Improved Queuing Algorithms in QoS Enabled Internet Node

by
Matthew (Yun Lam) Wong

Chief Supervisor Dr. C. K. Li

Department of Electronic and Information Engineering
The Hong Kong Polytechnic University,
Hunghom, Kowloon,
Hong Kong

A THESIS SUBMITTED FOR THE DEGREE OF MASTER OF PHILOSOPHY TO THE HONG KONG POLYTECHNIC UNIVERSITY

ABSTRACT

This project is supported by the Hong Kong Polytechnics University research grant no. G-V989. In this project, traditional queuing algorithms were studied and analyzed. Two new algorithms are proposed. Weighted Deficit Probability Drop (WDPD) further enhances the throughput fairness among groups of different weighted flows. Weighted Deficit Round Robin (WDRR) enables low complexity QoS.

There are many existing queuing algorithms and some of them are implemented and introduced in the commercial market. Weighted Fair Queuing (WFQ), which was introduced in 1993 and is being implemented in most of the Cisco Routers, which use to aggregate traffics less than 2 Mbps. Although the WFQ and Worst-case Fair Weighted Fair Queuing Plus (WF^2Q+) are relatively fair among the same weighted flow, their algorithm needs high computational cost to calculate the virtual finish time and the eligible virtual finish time respectively. They are used to approximate the General Processor Sharing (GPS) system. The high computational requirement of the algorithm makes it impractical to be implemented on high-speed link to support similar Quality of Service (QoS).

In this thesis, a number of queuing algorithms are introduced and analyzed. Network simulations are used to validate the network properties of the algorithms. Fairness is hard to define and quantify, a different point of view on the ambiguous term "fairness" will be discussed. At last, two new algorithms: the Weighted Deficit Probability Drop (WDPD) and the Weighted Deficit Round Robin (WDRR) are developed and discussed. WDPD enables a higher fairness between different weighted flows and WDRR enables a low complexity QoS support in Internet node.

ACKNOWLEDGMENTS

The author wishes to convey sincere thanks to his supervisor, Dr. Chi-Kwong Li, for excellent supervision of the project, invaluable advises and guidance, kindness help and support. Also thanks to Dr. Chi-Kin Leung, Dr. Wai-Kin Lam, Prof. Prasant Mohapatra, Dr. King-Tim Ko and Dr. Kwok-Tung Lo for their patience and endless advises. Furthermore, thanks to my colleagues Dr K. F. Wan and Mr. Kam-Tai Yam for their generous contribution and support on maintaining the simulation computers and software. Thanks to the supporting community in the Network Simulator 2 forum, which gives me advices on the queuing algorithm simulation codes, network traffic models and the NS2 network parameter logging. Some individuals in the NS2 forum are very kind and generous to give me their own implementation program codes for the existing queuing algorithms. Last but not the least, I must give thanks to my family for their love and kindness. Without one of the above people this research project and my thesis will never come to existence.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

ABBREVIATIONS

ASIC	Application Specific Integrated Circuit
ATM	Asynchronous Transfer Mode
BR	Bit-by-bit Round Robin
CBR	Constant Bit Rate
DRR	Deficit Round Robin
Delay-EDD	Delay Earliest-Due-Date
EDF or EDD	Earliest-Due-Date-First
FBFQ	Frame-Based Fair Queuing
FIFO	First-In-First-Out
FPGA	Field Programmable Gate Array
FQ	Fair Queuing
FQS	Fluid Queuing System
FRED	Flow Random Early Drop
GPS	General Processor Sharing
GUI	Graphical User Interface
HRR	Hierarchical Round Robin
INSANE	Internet Simulated ATM Networking Environment – A Network Simulator by University of California Berkeley
IP	Internet Protocol
Jitter-EDD	Jitter Earliest-Due-Date
LBNL	Lawrence Berkeley National Laboratory – Organization for NS1 development
LFVC	Leap Forward Virtual Clock N
NEST	Network Simulation Test-bed – A Network Simulator by department of Computer Science at Columbia University
NS2	Network Simulator 2 – A simulator by University of California Berkeley
OPNET	Optimized Network Engineering Tools – A Network Simulator by MIL3 Inc.

PDES	Parallel Discrete-Event Simulation
PFQ	Packetized Fair Queuing
PGPS	Packetized General Processor Sharing
QoS	Quality of Service
RCAP	Real-time Channel Administration Protocol
RCSP	Rate Controlled Static Priority
REAL	Realistic And Large – A Network Simulator by S. Keshav at Cornell University
RED	Random Early Drop
RIO	Random Early Drop with In and Out
RR	Round Robin
S&G	Stop and Go
SCFQ	Self-Clocked Fair Queuing
SFQ	Stochastic Fairness Queuing
TCL	Tool Command Language
TCP	Transmission Control Protocol
ToS	Type of Service
UDP	User Datagram Protocol
VC	Virtual Clock
VFT	Virtual Finish Time
VST	Virtual Start Time
VT	Virtual Time
WDRR	Weighted Deficit Round Robin
WDPD	Weighted Deficit Probability Drop
WFI	Worst-case Fairness Index
WFQ	Weighted Fair Queuing
WF^2Q	Worst Case Fair Weighted Fair Queuing
WF^2Q+	Worst Case Fair Weighted Fair Queuing Plus

CHAPTER 1 INTRODUCTION TO NETWORK QUEUING ALGORITHMS

Stepping into the Information Era, information technology industry grows rapidly with Internet transforming human's living style and the way they access information. In the past, people mainly receive information from newspapers, magazines and mail. Nowadays more options are open to them such as electronic handheld devices, computer terminal and electronic mailing system. The media for the information storage is improving. Since most of the data need to be accessed remotely, the way of transmitting information needs to be improved as well. Networking is the key. In order to get all the devices to communicate in the same network, protocol standard is very important. Many consortiums are formed in recent year in order to develop and standardize common protocols over different media.

Number of devices that need network access have been boosted up recently, the network quality also needs to be improved. The quality of a network can be defined in terms of latency, throughput, packet drop and delay jitter. Different types of traffic require different network quality parameters. For example, the real time interactive application such as Internet Phone requires low latency, the online video show requires high throughput and low delay jitter, banking information with high data integrity requires low packet drop. The Quality of Service (QoS) requirements for different application types are shown in Table 1-1. The key to control these network parameters is the packet schedule algorithm in the QoS enabled network node.

Traditional Network does not have any control over the traffic and the packets are sent without any manipulation and scheduling mechanism. Best Effort is the simplest service model for the traditional network and the existing Internet. The "best-effort" means that the network providing the best service according to the existing resources. Packet can get lost, corrupted, mis-delivered or fail to reach its intended destination in any way. The network did nothing to handle these problems. It tries the best effort to provide the services. Best Effort does not offer "quality control", this has to be done by the protocol stack at the end host. For example, TCP acknowledges packets to confirm that they have been received correctly. If the source of a flow does not receive an acknowledgement, it simply tries again. For real-time services such as VoIP, the receiver does not have time to wait for retransmission of lost packets. Since, even small losses in packets will result in decreased quality.

Application Types	**QoS requirements**			
	Bandwidth	**Latency**	**Jitter**	**Packet Loss**
E-Mail	Low to Moderate	-	-	-
File Transfer	High Burst	-	-	-
Telnet	Low Burst	Moderate	-	-
Streaming Media	Sustained Moderate to High	Sensitive	Sensitive	Sensitive
Videoconferencing	Sustained High	Critical	Critical	Sensitive
Voice over IP	Sustained Moderate	Critical	Critical	Sensitive

Table 1-1 QoS requirements for common application types.

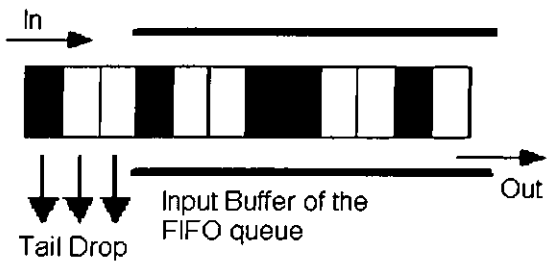

Figure 1-1 FIFO queuing for Best Effort Service

The network aggregation node in the best effort service model puts all packets from different flows into one incoming First-In-First-Out (FIFO) queue. Packets drop from the

tail if the buffer is full. The packet from different flows compete for the bandwidth of the output link in the aggregation node. This is similar to Ethernet, the hosts in the network segment compete for the bandwidth in a random manner resulting a non-efficient network. The network cannot categorize the packet according to their needs and provide the required service level. How can we solve this problem? A suggested improvement for Best Effort is the use of the 3-bit IP precedence field. By using this field, a packet with higher precedence arrives before than a packet with lower precedence if both packets are passed into the network at the same time with the same destination and IP options. In addition, the IP precedence field is related to discard probability, meaning that if a sequence of marked packets is passed along a path, the packet sequence with the elevated precedence should experience a lower discard probability. However, network hardware and software have traditionally not been configured to use this field. The other option is to handle traffic in a way that always has enough capacity available in the network. However at the speed that the Internet is evolving today, this would require an over-dimensioning in capacity to handle the ever-increasing traffic. This resulting high costs and therefore impractical.

In concern to fairness, Best Effort has two primary user groups: Ordinary users and skillful users with considerable ability to tune their computer system. The harmful part of the skillful user group always tries to exploit the network in order to receive the best throughput possible, and this will degrade the service delivered to the other users. If this happens in a large scale, it does not only deteriorate overall fairness but also the service for all users. The Best Effort cannot deliver the Quality of Service that is necessary to be able to transfer real-time streaming at the required quality. In order to separate the time-dependent flows from the rest, the network has to promise service guarantees. This can be achieved by more sophisticated queuing algorithms that support QoS.

Queuing algorithm is working in a way that schedules how the packets in different flows passing to the output queue of a node. Some queuing algorithms are claimed to be fair so that a misbehaving application (one that continues to send during times of congestion) will not be destructive to other better-behaved applications. Queuing algorithm also determines how packets are dropped when congestion occurs in a router. A scheduler can provides rate shaping by enforcing a packet stream to be conformant to a traffic profile. A bursty interactive flow could be shaped to a constant bit rate in a scheduler. Shaping is often based on the leaky bucket mechanism, since packets leak out of the queue at a fixed rate. Thus, the bursts of a stream will be smoothed and non-conforming packets will be delayed.

In this thesis, firstly different queuing algorithm simulations and testing for their network performance will be introduced. Secondly the simulation models are validated by analyzing the simulation results. Thirdly, the characteristics of the algorithm are summarized and analyzed. Finally, two algorithms – Weighted Deficit Probability Drop (WDPD) and Weighted Deficit Round Robin (WDRR) are developed and further analyzed.

CHAPTER 2 SURVEY OF EXISTING ALGORITHMS

In this chapter the current approach of the queuing algorithms will be introduced. The queuing / packet scheduling algorithms are divided into two categories, they are Non-Work-Conserving disciplines and Work-Conserving disciplines. In non-work-conserving discipline, each packet is assigned either explicitly or implicitly an eligibility time. Even when the server is idle, if no packets are eligible, no packet will be transmitted. The non-work conserving server may be idle even when there are packets backlogged. The discipline emphasizes on reshaping traffic inside the network. Work-conserving server is never idle when there are packets backlogged. Outgoing link is always fully utilized if any packet present. It emphasizes on fair queuing. The queuing algorithms can also be divided into rate allocating service discipline and rate controlled server discipline. In rate allocating service discipline, the server will serve packets at the higher rate as long as it will not affect the performance guarantees made to other flows. Rate controlled service disciplines will not serve packets at a higher rate under any circumstances. All non-work-conserving disciplines are in the rate controlled services category. This is because only non-work-conserving disciplines get an upper bound on service rate of a flow.

In the work-conserving disciplines, there are many kinds of queuing algorithms such as Generalized Processor Sharing (GPS) ideal model. Other algorithms are evolved from the ideal model and implement to work in practical environment. The ideal model also simplified to lower the computational requirement. Other evolved queuing algorithms in this discipline are called as Packetized Generalized Processor Sharing or Packet Fair

Queuing. They are Worst-case Fair Weighted Fair Queuing Plus (WF^2Q+), Worst-case Fair Weighted Fair Queuing (WF^2Q), Weighted Fair Queuing (WFQ), Fair Queuing (FQ), Self Clocked Fair Queuing (SCFQ), Virtual Clock (VC), Stochastic Fair Queuing (SFQ) and Delay-Earliest-Due-Date (Delay-EDD). In the non-work-conserving disciplines, there are Jitter-Earliest-Due-Date (Jitter-EDD), Stop and Go (S&G), Round Robin (RR), Deficit Round Robin (DRR), Hierarchical Round Robin (HRR), Random Early Drop (RED), Random Early Drop with In and Out (RIO) and Flow Random Early Drop (FRED). Each queuing algorithm will be introduced in this chapter.

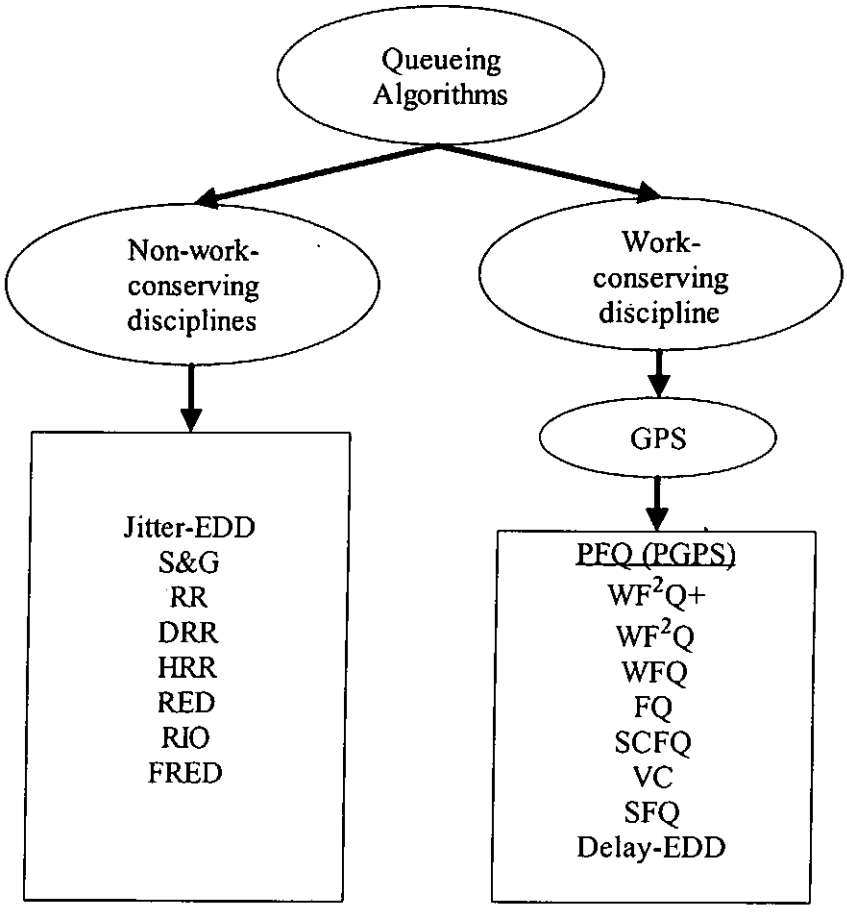

Figure 2-1 Overview of Queuing Disciplines

2.1 WORK-CONSERVING DISCIPLINE

2.1.1 General Processor Sharing

GPS [25, 26], proposed by Parekh and Gallager, it is the traditional idea of work-conserving queuing algorithms. GPS is an ideal fluid system in which the traffic is infinitely divisible and multiple traffic streams can receive service simultaneously. At any given time, the head-of–queue cells from multiple FIFO queues can simultaneously be in the process of transmission from the fluid GPS. It distributes the idle bandwidth among non-idle buffers in proportion to their bandwidth allotments. The minimum service unit is a bit. This is the most fairness queuing algorithm that never implement in the real networking environment. It involves high computational complexity. In the real life environment the packet in networks cannot transmit bit by bit. The packet header and the payload need to be transmitted together to ensure the data integrity and correct routing path. Packet Fair Queuing (PFQ), a packet system, which a traffic stream can, receives service at a time and the minimum service unit is a packet. It enables the implementation of an approximated GPS system in the real life. It also called Packetized General Processor Sharing.

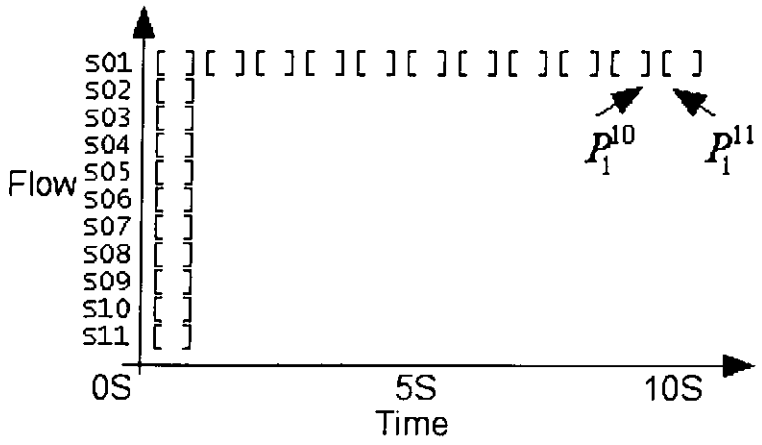

Figure 2-2 The Packet Arrival Pattern

Figure 2-2 shows the input packet pattern of the flows. There are 11 flows sharing a one packet per second link in a node. The first flow sends 11 packets and the rest of the flows send 1 packet. All the 11 flows send packets at the zero second and all packets with the same packet length 1. In the GPS system, the flows go through the packet policy and the first flow receives service at 0.5 packet/sec, the rest of the flows receive service at 0.05 packet/sec. After processed in the GPS system, the packet output pattern is shown in figure 2-3. The figure shows GPS is a fluid system and all the packets in the 11 flows can process simultaneously.

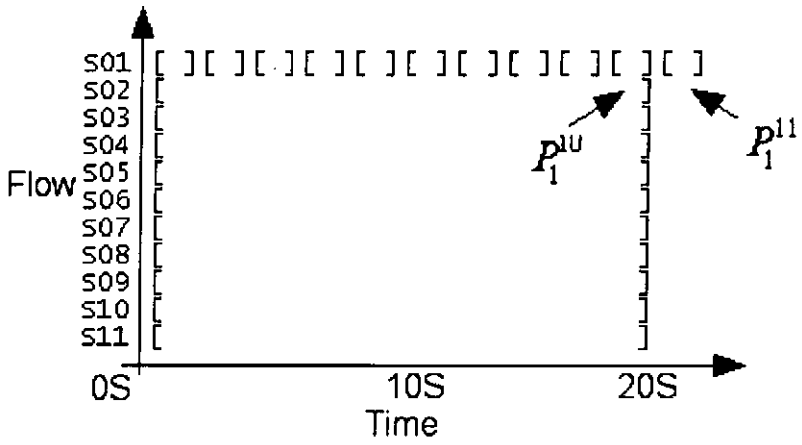

Figure 2-3 The packet output pattern of the GPS system

In figure 2-3, the packets in flow 1 received 0.5 packet/sec and it sends packet at 2 seconds interval. The last packet totally sent at 22 second. All the other flows received 0.05 packet/sec and their packet totally sent at 20 second.

2.1.2 Weighted Fair Queuing

WFQ, the most commercially acceptable fair queuing algorithm which implemented by Cisco in the low line speed router with less than 2Mbps serial interface. In WFQ, every session gets minimum amount of guaranteed bandwidth and guaranteed upper bound on delay. Active session proportionally share the total link bandwidth. The link can be fully utilized. It simplified the end-to-end congestion control mechanism. It separates handling

of different sessions in different queues so the misbehaving or greedy session only punishes itself. This algorithm can protect well-behaved sources from the ill-behaved source. Because the WFQ approximates the GPS in the packetized system, it is called Packetized Generalized Processor Sharing (PGPS). It extends the idea of A. Demers, S. Keshav and S. Shenkar in Fair Queuing (FQ) [1]. The virtual time concept is developed to approximate the GPS system time. The virtual time depends on how many other connections are active in the system. The server chooses the next packet for transmission among all the packets that are backlogged at all active flows, the first packet that would complete service in the corresponding GPS system will be transmitted first (that means the backlogged packet with the smallest virtual finish time will be transmitted first). Because the system selects packet depending on the real-time information, it is called a Fluid Queuing System (FQS). The FQS always requires high computational cost. The following are the equations of WFQ to calculate the virtual start time (VST) S_i^k and the virtual finish time (VFT) F_i^k.

$$S_i^k = \max\{F_i^{k-1}, V(a_i^k)\} \tag{2.1}$$

$$F_i^k = S_i^k + \frac{L_i^k}{\phi_i} \tag{2.2}$$

where k^{th} session with packet i arrives at time a_i^k with the packet size L_i^k. ϕ_i is the weighted share of the flow. $V(a_i^k)$ is the system virtual time function, which approximates the GPS clock. The Virtual function is illustrated below:

A GPS server serving N sessions is characterized by N positive real numbers, ϕ_1, ϕ_2, ϕ_3, ... ϕ_N. The server operates at fixed rate r and the server rate for flow i is r_i:

$$r_i = \frac{\phi_i}{\sum_{j=1}^{N} \phi_j} \times r \tag{2.3}$$

The virtual function is given by:

$$V(0) = 0$$
$$V(t_{j-1} + \tau) = V(t_{j-1}) + \frac{\tau}{\sum_{j=1}^{N} \phi_j} \qquad (2.4)$$
$$\tau \le t_j - t_{j-1}, j = 2,3,.....$$

j is the session of a specific flow.

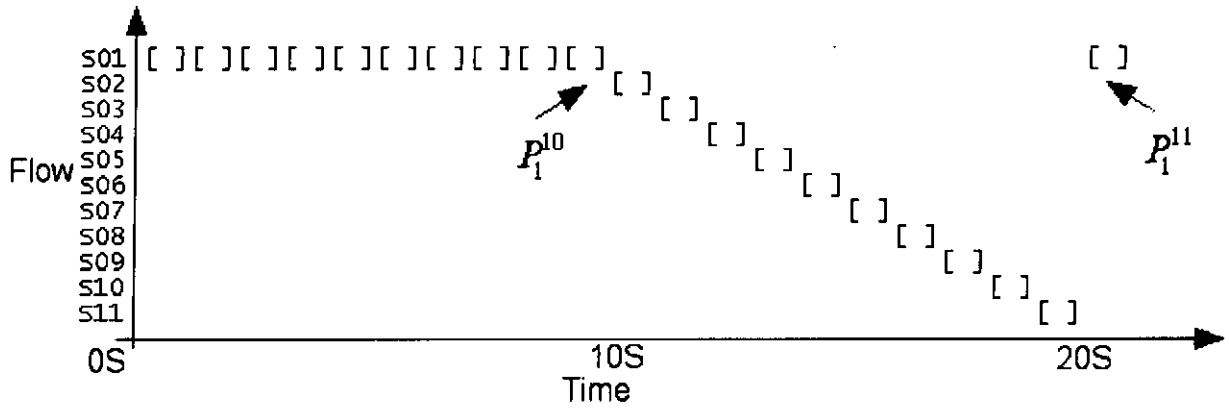

Figure 2-4 The packet output pattern of the WFQ system

WFQ de-queue the flow, which gets the smallest virtual finish time among all the active flows, which backlogged, with packets. The WFQ FQS needs large computational effort and suffers from scalability problem. Figure 2-4 is the packet output pattern of WFQ with the packet arrival pattern shows in figure 2-2. This figure shows that the packets cannot process simultaneously in the real practical environment. The VFT of the first session packet P_1^1 is 2, P_1^2 is 4, P_1^3 is 6, ... P_1^{10} is 20 and P_1^{11} is 21 (p_k^i - i^{th} packet at k^{th} session). The VFT of the first packet in the other ten sessions P_2^1, P_3^1, P_4^1, ... P_{11}^1 are 20. All the packets with the smallest VFT will transmit first, so the first ten packets in flow 1 transmit first and then the packets of the other tenth flows transmit next. The last packet of the first flow with the largest VFT transmits last. This WFQ becomes unfair when the number of sharing sessions increasing dramatically.

2.1.3 Worst-case Fair Weighted Fair Queuing

In 1996, the Worst-case Fair Weighted Fair Queuing (WF^2Q) was proposed by Jon C.R. Bennett and Hui Zhang [2]. In WF^2Q, the virtual time function, the virtual start and finish time calculation is exactly the same as the WFQ. Only the packet selection policy is changed. This is the most accurate discipline that approximating the fluid GPS. WF^2Q uses both the start times and the finish times to select the priority of the packet. In WF^2Q system, the next transmitting packet is NOT chosen among all backlogged packet in all active flow. The system only considers the packets that are started or finished receiving service in the corresponding GPS system.

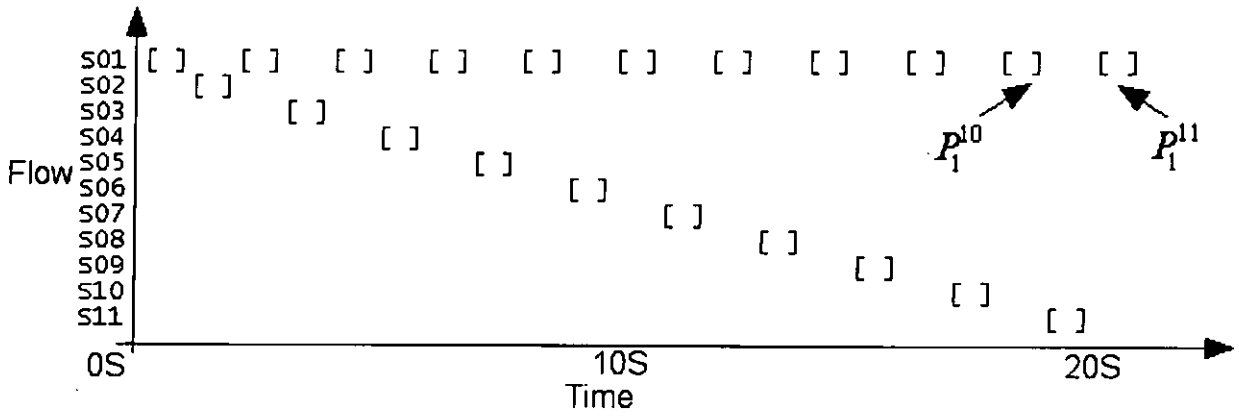

Figure 2-5 The packet output pattern of the WF^2Q system

Figure 2-5 is the packet output pattern of the WF^2Q system with the packet arrival pattern shown in figure 2-2. The Virtual Start Times (VST) of the first packet in the 11 sessions P_1^1, P_2^1, P_3^1, ... P_{11}^1 are zero. The VST of P_1^2 is 2, P_1^3 is 4, ... P_1^{11} is 20. The VFT of P_1^1 is 2, P_1^2 is 4, P_1^3 is 6, ... P_1^{10} is 20 and P_1^{11} is 21. The VFT of P_2^1, P_3^1, P_4^1, ... P_{11}^1 are 20. Consider at time 0s, only packet with VST < 0^+ will be compared, that mean only all the P_k^1's will be compared. Among the P_k^1 packets, the P_1^1 with the smallest VFT which is 2, it will be transmitted first. At time 1s, the next packet needs to be selected among the

packet with VST < 1^+. Only the P_2^1, P_3^1, ... P_{11}^1 are eligible for the selection, all the P_2^1, P_3^1, P_4^1, ... P_{11}^1 are with the VFT 20, the first flow (S2) with packet P_2^1 is being transmitted. At time 2s, the next packet needs to be selected among the packet with VST < 2^+. Only, P_1^2, P_3^1, P_4^1, ... P_{11}^1 are in the selection pool, the P_1^2 with the smallest VFT and it is being transmitted. The WF^{2}Q is the most accurate algorithm in approximating the GPS system and it can provide the tightest delay bound.

2.1.4 Worst-case Fair Weighted Fair Queuing Plus

WF^{2}Q+ [6] further modifies the packet selection policy and enhances the idea of the VFT, the smallest eligible virtual finish time first (SEFF) was introduced. The packet is said to be eligible at time t if the virtual start time is no greater than the current system virtual time. Furthermore a new system virtual time function $V_{WF^2Q+}(t)$ is used. The enhancement reduces the computational complexity significantly and allows supporting the higher speed network. The virtual time function equation is shown below:

$$V_{WF^2Q+}(t+\lambda) = \max\{V_{WF^2Q+}(t) + W(t,t+\tau), \min_{i \leftarrow B(t)}\{S_i^{h_i(t)}\}\} \qquad (2.5)$$

In the equation 2.5, $W(t,t+\tau)$ is the total amount of service provided by the server during the period t to $t+\tau$, $B(t)$ is the set of sessions backlogged at time t, $S_i^{h_i(t)}$ is the VST of the packet at the head of the session i. To simplify the implementation of the algorithm, the definitions of the VST and VFT are modified. The old definitions need to maintain the VST and VFT on a per packet basis. Each packet on the queue maintains a set of F_i^k and S_i^k. This will cost a lot of memory usage on the small packet system. In the new definition, each session i only keeps a set of F_i and S_i. The equation of VST and VFT in WF^{2}Q+

are shown below:

$$S_i = F_i \quad \text{when } Q_i(a_i^k -) \neq 0$$

$$S_i = \max\{F_i, V(a_i^k)\} \quad \text{when } Q_i(a_i^k -) = 0$$

$$F_i = S_i + \frac{L_i^k}{\phi_i} \tag{2.6}$$

Where $Q_i(a_i^k -)$ is the queue size of session i just before time a_i^k. By using this approach, the worse case complexity is claim to be O(log N) which improved from WF^2Q with O(N) where N is the maximum number of connections that can share an output link.

2.1.5 Self-Clocked Fair Queuing

Most of the work-conserving disciplines emulate a reference FFQ server that requires high computational cost. SCFQ [11] is simplified in implementing the reference server. It is a simplified version of WFQ and WF^2Q. The author observed the virtual time function *v(t)* and summarized two properties. First, FFQ system is a piecewise linear function with the slope at any point of time *t* inversely proportional to the sum of the service shares of active sessions. Second, when some sessions become backlogged or cease to be backlogged in the FFQ system, the slope of *v(t)* changes. This will forms the breakpoint of the piecewise linear form of the slope. SCFQ tracks the active session and finds the breakpoint of the piecewise linear line *v(t)*, the algorithm uses this concept to evaluate the virtual time function and hence simplifies the implementation. This algorithm is simple but the breakpoint in *v(t)* changes infrequently and it can happen many times in a short period of time (during a single packet transmission time) so it is not very accurate in approximating the GPS system. Because it uses a self-defined approximate virtual time clock, it is called a self-clocked fair queuing. The SCFQ is based on the following steps:

Each arrival packet p_k^i is tagged with a service tag $\hat{F}_k^i$ before it places in the queue. The packets in the queue are picked up service in increasing order of the associated service tags. For each session *k,* the service tags of the arriving packets are iteratively computed for the VST:

$$\hat{F}_k^i = \frac{L_k^i}{r_k} + \max(\hat{F}_k^{i-1}, \hat{v}(a_k^i)) \tag{2.7}$$

Where $i = 1,2,.....,k \leftarrow K$ and $\hat{F}_k^0 = 0$, L_k^i is the packet length of p_k^i, r_k is the service rate of flow *k* and $\hat{v}(t)$ is the approximated virtual time function.

$\hat{v}(t)$ is the virtual time of the system at time *t*, it is defined equal to the service tag of the packet receiving service at that time. More specifically,

$$\hat{v}(t) \cong \hat{F}_l^j \quad \hat{s}_l^j < t \leq \hat{d}_l^j \tag{2.8}$$

where $\hat{s}_l^j$ is the start service time and $\hat{d}_l^j$ is the finish service time of packet p_k^i.

Once a busy period is over, when the server is free and no more packets are found in the queue, the algorithm reinitializes the $\hat{v}(t)$ and the packet counts *i* to zero for each session *k*.

Actually it has shown that the difference between $\hat{v}(t)$ and *v(t)* is not necessarily bounded and may approach infinity. Because $\hat{v}(t)$ is an approximated virtual time function, it is equal to the service tag of the most recent packet transmitted prior to *t*, instead of the service tag of the packet receiving service at time *t*. Although SCFQ is a good choice for lowering the computational requirement of the algorithm, it is not fair compare to the original GPS system.

Figure 2-6 is the packet output pattern of the SCFQ system with the packet arrival pattern shown in figure 2-2. Since SCFQ uses the finish time of the packet in service as the current virtual time value, $\hat{v}(1) = \hat{v}(2) = F_2^1$. At t=2, the P_1^2 arrives with VFT $\hat{F}_1^2 = \frac{1}{0.5} + \max(2,20) = 22$ (substitute packet size $L_1^2 = 1$, service rate $r_1 = 0.5$, Virtual Time $\hat{v}(2) = 20$ and $\hat{F}_1^1 = 2$). Among all head packet in different flows, P_1^2 gets the largest finish number so it starts the service when $P_i^1, i = 2,3,.....11$ finished servicing. P_1^2 leaves at t =12.

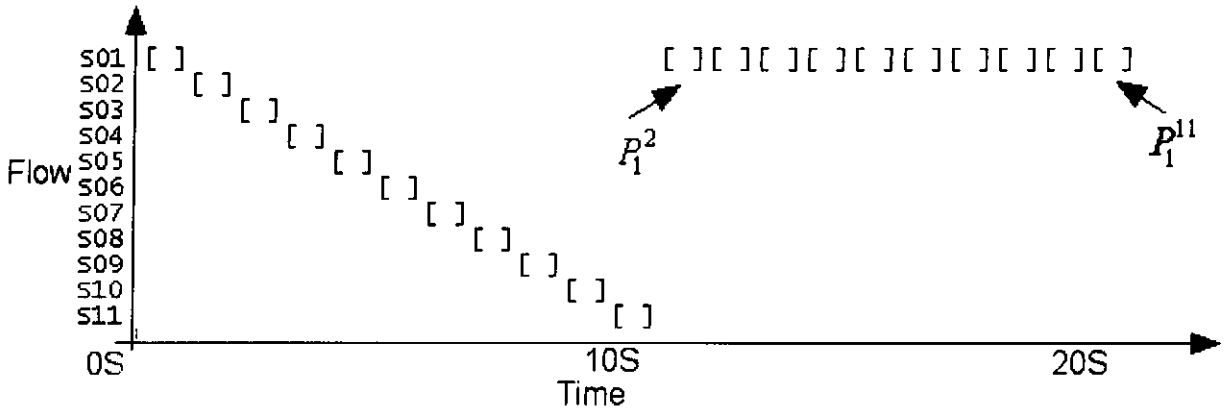

Figure 2-6 The packet output pattern of the SCFQ system

2.1.6 Fair Queuing

In Fair Queuing [1], if there are N flows sharing a link with bandwidth BW, then each flow can get BW / N. If one of the flow uses less than the allotted share, the surplus will equally distributed among the rest of the flows. Fair Queuing emulates the bit-by-bit round robin (BR). Each packet gets into the queue given a finish number. The service round of BR is

$$\frac{\partial R}{\partial t} = \frac{\mu}{N_{ac}(t)} \tag{2.9}$$

where *R(t)* is the number of round made, μ is the line speed of the link and $N_{ac}(t)$ is the number of active conversations. So a packet of size *P* whose first bit gets serviced at time t_0 will have its last bit serviced *P* rounds later at time *t* such that $T(t) = R(t_0) + P$.

The finish time of BR is

$$F_i^k = S_i^k + P_i^k \tag{2.10}$$

where S_i^k is the start service time and F_i^k is the finish service time of packet i^{th} of session k^{th}. The start time of BR is

$$S_i^k = \max(F_{i-1}^k, R(t_i^k)) \tag{2.11}$$

$R(t_i^k)$ is a continuous monotonically increasing function whenever there are bits at the gateway. In order to emulate the bit-by-bit round-robin algorithm, the quantity F_i^k defines the sending order of the packets. Whenever a packet finishes transmission, the packet with the smallest value of F_i^k will be sent next. The fair queuing allocates less delay to a user who utilizes less than their fair share of bandwidth. It allocates delay independents of the bandwidth allocation by introducing two parameters δ and bid B_i^k. The formula to calculate the bid in fair queuing become:

$$B_i^k = P_i^k + \max(F_{i-1}^k, R(t_i^k) - \delta) \tag{2.12}$$

If δ is small, then B_i^k is likely independent of the previous history of flow *k*. If δ is large, then B_i^k is likely to depend on the previous finish time of the packet. The Fair Queuing provides a mechanism on the delay manipulation of a specific flow.

2.1.7 Virtual Clock

Virtual Clock (VC) [19] emulates the static Time Division Multiplexing (TDM) system. Each packet gets into the queue given a virtual transmission time based on the arrival pattern and the reservation of the connection. The packet should be transmitted if the virtual transmission time is passed. The virtual transmission time is independent of the behaviors of other connections. The delay of a packet depends on the entire arrival history of the connection itself.

In VC, the state variable is auxiliary VC ($auxVC$, virtual transmission time) and it is computed by:

$$auxVC_{i,j}^{k} \leftarrow \max\{a_{i,j}^{k-1}, auxVC_{i,j}^{k-1}\} + Vtick_{i,j} \tag{2.13}$$

The i, j and k are the server number, connection number and packet number respectively. $Vtick_{i,j}$ is the average packet inter-arrival time for connection j. $a_{i,j}^{k}$ is the real time for the packet k of connection j leaving server i. If the virtual clock is larger than the real time, that means the flow has been sending faster than the specified rate. If $VC >> a_{i,j}^{k}$ then it slows down the specific flow. If the aux virtual clock < real time then the $VC = a_{i,j}^{k}$, that's why $\max\{a_{i,j}^{k}, auxVC_{i,j}^{k}\}$ is in the equation.

The Vtick is calculated by:

$$Vtick_{i} = \frac{1}{AR_{i}} \tag{2.14}$$

AR_i is the average transmission rate in packet per second.

This algorithm uses simple multiplexing and it can provide weighted sharing of the node by using different AR_i for different flows. The control mechanism is limited and it does not

use GPS as reference system. It is known to be not fair but it builds a simple statistical multiplexing concept for node sharing.

2.1.8 Stochastic Fairness Queuing

Stochastic Fairness Queuing (SFQ) [22] is proposed by Paul E. McKenney in 1990. It is relatively simple by comparing to strict fairness queuing algorithms. The queues are serviced in strict round-robin order and a simple hash function is used to map source-destination address pair into a fixed set of queues. If the number of queues is large compared to the number of conversations, each conversation will be assigned to its own queue with high probability. If two conversations collide, they will continue to receive less than its share of bandwidth. This situation is alleviated by periodically perturbing the hash function so the conversations that collide during one time period are very unlikely to collide during the next. The author aimed to use the algorithm in the high-speed network and he supposes the traditional per-flow first come first serve queuing mechanism needs to consume much higher computational power to distribute incoming packets to the corresponding queues. The traditional approach needs to use numerous memory references to map the packet to the incoming and the outgoing address pair.

In figure 2-7, the operating mechanism of the Stochastic Fairness Queuing is almost the same as the traditional per-flow first come first serve queuing. Only the packet distribution method is modified. The packets are distributed by the hash function below:

$$hash = ROL(src, seq) + dst \tag{2.15}$$

where "ROL" is the circular rotate-left function implemented as a single instruction on many computers, "src" is the Internet Protocol source address, "seq" is a sequence number in the range from 0 to 31 and it is used to perturb the hash function. "dst" is the IP destination address.

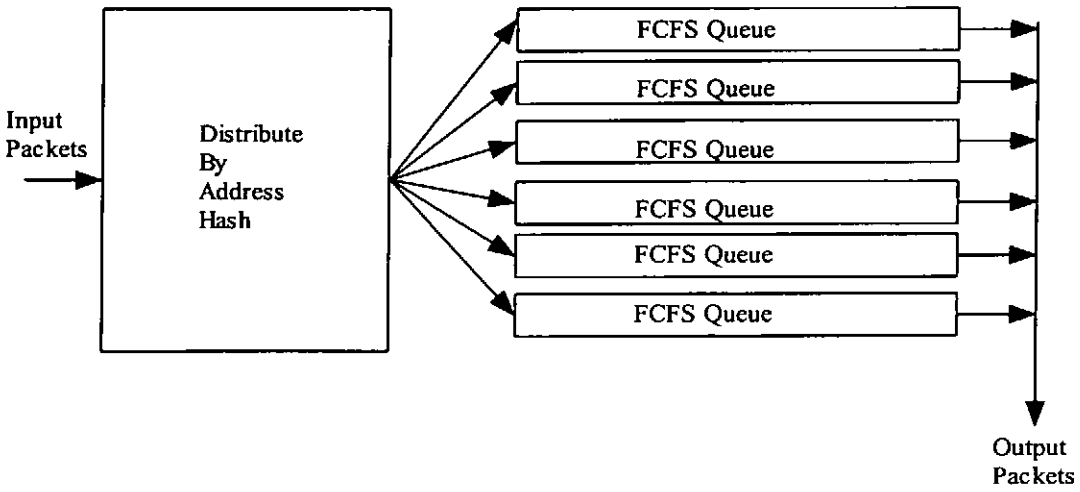

Figure 2-7 Operating mechanism of the Stochastic Fairness Queuing

Although SFQ is less fairer than the other fair queuing algorithm such as VC and FQ. The computation requirement is much lowered. The fairness of SFQ can be improved by increasing the number of queues or increasing the size of buffers. The fairness can also be increased by shorten the hash switch interval. It is an alternative choice from the other fair queuing algorithms.

2.1.9 Delay Earliest-Due-Date

Delay Earliest-Due-Date (Delay-EDD) [13], an improved version of Earliest-Due-Date-First (EDD or EDF). Each packet in the EDD from a periodic traffic stream is assigned an Expected Deadline ($ExD_{i,j}^{k}$) and the packets are sent in order of the increasing deadlines. The deadline of a packet is the sum of its arrival time and the period of traffic stream. In Delay-EDD, the server negotiates a service contract with each source. The contract states that if a source obeys the promised traffic specification (peak and average sending rate), then the server will provide a delay bound. The expected deadline is calculated by the formula below:

$$ExD_{i,j}^{k} \leftarrow \max\{a_{i,j}^{k} + d_{i,j}, ExD_{i,j}^{k} + X_{\min j}\} \qquad (2.16)$$

$X_{\min j}$ is the minimum packet inter-arrival time for connection j. $d_{i,j}$ is the local delay bound assigned to connection j of server i at the connection establishment time. This algorithm can provide a rough controlling mechanism for packet sending order.

2.2 NON-WORK-CONSERVING DISCIPLINES

2.2.1 Round Robin and Deficit Round Robin

Round Robin is one of the old fashioned scheduling algorithm to share resource. It is a frame based algorithm which the computational complexity is O(1) without any timestamp calculation. These algorithms yield delay bounds may grow linearly with the number of sessions sharing the outgoing link. In a real computer network, many data link layers support variable packet size and the old fashioned round robin become unfair in the variable packet size environment. DRR [32] was proposed by M. Shreedhar and George Varghese to solve this problem, they introduced the quantum concept. In DRR, each flow keeps a deficit counter. The deficit counter increment Y at each round. Y was called the quantum, which is a credit to de-queue the packet. At the de-queuing moment, each flow can de-queue the packet in the queue until all the deficit counter credit has been used up. Large packet consumes more credit and small packet consumes less credit. This algorithm claims to be fair in the variable packet size environment. The number of bytes send in DRR is calculated by:

$$bytes_{i,k} = Quantum_i + DeficitCounter_{i,k-1} - DeficitCounter_{i,k} \tag{2.17}$$

The total number of bytes sent up to round K is calculated by:

$$sent_{i,k} = K \times Quantum_i - DeficitCounter_{i,k} \tag{2.18}$$

i is the flow number, K is the number of Round in the DRR, $bytes_{i,k}$ is the number of bytes send in flow i at K^{th} round. $sent_{i,k}$ is the total number of byte sent up to round K.

2.2.2 Hierarchical Round Robin

Hierarchical Round Robin (HRR) [5] extends the traditional round robin. It was proposed to provide the flows that sharing a node to get several service levels, each level provides round-robin service to a fixed number of slots. A channel is allocated some number of service slots at a selected level and the server cycles through the slots at each level. The topmost level gets the shortest frame length and it is used to serve connections that are allocated the highest service rate. Since a server always completes one round through its slots once every frame time, it can provide a maximum delay bound to the flows allocated to a specific level.

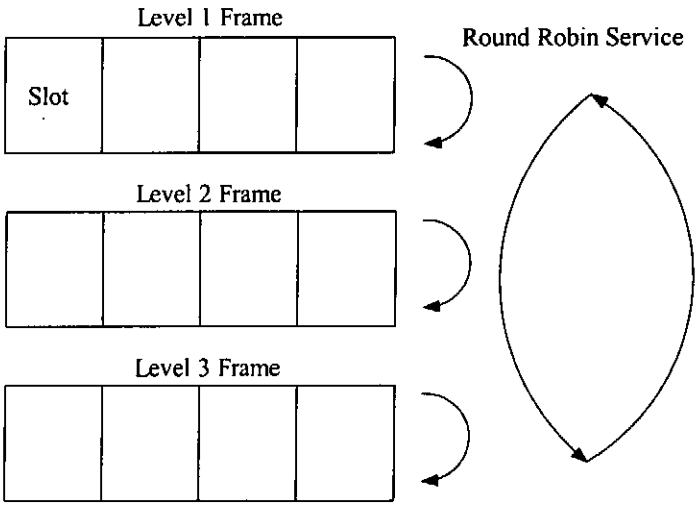

Figure 2-8 Frames of Hierarchical Round Robin

2.2.3 Stop and Go

Stop-and-go [33] service aims to preserve the "smoothness" of traffic as it traverses the network. It composed by two parts, a packet admission policy and a framing strategy. Time is divided into frames and each frame is T second long, only packet that arrived in the previous frame time is eligible to send. Because of this delay, there may be packet in the queue when the output link is idle. Stop-and-go is a non-work-conserving policy. It defines departing and arriving frame for each flow. At each hop, the arriving frame of each

incoming link is mapped to the departing frame of the output link by introducing a constant delay θ , where $0 \le \theta < T$. In the packet admission policy, the algorithm claims to provide (r_k, T) smoothness where r_k is the bandwidth allocated to connection k and T is the frame length. In each frame length the connection k cannot receive more than $r_k \times T$ bits services. This service discipline can provide minimum and maximum delay from the source to the destination. The length of the frame time affects the delay and delay-jitter bound, stop-and-go uses multiple frame size.

2.2.4 Random Early Drop And Flow Random Early Drop

RED [3, 30] was proposed by Sally Floyd and Van Jacobson, it is used to replace the simple tail drop. RED monitors the average queue size by using the previous history of the queue and probabilistically drops packets when the queue exceeds certain thresholds. By dropping packets before the buffer is full, RED provides an early signal to the end systems to back off. However, RED cannot ensure fairness among competing flows. Flow Random Early Drop (FRED) [38] improves the fairness of bandwidth allocation in RED by maintaining state for all backlogged flows. FRED drops packets from flows that have had many packets dropped in the past. The packets are dropped if the flows that have queues larger than the average queue lengths.

The Equation, 2-19, 2-20 and 2-21 are governing the dropping probability of RED.

When the RED scheduler receives a packet, it will perform the following task:

The intermediate dropping probability of RED is calculated by

$$P_b = \frac{\max_p (avg - \min_{th})}{\max_{th} - \min_{th}} \tag{2.19}$$

P_b is the intermediate dropping probability, *avg* is the estimate queue size, $\max_{th}$ is the maximum threshold for queue, $\min_{th}$ is the minimum threshold for the queue and $\max_p$ is the maximum value for P_b.

The *avg* varies from $\min_{th}$ to $\max_{th}$ that imply the P_b is varies from 0 to $\max_p$.

The dropping probability of RED is calculated by:

$$P_a = \frac{P_b}{1 - count \times p_b} \tag{2.20}$$

P_a is the final dropping probability and count is used to cater for the past performance of the queue. The count will be minus one if the estimate queue size is less than the $\min_{th}$ and the count will be reset if the estimate queue size is greater than $\max_{th}$. This mechanism ensures the past performance of the queue will affect the dropping probability. If the flows always en-queue packet, the dropping probability will be low. If the flows always drop packet, the probability will be high. The following are the decision making section of the RED algorithm:

If the queue is nonempty then the estimated queue size is computed below:

$$avg = (1 - w_q) \times avg + w_q \times q \tag{2.21}$$

avg is the estimate queue size, w_q is the queue weight, q is the actual queue size at the packet en-queuing moment. The estimate queue size is based on both the past estimate queue size and the current queue size. If w_q is large, the estimate queue size will depend more on the current queue size and vice versa.

The following are the three cases that will happen depending on the estimate queue size:

(i) If the estimate queue size (*avg*) is smaller than the $\min_{th}$ then the packet is en-queue and count = count – 1.

(ii) If the estimated queue size (*avg*) is between $\max_{th}$ and $\min_{th}$ then the packet is dropping with probability P_a.

(iii)If the estimate queue size (*avg*) is greater than the $\max_{th}$ then the packet is definitely drop and the *count* is reset to zero.

Combine the equation 2.19 and the above three conditions. The RED can from a discontinuous dropping probability function which shown below.

FRED improves RED in the fairness among different flows by modifying the dropping probability equation.

$$P_a^i = \frac{P_b^i}{1 - count^i \times p_b^i} \tag{2.22}$$

P_a^i, P_b^i and $count^i$ is introduced. Each flow/queue gets their dropping probabilities and past performance counter. This ensures the control mechanism operating independently for each flow.

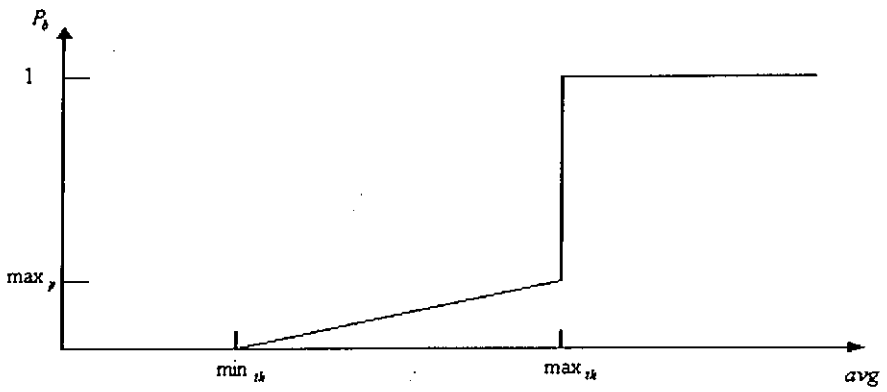

Figure 2-9 Dropping probability function for RED

The performance and behavior of RED are extremely sensitive to the parameters $\max_p, \max_{th}$, $\min_{th}$ and w_q. A mis-parameterized RED queue can fail to control the transmission rates of low priority sources, allowing them to fill the queue with packets and cause high priority packets to drop. Another problem is that RED's congestion detection mechanism is based on the calculation of an average queue length. Since the instantaneous queue length can change quite dramatically over short time intervals, queue overflow and loss of priority packets can occur before increasing in average queue length can trigger RED congestion control mechanisms. In order to address the shortcomings of RED active queue management algorithm, recently improvement labeled BLUE [37] has been proposed. These improvements can potentially reduce the amount of packet loss. RED and FRED keeps the overall throughput high while maintaining a small average queue length and tolerate transient congestion without causing global synchronization. They require lower computational power than the queuing algorithms, which need to maintain the virtual clock and emulate the fluid queuing system. They drop the packet early and good for notifying the end host in the transport layer to reduce the sending rate and the window size of the TCP protocol. It provides the congestion control and further lowers the computational requirement of the hardware in the queuing node.

2.2.5 RED with In and Out (RIO)

RIO [18] combines the capabilities of the RED algorithm with drop precedence and this algorithm is using in Differentiated Services (Diffserv) Internet resources reservation Model. This combination provides preferential traffic handling for higher priority packets. It can selectively discard lower priority traffic when the interface begins to get congested. It can also provide differentiated performance characteristics for different services. In addition, RIO operates in both the input and the output queues. It uses twin RED

algorithms for dropping packets, one for "IN" and one for "Out". By tuning two sets of parameters ($\max_{p\_in}, \max_{th\_in}, \min_{th\_in}, \max_{p\_out}, \max_{th\_out}, \min_{th\_out}$ and w_q), RIO can discriminate against "Out" packets in times of congestion. It differs from other congestion management techniques such as queuing strategies because it attempts to anticipate and avoid congestion rather than controlling congestion once it occurs.

The algorithm of RIO is same as the RED but it keeps two sets of parameters. In equation 2.19, the input queue uses the average queue size for the In packet to compute the dropping probability and the output queue uses the average queue size for all (In and Out) arriving packets to compute the probability. Figure 2-10 shows how RIO works, the drop probability on the y-axis increases as average queue length increases along the x-axis. RIO has one curve for traffic that is conformant with the boundaries for packet in and the other is for packet out.

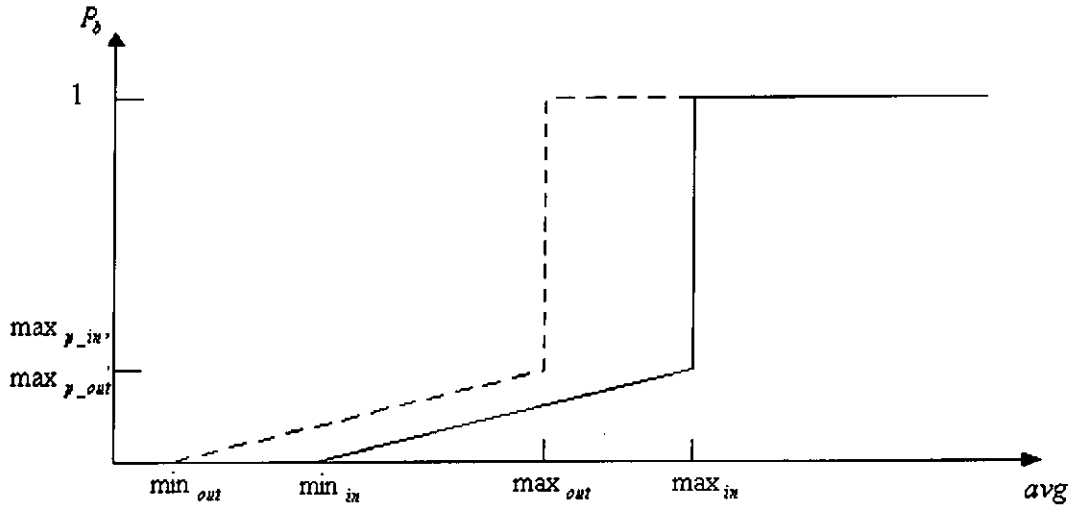

Figure 2-10 RED with In and Out drop probabilities

Same as RED, during the normal operation phase ($avg < min_{out}$ and $avg < min_{in}$), there is no packet drop. When the average queue size is between the two thresholds, it goes into the congestion phase and the packets start to drop. When the average queue size is above max_{th}, the node sustains large sized queue and it drops every arriving packet hoping to

maintain a short queue size. RIO uses the Type of Service (ToS) field marked by the edge routers to determine how it treats different types of traffic. It provides separate thresholds and weights for different packet classes; allow providing different QoS for different traffic. Standard traffic dropped more frequently than premium traffic during periods of congestion.

Just like RED, the efficiency of RIO depends on some extent of correct parameter choices. One interesting property of RIO is that it does not change the order of "in" and "out" packets. If a TCP connection is sending packets through a profile meter, and some packets are being marked "in" while others are marked "out," those packets will receive different drop probabilities in the router queues, but they will be delivered to the receiver in the same order as they were sent.

2.2.6 Jitter Earliest-Due-Date

Jitter Earliest-Due-Date (Jitter-EDD) [13] extends Delay-EDD, which described previously in this chapter. Jitter-EDD provides delay-jitter bound which is a bound on minimum and maximum delay. The expected deadline calculation, which ensures the maximum delay bound, is exactly the same as Delay-EDD. In figure 2-11, the *PreAhead* time stamp labeling concept is added to provide the minimum delay bound. Each packet leaves the server will label a *PreAhead* timestamp. It is the difference between packet deadline and actual finishing time. If the packet is transmitting too fast, the regulator at the entrance of the next hop will hold the packet for *PreAhead* seconds that specified in the time stamp before it is eligible to be scheduled. This mechanism provides the minimum delay bound for the Jitter-EDD.

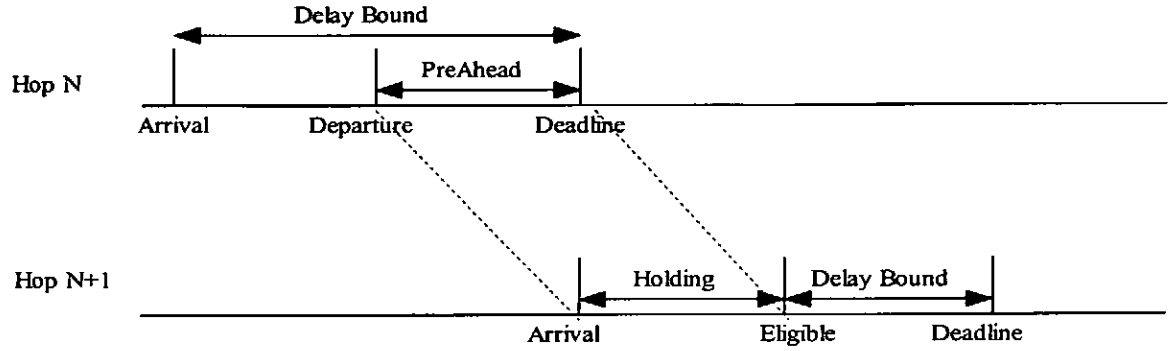

Figure 2-11 Packet Service in Jitter-EDD

2.3 CONCLUSION AND COMPARISON

Different packet scheduling algorithms provide different desired properties. The general properties of the algorithms are summarized in table 2-1.

	WF^2Q+	WF^2Q	WFQ	FQ	SCFQ	VC	SFQ	D-EDD
Work Conserving	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Throughput Guarantee	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Delay Guarantee	Yes	Yes	Yes	No	Yes	No	Yes	Yes
Jitter Guarantee	No	No	No	No	No	No	No	No
Constant Buffer	No	No	No	No	No	No	No	No
Protection	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Table 2-1a Network properties comparison among different queuing algorithms

	J-EDD	S&G	RR	DRR	HRR	RED	FRED	RIO
Work Conserving	No	No	No	No	No	No	No	No
Throughput Guarantee	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Delay Guarantee	Yes	Yes	Yes	Yes	Yes	No	No	No
Jitter Guarantee	Yes	Yes	No	No	No	No	No	No
Constant Buffer	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Protection	Yes	No	Yes	Yes	Yes	No	Yes	No

Table 2-2b Network properties comparison among different queuing algorithms

From the table 2-1, it can be seen that most of the algorithms support delay guarantee, but only two of the algorithms (J-EDD and S&G) support the jitter guarantee. The jitter guarantee provides the limit for the maximum delay bound and the minimum delay bound, but the delay guarantee only provides the limit for the maximum delay bound. The

fairness among the work-conserving discipline can be generally summarized as follow:

GPS > WF^2Q+ = WF^2Q > WFQ > FQ > SCFQ > VC > SFQ

The fluid system GPS is the ideal system and it is the fairest algorithm. Most of the other algorithms are approximating the GPS by virtual clock. WF^2Q+ and WF^2Q are same in the fairness. They are the most accurate algorithms that approximate the GPS system. The complexity of the algorithms can be summarized as follow:

WF^2Q+	WF^2Q	WFQ	FQ	SCFQ	VC	DRR	RR	SFQ
O(logN)	O(N)	O(N)	O(logN)	O(logN)	O(logN)	O(1)	O(1)	O(1)

Table 2-3 Complexity among different Algorithm

In table 2-2, N is the maximum number of connections that can share an output link. Although WF^2Q+ and WF^2Q get the same fairness, they have different complexities. WF^2Q+ is the improved version of WF^2Q. The computational complexity is much lowered. The complexity for packet selection in RR, DRR and SFQ are O(1). The Round Robin Pointer already determines which packet is going to leave in the coming round.

Different algorithms get different desired network properties, different fairness and different computation complexities. The choice of scheduling algorithm in the sharing node depends on the application and the traffic model that will be used in that specific network.

CHAPTER 3 NETWORK PERFORMANCE EVALUATION AND FAIRNESS MODELLING

In Queuing Algorithms, delay, delay jitter, queue size, throughput and packet drops are used as the parameters to compare the performance. Although these parameters are closely related to each other, they reflect different network properties for different network application requirements. Up to now, there is no public accepted index to measure the fairness among different flows. The max-min fairness criterion [8, 9, 17] states that an allocation is fair if the following conditions are matched: (1) no user receives more than its request, (2) no other allocation scheme satisfying condition 1 has a higher minimum allocation, and (3) condition 2 remains recursively true as we remove the minimal user and reduce the total resource accordingly. These rules give an idea in theoretical concept on fairness but it cannot quantify fairness numerically and compare them easily. Another fairness definition was introduced by S. Golestani [11]. It measures the fairness by the services offered to connections. If the difference in services offered to any two connections that are continuously backlogged in the system is zero, then the scheduler is perfectly fair. Up to now, it is the mostly accepted fairness definition but it is not easy to measure the services received in the real networking environment without changing the firmware in the equipment. It is hard to measure the received service and quantify the fairness. In this chapter we discuss the network evaluation techniques for queuing algorithms comparison.

3.1 BASIC STATISTICAL EVALUATION METHOD

This is the traditional mathematical method that uses to analyze data. Large pool of samples is required. Mean (equation 3.1), standard deviation (equation 3.2) and correlation (equation 3.3) can be used as the analyzing tools.

Equation of Sample Mean:

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} \tag{3.1}$$

Equation of Standard Deviation:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n-1}} \tag{3.2}$$

Equation of Correlation:

$$r = \frac{\sum_{i=1}^{n} [(x_i - \bar{x}) \bullet (y_i - \bar{y})]}{\sqrt{\sum_{i=1}^{n} (x_i - \bar{x})^2 \bullet \sum_{i=1}^{n} (y_i - \bar{y})^2}}$$

$$\text{where } \begin{matrix} -1 \le r \le 1 \\ 0 \le r^2 \le 1 \end{matrix} \tag{3.3}$$

Central Limit Theorem states that no matter the distribution is, if the sample size is reasonably large then it is confident to say that the distribution of the sample mean will be approximately normal. In statistic $N \ge 30$ is generally considered as large sample size. It is a tool to measure the accuracy of the estimated sample mean $\bar{x}$. In Central Limit Theorem, $E(\bar{x}) = E(x) = \mu$ and $SD(\bar{x}) = \frac{\sigma}{\sqrt{n}}$.

Statistical method is dimensionless (not depends on units) and independent of scale. It can be applied to a continuous function; slightly changes of x_i will affect the whole result (robust). These properties allow the method to apply in all mathematics models. The only

deficiency is population size dependent. Little sample size cannot achieve high accuracy. Central Limit Theorem can be used so the sample size requirement can be lowered.

3.2 NETWORK POWER

The network power [4, 10] is defined by one of the renowned Internet researcher, Leonard Kleinrock. It uses throughput and delay to find the efficiency of the networking system. The network power can be calculated by the following equation 3.4.

$$P(\lambda) = \frac{\gamma(\lambda)}{T(\lambda)} \tag{3.4}$$

The network power $P(\lambda)$ is expressed as a ratio of throughput $\gamma(\lambda)$ to delay $T(\lambda)$. It gives a good indication of network efficiency and the optimal throughput of a network system. In a communications system, packet processing is divided into three stages: packets arrive, receive service and depart. Let the input load applied to the system be λ ($0 \leq \lambda \leq 1$). The following fundamental system performance characteristics are defined:

- Throughput, $\gamma = \gamma(\lambda)$. The rate at which packets arrive at the receiving host.
- Mean delay, $T = T(\lambda)$. The average time spent by a packet while it resides in the system.

It is often the case that a system cannot reach its highest capacity, thus not yielding a 100% throughput. This means that the system reaches its maximum throughput at input loads where $\lambda < 1$. Exceeding that point of input load drives the system to overloaded situations, delay and queue length will grow to infinity.

$$\gamma = \gamma(\lambda) \text{ when } \lambda < \lambda_s$$

$$\gamma = \gamma_{\max} \text{ when } \lambda \geq \lambda_s \tag{3.5}$$

Equation 3-5 shows a throughput-input load function where λ_s represents the maximum input load of the stable system, and γ_{max} the maximum throughput achieved. Thus, in order

for a system to be stable we require that $\lambda < \lambda_s$. The network system model shown in figure 3-1 refers to an ideal no-loss system.

In a non-ideal system, throughput normally degrades when $\lambda > \lambda_s$, unless some types of flow control are implemented. As λ approaches λ_s, the system becomes saturated. It is then apparent that λ_s is a critical value. It is a key point in a throughput analysis. Figure 3-2 shows a mean delay profile of a network system. The minimum mean delay T_{min}, measured in a system is found when $\lambda \rightarrow 0^+$. As $\lambda \rightarrow \lambda_{max}$ the delay goes to infinity.

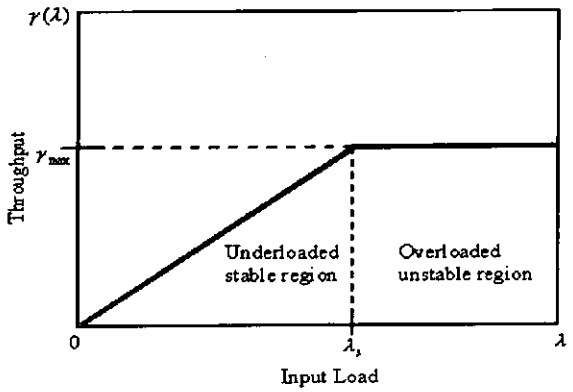

Figure 3-1 The ideal throughput characterisitc of a network system

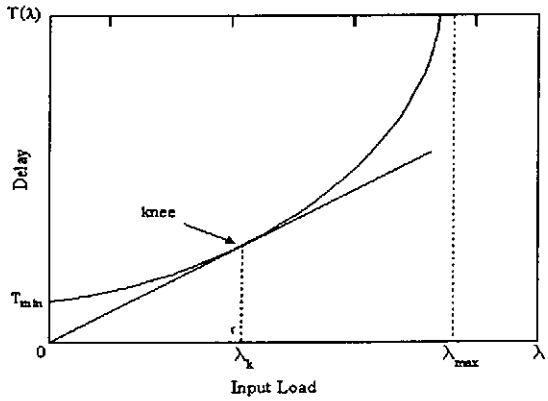

Figure 3-2 The delay characteristic of a network system

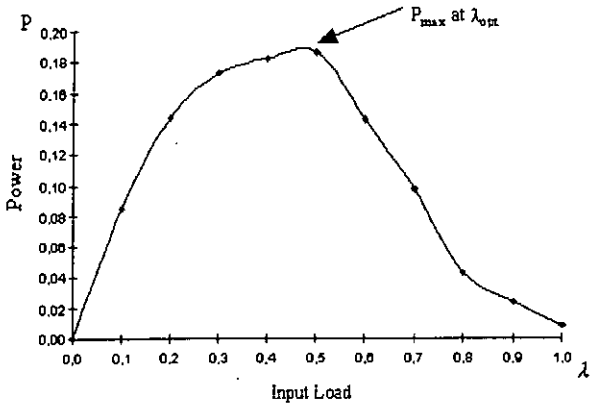

Figure 3-3 The network power of a system

By combining figure 3-1 and figure 3-2, figure 3-3 is formed. The maximum power P_{max} identifies the operating point where a network delivers its best performance when the input

load is at λ_{opt}. In figure 3-2, the load at which power is maximized namely λ_k. It can be found by taking the tangent to $T(\lambda)$ from the origin. This tangency point is referred to the "knee" of the curve. Increasing input load affect the throughput to increase but delay becomes higher as the system tends to saturate. Tradeoff has to be made between high throughput and low delay. The notion of power proves to be useful in addressing this issue. It combines the two network parameters and enables an easy measure for network system performance comparison.

3.3 JAIN'S FAIRNESS EVALUATION

Raj Jain [29] proposes a function to quantitative measure the fairness of a system. Assuming a system which allocates resources to n contending users such that the n^{th} user receives allocation X_n where $n \geq 0$. Applying this idea to the networking system, given a set of flow throughputs $(x_1, x_2, x_3 \ldots, x_n)$. The fairness index to the network system is:

$$f(x_1, x_2, x_3 \ldots, x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \times \sum_{i=1}^{n} x_i^2} \tag{3.6}$$

The index gets the following properties:

(i) Population Size Independence: This index can apply to any number of users, finite or infinite. It can apply to two users sharing a resource,

(ii) Dimensionless: It does not depend on the dimension of X_n (bytes, meters, delay,..), it can apply to any resource sharing or allocating system,

(iii)Independent of scale (kilo-, nano-, tera-, pico-,...) and bounded between 0 and 1: the quantified value is easy to understand, interpret and compare. The closer to one

the fairer of the system, which can easily compare fairness between different network systems and

(iv)Continuous: The slight changes of X_n will affect the fairness index.

The quantitative measure of fairness index $f(x_n)$ measures the equality of the system resources allocation x. If all users get the same amount of resources ($X_1 = X_2 = X_3 \ldots = X_n$) then $f(x_n) = 1$ and the system is 100% fair. As the difference between the shared resources amount the user increases, the fairness will decrease. A user's perception of fairness can be found by rewriting the proposed fairness index to:

$$f(x) = \frac{1}{n \times x_f} \times \sum_{i=1}^{n} x_i \tag{3.7}$$

x_f is the fair allocation mark which computed as follows:

$$X_f = \frac{\sum_{i=1}^{n} x_i^2}{\sum_{i=1}^{n} x_i} \tag{3.8}$$

The fairness index of individual user is $\frac{x_i}{x_f}$. It can be described that the i^{th} user received $\frac{x_i}{x_f}$ fairness. By summing the fairness of all individual users $\frac{1}{n}\sum_{i=1}^{n}\frac{x_i}{x_f}$, it will become equation 3.6, the overall fairness index of the system. If $x_i < x_f$ then the system is unfair to user i and is a discriminated user. If $x_i \geq x_f$ then the system is favored to user i and is a favored user.

Consider an example with 10Mbps shared among 30 flows where 3 flows receive 10 times bandwidth of the other 27 flows. That means the 3 higher bandwidth flows get 1.754Mbps

and the other 27 flows get 0.1754Mbps. The fairness allocation mark is calculated to be 1.0065Mbps. The 3 higher bandwidth flows received 1.754Mbps > 1.0065Mbps and they are the favor flows and the 27 lower bandwidth flows are the discriminated flows which received 0.1754Mbps < 1.0065Mbps. The overall fairness index is 0.09872.

3.4 WORST CASE FAIR INDEX (WFI)

To quantify the discrepancy between the services provided by packet schedule discipline and the fluid GPS discipline, the WFI is proposed by the renowned packet queuing researcher Hui Zhang [2]. The definition of WFI is shown below:

A service discipline *s* is called worst-case fair for session *i* if for any time τ, the delay of a packet arriving at τ is bounded by

$$d_i^k - a_i^k \leq \frac{Q_i(a_i^k)}{r_i} + A_{i,s} \tag{3.9}$$

Where r_i is the rate guaranteed to session *i*, $Q_i(\tau)$ is the number of bits in the session queue at time τ(including the packet that arrives at time τ). a_i^k and d_i^k are the arrival and departure times of the k^{th} packet of session *i* respectively. The arrival time defined here is the time that the first bit of the arrival packet comes to the server and the departure time defined here is the time that the last bit of the departure packet leaves the server. $A_{i,s}$ is the maximum time for a packet coming to an empty queue needs to wait before receiving its guaranteed service rate. It is also called the Worst-case Fair Index (WFI) for session *i* at server *s*. The dimension of the WFI is time and the unit is mini-second. WFI is a constant which independent of queues at the other sessions sharing the multiplexer. A service discipline can only be called worst-case fair if it is worst-case fair for all the sessions. In order to compare and interpret the fair index, it defines the ideal GPS fluid system with $A_{i,s}$

=0. The algorithm with the fairness index closer to 0 will be fairer and considered more accurate in approximating the GPS system. WFI is measured in absolute time and not suitable for comparing different fair index of sessions with different guaranteed rates.

In order to compare the index with different guaranteed rate the normalized worst-case fair index is introduced:

$$a_{i,s} = \frac{r_i \times A_{i,s}}{r} \tag{3.10}$$

Where r is the total guaranteed bandwidth to all of the flows.

For a server that is worst-case fair, the normalized worst-case fair index is:

$$a_s = \max\{a_{i,s}\} \tag{3.11}$$

3.5 LATENCY RATE (*L-R*) SERVER MODEL

The Latency Rate Server Model was introduced by Dimitrios Stiliadis and Anujan Varma [36] to analyze traffic scheduling algorithms. The behavior of an LR scheduler is determined by two parameters, the latency and the allocated rate. This model only studies the class of scheduling algorithms that capable of providing bandwidth guarantees to individual session. The model can be used to derive the deterministic end-to-end delay guarantees that are independent of the behavior of other sessions. In Cruz's work [27] individual schedulers are studied in isolation and the session delay is analyzed by accumulation method. In this model the behavior of the chain of schedulers on the end-to-end path is studied on a whole. This model estimates the latency parameters for individual schedulers considering the internal structure. This model also derives the internal burstiness and buffer requirements of individual sessions.

In the implementation complexity estimation, this model considering three factors:

(i) Timestamp calculation,

(ii) Packet insertion in the queue and

(iii) Selection of the packet for forwarding

The maximum backlog bit $Q_i^{(S_k)}(t)$ in the k^{th} node of session i is bounded by

$$Q_i^{(S_k)}(t) \le \sigma_i + \rho_i \sum_{j=1}^{k} \Theta_i^{(S_j)} \tag{3.12}$$

The maximum delay D_i of session i in a network is

$$D_i \le \frac{\sigma_i}{\rho_i} + \sum_{j=1}^{k} \Theta_i^{(S_j)} \tag{3.13}$$

σ_i is the burstiness in bit and ρ_i is the average arrival rate of session i in bit/second. $\Theta_i^{(S_j)}$ is the worst case delay of the j^{th} scheduler on the path of the session. Worst case delay is defined as the delay by the first packet of a busy period in a session.

For the fairness of the *LR*-Server, this model uses the method proposed by Golestani [11] to compare the fairness:

$$\left| \frac{W_i^S(t_1,t_2)}{\rho_i} - \frac{W_j^S(t_1,t_2)}{\rho_j} \right| \le F^s \tag{3.14}$$

$W_i^S(t_1,t_2)$ and $W_j^S(t_1,t_2)$ are the service offered to connection i and j in the interval between t_1 and t_2 by server S, ρ_i is the service rate of session i and ρ_j is the service rate of session j. F^s is a constant which is the fairness of server S. This fairness comparison method depends on the services received and it is hard to obtain in the real practical environment.

3.6 CONCLUSION

In this chapter, four networking evaluation methods are introduced. They are Network Power by Leonard Kleinrock, Fairness Model by Raj Jain, Worst-Case Fairness Index by Hui Zhang and Latency Rate Server by Dimitrios Stiliadis and Anujan Varma. Network Power and Fairness Model are developed for years and they are mostly accepted by the industry and the academic research arena due to easy implementation, understand and interpretation. Both of the derived network parameters are based on the raw networking parameter such as delay and throughput which are easy to measure and obtain statistically in the real operating network. The latest derived network evaluation techniques, WFI and LR-Server are based on operating network parameters such as packet arrival timestamp, packet depart timestamp, current queue size and service rate which is hard to obtain in a real practical networking environment. They can give a good theoretical meaning on the term "fairness" in the general sense. Due to the above reasons, the two traditional network evaluation techniques, Network Power and Fairness Model will be used for our research analysis.

CHAPTER 4 SIMULATION SETUP AND QUEUING DISCIPLINE COMPARISON

The whole research is based on the simulation result on the Network Simulator 2 (discuss in appendix A and B). The simulations focus on the fairness among the same weighted flows and the fairness among different weighted flows. The first simulation bases on this simulation topology below, thirty sources and thirty sinks are used to pump the traffic into and out of a central node. The total available bandwidth 10Mb/s is shared among the thirty flows and the link delay is 10ms from the central node to the destination. All thirty source links get 10Mb/s bandwidth and 10ms link delay. The buffer of the central node gets 1500 packets space and the packet size is 500 bit each. This is the simulation topology setup to test the fairness among all same weighted flows.

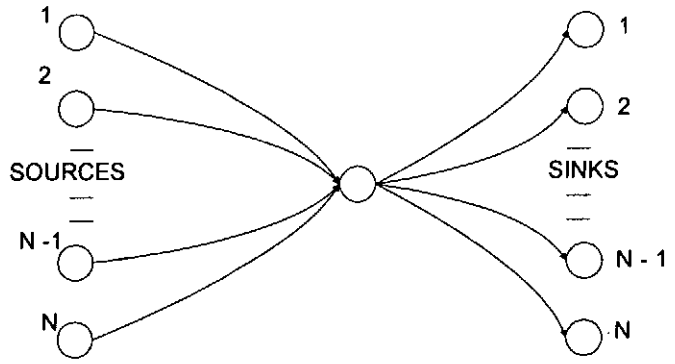

Figure 4-1 Simulation Network Topology

In the second simulation, the same network topology is used but the network parameters in the topology are changed. In the simulation, fairness among different weighted flows is researched. Thirty flows share the central node with total available bandwidth of 10Mb/s, flows numbered 10, 20, 30 set ten times weighted over all the other 27 flows. This implies

the 3 higher weighted flows will get 1.754Mbps individually and the other 27 flows will get 0.1754Mbps bandwidth per individual flow.

In the third simulation, the same network topology is used but with sixty flows sharing the central node with 10Mb/s, all the even numbered flows set ten times weighted over all the odd flows. Each higher weighted even numbered flow would get 0.303Mb/s and each lower weighted odd numbered flow would get 0.0303 Mb/s. This simulation tests how the weighted sharing pattern and the number of sharing node affect the performance of the queuing algorithms.

The loading condition and traffic arrival models will heavily affect the fairness factor, delay jitter and other network parameters such as throughput and delay. Four different loading conditions with four traffic arrival models are used in the first simulation and six different loading conditions with four traffic arrival models are used in the other two simulations.

The traffic arrival models are

(i) Poisson traffic of 4ms idle time and 4ms burst time,

(ii) Heavy burst Poisson traffic of 4ms idle time and 100ms burst time,

(iii) Constant bit rate IP phone model and

(iv) Constant bit rate with TCP model.

The simulation codes of the four traffic generation procedures can be found in appendix C. The Poisson traffic is modeled in NS2 as the exponential User Datagram Protocol (UDP) traffic which the packet inter-arrival time is exponentially distributed. The IP phone model is modeled as UDP traffic with constant bit rate source. The TCP sources are using the

sliding windows by Tahoe with Constant Bit Rate. The sliding windows by Tahoe (1998) enhanced the early implementation of TCP/IP by using slow-start algorithm and multiplicative decrease to deal with congestion control and congestion avoidance respectively. Furthermore fast retransmit is used to enable high efficiency.

4.1 Simulation Model Validation and General Findings

In figure 4-2, it shows the queues accumulate dramatically fast when the traffic loading is increasing. If the queue size increases, the packet waiting time (delay) in the FIFO queues will increase too. This property agrees with figure 4-2 which the delay rising dramatically fast when the traffic loading is increasing. In figure 4-2, when the arrival traffic is just saturated (WFQ with 0.35Mbps), the queue size is relatively small, but when the traffic is doubled (WFQ with 0.7Mbps), the queue size is boost up. When the traffic is keeping double up, the two lines with the highest loading (WFQ with 1.4Mbps and 2.8Mbps) show the queue saturated very quickly. They saturated at less than 0.5 second after the traffic is generated and occupied the whole queue size with 750000 bits which causing packet to drop. In figure 4-3 the Tahoe TCP sliding window is used for congestion control and the queue size starts to be controlled when the traffic is just saturated (WFQ with 0.35Mbps). When the traffic loading is boosting up, the queue size is controlled around 250000 bits. This two graphs show that the Tahoe TCP Algorithm provides an effective congestion control mechanism which lower the packet drop and keep the required queue size small. Figure 4-4 shows the queue size accumulating slowly for the saturated loading (0.35Mbps) at heavily burst Poisson traffic with 100ms burst time for 30 incoming flows. In figure 4-5, the queue size accumulating faster because more traffic is generated for the UDP/CBR IP Phone model. The queue size situation time is almost the same as the Poisson traffic but slower than the TCP traffic flows with congestion control. These four simulated graphs

validate the traffic simulation models. They are same as the expected result. RED algorithm uses the previous record of the queue size to predict the future queue size. It drops the packet with a real time dropping probability. It can control the queue size effectively even using UDP traffic flow and without the TCP congestion control. Figures 4-4 shows that the RED controls the queue size less than 250000 bits and saturated within 2 seconds after the traffics are generated. The queue sizes fluctuate greatly before saturation. Figures 4-5 shows that the FRED controls the queue size at about 200000 bits and saturated within 0.5 seconds for the overloaded and the heavily overloaded flows (1.4Mbps and 2.8Mbps). The congestion control response time for the lightly overloaded flow is slow in this case but still reasonable compare to the RED. It takes 2 seconds to saturate the queue size. In the lightly overloading condition (FRED with 0.7Mbps), the queue size is kept at below 170000 bits, this means more packets drop then required. The FRED using individual variables to estimate queue size of a specific flow and calculate the real time dropping probability for each flow. It claims to be fairer than the RED and it is more robust to the traffic pattern change.

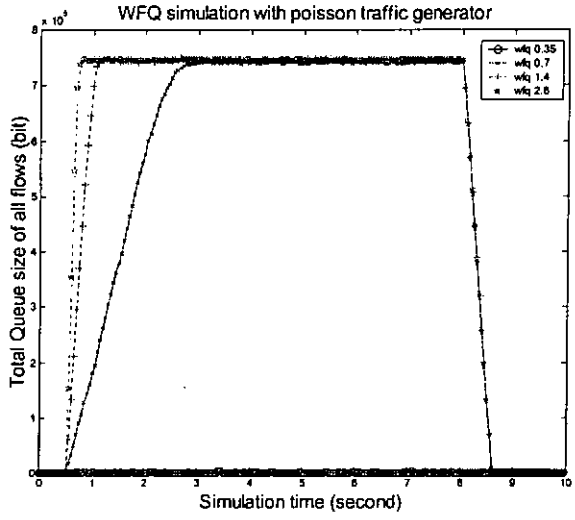

Figure 4-2 WFQ queue size simulation with poisson traffic at 4 different loadings

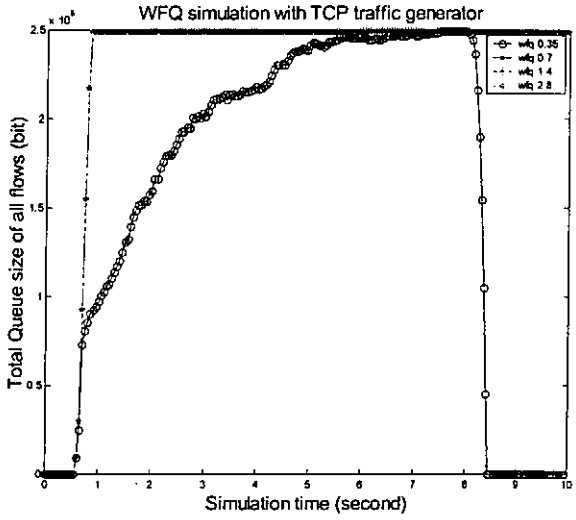

Figure 4-3 WFQ queue size simulation with TCP traffic at 4 different loading

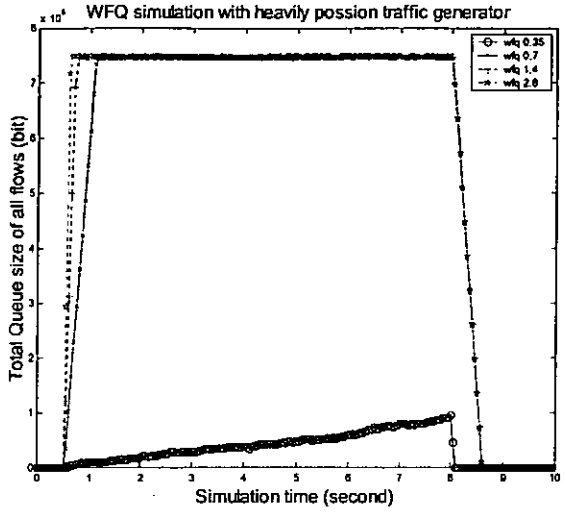

Figure 4-4 WFQ queue size simulation with heavily burst poisson traffic

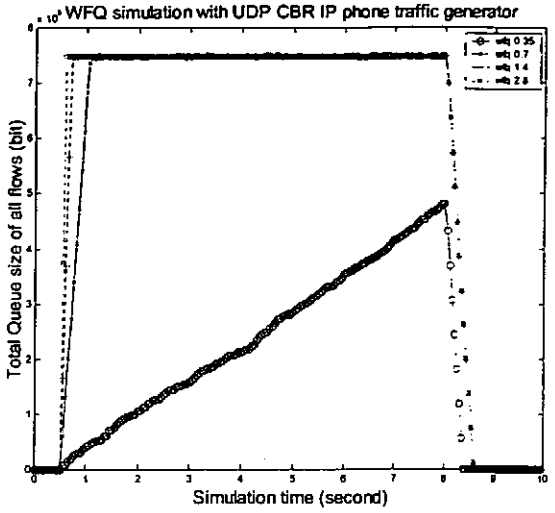

Figure 4-5 WFQ queue size simulation with IP phone (UDP/CBR) traffic

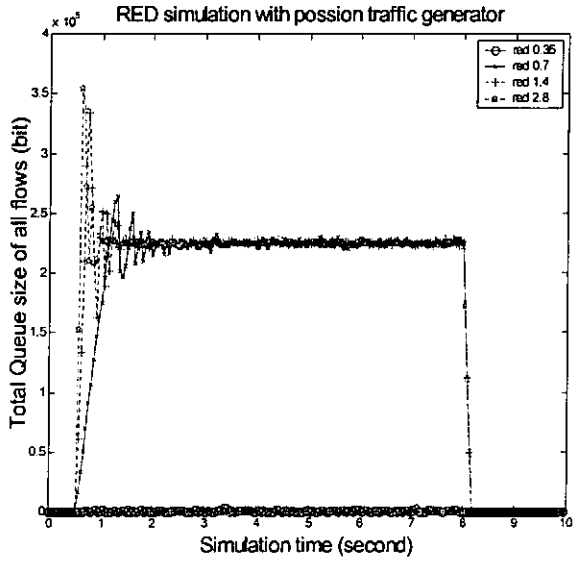

Figure 4-6 RED queue size simulation with poisson traffic

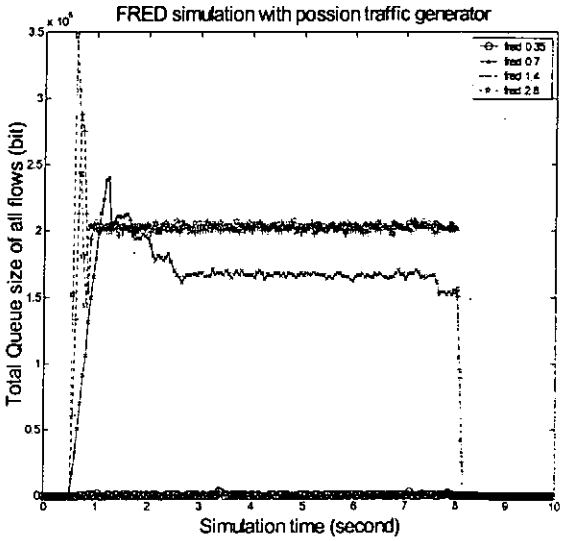

Figure 4-7 FRED queue size simulation with poisson traffic

In table 4-1, the standard deviation of throughput and the Jain's fairness index of throughput can test the fairness among same weight. The standard deviation which approach to 0 and the Jain's fairness index which approach to 1 will be the fairer algorithm. The results from the table validate that the FRED is fairer than RED in almost all the loading situations and traffic arrival patterns. The fairness different between the FRED and RED becomes larger when the loading increased. From the table 4-1, when the incoming

	Average Drop Per Flow	Total Drop	Average Delay	Delay Fairness	Average Thruput	Standard Deviation of Thruput	Fairness of Thruput
Poisson							
FRED							
0.35	0	0	0.000472709	99.6339	0.26033778	0.002520046	0.999909431
0.7	184	5523	0.132971367	99.9992	0.33854222	0.002052173	0.999964481
1.4	794	23832	0.159024833	99.9983	0.34032	0.002688098	0.999939693
2.8	2051	61540	0.1619232	99.997	0.34058667	0.003348581	0.999906567
RED							
0.35	0	0	0.000472709	99.6339	0.26033778	0.002520046	0.999909431
0.7	178	5357	0.171025633	99.9988	0.34108444	0.009205411	0.999296387
1.4	789	23675	0.177125433	99.9982	0.34115556	0.009993796	0.999171156
2.8	2049	61491	0.178633667	99.9974	0.34119111	0.010104674	0.999152855
Heavy Burst Poisson							
FRED							
0.35	0	7	0.034570187	99.9986	0.33580444	0.001973226	0.999966623
0.7	620	18622	0.1605645	99.9997	0.34035556	0.001321944	0.999985418
1.4	1886	56600	0.164110233	99.9992	0.34055111	0.001487102	0.999981567
2.8	4404	132128	0.1655446	99.9981	0.34060444	0.002383014	0.999952684
RED							
0.35	0	8	0.034869247	99.9981	0.33605333	0.001932374	0.999968038
0.7	623	18694	0.1765259	99.9996	0.34112	0.013251501	0.998543336
1.4	1887	56625	0.178465433	99.999	0.34117333	0.009479861	0.999254227
2.8	4408	132264	0.1790687	99.998	0.34122667	0.011141082	0.998970567
	Average Drop Per Flow	**Total Drop**	**Average Delay**	**Delay Fairness**	**Average Thruput**	**Standard Deviation of Thruput**	**Fairness of Thruput**
CBR							
FRED							
0.35	20	611	0.115746467	99.9988	0.33978667	0.003805799	0.999878744
0.7	674	20224	0.160896	99.9998	0.34060444	0.000965129	0.999992239
1.4	1988	59646	0.1643838	99.9998	0.34067556	0.001006908	0.999991556
2.8	4612	138387	0.165826867	99.9998	0.34071111	0.00087561	0.999993616
RED							
0.35	19	597	0.1118772	99.99986	0.33973333	0.004572917	0.99982489
0.7	672	20169	0.176998733	99.9997	0.34135111	0.008084135	0.999458117
1.4	1984	59520	0.178662667	99.9998	0.34129778	0.011852216	0.998835598
2.8	4611	138333	0.179193	99.9999	0.34133333	0.012953107	0.998609845
CBR/TCP							
FRED							
0.35	4	122	0.068971587	99.8046	0.35047111	0.003492342	0.999904024
0.7	7	226	0.081566327	99.9781	0.41523556	0.048840394	0.986802963
1.4	7	226	0.08163108	99.9823	0.41536	0.041114383	0.990617449
2.8	7	218	0.080584053	99.9583	0.41591111	0.042562881	0.989977797
RED							
0.35	4	120	0.067497593	99.7185	0.34967111	0.003318816	0.999912927
0.7	7	237	0.07831527	99.9699	0.41610667	0.076502156	0.968358934
1.4	7	237	0.07793901	99.9835	0.41585778	0.064752955	0.977099495
2.8	7	233	0.077441713	99.978	0.41633778	0.068796148	0.974284259

Table 4-1 RED and FRED comparison

traffic loading becomes higher, the fairness of the algorithm becomes lower in both of the algorithms. It is the properties of almost all the queuing algorithms. It is because the scheduler needs to process more packets in relatively shorter time. The congestion control or scheduling ability will be lowered. In the average drop and the total drop columns, RED drops less packet than FRED in Poisson, CBR and CBR/TCP but not the heavy burst Poisson case, this explains the lowered 0.7Mbps loading line in figure 4-7. Because the packet drop of FRED is more, the latency of the algorithm is also smaller. In the table, the number of packet drop and delay fairness of FRED are generally higher than RED but the throughput of FRED is lower than RED.

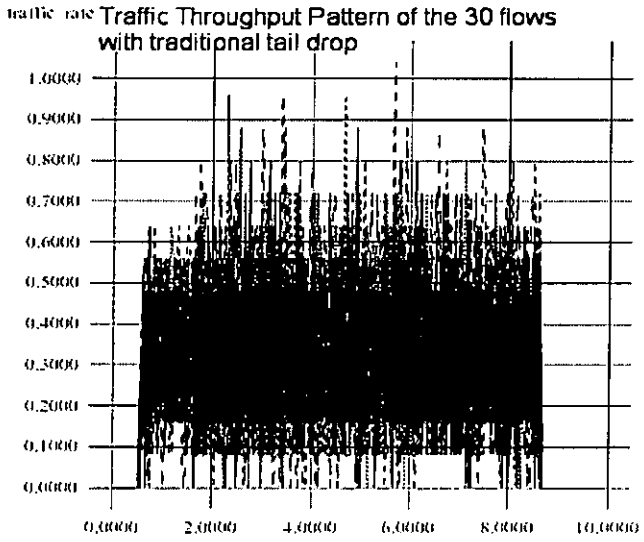

Figure 4-8 Traffic Troughput Pattern with traditional tail drop

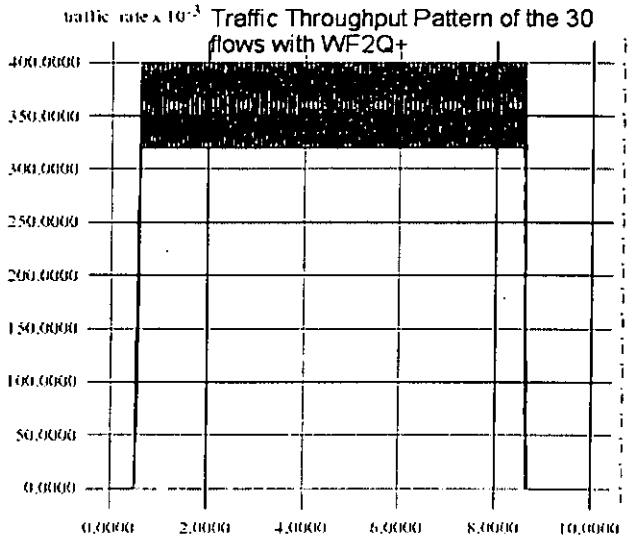

Figure 4-9 Traffic Troughput Pattern with WF^2Q+

In figure 4-8 and 4-9, both are the throughput graphs of the 30 flows sharing the link. Y-axis is the traffic rate (Mb/s in figure 4-8 and Kb/s in figure 4-9). X-axis is the time in second. It can be seen that in figure 4-8, the individual flow heavily and randomly competes for the shared bandwidth. The throughputs of all flows fluctuate heavily. The central line of the data chunk is about 0.35 Mbps. In figure 4-9, the traffic sharing is being controlled. The fluctuation is greatly reduced and it only happens around the saturated

throughput. The central line of the fluctuated data chuck is also about 0.35 Mbps. From the two figures, although the saturated bandwidth is at about 0.35 Mbps in both of the cases, the Traditional Tail Drop cannot control the traffic of the individual flow, they compete for the required bandwidth. In WF^2Q+ the traffic of individual flow can be controlled and the traffic throughput pattern shows a much higher regulated form.

4.2 SIMULATION STUDY ON FAIRNESS AMONG THE SAME WEIGHTED FLOWS

The four loading conditions for the first simulations are:

(i) Saturated at 0.35Mb/s sharing among all 30 flows,

(ii) Lightly Overloaded at 0.7Mb/s sharing among all flows,

(iii) Overload at 1.4Mb/s sharing among all flows and

(iv) Heavily Overloaded at 2.8 Mb/s sharing among all flows.

The compared queuing algorithms are Deficit Round Robin (DRR), Fair Queuing (FQ), Flow Random Early Drop (FRED), Random Early Drop (RED), Stochastic Fair Queuing (SFQ), Traditional Tail Drop (Tail), Virtual Clock (VC), Weighted Fair Queuing (WFQ) and Worst Case Fair Weighted Fair Queuing Plus (WF^2Q+).

When talking about fairness in queuing, it always means the throughput fairness. But different applications and users require different network parameters. Can the throughput fairness conclude about the network performance? There is no general public accepted terminology about the term "fairness". In the simulation, the drop fairness, delay fairness and the throughput fairness are all studied to give the reader a comprehensive view on the ambiguous term "fairness".

4.3 THROUGHPUT FAIRNESS

Table 4-2, it is easy to discover that WF^2Q+ is the fairest algorithm among the same weighted flows. The second fair algorithm is WFQ and the third fair algorithm is FQ, follow by VC and then FRED. The sixth fair algorithm is RED and seventh fair algorithm is the traditional tail drop. The DRR is just worse than the traditional tail drop. SFQ is the worst. It can be noticed that, the higher computational required algorithms always have better fairness. Is it cost effective to implement such fairness with the high expenses? Is the fairness level required to be high standard? Are the customers really interested on high

		VC	WFQ	WF2Q+	FQ	SFQ	DRR	Tail	FRED	RED
	MB/s									
Little	0.35	99.99	99.991	99.991	99.991	99.991	99.991	99.991	99.9909	99.991
Burst	0.70	99.95	100.000	100.000	100.000	98.429	98.933	99.950	99.9964	99.930
Poisson	1.40	99.93	100.000	100.000	100.000	91.418	97.467	99.854	99.994	99.917
	2.80	99.92	100.000	100.000	100.000	91.381	97.449	99.839	99.9907	99.915
Heavy	0.35	99.997	99.997	100.000	99.997	99.997	99.997	99.997	99.997	99.997
Burst	0.70	99.417	100.000	100.000	100.000	91.082	97.451	99.417	99.999	99.854
Poisson	1.40	99.654	100.000	100.000	100.000	91.196	97.296	99.288	99.998	99.925
	2.80	99.409	100.000	100.000	100.000	91.290	97.272	98.771	99.995	99.897
Constant	0.35	99.984	99.984	99.999	99.984	99.811	99.984	99.984	99.988	99.982
Bit Rate	0.70	99.942	100.000	100.000	100.000	91.436	97.511	99.918	99.999	99.946
IP Phone	1.40	99.955	100.000	100.000	100.000	91.409	97.494	99.877	99.999	99.884
	2.80	99.961	100.000	100.000	100.000	91.411	97.503	99.892	99.999	99.861
Constant	0.35	99.982	99.982		99.982	99.982	99.982	99.982	99.9904	99.991
Bit Rate	0.70	99.994	100.000		100.000	94.395	97.590	99.985	98.6803	96.836
TCP	1.40	99.994	100.000		100.000	91.600	97.731	99.985	99.0617	97.710
	2.80	99.994	100.000		100.000	91.600	97.731	99.985	98.9978	97.428
Average		99.879	99.997	99.999	99.997	94.152	98.211	99.795	99.792	99.442

Table 4-2 Jain's Fairness of throughput among the same weighted flows simulation

fairness? It is believed that the customers are more interested on the weighted fairness which is the fairness among the different weighted flows rather then the traditional fairness which is the fairness among all same weighted flows. It is because the customers want to ensure the different performances between the low paid users and the high paid users when they pay more money.

Figure 4-10 is simulated in overloading (2.8Mbps) situation with Poisson Traffic Arrival Model. The SFQ is the most fluctuated one with the lowest fairness. The DRR is the second most fluctuated algorithm. Even the traditional best effort service is relatively stable than these two algorithms. WF^2Q+, WFQ and FQ provide general higher and stable throughput among the other algorithms. These three algorithms show almost straight lines across different flows. The fairness of FRED and RED are in between the two sets of algorithms. They are compared to be less fluctuation and stable when comparing to the traditional best effort service. These properties also occur in the lower input traffic loading condition with constant bit rate traffic over UDP & TCP and the heavy Poisson traffic model. The throughput fairness can be concluded by WF^2Q+ > WFQ > FQ > VC > FRED > RED > TAIL > DRR > SFQ.

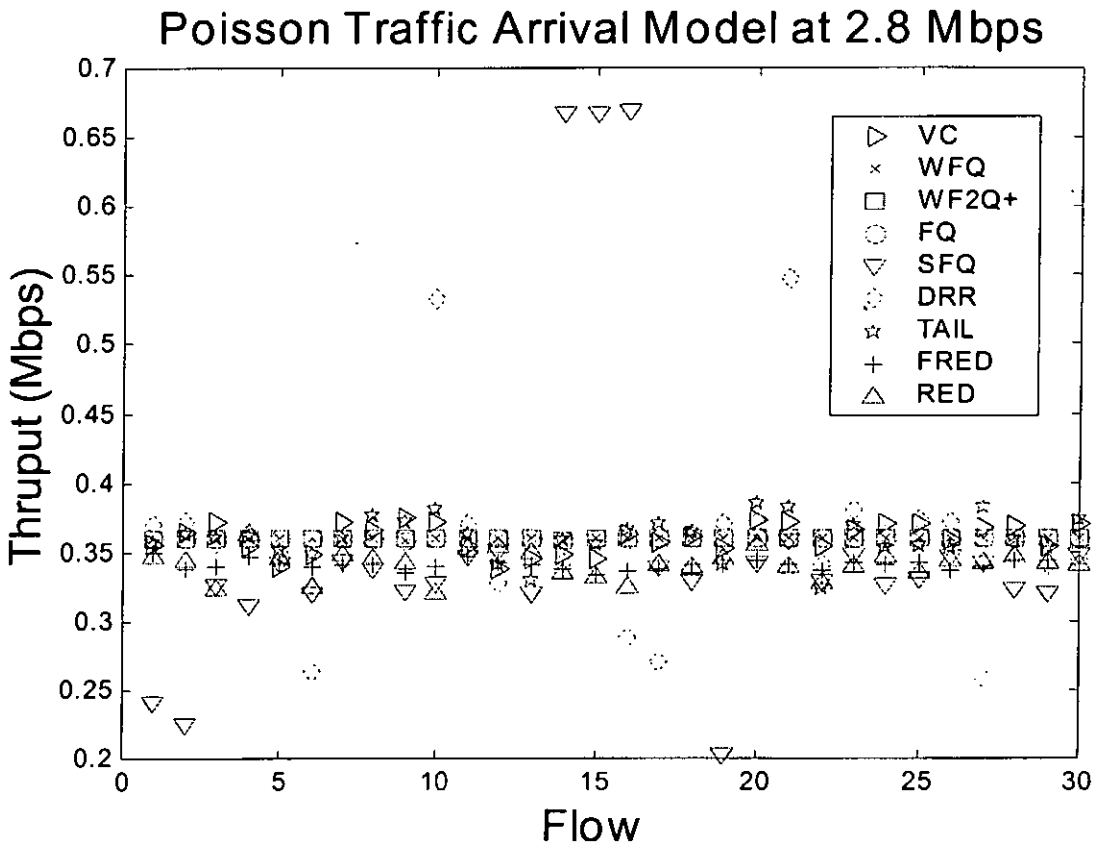

Figure 4-10 The throughput of the 30 flows with the 8 algorihtms

4.4 PACKET DROP FAIRNESS

	Input Load (Mbps)	VC	WFQ	WF2Q+	FQ	SFQ	DRR	TAIL	FRED	RED
Poisson	0.35	0	0	0	0	0	0	0	0	0
	0.7	10.17604	10.23348	10.28558	10.25536	86.03748	72.60474	10.17604	8.413864	11.41838
	1.4	27.90501	34.12401	34.0218	34.02332	210.1134	108.6565	22.86844	31.95903	18.51281
	2.8	51.08613	58.45364	58.39107	58.41321	218.1401	120.5512	53.64635	52.95737	44.23721
Heavy Burst Poisson	0.35	0	0	0	0	0	0	0	0.557086	0.525226
	0.7	52.27118	5.965418	5.985615	5.965418	211.7529	113.3711	52.27118	7.908965	26.12305
	1.4	36.20154	18.64551	18.81947	18.61405	209.5252	117.6595	56.50938	14.60043	18.85608
	2.8	61.69949	36.33702	36.30569	36.33702	224.3999	120.5803	88.26645	28.56692	31.22223
CBR	0.35	0	0	0	0	5.391564	0	0	3.429085	5.391564
	0.7	13.5201	8.913376	8.924975	9.005745	12.83449	109.1715	14.27682	10.89131	12.83449
	1.4	20.63977	16.78464	16.90644	16.90644	23.4844	108.7095	25.72936	13.73895	23.4844
	2.8	27.33887	22.79141	22.82467	22.82467	31.56657	110.4258	33.92842	16.937	31.56657
TCP/CBR	0.35	0	0	0	0	0	0	0	1.618854	1.414214
	0.7	0	0	0	0	0	0	0	1.203443	1.938716
	1.4	0	0	0	0	0	0	0	1.114172	1.790781
	2.8	0	0	0	0	0	0	0	0.870988	1.564697

Table 4-3 Standard Deviation of the packet drop in four loading condition with four different input loading traffic models.

In the packet drop fairness, the standard deviation is used to compare the fairness instead of using the Jain's Fairness Index that used in the throughput and delay fairness comparison. It is because if the flow is without any packet drop, the statistics samples in the packet drop can be zero. This situation always happens especially in the low loading condition. In the throughput and delay fairness, the statistics samples always get some quantify values and Jain's Fairness Index can be used. The Jain's Fairness Index will become undefined (zero divided by zero) if all the flows get no packet drop in the simulation. This cause troubles in numerical comparison and further manipulation of the quantified values. Because of this reason, standard deviation instead of the Jain's Fairness Index will be used in all fairness comparison of the packet drop.

In the table 4-3, it can be noticed that the standard deviation in some of the conditions are zero. It is because there is no packet drop in the specific conditions and it does not mean that the algorithm is extreme fair in packet dropping. It can be noticed that the TCP can

control the input loading traffic effectively, so the packet drop is zero in most of the cases. FRED and RED are the active dropping algorithms which drop packet even the queue is not overflow. They intended to drop packet to keep the throughput fairness among different flows and prevent the anticipated congestion, so it will drop packet in TCP and in some low loading situations. The packet drop fairness decreases with the input traffic loading increasing for almost all the algorithms. The packet drop fairness of FRED and RED are better than WF^2Q+ and WFQ in most of the case. It is because FRED and RED control the packet throughput by dropping the packet accordingly. All algorithms get higher packet drop fairness (or lower standard deviation) in the Constant Bit Rate loading condition. It is because constant bit rate is easier and more accurate in the queue size prediction. The packet drop fairness of DRR and SFQ are low and they are even worst than the best effort service (traditional tail drop).

Figure 4-11 is simulated in the overloading situation at 2.8Mbps with Poisson traffic arrival model. The packet drop numbers among the flows of DRR and SFQ compare to be fluctuated. They are worse than the best effort service. It is hard to determine the stability among all the other algorithms. The numerical data in table 4-3 is required to determine the little different between the stability among all the other algorithms. By summarizing the table and the figure, the packet drop fairness can be concluded as follow: FRED > RED > WFQ > FQ > WF^2Q+ > VC > TAIL > DRR > SFQ.

4.5 DELAY FAIRNESS

From the table 4-4, the difference of fairness between the queuing algorithms are very small. The fairness increases with the incoming traffic loading. Actually it is hard to rank the fairness among the algorithms. It is because they get different fairness performance in different traffic models and with different loading. However it can be generally

summarized and the delay fair of SFQ is the worst, FQ is the best. It can be concluded VC > TAIL > FRED > RED > FQ > WFQ > WF^2Q+ > DRR > SFQ.

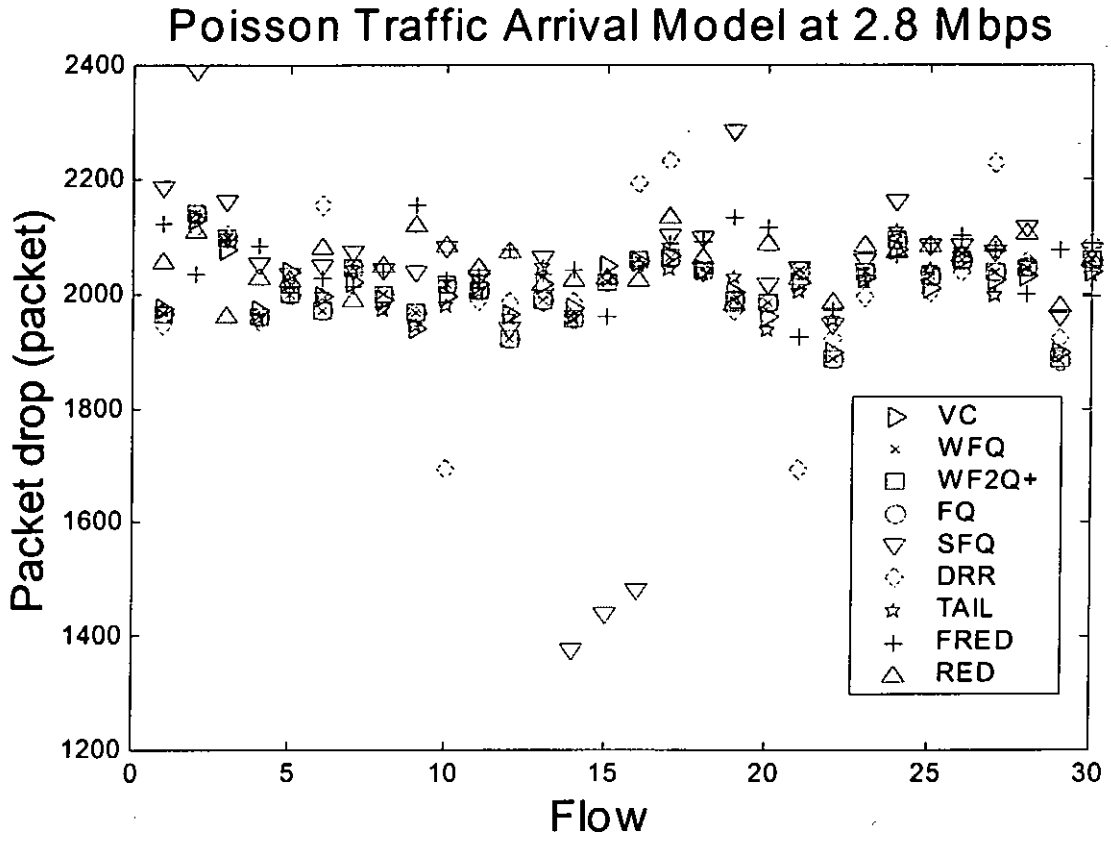

Figure 4-11 The packet drop pattern among 30 flows with the 8 algorithms

		VC	WFQ	WF2Q+	FQ
	MB/s				
Little	0.35	99.634	99.634	97.956	99.634
Burst	0.70	99.998	99.953	99.954	99.954
Poisson	1.40	99.996	99.997	99.997	99.997
	2.80	99.999	99.999	99.999	99.999
Heavy	0.35	99.999	89.130	86.826	92.327
Burst	0.70	99.995	100.000	100.000	100.000
Poisson	1.40	99.999	100.000	100.000	100.000
	2.80	99.998	100.000	100.000	100.000
Constant	0.35	99.998	93.544	94.344	93.647
Bit Rate	0.70	99.999	100.000	99.999	99.999
IP Phone	1.40	100.000	100.000	100.000	100.000
	2.80	100.000	100.000	100.000	100.000
Constant	0.35	99.997	98.819		98.739
Bit Rate	0.70	99.998	100.000		100.000
TCP	1.40	99.998	100.000		100.000
	2.80	99.998	100.000		100.000
Average		99.975	98.817	98.256	99.018

Table 4-4a Jain's Fairness of delay among the same weighted flows simulation

	SFQ	DRR	Tail	FRED	RED
Little	99.682	99.690	99.634	99.634	99.634
Burst	89.893	93.433	99.998	99.999	99.999
Poisson	99.649	98.461	99.994	99.998	99.998
	99.986	98.293	99.993	99.997	99.997
Heavy	13.235	16.269	99.999	99.999	99.998
Burst	97.958	98.587	99.995	100.000	100.000
Poisson	99.989	98.251	99.997	99.999	99.999
	99.994	98.582	99.995	99.998	99.998
Constant	19.809	24.332	99.998	99.999	99.999
Bit Rate	99.156	98.531	99.999	100.000	100.000
IP Phone	99.991	98.319	99.999	100.000	100.000
	99.998	98.633	100.000	100.000	100.000
Constant	81.095	87.213	99.997	99.805	99.719
Bit Rate	93.735	97.222	99.997	99.978	99.970
TCP	93.666	97.262	99.997	99.982	99.983
	93.666	97.262	99.997	99.958	99.978
Average	86.344	87.521	99.974	99.959	99.954

Table 4-5b Jain's Fairness of delay among the same weighted flows simulation

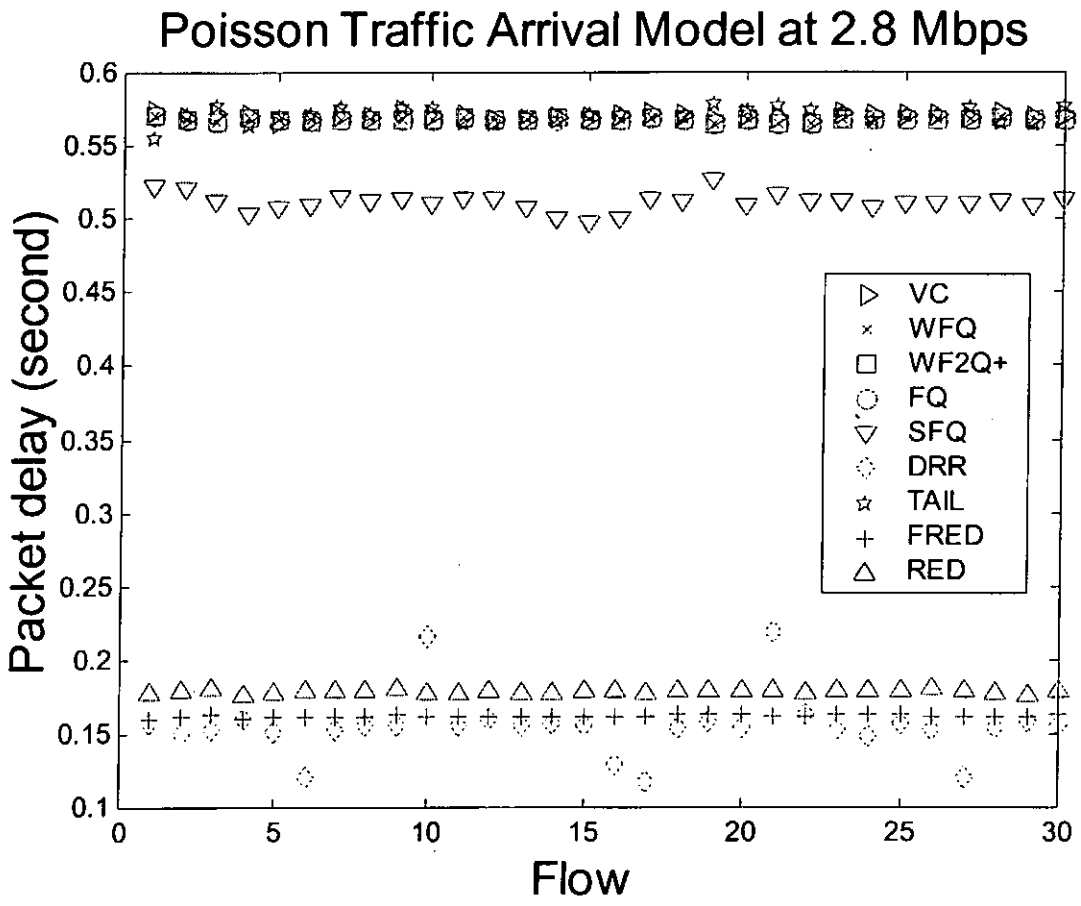

Figure 4-12 The packet delay pattern among 30 flows with the 8 algorithms

In figure 4-12, it can be noticed that the DRR and the SFQ are quite fluctuated. All the other algorithms seem to be the same. The delay timings of VC, WFQ, WF^2Q+, FQ and

Traditional Tail Drop are very close. They seem to form a straight line at the upper part of the graph. The delay timing of DRR is the lowest one followed by FRED and RED. FRED and RED are algorithms which drop packet actively to keep the congestion and the queue size low. That is why the delay time is low as well. FRED drops more packets so the delay time is lower than RED.

4.6 CONCLUSION ON THE FAIRNESS AMONG THE SAME WEIGHTED FLOWS

After the research on throughput fairness, packet drop fairness and delay fairness, it is hard to rank which algorithm is the fairest. Different algorithms get their own characteristic and they get their own advantage in the specific network parameter. Even in the specific network parameter simulation, the difference between the specific fairness is very small. For the network performance improvement, it depends on what applications or services are required on the link and which network parameter is more important rather than choosing a specific algorithm. It is because there is no all around algorithm in the fairness simulation. The simulation also concludes that it is not worth to spend huge computational cost to keep little improvement on the fairness among the same weighted flows, which the customers in the commercial market are not interested in.

4.7 WFQ SIMULATION STUDY ON DIFFERENT WEIGHTED FLOWS

The figure 4-13 is based on the simulation 2 that described in the beginning of this chapter. Flows 0 to 30 use the Poisson arrival model, flows 31 to 60 use the heavy burst Poisson arrival model, flows 61 to 90 use the Constant Bit Rate UDP IP phone model and flows 91 to 120 use the Constant Bit Rate TCP model. It can be seen that the network throughput is not conformed to the assigned weight when the shared link is just saturated. While the traffic loading is increasing and the shared link becomes overloaded, the network throughput is conformed to the assigned weight gradually. It is because at the saturated

loading condition, there are bandwidths left after the 3 higher weighted flows used the bandwidth on the trunk. All the remained bandwidth is equally shared by all the other flows with lower weighted settings. In the Poisson model, the point of the three higher weighted flows 11.2Mbps (the square) and 5.6Mbps (the pointed down triangle) are overlapping. This means something happened in between 2.8Mbps and 5.6Mbps input loading. When the input loading is increasing, it will reach a point where the 3 higher weighted flows are sharing all the weighted bandwidth and no extra bandwidth for the other lower weighted flows to share, that point is called the fairness situation point. In the Poisson Model, the fairness saturation point is between 2.8Mbps and 5.6Mbps. Three points (square, pointed up triangle and pointed down triangle) of higher weighted flows are overlapped in all the other three models. The fairness saturation point is between 1.4Mbps and 2.8Mbps.

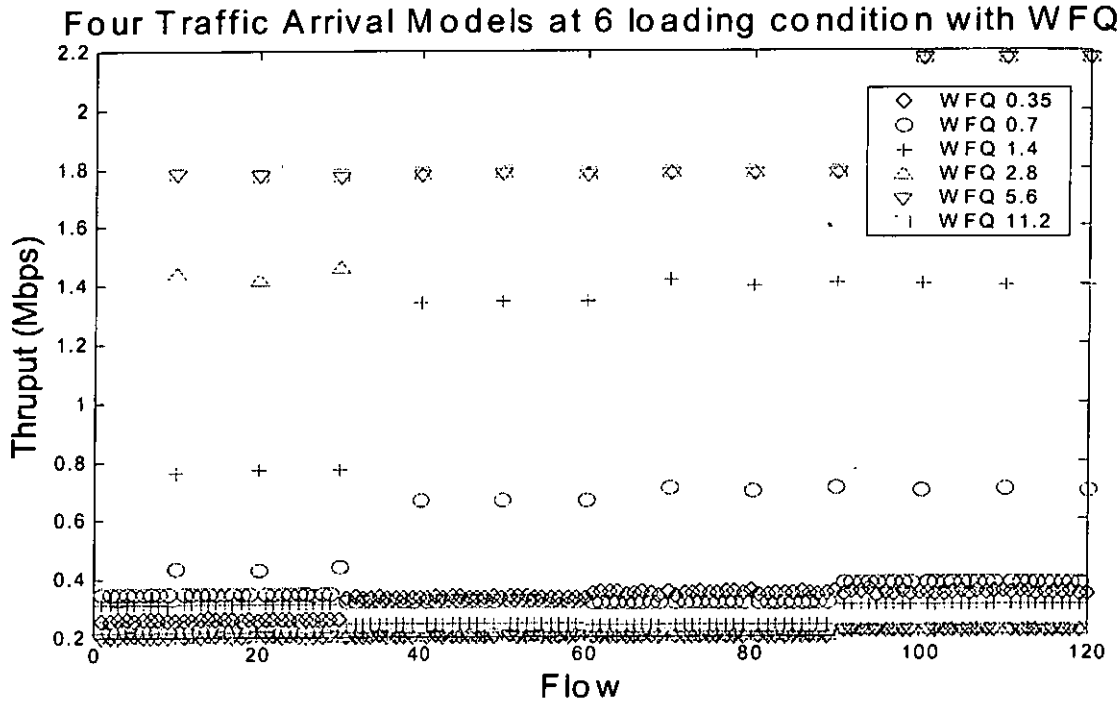

Figure 4-13 Weighted throughput simulation on four traffic arrival models with six input traffic loading conditions.

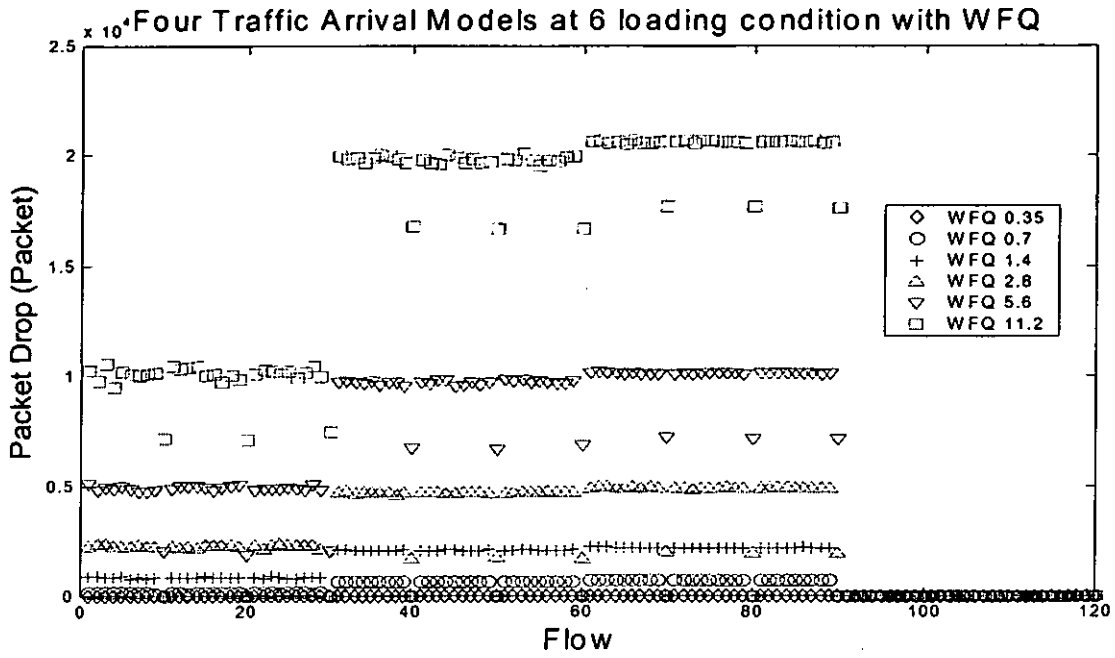

Figure 4-14 Weighted packet drop simulation on four traffic arrival models with six input traffic loading conditions.

As same as figure 4-13, figure 4-14 is based on the simulation 2 that described in the beginning of this chapter. Flows 0 to 30 use the Poisson arrival model, flows 31 to 60 use the heavy burst Poisson arrival model, flows 61 to 90 use the Constant Bit Rate UDP IP phone model and flows 91 to 120 use the Constant Bit Rate TCP model. It can be noticed that in the CBR/TCP model, there is no packet drop. It is because TCP control the queue size effectively and it involves packet acknowledgement and retransmission mechanism. There are 3 points, which are specifically lowered in each set of data. They are the 3 higher weighted flows, which have higher throughput and lower packet drops. It can be noticed that when the input loading becomes higher, the link cannot accommodate the loading and drop more packets. Out of the four models, the CBR/UDP models get the highest packet drops, heavy burst Poisson with the middle packet drops and the Poisson model gets the lowest packet drops. This is because the CBR/UDP generate much more incoming packets in the simulation. It can be noticed that the difference between the higher weighted drop

and the lower weighted drop is almost constant among all the loading conditions across the three input loading models.

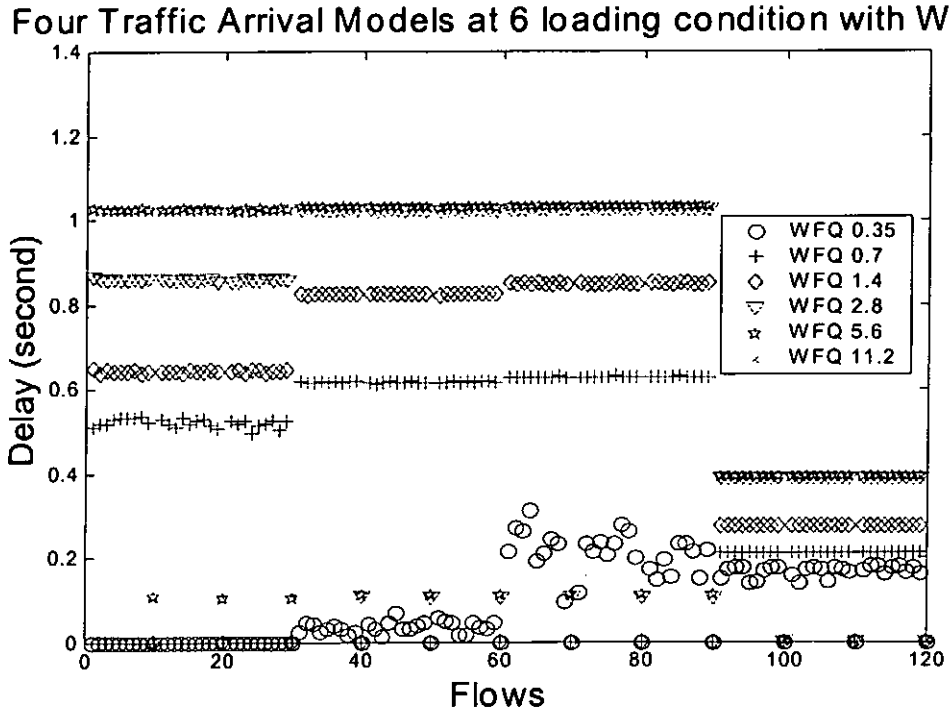

Figure 4-15 Weighted packet delay simulation on four traffic arrival models with six input traffic loading conditions.

In figure 4-15, it can be seen that the delay timing is quite fluctuated in the burst Poisson Model, TCP IP Phone Model and CBR UDP Model. It can be noticed that when the loading is increasing, the delay time of the lower weighted flows are increasing dramatically but the delay time of the higher weighted flows almost the same. In the Poisson Model, the lower weighted flow of WFQ 5.6Mbps and WFQ 11.2Mbps lines overlap. This indicates that the queue is saturated so the delay doesn't increase with the input traffic loading. In all the other three models, the traffic is saturated beyond 2.8Mbps. In the TCP model, the delay times are extremely low even in the extremely overloading traffic condition. This shows that the TCP sliding window algorithm controls the

congestion very well. It can alleviate the queue size requirement and lower the delay dramatically.

4.8 SIMULATION COMPARISON ON WFQ AND WF²Q+

Actually the difference between WFQ and WF²Q+ is just the packet selection policy. WF²Q+ is improved from WFQ and it uses the smallest eligible virtual finish time first (SEFF) which increases the fairness and lower the computational complexity. Because the performance difference between WFQ and WF²Q+ is small, quantified numerical values are used for comparison instead of the figures.

		Average Delay	Average Thruput	Total Drop	Average Network Power	Power Fairness
Poisson						
WF2Q+	0.35	0.000505588	0.260337778	0	526.1419919	97.80687765
	0.7	0.499798567	0.359253333	4272	0.719130716	99.9526637
	1.4	0.555571567	0.359324444	22499	0.646784906	99.99700244
	2.8	0.5676572	0.359857778	60356	0.633938925	99.99930038
WFQ						
	0.35	0.000472709	0.260337778	0	552.6196706	99.66981924
	0.7	0.499786467	0.359413333	4275	0.719474937	99.95167626
	1.4	0.5556523	0.359662222	22499	0.64729925	99.99674387
	2.8	0.567658967	0.359857778	60356	0.633937815	99.99904647
Heavy Burst Poisson						
WF2Q+	0.35	0.052353677	0.333742222	0	7.846003681	75.34053726
	0.7	0.5530405	0.359608889	17599	0.650242649	99.9994104
	1.4	0.5685875	0.35984	55427	0.632866984	99.99985332
	2.8	0.573424967	0.359893333	131101	0.627620751	99.99994529
WFQ						
	0.35	0.034827047	0.336177778	0	11.01576627	86.9241968
	0.7	0.5529668	0.359555556	17602	0.650233892	99.99916658
	1.4	0.568589933	0.359822222	55428	0.632833226	99.99973523
	2.8	0.573423133	0.359875556	131103	0.627592103	99.99977243
UDP CBR IP Phone						
WF2Q+	0.35	0.146974241	0.344711111	0	1.873782234	88.45806804
	0.7	0.198778299	0.359893333	19174	0.648074655	99.99928113
	1.4	0.21321716	0.359928889	58481	0.631940112	99.99986849
	2.8	0.217816065	0.359964444	137252	0.626985552	99.99993955
WFQ						
	0.35	0.160776214	0.350453333	0	1.994385545	88.63397318
	0.7	0.198697101	0.359893333	19174	0.648165693	99.99966178
	1.4	0.213238343	0.359911111	58482	0.631904987	99.99996744
	2.8	0.217780029	0.36	137250	0.627047061	99.99999786

Table 4-6 WFQ and WF²Q+ comparison

Refer to the fairness study in this chapter (Table 4-2, 4-3 and 4-4). It can be noticed that WF^2Q+ is fairer in the throughput fairness but it is less fair in the delay fairness. In the packet drop fairness, the performances of the two algorithms are about the same.

In the table 4-5, it can be noticed that the average delay of WF^2Q+ is generally a bit higher than WFQ. This is because the total packet drop of the WFQ is generally higher. More packet drop means decreasing in queue size and packet waiting time in the queue. The average throughput of WFQ is generally high. WFQ with low delay and high throughput, the network power is also higher. For comprehensive fairness comparison, the network power fairness is calculated and WF^2Q+ is found to be with higher fairness.

	MB/s	Normalized Thruput	Normalized Drop	Normalized Delay	Normalized Thruput	Normalized Drop
		Poisson			Heavy Burst Poisson	
WF2Q+	0.35	0.996662621		4.861446241	0.999056771	
	0.7	1.259805317	0	2431.576046	2.077220314	0
	1.4	2.476329669	0	2813.286405	5.42414564	0
	2.8	6.703751498	0	180.1824949	8.802033434	0.366345123
	5.6	9.962502773	0.408225905	9.839640127	8.808192394	0.692988778
	11.2	8.802033434	0.714271576	9.354822064	8.809556314	0.843343091
WFQ						
	0.35	0.996662621		3.243670803	0.99530158	
	0.7	1.256443738	0	2378.927137	2.075947039	0
	1.4	2.473500476	0	2828.246643	5.415065763	0
	2.8	6.053687365	0	533.1321828	8.802149487	0.365916399
	5.6	8.787325456	0.406793534	9.678279713	8.80826737	0.692885823
	11.2	8.79783182	0.714172605	9.359171133	8.811311555	0.843250063

Table 4-7a Weighted Fairness Comparison between WFQ and WF^2Q+

	MB/s	Normalized Delay	Normalized Thruput	Normalized Drop	Normalized Delay
			UDP CBR IP Phone		
WF2Q+	0.35	296.1783258	1.033581662		1045.0029
	0.7	2817.561323	2.224272931	0	2769.495743
	1.4	3057.875703	6.226162333	0	1711.053567
	2.8	9.303515438	8.833333333	0.39535839	9.259767405
	5.6	9.187945648	8.834215168	0.709301176	9.174681048
	11.2	9.151233197	8.833773087	0.857766355	9.156241849
WFQ					
	0.35	185.0235048	1.018580223		993.6737494
	0.7	2814.231531	2.220409429	0	2315.432076
	1.4	3055.227404	5.83099631	0	2547.933723
	2.8	9.317261738	8.810887412	0.39535839	9.364528969
	5.6	9.191763685	8.810026385	0.70940002	9.174303997
	11.2	9.156497665	8.820035115	0.85771786	9.157417296

Table 4-8b Weighted Fairness Comparison between WFQ and WF^2Q+

Because WFQ and WF^2Q+ are the weighted queuing algorithms, the weighted fairness is also interested. The weighted fairness is to measure how fair the algorithm assigns network resources according to the preset weighted value. The algorithm with higher weighted fairness means that it treats the higher paid user and the low paid user fairly.

In table 4-6, the normalized value is used for weighted fairness comparison. The normalized value is calculated by the formula below:

$$\overline{\mu_n} = \frac{\overline{\mu_H}}{\overline{\mu_L}} \tag{4.1}$$

$\overline{\mu_n}$ is the normalized value, the value can represent throughput, drop or delay. $\overline{\mu_H}$ is the average of all higher weighted flow values and $\overline{\mu_L}$ is the average of all lower weighted flow values. In table 4-6, the weighted fairness is studied by using the normalized value in three traffic arrival models and six loading conditions. In the Poisson Model, the weighted fairness of WF^2Q+ in throughput, packet drop and delay are higher than WFQ. In the CBR UDP IP Phone Model, the fairness of WF^2Q+ in throughput and packet drop is higher than WFQ. The delay-weighted fairness in the IP Phone models are equally fair for the two algorithms. In the Heavy Burst Poisson Model, the weighted fairness of delay and throughput are equally fair but WF^2Q+ is fairer in the drop-weighted fairness. In the throughput and delay weighted fairness of the Heavy Burst Poisson model, it is noticed that the WF^2Q+ algorithm is fairer in the low loading condition (0.35 to 1.4Mbps). In the high loading condition (2.8 to 11.2Mbps), the WFQ algorithm is fairer.

After the weighted fairness study, there is no best solution. The weighted performance of queuing algorithm depends on the traffic arrival model, loading condition and the specific interested network parameter. For the general sense, WF^2Q+ algorithm is fairer in the weighted simulation.

CHAPTER 5 WEIGHTED DEFICIT PROBABILITY DROP

The simulations in chapter 4 show WF^2Q and WF^2Q+ are extremely fair on the same weighted flows. They require high computation power to compute the flow state variables. Recent research on queuing algorithm focuses on fairness with different traffic conditions and reduction of the computation requirement. Is the fairness among the same weighted flows the highest important factor in designing the algorithm? Are the users of the network really concern about this fairness? Is it cost-effective to allocate computer power in the same weighted flow fairness? In the commercial arena, the users paid the same monthly fees, they do not concern too much on the fairness on the network quality among the other users who paid the same money. They concern more on how large the quality different if they paid more and ensure the difference between the high paid and low paid user. That means the network resources need to be fairly proportionate according to the preset weight. This is the fairness among different weighted flows. The queuing algorithm design has to be based on the "quality fairness" in this sense.

In order to lower the computation requirement on the algorithm, the algorithm design has to be based on dropping algorithms such as RED and FRED. They share the congestion control responsibility between the network layer and the transportation layer. The TCP mechanism in the transportation layer uses sliding window and the shrink of the window size is activated by packet drops. It is a widely deployed protocol using in the Internet nowadays. It can distribute the computational power from the network process node itself to the user's computer that using the network.

Stiliadis and Varma [36] proposed the following in order to minimize the end-to-end delay in the network server. The ideal algorithm needs to include the following attributes:

Insensitivity to traffic patterns of other sessions: Ideally, the end-to-end delay guarantees for a session should not depend on the behaviour of other sessions. Delay bounds that are independent of the number of sessions sharing the outgoing link. Ability to control the delay bound of a session by controlling only its bandwidth reservation.

The algorithm design will be based on the above properties. It is called Weighted Deficit Probability Drop (WDPD) algorithm that can provide a fair proportion of a specified shared bandwidth. It keeps the quantum values in the en-queuing module with the packet being dropped according to the corresponding value of a specific flow. The flow with a quantum size larger than the corresponding tail packet size will force the incoming packet to be dropped. This algorithm extends the idea of the deficit round robin but the definition of quantum is changed. The meaning of quantum in the deficit round robin is used to de-queue the packet and the flow. The flow with larger quantum can de-queue a larger packet or few smaller packets. In the proposed algorithm, the quantum is used to drop the tailing packet at the end of the queue. The flow that has a larger quantum will be dropped earlier. Hence, the corresponding flow throughput can be reduced. This quantum mechanism is used to ensure the algorithm worked perfectly in the variable packet network such as the Ethernet. In the constant cell size environment, like the ATM, the quantum calculation algorithm can be eliminated to further reduce the required computation.

The packet in each flow is dropped according to the pre-calculated probability. This dropping probability is calculated at the flow setup stage. When the specified flow is random dropping, it utilizes the pre-calculated probability which increasing the dropping

deficit counter with the pre-defined quantum value. The detail of the algorithm flow is shown below.

Initialization:

```
remain bandwidth=total throughput;
remain weight= total weight;
for (i = 0; i <=total node; i++)
        flag[i]=1;
do {
        Consumed Bandwidth=0;
        Consumed Weight=0;
        flag1=1;
        for (i = 1; i <=total node; i++) {
                if flow[i].weight/(remain weight)*(remain bandwidth)>=fs[i].(request bandwidth)
                *(traffic arrival factor) && (flag[i]==1)) {
               fs_[i].dropprob=0;
                        flag[i]=0;
                        flag1=0;
                        (consume bandwidth)+=fs[i].(request bandwidth) *(traffic arrival factor)
                        (consumed weight)+=fs[i].weight;
        }
        }
        (remain bandwidth)-=(consumed bandwidth)
        (remain weight)-=(consumed weight);
} while (flag1==0)
for (i = 1; i <=totalnode; i++) {
        if fs[i].weight/(remain weight)*(remain bandwidth < fs[i].(request bandwidth)* (traffic
        arrival factor) && (flag[i]==1)) {
        fs[i].dropprob=1-fs[i].weight/(remain weight) * (remain bandwidth)/(fs[i].(request
        bandwidth)*(traffic arrival factor));
        }
}
for (i = 1; i <=totalnode; i++)
        {DeficitCounter[i]=0;}
```

En-queuing:

```
flowId=ExtractFlow(pkt);
u = Random Number generated between (0 to 1)
if (u <= fs[flowId].prob) {
        fs[flowId].(deficit counter)+=Quantum
        droptype = DTYPE_UNFORCED
}
if (fs[flowId].(flow length)>=fs[flowId].(flow length limit)) {
        droptype = DTYPE_FORCED
}
FreeBuffer();
Enqueue(flowId,pkt);
```

De-queuing:

```
if (droptype == DTYPE_FORCED)
        {drop(pkt);}
else if (droptype == DTYPE_UNFORCED) {
        while (fs[flowId].(deficit counter)>(the size of tail packet at flowId)) {
                drop(pkt);
                deficit counter=deficit counter - (the size of dropped packet)}
        }
}
```

By using the above algorithm, the weighted random dropping probability can be computed.

Figure 5-1 and 5-2 show the details operation of the weighted deficit probability drop.

In a Markov M/M/1/K queue the dropping probability, P_k, can be calculated by:

$$P_k = \frac{(1-\rho)\rho^K}{1-\rho^{K+1}} \tag{5.1}$$

where ρ is the traffic intensity and is given by λ/μ, λ is the traffic arrival rate, μ is the traffic departure rate and K is the buffer size in packet. Dividing from both the denominator and the nominator of equation 5.1 by ρ^{K+1}, the equation can be simplified as:

$$P_k = \frac{1-\frac{\mu}{\lambda}}{1-(\frac{\mu}{\lambda})^{K+1}} \tag{5.2}$$

The traffic arrival rate must be greater than the traffic departure rate, that is $\mu/\lambda < 1$, and K is being a large number, then equation 5.2 becomes:

$$P_k = 1-\frac{\mu}{\lambda} \tag{5.3}$$

The initialisation part of the algorithm employs the above equation to compute the dropping probability for each flow, which will also assign bandwidth to the flow with higher weight first, followed by those with lower weight. If the computed weighted-bandwidth is larger than the requested bandwidth in the higher weighted flows, then the algorithm will provide more bandwidth to the lower weighted flow.

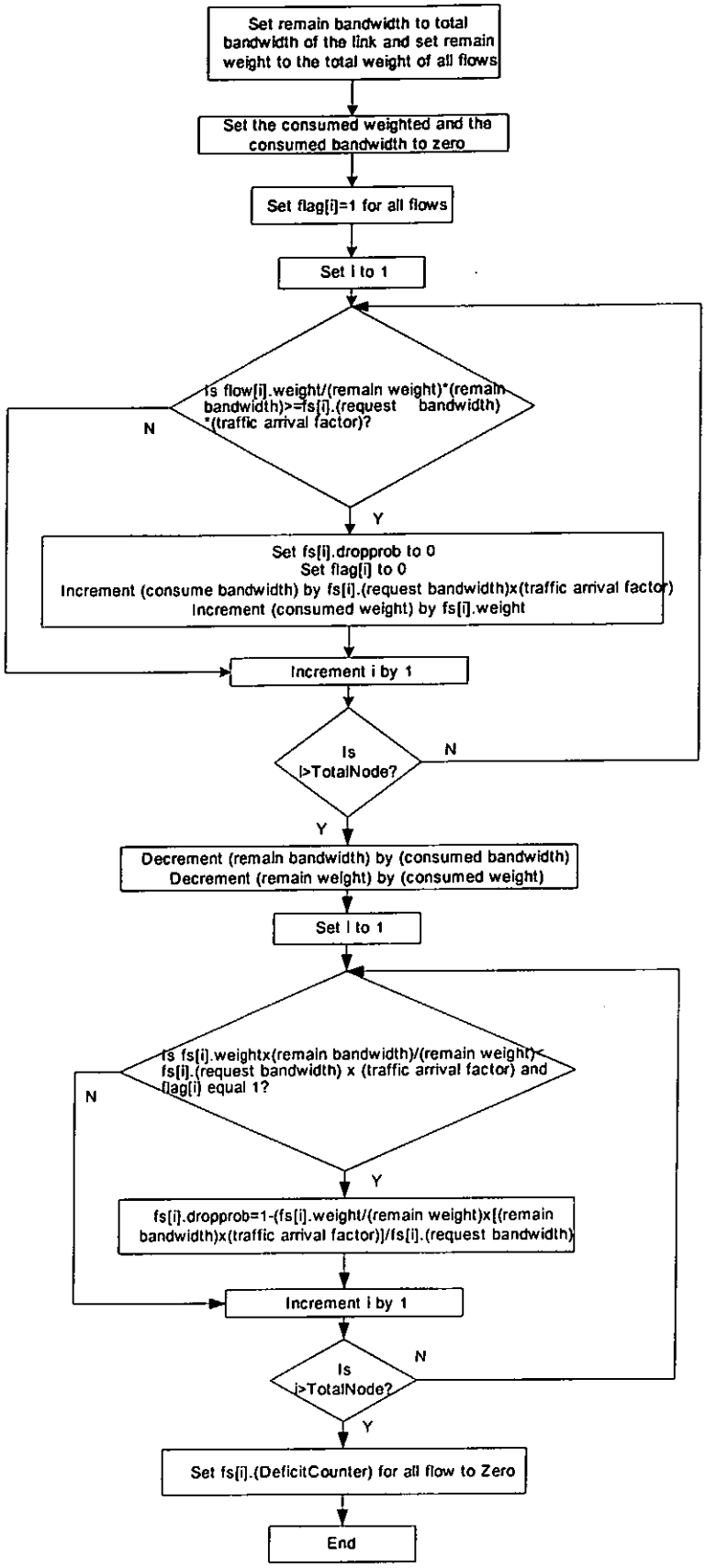

Figure 5-1 The flow chart represents the initalization module of the weighted deficit probability drop (WDPD)

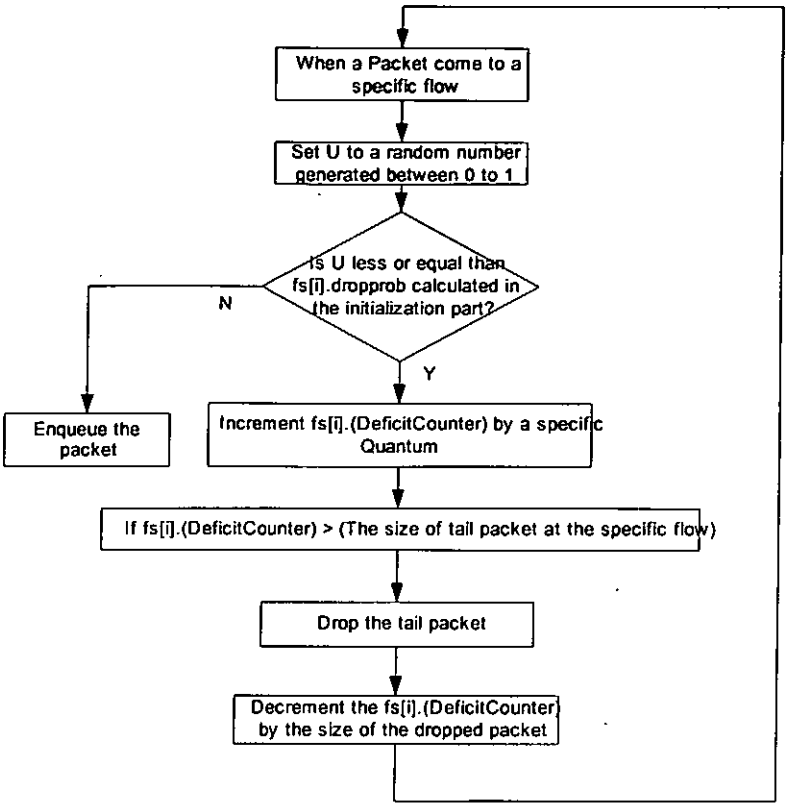

Figure 5-2 The flow cheat represents the en-queue and de-queue modules of the weighted deficit probability drop (WDPD)

For example, consider that there are thirty demand flows sharing a fixed 10Mb/s channel with all the requests at a same 1.4Mb/s, where three of the flows are with a weighting of ten times higher than that of the remaining twenty-seven flows. Using the proposed algorithm, the three flows of higher weighting were given a bandwidth of 1.4Mb/s each while the remaining twenty-seven can only have a limited bandwidth of 0.2148Mb/s each. The dropping probability for the higher and lower weighted flows are 0 and 0.8466 respectively. The probability-dropping algorithm can avoid congestion in the node and competing for bandwidth. Most of the calculations are done in the initialization stage so the computation power needed to en-queue and de-queue is relatively low.

When a new packet comes to a specific flow, it is marked to drop with a probability. The following is an example to illustrate the operation of WDPD.

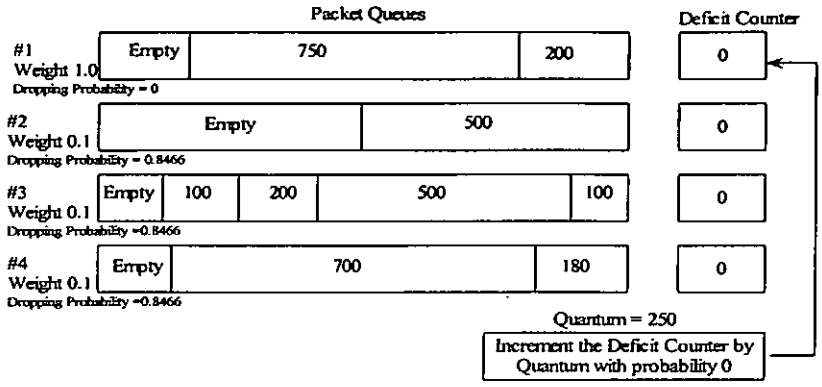

Figure 5-3 Weighted Deficit Probability Drop: Stage 1

In figure 5-3, the packet with size 750 just arrive flow 1, dropping probability is zero so the deficit counter remains the same.

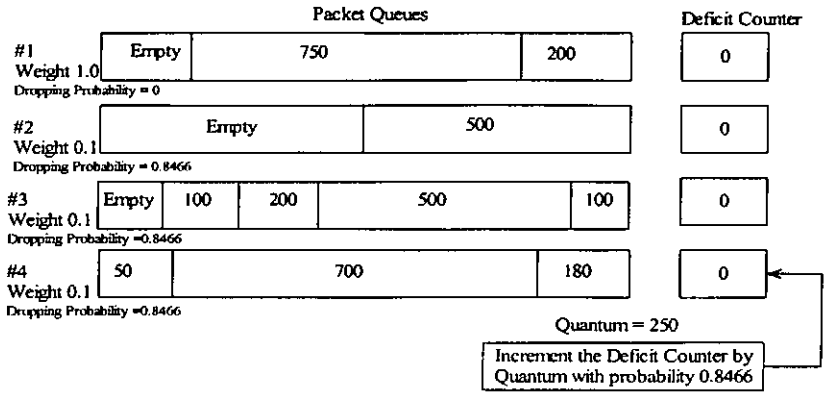

Figure 5-4 Weighted Deficit Probability Drop: Stage 2

In figure 5-4, the packet with size 50 just arrive flow 4, the dropping probability is 0.8466. The new come packet luckily within the other 15.34% is not marked to drop. The deficit counter remains the same.

In figure 5-5, the packet with size 50 just arrive flow 3, the dropping probability is 0.8466. It is unlucky, the packets are marked to drop. The deficit counter increases 250, deficit counter > 50+100 so the last 2 packets in flow 3 are marked to drop.

In the WDPD, the specific quantum number of each flow is chosen to be the mean packet size of that particular flow. This information is needed before the flow set-up, in order that

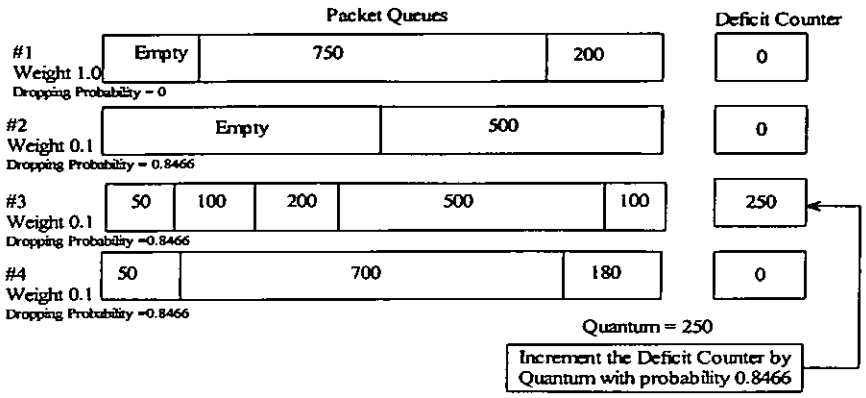

Figure 5-5 Weighted Deficit Probability Drop: Stage 3

the algorithm need can be operated without an added burden to calculate the mean packet size of each flow from the previous queuing record. The traffic arrival factor depends on how the packet is arrived.

The traffic arrival factors (**a**) are found by:

$$a = \frac{\overline{\mu_t}}{\mu} \tag{5.4}$$

μ is the expected throughput and $\overline{\mu_t}$ is the statistical average of the flow throughput in the arrival model of the tail drop queuing simulation.

The average throughput $\overline{\mu}$ can be found by:

$$\overline{\mu} = \frac{P_\lambda - P_d - P_s}{T_f - T_s} \tag{5.5}$$

T_s is the start time of the considered period, T_f is the finish time of the considered period. P_λ is the number of packet arrival to the flow in the considered period, P_d is the number of packet drop in the flow and P_s is the number of packet servicing in the queue. WDPD uses the traffic arrival factor (**a**) to estimate P_λ, P_s can be measured in the queue. It uses the dropping probability to control P_d and hence controls the average throughput of a specific

flow. Because the dropping probabilities of different weighted flows are different, it can control the fairness among different paid users accurately.

The implementations of packet scheduling algorithms are different, so the complexity is different as well. The complexity directly influences the cost of the hardware implementation especially in the high-speed networking environment. In the queuing algorithm, the complexity can be divided into three scopes: Timestamp Calculation Complexity, En-queuing Complexity and De-queuing Complexity.

The proposed algorithm is different from the other weighted queuing algorithm such as WFQ and WF^2Q+, it needs not to maintain the timestamp and the flow state variables, hence lower the computational requirement significantly. There is no special mechanism for WDPD to de-queue, all the outgoing flows compete for bandwidth. It is because the traffic is already shaped at the incoming queue. When the packet comes to a specific flow, the pointer will point to that flow. In the de-queue complexity, it is totally dependant from number of flows. That is why WDPD get the order of 1 in complexity.

The complexities of different queuing algorithms are compared below:

PGPS	WF^2Q+	LFVC [34]	WFQ	FBFQ [7]	SCFQ	DRR	VC	WDPD
O(N)	O(logN)	O(loglogN)	O(N)	O(logN)	O(logN)	O(1)	O(logN)	O(1)

Table 5-1 Complexity Comparison

In the table 5-1, N is the maximum number of connections that can share an output link. The WDPD and DRR with the same complexity are the low complexity algorithms.

The actual computational complexity in the hardware depends on the design implementation. In router, the firmware stores in flash memory. After it is booted up, the required part of the kernel will load into the DRAM. The firmware program is executed by a CPU in the router. For this kind of implementation, the computational complexity can be calculated by how many instructions that involved in en-queue and de-queue routines and

how many CPU cycle will consumed for each specific instruction. For our proposed algorithm WDPD, the main workload is on the initialization part. The instruction in the routine en-queue and de-queue modules are even less than the basic RED algorithms.

In level 3 switch implementation, the packet forwarding decision is made by the hardware gate logic in the Application Specific Integrated Circuit (ASIC) instead of made by the CPU. The actual computational complexity is really depends on the FPGA designs in the specific ASIC.. Although the actual computational complexity is high to predict, it really depends on the complexity of the routine parts in the algorithm itself. Our proposed algorithm is expected to have the actual computational complexity that higher than per flow tail drop queuing and basic round robin but less than the RED and VC. The computational complexity will be similar to DRR.

5.1 SIMULATION ON WDPD

Three different simulations are performed for WDPD performance study. All the simulations are based on the topology that introduced in the beginning of chapter 4 and figure 4-1. On the first simulation of this chapter which studies on the fairness among the same weighted flows. 30 flows with equal weight share a 10Mb/s link with 1500 packet buffer space and 10ms delay for all the links. Four different incoming traffic loadings - 0.35Mbps, 0.7Mbps, 1.4Mbps and 2.8Mbps are studied. The second and third simulation of this chapter study on the fairness among different weighted flows. In second simulation, 30 flows share a 10Mb/s link with 3 tenth flows (flow number 10, 20 and 30), which weighted 10 times more than the other 27 flows. Six different incoming traffic loadings - 0.35Mbps, 0.7Mbps, 1.4Mbps, 2.8Mbps, 5.6Mbps and 11.2Mbps are studied. On the third simulation, different weighted pattern are used. 60 flows share a 10Mb/s link with 30 even numbered flows, which weighted 10 times more than the other 30 odd numbered flows. Six

different incoming traffic loading – 0.175Mbps, 0.35Mbps, 0.7Mbps, 1.4Mbps, 2.8Mbps and 5.6Mbps are studied. WDPD is studied with 3 different traffic arrival factors, 1, 0.5 and the specific traffic arrival factor. In all these simulations, all algorithms are studied with four traffic arrival models and the specific traffic arrival factor are found below: (i) Poisson traffic of 4ms idle time and 4ms burst time with a traffic arrival factor of 0.7552, (ii) Heavy burst Poisson traffic of 4ms idle time and 100ms burst time with a traffic arrival factor of 0.9605, (iii) Constant bit rate IP phone model with a traffic arrival factor of 1.0013 and (iv) Constant bit rate with TCP model with a traffic arrival factor of 1.00114.

5.2 Simulation Results on WDPD and Fairness Comparison among same weighted flows

Delay Fairness, Drop Fairness and Throughput Fairness of ten queuing algorithms are compared. The proposed WDPD is simulated with three traffic arrival factors which are 0.5, 1 and the specific traffic arrival factor for the corresponding traffic type. All the flows in this simulation are using the same weight.

Table 5-2 indicates that WDPD with traffic arrival factor 0.5 is fairer than WDPD with the specific traffic arrival factor and with the factor 1.0. It out-performs VC, traditional tail drop, SFQ, DRR. It is fairer then RED and FRED in most of the case. The WFQ and WF^2Q+ are specially designed to provide fairness among the same weighted flows with high computational complexity. The WDPD is specially designed to provide a fair weighted proportion of a given bandwidth among different weighted flows with low computation complexity. The WDPD is less fair than the FQ, WFQ and WF^2Q+ among the same weighted flows but it out-performs many other low computation complexity algorithms in fairness. In table 5-3, the common statistical tools - standard deviation is used in the further analysis of the fairness among the same weighted flows. Even different

evaluation tools are used, the results are much or less the same as table 5-2. WDPD with traffic arrival factor 0.5 is the fairest among the other traffic arrival factor. It is out-perform VC, tail drop, SFQ, DRR and comparable to FRED, RED. It is still less fair then WFQ and WF^2Q+ in the fairness among the same weighted flows.

In table 5-4, the Jain's Fairness of Delay can be studied. The difference of fairness between all ten algorithms is very small. In general FQ is the fairest algorithm and SFQ is the worst. It can be concluded as follow: VC > Tail > WDPD 0.5 > FRED > RED > WDPD S > WDPD 1 > FQ > WFQ > WF^2Q+ > DRR > SFQ. In this simulation, VC is the most fair on delay, the packet dropping algorithms are out-performed the WFQ and WF^2Q+.

In the table 5-5, the packet drop fairness is studied. In the dropping fairness, the ranking of algorithms can be generally summarized as follow: FRED > RED > WDPD 0.5 > WF^2Q+ > FQ > WFQ > WDPD S > WDPD 1 > VC > TAIL > DRR > SFQ.

		VC	WFQ	WF2Q+	FQ	SFQ	DRR
	Mb/s						
Little	0.35	99.99	99.991	99.991	99.991	99.991	99.991
Burst	0.70	99.95	100.000	100.000	100.000	98.429	98.933
Poisson	1.40	99.93	100.000	100.000	100.000	91.418	97.467
	2.80	99.92	100.000	100.000	100.000	91.381	97.449
Heavy	0.35	99.997	99.997	100.000	99.997	99.997	99.997
Burst	0.70	99.417	100.000	100.000	100.000	91.082	97.451
Poisson	1.40	99.654	100.000	100.000	100.000	91.196	97.296
	2.80	99.409	100.000	100.000	100.000	91.290	97.272
Constant	0.35	99.984	99.984	99.999	99.984	99.811	99.984
Bit Rate	0.70	99.942	100.000	100.000	100.000	91.436	97.511
IP Phone	1.40	99.955	100.000	100.000	100.000	91.409	97.494
	2.80	99.961	100.000	100.000	100.000	91.411	97.503
Constant	0.35	99.982	99.982		99.982	99.982	99.982
Bit Rate	0.70	99.994	100.000		100.000	94.395	97.590
TCP	1.40	99.994	100.000		100.000	91.600	97.731
	2.80	99.994	100.000		100.000	91.600	97.731
Average		99.879	99.997	99.999	99.997	94.152	98.211

Table 5-2a Jain's Fairness of Throughputs among ten queuing algorithms

		Tail	FRED	RED	WDPD_1	WDPD_0.5	WDPD_S
	Mb/s						
Little Burst Poisson	0.35	99.991	99.9909	99.991	99.975	99.988	99.98834
	0.70	99.950	99.9964	99.930	99.836	99.989	99.93423
	1.40	99.854	99.994	99.917	99.605	99.853	99.74676
	2.80	99.839	99.9907	99.915	99.688	99.839	99.72807
Heavy Burst Poisson	0.35	99.997	99.997	99.997	99.987	99.996	99.99375
	0.70	99.417	99.999	99.854	99.914	99.998	99.93041
	1.40	99.288	99.998	99.925	99.890	99.991	99.90944
	2.80	98.771	99.995	99.897	99.857	99.988	99.87094
Constant Bit Rate IP Phone	0.35	99.984	99.988	99.982	99.980	99.985	99.979
	0.70	99.918	99.999	99.946	99.913	99.999	99.913
	1.40	99.877	99.999	99.884	99.928	99.988	99.928
	2.80	99.892	99.999	99.861	99.848	99.983	99.848
Constant Bit Rate TCP	0.35	99.982	99.9904	99.991			
	0.70	99.985	98.6803	96.836			
	1.40	99.985	99.0617	97.710			
	2.80	99.985	98.9978	97.428			
Average		99.795	99.792	99.442	99.868	99.966	99.898

Table 5-3b Jain's Fairness of Throughputs among ten queuing algorithms

		VC	WFQ	WF2Q+	FQ	SFQ	DRR
	Mb/s	$x10^{-3}$	$x10^{-3}$	$x10^{-3}$	$x10^{-3}$	$x10^{-3}$	$x10^{-3}$
Little Burst Poisson	0.35	2.52	2.52	2.52	2.52	2.52	2.52
	0.70	8.16	0.51	0.30	0.55	45.76	38
	1.40	10.02	0.45	0.24	0.45	111.22	59
	2.80	10.63	0.39	0.28	0.42	111.49	59.23
Heavy Burst Poisson	0.35	1.97	1.97	0.75	1.97	1.97	1.968
	0.70	28.02	0.37	0.31	0.37	113.55	59.18
	1.40	21.57	0.40	0.29	0.43	112.79	61.02
	2.80	28.23	0.39	0.22	0.42	112.15	61.31
Constant Bit Rate IP Phone	0.35	4.58	4.58	1.22	4.58	15.30	4.578
	0.70	8.82	0.22	0.22	0.24	111.13	58.5
	1.40	7.79	0.20	0.18	0.18	111.33	58.7
	2.80	7.20	0.00	0.14	0.14	111.31	58.6
Constant Bit Rate TCP	0.35	4.78	4.78		4.78	4.78	4.776
	0.70	3.32	0.43		0.53	103.36	66.66
	1.40	3.32	0.43		0.53	128.45	52.72
	2.80	3.32	0.43		0.53	128.45	52.72

Table 5-4a Standard Deviation of Throughputs among ten queuing algorithms

		Tail	FRED	RED	WDPD_1	WDPD_0.5	WDPD_S
	Mb/s	$x10^{-3}$	$x10^{-3}$	$x10^{-3}$	$x10^{-3}$	$x10^{-3}$	$x10^{-3}$
Little	0.35	2.52	2.52	2.52	4.04	2.86	2.86
Burst	0.70	8.16	2.05	9.21	8.56	3.90	7.19
Poisson	1.40	14.00	2.69	9.99	11.56	13.89	12.22
	2.80	14.71	3.35	10.10	9.62	13.96	11.99
Heavy	0.35	1.97	1.97	1.93	3.75	2.17	2.68
Burst	0.70	28.02	1.32	13.25	9.56	1.55	8.98
Poisson	1.40	30.99	1.49	9.48	10.80	3.41	10.19
	2.80	40.83	2.38	11.14	12.29	4.09	12.12
Constant	0.35	4.58	3.81	4.57	4.87	4.42	4.89
Bit Rate	0.70	10.51	0.97	8.08	10.07	1.41	10.06
IP Phone	1.40	12.85	1.01	11.85	9.08	4.02	9.06
	2.80	12.07	0.88	12.95	13.07	4.80	13.06
Constant	0.35	4.78	3.49	3.32			
Bit Rate	0.70	5.22	48.84	76.50			
TCP	1.40	5.22	41.11	64.75			
	2.80	5.22	42.56	68.80			

Table 5-5b Standard Deviation of Throughputs among ten queuing algorithms

		VC	WFQ	WF2Q+	FQ	SFQ	DRR
	Mb/s						
Little	0.35	99.634	99.634	97.956	99.634	99.682	99.690
Burst	0.70	99.998	99.953	99.954	99.954	89.893	93.433
Poisson	1.40	99.996	99.997	99.997	99.997	99.649	98.461
	2.80	99.999	99.999	99.999	99.999	99.986	98.293
Heavy	0.35	99.999	89.130	86.826	92.327	13.235	16.269
Burst	0.70	99.995	100.000	100.000	100.000	97.958	98.587
Poisson	1.40	99.999	100.000	100.000	100.000	99.989	98.251
	2.80	99.998	100.000	100.000	100.000	99.994	98.582
Constant	0.35	99.998	93.544	94.344	93.647	19.809	24.332
Bit Rate	0.70	99.999	100.000	99.999	99.999	99.156	98.531
IP Phone	1.40	100.000	100.000	100.000	100.000	99.991	98.319
	2.80	100.000	100.000	100.000	100.000	99.998	98.633
Constant	0.35	99.997	98.819		98.739	81.095	87.213
Bit Rate	0.70	99.998	100.000		100.000	93.735	97.222
TCP	1.40	99.998	100.000		100.000	93.666	97.262
	2.80	99.998	100.000		100.000	93.666	97.262
Average		99.975	98.817	98.256	99.018	86.344	87.521

Table 5-6a Jain's Fairness of Delay among ten queuing algorithms

		Tail	FRED	RED	WDPD_1	WDPD_0.5	WDPD_S
	Mb/s						
Little Burst Poisson	0.35	99.634	99.634	99.634	99.802	99.790	99.790
	0.70	99.998	99.999	99.999	99.669	99.997	99.819
	1.40	99.994	99.998	99.998	99.347	99.971	99.711
	2.80	99.993	99.997	99.997	99.259	99.926	99.648
Heavy Burst Poisson	0.35	99.999	99.999	99.998	99.664	99.999	99.996
	0.70	99.995	100.000	100.000	99.928	99.999	99.989
	1.40	99.997	99.999	99.999	99.905	99.995	99.973
	2.80	99.995	99.998	99.998	99.922	99.995	99.922
Constant Bit Rate IP Phone	0.35	99.998	99.999	99.999	99.996	99.997	99.996
	0.70	99.999	100.000	100.000	99.970	99.999	99.965
	1.40	99.999	100.000	100.000	99.917	99.995	99.921
	2.80	100.000	100.000	100.000	99.919	99.993	99.915
Constant Bit Rate TCP	0.35	99.997	99.805	99.719			
	0.70	99.997	99.978	99.970			
	1.40	99.997	99.982	99.983			
	2.80	99.997	99.958	99.978			
Average		99.974	99.959	99.954	99.775	99.971	99.887

Table 5-7b Jain's Fairness of Delay among ten queuing algorithms

		VC	WFQ	WF2Q+	FQ	SFQ	DRR
	Mb/s						
Little Burst Poisson	0.35	0.000	0.000	0.000	0.000	0.000	0.000
	0.70	10.176	10.233	10.286	10.255	86.037	72.605
	1.40	27.905	34.124	34.022	34.023	210.113	108.657
	2.80	51.086	58.454	58.391	58.413	218.140	120.551
Heavy Burst Poisson	0.35	0.000	0.000	0.000	0.000	0.000	0.000
	0.70	52.271	5.965	5.986	5.965	211.753	113.371
	1.40	36.202	18.646	18.819	18.614	209.525	117.660
	2.80	61.699	36.337	36.306	36.337	224.400	120.580
Constant Bit Rate IP Phone	0.35	0.000	0.000	0.000	0.000	28.008	0.000
	0.70	13.520	8.913	8.925	9.006	209.554	109.171
	1.40	20.640	16.785	16.906	16.906	210.968	108.710
	2.80	27.339	22.791	22.825	22.825	212.340	110.426
Constant Bit Rate TCP	0.35	0.000	0.000		0.000	0.000	0.000
	0.70	0.000	0.000		0.000	0.000	0.000
	1.40	0.000	0.000		0.000	0.000	0.000
	2.80	0.000	0.000		0.000	0.000	0.000

Table 5-8a Standard Deviation of packet drop among ten queuing algorithms

		Tail	FRED	RED	WDPD_1	WDPD_0.5	WDPD_S
	Mb/s						
Little	0.35	0.000	0.000	0.000	4.738	0.000	0.000
Burst	0.70	10.176	8.414	11.418	13.358	11.305	10.778
Poisson	1.40	22.868	31.959	18.513	28.775	24.670	26.023
	2.80	53.646	52.957	44.237	56.373	54.420	52.849
Heavy	0.35	0.000	0.557	0.525	5.574	0.000	2.393
Burst	0.70	52.271	7.909	26.123	16.940	6.636	15.573
Poisson	1.40	56.509	14.600	18.856	27.120	17.669	26.420
	2.80	88.266	28.567	31.222	32.366	32.821	35.586
Constant	0.35	0.000	3.429	5.392	4.920	0.000	4.945
Bit Rate	0.70	14.277	10.891	12.834	17.196	10.281	17.182
IP Phone	1.40	25.729	13.739	23.484	19.028	15.422	18.808
	2.80	33.928	16.937	31.567	23.173	18.377	23.011
Constant	0.35	0.000	1.619	1.414			
Bit Rate	0.70	0.000	1.203	1.939			
TCP	1.40	0.000	1.114	1.791			
	2.80	0.000	0.871	1.565			

Table 5-9b Standard Deviation of packet drop among ten queuing algorithms

5.3 FAIRNESS AMONG THE SAME WEIGHTED FLOWS IN THE WEIGHTED LOADING SITUATION

In this simulation, the fairness among the same weighted flow are compared in the weighted loading situation. Two different flow weighted patterns described in the section of "simulation on WDPD" are used.

In figure 5-6, 0 to 30 flows use the Poisson arrival model, 31 to 60 flows use the heavy Poisson arrival model and 61 to 90 flows use the Constant Bit Rate. The 9 tenth flows (the points that distribute at the upper part of the graph) are the flows which weighted 10 times more than the rest 81 lower weighted flows (the points that concentrate at the bottom part of the graph). WFQ and WF^2Q+ demonstrated a better fairness among the flows but with a much higher computation cost. The lower weighted flows of WFQ and WF^2Q+ show two completely flatness lines (the circle point and the cross point), that means they are very fair among the same weighted flows. WDPD with traffic arrival factor 0.5 is the most

fluctuated one. The WDPD with traffic arrival factor 1 and with the specific traffic arrival factor are in between the two groups. For the fairness between the higher weighted flows, it is hard to compare by using figure 5-6 with only nine reference points.

In figure 5-7, three traffic models are used in this simulation. Flows 0 to 60 uses the Poisson model, flows 61 to 120 uses the heavy burst Poisson model and flows 121 to 180 uses the UDP/CBR IP phone model. WF2Q+ and WFQ show two straight lines which imply they are very fair. WDPD is relatively more fluctuated. It is hard to tell which traffic arrival factor is the fairest among the WDPD algorithms. The numerical values need to be used for quantified comparison.

In Table 5-6, the average row is calculated by averaging the Jain's Fairness Indexes of all high weighted flows and all low weighted flows in the specific weighted pattern. The Jain's Throughput Fairness Index is almost the same with the two different weighted patterns. The fairness order of the algorithm is the same in both patterns. WF^2Q+ is the algorithm with the best fairness index followed by WFQ. In WDPD fairness, traffic arrival factor 0.5 is the best followed by the specific traffic arrival factor and then traffic arrival factor 1.

Actually it is hard to rank the delay fairness order in table 5-7, the difference in fairness is very small. It can be summarized that WDPD 0.5 > WDPD S > WDPD 1 > WFQ > WF^2Q+ in simulation 2 and WDPD 0.5 > WFQ > WDPD S > WF^2Q+ > WDPD 1 in simulation 3. This fairness order shows that WDPD 0.5 is out perform WFQ and WF^2Q+ in the delay fairness. WDPD with other traffic arrival factors are comparable to WFQ and WF^2Q+ in delay fairness. This simulation shows that the weighted pattern will affect the delay fairness.

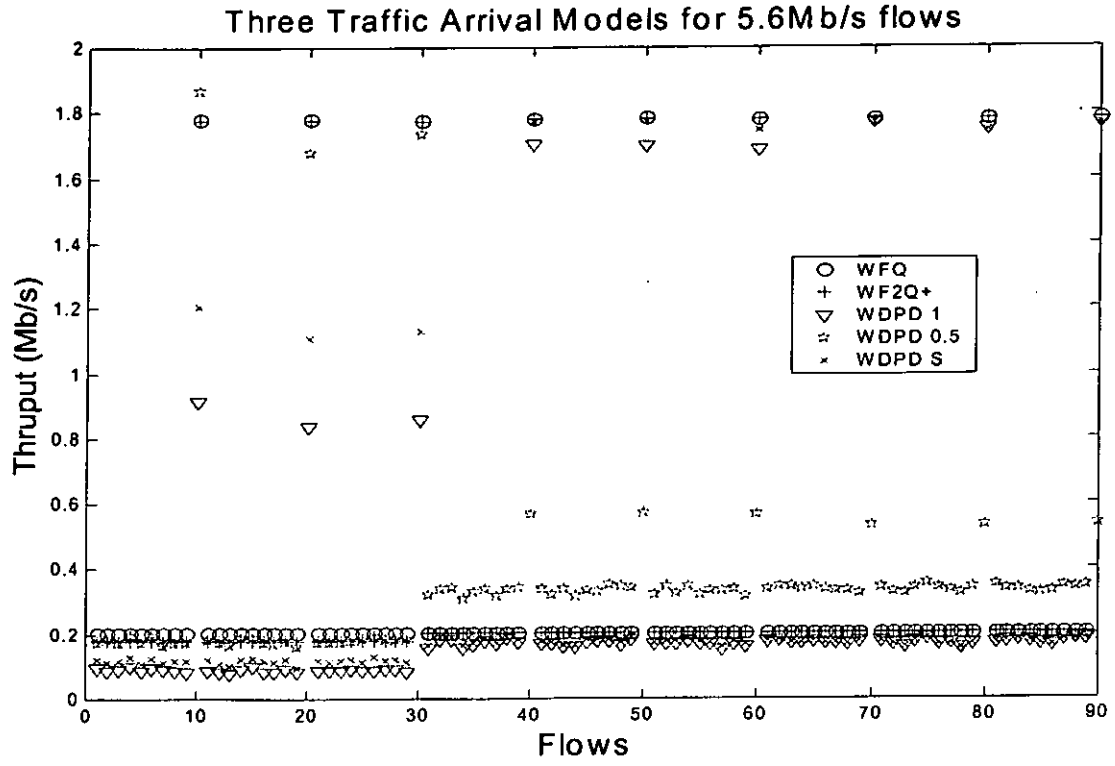

Figure 5-6 Simulation 2 on the Three Traffic Models for the 3 weight algorithms

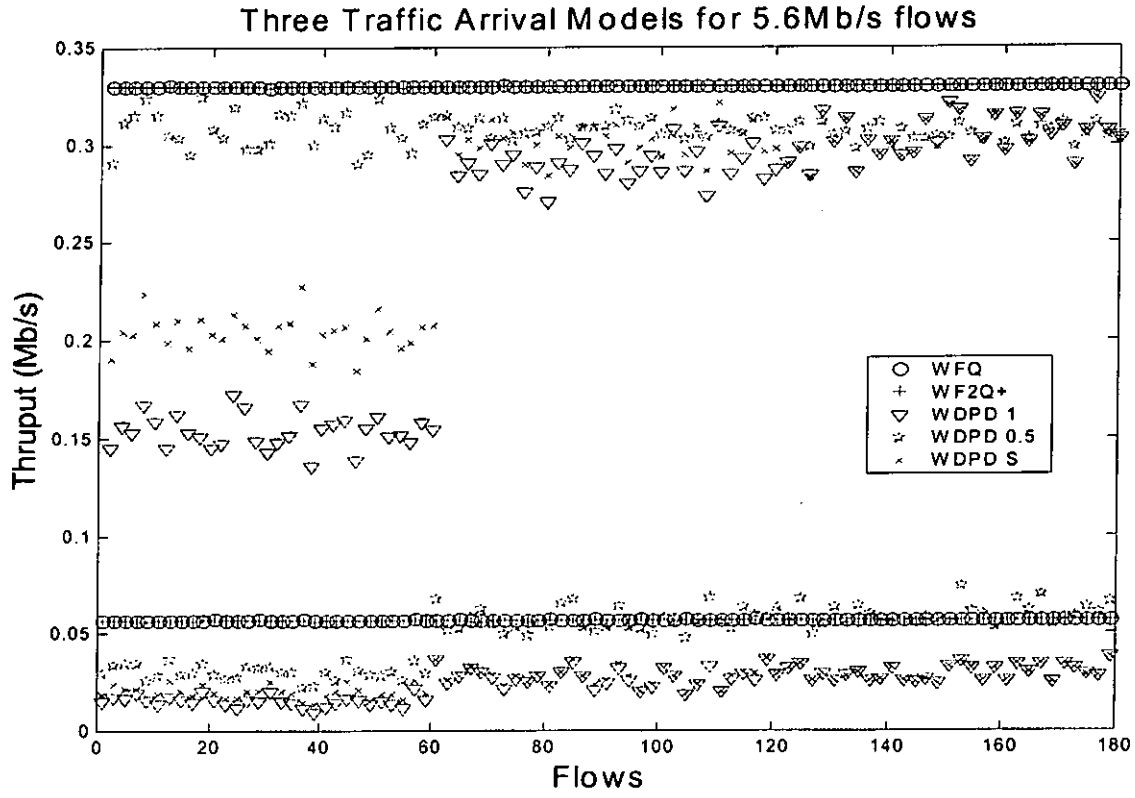

Figure 5-7 Simulation 3 on the Three Traffic Models for the 3 weight algorithms

		Simulation 2				
		WFQ		WF2Q+		WDPD_1
		High weighted flow	Low weighted flow	High weighted flow	Low weighted flow	High weighted flow
	Mb/s	$x10^3$	$x10^3$	$x10^3$	$x10^3$	$x10^3$
Little Burst Poisson	0.35	99.9996247	99.99009636	99.9996247	99.99009636	99.98634107
	0.7	99.9834978	99.99984078	99.9834978	99.99990992	99.99128586
	1.4	99.99983966	99.99009636	99.99506129	99.99996285	99.94546347
	2.8	99.98774165	99.99979492	99.98774165	99.99990654	99.9419396
	5.6	99.99976595	99.99959763	99.99994576	99.99988677	99.84712733
	11.2	99.99996207	99.99957076	99.99998601	99.99991207	99.99240442
Heavy Burst Poisson	0.35	99.99977437	99.99657019	99.99977437	99.99927575	99.99894182
	0.7	99.99960453	99.9998715	99.99960453	99.99996539	99.99724202
	1.4	99.99958928	99.99982335	99.99958928	99.99994135	99.99556901
	2.8	99.99996208	99.99969453	99.99996203	99.99991207	99.99747299
	5.6	99.9999681	99.99974216	99.99995809	99.99991207	99.99773738
	11.2	99.9999504	99.99974136	99.99995832	99.9998954	99.9886415
Constant Bit Rate IP Phone	0.35	99.9989542	99.98562039	99.9989542	99.99865553	99.99423327
	0.7	99.99566295	99.99996921	99.99566295	100	99.99339785
	1.4	99.99634459	99.99989107	99.99634459	100	99.99855809
	2.8	100	99.99997517	100	100	99.99015091
	5.6	100	100	99.99999801	100	99.99674921
	11.2	100	99.99985538	100	100	99.99367394
Average			99.9977776		99.99841377	
		Simulation 3				
		WFQ		WF2Q+		WDPD_1
		High weighted flow	Low weighted flow	High weighted flow	Low weighted flow	High weighted flow
	Mb/s	$x10^3$	$x10^3$	$X10^3$	$x10^3$	$x10^3$
Little Burst Poisson	0.175	99.9950307	99.99490705	99.9950307	99.99725847	99.99584363
	0.35	99.98897148	99.99990689	99.98897148	99.99882719	99.96830482
	0.7	99.99979165	99.99920052	99.99986555	99.99857064	99.78044137
	1.4	99.99991722	99.99920052	99.99994875	99.99857064	99.71845102
	2.8	99.99993728	99.99897415	99.9999837	99.9991975	99.69205204
	5.6	99.9999071	99.99897415	99.99995666	99.99896898	99.69402567
Heavy Burst Poisson	0.175	99.99511761	99.99625873	99.99511761	99.99972667	99.99594798
	0.35	99.99987491	99.99897415	99.99993545	100	99.96642107
	0.7	99.9999058	99.99944692	99.99993708	99.99857064	99.8613576
	1.4	99.99993465	99.99897415	99.9999837	99.99857064	99.78899463
	2.8	99.99992772	99.99920052	99.99999156	99.99896898	99.83010273
	5.6	99.9999315	99.99858138	99.99998371	99.9991975	99.89902844
Constant Bit Rate IP Phone	0.175	99.96906931	99.97066749	99.96906931	99.9999396	99.97674247
	0.35	99.99992325	100	99.99995313	100	99.95831048
	0.7	99.99995822	100	99.9999837	100	99.91999253
	1.4	99.9999765	100	99.99998254	100	99.84147021
	2.8	99.99998376	100	99.99998371	100	99.77823533
	5.6	100	100	99.99997644	100	99.88070373
Average			99.99723404		99.99816728	

Table 5-10a Jain's Throughput Fairness Index comparison of the same weighted flow in the flow weighted situation

			WDPD_0.5		WDPD_S	
	Mb/s	Low weighted flow x10^3	High weighted flow x10^3	Low weighted flow x10^3	High weighted flow x10^3	Low weighted flow x10^3
Little Burst Poisson	0.35	99.97402693	99.98634107	99.98878487	99.98634107	99.98878487
	0.7	99.77667107	99.99479767	99.98874346	99.99128586	99.91663814
	1.4	99.20781167	99.97918117	99.84150924	99.94546347	99.59957892
	2.8	99.52180542	99.98833995	99.75904846	99.99093338	99.62383949
	5.6	99.55058478	99.79567937	99.76172395	99.86630678	99.58256857
	11.2	99.46261998	99.9966672	99.78739097	99.9988358	99.68262313
Heavy Burst Poisson	0.35	99.98574489	99.99894182	99.99565031	99.99894182	99.99351495
	0.7	99.85579886	99.99907969	99.99821042	99.99724202	99.89662492
	1.4	99.79905699	99.99852784	99.99280637	99.99556901	99.82390127
	2.8	99.6750008	99.99942957	99.95741167	99.99606493	99.65979813
	5.6	99.74228235	99.99939111	99.88916837	99.99547989	99.75862349
	11.2	99.83462805	99.99725473	99.87797818	99.99140477	99.84744962
Constant Bit Rate IP Phone	0.35	99.97879165	99.99423327	99.98403856	99.99423327	99.9783828
	0.7	99.84455951	99.99956748	99.9984512	99.99339785	99.84020237
	1.4	99.81528104	99.99987891	99.98355762	99.99855809	99.80860453
	2.8	99.73303109	99.9988503	99.95651274	99.98965807	99.72815693
	5.6	99.82259612	99.99793338	99.9451339	99.99705107	99.81868421
	11.2	99.58421621	99.99902135	99.87231588	99.99331943	99.57314545
Average		99.85587325		99.95282089		99.88447801

			WDPD_0.5		WDPD_S	
	Mb/s	Low weighted flow x10^3	High weighted flow x10^3	Low weighted flow x10^3	High weighted flow x10^3	Low weighted flow x10^3
Little Burst Poisson	0.175	99.96748816	99.99584363	99.99212541	99.99584363	99.99212541
	0.35	97.75781274	99.99347734	99.99113541	99.98892676	99.45063145
	0.7	98.19811031	99.9800574	98.95402697	99.90020572	98.67577332
	1.4	97.32477212	99.89330182	98.31687419	99.77429444	97.63106232
	2.8	97.23645378	99.8216255	97.98457468	99.78908777	97.62538196
	5.6	96.88363237	99.89504683	98.49457947	99.80332938	97.36217376
Heavy Burst Poisson	0.175	99.96347405	99.99594798	99.99556493	99.99594798	99.98276885
	0.35	98.50855577	99.99875188	99.99705669	99.97537587	98.29448601
	0.7	97.01929858	99.99795091	98.90092542	99.85513108	97.30300314
	1.4	97.93086035	99.98922238	99.13913423	99.80246138	97.88740601
	2.8	98.20981641	99.98485272	99.04966106	99.85380172	98.34125958
	5.6	96.86429316	99.98170375	98.92871954	99.91212048	97.02459397
Constant Bit Rate IP Phone	0.175	99.92555729	99.97674247	99.97453289	99.97674247	99.92377832
	0.35	98.89206093	99.99444092	99.99126388	99.95816868	98.89603726
	0.7	98.61540143	99.99683095	99.02053498	99.9217873	98.59962132
	1.4	97.53582001	99.9910634	99.03164523	99.84138082	97.5095255
	2.8	98.33249857	99.98015105	99.09581918	99.77961441	98.32332305
	5.6	98.44567049	99.97941521	99.22015853	99.88091356	98.43415123
Average		99.03216673		99.59790997		99.14617322

Table 5-11b Jain's Throughput Fairness Index comparison of the same weighted flow in the flow weighted situation

		Simulation 2				
		WFQ		WF2Q+		WDPD_1
	Mb/s	High weighted flow	Low weighted flow	High weighted flow	Low weighted flow	High weighted flow
Little Burst Poisson	0.35	99.80608297	99.63841905	99.4123208	98.55886375	99.97845494
	0.7	99.92889234	99.9659645	99.68902994	99.96648152	99.88936066
	1.4	99.99281168	99.99794775	98.90844275	99.99796216	99.99602414
	2.8	95.4961125	99.9993463	99.58839842	99.99954888	99.97181477
	5.6	99.99730918	99.99965428	99.99681038	99.99964496	99.81571346
	11.2	99.99876724	99.99972773	99.99888476	99.99989838	99.52311576
Heavy Burst Poisson	0.35	99.98939402	89.71726845	99.81455361	89.0024121	99.92956511
	0.7	99.89784523	99.99956781	99.52066946	99.99968212	99.99399073
	1.4	99.98783578	99.99989599	95.94936196	99.99994556	99.99387918
	2.8	99.99991925	99.99985601	99.99993928	99.99997566	99.99980544
	5.6	99.99995339	99.99988522	99.99995775	99.9999877	99.97394268
	11.2	99.99983244	99.99988578	99.99987778	99.99998633	99.95858717
Constant Bit Rate IP Phone	0.35	99.9711306	95.09547177	99.67070629	95.92187279	99.99813694
	0.7	99.95845053	99.9998123	99.09100612	99.99962956	99.99876619
	1.4	99.97790852	99.9999859	99.93553706	99.99994431	99.99718583
	2.8	99.99992652	99.99999249	99.99986012	99.99996719	99.99645946
	5.6	99.99999556	99.99999517	99.99999087	99.99997477	99.99536111
	11.2	99.99981848	99.99992025	99.99981638	99.99997526	99.93335324
Average			99.42818286		99.30613658	
		Simulation 3				
		WFQ		WF2Q+		WDPD_1
	Mb/s	High weighted flow	Low weighted flow	High weighted flow	Low weighted flow	High weighted flow
Little Burst Poisson	0.175	99.7552816	99.53209697	97.72835093	86.36257345	99.388818
	0.35	99.8652097	99.99801575	99.41344802	99.99855635	99.65342153
	0.7	99.98270489	99.99920015	99.98207244	99.99871975	99.64303628
	1.4	99.9988338	99.9989659	99.99881086	99.99904144	99.63629712
	2.8	99.9994562	99.99901021	99.99942345	99.99944691	99.51010114
	5.6	99.99969029	99.99944113	99.99970296	99.99958011	98.94966079
Heavy Burst Poisson	0.175	99.41858876	93.20642449	97.31420338	94.65107677	99.4651632
	0.35	99.77487555	99.9992479	99.78448459	99.99304139	99.81297854
	0.7	99.99970279	99.99957849	99.99970903	99.9992418	99.89787643
	1.4	99.99992358	99.99941038	99.99991853	99.99929744	99.91255463
	2.8	99.99996298	99.99953484	99.99997882	99.99943884	99.84846954
	5.6	99.99997054	99.99922857	99.99998613	99.99958358	99.87192197
Constant Bit Rate IP Phone	0.175	99.81811884	97.37920094	97.61677812	99.97678971	99.99742092
	0.35	99.77061348	99.99974433	99.76567107	99.99950053	99.98748366
	0.7	99.9996788	99.99997013	99.99946084	99.99980945	99.97264323
	1.4	99.99999069	99.99999852	99.9999242	99.99988634	99.95501344
	2.8	99.9999985	99.99999796	99.99996523	99.99990598	99.91380554
	5.6	99.99999429	99.9999945	99.99997658	99.99991524	99.8959534
Average			99.68032379		99.23825751	

Table 5-12a Jain's Delay Fairness Index comparison of the same weighted flow in the flow weighted situation

			WDPD_0.5		WDPD_S	
	Mb/s	Low weighted flow	High weighted flow	Low weighted flow	High weighted flow	Low weighted flow
Little Burst Poisson	0.35	99.78372373	99.99190757	99.79291717	99.99190757	99.79291717
	0.7	99.76641768	99.99967966	99.99632974	99.98445822	99.85553598
	1.4	99.08321559	99.99424378	99.96961374	99.89701357	99.60360342
	2.8	99.32963727	99.99936772	99.94999082	99.99130213	99.56652128
	5.6	98.55280452	99.99198317	99.83078844	99.95581033	99.25137967
	11.2	98.15144694	99.99475681	99.89283051	99.93460936	98.39350565
Heavy Burst Poisson	0.35	99.69236719	99.99882141	99.99867619	99.99884174	99.99606861
	0.7	99.89116971	99.99995637	99.99931324	99.99988907	99.97145973
	1.4	99.56548783	99.9995394	99.99629307	99.99911198	99.97149951
	2.8	99.69981283	99.99998699	99.99275603	99.99910871	99.9231996
	5.6	99.7960895	99.99919156	99.99341769	99.99621662	99.96212215
	11.2	99.62763326	99.99762184	99.98841283	99.97170147	99.60695942
Constant Bit Rate IP Phone	0.35	99.99434768	99.99723491	99.99714594	99.99813441	99.99489312
	0.7	99.94473193	99.99962657	99.99936046	99.99888016	99.94824385
	1.4	99.86693145	99.99989162	99.99512799	99.99749794	99.85939325
	2.8	99.86757198	99.9994671	99.99139634	99.99716159	99.86263438
	5.6	99.89390326	99.99900234	99.99526505	99.99623596	99.89216757
	11.2	99.51814996	99.99982204	99.9939456	99.93229054	99.51268191
Average		99.74913775		99.98154671		99.85013771

			WDPD_0.5		WDPD_S	
	Mb/s	Low weighted flow	High weighted flow	Low weighted flow	High weighted flow	Low weighted flow
Little Burst Poisson	0.175	98.95146473	99.45364846	99.18439781	99.45364846	99.18439781
	0.35	97.73565152	99.99694118	99.9962453	99.98560866	99.62792443
	0.7	96.0188656	99.99476935	99.83110037	99.90403266	98.50736578
	1.4	93.56094461	99.9630419	99.61250595	99.83135732	97.50315106
	2.8	93.15792564	99.92899056	99.3625609	99.43122244	97.27368787
	5.6	91.52964272	99.93240327	99.5575157	99.33162103	96.65419171
Heavy Burst Poisson	0.175	99.48033505	99.99795659	99.99826292	99.99259643	99.99363009
	0.35	99.38243951	99.99950017	99.99841442	99.9850302	99.61659945
	0.7	97.77444065	99.99875701	99.97647218	99.94692414	99.15241215
	1.4	98.75861312	99.99424762	99.97236307	99.96645146	99.60880346
	2.8	98.30752742	99.99284271	99.96423773	99.91187978	99.30612872
	5.6	98.79678655	99.99250023	99.97450503	99.9258217	99.1519647
Constant Bit Rate IP Phone	0.175	99.99197931	99.99056833	99.99335798	99.99796575	99.99240409
	0.35	99.62047	99.9978536	99.99688142	99.98714268	99.53400656
	0.7	99.33846262	99.99861196	99.9795447	99.96331919	99.16930808
	1.4	98.39641344	99.9953662	99.97360145	99.95712702	98.34885299
	2.8	98.72863013	99.99174455	99.96265083	99.90825449	98.64322024
	5.6	98.91445724	99.98835564	99.97133168	99.89702145	98.96302217
Average		98.71549081		99.90316802		99.37800267

Table 5-13b Jain's Delay Fairness Index comparison of the same weighted flow in the flow weighted situation

The complexities of WF^2Q+, WFQ and WDPD are O(logN), O(N) and O(1) respectively. N is the maximum number of connections that can share an output link. The WDPD is a more acceptable alternative in terms of fairness among different flows and the required computational cost.

5.4 Simulation Results on WDPD AND Fairness Comparison Among Different Weighted Flows

In the commercial environment, a fair weighting among different weighted flows are the major interest. The weighted fairness on WFQ, WF^2Q+ and the proposed WDPD will be compared in this section. In the fair weighted simulations, two different weighted patterns are considered. The same topology in figure 4-1 is used but with different weighted parameters. The results shown in this section base on the second and third simulation that described in the section "simulation on WDPD" in this chapter. In simulation 2, the higher weighted flow traffic will be saturated by the weight constrict when the traffic loading is up to $\frac{10}{27\times1+3\times10}\times10Mbps=1.7544Mbps$. In simulation 3, the situation point for the higher weighted traffic will be $\frac{10}{30\times1+30\times10}\times10Mbps=0.30303Mbps$. That means the bandwidth usage will be saturated much faster in the simulation 3. The normalized weight is the reference for the weight comparison. It can be calculated by the formula:

$$\mu_n=\frac{\overline{\mu_H}}{\overline{\mu_L}} \tag{5.6}$$

μ_n is the normalized weight, $\overline{\mu_H}$ is the average throughput of the higher weighted flows during the simulation period and $\overline{\mu_L}$ is the average throughput of the lower weighted flows during the simulation period. The normalized weights of the four algorithms will be

compared with the three traffic arrival models (without the TCP model) in four loading conditions.

Simulations are performed to find out the normalized weight of the flows in the two different weighted patterns. In simulation 2, when all 30 flows request 0.35Mb/s bandwidth, the 3 higher weighted flows can get 0.35Mb/s and the 27 lower flows share the remaining bandwidth and get 0.3315Mb/s each, the normalized weight is 1.0558. When all flows request 0.7 Mb/s bandwidth, then the 3 higher weighted flows can get 0.7Mb/s and the 27 lower weighted flows can get 0.2926Mb/s. The normalized weight is 2.3923. When all flows request 1.4Mb/s, the normalized weight is 6.5172. When all flows request 2.8Mb/s, the 3 higher weighted flows cannot get the whole 2.8Mb/s, this is because the request bandwidth is larger than the calculated weighted bandwidth 1.754Mb/s, it can only get 1.754Mb/s and the 27 flows can get 0.1754Mb/s, the normalized weight is 10. All flows request larger than 1.754Mb/s will still get 1.754Mb/s bandwidth. It is restricted by the weighted bandwidth setting which sharing the insufficient link. Beyond this point, the nominal weight is 10. This point is called the fairness saturation point. The normalized weights are 1.0558, 2.3923, 6.5172, 10, 10 and 10 with bandwidth request at 0.35Mb/s, 0.7Mb/s, 1.4Mb/s, 2.8Mb/s, 5.6Mb/s and 10.2Mb/s respectively. In simulation 3, the normalized weights are 1.10526, 10, 10, 10, 10 and 10 with bandwidth request at 0.175Mb/s, 0.35Mb/s, 0.7Mb/s, 1.4Mb/s, 2.8Mb/s and 5.6Mb/s respectively. These two set of values form the nominal models for the two simulations. Two different weighted patterns are researched in order to find out how the normalized weights are affected by the sharing patterns of the incoming flows. By comparing the nominal model with the other four algorithms the weighted fairness among different weighted flows can be compared. In the figures 5-8 to 5-13, the traffic arrival factor of the WDPD can be adjusted to get the similar weighted performance as WFQ and WF^2Q+.

In figures 5-8 to 5-13, the fairness between different weighted flows are shown. The WDPD with traffic arrival factor 1 and WDPD with the specific traffic arrival factor are more likely conforming to the nominal model in both of the weighted patterns. In contrast to WFQ and WF^2Q+, WDPD 1 and WDPD S immune to the change of the weighted pattern and the traffic loading. The fairness saturation points remain the same and keep tracking to the nominal model closely. The WDPD 1 and WDPD S get faster response to

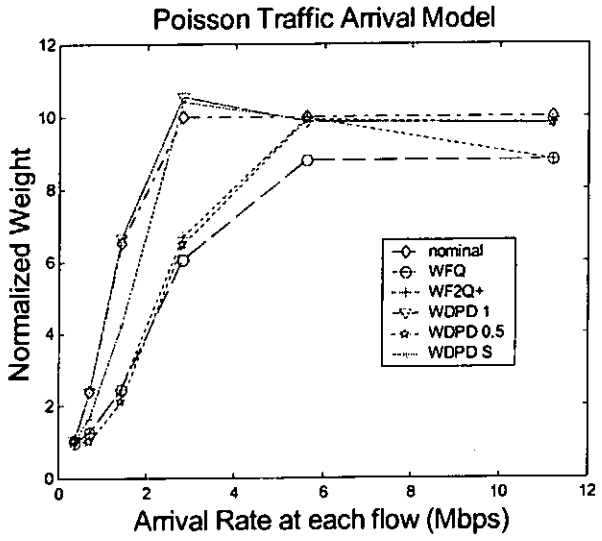

Figure 5-8 Normalized Weight of the Poisson Model in simulation 2

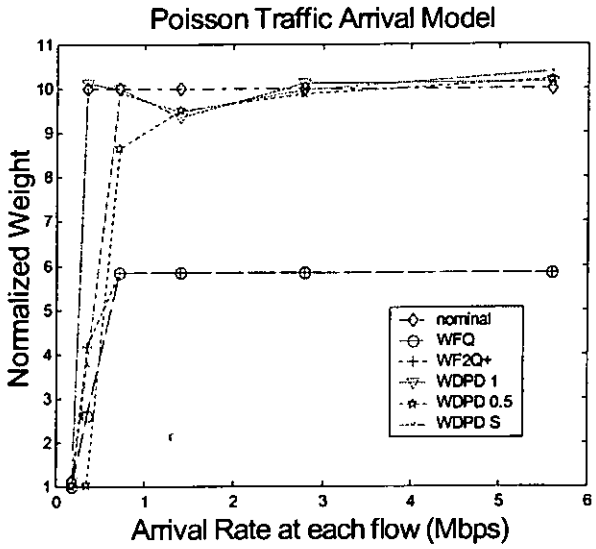

Figure 5-9 Normalized Weight of the Poisson Model in simulation 3

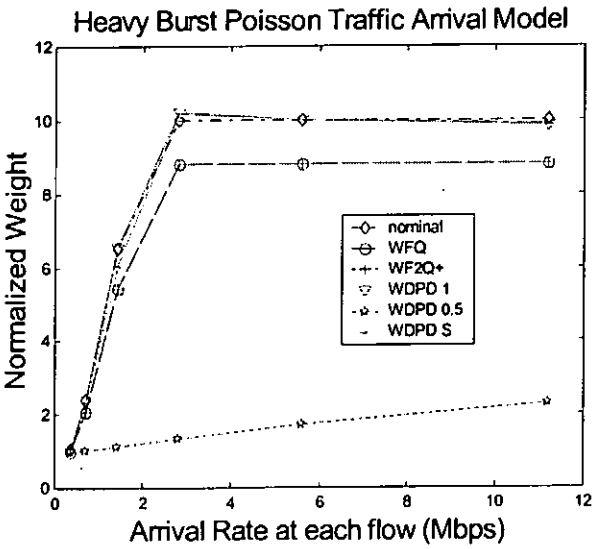

Figure 5-10 Normalized Weight of the Burst Poisson Model simulation 2

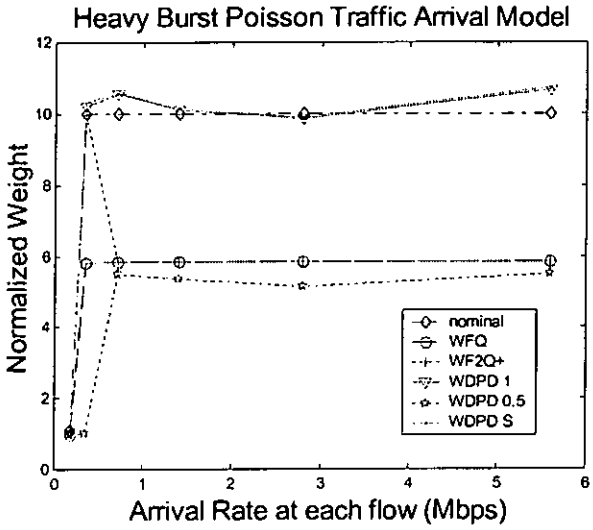

Figure 5-11 Normalized Weight of the Burst Poisson Model simulation 3

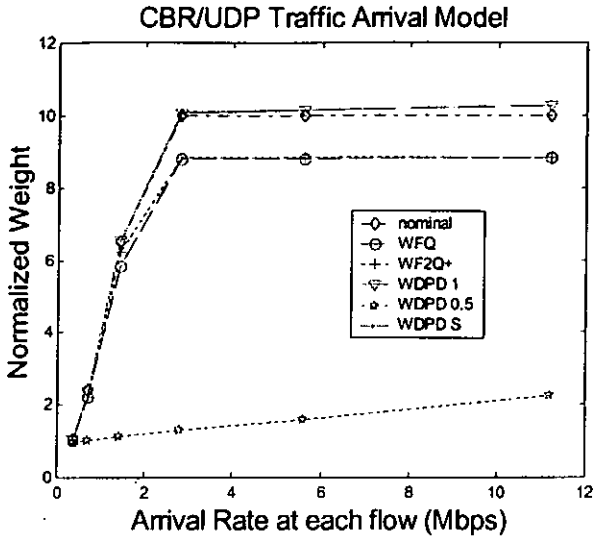

Figure 5-12 Normalized Weight of the IP Phone Model simulation 2

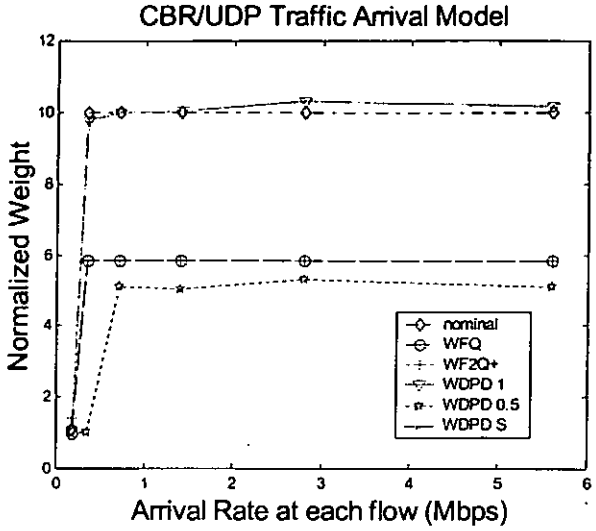

Figure 5-13 Normalized Weight of the IP Phone Model simulation 3

the congestion in the sense of the fairness between different weighted flows. WFQ and WF^2Q+ have difficulty in keep tracking of the nominal line. The expected fairness saturation point is never reached even in the extreme overloading condition. The congestion responses of WFQ and WF^2Q+ are faster on the Heavy Burst Poisson model, they reach the fairness saturation point more quickly at lower loading condition. WDPD with traffic arrival factor 0.5 is tried hard to conform to the nominal model in most of the cases but it is quite good in tracking the nominal lines in the Poisson traffic model which most applications fall in this category. It can be noticed that the weighted fairness of WFQ, WF^2Q+ and WDPD with traffic arrival factor 0.5 are heavily affected by the traffic loading and the weighted pattern of the incoming flows.

In the two weighted patterns of the three algorithms with three traffic arrival models and six different loading conditions, the WDPD with traffic arrival factor 1.0 is the best performer in the fairness among different weighted flows. It is strictly conformed to the nominal model.

5.5 NETWORK POWER AND QUEUE SIZE SIMULATION STUDY

The network power [4, 10] is explained in chapter 4. It is the ratio of throughput to delay. It gives a good indication of network efficiency and the optimal throughput of a network system. The network power simulation study is divided into 3 parts. The first and the second parts are using the topology and network parameters of the simulation 2 and 3. The last part of the study is based on the topology and network parameters that used in the fairness comparison on the same weighted flows in simulation 1.

In table 5-8, the averaged powers are the averaging network power that across all the simulated flows in the whole simulation period. It is easy to notice that, the network power trends to lower when the incoming traffic loading is increasing. Actually when the loading becomes higher, the occupied queue size will be increased and then the delay time will be increased as well. The network power is in terms of the delay so,it will be decreased. In the figure 3-3 of chapter 3, it can be noticed that when the input loading increases, the network power is also increasing in the first phrase. After the maximum power P_{max} is reached and input loading is greater than λ_{opt}, the network power is beginning to drop. This forms the second phrase of the graph. In simulation 3, most of the data in the table fall into the second phrase except the network power of the WDPD algorithm. It can be observed that the WFQ and WF^2Q+ reach the optimal input loading before 0.175Mbps and the network power lowers dramatically after the maximum power has been reached. In simulation 2, the WFQ and WF^2Q+ reach the optimal input loading at about 1.4Mbps. By using the active dropping of the predicted overloading queues, the WDPD algorithm can effectively delay the optimal input loading point at about 2.8Mbps and increase the network power at least ten times then the WF^2Q+. The network powers decrease gradually after passed the optimal input loading point. That means more overloaded incoming traffic loading can be

handled by WDPD and with a higher network performance. WDPD with traffic arrival factor of 1 is the best performer in the network power comparison. At the last part of the network power simulation, more algorithms are compared with the proposed WDPD. Some similar active dropping algorithms such as FRED and RED are also compared.

In table 5-9, it can be seen that WDPD obviously out-perform all the other algorithms in the network power comparison. It also out performs the FRED and RED which sharing the same idea on the active dropping of the predicted overloading sessions. Because WDPD drops significant amount of predicted overloading packets. The network power is significantly increased and the required queue sizes are significantly reduced. This can save the hardware construction cost on high bandwidth link which consumes a lot of buffers. In the cross-bar switch core design, the buffer is needed at each intersection between every input port and output port. Ten input and ten output ports will need hundred of buffer spacing. So the scalability is a concern for the traditional queuing implementation in the cross-bar switch core. The proposed algorithm will solves this problem.

In the table 5-10, it can be noticed that the queue size is generally required more in the Constant Bit Rate IP phone model and require more in the Burst Poisson Traffic model but require less in the Poisson model. In TCP, the sliding window congestion control is used, the queue size is also controlled and kept in low usage level. It can be noticed that WDPD with traffic arrival factor 1 requires the lowest queue size, WDPD with the specific traffic arrival factor requires the second lowest queue size. WDPD with traffic arrival factor 0.5 requires almost the same queue size as the traditional tail drop. It can be noticed that, the required queue size of WDPD is heavily affected by the traffic arrival factor.

Average Power	MB/S^2	Simulation 2					
Traffic Loading	MB/S	0.35	0.7	1.4	2.8	5.6	11.2
Poisson	WFQ	629.3211	199.9735	339.2421	93.072626	1.858933	1.798581
	W2FQ+	478.3411	203.9016	340.8483	27.472378	1.821552	1.796574
	WDPD_1	584.0947	759.9898	998.7238	1039.6021	837.487	732.7807
	WDPD_0.5	542.0203	0.749242	1.428307	3.6855159	11.06521	12.63579
	WDPD_S	542.0203	458.2892	606.6238	682.07269	513.3897	425.971
Heavy	WFQ	169.2155	305.7283	497.6696	1.7942108	1.767347	1.762604
Burst	W2FQ+	161.4266	307.0242	519.6871	1.7909256	1.766111	1.761616
Poisson	WDPD_1	288.3182	148.7495	96.8947	120.86969	68.68897	198.0431
	WDPD_0.5	7.311483	0.653854	0.664694	0.7070752	0.869064	1.135142
	WDPD_S	24.61936	17.16976	11.06846	12.858016	9.89322	125.9893
CBR	WFQ	166.8532	261.2825	421.6191	1.8006485	1.762871	1.761923
	W2FQ+	163.8325	314.9795	265.5712	1.7803613	1.760774	1.759942
	WDPD_1	12.97147	26.76775	47.50649	45.987506	24.26222	217.2898
	WDPD_0.5	1.738085	0.651465	0.659868	0.6999489	0.807101	1.103965
	WDPD_S	14.70986	28.60433	52.82144	49.875668	26.50613	217.8931

Table 5-14a Network Power Comparison in two weighted patterns.

Average Power	MB/S^2	Simulation 3					
Traffic Loading	MB/S	0.175	0.35	0.7	1.4	2.8	5.6
Poisson	WFQ	287.2625	189.7432	0.29895	0.277003	0.272368	0.270468
	W2FQ+	252.1259	142.236	0.297921	0.276386	0.271864	0.269986
	WDPD_1	164.2021	283.1239	378.3162	443.1524	445.9445	407.0559
	WDPD_0.5	142.946	0.205434	0.345514	0.533711	1.782983	2.612005
	WDPD_S	142.946	44.56195	197.3945	286.2283	277.7116	258.2814
Heavy	WFQ	214.0565	0.435491	0.278508	0.271862	0.269774	0.268905
Burst	W2FQ+	219.3215	0.428443	0.277898	0.271372	0.269268	0.268427
Poisson	WDPD_1	105.9602	103.4742	49.79088	47.09259	38.43021	50.26276
	WDPD_0.5	3.593681	0.18849	0.282004	0.281347	0.281661	0.284399
	WDPD_S	10.66032	18.89356	13.8384	7.66109	7.763743	16.57837
CBR	WFQ	197.0175	0.371413	0.277775	0.271595	0.269572	0.268735
	W2FQ+	223.8541	0.368422	0.277129	0.271034	0.269055	0.268215
	WDPD_1	5.175163	15.38787	16.07135	3.368188	17.35089	7.425901
	WDPD_0.5	0.852759	0.187672	0.277167	0.277767	0.28198	0.280364
	WDPD_S	6.332234	16.84129	18.04003	3.651393	17.97158	8.03211

Table 5-15b Network Power Comparison in two weighted patterns.

Average Power	Mb/S^2				
Traffic Loading	Mb/S	0.35	0.7	1.4	2.8
Poisson	DRR	552.3103	5.962071	1.340313	2.3150969
	FQ	552.6197	0.719302	0.647316	0.6339029
	FRED	552.6197	2.546015	2.14011	2.1034719
	RED	552.6197	1.994362	1.926124	1.9101283
	SFQ	552.41	7.818174	0.719539	0.7002709
	Tail	552.6197	0.712167	0.643493	0.6310868
	VC	552.6197	0.712167	0.643541	0.6312001
	WDPD_1	587.1285	770.7617	864.787	912.19882
	WDPD_0.5	542.0203	0.750484	1.204997	3.357861
	WDPD_S	542.0203	454.9368	567.527	610.33421
	WF2Q+	526.142	0.719131	0.646785	0.6339389
	WFQ	552.6197	0.719475	0.647299	0.6339378
Heavy Burst Poisson	DRR	76.1845	1.206765	2.2021	3.8500369
	FQ	10.4708	0.650241	0.632834	0.6276564
	FRED	9.713874	2.119756	2.07517	2.0575583
	RED	9.637743	1.932347	1.911715	1.9056135
	SFQ	59.76319	0.765811	0.700988	0.6962415
	Tail	9.637722	0.647943	0.63174	0.627241
	VC	9.637722	0.647943	0.631779	0.6272501
	WDPD_1	286.8539	122.1735	86.9166	95.882448
	WDPD_0.5	7.311483	0.65391	0.659152	0.6609482
	WDPD_S	26.64592	10.17235	15.01074	28.428186
	WF2Q+	7.846004	0.650243	0.632867	0.6276208
	WFQ	11.01577	0.650234	0.632833	0.6275921
CBR/UDP	DRR	11.31904	1.248467	2.274263	3.9651193
	FQ	1.984931	0.64799	0.631937	0.6269856
	FRED	2.935659	2.116931	2.072445	2.0546246
	RED	3.036623	1.928551	1.910287	1.9048323
	SFQ	11.38174	0.732202	0.700166	0.6957101
	Tail	1.815469	0.645702	0.630956	0.6266875
	VC	1.815469	0.645714	0.630984	0.6267068
	WDPD_1	16.26796	16.2175	28.86685	31.603335
	WDPD_0.5	1.738085	0.651234	0.656066	0.6573888
	WDPD_S	19.27504	18.76587	30.67185	33.75081
	WF2Q+	1.873782	0.648075	0.63194	0.6269856
	WFQ	1.994386	0.648166	0.631905	0.6270471
CBR/TCP	DRR	8.006976	2.238358	1.837078	1.8370779
	FQ	2.185343	2.154983	2.154983	2.1549832
	FRED	5.092463	5.094315	5.09163	5.1674679
	RED	5.19682	5.323805	5.335601	5.3787754
	SFQ	8.829502	2.477279	2.488605	2.4886048
	Tail	2.077374	2.158706	2.158695	2.1586949
	VC	2.076008	2.156211	2.156211	2.1562113
	WFQ	2.181365	2.15495	2.15495	2.1549499

Table 5-16 Network Power Comparison between the queuing algorithm.

Maximum used Queue Size (packets)					
Traffic Loading	Mb/s	0.35	0.7	1.4	2.8
Poisson	DRR	7	1500	1500	1500
	FRED	7	481	576	694
	RED	7	528	667	709
	SFQ	7	1301	1334	1335
	Tail	7	1499	1499	1499
	VC	7	1499	1499	1499
	WDPD_1	6	4	4	3
	WDPD_0.5	6	1476	1334	529
	WDPD_S	6	6	11	5
	WF2Q+	7	1492	1495	1496
	WFQ	7	1491	1496	1496
Heavy Burst Poisson	DRR	189	1500	1500	1500
	FRED	173	562	632	689
	RED	182	653	695	718
	SFQ	189	1334	1335	1339
	Tail	189	1499	1499	1499
	VC	189	1499	1499	1499
	WDPD_1	9	21	35	34
	WDPD_0.5	209	1496	1491	1486
	WDPD_S	58	127	131	109
	WF2Q+	239	1495	1499	1499
	WFQ	189	1497	1499	1500
CBR/UDP	DRR	961	1499	1499	1499
	FRED	390	575	724	740
	RED	367	663	725	741
	SFQ	689	1334	1334	1336
	Tail	961	1498	1498	1498
	VC	961	1498	1498	1498
	WDPD_1	91	165	88	89
	WDPD_0.5	990	1495	1490	1485
	WDPD_S	71	152	84	87
	WF2Q+	989	1496	1499	1499
	WFQ	961	1496	1498	1499
CBP/TCP	DRR	417	498	498	498
	FQ	251	498	465	468
	FRED	251	468	465	468
	RED	269	474	473	472
	SFQ	373	498	498	498
	Tail	498	498	498	498
	VC	498	498	498	498
	WFQ	498	498	498	498

Table 5-17 The maximum used queue size comparison between the queuing algorithm.

5.6 CONCLUSION

The simulations show WDPD can provide an acceptable fairness on the same weighted flows. It is fairer than the other low computation algorithm. It out performs the WFQ and WF^2Q+ on the fairness among different flows. This is very important in the commercial issues. Users want to ensure the paid bandwidth. Other same weighted users fair share the rest of the remaining bandwidth. Furthermore the network power of WDPD is generally higher than all the other algorithms in the simulation. It predicts the overloading flows and drops packet actively. It requires comparable less queue size so the hardware construction cost is relatively low.

In the simulations of this chapter, it can be observed that WDPD with a low traffic arrival factor 0.5 can provide a reasonable fairness among the same weighted flows but it is the worst to provide fairness among different weighted flows. WDPD with a high traffic arrival factor 1.0 can provide a downgraded fairness among the same weighted flows and it is strictly conformed to the nominal model. It can provide a good fairness among different weighted flows. The simulations conclude that the traffic arrival factor of WDPD can be adjusted to balance between the performance in the fairness among same weighted flows and fairness among different weighted flows. WDPD is a flexible algorithm that the required fairness parameters can be customized by adjusting the traffic arrival factor.

CHAPTER 6 WEIGHTED DEFICIT ROUND ROBIN

The other algorithm – Weighted Deficit Round Robin (WDRR) is also proposed in this thesis. It is an improved version of Deficit Round Robin (DRR) which can provides weighted network throughput in the QoS enabled network node. The Weighted Deficit Round Robin (WDRR) uses a simple mechanism to provide the weighted control in the node. The weight sharing ability of the algorithm is comparable to Weighted Fair Queuing (WFQ) and Worst-case Fair Weighted Fair Queuing (WF^2Q+). The computational power required is much lower than WFQ and WF^2Q+. It can be used to provide fair queuing among different flows in the Constant Bit Rate TCP traffic condition. It is a good choice to enable QoS with weighted function in the network node.

Round Robin is an old fashioned scheduling algorithm to share resources. It was employed to schedule the packet in different flows of a network node. In the computer network, many data link layers support variable packet size, the old fashioned round robin becomes unfair in the variable packet size environment. Deficit Round Robin was proposed to solve the variable packet size problem by introducing the quantum concept. In Deficit Round Robin, each flow keeps a deficit counter. The deficit counter increments Y at each round. Y was called the quantum, which is a credit to de-queue the backlogged packet. At the de-queuing moment, each flow can de-queue the packet in the queue until all the deficit counter credit has been used up. Large packet consumes more credit and small packet consumes less credit. This algorithm claims to be fair in the variable packet size environment.

The Weighted Deficit Round Robin (WDRR) extends the idea of quantum credit and provides the weighted function with low computational complexity. Different flows get different weight, the WDRR assigns different quantum values to different flows at each round. The quantum value of each flow is positively proportional to the assigned weight. At each round, the scheduler processes the packet flow by flow. The deficit counter of each flow is incremented by a quantum value which positive proportion to the weight of the specific flow. Different flows get different quantum sizes. Because the algorithm extends the quantum idea from the deficit round robin, it provides weighted throughput in a varying packet size environment. It inherits the fairness of the Deficit Round Robin Algorithm and provides the weighted feature in the network node. The proposed algorithm consists of three major components, namely, the initialization, the en-queuing and the de-queuing components. They are summarized as the following:

Initialisation:

```
for (i = 1; i <=totalnode; i++)
{
    DeficitCounter[i]=0;
    Quantum[i]=DefaultQuantum * weight[i];
}
```

En-queuing:

```
flowId=ExtractFlow(pkt);
if (ExistsInActiveList(flowId) == FALSE)
{
   InsertActiveList(flowId);
   DeficitCounter[flowId]=0;
}
If no free buffers left
   FreeBuffer();
Enqueue(flowId,pkt);
```

De-queuing:

```
While (TRUE) do
{
    If ActiveList is not empty
    {
        Remove head of ActiveList[i];
        DeficitCounter[i]+=Quantum[i];
        While ((DeficitCounter[i]>0) and (Queue[i] not empty)) do
        {
                PacketSize=Size(Head(Queue[i]));
                If (PacketSize<=DeficitCount[i])
                {
                        Dequeue(Queue[i]);
                        DeficitCounter[i]-=PacketSize[i];
                } Else {
                 break the loop;
                }
        }
        if (Empty(Queue[i]))
                DeficitCounter[i]=0;
        Else
                InsertActiveList[i];
    }
}
```

By using the above algorithm, the weighted features can be provided to network node.

Figures 6-2 to 6-4 show the detail operation of the algorithm.

In figure 6-2, there are four queues with the weight 1.0, 0.5, 0.2 and 0.8. Four queues are filled up with different sized packets. At the initial stage, all the Deficit Counters are filled up with zeros. When the Round Robin Pointer is pointing to flow 1, the calculated Quantum Size which proportional to the weight is added to the Deficit Counter [1].

In figure 6-3, the packet with size 200 in flow 1 is left the queue and the deficit counter is decreased by 200 and becomes 300. A packet with size 250 from flow 1 is added to the tail of the queue. The Round Robin Pointer is now pointing to flow 2. The weight of flow 2 is 0.5, the calculated quantum size is 250 and it is added to the Deficit Counter of the second flow.

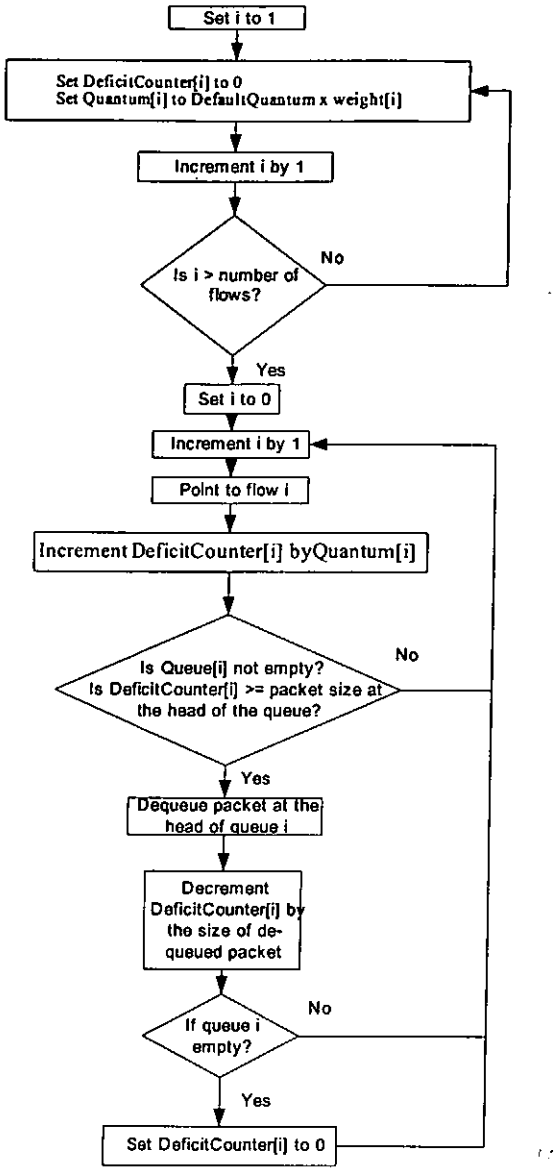

Figure 6-1 The flow chart represents the operation of the weighted deficit round robin (WDRR)

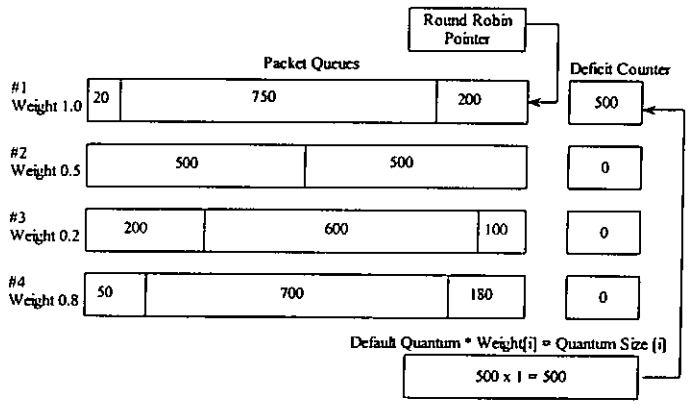

Figure 6-2 Weighted Deficit Round Robin: Stage 1

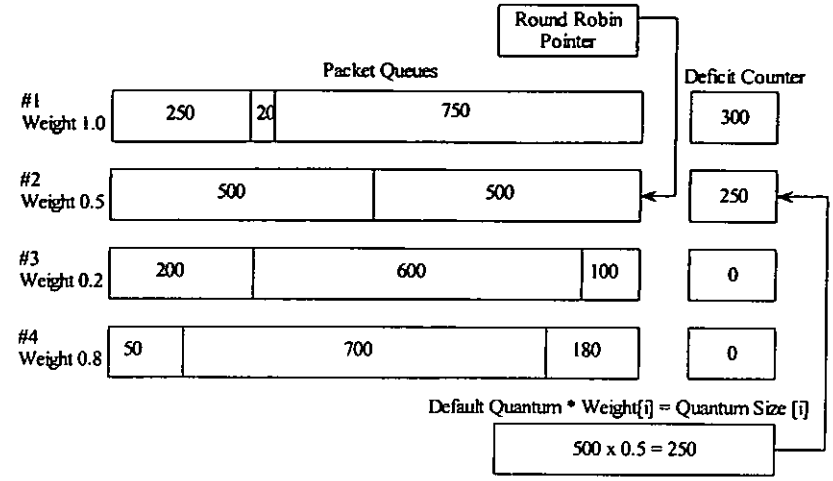

Figure 6-3 Weighted Deficit Round Robin: Stage 2

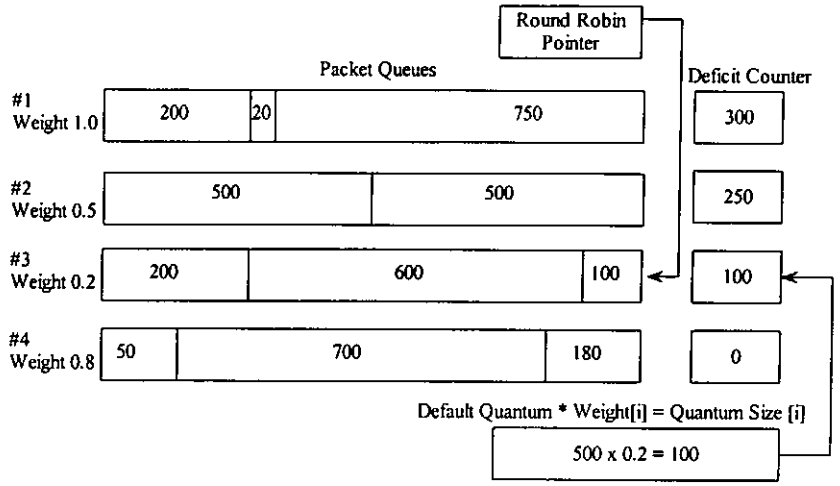

Figure 6-4 Weighted Deficit Round Robin: Stage 3

In figure 6-4, the packet with size 500 in flow 2 is still in the queue. It is because the deficit counter is smaller than the head packet in flow 2. Flow 2 does not gain enough credit for de-queuing the head packet. It needs to wait the second round and gains enough credit to de-queue the head packet. The Round Robin Pointer is now pointing to flow 3, the weight of flow 3 is 0.2 and the calculated quantum size is 100. It is added to the deficit counter [3] and the head packet of flow 3 just gain enough credit for de-queuing.

The Deficit Round Robin is a subset of the Weighted Deficit Round Robin with equal weights for different flows.

The complexities of different queuing algorithms are compared below:

PGPS	WF^2Q+	LFVC [34]	WFQ	FBFQ [7]	SCFQ	DRR	VC	WDPD	WDRR
O(N)	O(logN)	O(loglogN)	O(N)	O(logN)	O(logN)	O(1)	O(logN)	O(1)	O(1)

Table 6-1 Complexity Comparison

The proposed WDRR get the complexity that equal to WDPD, DRR and RR. It gets O(1) complexity as there is no calculation required to compute the packet which will be de-queue next. The queuing pointer will make the straight forward decision. The de-queue complexity is totally independent from the number of flows.

The actual computational complexity that implement in the hardware is expected to be a little bit more than DRR and RR. It is because the per-flow weight and per-flow quantum is introduced.

6.1 SIMULATION OF THE FAIRNESS AMONG SAME WEIGHTED FLOWS

For WDRR, the fairness among the same weighted flows are same as the DRR. It is because Deficit Round Robin is a subset of the Weighted Deficit Round Robin with equal weights for different flows. The DRR is not special fairer than the other algorithms so the WDRR as well. WDRR is modified from DRR and aimed to provide weighted ability which easy to implement with low computational requirement. The fairness comparison among the same weighted flows can refer to the DRR column of table 5-2 (Throughput Fairness), 5-4 (Delay Fairness) and 5-5 (Drop Fairness) in chapter five. It is hard to compare the fairness by using Jain's fairness index (table 5-2) because the difference is small. Standard deviation (table 5-3) is used for the WDRR fairness comparison purpose. The simulation shows WDRR is not good at the same weighted flow fairness but it is compared to be good in the TCP CBR traffic model. In this model, it out-performs SFQ, RED and comparable to FRED. WDRR is favourable to use in the TCP CBR traffic flow.

6.2 SIMULATION OF THE FAIRNESS AMONG DIFFERENT WEIGHTED FLOWS

The simulation setup in this section is same as the simulation 2 and 3 in chapter 5 which WDPD is studied.

In figure 6-5, 0 to 30 flows use the Poisson arrival model, 31 to 60 flows use the heavy Poisson arrival model and 61 to 90 flows use the Constant Bit Rate. The 9 tenth flows (the points that distribute at the upper part of the graph) are the flows which weighted 10 times more than the rest 81 lower weighted flows (the points that concentrate at the bottom part of the graph). The graph shows that WDRR is quite fluctuated among the same weighted flows. The weighted performance is not as good as WFQ and WF2Q+ but WDRR can provide limited weighting function and QoS to the congestion node. This is highly demanded by the network management industry.

In figure 6-6, three traffic models are used in this simulation. Flows 0 to 60 use the Poisson model, flows 61 to 120 use the heavy burst Poisson model and flows 121 to 180 use the UDP/CBR IP phone model. There are two sets of points that distribute in the higher and lower regions of the graph respectively. The points of WFQ and WF^2Q+ can join together and form twp straight lines, that means they are very fair among the same weighted flows. The WDRR points are more fluctuated especially in the higher weighted flows regions. The fluctuation range of the higher weighted flows is much larger. WDRR is less fair among the same weighted flows. The points of WDRR are distributed up and down around the WFQ and WF^2Q+ lines equally. That means averaging of the point will form a straight line with the throughput about the same value as WFQ and WF^2Q+ lines. The weighted performance of the WDRR is similar to WFQ and WF^2Q+.

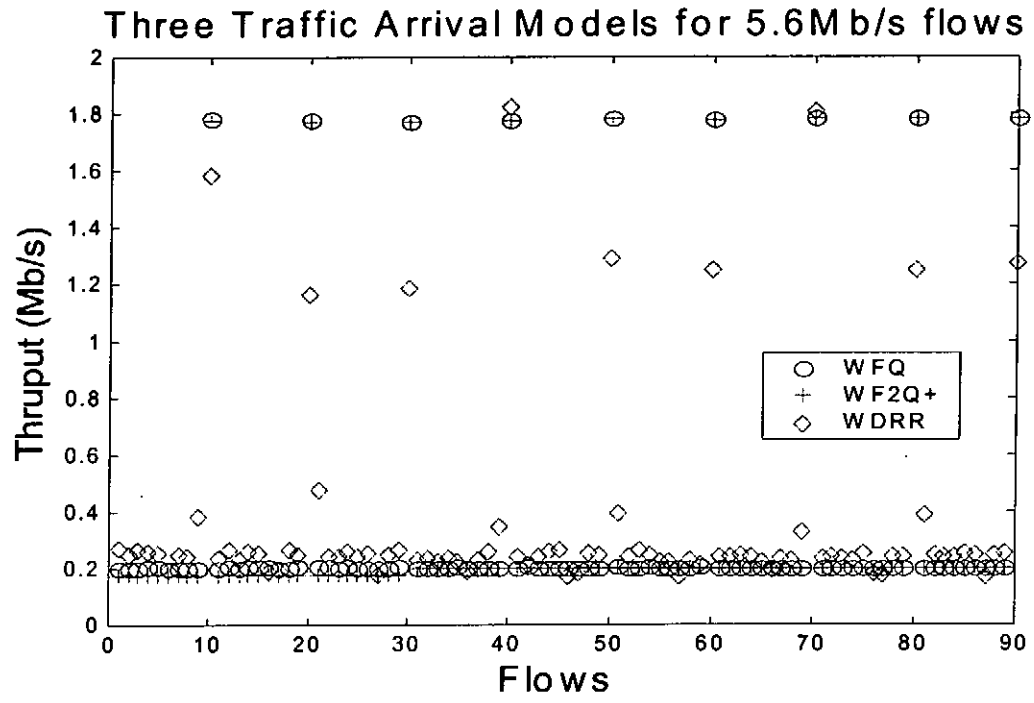

Figure 6-5 Simulation 2 on the Three Traffic Models for the 3 weight algorithms

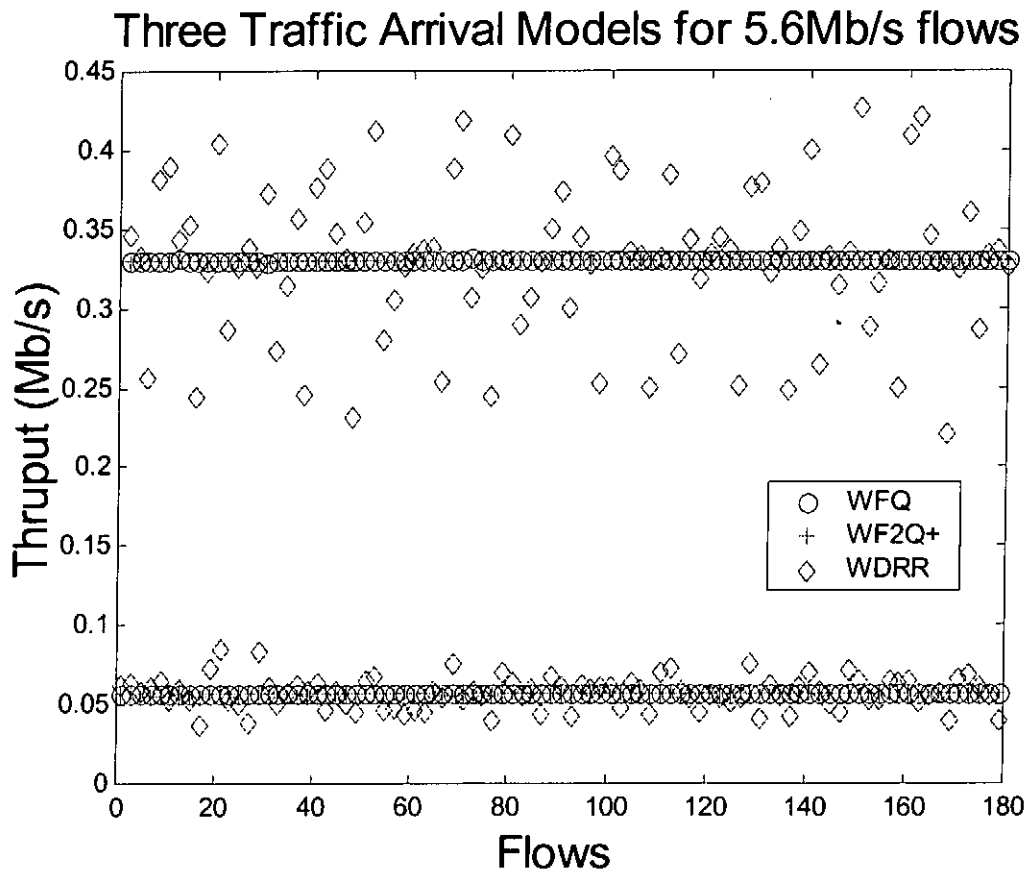

Figure 6-6 Simulation 3 on the Three Traffic Models for the 3 weight algorithms

In table 6-2, it can be noticed that the standard deviation of the higher weighted flows is higher than the standard deviation of the lower weighted flows. This property occurs across all the loading levels and all the simulation models. This agrees with the figure 6-6 where the fluctuation range of the higher weighted points is larger than the lower weighted points.

WDRR	Poisson		Burst Poisson		CBR	
Input Load (Mbps)	S. D. Hi.	S. D. Low	S. D. Hi.	S. D. Low	S. D. Hi.	S. D. Low
0.175	0.001068	0.00108	0.001194	0.001046	0.003136	0.003057
0.35	0.017458	0.04078	0.018998	0.028318	0.021736	0.024575
0.7	0.028252	0.014808	0.039247	0.010662	0.041383	0.008832
1.4	0.042283	0.008933	0.048544	0.009333	0.047986	0.009481
2.8	0.049845	0.009356	0.050381	0.009513	0.048152	0.010043
5.6	0.047974	0.01123	0.04692	0.009586	0.050816	0.009403

Table 6-2 Standard Deviation comparison on the throughput among the higher weighted flows and the lower weighted flows in simulation 3.

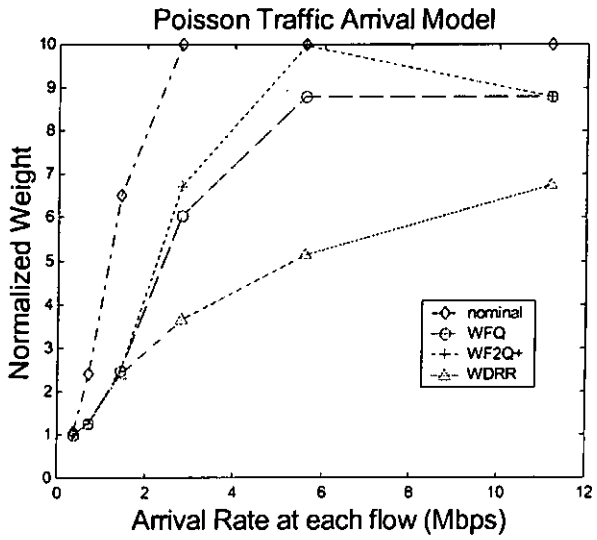

Figure 6-7 Normalized Weight of the Poisson Model in simulation 2

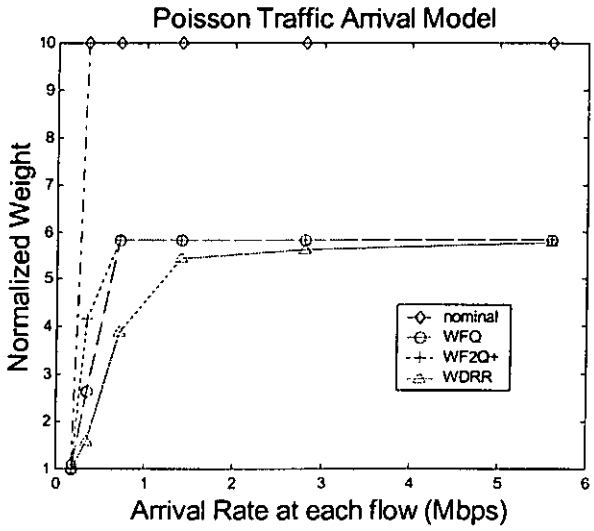

Figure 6-8 Normalized Weight of the Poisson Model in simulation 3

The weighted performance of the modified DRR is shown in figure 6-7 to 6-12. In these figures, the WDRR are studied with 3 incoming traffic arrival models and 6 loading conditions. In the simulation 3 with 60 weighted flows, all algorithms in the graph cannot track to the nominal model. That means all algorithms cannot provide good weighted fairness. WDRR can keep tracking with the WFQ and WF^2Q+ lines. The weighted

performance can meet the WFQ and WF^2Q+ in simulation 3. In simulation 2 which 30 weighted flows share a link, three graphs show that WDRR is inability to keep tracking to the nominal model. It tries hard to keep with the WFQ and WF^2Q+ models but never success. In simulation 2 of CDP/UDP model, the algorithm seems to corrupt in the weighting mechanism in the extreme overloading situation (11.2Mbps).

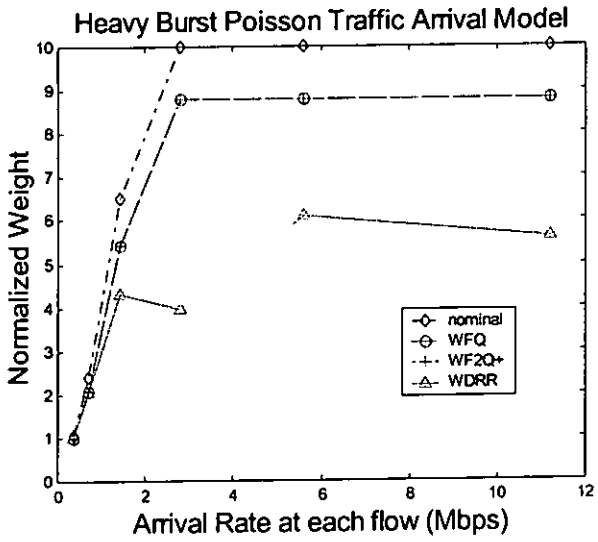

Figure 6-9 Normalized Weight of the Burst Poisson Model simulation 2

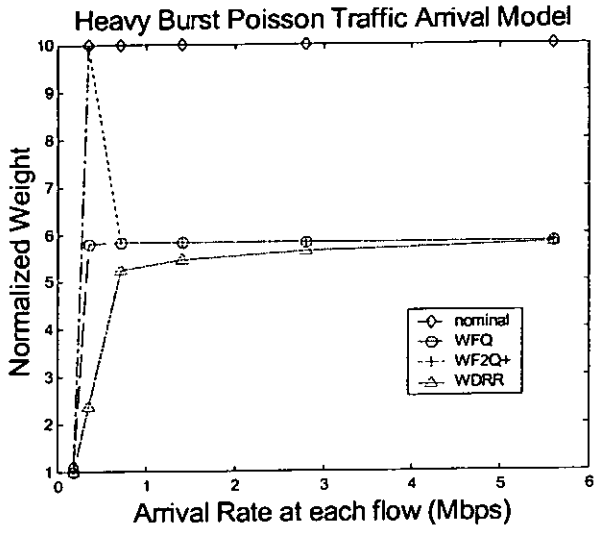

Figure 6-10 Normalized Weight of the Burst Poisson Model simulation 3

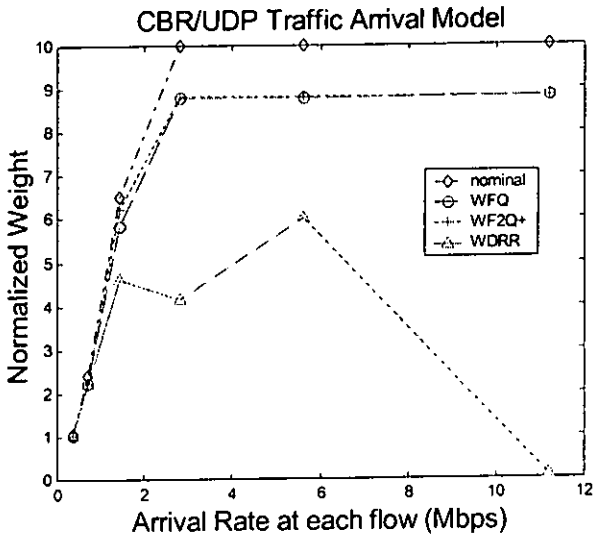

Figure 6-11 Normalized Weight of the IP Phone Model simulation 2

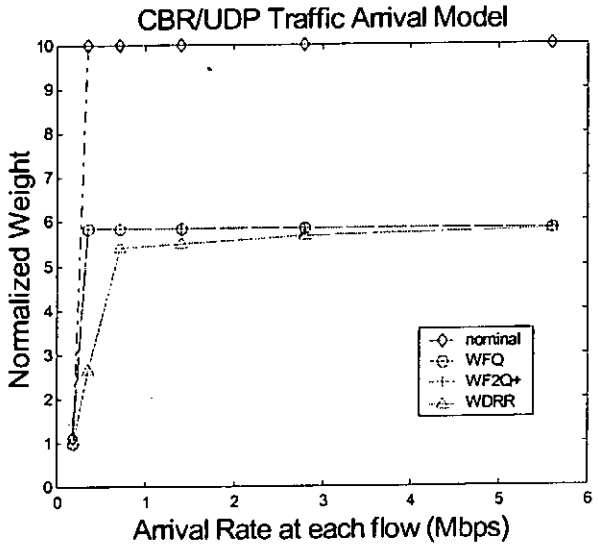

Figure 6-12 Normalized Weight of the IP Phone Model simulation 3

6.3 CONCLUSION

The proposed WDRR is a low complexity weighted queuing algorithm. It is an algorithm which modified from the DRR and provides the weighted sharing ability among different flows. The proposed algorithm also inherits the good properties of DRR. It uses a weighted quantum to control the weight between the flows and ensures the weighted fairness in the variable packet size environment. The weighted sharing ability of the algorithm is comparable to the WFQ and WF^2Q+. It is compared to be fair among the same weight flows in the constant bit rate TCP traffic model with the other low complexity queuing algorithms such as SFQ, RED and FRED. Because DRR is a subset of the WDRR, both algorithms share the same fairness among the same weighted flows. In the fairness simulations, WDRR inherits the unfavourable properties of the DRR. It is compared to be less fair in most of the same weighted fairness simulations. It is designed to be a low complexity approach to provide an alternative for weighted sharing algorithm which is highly demanded in the market.

CHAPTER 7 CONCLUSION AND FURTHER WORK

The investigation of nine existing queuing algorithms (VC, WFQ, WF^2Q+, FQ, SFQ, DRR, Traditional Tail Drop, FRED, RED) and two proposed algorithms (Weighted Deficit Probability Drop - WDPD and Weighted Deficit Round Robin - WDRR) are analytically studied in this thesis. The investigation started by introducing the work conserving and non-work conserving queuing disciplines. The evolution of the queuing strategies are introduced. The working conserving disciplines basically starts with the ideal fluid model – GPS, it is a perfect fair algorithm which can share the link between different flows. All flows can serve by the server simultaneously and the serving unit is infinitely dividable. It is perfect fair but it cannot be implemented in the real life. Firstly, it is impossible to serve several queues which go into the same output link simultaneously. Secondly, the smallest serving unit in network is a packet and it includes the payload and the header. It is impossible to process the packet bit by bit and it is impossible to have the infinitely dividable payload. Almost all successors in the queuing research of the working conserving discipline refer and modify the GPS ideal fluid system. The successors research focuses on implementing the GPS in the real life network, lowers the required computational complexity in approximating the fluid system and the fairness. From the academic research point of view, GPS is an ideal and the fairest algorithm. Some lately developed fairness indexes (such as WFI) rank the fairness by the difference between the delay of the packet in the testing algorithm and the GPS system. Fairness and computational complexity become a tradeoff. Many recent researches on queuing try to

lower the computational complexity and achieve a higher fairness performance at the same time, the huge research efforts alleviate the situation a bit but they cannot get breakthrough and gain extremely high fairness to complexity ratio. Actually fairness is quite ambiguous in the research arena. Many persons tried to explain the term "fairness" theoretically and few peoples tried to quantify the term "fairness" and develope on the fairness index. But up to now, there is no public acceptance and agreement on the term fairness. Different researches use their own fairness benchmark. Most of the fairness refer to the throughput fairness in the recent research. Actually, different network applications require different fairness on network parameters. In this research, we tried to generally describe the fairness term and investigate the fairness in different point of views. In all fairness simulations, the packet drop fairness, throughput fairness and delay fairness are compared. The traditional statistic tools and the Jain's Fairness Index which introduced in 1984 are used in fairness comparison in order to gain more public acceptance on the simulation results. Different input traffic loadings and different traffic arrival models are also used in order to gain the full picture on the "fairness". After all simulations, we discovered that there is no all around solution for the queuing algorithm. There are different pros and cons for the algorithms. Actually, the performances of the algorithms depend on the application and which network parameters that peoples are interested in. WFQ and WF^2Q+ are good at the throughput fairness. FRED and RED are good at the packet drop fairness and VC is good at the delay fairness.

Recent researches focused on the fairness among the same weighted flows and the computational complexity. But is it the interest of the commercial market? Are the users interested in the perfect fairness that between all the same paid users? There is lack of research on the weighted fairness and the weighted queuing algorithms. In our research, only WFQ and WF^2Q+ support weighted sharing among different flows. Two weighted

algorithms WDPD and WDRR are proposed in this thesis. The idea of the weighted fairness is described as the fairness between the high paid users and the low paid users. The amount of network resources needs to be allocated fairly and proportionate according to the preset weight. This also measures the ability of the queuing algorithm to assign resources according to the user paid model. The normalized value that described in chapter 4 is used for weighted fairness comparison purpose.

The proposed WDPD is a low complexity weighted queuing algorithm, which can provide an acceptable fairness on the same weighted flows. WFQ with complexity O(N), WF^2Q+ with complexity O(logN) and the WDPD with complexity O(1) where N is the maximum number of connections that sharing an output link. It out performs WFQ and WF^2Q+ on the computational complexity. It can work on the varying packet size environment. It can also ensure the fairness among different weighted flows.

Network power is used to measure the network performance for years. It was introduced by the renowned Internet researcher Leonard Kleinrock in 1981. In order to increase the network power, either increases the throughput or lowers the delay. Most of queuing researches focus on the worst case study. As the input traffic loading increases, the delay increases dramatically. In order to increase the network power, the delay is the key. To keep the delay small, the queue size must keep small in the overloading situation. This must be achieved by active dropping the packet. The proposed algorithm WDPD predicts the overloading flows and then drops the packets in the specific flow according to a pre-calculated probability. The network power simulation in chapter 5 shows that the WDPD getting the highest network power among the other 9 algorithms in all the loading levels and all the traffic arrival models. The network power of WDPD with traffic arrival factor 1

is at least ten times over the network power of WF^2Q+ and WFQ. The required queue size of WDPD is also greatly reduced.

In the fairness comparison test among the same weighted flows, WDPD with traffic arrival factor 0.5 is the best performer compared to the other traffic arrival factor. In the throughput fairness test, the WDPD with traffic arrival factor 0.5 is over the other 7 algorithms such as VC, FRED, RED etc but it is still less fair than WFQ and WF2Q+. In the delay fairness test, the WDPD with traffic arrival factor 0.5 is over other 7 algorithms such as WF^2Q+ and WFQ. In the packet drop fairness test, the WDPD 0.5 is just behind RED and FRED but it is still better than WF^2Q+ and WFQ.

In the fairness comparison test among different weighted flows, the weighted fairness of WDPD with traffic arrival factor 1.0 is the best algorithm which tracks closely with the nominal model. All other algorithms such as WF^2Q+ and WFQ show the inability to track with the model. Two weighted sharing patterns are used in the simulations in order to find out how they affect the fairness performance.

The other proposed weighted sharing algorithm is called Weighted Deficit Round Robin. Instead of push the computational power to the limit and gain a bit on fairness, a low computational weighted queuing algorithm is proposed. WDRR is same as WDPD with the computational complexity at O(1). It is modified from the Deficit Round Robin and provides the weighted capability among the flows that sharing a link. It uses different quantum sizes for different flows. The quantum size is positively proportional to the weight of the specific flow. The higher weighted flow can gain more credit and the packets in the flow get the priority to leave first.

In the fairness simulation among the same weighted flow, it is discovered that the throughput fairness, delay fairness and packet drop fairness of DRR are fairer than SFQ in

all the traffic arrival models and all the loading levels. DRR is comparable to RED and FRED in fairness. It is found to favourable in the TCP Constant Bit Rate Model. In this model, the DRR is even fairer than RED in throughput. It is also discovered that the standard deviations of the higher weighted flows in WDRR are always larger than the standard deviations of the lower weighted flows. That means the higher weighted flow is less fair than the lower weighted flows in WDRR.

In the weighted fairness among different weighted flows, WDRR is comparable to WFQ and WF^2Q+. Although all the algorithms (WFQ, WF^2Q+ and WDRR) cannot track the nominal model in the simulations of chapter 6, they still try their best to perform the bandwidth allocation according to the preset weight. The simulations also show that WDRR is quite sensitive and affected by different weighted patterns. In simulations 2 which 30 flows sharing the link, the weighted performance of the algorithm is quite poor but in simulation 3 which 60 flows sharing the link, the weighted performance is comparable to WFQ and WF^2Q+.

All the simulation in this research is based on the topology shown in figure 4-1 which is a single Internet node. By induction, the multi-node environment is the replication of many single-node segments if regardless of the routing and traffic pattern changing across the source-to-sink link. It is expected that the performance of the proposed WDPD and WDRR will be the same when applied to the multi-node simulation environment.

7.1 FURTHER IMPROVEMENT ON THE PROPOSED ALGORITHMS AND FURTHER RESEARCH DIRECTION

From the research, it can be noticed that the throughput fairness, delay fairness and packet drop fairness are affected by input traffic loading level, traffic arrival model and flow weighted pattern. Further research can be done on the correlation between these factors and

how they affect the algorithms. From the research, it is noticed that the fairness is generally lowered then the input traffic loading or the number of flows that sharing the link is raised.

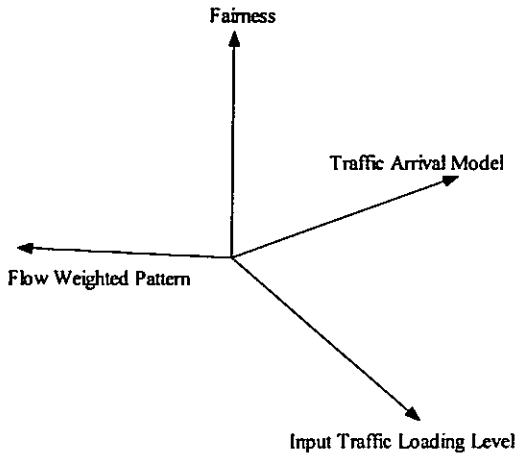

Figure 7-1 Four dimensions diagram on the factors which affect the fairness.

Figure 7-1 shows the three factors that affect the fairness of the queuing algorithm. Stochastic and chaotic process can be used to further analyze the simulation results.

On the proposed WDPD algorithm, further improvement can focus on the following points:

(i) Improve the fairness among the same weighted flows and the idea of the FRED can be added to approximate the real-time average size of the queue. Manipulation with the dropping probability of WDPD and FRED to form a new dropping probability. Enable the WDPD with real time capability to due with the fast traffic fluctuating situation.

(ii) The better algorithm can be designed to prevent the unnecessary drop of the packets, which leads to higher link utilization rate and prevent unnecessary packet retransmission in the TCP traffic model.

(iii)In the simulation in chapter five, it can be noticed that the traffic arrival factor 0.5 is good in fairness among same weighted flows and the traffic arrival factor 1 is good in the fairness among different weighted flows. Improvement can be focused on auto adjusting the traffic arrival factor due to the traffic loading and the input traffic arrival model. The traffic arrival factor can be positively proportional to the traffic loading level so it can provide a good fairness among the same weighted flows in the low loading condition and it can provide a good fairness among different weighted flows in the high loading conditions.

Further improvement on the WDRR algorithm can focus on:

(i) Further simulation on WDRR and find whether the weighted fairness can be improved in the heavily overloaded situation by reducing the queue size of the flows.

(ii) Add mechanism to approximate the overloading flows in real time and drop the packet accordingly. This can improve the consistence of the queuing performance across different traffic arrival models.

REFERENCES

[1] A. Demers, S. Keshav, S. Shenker, "Analysis and Simulation of a Fair Queuing Algorithm," SIGCOMM, vol. 19, No. 4, 1989, pp. 1-12.

[2] Bennett J.C.R., Hui Zhang, "WF^2Q: worst-case fair weighted fair queuing," INFOCOM '96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation., Proceedings IEEE , vol. 1 , 1996, pp. 120 –128.

[3] Bonald, T., May, M., Bolot, J.-C, "Analytic evaluation of RED performance,"INFOCOM 2000, Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE , vol. 3 , 2000, pp. 1415 –1424.

[4] C. Kolias and L. Kleinrock: The Power Function as a Performance and Comparison Measure for ATM Switches," in proceedings of Globecom '98, Sydney, Australia, Nov. 1998, pp. 381-386.

[5] C. R. Kalmanek, H. Kanakia and S. Keshav, "Rate controlled servers for very high speed networks," GLOBECOM '90 IEEE, vol. 1, 1990, pp. 12-20.

[6] C. R. Bennett and Hui Zhang, "Hierarchical Packet Fair Queuing Algorithms," IEEE/ACM Trans. Networking 5, 5 (Oct. 1997), pp. 675 – 689.

[7] D. Stiliadis and A. Varma, "Design and analysis of Frame-based Fair Queuing: A New Traffic Scheduling Algorithm for Packet-Switched Networks", in ACM SIGMETRICS May. 96.

[8] E. Gafni and D. Bertsekas, "Dynamic Control of Session Input Rates in Communication Networks," IEEE Transactions on Automatic Control, vol. 29, No. 10, 1984, pp. 1009-1016.

[9] E. Hahne, "Round Robin Scheduling for Fair Flow Control in Data Communication Networks," Report LIDS-TH-1631, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, Massachusetts, Dec, 1986.

[10] Gail, R. and L. Kleinrock, "An Invariant Property of Computer Network Power", Proceedings of the International Conference on Communications, Denver, Colorado, June 14-18, 1981, pp. 63.1.1-63.1.5, 1981.

[11] Golestani, S.J., "A self-clocked fair queuing scheme for broadband applications, " INFOCOM '94, Networking for Global Communications, 13th Proceedings IEEE vol. 2, 1994, pp. 636 – 646.

[12] Hui Zhang, "Service disciplines for guaranteed performance service in packet-switching networks," Proceedings of the IEEE, Vol. 83, No. 10, Oct. 1995, pp. 1374 –1396.

[13] Hui Zhang and Srinivasan Keshav, "Comparison of rate-based service disciplines," In Proceedings of ACM SIGCOMM Aug. 91, pp. 113 – 121.

[14] Ion Stonica, "CSFQ (Core-Stateless Fair Queuing) source code for NS," http://www.cs.cmu.edu/~istoica/csfq/step-by-step.html.

[15] Ion Stonica, "FRED (Fair Random Early Drop) source code for NS," http://www.cs.cmu.edu/~istoica/csfq/step-by-step.html.

[16] Ki-Ho Cho, Hyunsoo Yoon , "Design and analysis of a fair scheduling algorithm for QoS guarantees in high-speed packet-switched networks," Communications, 1998. ICC 98. Conference Record. 1998 IEEE International Conference, vol. 3 , 1998, pp. 1520 –1525.

[17] K. K. Ramakrishnan, D. M. Chiu and R. Jain, "Congestion Avoidance in Computer Networks with a Connectionless Network Layer," Part IV-A Selective Binary Feedback Scheme for General Topologies, DEC Technical Report TR-510, Digital Equipment Corporation, Nov 1987.

[18] Larry L. Peterson & Bruce S. Davie, "Computer Networks: A Systems approach," San Francisco, Calif.: Morgan Kaufmann Publishers, October 1999, pp 506-509.

[19] Lixia Zhang, "VirtualClock: a new traffic control algorithm for packet-switched networks," ACM Trans. Comput. Syst. 9, 2 May. 1991, pp. 101 – 124.

[20] Markus Wischy, "VirtualClock source code for NS," http://studweb.stud.uni-stuttgart.de/studweb/users/inf/inf13425/projects/virtualclock/index.html.

[21] Marc Greis, "IntServ source code for NS," http://www.teltec.dcu.ie/~murphys/ns-work/rsvp/index.html.

[22] McKenney, P.E., "Stochastic fairness queuing," INFOCOM '90, Ninth Annual Joint Conference of the IEEE Computer and Communication Societies. The Multiple Facets of Integration. Proceedings, IEEE, 1990, vol.2, pp. 733 –740.

[23] Nortel, "DiffServ source code for NS," http://www7.nortel.com:8080/CTL/.

[24] Paolo Losi, "WFQ source code for NS," http://www.pao.lombardiacom.it/wfq.html.

[25] Parekh A.K., Gallager R.G., "A generalized processor sharing approach to flow control in integrated services networks-the single node case," INFOCOM '92. Eleventh Annual Joint Conference of the IEEE Computer and Communications Societies, IEEE, vol.2, 1992, pp. 915 – 924.

[26] Parekh A.K., Gallager R.G., "A generalized processor sharing approach to flow control in integrated services networks-the multiple node case," INFOCOM '93. Proceedings.Twelfth Annual Joint Conference of the IEEE Computer and Communications Societies. Networking: Foundation for the Future, IEEE , 1993, vol.2, pp. 521 – 530.

[27] R. Cruz, "A calculus for network delay: I & II Network elements in isolation," IEEE Transactions on Information Theory, vol. 37, Jan. 1999, pp. 114 – 141.

[28] Rexford J.L., Greenberg A.G., Bonomi, F.G., "Hardware-efficient fair queuing architectures for high-speed networks," INFOCOM '96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation., Proceedings IEEE , vol. 2 , 1996, pp. 638 – 646.

[29] R. Jain, D. Chiu, and W. Hawe, "A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems," DEC Research Report TR-301, September 1984.

[30] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," IEEE/ACM Trans. Networking 1, 4 (Aug. 1993), pp. 397 – 413.

[31] Shahzad Ali, "WF^2Q+ source code for NS," http://www.cs.cmu.edu/~cheeko/wf2q+/.

[32] Shreedhar, M. and Varghese, G., "Efficient fair queuing using deficit round-robin Networking," IEEE/ACM Transactions, Vol. 4, No.3, June 1996, pp. 375 –385.

[33] S. J. Golestani, "A stop-and-go queuing framework for congestion management," Proceedings of the ACM symposium on Communications architectures & protocols, 1990, pp. 8 – 18.

[34] Suri, S.; Varghese, G.; Chandranmenon, G., "Leap forward virtual clock: a new fair queuing scheme with guaranteed delays and throughput fairness," INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies Proceedings IEEE, Vol. 2, 1997, pp. 557 – 565.

[35] Stephens D.C., Bennett J.C.R., Hui Zhang, "Implementing scheduling algorithms in high-speed networks," Selected Areas in Communications, IEEE Journal, vol. 17, Issue 6, June 1999, pp. 1145 –1158.

[36] Stiliadis, D. and Varma, A., "Latency-rate servers: a general model for analysis of traffic scheduling algorithms," IEEE/ACM Transactions on Networking, Volume: 6 5, Oct. 1998, pp. 611 –624.

[37] W. Feng, D. Kandlur, D. Saha, K. Shin, "Blue: A New Class of Active Queue Management Algorithms," April 1999. http://www.thefengs.com/wuchang/blue/CSE-TR-387-99.pdf.

[38] W. J. Kim and B. G. Lee, "FRED-fair random early detection algorithm for TCP over ATM networks," Electronics Letters Volume: 34 2, 22 Jan. 1998, pp. 152 –154.

APPENDIX A NETWORK SIMULATOR STUDIES

The Network Simulator 2 (NS2) version 2.1b6 is used to conduct the whole research. This network simulator was developed by University of California Berkeley and now developing by the VINT project. This software is renowned in the networking academic and industrial research. Over hundred of network research projects are performed using the NS2. The NS2 itself is written by TCL/TK and C++ language. It is highly customizable. The simulation and customization are performed by TCL scripts, no compilation is required to construct new simulation. TCL is an interpreted language and there is no performance degradation in using it, the actually computation and simulation are performed by the compiled C++ code.

A.1 NETWORK SIMULATORS CONSIDERED IN THE RESEARCH

Seven different network simulators are considered by using in this research, they are NETSIM by MIT, CPSIM by Boyan Tech Inc., INSANE by UCB, NEST by Columbia University, REAL by Cornell University, OPNET by MIL3 Inc and NS by University of California Berkeley. These simulators will be introduced one by one.

A.2 NETSIM

NETSIM is developed by MIT Advanced Network Architecture group. This is an event driven simulator for packet switched networks. The simulator framework provides the functions to schedule events and to communicate with user. It provides X window graphic

user interface. This package comprises an event manager, I/O routines, various structural tools such as queues and lists used to build components and a toolkit. The toolkit is a library of C functions to ease the manipulation of components. Components such as Ethernet link, point-to-pint link, switch, host, Purdue's implementation of TCP, data supplier and consumer from TCP, simple Poisson traffic source and packet sinks include in the package. The simulation engine is simple. It is a single process written in C to schedule events. Logical time is an unsigned 32-bit value, the simulator can runs for almost 12 hours of simulated time. National Institute of Standards and Technology (NIST) based on NETSIM and developed an ATM network simulator for studying and evaluating the performance of ATM network. In order to simulate the queuing algorithms in networks, new components in C needs to be written. This simulator is lack of support and huge effort required to develop the traffic generation model and queuing modules. It is rejected and not use for the research simulation. More information about the NETSIM can be found at ftp://allspice.lcs.mit.edu/pub/netsim/.

A.3 CPSIM

CPSIM is a parallel general-purpose simulation tool commercially available. It was created by Dr. Bojan Groselj and now the product can be found at Boyan Tech Inc. It uses the parallel discrete-event simulation (PDES) to suit the large discrete-event simulations purpose. It can be used for computer network simulations. The designer focuses on the performance and there is no graphical user interface. There is a strict separation between CPSIM kernel and CPSIM library. The simulation kernel provides synchronization, scheduling, deadlock prevention and message passing. This kernel can be used for parallel processing or uni-processor. The simulator is written in C and it is formed by communicating objects which are partitioned among processors. The underlying

programming model is an event message passing between objects. User can defines the granularity of the object in the simulator. The user needs to define the event data structure, the network object graph and writes code for invoking the events. This simulator is no longer on the web and it is not specially design for the network simulation, it is rejected to and not use in the research.

A.4 INSANE

INSANE (Internet Simulated ATM Networking Environment) is specially designed for network simulation at University of California Berkeley in 1996. The simulator is created by Bruce Mah and used to simulate a wide-area ATM backbone (similar to XUNET II). It is used to test various IP-over-ATM (Asynchronous Transfer Mode) algorithms with realistic traffic loads derived from empirical traffic measurements. The ATM stack

protocol of the simulator provides real-time guarantees ATM virtual circuits (VC) by using Rate Controlled Static Priority (RCSP) or First In First Out (FIFO) queuing. ATM signaling is performed using a protocol similar to the Real-Time Channel Administration Protocol (RCAP). It also supports Internet Protocol (IP), Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). It is an object-oriented, discrete-event simulator with library of TCL scripts. Simulation and customization are performed with TCL scripts. The simulator is not a multi-threading program. It is a single process so it cannot make the benefit from the multiprocessor CPU. The objects in the simulator are basically implemented in the fashion of finite state machine. Objects communicate by posting events to each other. The event scheduler delivers events to the relevant object according to chronological order and based on calendar queue. This simulator is specifically written for ATM simulation and lack of ready built queuing policy, it is

rejected to use in the queuing study research. The INSANE can be found at http://www.employees.org/~bmah/Software/Insane/.

A.5 NEST 2.5

NEST (Network Simulation Test-bed) was developed by department of Computer Science at Columbia University. It is targeted for simulating and prototyping distributed algorithm and systems. It is well suited to understand the behavior of routing protocol and routing loops for instance. It is well used by other studies, such as load-balancing system for microeconomic principles, ARPANET topology update problem and network architecture based on flooding protocol. The NEST 2.5 uses a client-server model which display clients are connected to the simulation server by a socket. The simulation server is responsible for execution of simulation and clients are independent programs used to create, configure a simulation model and control its execution. The communication between the server and client uses GUI that communicates with the simulation through the TCP/IP connection. It allows dynamically to create or modify the network configuration. The client-server model provides several advantages. It saves CPU resources by running the simulation on a dedicated super-computer. The GUI control interface sits on the client side. The NEST is implemented as a C library of functions linked with user's code. The simulation topology is created by a set of graphical tools with links and nodes stored internally as a table. New node function and communication link behaviors created by user that link with the network model.

The simulation runs within a single Unix process and this architecture involves least amount of context switching overhead when comparing with multi-tasking implementations. It facilitates a lightweight process mechanism which used to simulate complex distributed system. Each node in the system runs as a separated thread of control

with its own stack for local variable. It is a non-discrete-event based simulator which the simulation proceeds in a series of synchronization passes. Round-robin scheduling are used and each simulation node receives equal amount of running time. Because of the inefficient design of the simulator, it is not appropriate for high-speed network link simulation in normal home used computer system. The server and client need to be built even for simple network simulation. It is rejected using in our research. Detail about this simulator can be found at ftp://ftp.cs.columbia.edu/nest.

A.6 REAL 5.0

REAL (Realistic And Large) is a network simulator written at Cornell University by S. Keshav in 1997. The author also published papers in reference [1, 5, 13]. It is based on a modified version of NEST 2.5. It is also written in C. A graphical user interface (GUI) which eases the creation of the simulation topology and written in Java by Hani T. Jamjoom. The NEST code becomes cleaner and faster, it is specially designed to study the dynamic behavior of flow and congestion control schemes in TCP/IP packet switched data network. It provides thirty modules written in C that emulate flow-control protocols such as TCP and five scheduling disciplines such as FIFO, Fair Queuing, DEC congestion avoidance and Hierarchical Round-Robin (HRR), however it inherits the slow performance of NEST and it is not used in our reach for high speed link and high demanding network simulation. Details of the simulator can be found at the following web site:

http://www.cs.cornell.edu/skeshav/real/overview.html.

A.7 OPNET

OPNET (Optimized Network Engineering Tools) is a commercial network simulation tool and has been developed by MIL3 Inc for 15 years. It is used by many universities in

network research. It includes OPNET Modeler, OPNET Planner, Model Library and Analysis tools. The simulator features with an event-driven scheduled simulation kernel which using hierarchical object based modeling. The OPNET Modeler intended for modeling, simulating and analyzing the performance of large communication networks, computer systems and applications. It is used to assess the feasibility of new design, optimize the developed communication systems and predict performance. The modeling methodology of OPNET is organized in a hierarchical structure. At the lowest level, the process models are structured as a finite state machine. State and transitions are specified graphically using state-transition diagrams whereas conditions that specify what happen within each state are programmed with Proto-C (a C-like language). The OPNET Planner is used to evaluate the performance of communication networks and distributed system. Models are built by using graphical interface. The user can choose pre-defined models (from the physical layer to the application layer) from the Model Library and sets attributes. The Model Library contains protocols and analysis environments, such as ATM, TCP, IP, Frame Relay, FDDI, Ethernet, link models such as point-to-point and bus, queuing disciplines such as First-in-First-Out, Last-in-First-Out, priority non-preemptive queuing, shortest first job, round-robin or preempt and resume. The analysis tool provides a graphical environment to view and manipulate data collected during simulation runs. Results can be analyzed for any network element.

Although OPNET is powerful and well developed, it is quite expensive and it does not allow users to develop new models. All new models must contact MIL3 for the modeling services. It is rejected and not used for our simulations. More information about OPNET can be found at http://www.mil3.com/home.html.

A.8 NETWORK SIMULATOR 2

NS2 (Network Simulator 2) is an object-oriented discrete-event simulator for networking research based on REAL simulator. The initial NS1 was developed by the Network Research Group at the Lawrence Berkeley National Laboratory (LBNL). The NS2 is now part of the VINT project. The VINT project is part of the University of Southern California and is funded by the Defense Advanced Research Projects Agency (DARPA) in collaboration with Xerox PARC and LBNL. The aim of the project is to build a network simulator that offer innovative methods and tools. The work focuses on scaling, correctness, performance of wide area networks and protocol interaction issues in integrated services Internet network at all levels, from routing to session protocol. The project is not to design a new network simulator but to unify the effort of all people working in the field of network simulation. Most of current network simulators focus only on single protocol and simulate protocol in isolation. They do not address interactions with the other components of the architecture. They also lack of comparability across simulations. Because the effort on unifying people in the network simulation field, NS2 is now a successful simulator with many readily build modules and some contribution modules by network researchers. The researchers and developers involved in the project have extensive experience with network simulation based on the experience with MIT's NETSIM, University of Maryland's MARS, UC Berkeley's REAL, University of Columbia's NEST and LBNL's NS. NS2 is extensively used by academic and industrial research community with simulations well-suited for packets switched network, queuing algorithms, transport protocol congestion control and multicast network. The VINT project is built on the NS1 and the NAM (Network Animator), an animation tool for viewing the simulations results and packet trace data. The NS2 offers a composable simulation network

in order to model the modularity of the Internet and to support component modules from many contributors, various abstraction techniques and tools provide the ability to vary the level of abstraction, an emulation interface to allow the actual nodes to interface with the simulator, extensible libraries of network topologies and traffic generators. It contains library for unicast, multicast, routing, bandwidth reservation, queuing algorithms, transportation and session layer protocols.

The NS is written in C++, TCL (Tool Command Language) and Tk. The packages provide compiled class, hierarchy of objects is written in C++ and an interpreted class hierarchy of objects is written in OTCL (MIT's object extension to TCL). The user creates new objects through the OTCL interpreter. New objects are closely mirrored by corresponding objects in the compiled C++ hierarchy. The TCL procedures are used to provide flexible and powerful control such as start and stop events, network failure, statistic gathering and network parameter configuration. The TCL interpreter has been extended with OTCL command to create network topology of links, nodes, traffic agents and sink. These architectures allow high simulating performance, easy topology creation and highly customized simulation situation. The other favourable feature of NS2 is the network emulation. It can introduce the simulator into a live network. There are two modes, in the first mode the live traffic can pass through the simulator and the endpoints cannot notice about it. This mode is suitable for implementing a software real-time packet scheduler and testing the scheduling algorithm in the real network. The second mode is traffic generation and statistics. The simulator can include traffic sources or sinks communicating with the real-world networking equipment.

The NS2 provides rich libraries and highly customized environment with source codes, it is totally free of charge and with satisfactory support by large groups of network simulation

communities. Researchers can discuss on the problems and find the best solution on the network simulation by using the well-developed mail archive distribution and search system which found in the NS website. Because of the hybrid-programming language architecture of the NS simulation core, the simulating performance is reasonable by running on a home PC machine. The VINT project is very successful by gathering the network research communities, different communities now get the same network performance benchmark on the same network simulation platform. Individual researchers contribute their self-developed modules for the NS2, when somebody developed an algorithm, others can test it and prove the performance on the simulator. The contributions also enrich the functional libraries and the NS2 launches new version every half year. Because of these advantages, the NS2 is our choice in the queuing simulation study. The detail about the NS can be found at http://www.isi.edu/nsnam/ns/.

Table A-0-1 summarized the major features of the seven network simulators that considered in this research. Network Simulator 2 is selected at last. It is a ready built simulator with good usage record by academic and industrial research. All queuing function can be modified in the C libraries and it is a highly customizable simulator with huge support in the network functional libraries. It is a simulator that modified and improved from REAL, NEST and NETSIM. Although it is free, it gets a huge supporting community, question and answer can be found in the NS newsletter forum. Problems can always solves by posting in the NS forum. Because the limited research period, research budget and computer resources, NS 2 is the best choice for our research purpose.

A.9 CONCLUSION AND SUMMARIZE ABOUT THE NETWORK SIMULATORS

Simulators	NETSIM	CPSIM	INSANE	NEST	REAL	OPNET	NS
Develop Organization	MIT Advanced Network Architecture Group	Boyan Tech Inc (Initiated by Dr. Bojan Groselj)	University of California Berkeley by Bruce Mah	Department of Computer Science in Columbia University	Cornell University by S. Keshav	MIL3 Inc.	Network Research Group at LBNL, VINT Project in University of Southern California, Developer of NETSIM
Simulator Architecture	Event driven simulator, run in a single process to schedule events	Parallel discrete-event simulation (PDES), general purpose, can uses for parallel processing or uni-processor.	Object-oriented, discrete event simulator, single process implemented by finite state machine	Use client-server model, server run as a single process, strong at routing protocol and routing loop simulation	A modified version of NEST 2.5, specially designed to study flow and congestion control, provide five scheduling disciplines in the library, faster performance	Well featured, developed for 15 years, hierarchical object based modeling with event-driven scheduled simulator	Object-oriented discrete –event simulator based on REAL simulator, fast, stable and many network modules already build in the C/C++ library
Written Language	C	C	C and TCL script	C	C, Java	Proto-C	C, C++, TCL, OTCL, Tk
Running Platform	Linux X windows	Unix based system	Unix based system	Unix based system	Unix based system	Windows NT/2000, Unix	Unix based System
Reason of Reject/ Acceptance	Lack of support and huge effort is needed to develop the traffic generation model and queuing modules.	The simulator is not specially designed for network simulation, no further support.	Specially designed for ATM backbone simulation and lack of ready built queuing policy.	Lack of support and huge effort for the queuing modules to be developed.	Slow performance and cannot use for high-speed link or high demanding network simulation.	Expensive and not allow to develop new models. Modeling service provides by the company.	Many ready built network modules, all source code provided, highly customizable, huge support community, rich scheduling/queuing functions, widely used in academic and industrial research, free of charge
Product status	Free	Commercial Product	Free	Free	Free	Commercial Product	Free

Table A-0-1 Summarization the features of the seven network simulators

APPENDIX B ARCHITECTURE AND QUEUING CAPABILITY OF NETWORK SIMULATOR 2

In order to develop packet queuing and scheduling modules for NS2, the object-oriented hierarch of the NS2 with the hybrid-programming language architecture will be introduced in this appendix chapter.

NS2 is an object oriented simulator written in C++ with OTCL interpreter as the front-end. The simulator supports a class hierarchy in C++ which is the compiled portion of the simulator core. This portion includes packet manipulation, packet scheduling, traffic generation and statistical logs generation modules. The similar class hierarchy is also within the OTCL interpreter which is the interpreted portion of the core. The two hierarchies are closely related and one-to-one mapping from an interpreted object to one compiled object. New simulator objects are created by using the interpreted portion (OTCL), the objects are instantiated within the interpreter and are closely mirrored by a corresponding object in the compiled hierarchy. The following figure describes the hybrid-programming language architecture of NS2. The simulation file is a TCL script file which contains the TCL simulation objects invoke the OTCL (an extend TCL library) interpreter and create nodes, traffic generators, sinks, links, topologies, packet scheduler. The simulation objects are interpreted and mapped to the corresponding C++ objects. The C++ objects execute the simulation sequence and actually simulate the network.

This architecture provides fast network simulation performance. The C++ is fast to run but slower to change and compile. Because of these properties, C++ is suitable for detailed protocol implementation. OTcl runs much slower but can be changed very quickly and interactively. It is ideal for configuration, setup and one-off purpose. It is used for changing simulation configuration in network topology, traffic generation, statistic procedure etc. NS uses TCLCL to provide glue to make objects and variables appear on both languages.

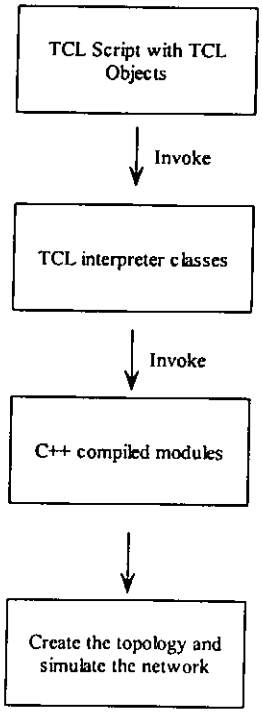

Figure B-0-1 Architecture of NS2 Simulation Core

The NS2 supports several queuing modules which includes First In First Out (drop-tail, FIFO), Fair Queuing (FQ), Stochastic Fair Queuing (SFQ), Deficit Round Robin (DRR), Random Early Drop (RED), Class-Based Queuing (CBQ), Weighted Round Robin (WRR), Priority Queuing, DiffServ [23] and Differential Service RED. Some contribution modules such as Worst-case Fair Weighted Fair Queuing Plus (WF^2Q+) [31], Weighted Fair Queuing (WFQ) [24], Core Stateless Fair Queuing (CSFQ) [14], Flow Random Early Drop

(FRED) [15], Virtual Clock (VC) [20] and IntServ [21] are available from the queuing researchers.

In order to develop the queuing modules, the class structure of the Queue Class must be known. Figure B-0-2 shows the object hierarchical structure of the Queue Class. The qlim_ is used to limit the queue size in the aggregating node. The unit of qlim_ is packets. The blocked_ is false by default. It becomes true when the node is unable to send a packet to its down-steam neighbor. The unblock_on_resume is set to true by default, it indicates that a queue should unlock itself when the last packet has been transmitted. The queue handler is used to manage the queue. When a queue receives a packet, the queue handler calls the queuing discipline-specific version of en-queue function. If the queue is not blocked, it sends a packet by calling a specific version of de-queue function. The specific de-queue function determines which packet to send, blocks the queue and sends the packet to the downstream neighbor of the queue. The handle function in the queue handler is used to pass information of the queue to the simulator scheduler. The handle function invokes the resume function, which sends the next-scheduled packet. The receive (recv) function is used to simulate the link and the processing delay. Figure B-0-2 only shows two queuing disciplines, DropTail and DRR. Actually there are over 7 disciplines in the NS2. There are some specific queuing discipline objects for each queuing discipline. In our example, there are four queuing discipline objects for the DRR discipline. They can be queuing parameters or queuing processing global variables. Buckets parameter is used to indicate the total number of buckets for hashing each of the flows. Blimit parameter is used to indicate the shared buffer size in bytes. Quantum is used to indicate how many packets each flow can send during each round. Mask is used for round-robin processing functions. By setting the queuing parameters, each queuing discipline can be finely tuned to suit certain traffic arrival model.

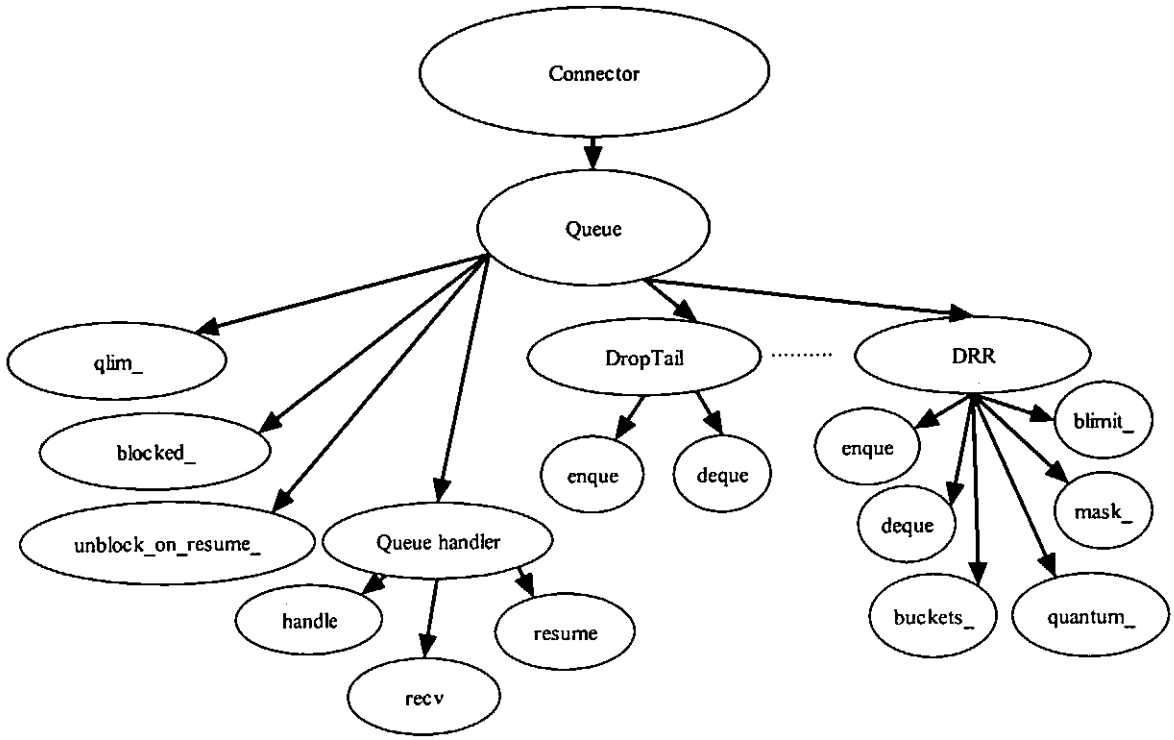

Figure B-0-2 Object Hierarchical Diagram of the Queue Class in NS2

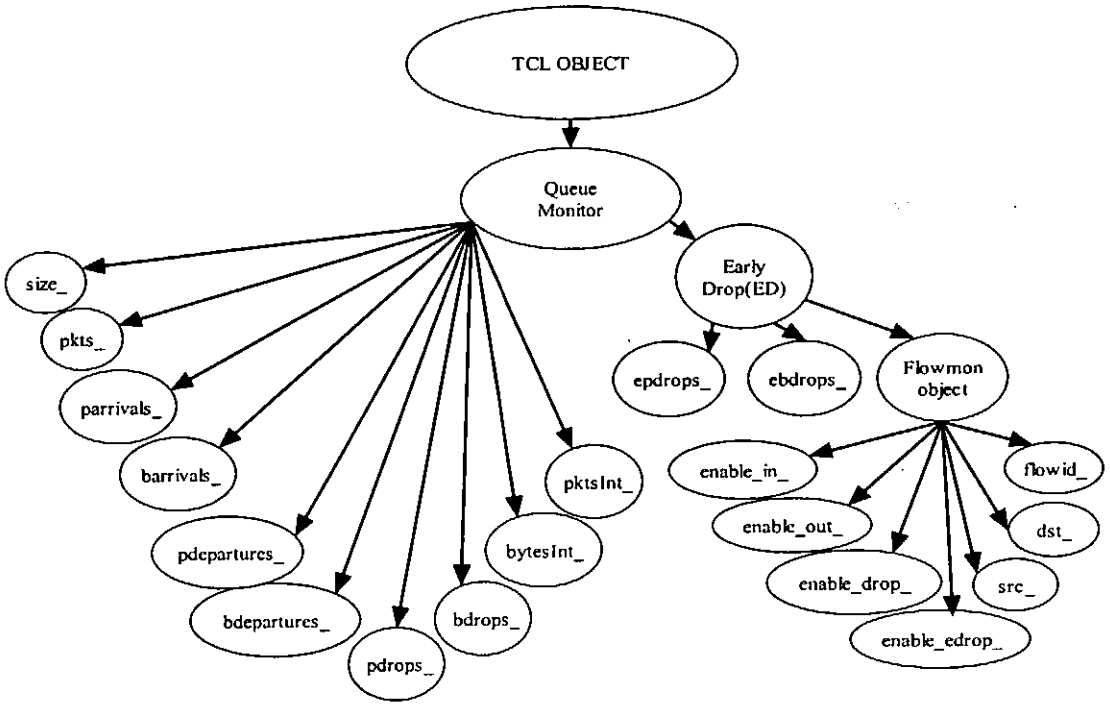

Figure B-0-3 Object Hierarchical Diagram of the Queue Monitor Class in NS2

Figure B-0-3 shows the network statistic parameters that supported by NS and use the Queue Monitor Class to develop the customized statistic parameters for queuing service

discipline study. The customized statistic parameters are used to form a queuing benchmark model, which compares the performance of different queuing disciplines. The derived Early Drop object is capable of differentiating regular packet drops (drops due to buffer exhaustion) from early drops (random drops in RED queues and FRED queues). Those drops need to put into statistic too. The Flow Monitor object is used in place of a conventional Queue Monitor object, collecting per-flow counts and statistics in addition to the aggregate counts and statistics provided by the basic Queue Monitor.

In order to develop our own Queue Monitor modules and to setup the comparison benchmark, the meaning of the parameters are shown in the table below.

Queue Monitor Objects	
Size_	Instantaneous queue size in bytes.
Pkts_	Instantaneous queue size in packets.
Parrivals_	Running total of packets that have arrived.
Barrivals_	Running total of bytes contained in packets that have arrived.
Pdepartures_	Running total of packets that have departed (not dropped).
Bdepartures_	Running total of bytes contained in packets that have departed.
Pdrops_	Total number of packets dropped.
Bdrops_	Total number of bytes dropped.
BytesInt_	Integrator object that computes the integral of the queue size in bytes. The sum_ variable of this object has the running sum (integral) of the queue size in bytes.
PktsInt_	Integrator object that computes the integral of the queue size in packets. The sum_ variable of this object has the running sum (integral) of the queue size in packets.

Queue Monitor / Early Drop Objects	
Epdrops_	The number of packets that have been dropped "early".
Ebdrops_	The number of bytes comprising packets that have been dropped "early".
Queue Monitor / Early Drop / Flow Monitor Objects	
Enable_in_	Set to true by default, indicates that per-flow arrival state should be kept by the flow monitor. If it set to false, only the aggregate arrival information is kept.
Enable_out_	Set to true by default, indicates that per-flow departure state should be kept by the flow monitor. If it set to false, only the aggregate departure information is kept.
Enable_drop_	Set to true by default, indicates that per-flow drop state for packet should be kept by the flow monitor. If it set to false only the aggregate drop information is kept.
Enable_edrop_	Set to true by default, indicates that per-flow drop state for bytes should be kept by the flow monitor. If it set to false only the aggregate drop information is kept.
Src_	The source address of packets belonging to this flow.
Dst_	The destination address of packet belonging to this flow
Flowid_	The flow id of packets belonging to this flow.

Table B-0-1 Meaning of the objects in the Queue Monitor Class

APPENDIX C SOURCE CODE FOR TRAFFIC GENERATION IN NS2 SIMULATIONS

C.1 TRAFFIC GENERATION FOR LIGHTLY AND HEAVY POISSON TRAFFIC

```
proc exptraffic { src dest sink size burst idle rate fid start_t stop_t} {
        set ns [Simulator instance]
        set udp [new Agent/UDP]
        $ns attach-agent $src $udp
        $udp set fid_ $fid
        set exp [new Application/Traffic/Exponential]
        $exp set packet_size_ $size
        $exp set burst_time_ $burst
        $exp set idle_time_ $idle
        $exp set rate_ $rate
        $exp attach-agent $udp
        $ns connect $udp $sink
        $ns at $start_t "$exp start"
        $ns at $stop_t "$exp stop"
        return $udp
}

# The code invokes the heavy Poisson traffic generation procedure
exptraffic $n(01) $n(101) $sink01 500 100ms 4ms 0.35M 12 0.5 8.0

# The code invokes the lightly Poisson traffic generation procedure
exptraffic $n(01) $n(101) $sink01 500 4ms 4ms 0.35M 30 0.5 8.0
```

C.2 TRAFFIC GENERATION FOR IP PHONE TRAFFIC

```
proc ipphonetraffic { src dest sink size burst idle rate fid start_t stop_t}
{
        set ns [Simulator instance]
        set udp [new Agent/UDP]
        $ns attach-agent $src $udp
        set cbr [new Application/Traffic/CBR]
        $udp set fid_ $fid
        $cbr set rate_ $rate
        $cbr set packetSize_ $size
        $cbr set random_ 1
        $cbr attach-agent $udp
        $ns connect $udp $sink
        $ns at $start_t "$cbr start"
        $ns at $stop_t "$cbr stop"
        return $udp
}

# The code invokes the IP Phone traffic generation procedure
ipphonetraffic $n(01) $n(101) $sink01 500 4ms 4ms 0.35M 14 0.5 8.0
```

C.3 TRAFFIC GENERATION FOR TCP/FTP TRAFFIC

```
proc tcp-ftp-traffic { src dest sink size burst idle rate fid start_t stop_t}
{
        set ns [Simulator instance]
        set tcp [new Agent/TCP]
        $tcp set class_ 1
        $tcp set packetSize_ $size
        $tcp set window_ 20
        $tcp set fid_ $fid
        $ns attach-agent $src $tcp
        set cbr [new Application/Traffic/CBR]
        $cbr set rate_ $rate
        $cbr set packetSize_ $size
        $cbr set random_ 1
        $cbr attach-agent $tcp
        $ns connect $tcp $sink
        $ns at $start_t "$cbr start"
        $ns at $stop_t "$cbr stop"
        return $tcp
}

# The code invokes the TCP/FTP traffic generation procedure
tcp-ftp-traffic $n(30) $n(101) $sink30 500 4ms 4ms 0.35M 30 0.5 8.0
```

APPENDIX D ALGORITHM OF QUEUING DISCIPLINES

D.1 RED

```
Initalization:
      avg=0
      count=-1
For each packet arrival
      Calculate the new average queue size avg
      If the queue is nonempty
            Avg=(1-w)*avg+w*q
Else
      m=f(time-q_time)
      avg=(1-w)*avg
If min_th<=avg<max_th
      Increment count
      Calculate probability p_a
      P_b=max_p*(avg-min_th)/(max_th-min_th)
      P_a=p_b/(1-count*p_b)
      With probability p_a drop the arrive packet
      Count=0
      Else if max_th<avg
            Drop the arrive packet
            Count=0
      Else count=01
When queue become empty
Q_time=time

Avg:        average queue size
Q_time:     start of the queue idle time
Count:            packets since last drop packet
W_q:        queue weight
Min_th:     minimum threshold for queue
Max_th:     maximum threshold for queue
Max_p:            maximum value for p_b
P_a:        current packet dropping probability
Time:             current time
F(t):             linear function of time t
```

D.2 RIO

```
For each packet arrive
      If it is an IN packet
            caculate the average IN queue size avg in;
      calulate the average queue size avg_total;
      If it is an IN packet
            if min_in<avg_in<max_in
                  calculate probaility P(in);
                  with probability P(in), drop this packet;
            else if max_in<avg_in
                  drop this packet
      If it is an OUT packet
            if min_out<avg_total<max_out
                  calculate probabilty P(OUT);
                  with probability P(OUT), drop this packet
            else if max_out<avg_total
                  drop this packet

avg_in:     the average queue for the In packets
avg_total:  the average total queue size for both In and Out packet
```

APPENDIX E PUBLICATION LIST

1. K.T. Yam, C.K. Li T.P.J. To and Matthew Y.L. Wong, "Real-time Network Control of Internet Telephony with Specified QoS Using RTP/RTCP", Proceedings of the 4th IASTED International Conference in Internet and Multimedia Systems and Applications (IMSA'2000), Las Vegas, USA, November 19-23, 2000, pp.145-150.
2. Matthew Wong and C K Li, "A Low Complexity Weighted Deficit Round Robin Queuing Algorithm", Proceedings of the 20th IASTED International Multi-conference in Applied Informatics 2002, Innsbruck, Austria, Feb 18-21, 2002, pp. 285-288.
3. M.L. Yan, C.K. Li, M. Y. L. Wong and C. K. Leung, " Improved Deficit Round Robin in High Speed QoS Network", Twenty-First IASTED International Multi-Conference, APPLIED INFORMATICS - AI 2003, February 10-13, 2003, Innsbruck, Austria. (submitted)
4. M.Y. L. Wong, C.K. Li, M. L. Yan and C. K. Leung, "Analysis of an Efficient Weighted Queuing using Weighted deficit probability drop", Twenty-First IASTED International Multi-Conference, APPLIED INFORMATICS - AI 2003, February 10-13, 2003, Innsbruck, Austria. (submitted)