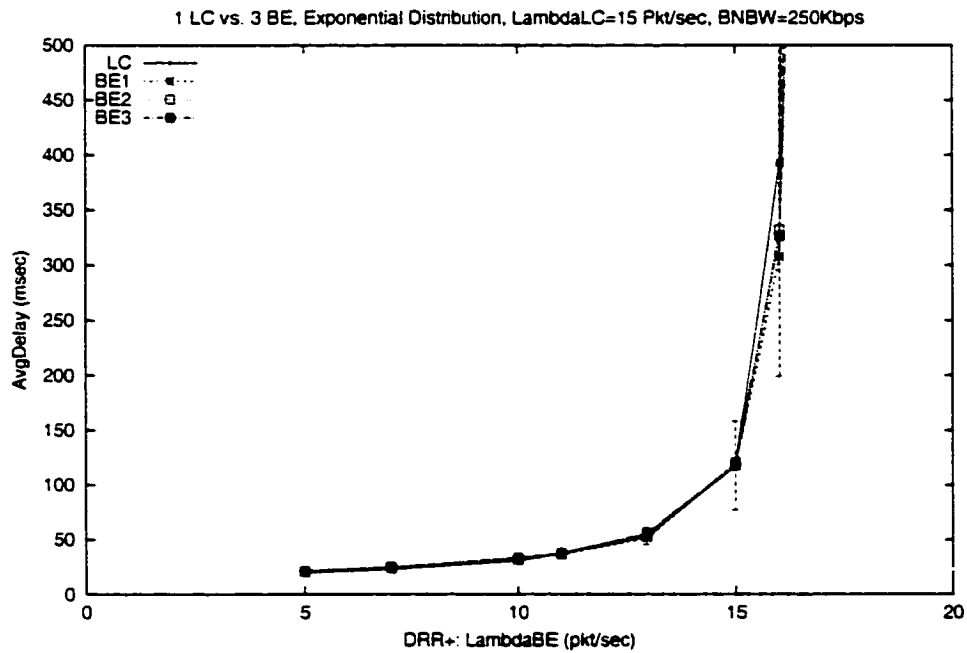


Figure 5.2: DRR+, varying background load.



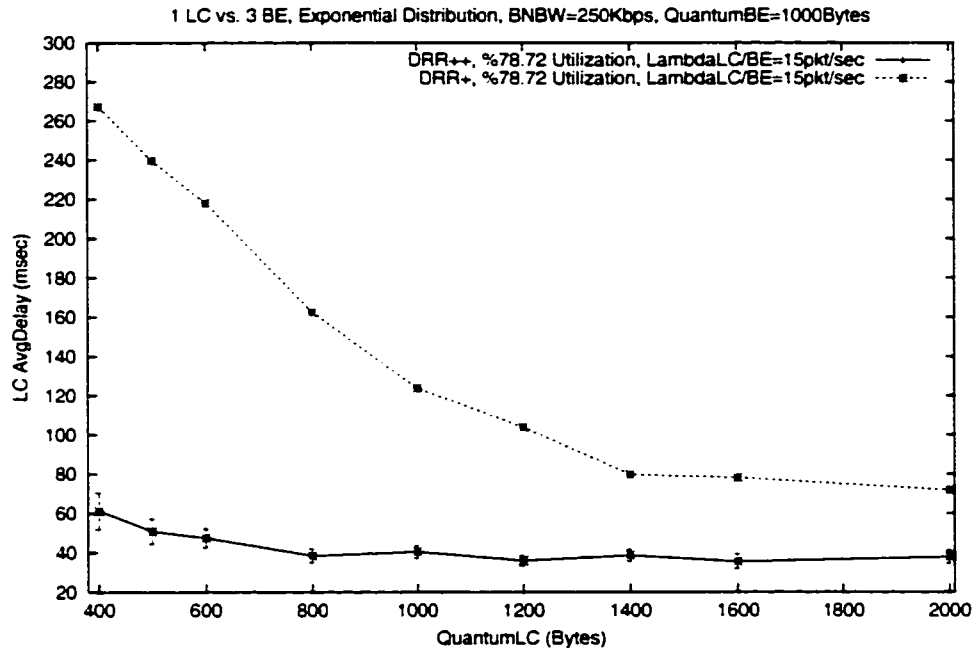


Figure 5.3: DRR++/DRR+, low bottleneck utilization, varying  $Q_{LC}$ .

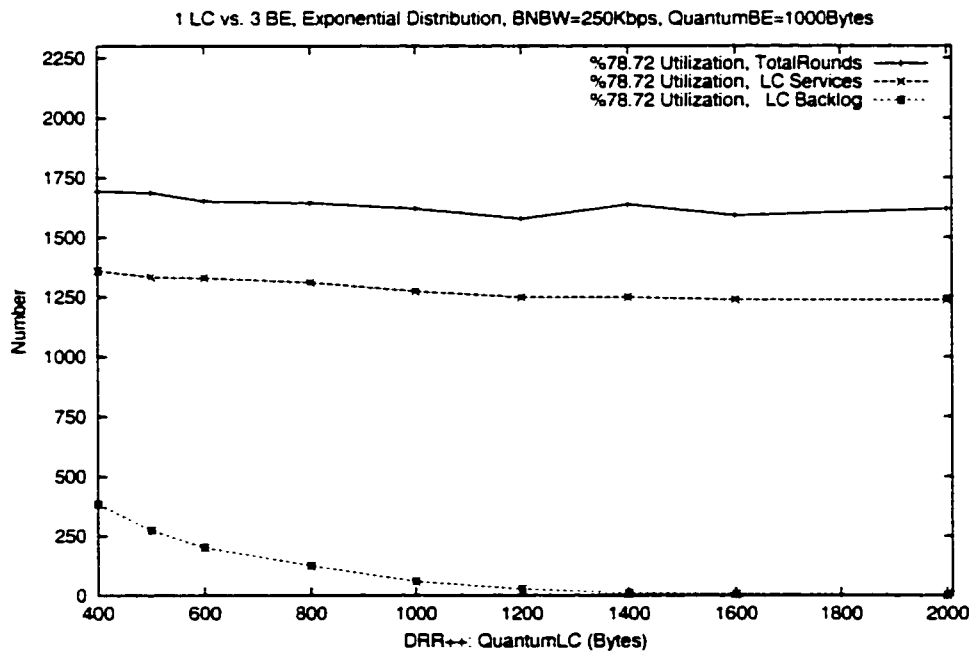


Figure 5.4: DRR++, low bottleneck utilization, varying  $Q_{LC}$ .

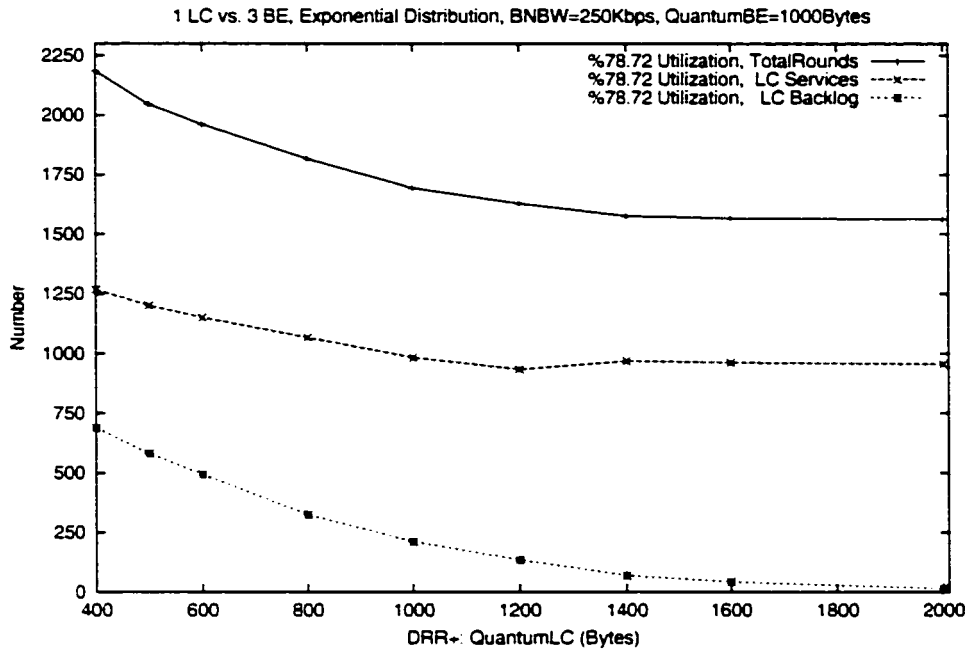


Figure 5.5: DRR+, low bottleneck utilization, varying  $Q_{LC}$ .

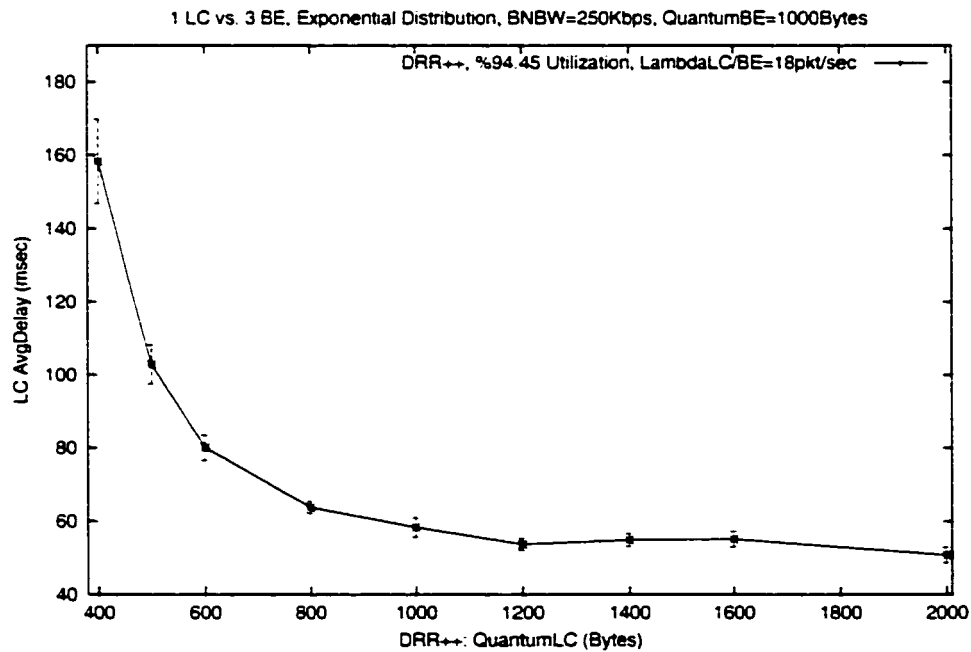


Figure 5.6: DRR++, high bottleneck utilization, varying  $Q_{LC}$ .

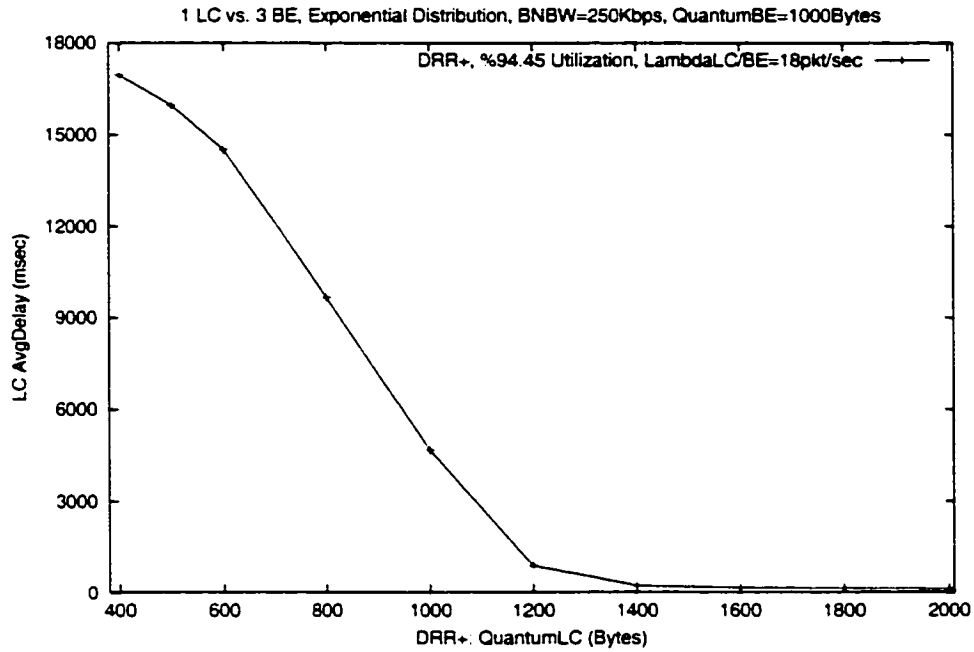


Figure 5.7: DRR+, high bottleneck utilization, varying  $Q_{LC}$ .

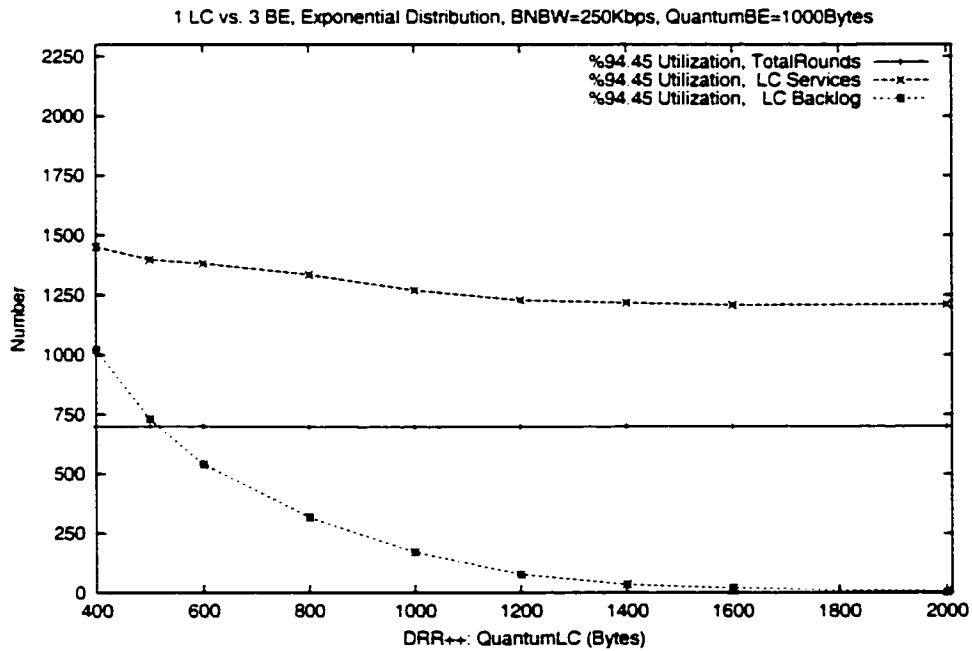


Figure 5.8: DRR++, high bottleneck utilization, varying  $Q_{LC}$ .

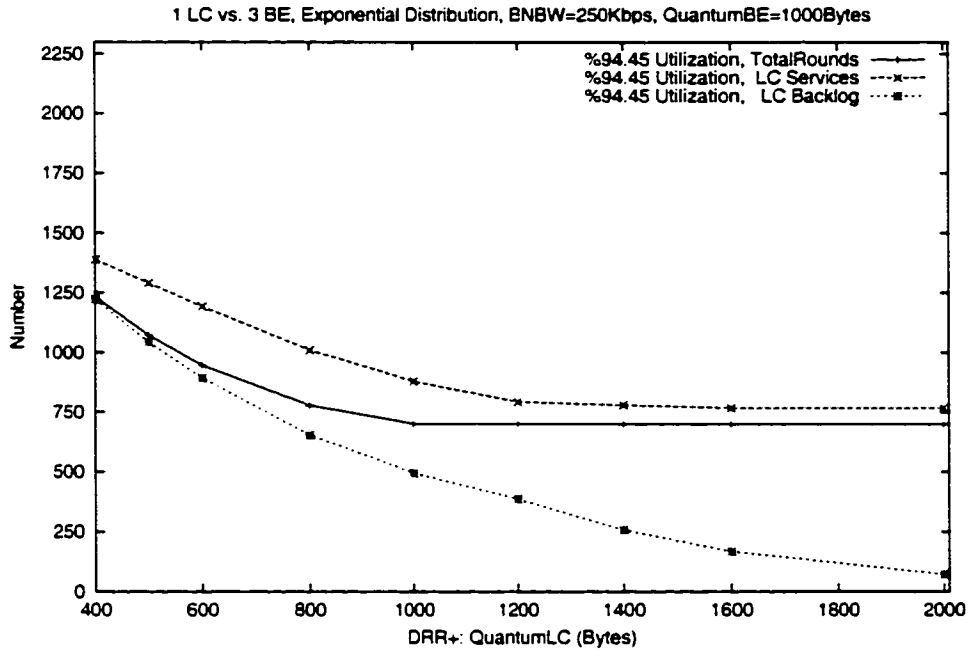


Figure 5.9: DRR+, high bottleneck utilization, varying  $Q_{LC}$ .

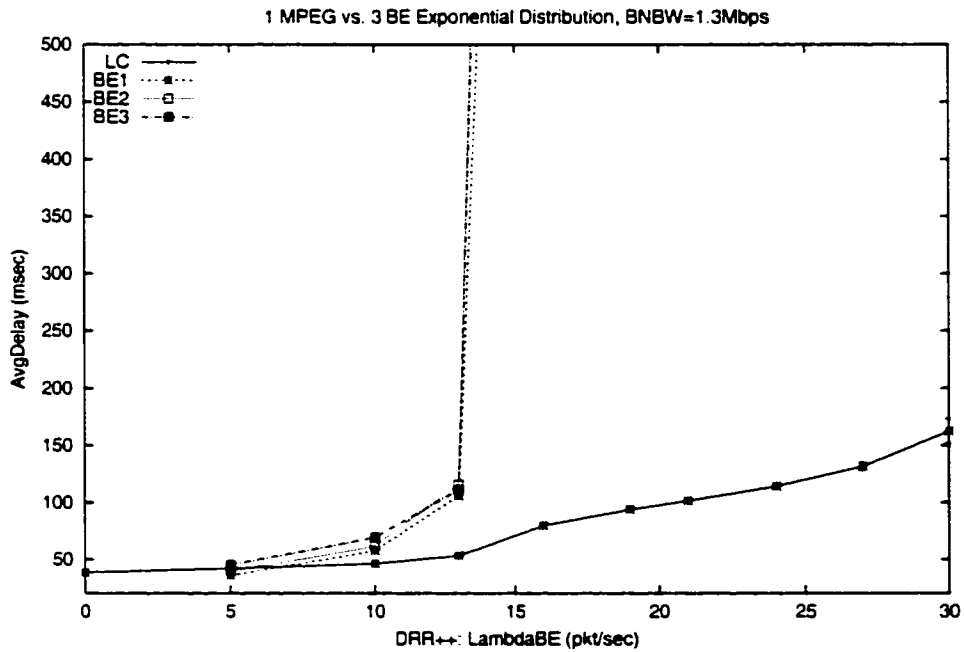


Figure 5.10: DRR++ carrying MPEG, varying background load.

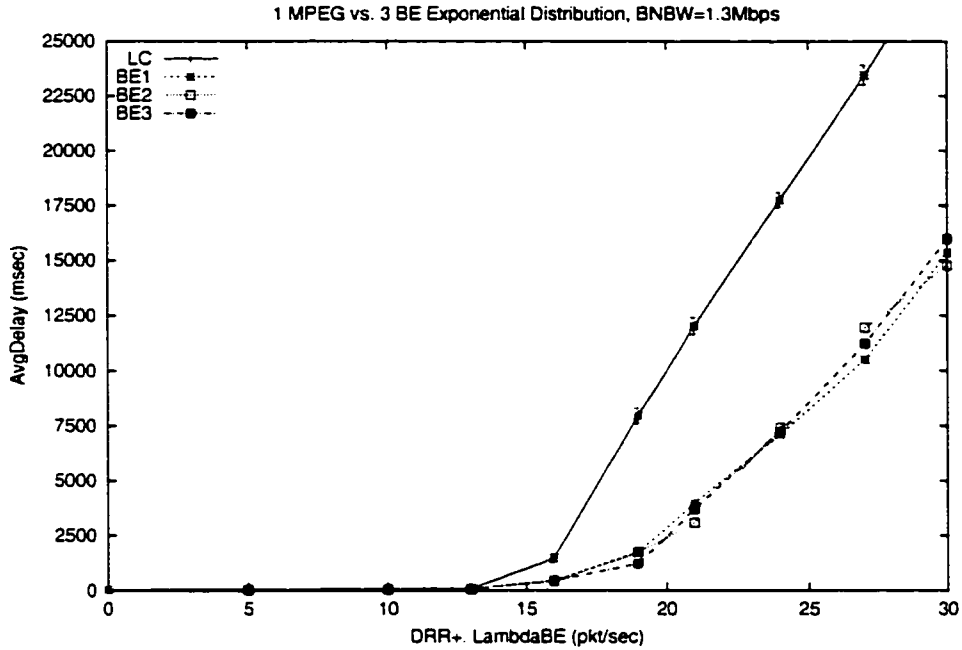


Figure 5.11: DRR+ carrying MPEG, varying background load.

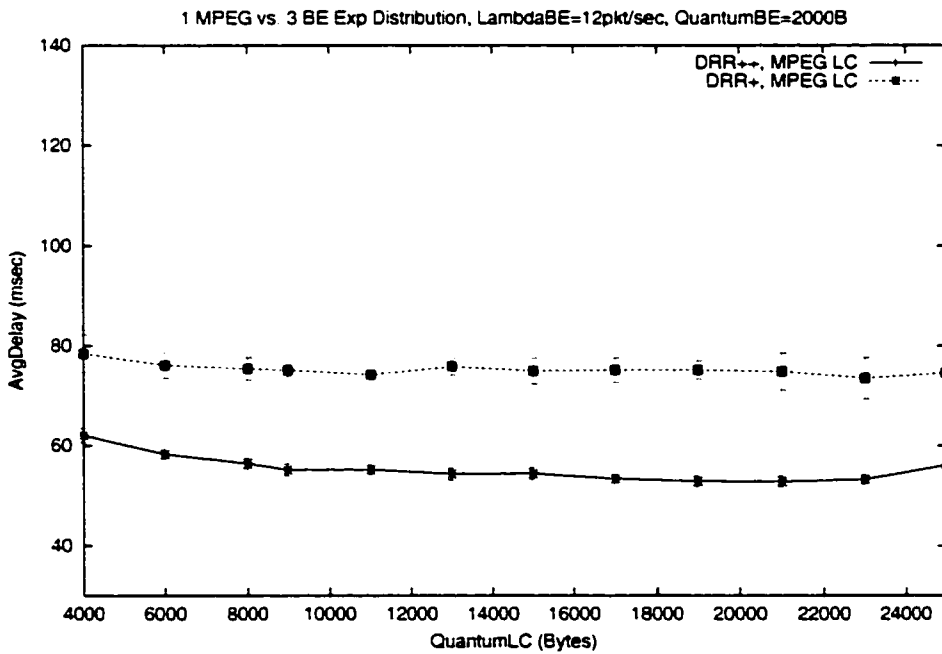


Figure 5.12: DRR++/DRR+ carrying MPEG, low bottleneck utilization, varying  $Q_{LC}$ .

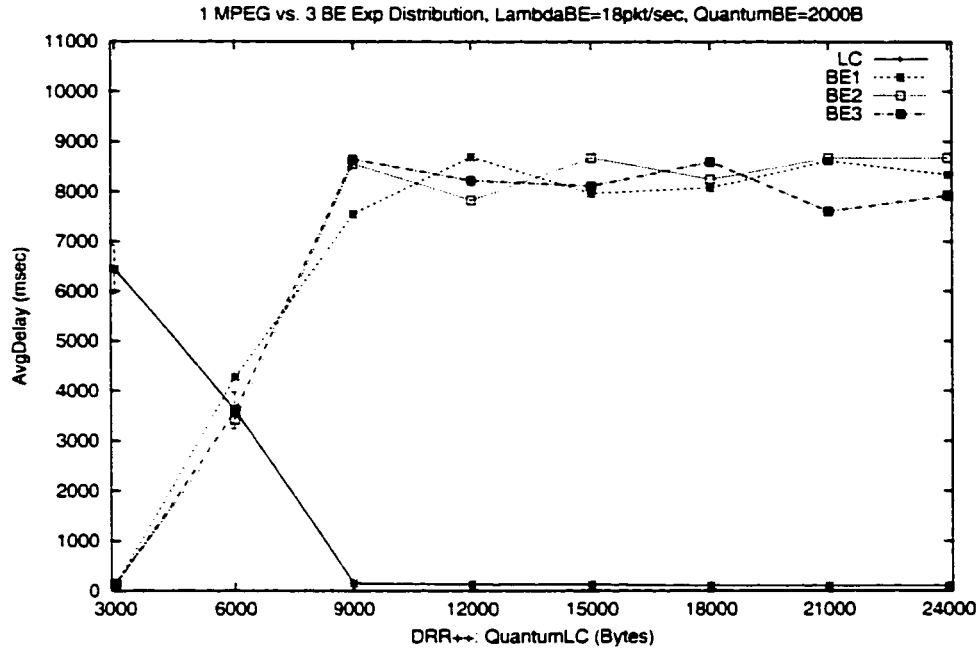


Figure 5.13: DRR++ carrying MPEG, high bottleneck utilization, varying  $Q_{LC}$ .

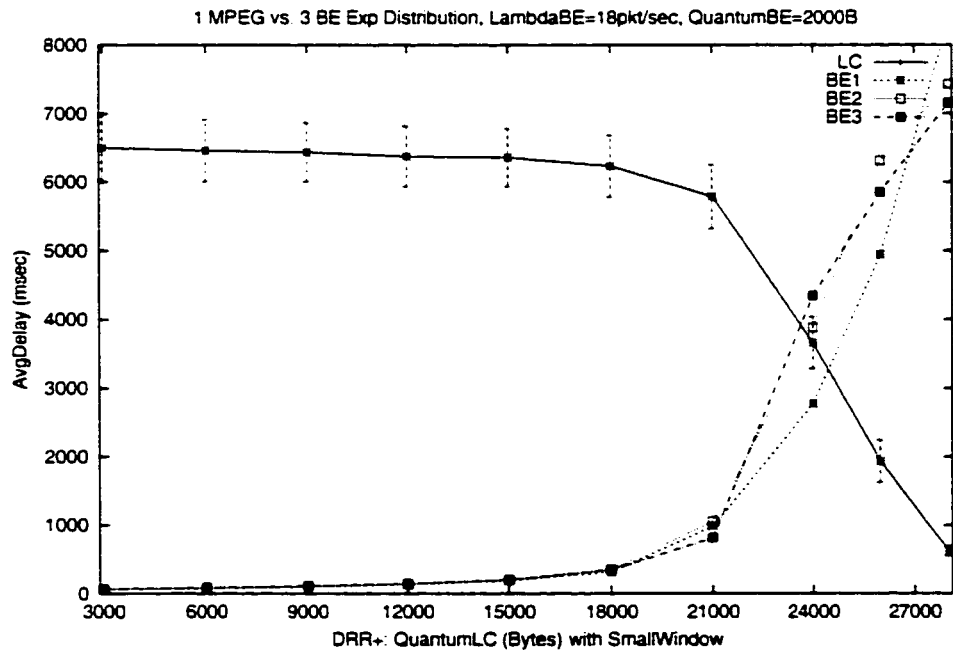


Figure 5.14: DRR+ carrying MPEG, high bottleneck utilization, varying  $Q_{LC}$ .

# Chapter 6

## Conclusions and Future Work

### 6.1 Conclusions

In this thesis, we report the results of a measurement-based evaluation of two packet scheduling algorithms, DRR+ and DRR++. Although both DRR+ and DRR++ work in packet switched networks and have low complexity  $O(1)$  inherited from DRR, DRR++ is a more ideal scheduler for latency critical traffic because it allows short term burstiness of latency critical traffic while DRR+ does not have this feature. Our experiments used two types of bursty traffic — one with bursty interarrival time, and the other with bursty packet length. The traffic models used in our measurements for these two burstiness are exponential distribution and MPEG traffic respectively. We investigated the impact of background traffic and the quantum value of the latency-critical flow on both algorithms with these two types of bursty traffic sources. We also discussed the issues of network topology and protocol type as applied to our tests. In order to overcome clock inaccuracy caused by clock skew and random jumps, we used two network analyzers with precisely synchronized clocks.

DRR++ offers delays for latency critical traffic that are two orders of magnitude less than those in DRR+. Our measurements also show that DRR++ performs much better in terms of isolating latency-critical and best-effort traffic from each other.

Under both types of bursty traffic, as the background traffic intensity

increases, the effects on the latency critical traffic are quite different under DRR+ and DRR++. The delay of latency critical traffic has small increases in DRR++ but large increases in DRR+. As a result, the delay bound of the latency critical traffic in DRR++ is much smaller than that in DRR+. Moreover, DRR++ offers much lower delay variation for the latency critical flow when the intensity of background traffic increases. Because high link utilization is much more challenging for the scheduler, DRR++ provides latency critical traffic with a much better delay guarantee than DRR+.

Since the quantum is an important parameter in both the DRR+ and DRR++ algorithms, we have investigated the behavior of the algorithms with various quantum values. Our results show that the quantum value of the latency critical flow has different effects on performance. In DRR++, the latency critical flow has low delay even when the value of the quantum is relatively small because the latency critical flow has more service opportunities and is allowed to transmit short term burstiness without being punished.

We have also analyzed the mean delay of the latency-critical flow in DRR++ under backlog and compared the analytical results with our measurements. The comparison shows that the analysis provides a good bound for the delay, and that the accuracy of the estimate improves as traffic intensity and link utilization increase.

## 6.2 Future Work

There are several related areas to be explored in the future:

- more complex topologies and traffic loads could be used to evaluate the performance of DRR++.
- buffer management mechanisms which work well with DRR++ need to be explored.
- DRR++ could be implemented within the Linux kernel.



# Bibliography

- [AA97] A. Adas, "Traffic Models in Broadband Networks", *IEEE Communications Magazine*, July 1997, pages 82-89.
- [AS89] A. Demers, S. Keshav, S. Shenker, "Analysis and simulation of a fair queueing algorithm", *Proc. of ACM SIGCOMM'89*, September 1989, pages 1-12.
- [AT96] A. S. Tanenbaum, *Computer Networks, 3rd edition*, Prentice Hall, Inc., 1996.
- [BB94] B. Kim, B.-Y. Kim, "Simulation Study of Weighted Round-Robin Queueing Policy", *Proc. Tech. Conf. on Telecommunications R&D*, Massachusetts, 1994.
- [DA95] D. Stiliadis, A. Varma, "Latency-Rate Servers: A General Model For Analysis of Traffic Scheduling Algorithms", *Technical Report UCSC-CRL-95-38*, University of California, Santa Cruz, July 1995.
- [DA98] D. Stiliadis, A. Varma, "Latency-Rate Servers: A General Model For Analysis of Traffic Scheduling Algorithms", *IEEE/ACM Transactions on Networking*, Vol 6. No.5, October 1998, pages 611-624.
- [DR92] D. Bertsekas, R. Gallager, *Data Networks*, 2nd edition, Prentice-Hall, Inc., 1992.
- [DS96] D. K. Y. Yau, S. S. Lam, "Adaptive Rate-Controlled Scheduling for Multimedia Applications", *Proc. of ACM Multimedia '96*, November 1996, pages 129-140.
- [DS98] D. Saha, S. Mukherjee, S. K. Tripathi, "Carry-Over Round Robin: A Simple Cell Scheduling Mechanism for ATM Networks",

- IEEE/ACM Transactions on Networking*, Vol. 6, No.6, December 1998, pages 779-796.
- [DV98] D. Stiliadis, A. Varma, "Rate Proportional Servers: A Design Methodology for Fair Queueing Algorithms", *IEEE/ACM Transactions on Networking*, Vol. 6, No. 2, April 1998, pages 164-174.
- [FH00] F. Tsou, H. Chiou, Z. Tsai, "Design and Simulation of an Efficient Real-Time Traffic Scheduler with Jitter and Delay Guarantees", *IEEE/ACM Transactions on Networking*, Vol. 2, No. 4, December 2000, pages 255-266.
- [HD93] H. Zhang, D. Ferrari, "Rate-Controlled Static-Priority Queueing", *Proc. of IEEE INFOCOM '93*, April 1993, pages 227-236.
- [HD94] H. Zhang, D. Ferrari, "Rate-Controlled Service Disciplines", *Journal of High Speed Networks*, 3(4):389-412, 1994.
- [HF94] H. Zhang, D. Ferrari, "Improving Utilization for Deterministic Service In Multimedia Communication", *IEEE International Conference on Multimedia Computing and Systems*, May 1994, pages 295-304.
- [HR95] H. Sariowan, R. L. Cruz, G. C. Polyzos, "Scheduling for Quality of Service Guarantees via Service Curves", *Proc. Int. Conf. on Computer Communications and Networks (ICCCN) 1995*, September 1995, pages 512-520.
- [HR99] H. Sariowan, R. L. Cruz, G. C. Polyzos, "SCED: A Generalized Scheduling Policy for Guaranteeing Quality of Service", *IEEE/ACM Transaction on Networking*, Vol. 7, No. 5, October 1999, pages 669-684.
- [HS91] H. Zhang, S. Keshav, "Comparison of Rate-Based Service Disciplines", *Proc. of ACM SIGCOMM'91*, September 1991, pages 113-121.

- [HZ95] H. Zhang, "Service Disciplines For Guaranteed Performance Service in Packet-Switching Networks", *Proc. IEEE*, Vol. 83, Oct. 1995, pages 1374-1396.
- [IH00] I. Stoica, H. Zhang, T.S.E. Ng, "A Hierarchical Fair Service Service Curve Algorithm for Link-Sharing, Real-Time, and Priority Services", *IEEE/ACM Transaction on Networking*, Vol. 8, No. 2, April 2000, pages 185-199.
- [JE97] J. Bruno, E. Gabber, B. Özden, A. Silberschatz, "Move-To-Rear List Scheduling: a new scheduling algorithm for providing QoS guarantees" *Proc. of ACM Multimedia '97*, November 1997, pages 63-73.
- [JK00] J. F. Kurose, K. W. Ross, *Computer Networking: A Top-Down Approach Featuring the Internet*. Addison Wesley Longman, 2000.
- [KC94] K. C. Claffy, *Internet Traffic Characterization*, Ph.D thesis, University of California, San Diego, 1994
- [KH94] M. Krunz, H. Hughes, "A Traffic Model for MPEG-Coded VBR Streams", *Proc. of ACM SIGMETRICS '95*, May 1995, pages 47-55.
- [LZ90] L. Zhang, "Virtual Clock: A new traffic control algorithm for packet switching networks". *Proc. of ACM SIGCOMM'90*, September 1990, pages 19-29.
- [MH95] M. Krunz, H. Hughes, "A Traffic Model for MPEG-Coded VBR Streams", *ACM SIGMETRICS'95*, May 1995, pages 47-55.
- [MG96] M. Shreedhar, G. Varghese, "Efficient Fair Queueing Using Deficit Round-Robin", *IEEE/ACM Transactions on Networking*, Vol 4, NO. 3, July 1996, pages 375-385.
- [MS96] M. Schwartz, "Broadband Integrated Networks", *Prentice Hall, Inc.*, 1996.

- [MW00] M. H. MacGregor, W. Shi, "Deficits for Bursty Latency-Critical Flows: DRR++", *Proc. of IEEE ICON '00*, Singapore, 2000, pages 287-293.
- [RJ91] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*, Wiley-Interscience, New York, NY, April 1991.
- [RV99] R. Guérin, V. Peris. "Quality-of-service in packet networks: Basic mechanisms and directions", *Computer Networks*. Vol. 31, Issue 3, February 1999, pages 169-189.
- [SG94] S. Golestani. "A self-clocked fair queueing scheme for broadband applications". *Proc. of IEEE INFOCOM'94*, June 1994, pages 636-646.
- [SI00] S. Iatrou, I. Stavrakakis. "A Dynamic Regulation and Scheduling Scheme for Real-Time Traffic Management". *IEEE/ACM Transactions on Networking*, Vol. 8, No. 1, February 2000, pages 60-70.
- [SK96] S. Keshav. *An Engineering Approach to Computer Networking*, Addison-Wesley Inc., 1996.
- [SK98] S. Bhattacharjee, K. L. Calvert, E. W. Zegura, *Network Support for Multicast Video Distribution*, Technical Report 98-16, Georgia Inst. of Tech., 1998.
- [SY01] S.-C. Tsao, Y.-D. Lin, "Pre-order Deficit Round Robin: A new scheduling algorithm for packet-switched networks", *Computer Networks*. Vol. 35, Issue 2-3, February 2001, pages 287-305.
- [UB98] U. Black, *ATM: Foundation for Broadband Networks*, Prentice Hall, Inc., 1998.
- [VG98] V. Paxson, G. Almes, J. Mahdavi, M. Mathis, *RFC2330: Framework for IP Performance Metrics*, May 1998.
- [VP98] V. Paxson, "On Calibrating Measurements of Packet Transit Times". *Proc. of ACM SIGMETRICS '98*, June 1998, pages 11-21.