# DELAY, FAIRNESS AND COMPLEXITY OF SELECTED SCHEDULING DISCIPLINES IN BROADBAND PACKET-SWITCHED NETWORKS



PADMINI VELLORE

# Delay, Fairness and Complexity of Selected Scheduling Disciplines in Broadband Packet-Switched Networks

by

© Padmini Vellore

A thesis submitted to the
School of Graduate Studies
in partial fulfillment of the
requirements for the degree of
Master of Engineering

Faculty of Engineering and Applied Science
Memorial University
May 2003

St. Johns Newfoundland and Labrador Canada

# Abstract

The explosion in the size of communication networks as well as the need for integration of voice, video and data have pioneered a need for fast packet switching. The economical implementation of fast packet switching has become a reality with the recent advances in VLSI technology. This has introduced opportunities for new applications like video conferencing, that demand severe performance requirements in terms of allocated bandwidth, delay, delay jitter, and loss rate. Packet scheduling is an effort to reduce delay, delay-jitter and losses thereby providing Quality of Service to such delay- and loss-sensitive applications.

The upshot of this research will influence the resolution of a most appropriate method of providing guaranteed but paid service to users of real-time applications like video conferencing, unlike the Internet, which is designed to provide best-effort service. Various packet-scheduling algorithms have been studied. Among the several existing packet-scheduling algorithms, Weighted Fair Queueing (WFQ) and Worst-case Fair Weighted Fair Queueing Plus ($WF^2Q+$) are selected in this thesis for comparison based on their delay properties. The performance is investigated for both fixed-sized packets and variable-sized packets. It has been found that $WF^2Q+$ is fair and introduces lower delay than WFQ. Due to the lower hardware as well as time complexity, $WF^2Q+$ can be considered as a prospective algorithm for use in high-speed packet-switched networks. The performance is studied on a network with and without the presence of cross-traffic for various traffic loads. Simulation results and hardware realization are discussed. A

brief canvassing on where these results lead us follows the conclusion. A proposal for future improvements is also presented.

# Acknowledgements

I express my sincere thanks to my supervisor Dr. R. Venkatesan for his continuous encouragement, full-fledged guidance, intellectual assistance, useful suggestions and financial support during the course of my research. I thank Dr. R. Venkatesan, Dr. T. Norvell, Dr. P. Gillard and Dr. H. Heys for the wonderful teaching offered by them when I was doing my course work, which in turn significantly helped in my research.

I thank Dr. C. Jablonski, Dean of Graduate Studies, for the financial support endowed on me during my stay in Memorial. I also thank the Associate Dean of Graduate Studies, and I would like to acknowledge Ms. M. Crocker, Office of the Associate Dean, for doing a yeoman's service in guiding all my concerns from commencement to completion of my work at Memorial University.

I thank the members of CCAE for the computing services provided by them during my study. I also thank Mr. N. White, System Administrator, Department of Computer Science, for his help with gaining access to Synopsys tools provided by CMC.

I thank my friends at CERL for their encouragement and support throughout my programme. Especially, I thank Prof. C. Li for spending his suggestions and criticisms and also for the invaluable discussions. I also thank R. Shahidi, the Lab Manager of CERL, for his uninterrupted help in making a comfortable environment to work and also solving frequent computing concerns.

# Table of Contents

# List of Figures

# Chapter 1

# 1. Introduction

## 1.1 Historical Background

In the late nineteenth century, telegraphy was the only means of long distance communication. The next progress started with introduction of the telephone network to carry voice. In 1890s, switches were introduced and the telephone service spread to a large extent. Initially, the telephones presented point-to-point communication, but with the introduction of digital telephony, other useful services such as teleconferencing were introduced. The idea of Internet (also called, Internet Protocol or IP) was introduced around 1960s for sharing the computing resources of researchers and for U.S military to have a robust communication against nuclear attacks [1]. Eventually, this emerged to be cheaper than the telephone network. Soon, a need to carry not only voice and data traffic, but also bursty video traffic arose. The telephone companies built an integrated voice/data network and called it Integrated Services Digital Network (ISDN). ISDN has a bandwidth of 128 Kbps which was not sufficient to carry video to customers in the United States and so, the concept of Broadband-ISDN or B-ISDN was introduced in mid-1980s which featured higher bandwidth. During the late 1980s the Asynchronous Transfer Mode (ATM) network was started and voice data could be carried along with other data in the network. Communicating through such integrated networks (ISDN or ATM) has captured the attention of several million people worldwide.

## 1.2 Integrated Services Networks

Integrated services network is the integration of video applications like video conferencing and online movies, voice applications like phone conversation and voice chat over the Internet, as well as, data applications like e-mail, fax and high-speed data transfer. Since it supports several applications through the same single network, the size of such an integrated network has grown at a rapid rate recently. This has created a need for fast switching through the nodes of the network and an assurance in bandwidth allocated to the users of the network. In general, the network consists of several nodes that switch packets from incoming links to one or more outgoing links. In general, each node in a network is a switch, which routes packets from an incoming link to one of several outgoing links. Each switch consists of an input port controller, the switch fabric and the output port controller (Figure. 1.1). Each switch also routes packets of different traffic classes according to the service requested by the user. This switching of traffic from various classes has to be done diligently and therefore various switching techniques have emerged. Due to the recent advances in Very Large Scale Integration (VLSI) technology, fast packet switching could be implemented economically.

Asynchronous Transfer Mode (ATM) switch is a good candidate for VLSI implementation due to its high bandwidth requirement [2]. ATM is based on small fixed-sized packet (cell) switching, and hence fast, while Internet Protocol (IP) is based on variable-sized packet switching. Another important difference between ATM and IP is that, ATM switching guarantees service while IP switching makes best efforts to enable packets reach their intended destination; hence the simplicity of the IP switching network.

Fast packet switching used in high-speed integrated services networks combines the above two switching technologies (ATM and IP), by providing guaranteed service to various traffic sources and sending packets of variable sizes, while also being simple and fast.

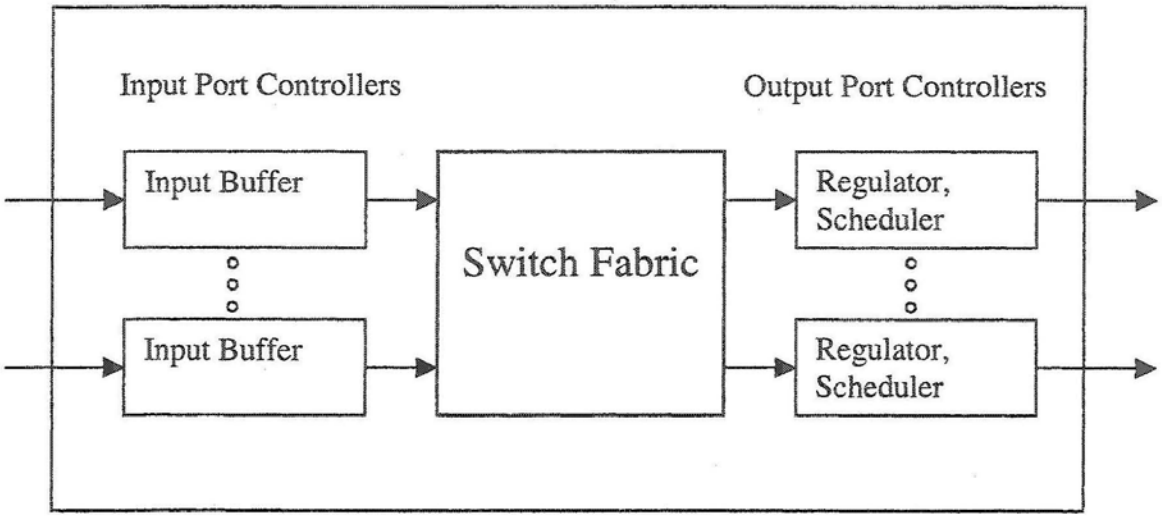



**Figure 1.1:** Block diagram of a switch

It is also necessary to make sure that the required throughput is achieved through the link. Thus the sources should be assured the desired quality of service they receive. Thus quality of service becomes an important issue.

## 1.3 Quality of Service

Due to the varied requirements of the users of video, voice and data, the integrated services network should be able to assure the service required by each class. That is, fast packet switching should not only be fast, but also provide quality of service (QoS) or guarantee the performance required by the various traffic classes. IP was initially a best-

effort service, that is, it simply routed the packets, but did not guarantee service to users requesting service guarantees. Recently, through the introduction of certain advanced protocols, the Internet has been used to provide QoS to users to a limited extent. Quality of service basically means, measuring certain characteristics (delay, delay jitter, packet loss, fairness, complexity, etc), improving them in order to meet the guarantees committed in advance. Packet switches in high-speed networks service packets belonging to two main kinds of applications, namely, best-effort and guaranteed-service applications [3]. Therefore, the switches in the nodes of these networks should be capable of serving packets based on their QoS requirements.

Some of the common QoS requirements are bounded end-to-end delay, fairness, simplicity, scalability, low loss rate and delay jitter. These are discussed in Section 1.5 of this Chapter. An arbiter or scheduler is required at the output port of the switch to order the transmission of packets to the output link based on their QoS guarantees. Selecting a packet scheduling discipline which operates at the output port of the switch is one of the key design criteria for providing QoS.

## 1.4 Scheduling

Within each switch, in the output port, there are queues - one for each service class or one for each user that will hold the packets that need to be sent through each link. A scheduler is required to select a packet buffered in one of these queues to be served next. Traditionally, the scheduling discipline used by the scheduler provided fair resource allocation by resolving contention among the network users. This policy was useful for

best-effort service. The integrated-service packet-switched networks simultaneously support multiple types of services over a single physical infrastructure [4]. Therefore, the scheduling disciplines in these networks play a critical role in controlling the interactions between different users. Thus, the scheduling discipline is different for different technologies like ATM and Internet.

In ATM networks, since the packets are transmitted as fixed-sized cells, the scheduling algorithms are usually implemented in hardware. In variable-sized packet-switched networks, since the packets are of larger size, they may be implemented in hardware or software. Many papers in the literature, such as [3], [5], [6], [7], [9], [10], [14], [16], [19], [20], [21], [23], [24] and [26], proposing packet scheduling algorithms study the performance problems on queuing systems. But they cannot be applied to integrated services packet-switched networks because of the bursty nature of the incoming traffic and also because the guarantee on performance bounds is on a per-connection basis. Recently, investigation of networks involving real-time bursty sources has resulted in scheduling disciplines that have the ability to provide bounds on end-to-end delay for a traffic source which is bursty and whose burstiness is constrained. Above all, the scheduling discipline should be simple enough to be implemented at high-speeds.

## 1.5 Properties of Scheduling Disciplines

The performance provided by a scheduling discipline is determined by the characteristics required of an application (either best-effort or guaranteed-service). The guaranteed-service applications require the server (scheduler or scheduling discipline) to

allocate a mean delay to each connection by choosing an appropriate service order. They also require the server to allocate different bandwidths to connections based on the share of the output link the connections require. Lastly, they require a guarantee on loss rate for each connection by limiting the number of packets entering the connections. Though neither the server nor the scheduling discipline needs to guarantee delay, loss or bandwidth to the best-effort applications, these attributes should be fair enough so that the best-effort connections receive some service. Thus the scheduling disciplines should satisfy the following properties as a minimum requirement.

- *Low end-to-end delay* – Real-time applications require low end-to-end delay. The scheduling discipline should be able to guarantee a lower bound on the end-to-end delay for certain applications possibly at the expense of increased delay to other non-real-time, best-effort applications.
- *Fairness* – The bandwidth available in the link should be shared among the applications in a fair manner. Primarily, the scheduling disciplines are simply a fair allocation of bandwidth among the users present.
- *Simplicity* – The scheduling discipline should be implemented as simple as possible, so that the time required to make a decision on the next packet to transmit is considerably low and as close as possible to the arrival time of packets. The scheduling discipline should also be implemented in hardware.
- *Scalability* – The scheduling algorithm should be able to support as many connections as possible. Typically, this number is in tens of thousands.

- *Delay jitter* – In simple terms, the delay jitter is the difference between the maximum and minimum delay a connection experiences. This difference should not be too high for feedback applications and applications carrying video.

## 1.6 Motivation for this Research

The integrated services network not only includes sources which have bursty traffic, but also constant rate sources, Poisson sources and sources which produce heavy traffic. Therefore, it is required to study various kinds of traffic the network encompasses. The scheduling disciplines we have considered in this thesis have not been tested exhaustively for various kinds of traffic, various traffic loads, or various packet-sizes. By and large, most studies assume a single server or a simple network with known traffic and fixed-sized cells that are easier to implement. In a real network, this is not the case. Thus a need to verify the behaviour of the scheduling disciplines by exposing them to various cases of traffic patterns and packet-sizes arose. In this work, the performance of some chosen scheduling disciplines is investigated under Internet traffic with variable-sized packets.

## 1.7 Organization of the Thesis

The rest of the thesis is divided into six chapters. Chapter 2 discusses the classification of scheduling disciplines and the properties required of a scheduling discipline. Chapter 3 introduces two chosen scheduling disciplines, discusses their

properties and the performance they guarantee. Chapter 4 details the traffic model used, the packet length distribution obtained, the software implementation and the delay results of the above two scheduling disciplines. Chapter 5 details the individual blocks involved in the hardware implementation and Chapter 6 identifies the contributions of the thesis and suggests areas of future work.

# Chapter 2

# 2. Scheduling Disciplines

## 2.1 Introduction

The output port of a switch consists of output buffers which contain packets that wait to be served on the output link. At each output port of a switch, a scheduler is present to manage these output buffers and arbitrate the access to the output line. The scheduler decides the order in which these requests are serviced onto the output link. The scheduler consists of a scheduling discipline which allocates different service qualities to users of various service requirements. The scheduling discipline does so by choosing a particular service order and also by deciding which packet to drop when there is excess traffic. Since scheduling is done at the output port of a switch, the scheduler is placed in the network layer. There are two main application types which the scheduling discipline has to consider while deciding the order in which to serve packets - guaranteed applications, which require a bound on the performance and so require resources to be reserved, and best-effort applications, which have elastic performance requirements and so do not need any reserved resources [1]. In order to support guaranteed applications, the scheduling discipline should be able to provide a bound on the per-connection delay, guaranteed bandwidth and a specified loss rate. In order to support best-effort applications, the scheduling discipline should be able to provide a fair allocation of resources to all the best-effort connections.

## 2.2 Work-conserving and Non-work-conserving Scheduling Disciplines

Apart from the classification of the applications requiring service, the service disciplines themselves are classified either as work-conserving or non-work-conserving disciplines. In a work-conserving scheduler, when a packet arrives, if the server is idle, the packet is served. Some of the work-conserving disciplines studied in the past include delay earliest-due-date [5], virtual clock [6], fair queueing [7] and its weighted version [8], self-clocked fair queueing [9] and worst-case fair weighted fair queueing [10]. In the non-work-conserving scheduler, the packet is held in the queue until it is eligible for service. The server may remain idle if the packet is not eligible, that is, if the packet does not conform to its agreed traffic profile [11]. In the non-work-conserving scheduling discipline, each packet is assigned an eligibility time and queued in the buffers. At any time, when the server is idle, the packet with the least finish time among the eligible packets is serviced. If none of the packets in the queue is eligible, none will be served.

To attain a bound on the end-to-end delay and also to determine the buffer space required, the traffic should be characterized inside the network. For a work-conserving discipline, the traffic is distorted inside the network due to fluctuations in the load inside the network, as depicted in Figure 2.1. Here, four packets are assumed to travel across a network with some inter-packet gap between them. In the Figure 2.1, each packet is represented by a vertical arrow. At the end of the first server, the first packet is delayed slightly longer than the second packet due to instantaneous cross-traffic. Thus the spacing

between the first two packets is small. At the end of the second server, the first two packets are further delayed and at the end of the third server, the first three packets are delayed, while the fourth packet passes without any delay. Thus the traffic pattern is distorted due to network load fluctuations and this makes the traffic burstier.

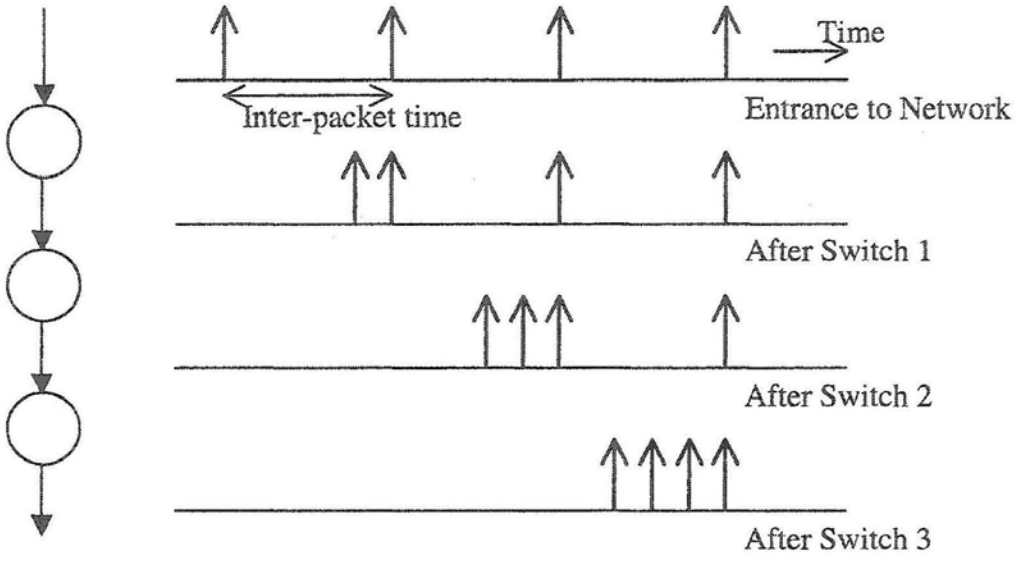



**Figure 2.1:** Traffic pattern distortions due to load fluctuations [12]

Thus it is hard to characterize the traffic pattern inside such a network. Moreover, users could misbehave by sending at a rate higher than the bandwidth allocated to them. This causes a higher instantaneous arrival rate at any switch. To avoid the distortion in the traffic, non-work-conserving scheduling disciplines are used. They reduce the traffic distortion at each switching node by fully or partially reconstructing the traffic. This increases the average delay, but the end-to-end delay is bounded. For guaranteed service, the bound on end-to-end delay is more important than the average delay. Though in non-work-conserving disciplines, the server remains idle sometimes, it assists in making the traffic more predictable in the nodes that follow. Thus the buffer space required in the

adjacent node can also be predicted. In this thesis, the buffer space available in each queue is assumed to be infinite in order to study the end-to-end delay bound on the traffic sources. One other disadvantage of non-work-conserving schedulers is that they waste bandwidth. But this is compensated by efficiently sending best-effort traffic whenever the server is idle. Some of the non-work-conserving scheduling disciplines include jitter earliest-due-date (jitter-EDD) [13], stop-and-go [14], hierarchical round robin (HRR) [15] and rate-controlled static priority (RCSP) [16]. Though current day switching uses only work-conserving scheduling disciplines, there is a good scope for non-work-conserving disciplines when more users join the network, or, when the link becomes heavily loaded.

## 2.3 Rate Controlled Service Disciplines

The scheduler, on its own, is capable of providing service guarantees on a per-connection basis only if the traffic entering that particular node satisfies certain traffic specifications. The traffic entering the network may conform to the constraints of the source, but the network load oscillates thereby distorting the traffic at a node. Thus the traffic entering a node may experience instantaneous burstiness. A class of non-work-conserving service disciplines are the rate controlled service disciplines [16]. The rate controlled servers tackle the problem of providing end-to-end delay bounds and managing traffic distortions by encompassing a separate rate-controller and a scheduler. A rate controller consists of a set of regulators, as shown in Figure 2.2, corresponding to each of the connections propagating through the switch; each regulator takes care of shaping the traffic of the corresponding connection into the desired traffic pattern. Some examples of

scheduling disciplines that can be used in the rate-controlled service disciplines are Stop-and-Go server, Jitter-Earliest Due Date, Hierarchical Round Robin, or even the simplest static priority queueing schedulers [3].

The rate-controller observes the traffic arrival rate for each connection, compares it to the expected arrival rate, and forces the connection to obey the required traffic

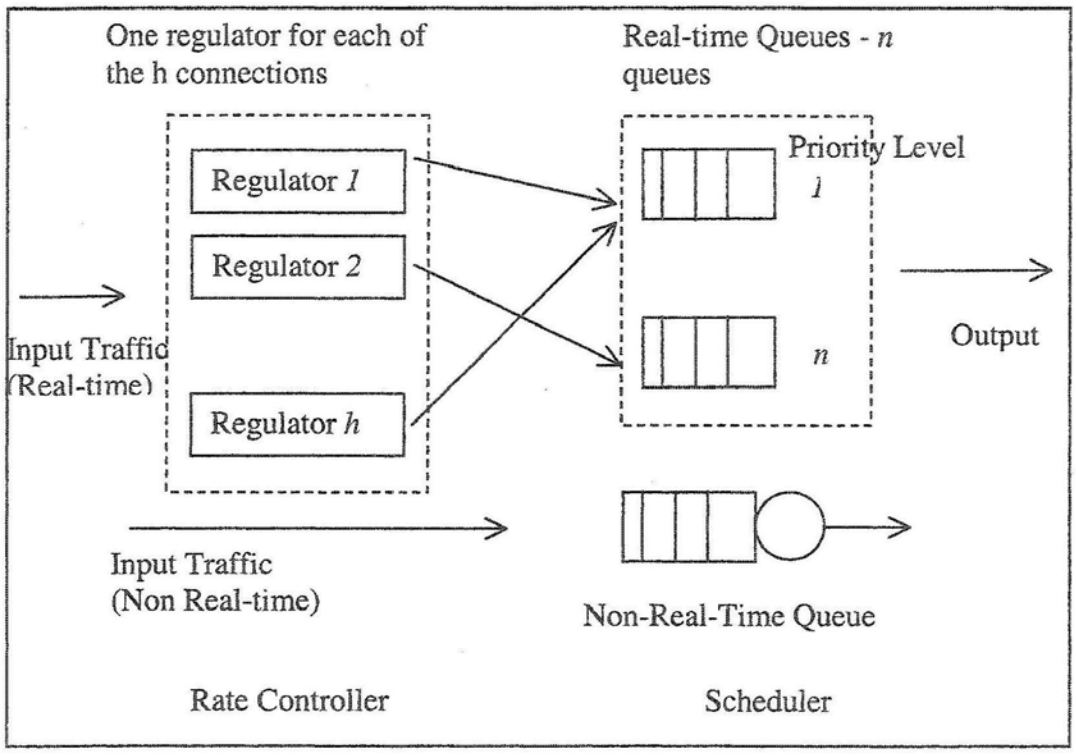



**Figure 2.2:** Rate-Controlled Service Discipline [16]

pattern by delaying packets from that connection if it sends packets at a rate higher than the tolerable arrival rate. Thus the traffic is reconstructed at each node, so that it is predictable at the node that follows. The scheduler multiplexes the packets based on their service priorities and also provides a bound on the end-to-end delay.

### 2.3.1 Regulators

A rate controller or regulator is a mechanism which enforces that traffic from a flow which is forwarded to a scheduler conforms to its original profile [17]. As mentioned earlier, the rate controller consists of a set of regulators, one for each connection, that shape the traffic entering the node. Several models are proposed in the literature for modeling the traffic arrival. According to Hui Zhang [11], there are three proposed models for traffic characterization:

- $(\sigma,\rho)$ model: A traffic stream satisfies this model if during any interval of length $u$, the number of bits in that interval is less than $\sigma + \rho u$. In the $(\sigma,\rho)$ model, $\sigma$ and $\rho$ are the maximum burst size and the long term bounding rate of the source, respectively [18].
- $(X_{min}, X_{ave}, I, S_{max})$ model: A traffic stream satisfies this model if the interarrival time between any two packets in the stream is more than $X_{min}$, the average packet inter-arrival time during any interval of length $I$ is more than $X_{ave}$, and the maximum packet size is less than $S_{max}$ [5].
- $(r, T)$ model: A traffic stream is said to satisfy this model if no more than $r \cdot T$ bits are transmitted on any interval of length $T$ [15].

The above characterizations are used to bind the traffic volume by placing a limit on the number of packets that can be received during an interval of time. Therefore, it is not possible to deduce the exact traffic pattern with these models. In this thesis, the regulators follow $(X_{min}, X_{ave}, I, S_{max})$ model.

Instead of having one regulator for each connection, this thesis models the rate controller with one regulator consisting of queues on a per-connection basis. Each connection has a predefined traffic model. At the arrival in the regulator, each packet, based on its corresponding connection's constraint, is delayed on the regulator queue until it is eligible and then sent to the scheduler. The regulator queue is modeled as a set of queues called, calendar queues. The calendar queue implementation, discussed in Section 5.2.7, reduces the complexity involved in the regulator queue maintenance to less than the number of connections in the network.

The key advantage in having a separate regulator and scheduler is that it allows arbitrary combinations of rate-control policies and packet scheduling schemes. Also, the regulator distributes the buffer space inside the network uniformly. Regulators control the interactions between switches and eliminate jitter. There are two kinds of jitter, namely, delay-jitter and rate-jitter. Delay-jitter is the maximum difference between the delays experienced by any two packets on the same connection [16]. Rate-jitter is defined as the maximum number of packets in the jitter averaging interval [16]. There are two classes of regulators, rate-jitter (RJ) controlling regulators and delay-jitter (DJ) controlling regulators.

#### 2.3.1.1 Delay-jitter Controlled Regulators

These regulators control the delay-jitter by fully reconstructing the traffic pattern. In such regulators, the eligibility time of a packet is defined with respect to the eligibility time of the same packet in the previous switch. To find out the amount of time the packet

was ahead of schedule in the previous switch, each packet has to have this value stamped in its header. This results in larger header size when a delay-jitter controller regulator is used than when a rate-jitter controlled regulator is used, and so, it is too expensive. For the delay-jitter controlling regulator [16]:

$$ET_0^k = AT_0^k;$$

$$ET_j^k = ET_{j-1}^k + d_{j-1} + \pi_{j-1,j},\ j > 0,$$

where, switch 0 is the source of the connection, $d_{j-1}$ is the delay bound, or the maximum waiting time of packets on the same connection at the scheduler of switch *j-1*, $\pi_{j-1,j}$ is the propagation delay between switch *j-1* and switch *j*, $ET_j^k$ is the eligibility time of the $j^{th}$ packet in the $k^{th}$ switch and $AT_j^k$ is the arrival time of the $j^{th}$ packet in the $k^{th}$ switch.

### 2.3.1.2 Rate-jitter Controlled Regulators

These regulators control the delay by partially reconstructing the traffic pattern. The eligibility time of a packet at a switch is defined with respect to packets arriving earlier at the same switch. Eligibility time for the $k^{th}$ packet on a connection at a switch $ET^k$ is defined with reference to the eligibility times of packets arriving earlier at the switch on the same connection [16]:

$$ET^1 = AT^1;$$

$$ET^k = \max(ET^{k-1} + X_{\min}, ET^{k-\lfloor I/X_{ave} \rfloor + 1}, AT^k),\ k > 1,$$

where, $AT^k$ is the time the $k^{th}$ packet on the connection arrived at the switch, $X_{min}$ is the minimum packet inter-arrival time, $X_{ave}$ is the average packet inter-arrival time over an interval of time $I$.

Since controlling delay-jitter completely reconstructs the traffic pattern at each switch along the path, if the traffic arriving into the network obeys the specifications, it will obey the specifications throughout the network. But the complexity of implementing delay-jitter controlling regulators is higher because they need to know information about the previous switch. Therefore, there is a trade-off between choosing delay-jitter & rate-jitter controlling regulators. In this thesis, the regulator is a rate-jitter controlling regulator and the traffic model used to characterize the arrival of packets is the ($X_{min}$, $X_{ave}$, $I$, $S_{max}$) traffic model.

#### 2.3.1.3 Trade-offs

The following are the trade-offs in implementing the regulator.

- *Implementation complexity:* In both the rate-jitter controlled and delay-jitter controlled regulators, the eligibility time is calculated on a per-packet basis. Thus the complexity is high. Moreover, for delay-jitter controlled regulators there is a need to synchronize either at the link level or at the switch level. After synchronization, the amount of time the packet was ahead of schedule is stamped in the packet's header.
- *Services provided:*
    - Though for a rate-jitter controlled regulator, the average delay is low, the delay-jitter is nearly three times higher than that of a delay-jitter controlled

regulator. As the number of nodes through which the connection traverses increases, the delay-jitter becomes higher and so, it can be used in applications where low average delay and bounded delay are needed.

- For clients with playback applications, the delay-jitter controlled regulators are better suited because they provide a bound on the delay-jitter and delay and not the average delay.

## 2.4 Discussion of Scheduling Disciplines

This section discusses some of the scheduling disciplines proposed in the literature, their properties, advantages and disadvantages. Generalized Processor Sharing (GPS) is an ideal scheduling discipline that provides a max-min fair allocation [7]. It was introduced as a scheduling discipline for the best-effort connections with the property of providing fair allocation of service to all the connections. But it cannot be implemented in practice, because it assumes to serve from each connection an infinitesimally small amount of data. Numerous scheduling disciplines have been proposed to emulate GPS as closely as possible.

The simplest emulation of the GPS is the round-robin (RR) scheduling discipline [1] which serves one packet from each of the non-empty connection queue in a round robin fashion. The weighted version of the round-robin, namely the Weighted Round Robin (WRR), serves packets from connections in proportion to their weights. However, it does not work if the source is unable to predict its mean packet size. In such a case, WRR cannot allocate bandwidth fairly. Moreover, it is fair only over a time scale longer

than a round time. A modified version of WRR, which is also easy to implement, is the Deficit Round Robin (DRR) [19]. The DRR can handle variable-sized packets even without knowing the value of the mean packet size. However, it is also unfair when the time scale is smaller than one round time. Smoothed Round Robin (SRR) [20] can emulate GPS well. When compared with RR schedulers, it reduces burstiness in the output, has better short-term fairness and also possesses good delay properties. At the same time, it also has an O(1) time complexity since it avoids time-stamp maintenance. It can be implemented in high-speed networks to provide QoS. However, it fails to provide strict local delay bound that is needed for guaranteed service applications. Therefore, it cannot be used in applications that require strict end-to-end delay bound.

Weighted Fair Queueing (WFQ) is an approximation of GPS. It is also called as the Packet-by-packet approximation of GPS (or, PGPS). Neither does WFQ require knowledge of the mean packet size nor does it consider the packets to be infinitesimally small data. The idea of WFQ is that it calculates the time (finish time) a packet would complete service in the corresponding GPS system and then serve packets in increasing order of these finish times. Since WFQ approximates GPS, it has the firewalling property of protecting the connections from each other. In other words, a heavy load on one of the connections will in no way affect the other connections. In addition, if any connection misbehaves, then it loses packets from its own buffers. It is possible for a connection to achieve end-to-end queueing delay independent of the number of nodes it is traversing through [1]. Thus WFQ provides real-time performance guarantees for guaranteed-service applications. Worst-case Fair Weighted Fair Queueing ($WF^2Q$) is almost identical to GPS differing by no more than one maximum size packet [10]. $WF^2Q$ disproves the previous

notion that WFQ is the closest approximation to GPS. $WF^2Q$ shares the bounded-delay and fairness properties of GPS. In this system, when the server has to make a decision on the next packet to transmit, it picks that packet which has the smallest finish time and which has already started service in the corresponding GPS system. $WF^2Q+$ [21] reduces the computational complexity of $WF^2Q$. More details about these scheduling disciplines appear in the subsequent chapters.

Another service discipline, namely, Self Clocked Fair Queueing (SCFQ) [9] speeds up the round number computation. In SCFQ, when a packet arrives at an empty queue, instead of using the round number to compute its finish number, it uses the finish number of the packet currently in service. Though the round number is easy to update, it is unfair for short time scales. Thus it has larger worst-case latencies than WFQ, and hence, greater unfairness in short time scales. Start-Time Fair Queueing (STFQ) [22] has the computational benefits of SCFQ, but differs from SCFQ in the sense that it services packets in increasing order of start numbers. Therefore, it does not have the large worst-case delay as SCFQ nor the short-term unfairness.

Virtual clock, proposed by Zhang, [6] is for scheduling guaranteed-service connections. It is similar to WFQ but emulates Time-Division Multiplexing (TDM). Each packet has a virtual transmission time. This is the time at which the packet would be transmitted if the server is implementing TDM. When it is used for best-effort connections, the relative fairness bound is infinity. That is, when there are two backlogged connections, one might receive infinitely more throughput than the other. In the classic Earliest Due Date (EDD) scheduling [5], each packet is assigned a deadline, and the scheduler serves packets in order of increasing deadlines. If the scheduler

commits more than its capability, then some packets miss their deadlines. Also, if packets are assigned deadlines closer to their arrival times, they receive lower delay and vice versa. Delay-Earliest Due Date (D-EDD) is an extension of EDD [1], in which case, each source agrees on a service contract with the scheduler. The server sets a deadline for the packet as the expected arrival time added to the delay bound. If the source disobeys then each packet receives worst-case delay, which is lower than the delay bound guaranteed. However, in this case, the packets should be placed in a priority queue as in WFQ. The scheduler also has to store finish numbers as in WFQ. Thus, it is as complex as WFQ, though it does not have to calculate the round number. Jitter-Earliest Due Date (J-EDD) algorithm provides end-to-end bandwidth, delay and delay-jitter bounds by trying to provide the same delay to all the connections for every hop, except the last one. After a packet is served by a server, it is stamped with the difference between its deadline and actual finishing time. A regulator at the entrance of the next switch holds the packet for this period before it is sent to the scheduler to be served. However, a connection should reserve highest bandwidth to obtain the worst-case delay bound. Earliest Deadline First (EDF) is an optimal scheduler for bounded-delay services. But the implementation requires sorting of packets which makes it complex for implementation in high speed networks. It is a dynamic scheduling algorithm for real-time scheduling purposes.

Rotating Priority Queues (RPQ) scheduler [23] is a hybrid of EDF and Static Priority (SP) scheduling. It has high efficiency (like EDF) and low complexity (like SP). Here, the scheduler has a set of prioritized FIFO queues and the scheduler, periodically changes the priorities of the FIFO queues. The scheduler transmits a packet from the highest non-empty priority FIFO queue. But it has a rotation anomaly that if a packet

resides in the highest priority queue at the time of queue rotation, it will be in the lowest priority queue. RPQ+ scheduler [24] approximates EDF with rotating FIFO queues. The idea is to have twice the number of FIFO queues as RPQ and add a newly arriving packet to the queue between the FIFO queues of RPQ. These queues are called the intermediate queues. Though this increases the cost, it is highly efficient.

The Stop-and-Go (SG) discipline [25] uses the framing strategy. Time is divided into frames and in each frame time, only those packets that arrived in the previous frame time are served. That is, it ensures that packets on the same frame at the source stay in the same frame throughout the network. It provides a bound on buffer space requirement and jitter. However, it is not possible to achieve low delay bound and fine granularity of bandwidth simultaneously since it uses the framing strategy. The Hierarchical Round Robin (HRR) is similar to stop-and-go since it also uses framing strategy [15]. It uses multilevel framing strategy. The main difference between SG and HRR is that, in SG the packets are maintained within the same frame throughout the network, whereas HRR has the property that the number of packets within each frame will remain the same from the entrance to the network to the end, but the packets need not be in the same frame inside the network. HRR also has the problem of coupling between delay and bandwidth allocation granularity. It is suitable only for fixed sized packets or cells. Therefore, it can be used only in ATM networks. Another algorithm for the ATM networks is the Carry-Over Round Robin (CORR) [26] which has low implementation complexity since it divides the time line into allocation cycles whose maximum length is fixed, and is not a function of number of connections. Its delay performance is comparable to that of PGPS

and SG. It also achieves near perfect fairness. The performance of CORR in terms of delay jitter is much worse than that of SG.

The Rate Controlled Static Priority Queueing (RCSP) [16] is similar to the Rate Controlled Service Disciplines (RCSD) in that, it has a separate rate controller and scheduler. The scheduler used in this case is the static priority scheduler. The rate controller can either be a rate-jitter regulator or a delay-jitter regulator as discussed in the earlier sections of this chapter. This scheme provides bounded delay, bounded delay jitter, decoupled delay and bandwidth allocation, and uniformly distributed buffer space.

## 2.5 Concluding Remarks

Among all the scheduling disciplines seen above, the WFQ is commonly used in current day networks and the $WF^2Q+$ is the most accurate approximation of GPS in terms of fairness and delay guaranteed. Thus these two scheduling disciplines are chosen for implementation in this thesis and their delay and fairness properties are compared.

# Chapter 3

# 3. WFQ and $WF^2Q$ Scheduling Disciplines

## 3.1 Introduction

After briefly introducing most of the scheduling disciplines known in the literature in Chapter 2, this chapter discusses in detail two, or, in some sense, three scheduling disciplines. The first of these disciplines is the well-known Weighted Fair Queueing (WFQ), which is understood to be the closest approximation of Generalized Processor Sharing (GPS). Although recently, Worst-case Fair Weighted Fair Queueing ($WF^2Q$) is demonstrated to be a better emulation of GPS, the popularity of WFQ still persists. Another scheduling discipline, $WF^2Q+$, which is a slight improvement of $WF^2Q$, is also considered here. The two disciplines that are implemented and analyzed for their performance in this thesis are WFQ and $WF^2Q+$.

For the best-effort connections, an ideal work-conserving scheduling discipline that can achieve a max-min fair allocation is the GPS. GPS assumes packets of each connection to be in separate queues. At any instant of time, the server serves an infinitesimally small amount of data from each backlogged queue simultaneously. It is also possible for connections to have weights. In this case, the server serves an amount of data, from each connection, which is proportional to its weight. In the case of real-time connections, the GPS has to be leaky bucket constrained to make the discipline non-work-conserving. In spite of the fact that GPS is ideal, it is unimplementable because it assumes

packets are infinitely divisible, which is not the case in practice. WFQ, and later on $WF^2Q$, were introduced as emulations of GPS.

## 3.2 Definition

Before going into the definition of WFQ and $WF^2Q$, a clear understanding of the scheduling discipline on which these two disciplines are based is required. In this regard, the GPS discipline is exemplified. Assume that fixed-sized packets of size 1 byte (for ease), from 11 different connections arrive at the server with the packet arrival pattern shown in Figure 3.1a [10]. We shall use notation $p_i^j$ to represent $i^{th}$ packet arriving at $j^{th}$ connection. The packets from these connections (marked C1 to C11) are destined to the same output link and therefore, share the link capacity, which is 1 byte/ms. The weights that are guaranteed during connection set up are: connection 1 has a weight of 0.5 while the remaining 10 connections have a weight of 0.05 each, summing up to a total weight of 1.0. Eleven back-to-back packets from connection 1 and, one packet from each of the other 10 connections are queued at time 0 (shown along the x-axis).

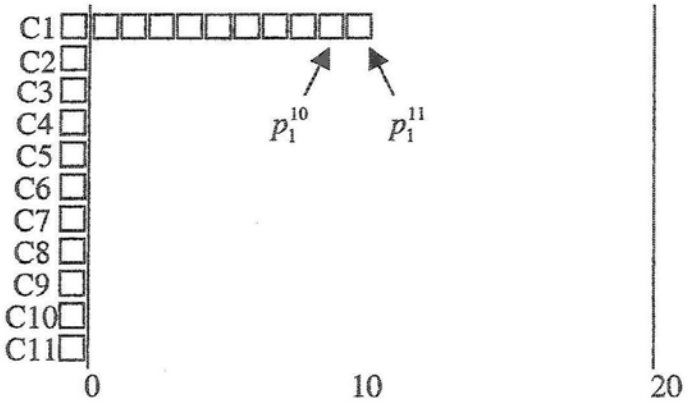



**Figure 3.1a:** Packet arrival pattern

The manner in which GPS schedules packets is shown in Figure 3.1b. The GPS discipline takes two time units to serve each packet from connection 1 and 20 time units to serve each packet from each of the other 10 connections in order to provide a fair share to all the 11 connections based on their weights. However in practice, it is only possible to serve a packet of size 1 byte, into a link whose capacity is 1 byte/ms, in one time unit (i.e. 1 ms). Therefore, GPS is not practically realizable. With this idea of GPS, the WFQ and $WF^2Q$ are defined and illustrated with the same example below.

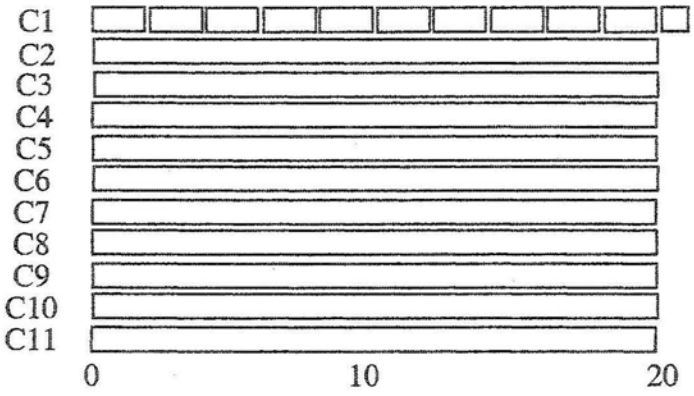



**Figure 3.1b:** GPS service order

## 3.2.1 Weighted Fair Queueing (WFQ):

In the WFQ discipline, when the server is ready to transmit the next packet at time $\tau$, it selects, among all the packets queued at $\tau$, the first packet that would complete service in the corresponding GPS system if no additional packets were to arrive after time $\tau$ [11]. For a better understanding of the working of WFQ algorithm, consider again the packet arrival pattern shown in Figure 3.1a. At time 0, since, in the GPS system, the first packet to finish service is packet $p_1^1$ (the first packet from all the other connections $p_i^1$

with $i = 2....11$, finish service at time 20), and so, this packet is served first. Similarly, the first 10 packets of connection 1 are served back to back before packets on other connections can be served. This is illustrated in Figure 3.1c.

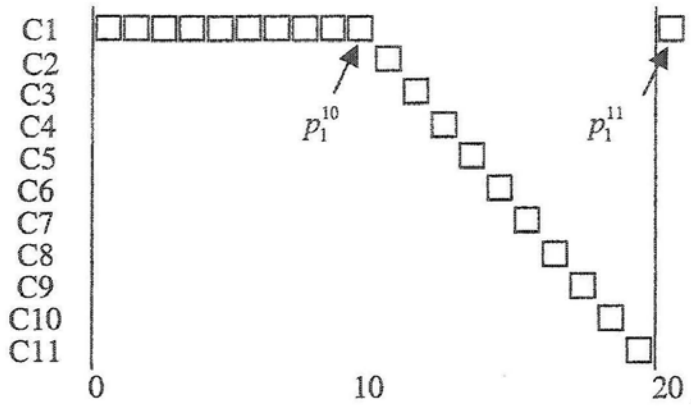



**Figure 3.1c:** WFQ service order

After serving all the 10 packets from connection 1, the 11th packet from this connection has a finish time which is higher than that of the first packet of the remaining 10 connections and so, this packet is not served next. Instead the first packets from each of the remaining 10 connections are served next in order of increasing connection number. Finally, the 11th packet of connection 1, $p_1^{11}$, is transmitted. Thus, in order to determine the next packet to serve, WFQ algorithm uses the GPS finish times of packets. On the other hand, WF$^2$Q uses the GPS finish as well as the start times of packets in order to determine the next packet to transmit.

## 3.2.2 Worst-case Fair Weighted Fair Queueing (WF$^2$Q):

In WF$^2$Q, when the next packet is chosen for service at time $\tau$, rather than selecting it from among all the packets at the server as in WFQ, the server only considers

the set of packets that have started (and possibly finished) receiving service in the corresponding GPS system at time $\tau$, and selects the packet among them that would complete service first in the corresponding GPS system [11]. Referring to the same example under consideration, the $WF^2Q$ serves the packets in the order shown in Figure 3.1d.

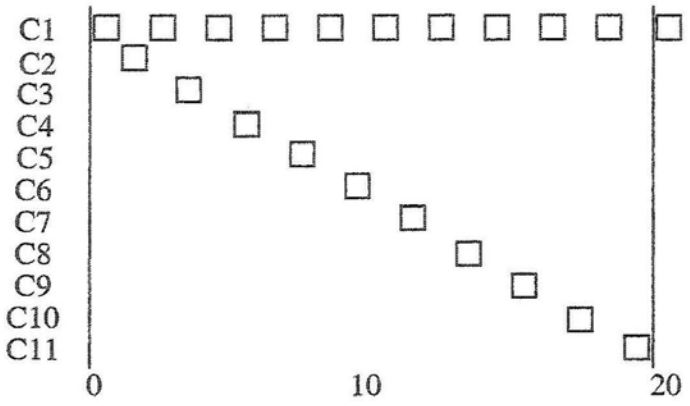



**Figure 3.1d:** $WF^2Q$ service order

A time 0, the first packets in all the connections start service in the corresponding GPS system. The second and subsequent packets, $p_1^i$, for $i = 2...11$, from connection 1 have not yet started service in the GPS system and therefore are not considered while selecting the next packet to transmit. Among the packets that are considered for selection, the packet that has the least finish time is the first packet of connection 1, $p_1^1$, and therefore, is served at time 0. At time 1, though the second packet of connection 1, $p_1^2$, has the least finish time among all the remaining packets, it is not considered for selection, since it does not start service until time 2 in the GPS system. Therefore, the next packet with the least finish time is the first packet of connection 2, $p_2^1$, and so, it will be served at time 1. At time 3, the second packet of the first connection would have started service and is

therefore considered for selection. Since it has the least finish time among all the packets considered for service, it is served next and so on. Thus there is a significant difference in the service provided by a $WF^2Q$ system compared to WFQ system as the $WF^2Q$ system selects the next packet to transmit based on the GPS finish as well as start times. The first packet from connection 2 $p_2^1$ is served much earlier in the $WF^2Q$ scheme than in the WFQ scheme. Similar explanation holds for packets of other connections too. Thus, $WF^2Q$ scheme is fairer WFQ scheme, not only with regard to real-time source but also with regard to packets from all other connections.

## 3.3 Properties

There are several properties of WFQ and $WF^2Q$ that need to be discussed to get a better understanding of the algorithms and also to conclude on a better algorithm so that they can be applied to real-time traffic in high-speed packet switching. Firstly, the reason all these disciplines try to approximate GPS is because GPS has two important properties [27]: (1) it can guarantee bounded end-to-end delay to connections and (2) it allocates bandwidth available to all the connections in a fair manner despite the consideration of whether they are rate-controlled or not. The following properties will be discussed in detail for the above two scheduling disciplines:

- System virtual time function
- Packet selection policy
- Implementation complexity
- Accuracy

- End-to-end delay and buffer space requirements
- Traffic characterization.

### 3.3.1 System Virtual Time Function

The fair queueing algorithms considered here, have to maintain a system virtual time $V(\cdot)$, a virtual start time $S_i(\cdot)$ and a virtual finish time $F_i(\cdot)$ for each connection $i$. The virtual start and finish times are updated every time a packet arrives or leaves the server. Every time an unbacklogged connection becomes backlogged or vice versa, the system virtual time is updated. The complexity and accuracy of any scheduling algorithm is based on that of its virtual time function. If the service provided by any scheduling algorithm matches that of GPS, then its virtual time function is said to be accurate.

The virtual time function of WFQ is defined based on the GPS virtual time function. For the WFQ system, let $t_j$ be the time at which the $j^{th}$ event occurs [8]. An event is defined as an arrival or a departure of a packet. The time of the first arrival of a busy period is denoted as $t_1 = 0$. Now, for each $j = 2,3,\ldots$, the set of connections that are busy in the interval $(t_{j-1}, t_j)$ is fixed, and is denoted as $B_j$. During the time at which the server is idle, the virtual time $V(t)$ is zero. Consider a busy period that starts at time zero. Then $V(t)$ is given as below,

$$V(0) = 0;$$

$$V(t_{j-1} + \tau) = V(t_{j-1}) + \frac{\tau}{\sum_{i \in B_j} \phi_i}, \quad \tau \le t_j - t_{j-1}, \quad j = 2,3\ldots$$

where, $\tau$ is the time of the event just prior to $t$ and $\phi_i$ is a positive real number used to characterize connection $i$. The virtual start and finish times for the $k^{th}$ packet of $i^{th}$ connection arriving at time $t_i^k$ with a length of $L_i^k$ is given by,

$$S_i^k = \begin{cases} \max\left(V(t_i^k), F_i^{k-1}\right) & \text{connection } i \text{ becomes active} \\ F_i^{k-1} & p_i^k \text{ finishes service} \end{cases}$$

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

From the above equations, three properties of virtual time interpretation of WFQ can be observed: (1) the virtual finish time can be calculated with the packet's arrival time, (2) packets are served in the order of finishing times and (3) the virtual time needs to be updated only when there are events in the GPS system. These are some of the advantages of the system virtual time function of GPS, and therefore of WFQ. The virtual time function of $WF^2Q$ is also defined with respect to that of GPS and has the same set of properties.

### 3.3.2 Packet Selection Policy

There are two commonly used packet selection policies, namely, Smallest virtual Finish time First or SFF policy and Smallest Eligible virtual Finish time First or SEFF policy [4]. In the WFQ system, when the server is ready to select the next packet to transmit, it selects from among all the packets available in the system, the one with the smallest virtual finish time and it thus belongs to the SFF policy. In the $WF^2Q$ system, when the server is ready to select the next packet to transmit, it selects from among all the

packets eligible (and not just available) in the system, the one with the smallest virtual finish time. A packet is said to be eligible if it has already started service in the corresponding GPS system, that is, a packet has arrived in the corresponding GPS system, since a packet starts service as soon as it arrives in the GPS system. Thus it can be concluded that $WF^2Q$ employs the SEFF policy.

Although scheduling algorithms that use the SFF policy assure delay bounds matching that of GPS, they still produce large service discrepancies from GPS. This is explained in detail in Section 3.3.4.

### 3.3.3 Implementation Complexity

There are three important costs involved in scheduling [4]: (1) the cost of computing the system virtual time function, (2) the cost of handling a queue for ordering the packets to be scheduled and (3) the cost of maintaining the queue to regulate the packets. The packets that need to be queued in the regulator have to be sorted based on their eligibility times and then placed in the queue. This has a complexity of $O(N)$, where $N$ is the number of connections. However, using calendar queues or some other mechanism to reduce the complexity of sorting can reduce this complexity [4]. Thus the complexity can be brought down to $O(\log N)$. Similarly, the packets waiting to be sent through the output link need to be queued in the output buffers. These packets are queued on a per-connection basis in the order of increasing virtual finish times. In this case, only the head of each queue (there is one queue per connection) need to be considered to pick the next packet for transmission and so the complexity is again $O(N)$. This can also be

reduced to $O(\log N)$ in the manner described previously. Thus it is possible to maintain at least several hundreds of connections at high speed. The only cost that cannot be reduced is the cost of computing the system virtual time function because both the algorithms, WFQ and $WF^2Q$, follow GPS to calculate the system virtual time function and according to the GPS system, the server is capable of serving data from all the connections simultaneously if all of them are backlogged at any instant of time. This implies that the server should be able to update the system virtual time $N$ times in the worst case, if $N$ connections have an event (connections become backlogged or unbacklogged) at the same time.

### 3.3.4 Accuracy

Parekh showed that the delay bound provided by WFQ is within one packet transmission time of that provided by GPS [8]. According to Parekh, who introduced WFQ, the relationship between GPS and WFQ are as listed below [10]:

- in terms of delay, a packet will finish service in a WFQ system later than in the corresponding GPS system no more than the transmission time of one maximum sized packet;
- in terms of the total number of bits served for each connection, a WFQ system does not fall behind a corresponding GPS system by more than one maximum sized packet.

This leads to the interpretation that WFQ discipline and the GPS discipline provide almost indistinguishable service except for a difference of one packet. The

Internet Engineering Task Force, a standards development body for Internet, recently proposed WFQ as a reference server for Internet supporting guaranteed service class, based on the above result. According to Bennett and Zhang, the above interpretation is erroneous [10]. According to them, there is a large inconsistency between the services provided by WFQ and GPS and this inconsistency affects the fairness of WFQ, thereby making it inaccurate. Consider the Figure 3.1c, showing the service order of WFQ service discipline. In this figure, a burst of 10 packets is served from connection 1, then the connection is idle for some time and then repeats itself. This kind of oscillation caused by burstiness in the packets entering the link affects the delay bound guaranteed for real-time traffic and causes unfairness in the service provided to best-effort connections. The reason for such an inaccuracy in WFQ is due to the fact that the service provided by WFQ to a connection (connection 1 in this case) is much more than that provided by GPS. In the example considered in Figures 3.1, within the first 10 time units, WFQ serves 10 packets from connection 1 while GPS serves only 5 packets. Thus WFQ is well ahead of GPS in the amount of service provided during any interval of time. This causes WFQ to be inaccurate and the inaccuracy may be as high as N/2 packets, where N is the number of connections in the switch. This is not the case with $WF^2Q$, which serves 5 packets from connection 1 within the first 10 time units, which is the same as that by GPS. Thus $WF^2Q$ serves within one packet transmission time of that of GPS in this example.

**Worst-case Fair Index (WFI):**

In order to have a tight delay bound, Worst-case Fair Index (WFI) is used to characterize the scheduling disciplines. A service discipline *s* is called worst-case fair for

connection $i$ if for any time $\tau$, the delay of a packet arriving at $\tau$ is bounded above by [10]

$$\frac{Q_{i,s}(\tau)}{r_i} + C_{i,s}.$$

That is,

$$d_{i,s}^k < a_i^k + \frac{Q_{i,s}(a_i^k)}{r_i} + C_{i,s},$$

where, $d_{i,s}^k$ is the delay of the $k^{th}$ packet in the $i^{th}$ connection at server $s$, $a_i^k$ is the arrival time of the $k^{th}$ packet in the $i^{th}$ connection, $r_i$ is the throughput guarantee to connection $i$, $Q_{i,s}(a_i^k)$ denotes the queue size of connection $i$ at time $a_i^k$, and, $C_{i,s}$ is a constant independent of the queue size of other connections. $C_{i,s}$ is called the worst-case fair index for connection $i$ at server $s$. Since $C_{i,s}$ is measured in absolute time, it is not suitable for comparing $C_{i,s}$'s of connections with different $r_i$'s. To perform such a comparison, the normalized worst-case fair index for a connection $i$ at server $s$ is given as,

$$c_{i,s} = \frac{r_i C_{i,s}}{r},$$

where, $r$ is the link speed or output link rate.. The normalized worst-case fair index of server $s$ is given by,

$$c_s = \max_i \{c_{i,s}\}.$$

For GPS, $c_{GPS} = 0$ and hence worst-case fair. The WFI of WFQ is a function of the number of connections and is given by,

$$c_{WFQ} \geq c_{1,WFQ} \frac{r_1}{r} = \frac{N-1}{2} \frac{L_{\max}}{r},$$

where, N is the number of connections at server $s$ and $L_{max}$ is the maximum packet size.

However, $WF^2Q$ is worst-case fair and its WFI is given by

$$C_{i,WF^2Q} = \frac{L_{i,\max}}{r_i} - \frac{L_{i,\max}}{r} + \frac{L_{\max}}{r},$$

where $L_{i,max}$ is the maximum packet size of connection $i$.

The normalized WFI is given by,

$$c_{WF^2Q} = \frac{L_{\max}}{r}.$$

This algorithm has a WFI smaller than most of the known algorithms. It is because of this reason that $WF^2Q$ got its name.

### 3.3.5 End-to-end Delay and Buffer Space Requirements

For WFQ and $WF^2Q$, the traffic specifications carried by the source at the entrance to the network is sufficient to provide end-to-end delay bound. In order to achieve a bound on end-to-end delay the rate of packet arrival must be guaranteed and this cannot be significantly less than the connection's average rate. Also, in order to prevent packet loss, a large buffer space needs to be allocated to the connection during call set-up. Therefore the crux of the problem is that there is a coupling between the bandwidth and end-to-end delay provided to each connection. A high bandwidth should be allocated for low end-to-end delay bound, but this, results in waste of resources if the low delay connection also has low throughput. This problem is avoided by separating the rate-control mechanism from the scheduling mechanism. In this thesis separate regulator and scheduler are used to overcome this problem. Inclusion of a regulator results in lower buffer space requirements at each node. This thesis analyzes only the delay bound of the

real-time connections and therefore no limit is placed on the buffer available for each connection at each node. In future the same simulator can be used to study the loss rate by limiting the buffer.

### 3.3.6 Traffic Characterization

In order to provide end-to-end delay bound, the local delay bound should first be obtained for each switch and then these delays can be summed to obtain the end-to-end delay bound. For this, the traffic should be characterized on a per-connection basis at every switch in the network. Although this is possible, a problem arises when there is traffic distortion inside the network. This would destroy the traffic characterization and so, this thesis uses a rate-controller in front of the scheduler (WFQ or $WF^2Q$) at every switch in the network to re-characterize the traffic entering the node, thereby overcoming the distortion caused by the network.

## 3.4 Discussion of Properties

From the discussion in Section 3.3 on the properties of WFQ and $WF^2Q$ disciplines, there are several advantages and disadvantages of the two disciplines. Firstly, both WFQ and $WF^2Q$ have system virtual time function which is based on that of GPS. But the complexity involved in updating the virtual time function is $O(N)$ in the worst case, where $N$ is the number of connections, as it has to keep track of the number of active sessions in the corresponding GPS system. This makes both WFQ and $WF^2Q$ unfit for

implementation in high-speed packet switched networks when the number of connections is large. Moreover, WFQ is worst-case unfair and therefore, it is inaccurate. $WF^2Q$ on the other hand, is worst-case fair and hence, accurate. Thus there is a need for another scheduling discipline that would be accurate and at the same time has a lower complexity so that it is feasible to operate at high speeds. Such an algorithm is the $WF^2Q+$, introduced by Bennett and Zhang [21], that has a more accurate virtual time function which provides low complexity, small WFI and low end-to-end delay bound.

$WF^2Q+$ uses a new system virtual time function $V_{WF^2Q+}(\cdot)$ is given by [21]:

$$V_{WF^2Q+}(t+\tau) = \max\left(V_{WF^2Q+}(t) + W(t,t+\tau), \min_{i\in\hat{B}(t)}\left(S_i^{h_i(t)}\right)\right)$$

where, $W(t, t+\tau)$ - total amount of service provided by the server during the period $[t, t+\tau]$

$\hat{B}(t)$ - set of sessions backlogged in the $WF^2Q$ system at time $t$

$h_i(t)$ – sequence number of the packet at the head of the session $i$'s queue

$S_i^{h_i(t)}$ - virtual start time of packet.

The virtual time function is a function of the amount of service and it increases with time with a minimum slope of 1. That is, it provides delay bounds to rate-controlled sources that are within one packet transmission time of that provided by GPS. Also, the virtual time function is such that it is at least as large as the minimum virtual start time. That is, if an unbacklogged connection becomes backlogged, it has a virtual start time that is at least as large as one of the already present backlogged connections, thus realizing low WFI. Also, even though the packets are being held until they are eligible (until they have started service in the corresponding GPS system) to be selected, this algorithm is

work-conserving as it ensures that there is at least one packet which has a virtual start time that is no greater than the current system virtual time. Thus this algorithm ($WF^2Q+$) maintains the same SEFF policy as that of $WF^2Q$.

Another advantage to be appreciated in $WF^2Q+$ is that, there is no need to maintain the virtual start and finish times on a per packet basis. Instead, it is sufficient to have just one pair of virtual start and finish times ($S_i$ and $F_i$ respectively) for each connection. When a packet is about to be served, the start and finish times are updated according to the following equation [21]

$$S_i = \begin{cases} F_i & \text{if } Q_i(a_i^k -) \neq 0 \\ \max\left(F_i, V(a_i^k)\right) & \text{if } Q_i(a_i^k -) = 0 \end{cases}$$

$$F_i = S_i + \frac{L_i^k}{r_i},$$

where, $Q_i(a_i^k -)$ is the queue size of session $i$ just before time $a_i^k$, $V(a_i^k)$ is the system virtual time at $a_i^k$, $L_i^k$ is the length of the $k^{\text{th}}$ packet on connection $i$ and $r_i$ is the guaranteed rate for connection $i$.

The two jobs of computing the system virtual time function, which has been reduced by the use of new system virtual time function which does not depend on the GPS system, and maintaining a queue for storing the sorted virtual finish times, which can be done in $O(\log N)$ using the calendar queue implementation discussed in Section 5.2.7, can be done in $O(\log N)$ complexity.

## 3.5 Concluding Remarks

The only difference between $WF^2Q$ and $WF^2Q+$ is that $WF^2Q$ uses a system virtual time function that emulates the GPS system, but $WF^2Q+$ uses a system virtual function that is calculated from the packet system itself. $WF^2Q$ is an accurate approximation of GPS and $WF^2Q+$ has all the properties of $WF^2Q$ along with the advantage of achieving all the properties and a lower complexity.

# Chapter 4

# 4. Software Implementation

## 4.1 Introduction

In the previous chapter, the scheduling disciplines that are analyzed in this thesis were discussed. This chapter discusses the software implementation details and the results obtained for the WFQ and $WF^2Q+$ disciplines. The network has been modeled so that the source under observation travels through three nodes with cross-traffic from every node. The traffic flow pattern in real networks is imitated as closely as possible. For variable-sized packets, the arrival pattern is based on the Internet traffic observed over a period of time. Thus the traffic arrival pattern reproduces the actual flow of traffic seen in the network of current day. The basic flow of the software implementation is shown and details about the working of each block are also presented. The software simulator is written in C++ and the code is made as modular as possible. The delays obtained by real-time data under various traffic loads and under the presence and absence of cross-traffic are presented.

## 4.2 Network Model

The network is modeled to replicate a portion of the entire network in a smaller version and with reference to the network model used by Bennett and Zhang in their

paper [21]. The network model chosen is shown in Figure 4.1. There are three nodes, named N1 to N3, in the network. The source under observation is the real-time connection, named RT, which has its source at node 1 (N1) and its destination at node 3 (N3). The best-effort traffic, named BE, also has its source at N1 and destination at N3. There is cross-traffic at nodes 2 and 3. The cross-traffic entering these nodes is composed of Poisson sources (PS1 to PS40) and/or constant sources (CS1 to CS10) and they interact with the packets entering the node from N1 containing real-time and best-effort traffic. This interaction may cause distortion in the traffic entering a node and thereby increase the delay in a particular node. This kind of cross-traffic is chosen intentionally to analyze the performance the two scheduling disciplines can guarantee to the real-time source in a networking environment when they experience disturbances as in a real network.

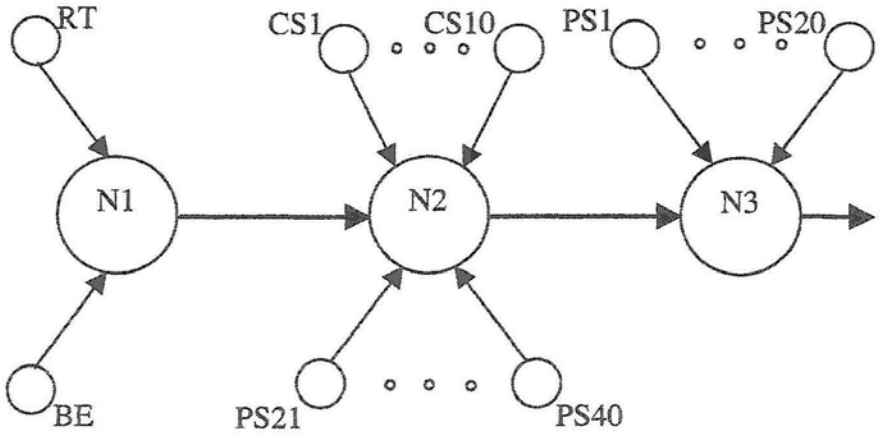



**Figure 4.1:** Network model

## 4.3 Traffic Model

This section details the traffic arrival pattern for each of the sources mentioned in the previous section. The weights are assigned to sources based on the bandwidth

guaranteed to each source. This detail is obtained from the paper by Bennett and Zhang [21], so as to verify the results obtained. Thus as an example, if the total link rate available is 45 Mbps, 30 Mbps is assigned to various connections. The remaining 15 Mbps is not assigned to any connection but is used when the traffic arrival rate in any connection exceeds its guaranteed rate. The real-time source is assigned 9 Mbps, each of the Poisson sources entering N2 (PS1-20) is assigned 500 Kbps, each of the Poisson sources entering N3 (PS21-40) is assigned 333 Kbps, each constant source entering N2 is assigned 333 Kbps and the best-effort source is assigned 1 Mbps making a total of 30 Mbps. The real-time source is a deterministic ON-OFF bursty source with an ON period of 5 packets/burst. The source consists of an ON period (active period) followed by an OFF period (idle period). In our case, the real-time source contributes to 20% of the total traffic entering the network at nodes 1, 2 and 3. The source has a weight of 1000 and acts as connection 1. There are a total of 52 connections in the network with source node for each of these connections being one of the three existing nodes and destination node being N3 for all the 52 connections. The length of ON and OFF period of the bursty traffic source is calculated by [28],

$$L = 1 + \left\lceil \frac{\ln(1-R)}{\ln(1-p)} - 1 \right\rceil,$$

where, $p$ = 1/average burst period (active or idle) length, $0 \le R \le 1$ is a random number generated, and $0 < p \le 1$ is the inverse of the average ON or OFF period length in packets.

Connections 2 to 21 are Poisson sources (PS 21-40) each with a weight of 37 and an inter-arrival time whose probability density function (pdf) is depicted in Figure 4.2.

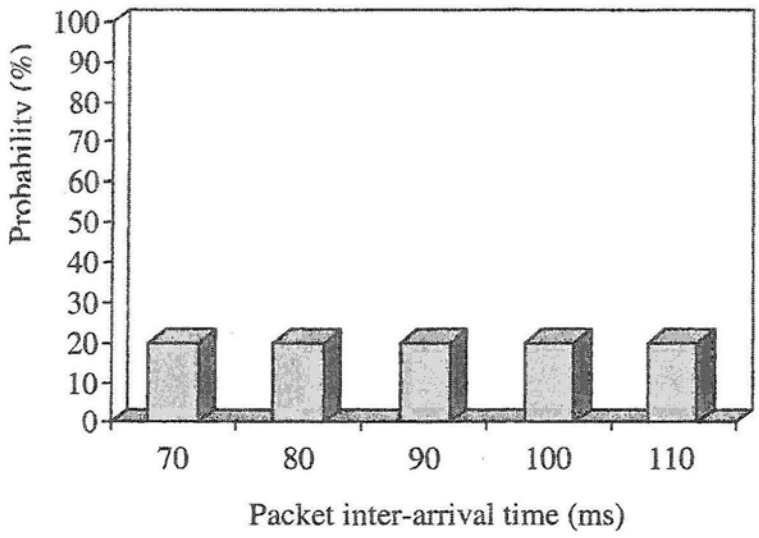



**Figure 4.2:** Inter-arrival time probability density function for PS 21-40 with weight 37

Thus, from Figure 4.2, the average inter-packet arrival time for PS 21-40 is 90 cycles, which means, packets entering N2 from these sources are spaced 90 cycles apart on average. A cycle duration of 1 ms is assumed in this thesis. Thus if the link rate is $l$ bytes/ms, it means that the link can carry $l$ bytes in 1 ms or, in our case, in 1 cycle. Connections 22 to 41 are also Poisson sources (PS 1- 20) each with a weight of 55 and an average inter-arrival time of 60 cycles, whose pdf is shown in Figure 4.3. For the above two Poisson sources, the stress is mainly on the average packet inter-arrival time and so, the inter-arrival times are distributed close to the average inter-packet arrival times. Connections 42 to 51 are constant sources, each having an inter-arrival time of 135 cycles with a hundred percent probability and with a weight of 37. The last connection (52) is the best effort source having a weight of 111. The best-effort source is expected to be backlogged continuously.

Simulations are run for various offered loads based on various arrivals of best-effort connection and, with and without the presence of cross-traffic from constant sources. The simulation details and results are discussed in Section 4.7 of this chapter.

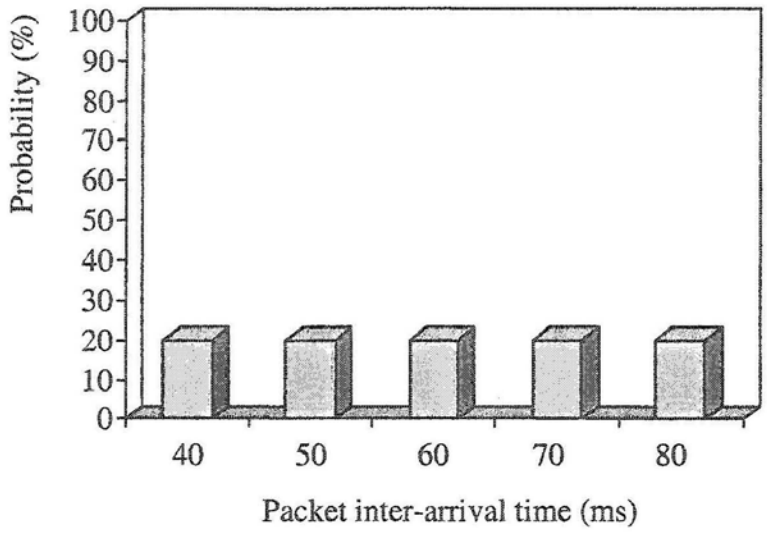



**Figure 4.3:** Inter-arrival time probability density function for PS 1-20 with weight 55

All the constant-rate connections have relatively random arrival times. The following parameters are chosen to characterize offered load: $X_{min}$, $X_{ave}$, I. The minimum packet inter-arrival time is $X_{min}$, $X_{ave}$ is the average packet inter-arrival time over an interval of duration I. The incoming connections are made to obey these restrictions on the input traffic by the use of the regulator (refer to Section 4.5.4). The switch is assumed to be non-blocking, that is, when packets arrive at the input link, they can be routed directly to appropriate output links without switching conflicts. Queueing occurs only at the output port of the switch.

# 4.4 Packet Length Distribution

Bennett and Zhang, in their simulations, assume packets to be of fixed size for ease of implementation [21]. In this thesis, both fixed and variable-sized packets are considered so that the switch can be used for high-speed packet switched networks and not just ATM networks. The packet length distribution used is obtained from the Internet traffic observed over approximately 84 million packets by Traffic CAIDA (Co-operative Association for Internet Data Analysis) organization for the years 1997-2000 at NASA Ames Internet Exchange (AIX). The results obtained were consistent and can be expressed in the Figure 4.4a [29]. Figure 4.4a shows a plot of the packet size and their arrival as a cumulative distribution.

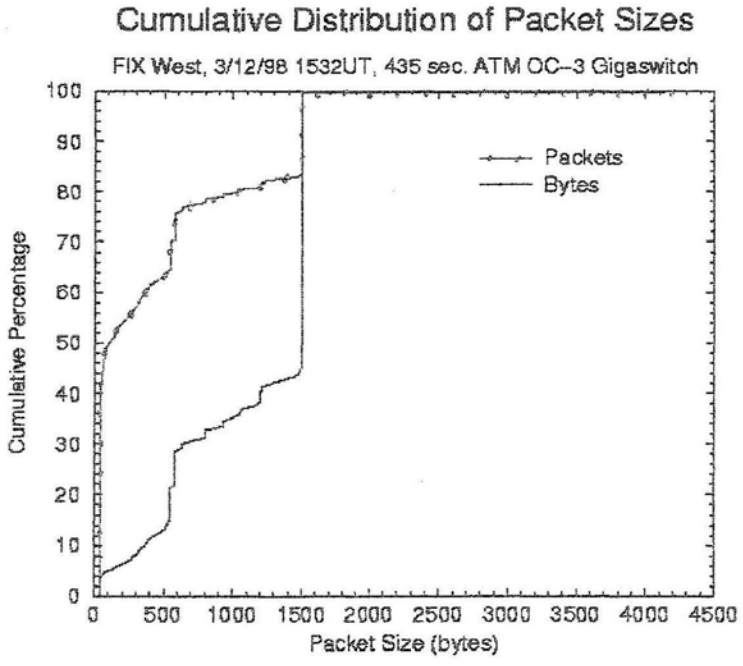



**Figure 4.4a:** Cumulative distribution of packet sizes [29]

From the plot, it can be seen that 50% of the packets have lengths ranging between 40 and 44 bytes and close to 75% of the packets have length less than 552 bytes. Also, less than 0.005% of the packets have lengths greater than 1500 bytes, and thus ignored. The maximum packet size can thus be assumed to be 1500 bytes. Packets of length 40 bytes correspond to TCP (Transmission Control Protocol) since the minimum packet size for TCP is 40 bytes. The plot has been interpreted by Traffic CAIDA organization as follows: 10% of the packets are of length 1500 bytes (which is also the maximum packet size), 5% of the packets vary between the lengths 550 bytes and 1500 bytes, 10% of the packets are of length 552 bytes, 15% of the packets range between the lengths 44 bytes and 500 bytes, and, 60% of the packets have lengths ranging between 40 and 44 bytes. The 40-44 byte packets are usually acknowledgement packets, and, they occur frequently.

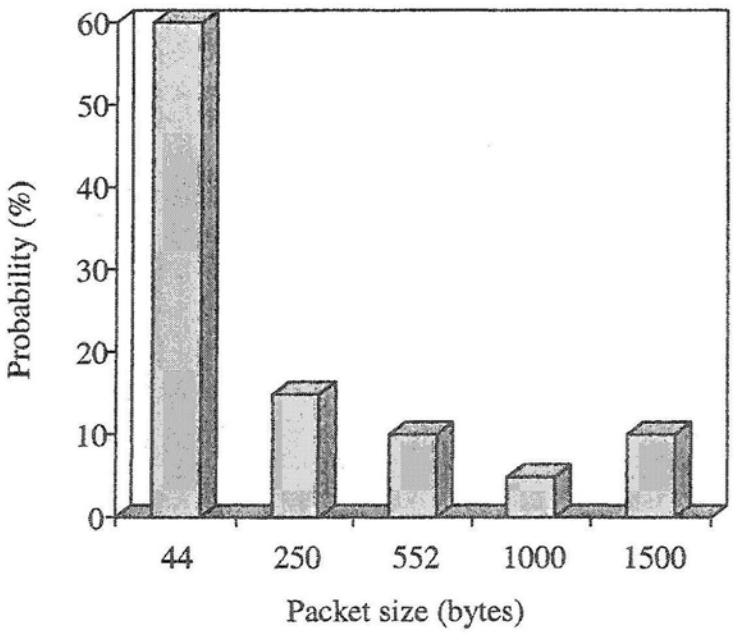



**Figure 4.4b:** Packet length distribution

The plot shows the result obtained for the year 1998. The results obtained for the years 1997, 1999 and 2000 are quite similar to the results obtained for the year 1998 and so, the

simulation conducted in this thesis assumes the packet sizes to follow the same distribution. This distribution is shown in Figure 4.4b, which shows the cumulative distribution converted into a pdf based on the stated approximations. The variable-sized packets arriving through each connection entering a node follow this distribution.

## 4.5 Implementation

The software simulator framework is obtained from Mehrotra's work [30] and modified and augmented based on the needs of the specific implementation and analysis details required, while preserving the modularity of the simulator. The block diagram of the software simulator is shown in Figure 4.5. The various blocks involved in the software implementation are traffic generators, input buffer, input and output links, rate controller (regulator) and scheduler. Each of these blocks is discussed below.

### 4.5.1 Traffic Generator

The traffic generators read the pdf information for each connection from the corresponding data file. There are two pdf files, one containing the probability information about the length (as discussed in Section 4.4 of this chapter), named *length_pdf.dat*, and the other containing probability information about the inter-arrival time between packets (as discussed in Section 4.3 of this chapter), named *iat_pdf.dat*. The traffic generator for each connection reads the connection's details from an initialization file called *sessionN.ini* where *N* is connection number.

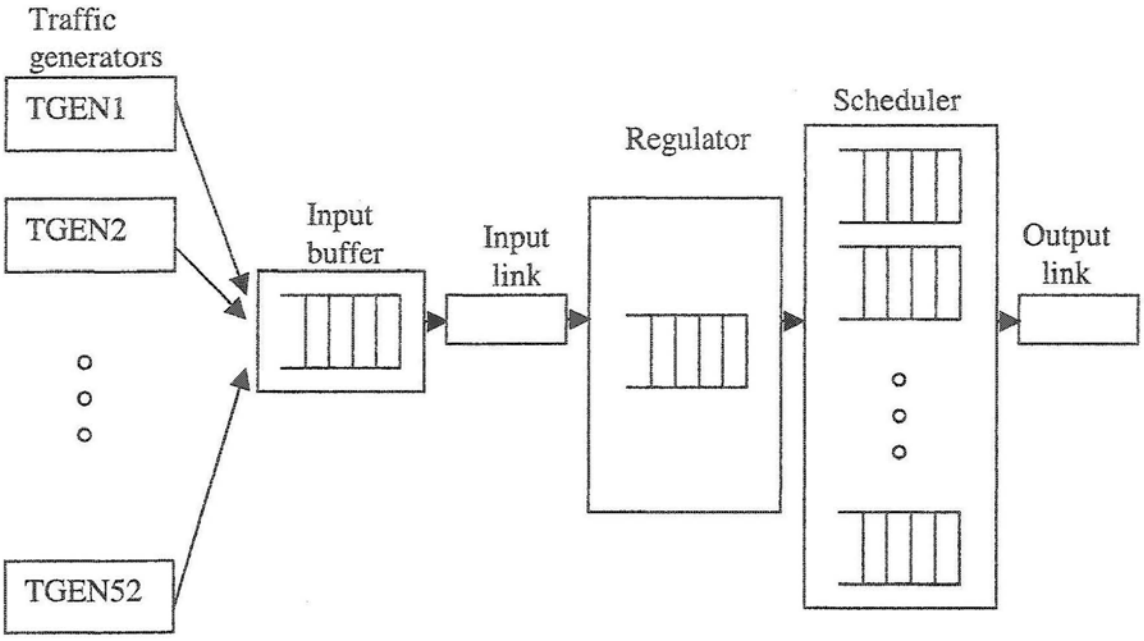



**Figure 4.5:** Block diagram of the scheduling simulator (software implementation)

In this file, the details such as the name of the file where the packet length details are found (*length_pdf.dat*), the name of the file where the packet inter-arrival time details are found (*iat_pdf.dat*), the source and destination nodes for the connection and most importantly, the connection's restrictions such as values of $X_{min}$, $X_{ave}$ and I that are allowed can be obtained. Based on the details read from the pdf files, the packets are generated randomly following the constraints of the pdf files.

The traffic generator is implemented as a Finite State Machine (FSM). It remains in one of the two states, *generate_packet* or *wait_for_next_packet*. During the *generate_packet* state, the traffic generator generates a packet and then finds the next packet arrival time randomly based on the pdf obtained from the files. When the packet is generated, a pointer is assigned to it and this packet pointer is sent to the next block instead of the packet itself. During the *wait_for_packet* state, a counter counts down during each cycle, until the time for the next packet arrives.

### 4.5.2 Input Buffer

The input buffer is used to temporarily store the packet pointers. There is only one input buffer for each node, which stores packet pointers from all the connections. When a packet is generated by the traffic generator, the control unit (to be discussed in a later subsection) obtains the packet's pointer and stores it in the input buffer. The packet passes through the input buffer without any queueing only when the input link has sufficient capacity to remove the packet immediately from the input buffer. The switch is assumed to be ideal and non-blocking. Also, an output-queued switch is assumed. The input buffer is used in this case only to temporarily store the packets generated, until they can be sent into the input link. Similarly, packets leaving the output link of one node enter the input buffer of the next node in the network and remain there, until the input link of the next node is ready to send the packets through it.

The control unit takes the packet pointer from the traffic generator once it is generated and sends it to the input buffer for storage. The input buffer accepts the packet pointer sent by the control unit and stores it in the buffer using the function *store_packet_pointer( )* and retrieves the packet pointer back to the control unit through the function *get_packet_pointer( )* if the input link is available to carry the packet forward into the next block. If the input link is not available, the packet remains in the input buffer. This thesis assumes the switch to be non-blocking and so, the input link is always available to carry packets to the regulator. We have used an input buffer, though it is not required, for the purpose of processing the packet arrival details obtained from the file

(*pif.dat*), which contains the packet arrival pattern captured from the previous simulation, in order to compare the two schemes.

### 4.5.3 Transmission Link

There are two instances of the transmission link. One is the input link and the other is the output link. The input link transports the packets from the traffic generators to the regulator, or, to the scheduler in the case of a work-conserving scheduler. The output link removes the packets from the scheduler and sends them to the input buffer of the next node or, in case of the last node, destroys the packet. The transportation of the packet from one block to another through the transmission link is based on the link capacity of the transmission link. In case the transmission link is not able to carry one full packet in one cycle because the link capacity available per cycle is less than the packet length, then the packet will be sent in more than one cycle. The main difference between the input link and the output link is that the input link has a capacity that is four times that of the output link. Actually, this means that there are four links entering each node bringing packets from various connections. In this simulator, instead of having four links, the input link is designed to have four times the capacity of the output link.

The transmission link can be in one of the three states, namely, *idle*, *busy* and *done*. When the link is in state *idle*, it is ready to receive packets from the input buffer (if it is the input link) or from the scheduler (if it is the output link). When the link is in state *busy*, it implies that the link is *busy* sending packets previously received. When the link goes to state *done*, it means, the link has finished sending the packet/packets and will go

to *idle* state next. As before, the transmission link also has two more functions, namely the *store_packet_pointer( )* and the *get_packet_pointer( )*. The function of these two functions is the same as discussed previously.

### 4.5.4 Regulator

The regulator implemented is the rate-jitter controlling regulator, which reconstructs the distorted traffic pattern partially as discussed in Chapter 2. Firstly, the regulator stores the packet pointer in its queue using the *store_packet_pointer( )* function. If sufficient space is available in the packet buffer, then the packet pointer is stored in it. Otherwise, the packet is dropped. In this simulator, for the purpose of delay analysis, the buffer is assumed to be of infinite size, and so, no packet is ever dropped due to buffer overflow. The regulator uses the ($X_{min}$, $X_{ave}$, I) traffic model which is described in Section 2.3.1, which illustrates that the inter-arrival time between successive packets in a connection should be less than $X_{min}$ and the average inter-arrival time of packets during an interval of length I should be no greater than $X_{ave}$. The regulator obtains the value of $X_{min}$, $X_{ave}$ and I for a particular packet pointer's connection by reading the *sessionN.ini* file, where N is the connection number. With these details, the regulator calculates the eligibility time for a packet by calling the function *compute_eligibility_time( )*. The manner in which the eligibility time is calculated is discussed in Chapter 2 where the formula for calculating the eligibility time is given. Once this eligibility time is calculated, another function called *packet_pointer_available()*, called by the control unit, checks to see if the packet pointer is eligible by comparing the eligibility time with the

current system time and if it is eligible, then it is available to be sent to the scheduler. So, the next function *get_packet_pointer( )* removes all the packet pointers that are available one behind the other from the regulator queue and passes them to the control unit. The control unit then passes these packet pointers to the scheduler to be served to the output link.

### 4.5.5 Scheduler

There is one instance (either WFQ or $WF^2Q+$) of the scheduler and the control unit chooses one of the two scheduling disciplines, namely, WFQ and $WF^2Q+$ while running the simulation and initiates only that instance. The constructor of the scheduler reads the scheduler buffer size, the connection's weight, and, source and destination nodes for each connection. The function *store_packet_pointer( )* stores the packet pointer in the connection's queue. There is one queue per connection. There is a function called *compute_finish_number( )*, which calculates the finish number for each packet based on the scheduling algorithm. Once the control unit calls the *get_packet_pointer( )*, the packet with the least finish number is selected to be served in both the cases of WFQ and $WF^2Q+$ algorithms. The only difference between the two algorithms in terms of implementation is that for the $WF^2Q+$, while storing the packet pointer there is no need to find the number of active connections to update the finish number and so, there is no need to go through all the connections once as in the case of WFQ algorithm.

Once the packet pointer has been stored in the scheduler queue, the *packet_pointer_available( )* checks to see if there is a packet pointer available in any of

the connection queues. If one is available, then the *get_packet_pointer( )* removes the packet pointer from the queue. This is done by selecting the packet pointer with the smallest finish number in the case of WFQ and the packet pointer with the smallest finish time in the case of $WF^2Q+$. Moreover, in the case of WFQ, the round number is updated every time a packet arrives.

### 4.5.6 Data Handler

The data handler is a special unit that collects the packet information and then processes this information to produce some useful results like generating the output which indicates the number of packets arrived, the number lost, the minimum, maximum and average delays of each connection, the total traffic load entering each node, etc. It also collects information such as packet length, the connection to which each packet belongs, the node it enters and exits etc. The constructor in the data handler collects the following information from the initialization file, *scheduler.ini*:

- *simulation_end_time* – the number of cycles for which the simulation is to be run
- *number_of_sessions* – the total number of connections in the network
- *number_of_nodes* – the total number of nodes in the network
- *write_session_stat_file* – flag to indicate the data handler to write the connection's details like total number of packets, packets lost, minimum delay, average delay and the maximum delay for each connection in each node, into a file
- *session_stat_file* – the file into which the connection's details are to be written

- *per_packet_info* – display the packet information such as, packet number, connection number, node number, packet length arrival time, time in regulator etc., when the packet is destroyed
- *write_packet_info_file* – write the packet information into a file rather than printing it out
- *packet_info_file* – the file name into which the packet information is to be written
- *time_data* – collect the time data
- *write_time_data_file* – write the time data to a file
- *time_data_file* – the file name into which the time data has to be written

The data handler has a function called *output_results( )*, which stores all the collected information into various files. It is possible to collect the per packet information and store it in a file. This file consists of the details of the packet event times such as *packet_created*, *packet_in_input_buffer*, *packet_leaves_input_buffer*, *packet_on_input_link*, *packet_leaves_input_link*, *packet_in_regulator*, *packet_leaves_regulator*, *packet_in_scheduler*, *packet_leaves_scheduler*, *packet_on_output_link* and *packet_leaves_output_link*. Some of this information such as *packet_created*, *packet_number*, *session_number* and *node_number* are also used in the hardware implementation.

### 4.5.7 Control Unit

The control unit runs the entire simulation by calling the functions of each of the objects it creates. Some of the objects created by the control unit are traffic generator,

input buffer, input link, regulator, scheduler, output link, data handler, packet buffer and simclock. It then initializes each of these objects with the values obtained from the scheduler.ini file. The control unit opens the *scheduler.ini* file, and retrieves information such as the simulation end time, random seed, input link rate, output link rate, number of nodes, number of sessions, traffic type, total buffer capacity, scheduler type – WFQ or $WF^2Q+$, regulator capacity, input buffer capacity and other packet information details. Depending on the scheduler type, the control unit calls one of the two scheduling disciplines. The flow of control unit is illustrated in the flow chart shown in Figure 4.6.

Once the simulation starts, a check is made to find out whether the simulation can be continued or not. In this check, if the simulation end time has already been reached and there are no more packets available anywhere in the simulator (input buffer, input link, output link, regulator and scheduler) then the simulation stops immediately. Otherwise, the simulation continues and the traffic generator is run first, where the packets are generated. Once a packet is generated, the packet pointer is captured and it is stored in the input buffer. This continues until all the generated packets have been stored in the input buffer. Now the input link is checked whether it is idle and ready to receive packets. If it is, then the packet pointers are retrieved from the input buffer and stored in the link. The number of packet pointers that can be stored in the link depends on the link capacity. Now the input link is run wherein the packets are sent through the link. Once this is done the packet pointers are obtained from the input link and stored in the regulator, if the scheduling simulator is implementing a non-work-conserving scheduler or, in the scheduler, otherwise.

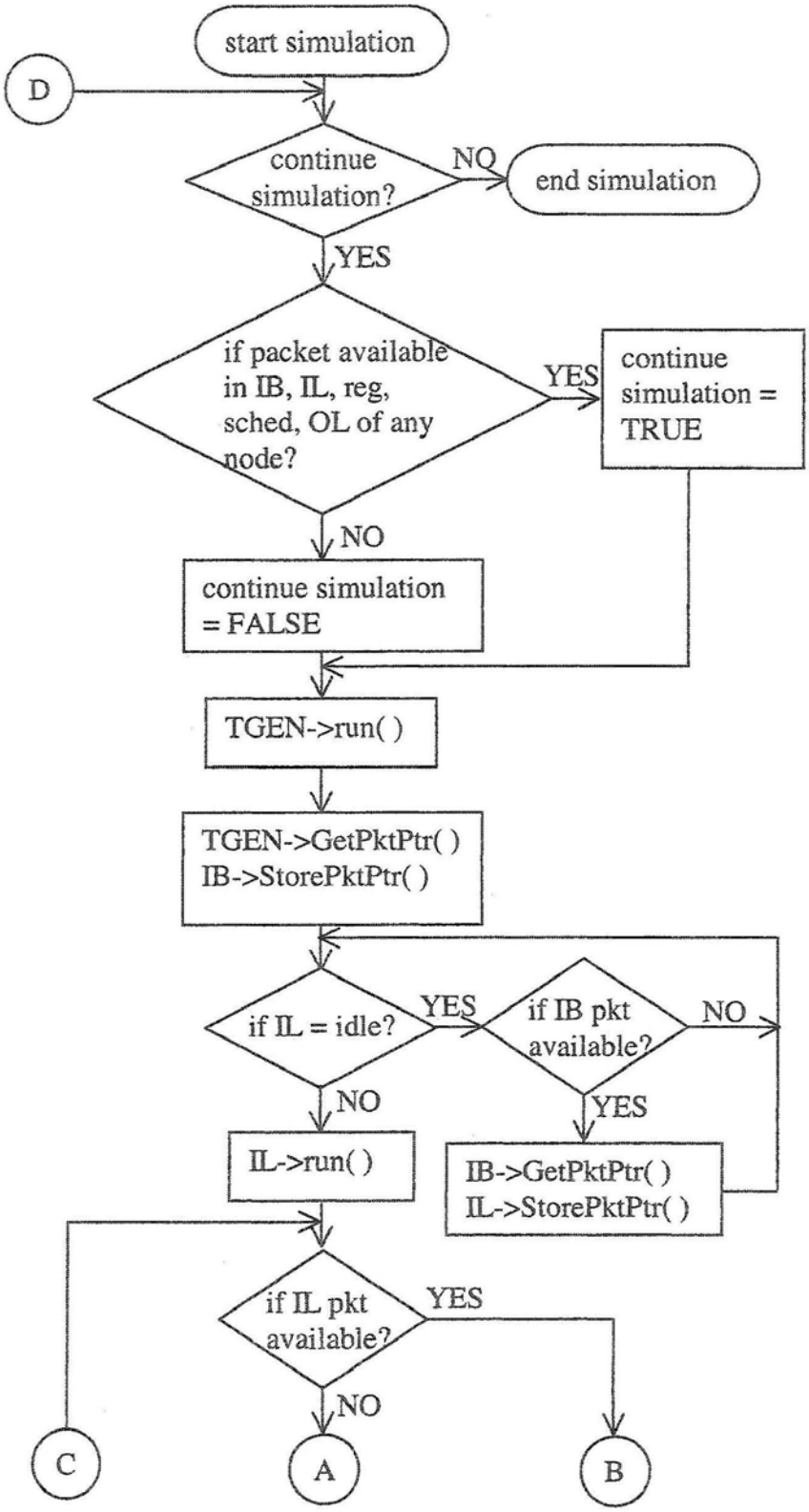



**Figure 4.6:** Flowchart of Control Unit

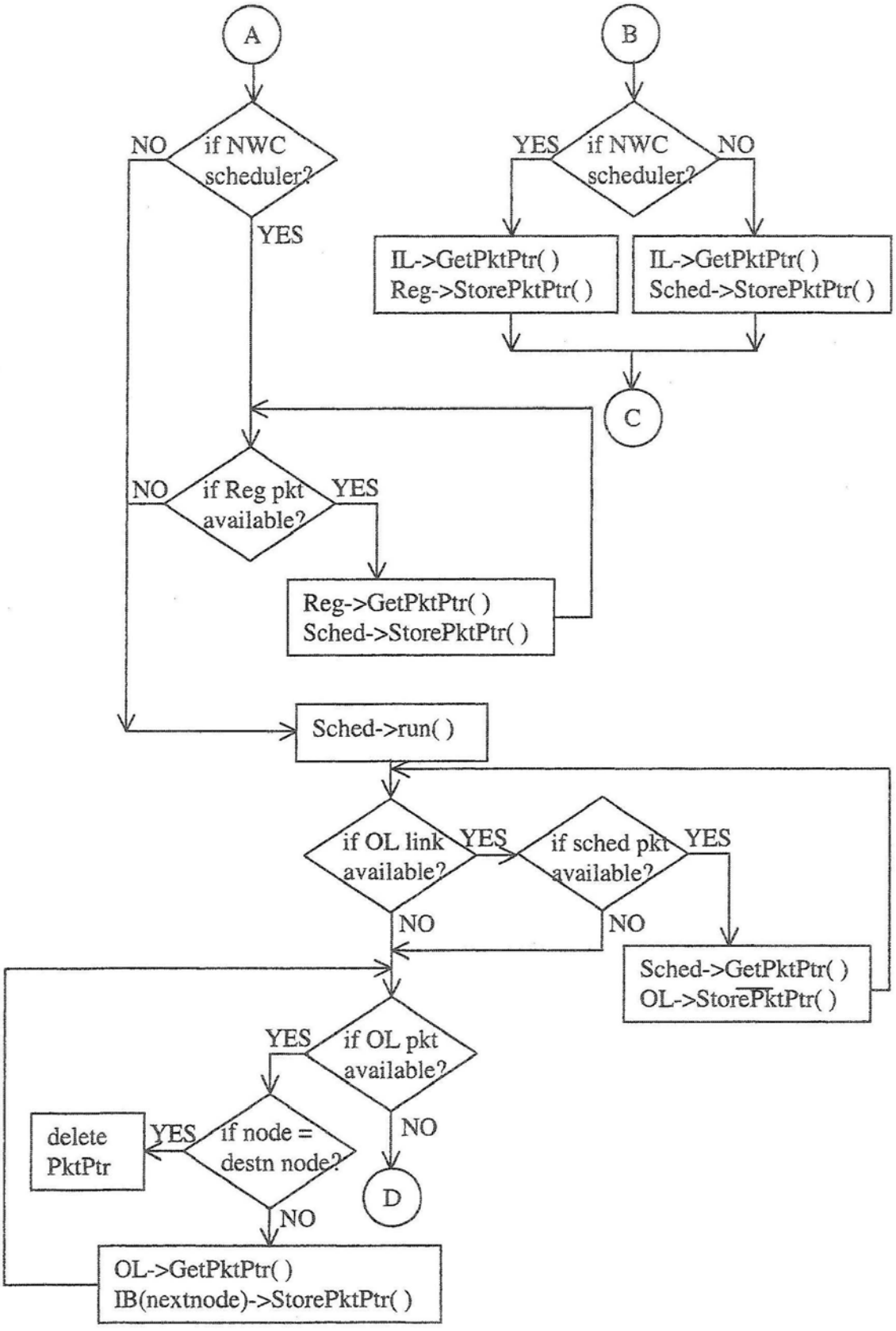



**Figure 4.6:** Flowchart of Control Unit (contd.)

Once the packet pointers are stored in the regulator, the eligibility time is calculated and when the packet pointer is available in the regulator, it is stored in the scheduler. Once in the scheduler, the finish number or finish time is calculated and the packets are sent out of the scheduler based on the least finish number or least finish time. These packet pointers are then obtained from the scheduler and sent to the output link, where they are again dispatched based on the output link rate into the input buffer of the next node or if it is the last node, it is destroyed and the packet details are collected by the data handler and stored in file. Every time a packet is destroyed, its event timings are recorded in a file, *pif.dat*, for future analysis. This file is used in the hardware implementation to obtain the arrival times of packets, thus facilitating comparison of hardware and software results.

## 4.6 Simulation Results

In this section, the delay characteristics of the real-time connection under the WFQ and $WF^2Q+$ disciplines are studied. The simulation is run for several cases and from the results obtained conclusions on the behaviour of the two disciplines can be arrived at. Each simulation run lasts for 10,000 cycle. The traffic arrival pattern chosen resembles that used in [21], but only fixed-sized packets case is considered in this paper. This thesis considers both fixed and variable-sized packets. The simulation is first run for the WFQ algorithm and then the traffic arrival pattern is captured. This arrival pattern is then used for the $WF^2Q+$ algorithm. These measures ensure fair comparison of algorithms. The simulation is conducted with and without cross traffic from constant source and with the Poisson sources exceeding their guaranteed rate (violating their traffic

constraint) by 50%. That is, the Poisson sources are sending at a rate of 1.5 times their guaranteed rate. The real-time source is a deterministic bursty traffic. The various cases considered are tabulated as shown in Table 4.1

| Packet size | Cross traffic | Best-effort source | Link rate |
|---|---|---|---|
| Fixed-sized packets of 50 bytes | Without cross-traffic from constant source | Least best-effort source (average packet inter-arrival time = 1000 cycles) | Link rates at the output of nodes N1, N2 and N3 = 50, 100, 150 bytes/cycle |
| | | Maximum best-effort source (average packet inter-arrival time = 3 cycles) | |
| | With cross-traffic from constant source | Least best-effort source (average packet inter-arrival time = 1000 cycles) | |
| | | Maximum best-effort source (average packet inter-arrival time = 3 cycles) | |
| Variable-sized packets | Without cross-traffic from constant source | Least best-effort source (average packet inter-arrival time = 1000 cycles) | Link rates at the output of nodes N1, N2 and N3 = 44, 320, 1500 bytes/cycle |
| | | Maximum best-effort source (average packet inter-arrival time = 3 cycles) | |
| | With cross-traffic from constant source | Least best-effort source (average packet inter-arrival time = 1000 cycles) | |
| | | Maximum best-effort source (average packet inter-arrival time = 3 cycles) | |

**Table 4.1:** Various cases considered for software simulation and hardware implementation

The three link rates chosen for fixed-sized packets indicate the following: the link rate of 50 bytes/cycle corresponds to sending only one packet per cycle through the output link (since each packet is of size 50 bytes), the link rates of 100 bytes and 150 bytes correspond to sending two and three packets per cycle respectively through the output link to test for lower load values. When the simulation was run for a link rate of 200 bytes/cycle, all the packets under the $WF^2Q+$ discipline experienced zero delay. Thus link rates of 200 bytes/cycle and beyond are not considered here. The three link rates chosen for variable-sized packets denote the following. The link rate of 44 bytes/cycle implies that the link is capable of allowing only a 44-byte packet (smallest sized packet) to be sent through the output link in one cycle. Any packet, which has a size larger than 44 bytes, will take more than one cycle (1 ms) to leave through the output link. The link rate of 1500 bytes/cycle implies that the link is capable of allowing a packet of a 1500 byte packet (maximum packet size) in one cycle. The intermediate link rate of 320 bytes/cycle is obtained from the packet length pdf. According to this distribution, on average 319 bytes of packets arrive in one cycle or 1 ms and therefore, the link rate is rounded to 320 bytes/cycle.

Firstly, considering fixed-sized packets, taking each packet size to be 50 bytes, the plot of end-to-end delay for real-time source for the case of least best-effort traffic and an output link rate of 50 bytes/cycle, without any cross traffic from constant source is shown in Figure 4.7a for the WFQ discipline. The plot of delay versus time for the $WF^2Q+$ discipline is shown in Figure 4.7b. The best-effort traffic has an average inter-arrival time of 1000 ms. In other words, 20% of the packets have an inter-arrival time of 600ms, 20% of the packets have an inter-arrival time of 800 ms, 20% of the packets have an inter-

arrival time of 1000 ms, 20% of the packets have an inter-arrival time of 1200 ms and the remaining 20% of the packet have an inter-arrival time of 1400 ms. The total load at the entrance of N3 is 85.5%. The simulation results for these two cases showing the total number of packets from each connection, the minimum, maximum and mean delay experienced by the connections in each node and the standard deviation for the connections in each node are given in Appendix A. Standard deviation is calculated to

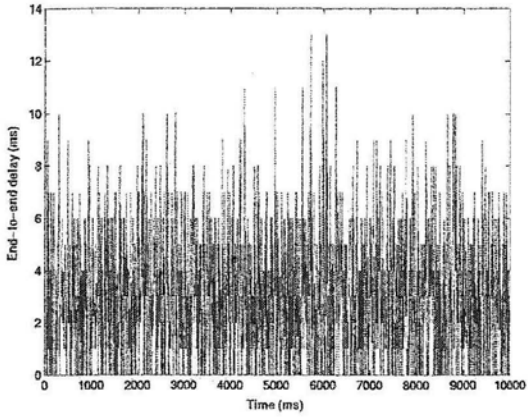



**Figure 4.7a:** End-to-end delay of WFQ scheduler for fixed-sized packets without cross-traffic with least best-effort traffic and an output link rate of 50 bytes/ms.

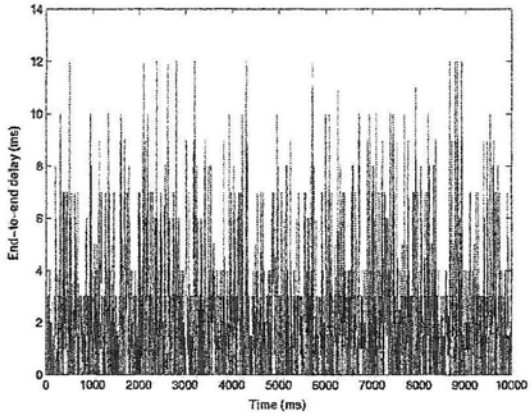



**Figure 4.7b:** End-to-end delay of $WF^2Q+$ scheduler for fixed-sized packets without cross-traffic with least best-effort traffic and an output link rate of 50 bytes/ms.

measure how much the individual delay values deviate from the average delay. From the above two plots, the worst-case delay experienced by the packets from the real-time connection in both the cases of WFQ and $WF^2Q$ + are almost the same, 13 ms and 12 ms respectively. The minimum delay is 0 ms for both the cases of WFQ and $WF^2Q+$. The average delay of the $WF^2Q+$ discipline (ms) is less than that of the WFQ discipline (ms). This accounts for the fact that $WF^2Q+$ follows the GPS service order more closely than the WFQ discipline. The standard deviations of the two disciplines do not show much difference. Similarly, plots of end-to-end delay of WFQ and $WF^2Q+$ when the output link

is 100 bytes/ms and 150 bytes/ms are shown in Figures 4.8a, 4.8b, 4.9a and 4.9b, respectively.

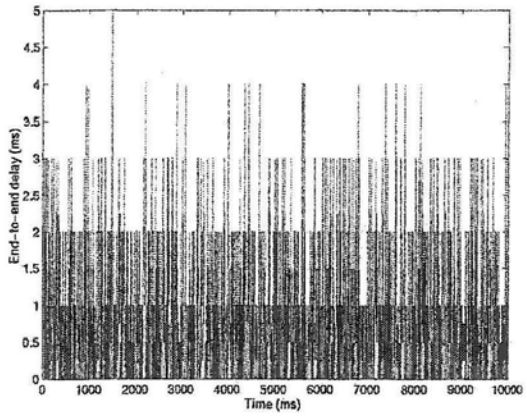



**Figure 4.8a:** End-to-end delay of WFQ scheduler for fixed-sized packets without cross-traffic with least best-effort traffic and an output link rate of 100 bytes/ms.

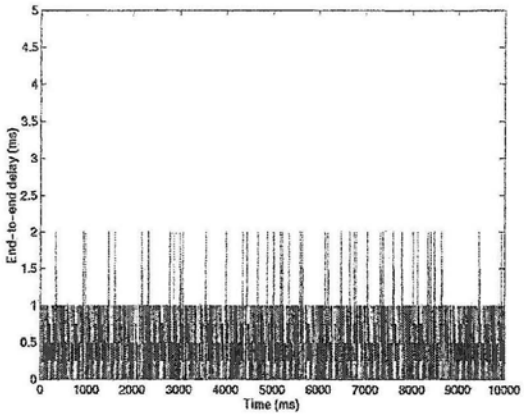



**Figure 4.8b:** End-to-end delay of $WF^2Q+$ scheduler for fixed-sized packets without cross-traffic with least best-effort traffic and an output link rate of 100 bytes/ms.

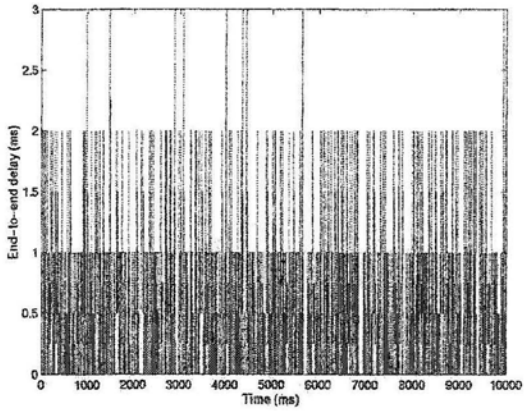



**Figure 4.9a:** End-to-end delay of WFQ scheduler for fixed-sized packets without cross-traffic with least best-effort traffic and an output link rate of 150 bytes/ms.

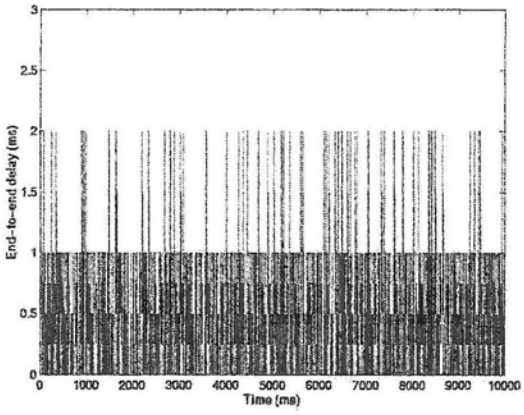



**Figure 4.9b:** End-to-end delay of $WF^2Q+$ scheduler for fixed-sized packets without cross-traffic with least best-effort traffic and an output link rate of 150 bytes/ms.

When the link rate is 100 bytes/ms, there is a clear difference between the delays experienced by packets under the WFQ scheme and that under the $WF^2Q+$ scheme (see plots on Figures 4.8a and 4.8b). In this case, the load at N3 is close to 43%. The

maximum delay experienced by a packet under the WFQ scheme is 5 ms while that under the $WF^2Q+$ scheme is only 2 bytes/ms. The average delay is also comparatively less in the $WF^2Q+$ scheme. The standard deviations of the two disciplines differ greatly in this case. The WFQ scheme shows much higher standard deviation than the $WF^2Q+$ scheme. In other words, the delays of packets under the WFQ scheme oscillate between the minimum and maximum value most of the times rather than remaining close to the average delay. A similar explanation holds when the link rate is 150 bytes/ms. The load at N3 in this case is around 28%. It is useful to compare the delay performance of the two disciplines by subjecting them to various traffic loads so as to ensure that the results are valid even when the network is heavily loaded. From this discussion, it can be said that $WF^2Q+$ performs better than WFQ for all the three cases of output link rates and therefore, traffic loads.

The next case is without constant source and with maximum best-effort traffic. The best-effort traffic has an average inter-packet arrival time of 3 ms in this case.

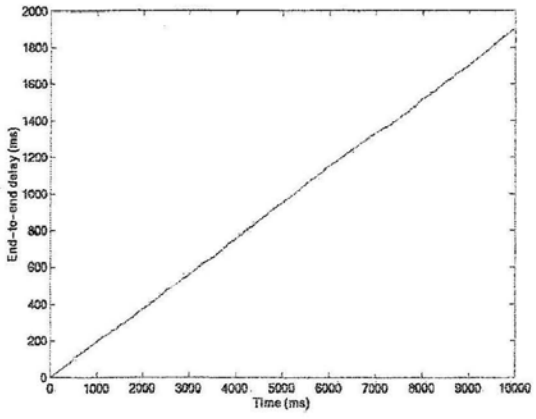



**Figure 4.10a:** End-to-end delay of WFQ scheduler for fixed-sized packets without cross-traffic with maximum best-effort traffic and an output link rate of 50 bytes/ms.

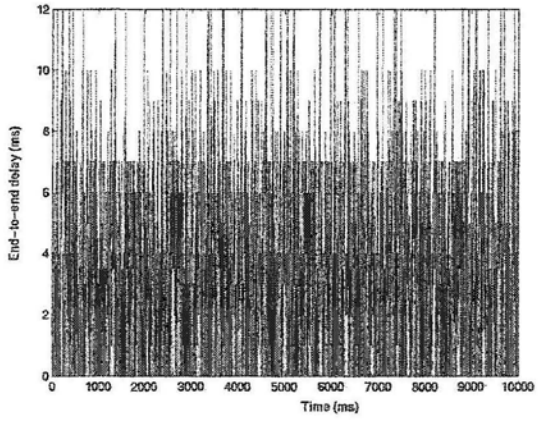



**Figure 4.10b:** End-to-end delay of $WF^2Q+$ scheduler for fixed-sized packets without cross-traffic with maximum best-effort traffic and an output link rate of 50 bytes/ms.

The results for this case can be seen from the Figures 4.10a, 4.10b, 4.11a, 4.11b, 4.12a and 4.12b for output link rates of 50, 100 and 150 bytes/ms respectively. When the output link rate is 50 bytes/ms, the load at N3 is above 118%. For such a high load, the real-time packets under the WFQ discipline experience a very large delay. Heavy traffic load leads to the scheduler queue build up leading to instability.

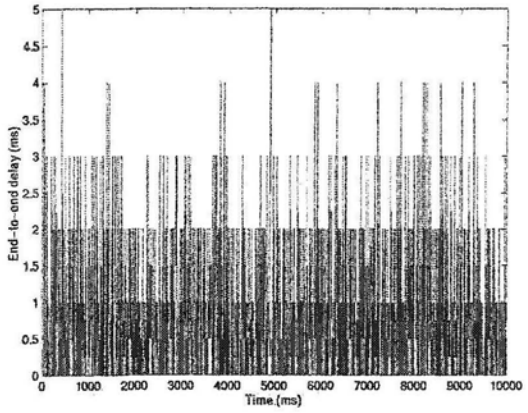



**Figure 4.11a:** End-to-end delay of WFQ scheduler for fixed-sized packets without cross-traffic with maximum best-effort traffic and an output link rate of 100 bytes/ms.

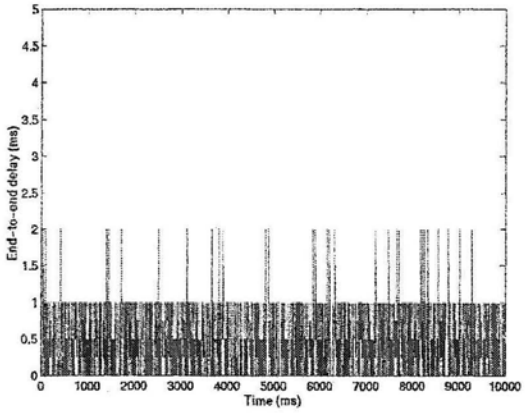



**Figure 4.11b:** End-to-end delay of $WF^2Q+$ scheduler for fixed-sized packets without cross-traffic with maximum best-effort traffic and an output link rate of 100 bytes/ms.

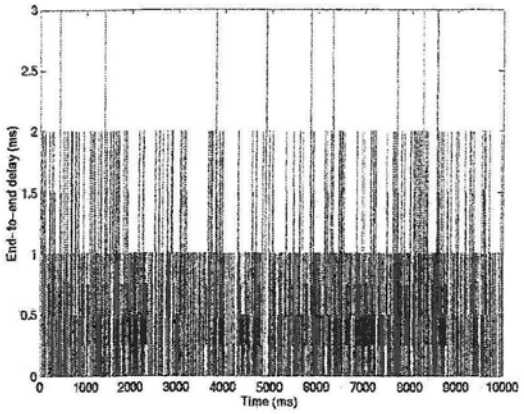



**Figure 4.12a:** End-to-end delay of WFQ scheduler for fixed-sized packets without cross-traffic with maximum best-effort traffic and an output link rate of 150 bytes/ms.

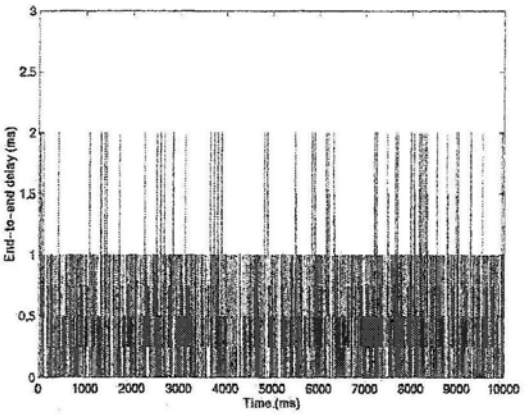



**Figure 4.12b:** End-to-end delay of $WF^2Q+$ scheduler for fixed-sized packets without cross-traffic with maximum best-effort traffic and an output link rate of 150 bytes/ms.

The reason for the load to exceed 100% at N3 is because of the cross-traffic from Poisson source at N2 and N3 which exceed their guaranteed rate by 50%. Moreover, since four input links are allowed to enter the switch, the load coming from each link would be so high that the load at the output link exceeds 100%.

The Figure 4.10a is completely different from the plots that were seen until this case. The difference in y-axis scale of the plot for the WFQ discipline (Figure 4.10a) and $WF^2Q+$ discipline (Figure 4.10b) should be noted. The average delay experienced by packets from the real-time connection at N3 is 947 ms for the WFQ discipline. This is unsuitable for any real-time application. Usually the network is not so heavily loaded as this leads to overflowing queues. Nevertheless, when compared with the average delay of packets under the $WF^2Q+$ discipline (which is 2.45ms), the delay of WFQ discipline is much higher. In the case of $WF^2Q+$ discipline, though the overall arrival rate at N3 is 118%, the arrival rate N1 from the real-time source is less than 100% and the load from the best-effort traffic at N1 does not affect the delay of packets from the real-time connection. However, in the case of WFQ discipline, the maximum load arriving from best-effort traffic affects (increases) the delay of packets from real-time source at N1. At N2, due to the traffic from Poisson sources, this delay further increases thus leading to queue build up. When these two figures (Figures 4.10a and 4.10b) are compared with the corresponding ones from the least best-effort case (Figures 4.7a and 4.7b), it can be seen that the $WF^2Q+$ discipline tries to retain the same delay for the packets of the real-time connections even when there is cross-traffic from the best-effort connection at N1, whereas the WFQ discipline is affected by the cross-traffic from best-effort connection at

N1. The results for output link rates of 100 bytes/ms and 150 bytes/ms are quite similar to the previous case of least best-effort traffic.

The next case considered is the end-to-end delay of real-time source with the presence of cross-traffic from constant source, with least best-effort traffic and various output link rates of 50, 100 and 150 bytes/ms. The packets from constant source are spaced 135 ms apart. The results are plotted in Figures 4.13a, 4.13b, 4.14a, 4.14b, 4.15a and 4.15b. When the output link rate is 50 bytes/ms, the load at N3 is close to 93%.

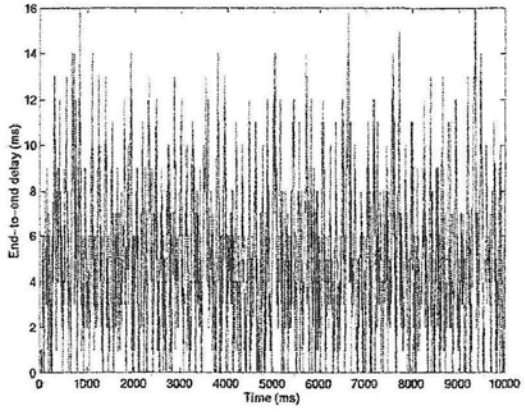



**Figure 4.13a:** End-to-end delay of WFQ scheduler for fixed-sized packets with cross-traffic with least best-effort traffic and an output link rate of 50 bytes/ms.

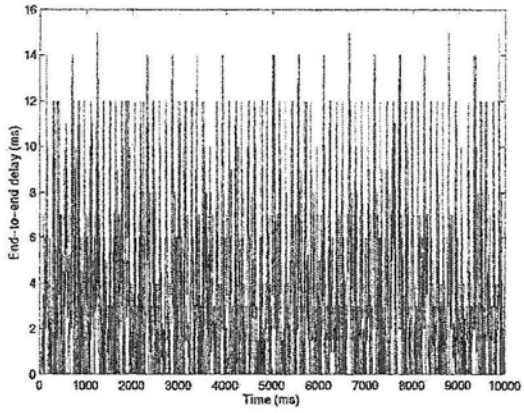



**Figure 4.13b:** End-to-end delay of $WF^2Q+$ scheduler for fixed-sized packets with cross-traffic with least best-effort traffic and an output link rate of 50 bytes/ms.

Again, the average delay of the $WF^2Q+$ discipline is less than that of the WFQ discipline (refer to Figures 4.13a and 4.13b). The standard deviation is close to the average delay in this case showing that the delays of most of the packets are distributed close to the average delay. Also, the maximum delays are almost the same for both the cases. The maximum, minimum, average delays and the standard deviation values are presented in Appendix B. When the output link rate is 100 bytes/ms, the load at N3 is 46%. Here

again, the average delay, maximum delay and the standard deviation of packets from the real-time connection are less for the $WF^2Q+$ discipline than the WFQ discipline. When the output link rate is 150 bytes/ms, the load at N3 is 30%. For the cases when the output link rate is 100 bytes/ms and 150 bytes/ms, the delays experienced by packets in the WFQ discipline vary between the maximum and the minimum values more often.

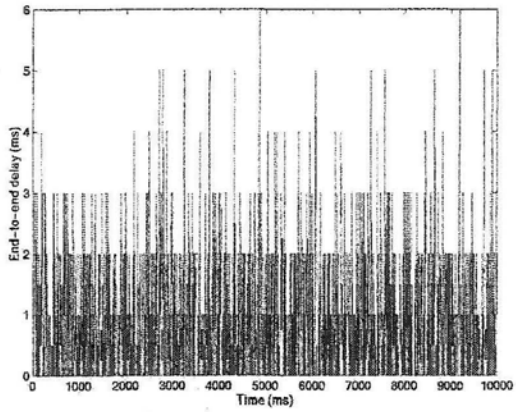



**Figure 4.14a:** End-to-end delay of WFQ scheduler for fixed-sized packets with cross-traffic with least best-effort traffic and an output link rate of 100 bytes/ms.

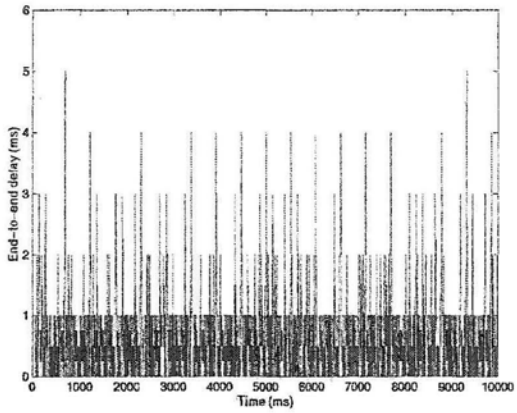



**Figure 4.14b:** End-to-end delay of $WF^2Q+$ scheduler for fixed-sized packets with cross-traffic with least best-effort traffic and an output link rate of 100 bytes/ms.

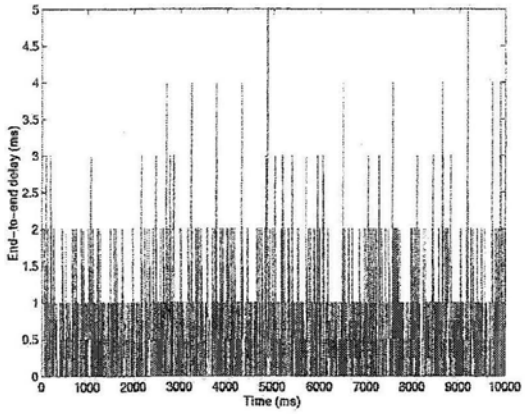



**Figure 4.15a:** End-to-end delay of WFQ scheduler for fixed-sized packets with cross-traffic with least best-effort traffic and an output link rate of 150 bytes/ms.

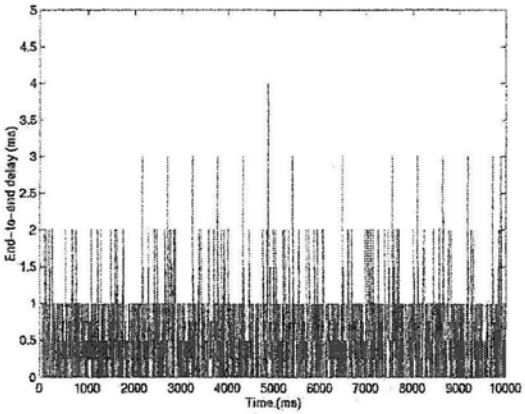



**Figure 4.15b:** End-to-end delay of $WF^2Q+$ scheduler for fixed-sized packets with cross-traffic with least best-effort traffic and an output link rate of 150 bytes/ms.

That is, the oscillations are high which means the delay-jitter is high. The same for the $WF^2Q+$ discipline are concentrated close to the average delay. Thus, from the Figures 4.13a, 4.13b, 4.14a, 4.14b, 4.15a and 4.15b, and also from the standard deviation values, it can be observed that $WF^2Q+$ discipline has a better delay performance in terms of the average delay, maximum delay and the delay jitter.

The last traffic pattern considered in the fixed-sized packets case is the case of maximum best-effort traffic with the presence of constant source. Here the load for a link

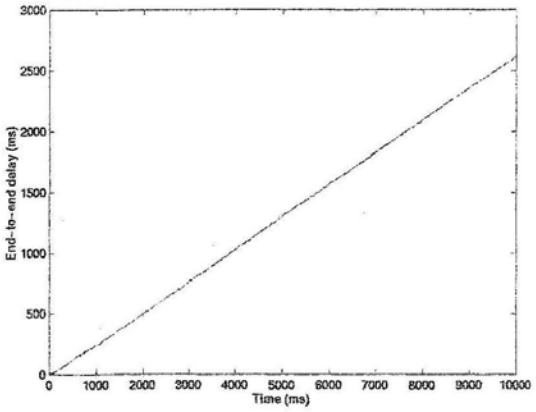



**Figure 4.16a:** End-to-end delay of WFQ scheduler for fixed-sized packets with cross-traffic with maximum best-effort traffic and an output link rate of 50 bytes/ms.

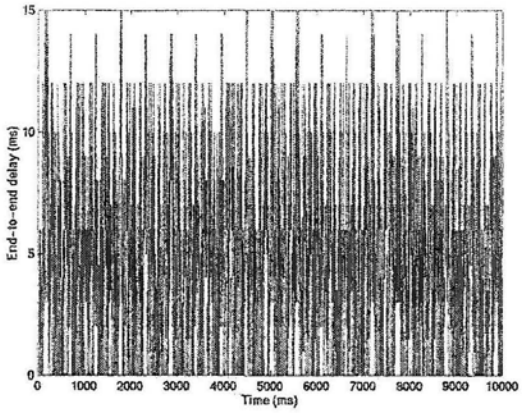



**Figure 4.16b:** End-to-end delay of $WF^2Q+$ scheduler for fixed-sized packets with cross-traffic with maximum best-effort traffic and an output link rate of 50 bytes/ms.

rate of 50 bytes/ms is higher than the previous case without cross-traffic and with maximum best-effort traffic for the same link rate. In this case, the load at N3 is 125%. As before, as the queue keeps building up the packets in the WFQ discipline experience more delay. The delays of the first few packets cause the rest of the packets to be delayed further and the queue builds up. As can be seen from Figures 4.16a and 4.16b, the maximum delay for a real-time connection under the WFQ scheme is above 2600 ms,

which is unsuitable for high-speed networks, whereas that under the WF$^2$Q+ scheme is still close to that of the least best-effort traffic case (seen in Figure 4.13b).

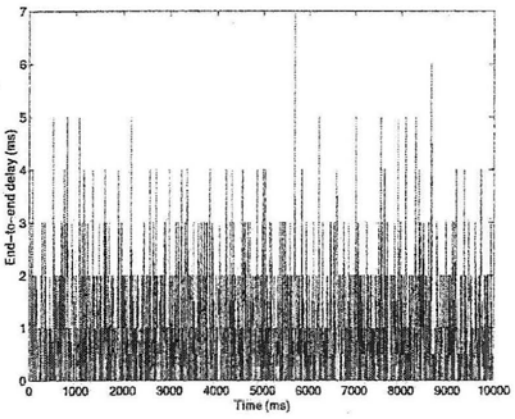



**Figure 4.17a:** End-to-end delay of WFQ scheduler for fixed-sized packets with cross-traffic with maximum best-effort traffic and an output link rate of 100 bytes/ms.

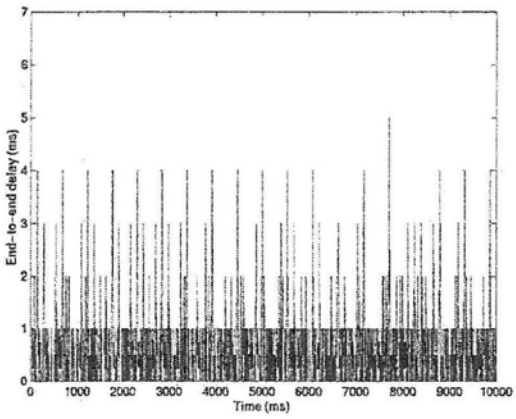



**Figure 4.17b:** End-to-end delay of WF$^2$Q+ scheduler for fixed-sized packets with cross-traffic with maximum best-effort traffic and an output link rate of 100 bytes/ms.

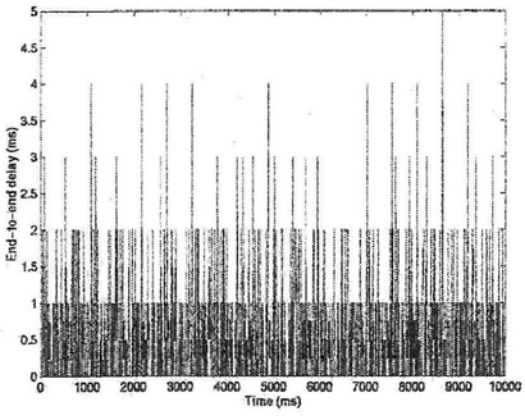



**Figure 4.18a:** End-to-end delay of WFQ scheduler for fixed-sized packets with cross-traffic with maximum best-effort traffic and an output link rate of 150 bytes/ms.

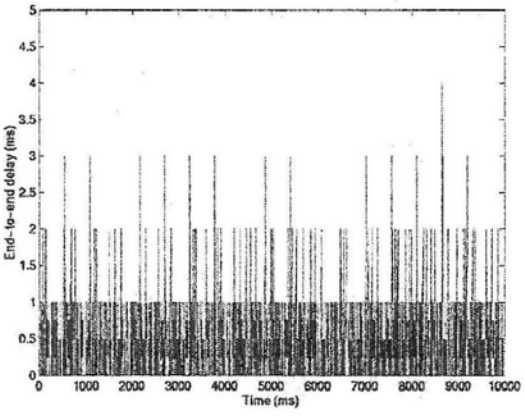



**Figure 4.18b:** End-to-end delay of WF$^2$Q+ scheduler for fixed-sized packets with cross-traffic with maximum best-effort traffic and an output link rate of 150 bytes/ms.

The results for the cases when the link rates are 100 bytes/ms and 150 bytes/ms are the same as the previous cases with the packets under the WF$^2$Q+ discipline having a lower

delay compared to those under the WFQ discipline (Refer to Figures 4.17a, 4.17b, 4.18a, 4.18b).

Thus it can be perceived that WF$^2$Q+ scheme remains unaffected by the presence of cross-traffic either from best-effort source or from constant source. The results obtained until now are consistent with those obtained by Bennett and Zhang in [21].

For certain applications that involve feedback-based algorithms that are used in data communication networks, oscillations as seen under the WFQ scheme for fixed-sized packets are undesirable. In this case, a data source has to balance between two considerations: on the one hand, it wants to send data to the network as fast as possible; on the other hand, it does not want to send data so fast that causes network congestion. The oscillations in WFQ make it unsuitable for such applications. Also, it is inferred that the scheduling algorithm should not send packets too fast that it causes network congestion in the next node [10]. For these reasons, WF$^2$Q+ discipline which has a smaller delay-jitter is preferred over WFQ discipline for fixed-sized packets.

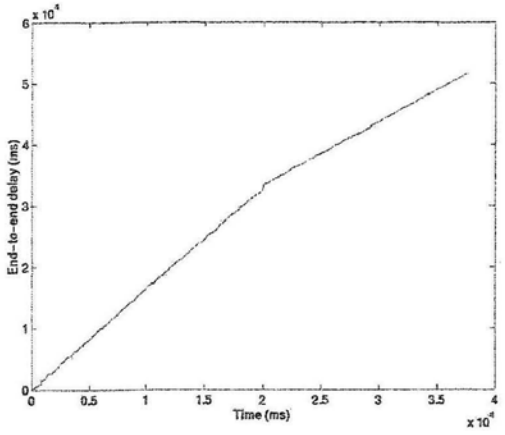



**Figure 4.19a:** End-to-end delay of WFQ scheduler for variable-sized packets without cross-traffic with least best-effort traffic and an output link rate of 44 bytes/ms.

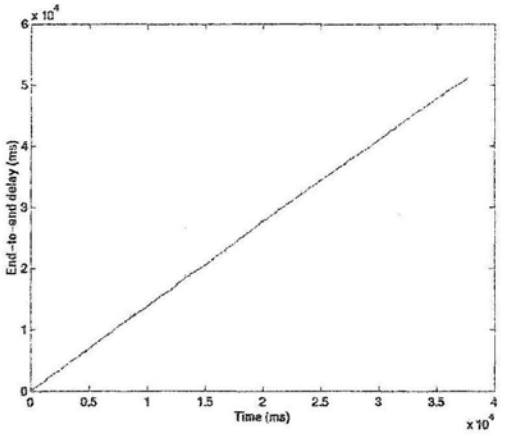



**Figure 4.19b:** End-to-end delay of WF$^2$Q+ scheduler for variable-sized packets without cross-traffic with least best-effort traffic and an output link rate of 44 bytes/ms.

The same algorithms are tested for variable-sized packets. The results obtained did not match that obtained for fixed-sized packets. In their paper, Bennett and Zhang, did not consider the case of variable-sized packets. Considering the case of variable-sized packets with least best-effort traffic and without constant source, Figures 4.19a and 4.19b show the delay distribution for both the disciplines when the link rate is 44 bytes/ms.

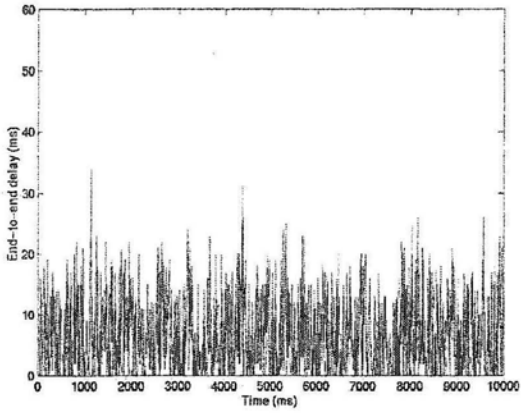



**Figure 4.20a:** End-to-end delay of WFQ scheduler for variable-sized packets without cross-traffic with least best-effort traffic and an output link rate of 320 bytes/ms.

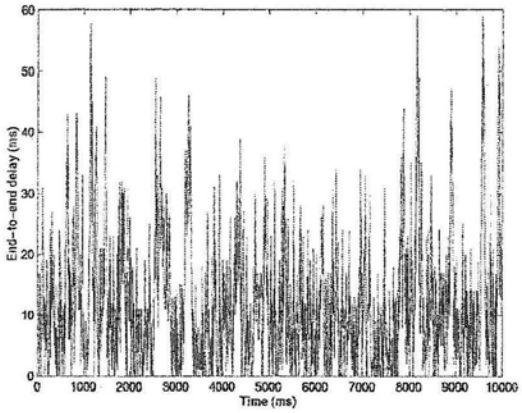



**Figure 4.20b:** End-to-end delay of $WF^2Q+$ scheduler for variable-sized packets without cross-traffic with least best-effort traffic and an output link rate of 320 bytes/ms.

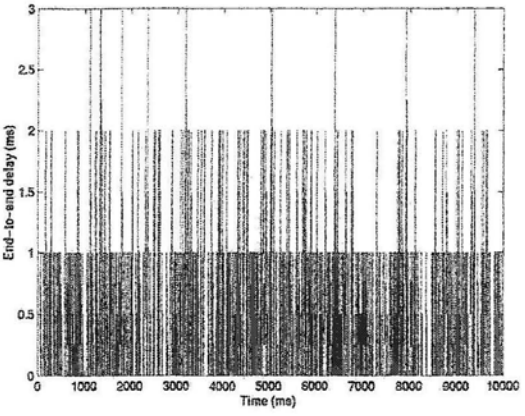



**Figure 4.21a:** End-to-end delay of WFQ scheduler for variable-sized packets without cross-traffic with least best-effort traffic and an output link rate of 1500 bytes/ms.

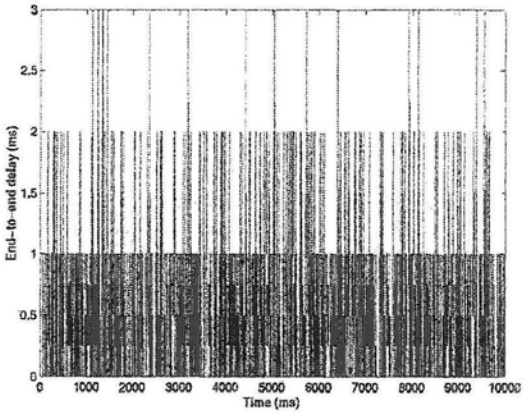



**Figure 4.21b:** End-to-end delay of $WF^2Q+$ scheduler for variable-sized packets without cross-traffic with least best-effort traffic and an output link rate of 1500 bytes/ms.

In this case, the load at N3 is 638%, which is practically not possible. The scale along the x-axis clearly shows that it takes around 40,000 ms to serve all the packets in the queue. Moreover, the delay is as high as 14,000 ms for a real-time source (see y-axis scale). Therefore, this case is ignored (not considered for comparison). Though the delay is high for this case, the reason for considering this case (and similar such cases where the load is extremely high and the queue is overflowing) is to compare this case (and similar such cases) with the case of fixed-sized packets with cross-traffic and with maximum best-effort traffic and an output link rate of 50 bytes/ms (Figures 4.16a and 4.16b) where the queue does not overflow for $WF^2Q+$ as it overflows in the case of WFQ eventhough the load exceeds 100%, in both the cases.

The next case is a link rate of 320 bytes/ms and the same arrival pattern as before. In this case, the load at N3 comes to around 82%. The plots of delays are shown in Figures 4.20a and 4.20b. Contrary to the previous results obtained for the fixed-sized packets, in this case, the average delay of the $WF^2Q+$ discipline is higher than that of the WFQ discipline. This is because $WF^2Q+$ tries to approximate GPS as closely as possible. In other words, WFQ is far ahead of GPS in the number of bits served during any interval of time. The maximum, minimum, average delays and the standard deviation values are presented in Appendix C. It can also be seen that the delay-jitter is high for $WF^2Q+$ discipline owing to the oscillations of the delay around the average delay. Although WFQ has higher delay and delay-jitter for fixed-sized packets, the delay and delay-jitter are lower for variable-sized packets. Though the delay and delay-jitter are higher for the $WF^2Q+$ discipline, the end-to-end delay is bound within the tolerable limits. When the output link is further increased to 1500 bytes/ms, the load is around 18%. For such a light

load, the delays under both the disciplines are almost identical as can be seen in Figures 4.21a and 4.21b.

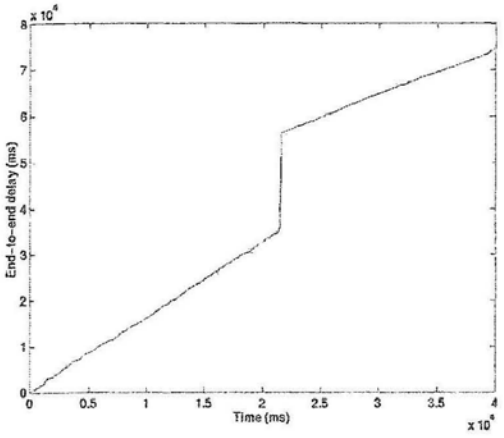



**Figure 4.22a:** End-to-end delay of WFQ scheduler for variable-sized packets without cross-traffic with maximum best-effort traffic and an output link rate of 44 bytes/ms.

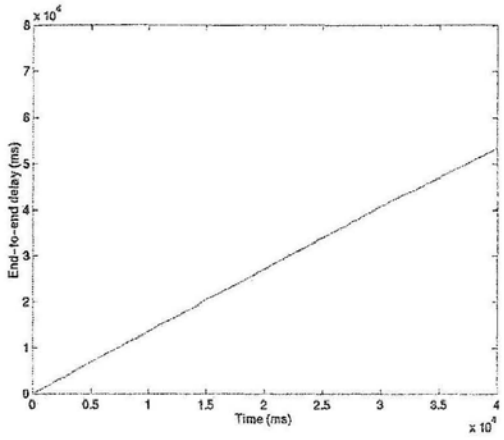



**Figure 4.22b:** End-to-end delay of $WF^2Q+$ scheduler for variable-sized packets without cross-traffic with maximum best-effort traffic and an output link rate of 44 bytes/ms.

For the case of maximum best-effort traffic and without cross-traffic from constant source the load at N3 is 864%. The plots are shown in Figures 4.22a and 4.22b. This is again not fit for comparison. The queues build up and the delays of the newly arriving packets increase because the previously arrived packets have not been transmitted yet. The reason for such a high traffic load is mainly because four input links are allowed to enter the switch. Therefore, it is possible that all four links have greater than 100% load or in other words, have sources that exceed their agreed traffic profile. In such a case, the traffic arrival load is beyond 400%. Moreover, the ouput link rate is so less (44 bytes/ms) that if a packet size is larger than 44 bytes, the packet will be sent in more than one cycle through the output link. This is the cause for such a high load of 864%. When the link rate is 320 bytes/ms, the load at N3 is 119%. The average delay of packets under the WFQ discipline is lesser than that under the $WF^2Q+$ discipline as can

be seen in Figures 4.23a and 4.23b. In this case, the delay of packets in the best-effort connection is higher under the WFQ discipline than under the $WF^2Q+$ scheme (Appendix C).

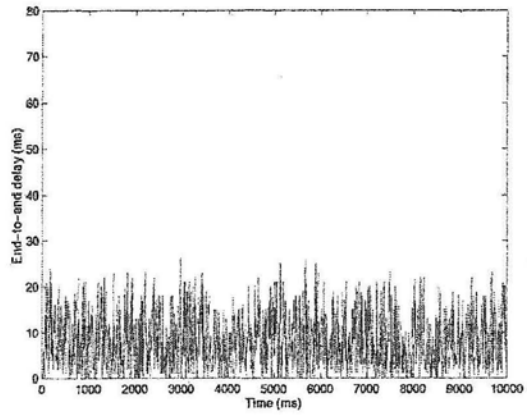



**Figure 4.23a:** End-to-end delay of WFQ scheduler for variable-sized packets without cross-traffic with maximum best-effort traffic and an output link rate of 320 bytes/ms.

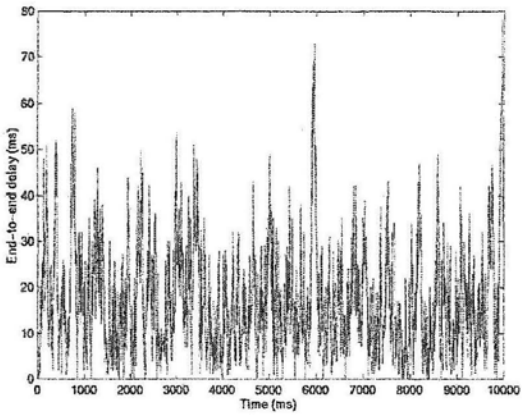



**Figure 4.23b:** End-to-end delay of $WF^2Q+$ scheduler for variable-sized packets without cross-traffic with maximum best-effort traffic and an output link rate of 320 bytes/ms.

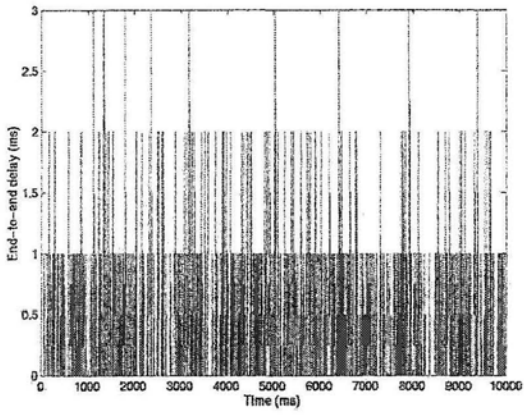



**Figure 4.24a:** End-to-end delay of WFQ scheduler for variable-sized packets without cross-traffic with maximum best-effort traffic and an output link rate of 1500 bytes/ms.

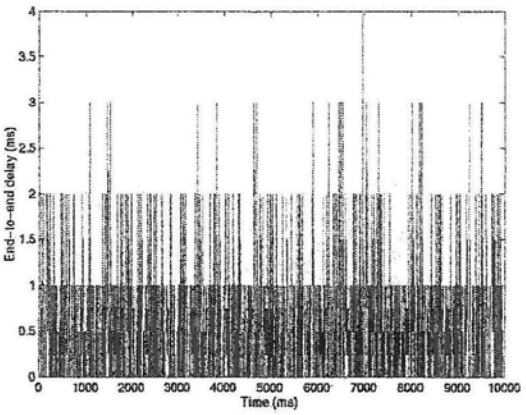



**Figure 4.24b:** End-to-end delay of $WF^2Q+$ scheduler for variable-sized packets without cross-traffic with maximum best-effort traffic and an output link rate of 1500 bytes/ms.

For a link rate of 1500 bytes/ms, the load at N3 is 25%. The delay under both the cases of WFQ and $WF^2Q+$ disciplines are almost identical for such a light load as can be seen in Figures 4.24a and 4.24b.

For the case of variable-sized packets, with constant source, with minimum best-effort traffic and a link rate of 44 bytes/ms, the load at N3 is 664%. The packets from each constant source are spaced 135 ms apart as before. Again, for this heavy traffic load, it is not possible to compare the two disciplines as the queues build up and the delay gradually increases from one packet to the next. The delay plots are shown in Figures 4.25a and 4.25b. For the case, when the link rate is 320 bytes/ms, the load at N3 is 91%. The maximum, minimum, average delays and the standard deviation values are presented in Appendix D. In this case, as before real-time packets under the $WF^2Q+$ experience more delay than those under the WFQ discipline as can be observed from Figures 4.26a and 4.26b.

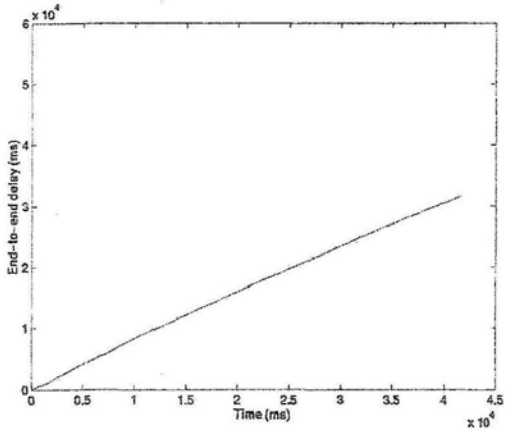



**Figure 4.25a:** End-to-end delay of WFQ scheduler for variable-sized packets with cross-traffic with least best-effort traffic and an output link rate of 44 bytes/ms.

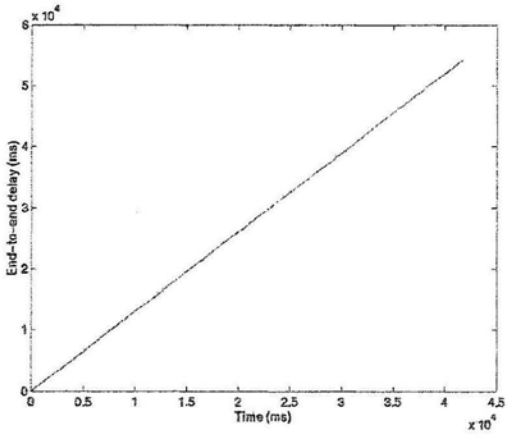



**Figure 4.25b:** End-to-end delay of $WF^2Q+$ scheduler for variable-sized packets with cross-traffic with least best-effort traffic and an output link rate of 44 bytes/ms.

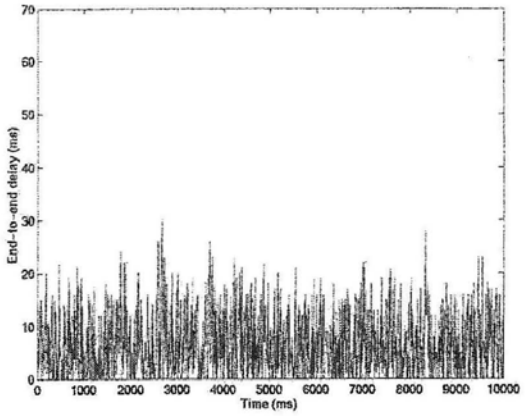



**Figure 4.26a:** End-to-end delay of WFQ scheduler for variable-sized packets with cross-traffic with least best-effort traffic and an output link rate of 320 bytes/ms.

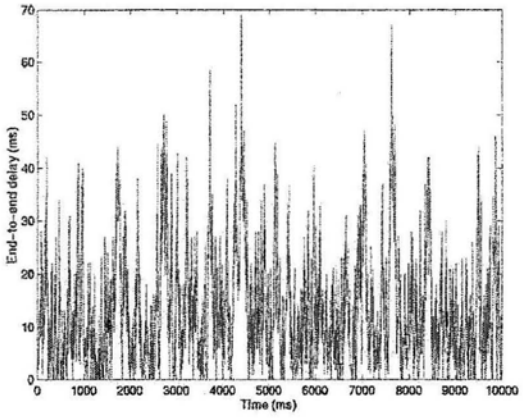



**Figure 4.26b:** End-to-end delay of $WF^2Q+$ scheduler for variable-sized packets with cross-traffic with least best-effort traffic and an output link rate of 320 bytes/ms.

When the link rate is 1500 bytes/ms, the load at N3 is around 19%. As before, this light load causes the delays under both the schemes to be identical for real-time source. This is illustrated in Figures 4.27a and 4.27b.

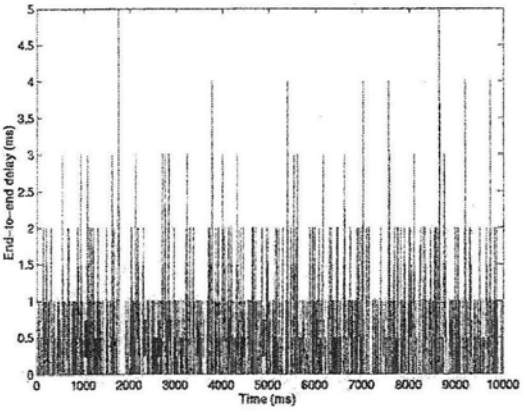



**Figure 4.27a:** End-to-end delay of WFQ scheduler for variable-sized packets with cross-traffic with least best-effort traffic and an output link rate of 1500 bytes/ms.

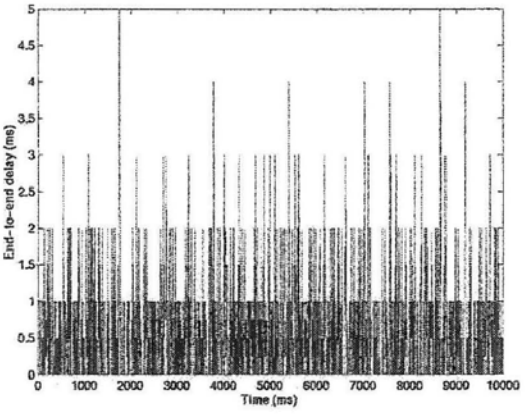



**Figure 4.27b:** End-to-end delay of $WF^2Q+$ scheduler for variable-sized packets with cross-traffic with least best-effort traffic and an output link rate of 1500 bytes/ms.

For the case when there is cross-traffic from constant source and with maximum best-effort traffic, the load at N3 is 927% when the link rate is 44 bytes/ms. This case is ignored for the reasons discussed before. The delay plots are shown in Figures 4.28a and 4.28b. When the link rate is 320 bytes/ms, the load at N3 is around 127%. Again WFQ shows lower delay than $WF^2Q+$ discipline as is shown in Figures 4.29a and 4.29b.

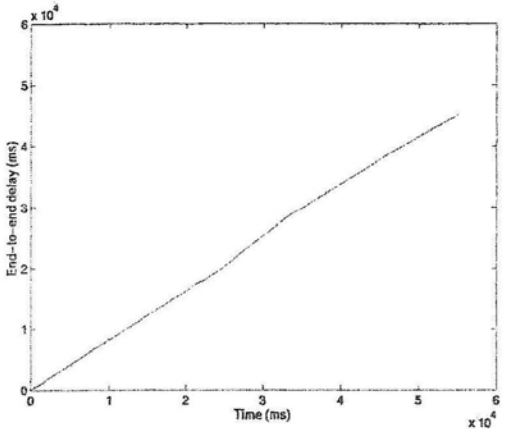



**Figure 4.28a:** End-to-end delay of WFQ scheduler for variable-sized packets with cross-traffic with maximum best-effort traffic and an output link rate of 44 bytes/ms.

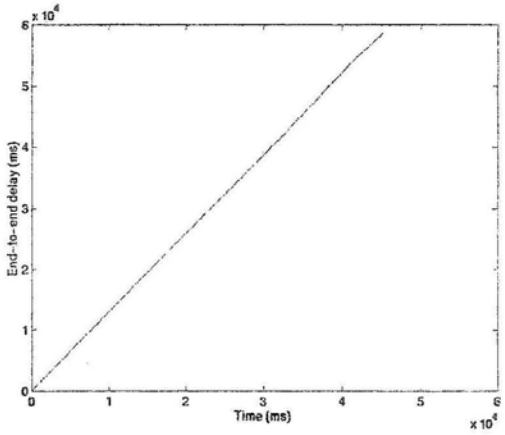



**Figure 4.28b:** End-to-end delay of $WF^2Q+$ scheduler for variable-sized packets with cross-traffic with maximum best-effort traffic and an output link rate of 44 bytes/ms.

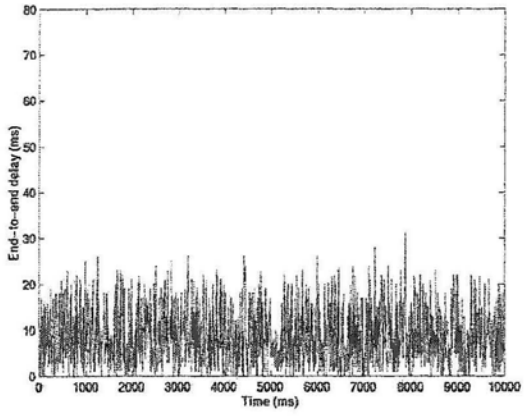



**Figure 4.29a:** End-to-end delay of WFQ scheduler for variable-sized packets with cross-traffic with maximum best-effort traffic and an output link rate of 320 bytes/ms.

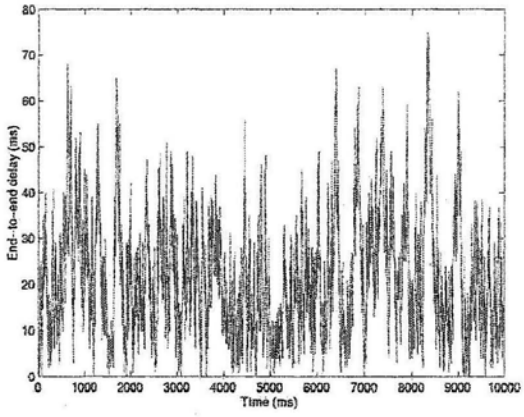



**Figure 4.29b:** End-to-end delay of $WF^2Q+$ scheduler for variable-sized packets with cross-traffic with maximum best-effort traffic and an output link rate of 320 bytes/ms.

From these two figures, it can also be observed that the delay-jitter is lower for WFQ discipline than the $WF^2Q+$ discipline. Again for the case when the link rate is 1500 bytes/ms, the load at N3 is around 27%. The delay plots are shown in Figures 4.30a and 4.30b. For this case, the delays are almost identical for the reasons discussed previously.

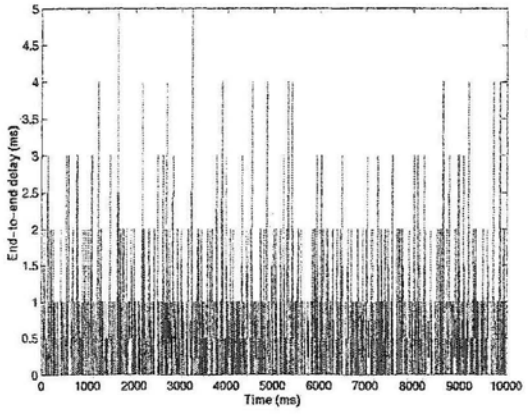



**Figure 4.30a:** End-to-end delay of WFQ scheduler for variable-sized packets with cross-traffic with maximum best-effort traffic and an output link rate of 1500 bytes/ms.

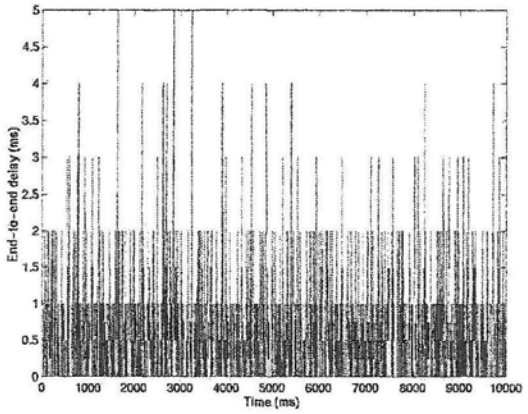



**Figure 4.30b:** End-to-end delay of $WF^2Q+$ scheduler for variable-sized packets with cross-traffic with maximum best-effort traffic and an output link rate of 1500 bytes/ms.

From the above discussion on variable-sized packets, it can be observed that the delays experienced by packets under the $WF^2Q+$ discipline are higher than those under the WFQ discipline. The oscillations are high in the case of $WF^2Q+$ discipline and therefore jitter is high, contrary to the results obtained for the fixed-sized packets case. Thus for the case of variable-sized packets, WFQ discipline could be a better choice for high-speed networks. For high-speed networks, the main requirement is lower complexity. High-speed networks can tolerate a slight increase in the average delay, but if the complexity of the algorithm is too high, it is not feasible to use the algorithm at high speeds. $WF^2Q+$ has reduced algorithmic complexity. So, it will be a good choice to go for

$WF^2Q+$ for high-speed networks even though it has a higher average delay and a higher delay-jitter for variable-sized packets. Although, $WF^2Q+$ is fairer than WFQ, the only justification for higher average delay and delay-jitter of $WF^2Q+$ is that it is maintaining the fairness property at any cost. It is possible to make WFQ implementation simpler by using the method used in $WF^2Q+$ to calculate the virtual time function. WFQ would be simpler but not fairer to all the connections because of its service order which is not fair to all the connections.

The results obtained for variable-sized packets do not mean that $WF^2Q+$ does not have a tight delay bound for variable-sized packets. It has a tight delay bound, but the delay bound is slightly higher than that of WFQ. That is, the average delay of $WF^2Q+$ for variable size packets is higher than WFQ, but the delay is bounded around the average delay. Also, it is possible that the average delay of $WF^2Q+$ is higher than WFQ, because this algorithm closely approximates the GPS system. That is, $WF^2Q+$ is no earlier than, nor, no later than GPS by one maximum packet size. In other terms, the lower delay obtained for WFQ is because it is far ahead of GPS in the number of bits served during any time interval. WFQ and GPS provide almost identical service except with a difference of one packet, according to Parekh [8]. Parekh meant that WFQ cannot fall behind GPS by more than one maximum size packet. However, WFQ can be far ahead of GPS in terms of the number of bits served and this might be one of the reasons for the higher delay in $WF^2Q+$ compared to WFQ. For $WF^2Q+$, when the plots for variable-sized packets are observed, the worst-case packet delay is large compared to that of WFQ. This is not the case for fixed-sized packets. The reason for this might be that $WF^2Q+$ is trying to be fair to all the other connections and so, in order that the other connections do not

have greater delay as in the case of WFQ, the packets from this connection are experiencing more delay. Moreover, only those packets, which have already started service in the corresponding GPS system, will be considered for scheduling and so, delay (or, worst-case delay) of $WF^2Q+$ is higher for variable-sized packets.

Applying the same arrival pattern shown in Figure 3.1a to the algorithms WFQ and $WF^2Q+$ under the case of variable-sized packets, assume the second packet arriving in connection 1 is of size 1500 bytes. Then, according to the WFQ algorithm, it might have a finish number which is greater than the finish number of the other connections. In such a case, one packet from connection 2 will be sent on the output link. The time taken to send this 1500 byte packet will affect the departure time of the first packet in the second connection. This causes an increase in the delay of the first packet in the second connection and thus makes the algorithm (WFQ) unfair in the service provided for all other connections except the real-time connection.

## 4.7 Discussion

The decision on which scheduling discipline to use depends on the specific application, whether it can tolerate the high algorithmic cost (in which case the choice would be WFQ), whether it can tolerate the higher delay or whether it consists of only fixed-sized packets (in which case the choice would be $WF^2Q+$). This chapter described the software implementation of a simulated system and discussed the results obtained in detail. Though the algorithmic complexity for $WF^2Q+$ is reduced, the cost of maintaining the queues in the regulator and scheduler is still $O(N)$ where $N$ is the number of

connections. The next chapter describes the hardware implementation of the two algorithms. In the hardware implementation, the cost of maintaining the regulator queues is reduced by the use of calendar queue implementation.

# Chapter 5

# 5. Hardware Implementation

## 5.1 Introduction

Chapter 4 discussed software implementation of the two scheduling disciplines and the end-to-end delays for these two disciplines. This chapter discusses the details of hardware implementation of the two scheduling algorithms. The same network model assumed in the software implementation is also assumed here. Each block in the hardware implementation is explained with flow diagrams, where necessary. The traffic arrival pattern used in the hardware implementation is obtained from the software simulator and the results (packet arrival and departure times) of the hardware implementation are processed in software to obtain the desired end-to-end delay for comparison. A behavioural level architecture is used for writing the VHDL codes. The hardware implementation verifies the feasibility of realizing the scheduling algorithms using the prevailing VLSI design tools.

## 5.2 Hardware Implementation

The blocks involved in the hardware implementation are shown in the block diagram of Figure 5.1. The figure has been obtained from the framework of Mehrotra in

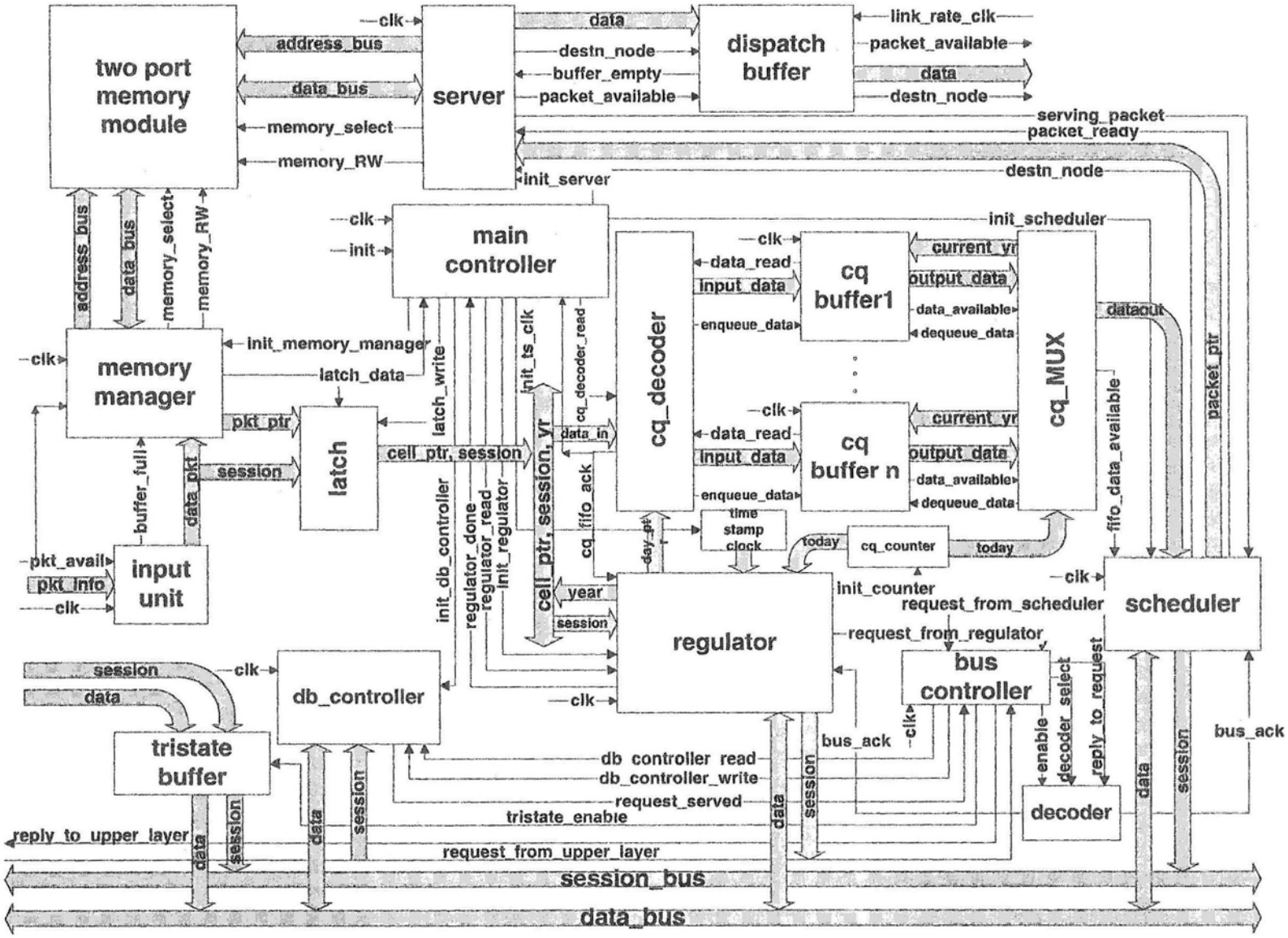



**Figure 5.1:** Block diagram for a single node

his post-doctoral work [30] with some modifications. The framework was originally designed for single node case and fixed-sized packets. With the introduction of multiple node case (network), more signals had to be introduced. Moreover, since variable-sized packets are also considered in our design, there was a need for more changes to be introduced to the existing framework. The figure details every signal flowing between the blocks. The main blocks involved in the hardware implementation are input unit, memory manager, two-port memory module, database controller, main controller, regulator, scheduler, bus controller, tristate buffer, server and the dispatch buffer. The regulator consists of calendar queue blocks, namely, decoder, buffer, counter and multiplexer to queue the packets until they are eligible. The implementation of each of these blocks is described below. The regulator queues are implemented using the calendar queues to reduce the implementation cost of maintaining the queues. Throughout the implementation, behavioural level description is used. Function verification has been done using simulator tools, and synthesis has not yet been carried out.

### 5.2.1 Input Unit

The input unit accepts the packet information and sends it to the memory manager for storage in the two-port memory. The packet information arrives from the testbench which reads the packet information file, *pif.dat*, created by the software simulator's data handler. The packet arrival information from this file is sent to the input unit, which then sends this information to the memory module. The packet information received by the input unit includes node number, connection number, packet number, packet length and

arrival time. The input unit is implemented as an FSM with two states, *wait_for_packet* and *receive_packet*.

## 5.2.2 Memory Manager

The memory manager is capable of accepting the packet information from the input unit and storing it in the two-port memory for future retrievals. It generates a pointer to the packet and stores the packet information in the two-port memory at this pointer. It also latches the packet pointer and the connection's details obtained from the input unit into the regulator and cq_decoder (these blocks will be discussed in Sections 5.2.6 and 5.2.7). The memory manager uses the pointer as an address location in the memory to store the packet information in memory. This detail is required by the server at the time of dispatching the packet.

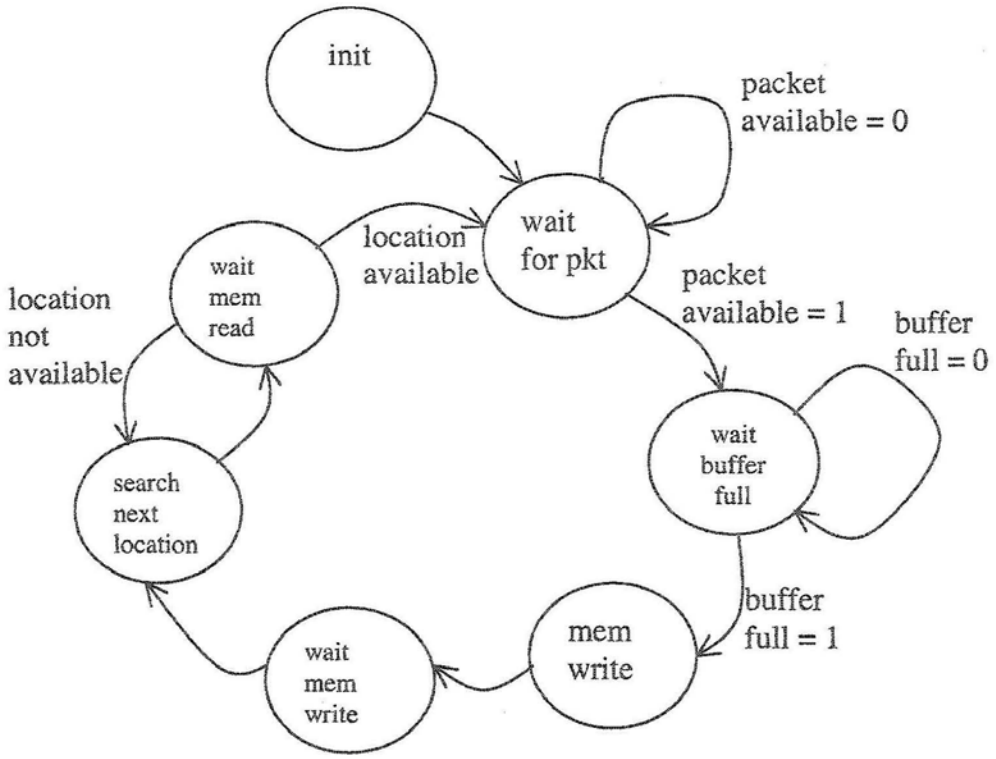



**Figure 5.2:** State diagram of memory manager

The memory unit is implemented as an FSM with seven states: *init*, *wait_for_packet*, *wait_for_memory_write*, *wait_buffer_full*, *memory_write*, *search_next_location* and *wait_for_memory_read*. The state diagram is shown in Figure 5.2. The memory manager receives the initialize signal from the main controller. It then goes to state, *wait_for_packet* and remains there until a packet becomes available. Once the packet is available, the memory manager writes the packet details into the data bus and the packet pointer into the address bus entering the memory module. The memory manager then waits for the packet details to be written into the memory and then searches for the next location in memory where the next packet can be written. Once the next location is found, the memory manager reads that location and is ready to receive the next packet. Sufficient time is allowed for the memory manager to search for the next location in memory. There are at most 256 searches because the size of the two-port memory is 256 locations.

### 5.2.3 Two-port Memory

The memory is implemented as 256 X 4, that is, 1K bits and does not use any delay to access the memory. So, the delay is assumed by the memory manager or the server. That is, the memory manager and server wait for sufficient time (memory access time) and then read from or write into the memory. This size of the memory is found to be sufficient for testing the scheduling algorithms.

### 5.2.4 Main Controller

The main controller initializes mainly the memory manager, database controller, regulator, scheduler, server and time stamp clock. It is built as a finite state machine with 5 states: *start*, *init*, *wait_for_cell*, *start_regulator* and *wait_for_cq_fifo*. In the *init* state, it initializes all the blocks and then waits for cell. When the memory manager gives the command to latch data from the latch, it latches the data onto a bus and initiates the regulator and waits for the regulator to finish reading the data. Once the regulator read is done, the controller asks the cq_decoder to read the packet pointer and waits for an acknowledgement from the cq_fifo. Once the acknowledgement is received, the main controller goes back to state *wait_for_cell* until the next packet arrives.

### 5.2.5 Database Controller

The tristate buffer block receives the connection number and the corresponding details of the connection like $X_{min}$, $X_{ave}$, I, source node, destination node and the connection's weight required for the regulator and scheduler blocks. These details are read through the tristate buffer and stored in the database controller (db_controller). When the regulator or scheduler requires this information for any particular connection, they request bus access to the bus controller and then read the connection's details from the db_controller.

## 5.2.6 Regulator

The regulator helps to smooth the traffic arrival pattern in the scheduler. The regulator delays packets from those connections that send transmit at a rate higher than that guaranteed to them. The regulator calculates the eligibility time for a packet and if the packet is not immediately eligible, then it is delayed by the delay unit (to be discussed in the next subsection) before sending it to the scheduler. The regulator block is omitted in a work-conserving scheduler. In such a case, the packets directly enter the scheduler from the memory manager. The regulator is implemented as an FSM with eight states: *start*, *init*, *wait_for_cell*, *request_session_data*, *wait_for_session_data*, *calc_eligible*, *store_cell* and *wait_for_ack*. The regulator is initialized by the main controller. When it goes to state *wait_for_cell*, if a packet arrives, the regulator gets the connection number from the packet and then checks to find out whether this particular connection's details are already available in the regulator's cache. If it is available then the details are obtained from the cache. Otherwise, the regulator tries to get hold of the bus by sending a request to the bus controller. Once the bus controller acknowledges, the regulator obtains the connection's information such as $X_{min}$, $X_{ave}$, I from the db_controller and then calculates the eligibility time of the packet in state *calc_eligibile*. In order to calculate the eligibility time, the regulator needs to know the arrival time of the packet into the regulator. For this, a time stamp clock is present, which gives the time at which the packet enters the regulator. The time stamp clock is simply a counter which is enabled when the regulator is initialized. With the help of the arrival time and the information about the connection,

the regulator calculates the eligibility time for the packet. Once the eligibility time is calculated, the eligibility details are sent to the delay unit.

### 5.2.7 Delay Unit

The delay unit consists of four blocks and is used to delay a packet by the required amount of time which is specified by the regulator. The delay unit is also not present in the work-conserving scheduler. The delay unit is implemented as a calendar queue [10]. As an example, one needs to schedule an event on a calendar by writing down the event at the appropriate page, with each page corresponding to one day. There may be any number of events for a particular day. The time of each event is based on its priority. Scheduling an event in the calendar corresponds to the *enqueue* operation and reading the today's page in the calendar and removing the first event for today is the *dequeue* operation. Implementing the same in hardware consists of a set of queues, one per page of the calendar. In this implementation, there are eight queues (cq_fifo) each corresponding to a day of the year. That means there are eight days in a year. If there is a packet in one of the queues but it is not currently eligible, because it does not match with the current year, then its eligibility will correspond to the same day of next year or the year after the next. In this implementation, each day corresponds to one cycle. The counter is incremented by one, every cycle.

- *Calendar queue decoder*: This unit passes on the packet information to the appropriate cq_fifo. It receives the day information from the regulator and uses this information to select one of the eight cq_fifos and then if the cq_fifo is ready

to accept the packet information, the cq_decoder sends the packet information to the cq_fifo and waits for an acknowledgement from the cq_fifo. Once the acknowledgement is received, the cq_decoder passes on this acknowledgement signal to the regulator and main controller.

- *Calendar queue fifo (buffer)*: This unit stores packet information in its queues. There are eight instances of the cq_fifo in this implementation corresponding to eight days in a year. It has two main operations: *enqueue* and *dequeue*. Thus it has the following states in its FSM: *init*, *wait_for_event*, *enqueue*, *dequeue*. When in state *enqueue*, the data is stored in the sorted queue. Each fifo queue has 4 buckets one per year and the data is entered into the bucket corresponding to the current year. The year value is obtained from the data which has two additional bits indicating the year of arrival of the packet. The data that goes out of the cq_fifo into the cq_mux does not have the two bit year information, as it is already used up to select the particular bucket in the cq_fifo. When in state *dequeue*, only that bucket which corresponds to the current year is checked and the data if there is any in that bucket of the cq_fifo queue is removed.
- *Calendar queue multiplexer*: The cq_mux selects one of the available cq_fifos based on the current year and date received from the cq_counter. From this cq_fifo, it dequeues the data available and sends this data to the scheduler block.
- *Calendar queue counter*: The cq_counter counts one day at a time.

### 5.2.8 Bus Controller

The bus controller controls the access to the bus between the regulator and the scheduler. When either the regulator or the scheduler wants to gain access to the bus, it places the request to the bus controller. The bus controller, first checks to see if there is no other request currently being served and if not, provides access of the bus to the regulator or scheduler. If there is a request from regulator and scheduler, the bus controller uses a decoder to select one of regulator and scheduler and provides bus acknowledge to it. The bus controller also enables the tristate buffer and sends the *db_controller_read* and *db_controller_write* control signals to the db_controller.

### 5.2.9 Scheduler

The scheduler is implemented as an FSM with 10 states: *start*, *init*, *wait_for_event*, *enqueue*, *request_session_data*, *wait_for_session_data*, *calc_finish_number*, *dequeue*, *wait_to_serve_packet* and *serve_packet*. The state diagram of the scheduler is shown in Figure 5.3.

The scheduler is initialized by the main controller. Once it is initialized, the scheduler goes to the *wait_for_event* state. If a packet is available, it is indicated by the cq_mux and the scheduler immediately enqueues the data. The scheduler now caches important data required in calculating the finish number. Then, the scheduler requests the connection's details from the db_controller by sending a request to access the bus. Once it gains control of the bus, the connection's details like the connection's weight, the source node, the destination node, etc., are obtained from the db_controller. With these details,

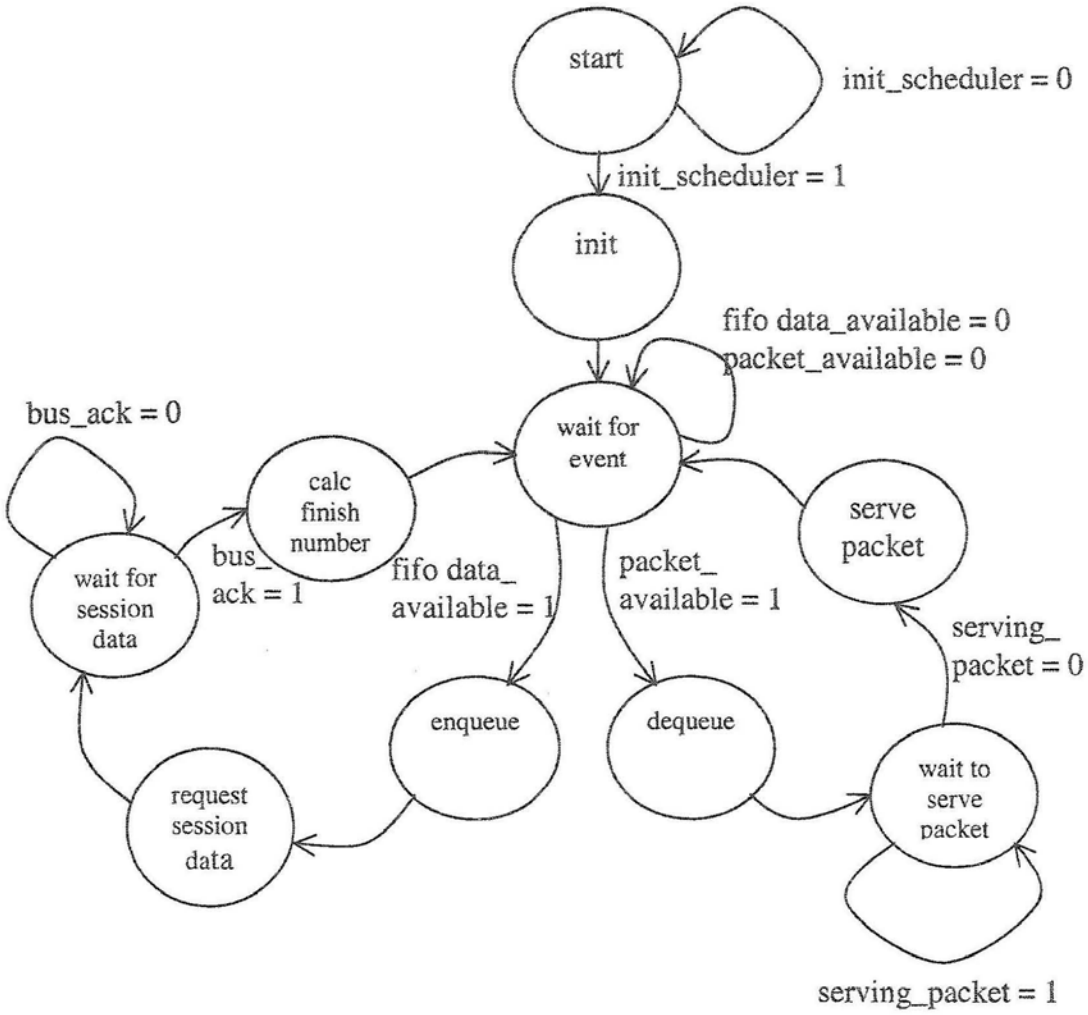



**Figure 5.3:** State diagram of the scheduler

the scheduler calculates the finish number of a packet and then selects a packet with the least finish number to be dequeued. The scheduler now waits for the server to be ready to receive a packet pointer. Once the server is ready, the scheduler sends the packet pointer to the server. Both the schemes (WFQ and $WF^2Q+$) follow the same procedure. The only difference is in the calculation of the finish number. During the state, *calc_finish_number*, the $WF^2Q+$ scheduler updates only one pair of start and finish number for a particular connection, while the WFQ scheduler maintains a pair of queues, one for start time and

the other for finish time for each connection. The new start time and finish time calculated will be placed in the appropriate positions in the two queues.

### 5.2.10 Server and Dispatch buffer

The server section has two blocks namely the server and the dispatch buffer.

- *Server*: The server receives the packet pointer from the scheduler and retrieves the corresponding packet from the two-port memory. The server is implemented as an FSM with eight states: *idle*, *init*, *wait_for_packet*, *wait_for_memory_read*, *memory_read*, *wait_for_buffer*, *write_to_buffer* and *wait_for_memory_write*. The server is initialized by the main controller. When the server is in state *wait_for_packet*, it sets the *serving_packet* signal low, to indicate the scheduler that it is not serving any packet currently. Now, the scheduler sends a packet and the server, sets the *serving_packet* signal high indicating the scheduler that it is currently busy serving a packet and is not free to receive any new packet pointer. The server then selects the memory module and sends the packet pointer in its address bus. The server now waits for the memory to read the address and map the corresponding data into the data bus. Once the data is available in the data bus, the server reads the data and checks the dispatch buffer until it is ready to receive a packet. When the dispatch buffer is empty, the server sends the packet to the dispatch buffer and clears that particular location in the memory, so as to allow other packets arriving in the memory to be written in that location. The dispatch

buffer clears the memory location by setting the *valid_bit* of that location to low which implies that the location is empty.

- *Dispatch buffer*: The dispatch buffer is clocked by the link rate clock. It sends the packets based on the output link rate. As soon as it receives data from the server, it informs the server that it is not free anymore by setting the *buffer_empty* signal low. It sends the packet into the link and if there is more space available in the link then it raises the *buffer_empty* signal. Otherwise, it will raise the signal only in the next link rate clock, when the packet has been served.

The above discussion of the hardware implementation is only for a single node. In the case of multiple nodes the whole block diagram described in Figure 5.1, will be repeated for each node. This is shown in Figure 5.4.

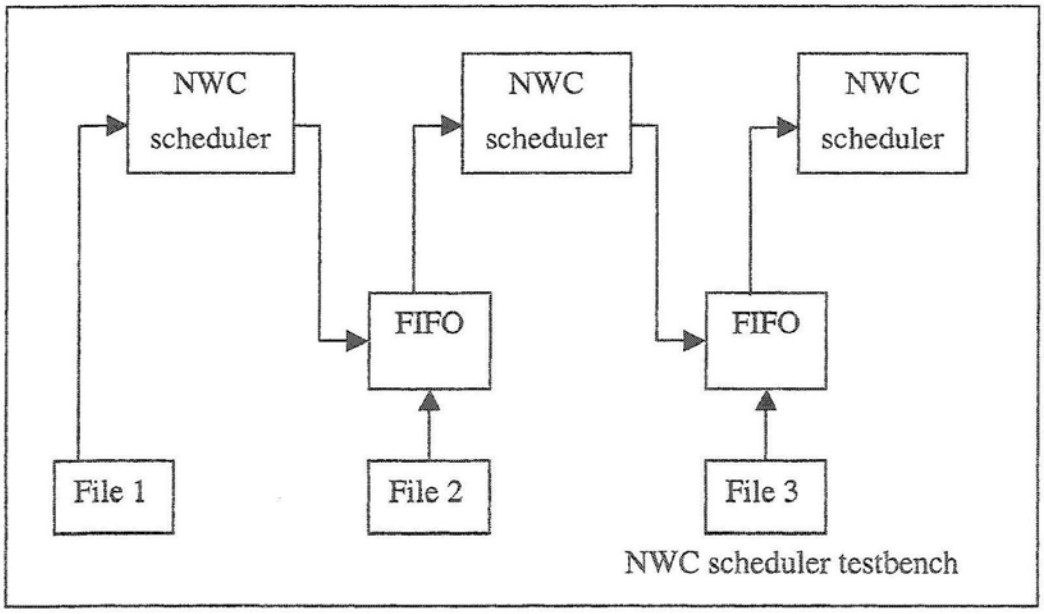



**Figure 5.4:** Block diagram of multiple node implementation

Two nodes are connected with the help of a FIFO in between, which helps in buffering new packets entering that particular node (from File 2 or File 3) and packets departing the previous node. The FIFO sends the packets into the node (input unit of node) one a first come first served (FCFS) basis. Files 1, 2 and 3 are the packet information files (pif.dat) which contain information such as packet number, connection number, node number arrival time and also the arrival time of the next packet. The packets are actually generated in the software simulator and the details are stored in the above mentioned files. Thus, the input unit of each node gets the information about the arrival time of the next packet while reading the current packet. This information helps the input unit to read from the file only when required, that is, only when a packet is available and not every cycle.

## 5.3 Testing

A testbench is written to test the functionality of the hardware implementation. Some of the important signals involved are traced in Figure 5.5. This first signal shown is the system clock. A clock with a frequency of 100 MHz is used. A system clock having a period of 12480 ns is derived from the base clock. This time is required to allow packets from four different links to arrive into the switch at the same time. The first packet arriving is read from the packet arrival information file. The details of the packet such as node number, connection number, packet number and arrival time of the packet are all stored in the file. This detail is stored into the signal, *packet_infoN*. The *packet_availableN* signal goes high when the packet has been read from the file and is

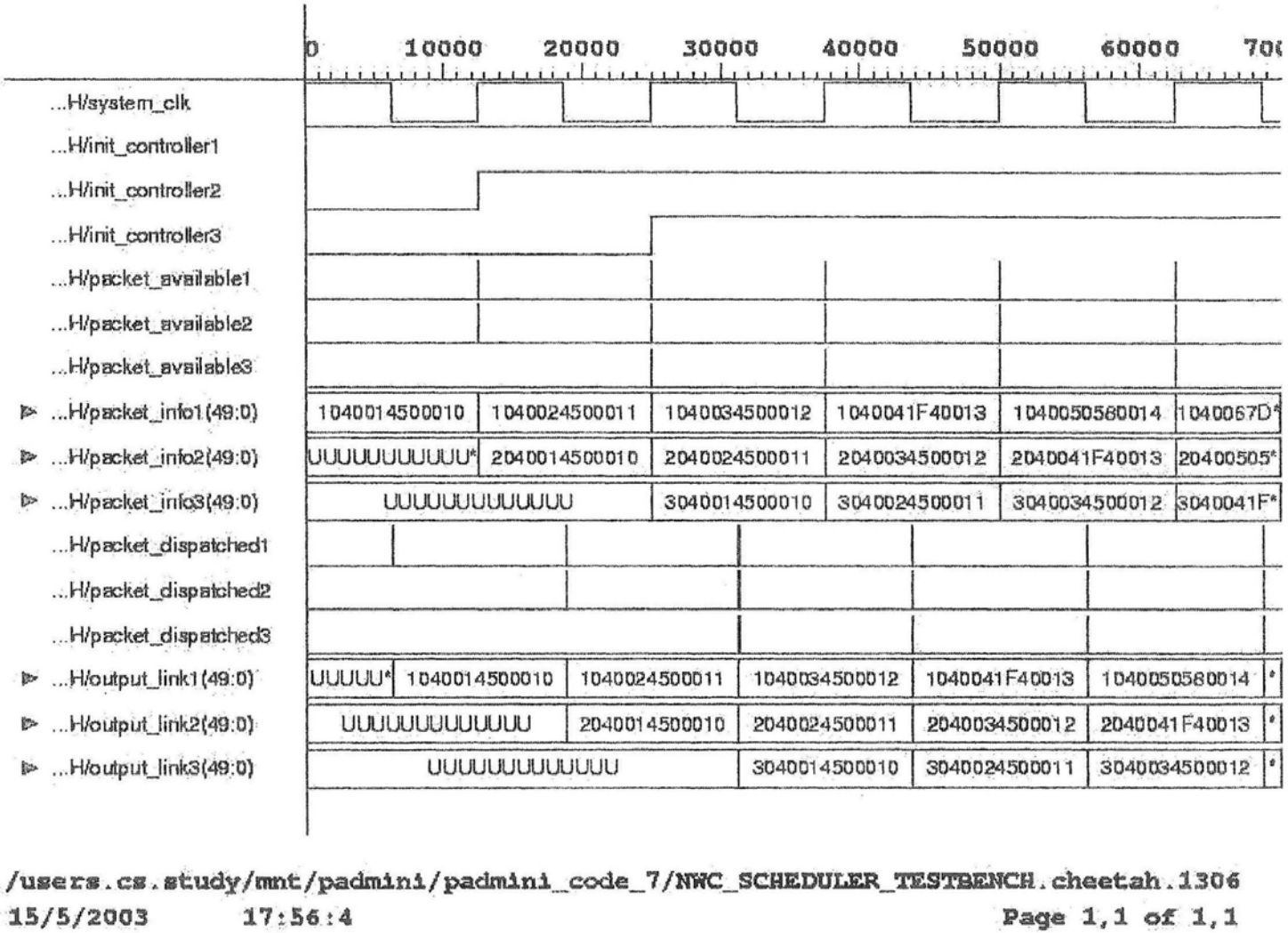
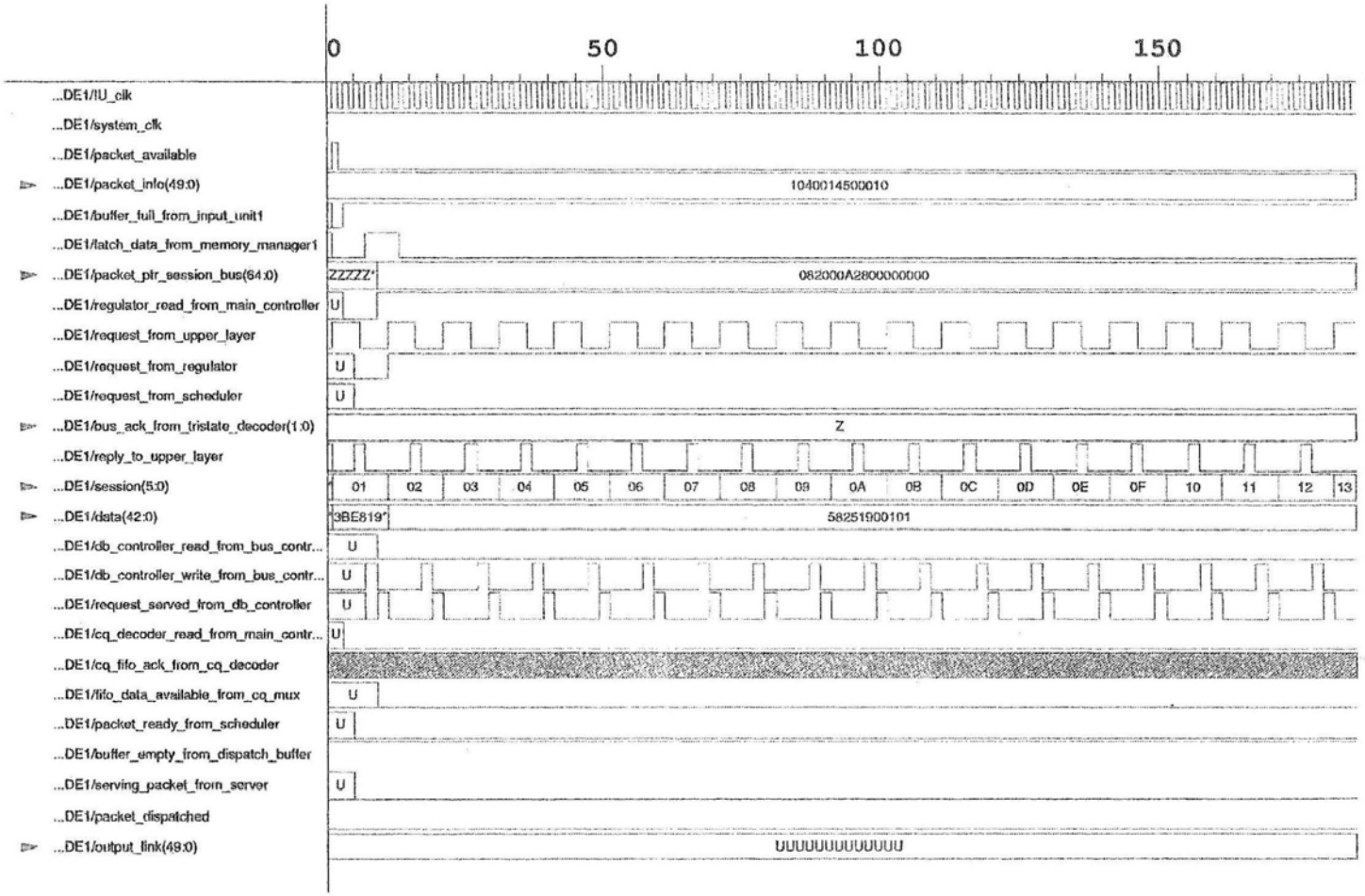


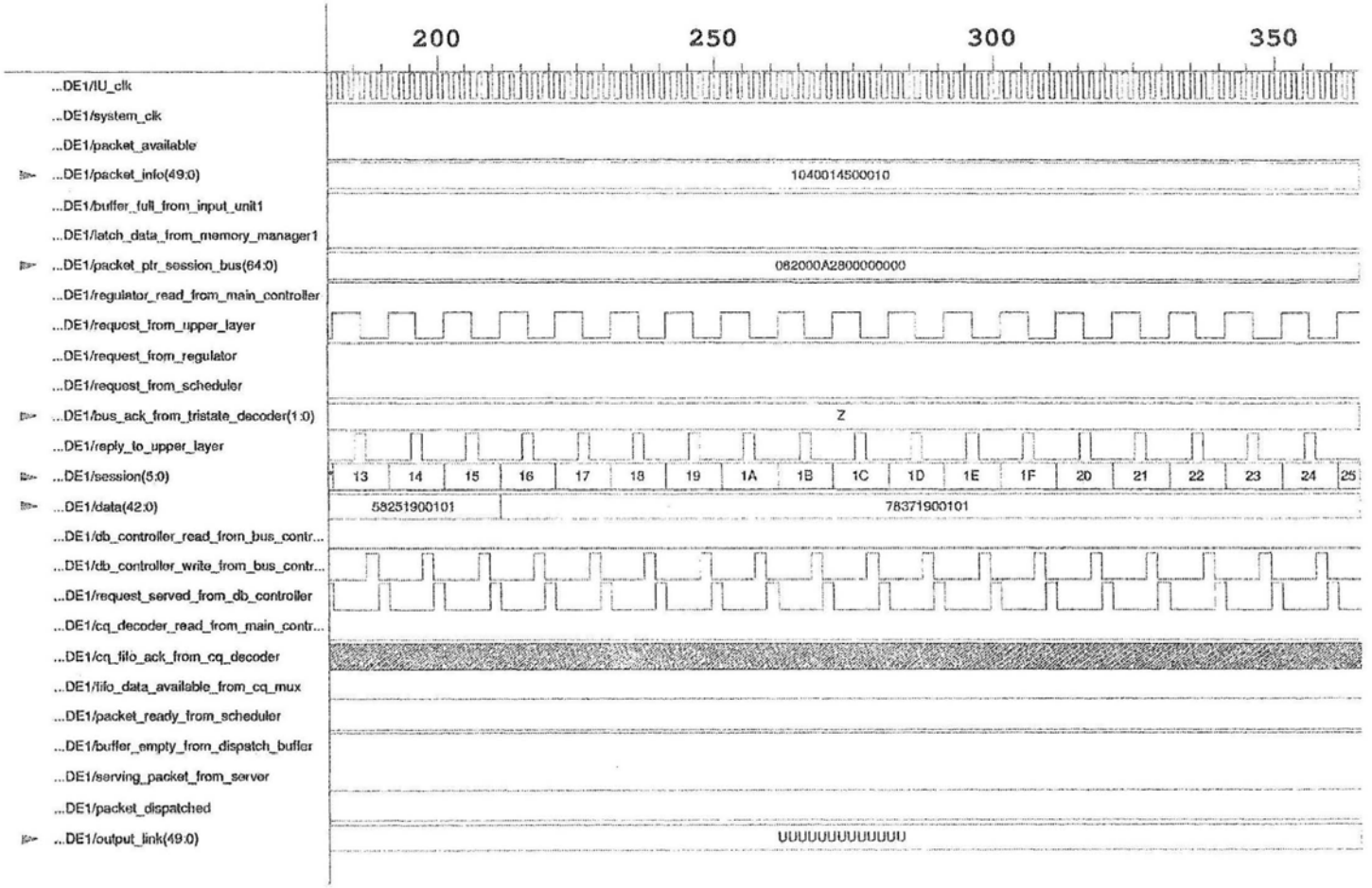

**Figure 5.5:** Timing diagram for the hardware implementation of WFQ scheduling discipline

found to have arrived at the current time. In the example shown in Figure 5.5, a packet arrives at node1. The destination node for all the packets entering at nodes N1, N2 or N3 is node 3. Therefore, the destinaltion node for the packet under consideration is node3. Once the packet information has been read, the details are sent through *packet_info1* and the *packet_available1* signal goes high. Once this signal goes high, the input unit reads the corresponding packet information from the *packet_info1* and sends this information to the memory manager which assigns an address to the packet and stores the packet in the two port memory. The packet address and the connection details are alone sufficient for the regulation scheduling. The packet details are recovered from the two port memory only when the packet is ready to be served. In the Figure 5.5, the packet arrives at time 1 ns, and is ready to be dispatched at time 6240 ns. At this time, the dispatch buffer sets the *packet_dispatched1* signal high and sends the packet through the *output_link1*. This packet then enters node 2. The code has been written such that the packet details are read by the node, during the first half of the system clock while the packet is dispatched through the output link at the second half of the system clock. Therefore, the packet which is dispatched from node 1 at time 6240 ns, will be available at node 2 at 12480 ns. At this time, the *packet_available2* signal goes high with the corresponding packet's information available in *packet_info2*. Notice the packet information when the packet enters node 1 is given in hexadecimal as "1040014500010". This information changes to "2040014500010" when the same packet dispatched form node 1 enters node 2. This is because, the first two bits of the *packet_info* are allotted to node information and when the packet is in node 1, the values of the first two bits are 1 $((01)_2)$ and when at node 2, these values change to 2 $((10)_2)$. The packet is dispatched from node 2 at time 18720 ns.



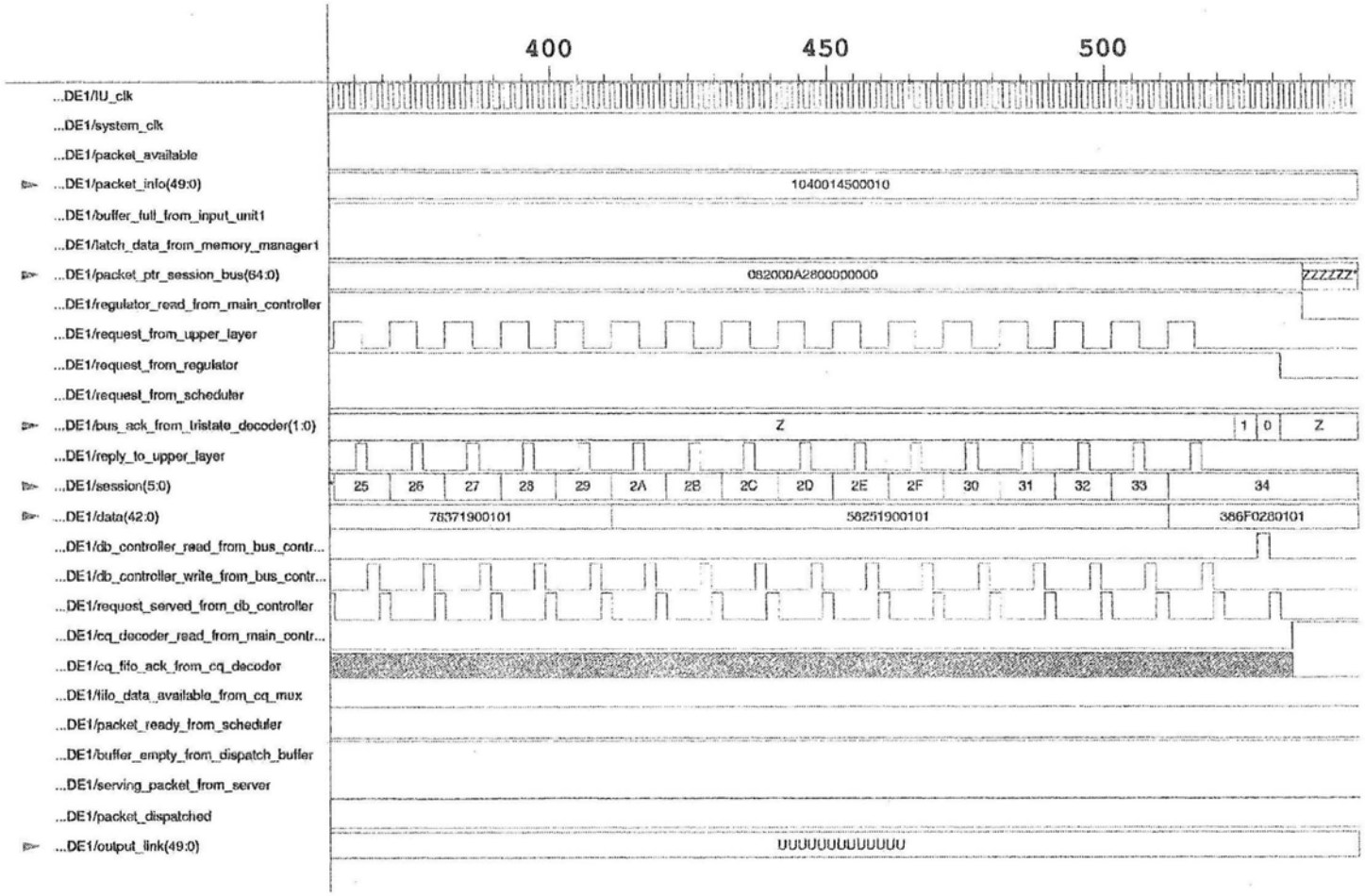

s/users.cs.study/mnt/padmini/padmini_code_7/NWC_SCHEDULER_TESTBENCH.cheetah.883
22/5/2003 22:42:46 Page 1,1 of 1,1

**Figure 5.6:** First packet arrival



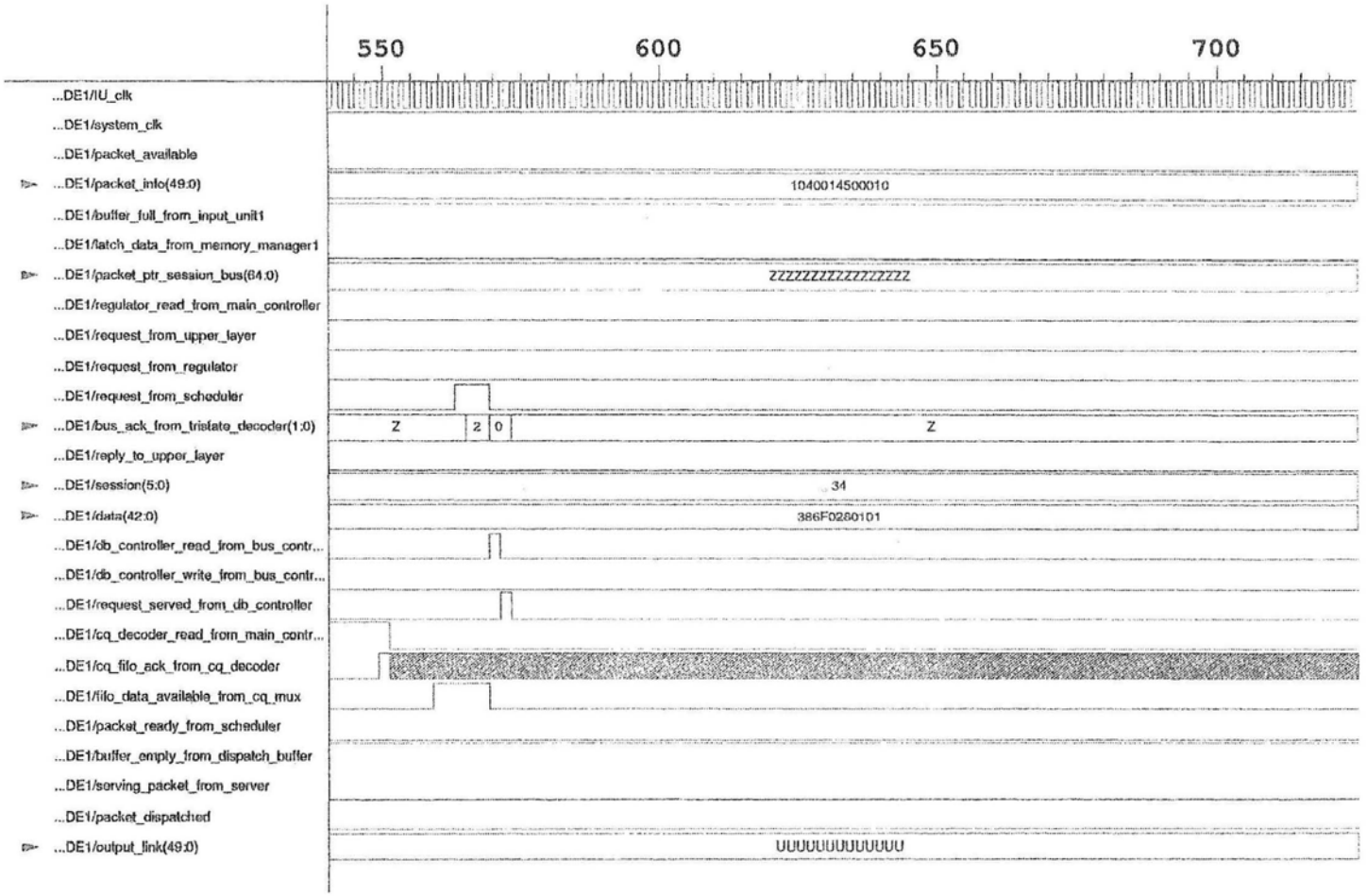

s/users.cs.study/mnt/padmini/padmini_code_7/NWC_SCHEDULER_TESTBENCH.cheetah.883
22/5/2003 22:14:35 Page 1,1 of 1,1

**Figure 5.7:** Reading connection details from upper layer



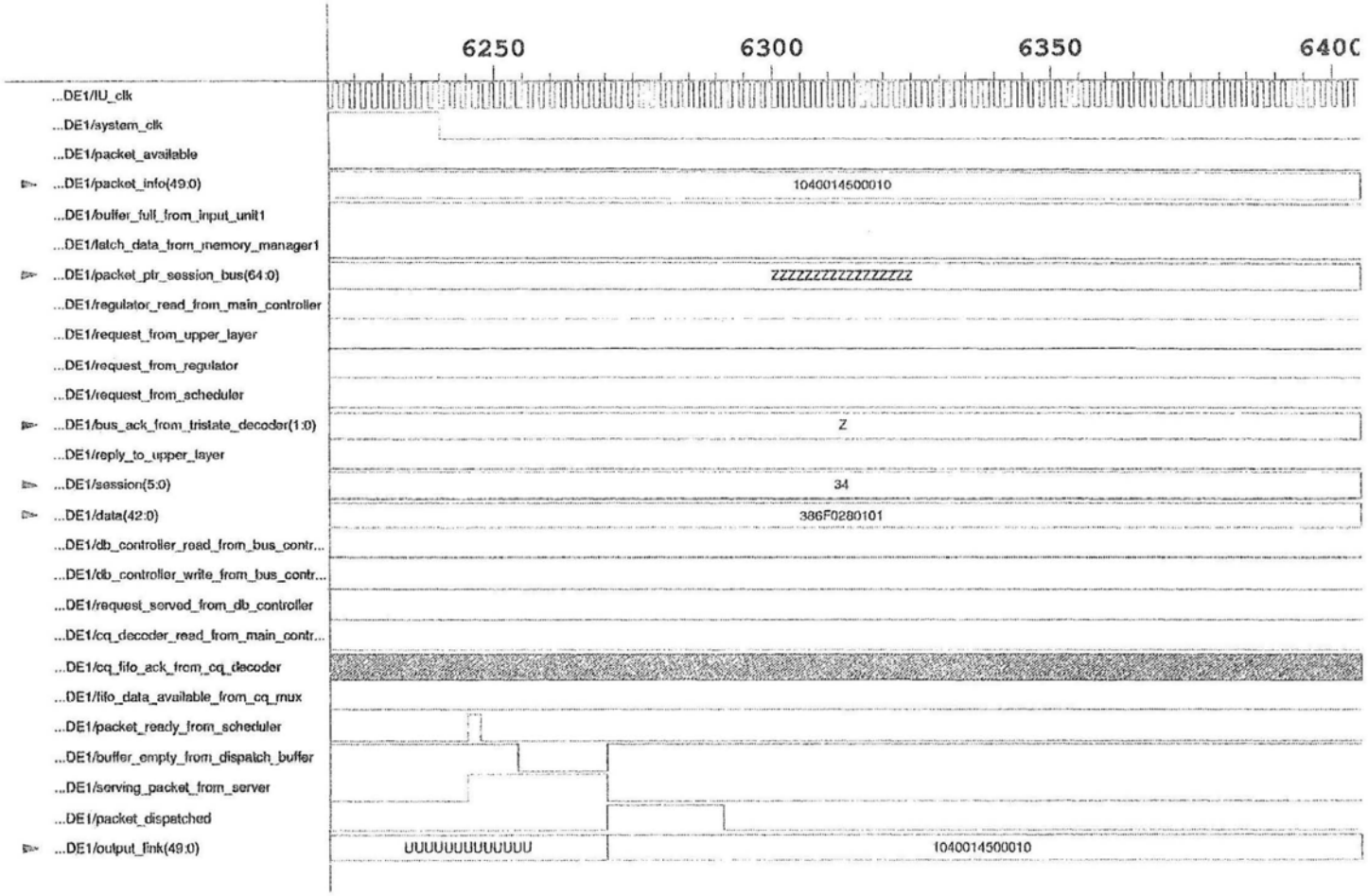

s/users.cs.study/mnt/padmini/padmini_code_7/NWC_SCHEDULER_TESTBENCH.cheetah.883
22/5/2003 22:15:7 Page 1,1 of 1,1

**Figure 5.8:** Regulator reads connection's details from db_controller




s/users.cs.study/mnt/padmini/padmini_code_7/NWC_SCHEDULER_TESTBENCH.cheetah.883
22/5/2003 22:15:54 Page 1,1 of 1,1

**Figure 5.9:** Scheduler reads connection's details from db_controller




s/users.cs.study/mnt/padmini/padmini_code_7/NWC_SCHEDULER_TESTBENCH.cheetah.883
22/5/2003 22:17:0 Page 1,1 of 1,1

**Figure 5.10:** Packet dispatched from server

This packet enters node 3 at time 24960 ns as shown in Figure 5.5 and is dispatched from node 3 at time 31200 ns.

The arrival and departure of the first packet at node 1 is explained below in detail. Figure 5.6 shows the timing diagram for the arrival of the first packet. From this figure, *packet_available* signal goes high showing that a packet is available and ready to be read by the memory manager. The corresponding packet information is present in *packet_info* signal. The memory manager now sends the data to the memory and latches the packet pointer information (*latch_data_from_memory_manager*). In the meanwhile, as soon as the simulation starts, a request arrives from the upper layer to read all the connection's details and store them in the *db_controller*. This is indicated by the signal *request_from_upper_layer*. Also, since the data has been latched by the memory manager, a request for the bus to read the connection's details arrives from the regulator. This is shown by the signal *request_from_regulator* in Figure 5.6. Since the request from upper layer arrives first and also because it has a higher priority than the request from either the regulator or scheduler, the bus acknowledges the request from upper layer. Therefore, *reply_to_upper_layer* signal goes high and the connection's details are read from *session* (connection number) and *data* ($X_{min}$, $X_{ave}$, $I$ values) signals. The request from upper layer goes high until all the connection's details have been written into the db_controller. Figure 5.7 shows the details of connection $(13)_{16}$ to connection $(25)_{16}$ being read. In Figure 5.8, all the connection's details upto connection $(34)_{16}$, which is connection 52, have been read. Now the regulator's request is acknowledged by the bus. This is shown by the signal, *bus_ack_from_tristate_decoder*, becomes 1 (acknowledging regulator's request). So, the regulator receives the details of the connection to which the

first packet belongs from the db_controller through the bus. Once this is received, the regulator calculates the eligibility time of the packet. Since the packet under consideration is the first packet in this connection (also the first packet in the simulation), it is immediately eligible. Therefore, the cq_fifo acknowledges the cq_mux and so, *fifo_data_available* signal from the cq_mux goes high as can be seen in Figure 5.9. The packet pointer is then read by the scheduler. The scheduler now requests access to the bus. Therefore, the *request_from_scheduler* signal goes high. The bus immediately acknowledges the scheduler's request and so *bus_ack_from_tristate_decoder* signal becomes 2 (acknowledging scheduler's request). Now the connection's detail (packet length) is sent from the db_controller through the bus_controller to the scheduler. This is shown by *db_controller_read_from_bus_controller* signal going high and therefore *request_served_from_db_controller* signal also goes high. Once the scheduler calculates the next packet to serve (in this case, the first and only packet), the *packet_ready* signal from the scheduler goes high as shown in Figure 5.10. This *packet_ready* signal goes high only during the second half of the clock cycle, in order to accommodate the arrival of other packets, if any, from other connections or input links, during the same time cycle. All the packets arrive during the first half of the system clock and during the second half the eligible packet(s) are served at link rate. Once the *packet_ready* signal goes high, since the buffer is empty in the dispatch buffer (*buffer_empty* signal is high), the server serves the packet (*serving_packet_from_server* signal goes high). Therefore, now, the *packet_dispatched* signal goes high. The next packet arrives at the beginning of the next system clock (Appendix E).

The end-to-end delay plots obtained from the simulation of the hardware testbench is the same as that obtained for software simulation. As an example, the end-to-end delay plots for the case of fixed-sized packets without cross-traffic, with least best-effort traffic and an output link rate of 50 bytes/ms for WFQ and $WF^2Q+$ ae shown in Figures 5.11a and 5.11b respectively. These two figures are exactly the same as Figures 4.7a and 4.7b, thereby showing that the results obtained for hardware are the same as that for software. For all the other cases considered in software simulation, the results obtained for hardware matched that obtained for software. The only difference is that the

sp

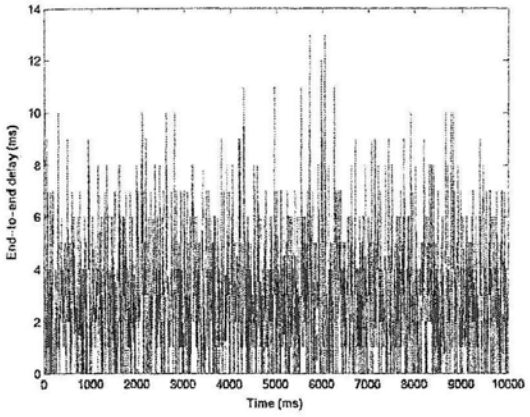



**Figure 5.11a:** End-to-end delay of WFQ scheduler for fixed-sized packets without cross-traffic with least best-effort traffic and an output link rate of 50 bytes/ms.

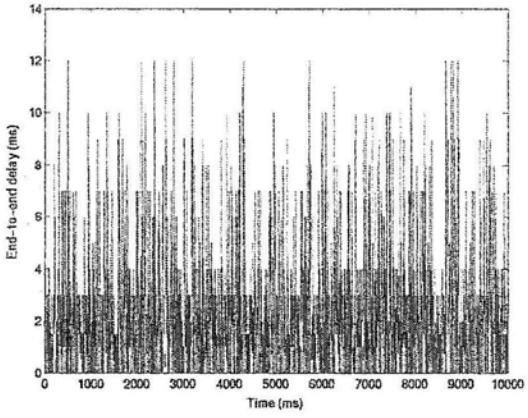



**Figure 5.11b:** End-to-end delay of $WF^2Q+$ scheduler for fixed-sized packets without cross-traffic with least best-effort traffic and an output link rate of 50 bytes/ms.

## 5.4 Discussion

This chapter described the hardware implementation in detail. The cost involved in implementing the regulator queues has been reduced by the use of calendar queues in

hardware. But the cost involved in selecting the next packet to be served by the scheduler could not be reduced. The code was written using the behavioral architecture and the individual blocks were tested with a testbench for each block and then the blocks were included into one big block constituting a single node (shown in Figure 5.1) and this code was simulated. Finally, the multiple node case was implemented. The synthesis has to be carried out.

# Chapter 6

# 6. Conclusions and Suggested Future Work

## 6.1 Conclusions

The choice of a particular scheduling discipline for high-speed packet-switched networks plays an important role in fast switching. This thesis compares the two scheduling disciplines WFQ and $WF^2Q+$ in terms of fairness, algorithmic complexity and end-to-end delay bound they guarantee for both fixed- and variable-sized packets. It had been reported that $WF^2Q+$ performs better than WFQ in terms of end-to-end delay in every sense. This belief is based on the results of fixed-sized packets (such as in ATM networks) only. Contrary to this belief, in this thesis, it is shown that for the case of variable-sized packets, as found in high-speed networks, the delay bound provided by WFQ is lower than that provided by $WF^2Q+$.

It was shown in Section 3.3.4 that $WF^2Q+$ which is the same as $WF^2Q$ with reduced complexity involved in calculating the system virtual time function, has a WFI that is not a function of the number of connections ($N$), while WFQ has a WFI that is a function of WFI. Therefore, in terms of the fairness property, $WF^2Q+$ is a better choice.

The speed with which a scheduling discipline serves packets should match the switching speed. Thus it is highly desirable to reduce the time complexity of the scheduling algorithm chosen. Among the two scheduling disciplines considered in this thesis, the three tasks of computing the system virtual time function, maintaining a set of

queues sorted by eligibility time in the regulator, and, maintaining the set of eligible connections sorted by virtual finish times can be accomplished with $O(\log N)$ complexity in WF$^2$Q+ discipline and with $O(N)$ complexity in WFQ discipline. Thus for high-speed networks WF$^2$Q+ discipline is a better choice in terms of the complexity involved in selecting the next packet to transmit.

From the plots on end-to-end delay for real-time connections shown in Section 4.7, it clear that WF$^2$Q+ has a lower end-to-end delay when the packet size is fixed. However, when the packet size is variable, the end-to-end delay of WFQ is lower. This leads to two observations. It can be said that WFQ is a better choice when the packet sizes are variable. However, the low delay provided by WFQ for variable-sized packets is at the cost of increased delay for packets of other connections (best-effort, Poisson sources and constant sources). Thus it is unfair in the service provided to other connections. In the case of WF$^2$Q+, the slightly higher delay obtained by real-time connections of variable-sized packets is because the algorithm is trying to be fair to all the connections. Thus WFQ is a better choice than WF$^2$Q+ for variable-sized packets only if the connections other than the real-time connections do not have a strict service guarantee requirement.

## 6.2 Other Contributions:

- A model for the distribution of packet lengths for variable-sized packets: From the data obtained from Traffic CAIDA Organization, various packet lengths and their arrival patterns have been plotted as a probability density function.

- With the help of framework provided by Mehrotra for the system [30], an $O(1)$ priority queue implementation (calendar queue implementation) of the regulator queues has been carried out.

## 6.3 Suggested Future Work

This section discusses some of the tasks that can be done in future in order to improve the existing implementation and also to extend it for the needs of the future communication networks.

- *Synthesizing the hardware blocks:* The hardware implementation currently consists of behavioral level architecture. The code needs to be synthesized. Currently, work is on progress in synthesizing the blocks involved in the hardware implementation, one by one. The result of this synthesis would help in deciding whether an ASIC is required or an FPGA is sufficient for building the scheduling disciplines in hardware for use in high-speed networks.
- *Reducing the space complexity:* It is possible to reduce the implementation complexity to be less than $O(\log N)$. The implementation complexity can be reduced such that it is no more a function of number of connections, rather a function of number of discrete rates [4]. In the case of ATM networks (fixed-sized cells), the server supports fixed number of rates and groups the connections with the same rate together. Thus the number of queues is reduced from being equal to the number of connections to the number of service rates. Now, the only requirement is to schedule among the connections at the head of each group. The

complexity grows with number of rates for virtual time completion and priority management.

In the case of ATM networks, all the connections that share a common rate are in one group. From each group, the connection with the smallest virtual starting time is placed in the scheduler. The advantage of such a policy is that if connections in a group are eligible, the connection within the group in the scheduler may also be eligible since it has the smallest virtual start time in the group. Since it also has the smallest virtual finish time in the group, the packet with the smallest eligible finish time in the scheduler is the one with the smallest eligible finish time among all packets within the group. Each rate group has a linked list of time stamps belonging to cells at the head of the queue for each connection in the group. The entries in the linked list are stored in order of increasing time stamp.

In the case of packet networks, it is possible to perform regulation (based on virtual start times) and scheduling (based on virtual finish times) in an integrated manner. This reduces the worst-case complexity of the overall system. Instead of using two 1-D priority queues, it is possible to use 2-D sorting structure. Here, the grouping is done such that connections having the same difference between their finish and start times are grouped together. This difference between the finish time and start time of a connection is called the service interval. Within each group, a timestamp is present to sort among connections within the group. However, in this case, a connection may not always be bound to the same group since its service interval depends on the rate and

length of the first packet in the connection's queue. The connection's queue will change its group based on the new packet that is at the head of the queue. This is because the service interval may change with change in packet length. When a connection changes its group, it can be inserted at an arbitrary position in the priority queue of new group. Therefore, it is not possible to maintain a sorted relationship within each group with FIFO queue. Now the virtual finish times of flows with similar service intervals should be sorted. The virtual finish times of connections in each group span a range of $L_{max}/L_{min}$ times the service interval, where, $L_{max}$ is the maximum packet size and $L_{min}$ is the minimum packet size for variable-sized packets. Therefore if an increase in the delay bound by a fraction of one service interval can be tolerated, then the complexity of sorting can be reduced by measuring the virtual finish times in units of fractions of the group's service interval. Thus a two-level hierarchical calendar queue (trie) is obtained.

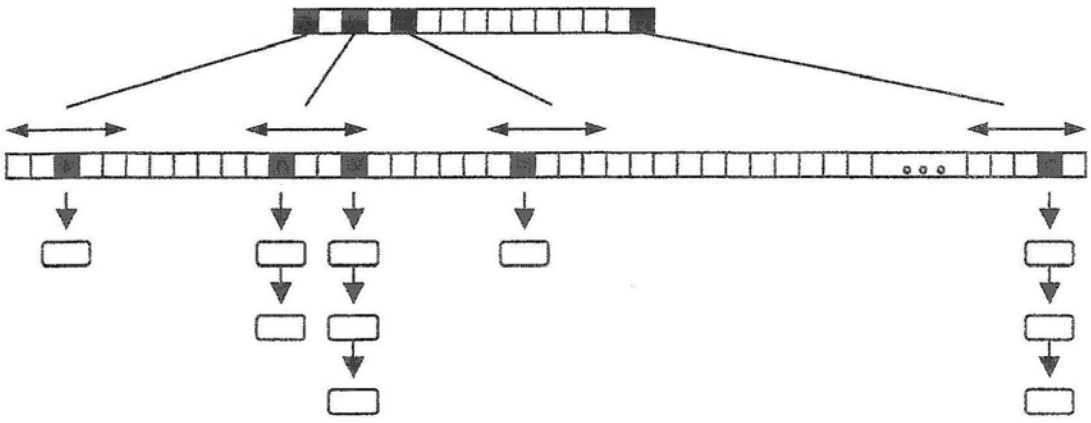


**Figure 6.1:** Hierarchical calendar queue for intra-group scheduling[4]

This is shown in Figure 6.1. The worst-case complexity is: an enqueue operation requires one insertion into a linked list along the leaves of the trie and one replacement of the value within the group data structure. A dequeue operation needs one scheduler selection among the elements within the group data structure, one removal of a connection from the head of the linked list and the cost of an enqueue. Thus the complexity is not a function of number of connections.

# References

[1] S. Keshav, "An Engineering Approach to Computer Networking – ATM Netowks, the Internet, and the Telephone Network," Addison-Wesley, 1997.

[2] P. Newman, G. Minshall and T. L. Lyon, "IP Switching – ATM Under IP," *IEEE/ACM Transactions on Networking*, Vol.6, No.2, April 1998.

[3] H. Zhang and D. Ferrari, "Rate-controlled service disciplines," *Journal on High Speed Networks*, Vol. 3, No. 4, pp. 389-412, 1994.

[4] D. C. Stephens, J. C. R. Bennett and H. Zhang, "Implementing Scheduling Disciplines in High-Speed Networks," *IEEE Journal on Selected Areas in Communications*, Vol.17, No.6, June 1999.

[5] D. Ferrari and D. Verma, "A scheme for real-time channel establishment in wide-area networks," *IEEE Journal on Selected Areas in Communications*, Vol.8, pp. 368-379, April 1990.

[6] L. Zhang, "Virtual clock: A new traffic control algorithm for packet switching networks," in *Proceedings of ACM SIGCOMM '90*, Philadelphia, PA, pp. 19-29, September 1990.

[7] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," in *Proceedings of ACM SIGCOMM '89*, pp. 3-12.

[8] A. Parekh and R. Gallager, "A generalized processor sharing approach to flow control – The single node case," in *Proceedings of INFOCOM '92*, 1992.

[9] S. Golestani, "A self-clocked fair queueing scheme for broadband applications," in *IEEE Infocom'94*, pp. 5c.1.1-5c.1.11, 1994.

[10] J. C. R. Bennett and H. Zhang, "$WF^2Q$: Worst-case fair weighted fair queueing," *Proceedings of IEEE INFOCOM '96*, July 1996.

[11] H. Zhang, "Service disciplines for guaranteed performance service in packet-switching networks," in *Proceedings of the IEEE*, Vol. 83, No. 10, October 1995.

[12] H. Zhang, "Providing end-to-end performance guarantees using non-work-conserving disciplines," *Computer communications*, Vol. 18, No. 10, Oct 1995.

[13] D. Verma, H. Zhang, and D. Ferrari, "Guaranteeing delay jitter bounds in packet switching networks," in *Proceedings of Tricomm '91*, Chapel Hill, NC, pp 35-46, April 1991.

[14] S. Golestani, "A stop-and-go queueing framework for congestion management," in *Proceedings of ACM SIGCOMM '90*, Philadelphia, PA, pp. 8-18, September 1990.

[15] C. Kalmanek, H. Kanakia, and S. Keshav, "Rate controlled servers for very high-speed networks," in *IEEE Global Telecommunications Conference*, San Diego, CA, pp. 300.3.1-300.3.9, December 1990.

[16] H. Zhang and D. Ferrari, "Rate-controlled static priority queueing," in *Proc. IEEE INFOCOM '93*, San Francisco, CA, pp. 227-236, April 1993.

[17] J. Liebeherr and E. Yilmaz, "Work conserving vs non-workconserving Packet Scheduling: An Issue Revisited," *Proceedings of IEEE/IFIP IWQoS '99*, May 1999.

[18] Cruz. R, "A calculus for network delay, Part I: Network elements in isolation," *IEEE Trans on Info. Theory*, Vol. 37, No.1, (1991), pp.114-121, 1991.

[19] M. Shreedhar and G. Varghese, "Efficient Fair Queuing using Deficit Round Robin," *Proceedings SIGCOMM '95*, Boston, August 1995.

[20] G. Chuanxiong, "SRR: An $O(1)$ Time complexity packet scheduler for flows in multi-service packet networks," in *Proceedings of SIGCOMM'01*, San Diego, California, USA, August 2001.

[21] J. C. R. Bennett and H. Zhang, "Hierarchical Packet Fair Queueing Algorithms," in *Proceedings of ACM SIGCOMM '96*, Palo Alto, CA, pp. 143-156, 1996.

[22] P. Goyal, H. M. Vin and H. Cheng, "Start-time fair queueing: A scheduling algorithm for integrated services packet switching networks," in *IEEE/ACM Transactions on Networking*, Vol. 5, No. 5, October 1997.

[23] J. Liebeherr and D. E. Wrege, "A versatile packet multiplexer for quality-of-service networks," in *Proceedings of 4th International Symposium on High-Performance Distributed Computing (HPDC-4)*, pp 148-155, August 1995.

[24] J. Liebeherr and D. E. Wrege, "A near-optimal packet scheduler for QoS networks," in *Proceedings IEEE Infocom '97 Conference*, April 1997.

[25] S. Golestani, "Congestion-free transmission of real-time traffic in packet networks," in *Proceedings of IEEE INFOCOM '90*, San Francisco, CA, June 1990, pp. 527-542, IEEE Computer and Communication Societies.

[26] D. Saha, S. Mukherjee and S. K. Tripathi, "Carry-Over Round Robin: A Simple Cell Scheduling Mechanism for ATM networks," *IEEE/ACM Transactions on Networking*, Vol. 6, No. 6, Dec 1998.

[27] J. C. R. Bennett and H. Zhang, " High speed, scalable, and accurate implementation of packet fair queueing algorithms in ATM networks," in *Proceedings of IEEE ICNP '97*, Atlanta, GA, pp. 7-14, 1997.

[28] R. Venkatesan, Y. El-Sayed, R. Thuppal and H. Sivakumar, "Performance analysis of pipelined multistage interconnection networks," *Informatica: An International Journal of Computing and Informatics*, Vol. 23, No. 3, September 1999, pp. 347-357.

[29] http://.traffic.caida.org/

[30] P. Mehrotra, "A Framework for studying work-conserving and non-work-conserving scheduling disciplines," *Centre for Digital Hardware Applications Research Technical Report Series*, MUNCEnTRe – 2000 – 001, June 2000.

# Appendix A
# Software Simulation Results

**Fixed-sized packets**
Without constant source
Least best-effort traffic pdf
600 20
800 20
1000 20
1200 20
1400 20

Output link rate = 50 bytes/ms

**WFQ scheduler**
Total traffic load at node 1 = 30.07 %
Total traffic load at node 2 = 52.26 %
Total traffic load at node 3 = 85.5 %
**Node Parameters**

| Node | Session | Entered | Total | Lost | MinDelay | AvgDelay | MaxDelay | SD |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2999 | 2999 | 0 | 0 | 0 | 0 | 0 |
| 1 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 13 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 14 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 15 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 17 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 18 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 19 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 20 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 21 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 22 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 23 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 25 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Node | Session | Entered | Total | Lost | MinDelay | AvgDelay | MaxDelay | SD |
|---|---|---|---|---|---|---|---|---|
| 1 | 26 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 27 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 28 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 29 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 30 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 31 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 32 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 34 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 35 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 36 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 37 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 38 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 39 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 40 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 41 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 42 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 43 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 44 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 45 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 46 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 47 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 48 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 49 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 50 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 51 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 52 | 8 | 8 | 0 | 1 | 1 | 1 | 0 |
| 2 | 1 | 2999 | 2999 | 0 | 0 | 0.0663555 | 2 | 0.192596 |
| 2 | 2 | 111 | 111 | 0 | 0 | 0.198198 | 2 | 0.422543 |
| 2 | 3 | 109 | 109 | 0 | 0 | 0.422018 | 2 | 0.627953 |
| 2 | 4 | 111 | 111 | 0 | 0 | 0.468468 | 2 | 0.685165 |
| 2 | 5 | 109 | 109 | 0 | 0 | 0.743119 | 3 | 0.906802 |
| 2 | 6 | 114 | 114 | 0 | 0 | 0.763158 | 3 | 0.924616 |
| 2 | 7 | 112 | 112 | 0 | 0 | 0.901786 | 3 | 0.919849 |
| 2 | 8 | 111 | 111 | 0 | 0 | 1.02703 | 4 | 1.01318 |
| 2 | 9 | 114 | 114 | 0 | 0 | 1.16667 | 3 | 0.949305 |
| 2 | 10 | 113 | 113 | 0 | 0 | 1.18584 | 4 | 1.07362 |
| 2 | 11 | 109 | 109 | 0 | 0 | 1.66972 | 5 | 1.05755 |
| 2 | 12 | 110 | 110 | 0 | 0 | 1.69091 | 4 | 1.14736 |
| 2 | 13 | 111 | 111 | 0 | 0 | 1.90991 | 11 | 1.34967 |
| 2 | 14 | 110 | 110 | 0 | 0 | 1.64545 | 5 | 1.20525 |
| 2 | 15 | 110 | 110 | 0 | 0 | 1.63636 | 5 | 1.14029 |
| 2 | 16 | 111 | 111 | 0 | 0 | 1.96396 | 5 | 1.17383 |
| 2 | 17 | 111 | 111 | 0 | 0 | 2.22523 | 11 | 1.32261 |
| 2 | 18 | 110 | 110 | 0 | 0 | 2.28182 | 11 | 1.35866 |
| 2 | 19 | 111 | 111 | 0 | 0 | 2.33333 | 11 | 1.31195 |
| 2 | 20 | 110 | 110 | 0 | 0 | 2.33636 | 11 | 1.44724 |
| 2 | 21 | 112 | 112 | 0 | 0 | 2.30357 | 6 | 1.19509 |

| Node | Session | Entered | Total | Lost | MinDelay | AvgDelay | MaxDelay | SD |
|---|---|---|---|---|---|---|---|---|
| 2 | 22 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 23 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 25 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 26 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 27 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 28 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 29 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 30 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 31 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 32 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 34 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 35 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 36 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 37 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 38 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 39 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 40 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 41 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 42 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 43 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 44 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 45 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 46 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 47 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 48 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 49 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 50 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 51 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 52 | 8 | 8 | 0 | 0 | 0.875 | 1 | 0.353553 |
| 3 | 1 | 2999 | 2999 | 0 | 0 | 1.87863 | 13 | 2.04402 |
| 3 | 2 | 111 | 111 | 0 | 0 | 4.07207 | 10 | 1.72691 |
| 3 | 3 | 109 | 109 | 0 | 0 | 3.91743 | 8 | 1.6763 |
| 3 | 4 | 111 | 111 | 0 | 0 | 4.27027 | 9 | 1.92979 |
| 3 | 5 | 109 | 109 | 0 | 1 | 4.33028 | 12 | 1.95772 |
| 3 | 6 | 114 | 114 | 0 | 0 | 4.47368 | 11 | 2.02798 |
| 3 | 7 | 112 | 112 | 0 | 0 | 3.67857 | 9 | 1.68325 |
| 3 | 8 | 111 | 111 | 0 | 0 | 4.00901 | 9 | 1.80249 |
| 3 | 9 | 114 | 114 | 0 | 0 | 4.16667 | 11 | 2.07635 |
| 3 | 10 | 113 | 113 | 0 | 0 | 4 | 10 | 1.88982 |
| 3 | 11 | 109 | 109 | 0 | 1 | 4.45872 | 11 | 2.33826 |
| 3 | 12 | 110 | 110 | 0 | 0 | 4.40909 | 12 | 2.26451 |
| 3 | 13 | 111 | 111 | 0 | 0 | 4.01802 | 8 | 1.88104 |
| 3 | 14 | 110 | 110 | 0 | 0 | 4.3 | 13 | 2.22393 |
| 3 | 15 | 110 | 110 | 0 | 0 | 3.91818 | 9 | 1.65198 |
| 3 | 16 | 111 | 111 | 0 | 0 | 4.11712 | 13 | 2.18184 |
| 3 | 17 | 111 | 111 | 0 | 0 | 3.99099 | 11 | 2.17402 |

| Node | Session | Entered | Total | Lost | MinDelay | AvgDelay | MaxDelay | SD |
|---|---|---|---|---|---|---|---|---|
| 3 | 18 | 110 | 110 | 0 | 1 | 4.27273 | 9 | 1.73638 |
| 3 | 19 | 111 | 111 | 0 | 0 | 4.02703 | 9 | 2.01188 |
| 3 | 20 | 110 | 110 | 0 | 0 | 4.15455 | 9 | 1.89043 |
| 3 | 21 | 112 | 112 | 0 | 0 | 4.09821 | 10 | 2.16004 |
| 3 | 22 | 164 | 164 | 0 | 0 | 0.689024 | 6 | 1.23134 |
| 3 | 23 | 166 | 166 | 0 | 0 | 0.963855 | 7 | 1.54877 |
| 3 | 24 | 164 | 164 | 0 | 0 | 1.03659 | 6 | 1.41374 |
| 3 | 25 | 165 | 165 | 0 | 0 | 1.24848 | 7 | 1.52382 |
| 3 | 26 | 166 | 166 | 0 | 0 | 1.42771 | 8 | 1.51884 |
| 3 | 27 | 162 | 162 | 0 | 0 | 1.6358 | 8 | 1.37876 |
| 3 | 28 | 166 | 166 | 0 | 0 | 1.57229 | 9 | 1.48278 |
| 3 | 29 | 170 | 170 | 0 | 0 | 1.85294 | 8 | 1.61361 |
| 3 | 30 | 170 | 170 | 0 | 0 | 2.15882 | 10 | 1.64848 |
| 3 | 31 | 167 | 167 | 0 | 0 | 2.29341 | 7 | 1.62386 |
| 3 | 32 | 169 | 169 | 0 | 0 | 2.39645 | 8 | 1.52515 |
| 3 | 33 | 165 | 165 | 0 | 0 | 2.33939 | 9 | 1.62769 |
| 3 | 34 | 165 | 165 | 0 | 0 | 2.64848 | 11 | 1.61952 |
| 3 | 35 | 171 | 171 | 0 | 0 | 2.74269 | 12 | 1.61846 |
| 3 | 36 | 166 | 166 | 0 | 0 | 2.88554 | 8 | 1.6457 |
| 3 | 37 | 169 | 169 | 0 | 0 | 3.2071 | 9 | 1.59627 |
| 3 | 38 | 171 | 171 | 0 | 0 | 3.2807 | 9 | 1.60417 |
| 3 | 39 | 164 | 164 | 0 | 0 | 3.82317 | 10 | 1.59519 |
| 3 | 40 | 164 | 164 | 0 | 0 | 3.73171 | 11 | 1.80842 |
| 3 | 41 | 160 | 160 | 0 | 0 | 3.99375 | 10 | 1.66547 |
| 3 | 42 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 43 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 44 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 45 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 46 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 47 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 48 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 49 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 50 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 51 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 52 | 8 | 8 | 0 | 2 | 5.125 | 9 | 2.6959 |

**$WF^2Q+$ scheduler**

Total traffic load at node 1 = 30.07 %

Total traffic load at node 2 = 52.26 %

Total traffic load at node 3 = 85.5 %

**Node Parameters**

| Node | Session | Entered | Total | Lost | MinDelay | AvgDelay | MaxDelay | SD |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2999 | 2999 | 0 | 0 | 0 | 0 | 0 |
| 1 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 13 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 14 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 15 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 17 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 18 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 19 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 20 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 21 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 22 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 23 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 25 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 26 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 27 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 28 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 29 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 30 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 31 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 32 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 34 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 35 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 36 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 37 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 38 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 39 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 40 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 41 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 42 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Node | Session | Entered | Total | Lost | MinDelay | AvgDelay | MaxDelay | SD |
|---|---|---|---|---|---|---|---|---|
| 1 | 43 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 44 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 45 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 46 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 47 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 48 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 49 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 50 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 51 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 52 | 8 | 8 | 0 | 1 | 1 | 1 | 0 |
| 2 | 1 | 2999 | 2999 | 0 | 0 | 0.267422 | 3 | 0.591677 |
| 2 | 2 | 111 | 111 | 0 | 0 | 0.009009 | 1 | 0.094916 |
| 2 | 3 | 109 | 109 | 0 | 0 | 0.192661 | 4 | 0.535362 |
| 2 | 4 | 111 | 111 | 0 | 0 | 0.27027 | 3 | 0.555075 |
| 2 | 5 | 109 | 109 | 0 | 0 | 0.449541 | 3 | 0.787474 |
| 2 | 6 | 114 | 114 | 0 | 0 | 0.54386 | 4 | 0.923061 |
| 2 | 7 | 112 | 112 | 0 | 0 | 0.660714 | 4 | 0.982308 |
| 2 | 8 | 111 | 111 | 0 | 0 | 0.765766 | 4 | 1.08674 |
| 2 | 9 | 114 | 114 | 0 | 0 | 0.842105 | 4 | 0.955459 |
| 2 | 10 | 113 | 113 | 0 | 0 | 0.893805 | 4 | 1.09677 |
| 2 | 11 | 109 | 109 | 0 | 0 | 1.41284 | 6 | 1.20981 |
| 2 | 12 | 110 | 110 | 0 | 0 | 1.48182 | 6 | 1.33225 |
| 2 | 13 | 111 | 111 | 0 | 0 | 1.62162 | 4 | 1.19499 |
| 2 | 14 | 110 | 110 | 0 | 0 | 1.43636 | 5 | 1.30118 |
| 2 | 15 | 110 | 110 | 0 | 0 | 1.34545 | 5 | 1.22123 |
| 2 | 16 | 111 | 111 | 0 | 0 | 1.79279 | 5 | 1.31625 |
| 2 | 17 | 111 | 111 | 0 | 0 | 1.81081 | 6 | 1.20084 |
| 2 | 18 | 110 | 110 | 0 | 0 | 1.96364 | 6 | 1.32147 |
| 2 | 19 | 111 | 111 | 0 | 0 | 2.01802 | 6 | 1.28036 |
| 2 | 20 | 110 | 110 | 0 | 0 | 2.02727 | 6 | 1.40323 |
| 2 | 21 | 112 | 112 | 0 | 0 | 1.95536 | 6 | 1.35588 |
| 2 | 22 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 23 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 25 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 26 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 27 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 28 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 29 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 30 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 31 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 32 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 34 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 35 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 36 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 37 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 38 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Node | Session | Entered | Total | Lost | MinDelay | AvgDelay | MaxDelay | SD |
|---|---|---|---|---|---|---|---|---|
| 2 | 39 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 40 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 41 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 42 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 43 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 44 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 45 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 46 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 47 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 48 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 49 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 50 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 51 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 52 | 8 | 8 | 0 | 0 | 0.125 | 1 | 0.353553 |
| 3 | 1 | 2999 | 2999 | 0 | 0 | 1.64388 | 11 | 2.34203 |
| 3 | 2 | 111 | 111 | 0 | 0 | 3.78378 | 10 | 1.85321 |
| 3 | 3 | 109 | 109 | 0 | 0 | 3.83486 | 10 | 1.84209 |
| 3 | 4 | 111 | 111 | 0 | 0 | 4.04504 | 13 | 2.03445 |
| 3 | 5 | 109 | 109 | 0 | 1 | 4.12844 | 14 | 2.01664 |
| 3 | 6 | 114 | 114 | 0 | 0 | 4.2807 | 16 | 2.22821 |
| 3 | 7 | 112 | 112 | 0 | 0 | 3.85714 | 15 | 2.27317 |
| 3 | 8 | 111 | 111 | 0 | 0 | 3.71171 | 14 | 2.08436 |
| 3 | 9 | 114 | 114 | 0 | 0 | 4.4386 | 16 | 2.85084 |
| 3 | 10 | 113 | 113 | 0 | 0 | 3.9646 | 16 | 2.21287 |
| 3 | 11 | 109 | 109 | 0 | 1 | 4.34862 | 15 | 2.77024 |
| 3 | 12 | 110 | 110 | 0 | 0 | 4.69091 | 16 | 3.09664 |
| 3 | 13 | 111 | 111 | 0 | 0 | 4.04504 | 16 | 2.33097 |
| 3 | 14 | 110 | 110 | 0 | 0 | 4.45455 | 16 | 2.80533 |
| 3 | 15 | 110 | 110 | 0 | 0 | 3.87273 | 10 | 1.92499 |
| 3 | 16 | 111 | 111 | 0 | 0 | 4.18018 | 16 | 2.80584 |
| 3 | 17 | 111 | 111 | 0 | 0 | 4.3964 | 19 | 3.1379 |
| 3 | 18 | 110 | 110 | 0 | 1 | 4.62727 | 14 | 2.67339 |
| 3 | 19 | 111 | 111 | 0 | 0 | 4.57658 | 15 | 3.16009 |
| 3 | 20 | 110 | 110 | 0 | 0 | 4.77273 | 16 | 3.09786 |
| 3 | 21 | 112 | 112 | 0 | 0 | 4.36607 | 14 | 2.70776 |
| 3 | 22 | 164 | 164 | 0 | 0 | 0.628049 | 11 | 1.70584 |
| 3 | 23 | 166 | 166 | 0 | 0 | 0.626506 | 12 | 1.50322 |
| 3 | 24 | 164 | 164 | 0 | 0 | 0.890244 | 10 | 1.63179 |
| 3 | 25 | 165 | 165 | 0 | 0 | 1.03636 | 9 | 1.55348 |
| 3 | 26 | 166 | 166 | 0 | 0 | 1.68072 | 16 | 2.52752 |
| 3 | 27 | 162 | 162 | 0 | 0 | 1.51235 | 10 | 1.4881 |
| 3 | 28 | 166 | 166 | 0 | 0 | 2.01205 | 17 | 2.72475 |
| 3 | 29 | 170 | 170 | 0 | 0 | 2.24706 | 17 | 3.15771 |
| 3 | 30 | 170 | 170 | 0 | 0 | 2.57059 | 17 | 3.03703 |
| 3 | 31 | 167 | 167 | 0 | 0 | 2.46108 | 21 | 2.64911 |
| 3 | 32 | 169 | 169 | 0 | 0 | 2.66272 | 17 | 2.32274 |
| 3 | 33 | 165 | 165 | 0 | 0 | 2.69697 | 18 | 2.8567 |
| 3 | 34 | 165 | 165 | 0 | 0 | 3.12727 | 20 | 2.97544 |

| Node | Session | Entered | Total | Lost | MinDelay | AvgDelay | MaxDelay | SD |
|---|---|---|---|---|---|---|---|---|
| 3 | 35 | 171 | 171 | 0 | 0 | 3.05263 | 20 | 2.51731 |
| 3 | 36 | 166 | 166 | 0 | 0 | 2.92771 | 21 | 2.54103 |
| 3 | 37 | 169 | 169 | 0 | 0 | 3.59172 | 23 | 3.05078 |
| 3 | 38 | 171 | 171 | 0 | 0 | 3.73684 | 16 | 2.74172 |
| 3 | 39 | 164 | 164 | 0 | 0 | 3.93902 | 14 | 2.11162 |
| 3 | 40 | 164 | 164 | 0 | 0 | 4.03659 | 19 | 2.85221 |
| 3 | 41 | 160 | 160 | 0 | 0 | 3.78125 | 17 | 2.26354 |
| 3 | 42 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 43 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 44 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 45 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 46 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 47 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 48 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 49 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 50 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 51 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 52 | 8 | 8 | 0 | 0 | 0.125 | 1 | 0.133631 |

# Appendix B
# Software Simulation Results – Contd.

**Fixed-sized packets**
Constant source traffic pdf
135 100
Least best-effort traffic pdf
600 20
800 20
1000 20
1200 20
1400 20
Output link rate = 50 bytes/ms
**WFQ scheduler**
Total traffic load at node 1 = 30.09 %
Total traffic load at node 2 = 59.61 %
Total traffic load at node 3 = 92.94 %
**Node Parameters**

| Node | Session | Entered | Total | Lost | MinDelay | AvgDelay | MaxDelay | SD |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2999 | 2999 | 0 | 0 | 0 | 0 | 0 |
| 1 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 13 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 14 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 15 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 17 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 18 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 19 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 20 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 21 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 22 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 23 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 25 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 26 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Node | Session | Entered | Total | Lost | MinDelay | AvgDelay | MaxDelay | SD |
|---|---|---|---|---|---|---|---|---|
| 1 | 27 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 28 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 29 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 30 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 31 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 32 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 34 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 35 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 36 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 37 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 38 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 39 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 40 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 41 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 42 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 43 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 44 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 45 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 46 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 47 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 48 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 49 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 50 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 51 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 52 | 10 | 10 | 0 | 1 | 1 | 1 | 0 |
| 2 | 1 | 2999 | 2999 | 0 | 0 | 0.1994 | 4 | 0.530483 |
| 2 | 2 | 112 | 112 | 0 | 0 | 1.07143 | 13 | 3.03349 |
| 2 | 3 | 109 | 109 | 0 | 0 | 0.825688 | 11 | 1.81478 |
| 2 | 4 | 113 | 113 | 0 | 0 | 0.858407 | 12 | 1.84133 |
| 2 | 5 | 113 | 113 | 0 | 0 | 1.23894 | 12 | 1.92388 |
| 2 | 6 | 110 | 110 | 0 | 0 | 1.74545 | 12 | 2.90345 |
| 2 | 7 | 112 | 112 | 0 | 0 | 1.76786 | 12 | 2.80277 |
| 2 | 8 | 108 | 108 | 0 | 0 | 2.30556 | 13 | 3.35851 |
| 2 | 9 | 111 | 111 | 0 | 0 | 1.83784 | 12 | 2.25445 |
| 2 | 10 | 111 | 111 | 0 | 0 | 2.04505 | 14 | 2.81855 |
| 2 | 11 | 108 | 108 | 0 | 0 | 2.40741 | 14 | 2.70353 |
| 2 | 12 | 111 | 111 | 0 | 0 | 2.26126 | 14 | 2.54414 |
| 2 | 13 | 112 | 112 | 0 | 0 | 2.60714 | 14 | 3.01551 |
| 2 | 14 | 110 | 110 | 0 | 0 | 2.45455 | 13 | 2.98495 |
| 2 | 15 | 111 | 111 | 0 | 0 | 2.75676 | 13 | 2.82534 |
| 2 | 16 | 110 | 110 | 0 | 0 | 2.90909 | 14 | 3.1732 |
| 2 | 17 | 109 | 109 | 0 | 0 | 3 | 21 | 3.51847 |
| 2 | 18 | 109 | 109 | 0 | 0 | 3.25688 | 21 | 3.21159 |
| 2 | 19 | 109 | 109 | 0 | 0 | 3.6055 | 22 | 3.70335 |
| 2 | 20 | 113 | 113 | 0 | 0 | 3.53982 | 15 | 3.22566 |
| 2 | 21 | 111 | 111 | 0 | 0 | 3.63063 | 21 | 3.28951 |

| Node | Session | Entered | Total | Lost | MinDelay | AvgDelay | MaxDelay | SD |
|---|---|---|---|---|---|---|---|---|
| 2 | 22 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 23 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 25 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 26 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 27 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 28 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 29 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 30 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 31 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 32 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 34 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 35 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 36 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 37 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 38 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 39 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 40 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 41 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 42 | 74 | 74 | 0 | 0 | 1.28378 | 11 | 1.87745 |
| 2 | 43 | 74 | 74 | 0 | 1 | 2.5 | 12 | 2.08878 |
| 2 | 44 | 74 | 74 | 0 | 2 | 4.12162 | 13 | 2.80467 |
| 2 | 45 | 74 | 74 | 0 | 3 | 6.78378 | 14 | 2.99277 |
| 2 | 46 | 74 | 74 | 0 | 4 | 8.27027 | 15 | 3.41383 |
| 2 | 47 | 74 | 74 | 0 | 5 | 9.43243 | 16 | 3.29883 |
| 2 | 48 | 74 | 74 | 0 | 6 | 10.4324 | 17 | 3.32415 |
| 2 | 49 | 74 | 74 | 0 | 7 | 11.4324 | 18 | 3.35883 |
| 2 | 50 | 74 | 74 | 0 | 8 | 12.5135 | 19 | 3.80446 |
| 2 | 51 | 74 | 74 | 0 | 9 | 13.7568 | 20 | 3.7641 |
| 2 | 52 | 10 | 10 | 0 | 0 | 1.2 | 3 | 0.918937 |
| 3 | 1 | 2999 | 2999 | 0 | 0 | 3.88663 | 16 | 3.25629 |
| 3 | 2 | 112 | 112 | 0 | 1 | 5.66964 | 13 | 2.74054 |
| 3 | 3 | 109 | 109 | 0 | 0 | 5.77064 | 13 | 2.98996 |
| 3 | 4 | 113 | 113 | 0 | 1 | 5.87611 | 14 | 3.03929 |
| 3 | 5 | 113 | 113 | 0 | 0 | 5.9115 | 14 | 2.96006 |
| 3 | 6 | 110 | 110 | 0 | 1 | 5.90909 | 16 | 3.0318 |
| 3 | 7 | 112 | 112 | 0 | 1 | 5.97321 | 14 | 2.85073 |
| 3 | 8 | 108 | 108 | 0 | 0 | 6.25 | 14 | 3.04215 |
| 3 | 9 | 111 | 111 | 0 | 0 | 5.94595 | 16 | 3.24976 |
| 3 | 10 | 111 | 111 | 0 | 2 | 6.21622 | 14 | 2.88391 |
| 3 | 11 | 108 | 108 | 0 | 1 | 6.12037 | 14 | 3.25852 |
| 3 | 12 | 111 | 111 | 0 | 1 | 6.15315 | 16 | 3.21186 |
| 3 | 13 | 112 | 112 | 0 | 0 | 5.78571 | 16 | 3.32418 |
| 3 | 14 | 110 | 110 | 0 | 1 | 6.49091 | 16 | 3.39088 |
| 3 | 15 | 111 | 111 | 0 | 1 | 5.72072 | 16 | 3.01324 |
| 3 | 16 | 110 | 110 | 0 | 1 | 5.88182 | 16 | 3.11922 |

| Node | Session | Entered | Total | Lost | MinDelay | AvgDelay | MaxDelay | SD |
|---|---|---|---|---|---|---|---|---|
| 3 | 17 | 109 | 109 | 0 | 1 | 6.09174 | 14 | 3.11887 |
| 3 | 18 | 109 | 109 | 0 | 1 | 6.36697 | 16 | 3.36135 |
| 3 | 19 | 109 | 109 | 0 | 1 | 6.30275 | 14 | 3.11779 |
| 3 | 20 | 113 | 113 | 0 | 0 | 5.81416 | 14 | 3.1257 |
| 3 | 21 | 111 | 111 | 0 | 1 | 6.21622 | 13 | 3.02366 |
| 3 | 22 | 172 | 172 | 0 | 0 | 2.27326 | 13 | 2.87168 |
| 3 | 23 | 171 | 171 | 0 | 0 | 2.5614 | 13 | 2.91457 |
| 3 | 24 | 164 | 164 | 0 | 0 | 2.81707 | 11 | 2.82029 |
| 3 | 25 | 162 | 162 | 0 | 0 | 3.24691 | 12 | 2.88509 |
| 3 | 26 | 167 | 167 | 0 | 0 | 3.0479 | 12 | 2.80019 |
| 3 | 27 | 168 | 168 | 0 | 0 | 3.30357 | 13 | 2.95629 |
| 3 | 28 | 164 | 164 | 0 | 0 | 3.5122 | 14 | 2.94848 |
| 3 | 29 | 166 | 166 | 0 | 0 | 3.80723 | 13 | 2.94089 |
| 3 | 30 | 170 | 170 | 0 | 0 | 3.53529 | 11 | 2.91555 |
| 3 | 31 | 170 | 170 | 0 | 0 | 3.77059 | 12 | 2.91129 |
| 3 | 32 | 166 | 166 | 0 | 0 | 3.6747 | 13 | 2.63684 |
| 3 | 33 | 167 | 167 | 0 | 0 | 4.29341 | 13 | 2.79282 |
| 3 | 34 | 169 | 169 | 0 | 0 | 4.53846 | 14 | 2.9794 |
| 3 | 35 | 160 | 160 | 0 | 0 | 4.48125 | 14 | 2.92352 |
| 3 | 36 | 166 | 166 | 0 | 0 | 4.66867 | 15 | 2.74021 |
| 3 | 37 | 166 | 166 | 0 | 0 | 5 | 15 | 3.05703 |
| 3 | 38 | 161 | 161 | 0 | 0 | 5.1118 | 15 | 2.97832 |
| 3 | 39 | 170 | 170 | 0 | 0 | 5.17647 | 15 | 2.85177 |
| 3 | 40 | 171 | 171 | 0 | 0 | 5.50292 | 13 | 3.01085 |
| 3 | 41 | 163 | 163 | 0 | 0 | 5.23926 | 13 | 2.69752 |
| 3 | 42 | 74 | 74 | 0 | 0 | 3.64865 | 9 | 2.1004 |
| 3 | 43 | 74 | 74 | 0 | 0 | 3.66216 | 9 | 2.15989 |
| 3 | 44 | 74 | 74 | 0 | 0 | 3.90541 | 9 | 2.34181 |
| 3 | 45 | 74 | 74 | 0 | 0 | 4.7973 | 10 | 2.33484 |
| 3 | 46 | 74 | 74 | 0 | 0 | 5.02703 | 10 | 2.34098 |
| 3 | 47 | 74 | 74 | 0 | 1 | 5.74324 | 10 | 1.99838 |
| 3 | 48 | 74 | 74 | 0 | 1 | 5.85135 | 10 | 1.97265 |
| 3 | 49 | 74 | 74 | 0 | 1 | 6.14865 | 10 | 2.05158 |
| 3 | 50 | 74 | 74 | 0 | 1 | 6.48649 | 11 | 2.20959 |
| 3 | 51 | 74 | 74 | 0 | 2 | 7.63514 | 14 | 2.50918 |
| 3 | 52 | 10 | 10 | 0 | 2 | 6.9 | 14 | 3.61632 |

**$WF^2Q+$ scheduler**
Total traffic load at node 1 = 30.09 %
Total traffic load at node 2 = 59.61 %
Total traffic load at node 3 = 92.94 %
**Node Parameters**

| Node | Session | Entered | Total | Lost | MinDelay | AvgDelay | MaxDelay | SD |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2999 | 2999 | 0 | 0 | 0 | 0 | 0 |
| 1 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 13 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 14 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 15 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 17 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 18 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 19 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 20 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 21 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 22 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 23 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 25 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 26 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 27 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 28 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 29 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 30 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 31 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 32 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 34 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 35 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 36 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 37 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 38 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 39 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 40 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 41 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Node | Session | Entered | Total | Lost | MinDelay | AvgDelay | MaxDelay | SD |
|---|---|---|---|---|---|---|---|---|
| 1 | 42 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 43 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 44 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 45 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 46 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 47 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 48 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 49 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 50 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 51 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 52 | 10 | 10 | 0 | 1 | 1 | 1 | 0 |
| 2 | 1 | 2999 | 2999 | 0 | 0 | 0.733911 | 9 | 1.50556 |
| 2 | 2 | 112 | 112 | 0 | 0 | 0.776786 | 11 | 2.52094 |
| 2 | 3 | 109 | 109 | 0 | 0 | 0.559633 | 10 | 1.39729 |
| 2 | 4 | 113 | 113 | 0 | 0 | 0.584071 | 11 | 1.61315 |
| 2 | 5 | 113 | 113 | 0 | 0 | 0.911504 | 11 | 1.71941 |
| 2 | 6 | 110 | 110 | 0 | 0 | 1.26364 | 11 | 2.14038 |
| 2 | 7 | 112 | 112 | 0 | 0 | 1.47321 | 12 | 2.6027 |
| 2 | 8 | 108 | 108 | 0 | 0 | 1.84259 | 13 | 2.84872 |
| 2 | 9 | 111 | 111 | 0 | 0 | 1.54054 | 11 | 2.2715 |
| 2 | 10 | 111 | 111 | 0 | 0 | 1.73874 | 14 | 2.71344 |
| 2 | 11 | 108 | 108 | 0 | 0 | 1.91667 | 13 | 2.45758 |
| 2 | 12 | 111 | 111 | 0 | 0 | 1.96396 | 11 | 2.30541 |
| 2 | 13 | 112 | 112 | 0 | 0 | 2.11607 | 13 | 2.48452 |
| 2 | 14 | 110 | 110 | 0 | 0 | 1.98182 | 12 | 2.40589 |
| 2 | 15 | 111 | 111 | 0 | 0 | 2.31532 | 14 | 2.36624 |
| 2 | 16 | 110 | 110 | 0 | 0 | 2.45455 | 12 | 2.5251 |
| 2 | 17 | 109 | 109 | 0 | 0 | 2.7156 | 18 | 3.25957 |
| 2 | 18 | 109 | 109 | 0 | 0 | 2.66055 | 14 | 2.35241 |
| 2 | 19 | 109 | 109 | 0 | 0 | 3.00917 | 16 | 2.86141 |
| 2 | 20 | 113 | 113 | 0 | 0 | 3.00885 | 13 | 2.66135 |
| 2 | 21 | 111 | 111 | 0 | 0 | 3.04505 | 14 | 2.53576 |
| 2 | 22 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 23 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 25 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 26 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 27 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 28 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 29 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 30 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 31 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 32 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 34 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 35 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 36 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Node | Session | Entered | Total | Lost | MinDelay | AvgDelay | MaxDelay | SD |
|---|---|---|---|---|---|---|---|---|
| 2 | 37 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 38 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 39 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 40 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 41 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 42 | 74 | 74 | 0 | 0 | 1.14865 | 7 | 1.38759 |
| 2 | 43 | 74 | 74 | 0 | 1 | 2.37838 | 9 | 1.49081 |
| 2 | 44 | 74 | 74 | 0 | 2 | 3.75676 | 11 | 1.3698 |
| 2 | 45 | 74 | 74 | 0 | 3 | 4.89189 | 13 | 1.52506 |
| 2 | 46 | 74 | 74 | 0 | 4 | 6.2973 | 15 | 2.16396 |
| 2 | 47 | 74 | 74 | 0 | 5 | 7.75676 | 16 | 2.30038 |
| 2 | 48 | 74 | 74 | 0 | 6 | 9.21622 | 17 | 2.56367 |
| 2 | 49 | 74 | 74 | 0 | 7 | 10.6892 | 18 | 2.89713 |
| 2 | 50 | 74 | 74 | 0 | 8 | 11.8784 | 19 | 3.46199 |
| 2 | 51 | 74 | 74 | 0 | 9 | 13.2973 | 20 | 3.62297 |
| 2 | 52 | 10 | 10 | 0 | 0 | 0.2 | 1 | 0.421637 |
| 3 | 1 | 2999 | 2999 | 0 | 0 | 2.8923 | 14 | 3.07643 |
| 3 | 2 | 112 | 112 | 0 | 1 | 5.83929 | 24 | 4.30546 |
| 3 | 3 | 109 | 109 | 0 | 0 | 6.14679 | 21 | 4.53681 |
| 3 | 4 | 113 | 113 | 0 | 1 | 6.11504 | 28 | 4.77799 |
| 3 | 5 | 113 | 113 | 0 | 0 | 6.68142 | 20 | 4.87819 |
| 3 | 6 | 110 | 110 | 0 | 1 | 6.85455 | 24 | 5.04315 |
| 3 | 7 | 112 | 112 | 0 | 1 | 6.64286 | 23 | 4.77903 |
| 3 | 8 | 108 | 108 | 0 | 0 | 7.46296 | 24 | 6.09663 |
| 3 | 9 | 111 | 111 | 0 | 0 | 7.10811 | 24 | 5.74847 |
| 3 | 10 | 111 | 111 | 0 | 1 | 7.40541 | 24 | 5.23361 |
| 3 | 11 | 108 | 108 | 0 | 1 | 7.14815 | 22 | 5.26004 |
| 3 | 12 | 111 | 111 | 0 | 1 | 7.37838 | 26 | 5.7114 |
| 3 | 13 | 112 | 112 | 0 | 0 | 6.84821 | 24 | 5.83619 |
| 3 | 14 | 110 | 110 | 0 | 1 | 7.15455 | 23 | 5.59591 |
| 3 | 15 | 111 | 111 | 0 | 1 | 7.14414 | 24 | 5.44359 |
| 3 | 16 | 110 | 110 | 0 | 1 | 6.41818 | 22 | 4.89534 |
| 3 | 17 | 109 | 109 | 0 | 1 | 7.01835 | 24 | 5.45121 |
| 3 | 18 | 109 | 109 | 0 | 1 | 7.73394 | 25 | 5.52108 |
| 3 | 19 | 109 | 109 | 0 | 1 | 7.50459 | 23 | 5.40515 |
| 3 | 20 | 113 | 113 | 0 | 0 | 7.37168 | 24 | 5.65299 |
| 3 | 21 | 111 | 111 | 0 | 1 | 7.36036 | 22 | 5.28235 |
| 3 | 22 | 172 | 172 | 0 | 0 | 2.44186 | 21 | 5.16308 |
| 3 | 23 | 171 | 171 | 0 | 0 | 1.89474 | 21 | 4.12746 |
| 3 | 24 | 164 | 164 | 0 | 0 | 2.15244 | 20 | 4.01774 |
| 3 | 25 | 162 | 162 | 0 | 0 | 2.48148 | 25 | 4.32704 |
| 3 | 26 | 167 | 167 | 0 | 0 | 3.36527 | 28 | 5.36876 |
| 3 | 27 | 168 | 168 | 0 | 0 | 3.29762 | 30 | 4.80464 |
| 3 | 28 | 164 | 164 | 0 | 0 | 3.14024 | 23 | 4.56555 |
| 3 | 29 | 166 | 166 | 0 | 0 | 3.3012 | 28 | 4.97518 |
| 3 | 30 | 170 | 170 | 0 | 0 | 3.54118 | 30 | 5.09004 |
| 3 | 31 | 170 | 170 | 0 | 0 | 3.84118 | 28 | 5.08982 |

| Node | Session | Entered | Total | Lost | MinDelay | AvgDelay | MaxDelay | SD |
|---|---|---|---|---|---|---|---|---|
| 3 | 32 | 166 | 166 | 0 | 0 | 3.31325 | 27 | 4.28155 |
| 3 | 33 | 167 | 167 | 0 | 0 | 4.97006 | 26 | 5.85709 |
| 3 | 34 | 169 | 169 | 0 | 0 | 4.26035 | 28 | 4.74241 |
| 3 | 35 | 160 | 160 | 0 | 0 | 4.125 | 33 | 5.05442 |
| 3 | 36 | 166 | 166 | 0 | 0 | 4.1988 | 23 | 4.50396 |
| 3 | 37 | 166 | 166 | 0 | 0 | 4.21687 | 26 | 4.59385 |
| 3 | 38 | 161 | 161 | 0 | 0 | 4.41615 | 28 | 4.58047 |
| 3 | 39 | 170 | 170 | 0 | 0 | 5.56471 | 31 | 5.54948 |
| 3 | 40 | 171 | 171 | 0 | 0 | 5 | 20 | 4.49182 |
| 3 | 41 | 163 | 163 | 0 | 0 | 4.93252 | 24 | 4.62732 |
| 3 | 42 | 74 | 74 | 0 | 0 | 4.09459 | 14 | 3.30441 |
| 3 | 43 | 74 | 74 | 0 | 0 | 4.74324 | 17 | 3.8537 |
| 3 | 44 | 74 | 74 | 0 | 0 | 5.22973 | 16 | 4.00187 |
| 3 | 45 | 74 | 74 | 0 | 0 | 6.2973 | 17 | 4.27714 |
| 3 | 46 | 74 | 74 | 0 | 0 | 7.21622 | 17 | 3.90189 |
| 3 | 47 | 74 | 74 | 0 | 1 | 8.14865 | 19 | 4.34544 |
| 3 | 48 | 74 | 74 | 0 | 1 | 8.63513 | 19 | 4.42815 |
| 3 | 49 | 74 | 74 | 0 | 1 | 9.7027 | 21 | 4.99956 |
| 3 | 50 | 74 | 74 | 0 | 1 | 10.8649 | 21 | 5.03799 |
| 3 | 51 | 74 | 74 | 0 | 2 | 12.3378 | 22 | 5.09658 |
| 3 | 52 | 10 | 10 | 0 | 0 | 0.2 | 1 | 0.333333 |

# Appendix C
# Software Simulation Results – Contd.

**Variable-sized packets**
44 60
250 15
552 10
1500 10
1000 5
Without constant source
Least best-effort traffic pdf
600 20
800 20
1000 20
1200 20
1400 20
Output link rate = 320 bytes/ms
**WFQ scheduler**
Total traffic load at node 1 = 30.2627 %
Total traffic load at node 2 = 52.9301 %
Total traffic load at node 3 = 87.8166 %
**Node Parameters**

| Node | Session | Entered | Total | Lost | MinDelay | AvgDelay | MaxDelay | SD |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2999 | 2999 | 0 | 0 | 1.6939 | 18 | 1.90682 |
| 1 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 13 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 14 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 15 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 17 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 18 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 19 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 20 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 21 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Node | Session | Entered | Total | Lost | MinDelay | AvgDelay | MaxDelay | SD |
|---|---|---|---|---|---|---|---|---|
| 1 | 22 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 23 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 25 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 26 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 27 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 28 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 29 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 30 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 31 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 32 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 34 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 35 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 36 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 37 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 38 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 39 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 40 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 41 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 42 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 43 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 44 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 45 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 46 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 47 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 48 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 49 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 50 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 51 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 52 | 10 | 10 | 0 | 0 | 5.2 | 44 | 9.18322 |
| 2 | 1 | 2999 | 2999 | 0 | 0 | 2.14538 | 10 | 2.09739 |
| 2 | 2 | 107 | 107 | 0 | 0 | 2.07477 | 17 | 2.89577 |
| 2 | 3 | 112 | 112 | 0 | 0 | 2.98214 | 23 | 3.90481 |
| 2 | 4 | 114 | 114 | 0 | 0 | 2.53509 | 12 | 3.133 |
| 2 | 5 | 108 | 108 | 0 | 0 | 2.03704 | 17 | 2.45746 |
| 2 | 6 | 111 | 111 | 0 | 0 | 2.04505 | 17 | 2.82486 |
| 2 | 7 | 113 | 113 | 0 | 0 | 2.11504 | 18 | 2.62104 |
| 2 | 8 | 111 | 111 | 0 | 0 | 2.40541 | 9 | 2.51593 |
| 2 | 9 | 110 | 110 | 0 | 0 | 2.29091 | 24 | 3.28692 |
| 2 | 10 | 112 | 112 | 0 | 0 | 2.8125 | 14 | 3.34701 |
| 2 | 11 | 109 | 109 | 0 | 0 | 2.98165 | 15 | 3.5065 |
| 2 | 12 | 110 | 110 | 0 | 0 | 3.36364 | 11 | 3.78843 |
| 2 | 13 | 111 | 111 | 0 | 0 | 2.16216 | 18 | 2.68804 |
| 2 | 14 | 114 | 114 | 0 | 0 | 2.34211 | 18 | 2.94615 |
| 2 | 15 | 110 | 110 | 0 | 0 | 2.36364 | 20 | 2.70097 |
| 2 | 16 | 107 | 107 | 0 | 0 | 2.81308 | 25 | 3.68932 |

| Node | Session | Entered | Total | Lost | MinDelay | AvgDelay | MaxDelay | SD |
|---|---|---|---|---|---|---|---|---|
| 2 | 17 | 112 | 112 | 0 | 0 | 2.45536 | 42 | 4.6749 |
| 2 | 18 | 111 | 111 | 0 | 0 | 2.44144 | 20 | 3.03934 |
| 2 | 19 | 113 | 113 | 0 | 0 | 2.50442 | 19 | 3.07248 |
| 2 | 20 | 109 | 109 | 0 | 0 | 2.14679 | 17 | 2.50326 |
| 2 | 21 | 110 | 110 | 0 | 0 | 2.23636 | 12 | 2.337 |
| 2 | 22 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 23 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 25 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 26 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 27 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 28 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 29 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 30 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 31 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 32 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 34 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 35 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 36 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 37 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 38 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 39 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 40 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 41 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 42 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 43 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 44 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 45 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 46 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 47 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 48 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 49 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 50 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 51 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 52 | 10 | 10 | 0 | 0 | 8.9 | 105 | 18.80432 |
| 3 | 1 | 2999 | 2999 | 0 | 0 | 3.33144 | 10 | 2.45307 |
| 3 | 2 | 107 | 107 | 0 | 0 | 7.20561 | 103 | 22.5675 |
| 3 | 3 | 112 | 112 | 0 | 0 | 8.94643 | 83 | 13.09741 |
| 3 | 4 | 114 | 114 | 0 | 0 | 7.01754 | 31 | 14.84314 |
| 3 | 5 | 108 | 108 | 0 | 0 | 9.87037 | 128 | 24.0418 |
| 3 | 6 | 111 | 111 | 0 | 0 | 8.4955 | 92 | 20.535 |
| 3 | 7 | 113 | 113 | 0 | 0 | 8.58407 | 82 | 20.1207 |
| 3 | 8 | 111 | 111 | 0 | 0 | 9.56757 | 99 | 21.3877 |
| 3 | 9 | 110 | 110 | 0 | 0 | 7.45455 | 47 | 11.24267 |
| 3 | 10 | 112 | 112 | 0 | 0 | 8.55357 | 66 | 15.04435 |
| 3 | 11 | 109 | 109 | 0 | 0 | 7.6055 | 45 | 16.09005 |

| Node | Session | Entered | Total | Lost | MinDelay | AvgDelay | MaxDelay | SD |
|---|---|---|---|---|---|---|---|---|
| 3 | 12 | 110 | 110 | 0 | 0 | 10.30909 | 85 | 29.69664 |
| 3 | 13 | 111 | 111 | 0 | 0 | 9.89189 | 55 | 17.77367 |
| 3 | 14 | 114 | 114 | 0 | 0 | 7.30702 | 82 | 13.1062 |
| 3 | 15 | 110 | 110 | 0 | 0 | 10.3 | 98 | 22.7672 |
| 3 | 16 | 107 | 107 | 0 | 0 | 13.68224 | 106 | 29.9756 |
| 3 | 17 | 112 | 112 | 0 | 0 | 12.38393 | 113 | 26.3317 |
| 3 | 18 | 111 | 111 | 0 | 0 | 9.81982 | 74 | 19.82557 |
| 3 | 19 | 113 | 113 | 0 | 0 | 9.45133 | 89 | 19.95564 |
| 3 | 20 | 109 | 109 | 0 | 0 | 10.80734 | 84 | 22.3993 |
| 3 | 21 | 110 | 110 | 0 | 0 | 11.23636 | 87 | 21.8242 |
| 3 | 22 | 169 | 169 | 0 | 0 | 8.46154 | 39 | 24.17393 |
| 3 | 23 | 166 | 166 | 0 | 0 | 8.89157 | 51 | 24.91196 |
| 3 | 24 | 162 | 162 | 0 | 0 | 8.41358 | 25 | 23.63157 |
| 3 | 25 | 162 | 162 | 0 | 0 | 8.46296 | 24 | 23.5397 |
| 3 | 26 | 164 | 164 | 0 | 0 | 9.17683 | 74 | 37.49651 |
| 3 | 27 | 167 | 167 | 0 | 0 | 8.15569 | 34 | 23.96197 |
| 3 | 28 | 170 | 170 | 0 | 0 | 9 | 27 | 24.42157 |
| 3 | 29 | 172 | 172 | 0 | 0 | 10.08721 | 61 | 25.62825 |
| 3 | 30 | 160 | 160 | 0 | 0 | 10.19375 | 40 | 24.98456 |
| 3 | 31 | 172 | 172 | 0 | 0 | 9.37209 | 51 | 24.63648 |
| 3 | 32 | 166 | 166 | 0 | 0 | 10.9759 | 50 | 26.18712 |
| 3 | 33 | 164 | 164 | 0 | 0 | 9.60976 | 27 | 23.65585 |
| 3 | 34 | 166 | 166 | 0 | 0 | 12.53614 | 41 | 25.68524 |
| 3 | 35 | 171 | 171 | 0 | 0 | 11.80117 | 58 | 26.74036 |
| 3 | 36 | 167 | 167 | 0 | 0 | 10.34132 | 32 | 25.53984 |
| 3 | 37 | 163 | 163 | 0 | 0 | 9.92025 | 106 | 22.2443 |
| 3 | 38 | 163 | 163 | 0 | 0 | 13.49693 | 54 | 26.97948 |
| 3 | 39 | 165 | 165 | 0 | 0 | 12.41212 | 69 | 26.5502 |
| 3 | 40 | 170 | 170 | 0 | 0 | 13.74706 | 92 | 21.9519 |
| 3 | 41 | 165 | 165 | 0 | 0 | 16.29091 | 84 | 20.3812 |
| 3 | 42 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 43 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 44 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 45 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 46 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 47 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 48 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 49 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 50 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 51 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 52 | 10 | 10 | 0 | 0 | 9.8 | 107 | 19.57762 |

**$WF^2Q+$ scheduler**
Total traffic load at node 1 = 30.2627 %
Total traffic load at node 2 = 52.9301 %
Total traffic load at node 3 = 87.8166 %
**Node Parameters**

| Node | Session | Entered | Total | Lost | MinDelay | AvgDelay | MaxDelay | SD |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2999 | 2999 | 0 | 0 | 1.69423 | 18 | 1.90693 |
| 1 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 13 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 14 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 15 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 17 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 18 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 19 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 20 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 21 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 22 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 23 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 25 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 26 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 27 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 28 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 29 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 30 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 31 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 32 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 34 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 35 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 36 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 37 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 38 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 39 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 40 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 41 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Node | Session | Entered | Total | Lost | MinDelay | AvgDelay | MaxDelay | SD |
|---|---|---|---|---|---|---|---|---|
| 1 | 42 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 43 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 44 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 45 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 46 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 47 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 48 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 49 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 50 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 51 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 52 | 10 | 10 | 0 | 0 | 1.2 | 4 | 1.18322 |
| 2 | 1 | 2999 | 2999 | 0 | 0 | 2.70257 | 22 | 3.01453 |
| 2 | 2 | 107 | 107 | 0 | 0 | 1.71963 | 16 | 2.16613 |
| 2 | 3 | 112 | 112 | 0 | 0 | 1.73214 | 15 | 2.22424 |
| 2 | 4 | 114 | 114 | 0 | 0 | 1.36842 | 11 | 1.81054 |
| 2 | 5 | 108 | 108 | 0 | 0 | 1.91667 | 19 | 2.56033 |
| 2 | 6 | 111 | 111 | 0 | 0 | 1.91892 | 17 | 2.51282 |
| 2 | 7 | 113 | 113 | 0 | 0 | 1.88496 | 14 | 2.18772 |
| 2 | 8 | 111 | 111 | 0 | 0 | 1.54054 | 18 | 2.12207 |
| 2 | 9 | 110 | 110 | 0 | 0 | 1.94545 | 9 | 2.04258 |
| 2 | 10 | 112 | 112 | 0 | 0 | 1.78571 | 23 | 2.78902 |
| 2 | 11 | 109 | 109 | 0 | 0 | 2.0367 | 11 | 2.23571 |
| 2 | 12 | 110 | 110 | 0 | 0 | 1.33636 | 8 | 1.44088 |
| 2 | 13 | 111 | 111 | 0 | 0 | 1.86486 | 15 | 2.1377 |
| 2 | 14 | 114 | 114 | 0 | 0 | 2.16667 | 17 | 2.2808 |
| 2 | 15 | 110 | 110 | 0 | 0 | 2.43636 | 21 | 2.99351 |
| 2 | 16 | 107 | 107 | 0 | 0 | 2.34579 | 18 | 2.56944 |
| 2 | 17 | 112 | 112 | 0 | 0 | 2.07143 | 19 | 2.51175 |
| 2 | 18 | 111 | 111 | 0 | 0 | 2.45946 | 20 | 3.00994 |
| 2 | 19 | 113 | 113 | 0 | 0 | 2.27434 | 19 | 2.73158 |
| 2 | 20 | 109 | 109 | 0 | 0 | 1.93578 | 16 | 2.15653 |
| 2 | 21 | 110 | 110 | 0 | 0 | 2.38182 | 16 | 2.63451 |
| 2 | 22 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 23 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 25 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 26 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 27 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 28 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 29 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 30 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 31 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 32 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 34 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 35 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 36 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Node | Session | Entered | Total | Lost | MinDelay | AvgDelay | MaxDelay | SD |
|---|---|---|---|---|---|---|---|---|
| 2 | 37 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 38 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 39 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 40 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 41 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 42 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 43 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 44 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 45 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 46 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 47 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 48 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 49 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 50 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 51 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 52 | 10 | 10 | 0 | 0 | 1.5 | 4 | 1.50923 |
| 3 | 1 | 2999 | 2999 | 0 | 0 | 8.55218 | 49 | 8.43839 |
| 3 | 2 | 107 | 107 | 0 | 0 | 6.38318 | 58 | 10.3064 |
| 3 | 3 | 112 | 112 | 0 | 0 | 4.72321 | 42 | 6.53506 |
| 3 | 4 | 114 | 114 | 0 | 0 | 6.91228 | 71 | 13.0769 |
| 3 | 5 | 108 | 108 | 0 | 0 | 4.77778 | 81 | 9.09589 |
| 3 | 6 | 111 | 111 | 0 | 0 | 7.25225 | 79 | 13.6515 |
| 3 | 7 | 113 | 113 | 0 | 0 | 8.11504 | 110 | 17.9473 |
| 3 | 8 | 111 | 111 | 0 | 0 | 4.1982 | 71 | 7.31098 |
| 3 | 9 | 110 | 110 | 0 | 0 | 6.11818 | 69 | 10.3279 |
| 3 | 10 | 112 | 112 | 0 | 0 | 5.08036 | 53 | 8.42006 |
| 3 | 11 | 109 | 109 | 0 | 0 | 5.72477 | 53 | 8.99016 |
| 3 | 12 | 110 | 110 | 0 | 0 | 4 | 35 | 4.78817 |
| 3 | 13 | 111 | 111 | 0 | 0 | 6.78378 | 134 | 16.3252 |
| 3 | 14 | 114 | 114 | 0 | 0 | 7.01754 | 65 | 10.7689 |
| 3 | 15 | 110 | 110 | 0 | 0 | 9.5 | 115 | 16.826 |
| 3 | 16 | 107 | 107 | 0 | 0 | 9.72897 | 100 | 17.453 |
| 3 | 17 | 112 | 112 | 0 | 0 | 6.32143 | 119 | 13.7943 |
| 3 | 18 | 111 | 111 | 0 | 0 | 6.62162 | 66 | 11.4764 |
| 3 | 19 | 113 | 113 | 0 | 0 | 8.37168 | 162 | 19.4588 |
| 3 | 20 | 109 | 109 | 0 | 0 | 8.00917 | 85 | 13.9124 |
| 3 | 21 | 110 | 110 | 0 | 0 | 6.83636 | 62 | 12.2137 |
| 3 | 22 | 169 | 169 | 0 | 0 | 7.10059 | 126 | 15.0329 |
| 3 | 23 | 166 | 166 | 0 | 0 | 5.51807 | 85 | 10.7055 |
| 3 | 24 | 162 | 162 | 0 | 0 | 4.87654 | 70 | 8.32431 |
| 3 | 25 | 162 | 162 | 0 | 0 | 7.25926 | 110 | 15.473 |
| 3 | 26 | 164 | 164 | 0 | 0 | 6.41463 | 76 | 13.0345 |
| 3 | 27 | 167 | 167 | 0 | 0 | 5.2994 | 80 | 11.1074 |
| 3 | 28 | 170 | 170 | 0 | 0 | 7.80588 | 128 | 16.9018 |
| 3 | 29 | 172 | 172 | 0 | 0 | 8.45349 | 99 | 14.2672 |
| 3 | 30 | 160 | 160 | 0 | 0 | 9.7125 | 91 | 18.1906 |
| 3 | 31 | 172 | 172 | 0 | 0 | 6.15116 | 72 | 11.4848 |

| Node | Session | Entered | Total | Lost | MinDelay | AvgDelay | MaxDelay | SD |
|---|---|---|---|---|---|---|---|---|
| 3 | 32 | 166 | 166 | 0 | 0 | 5.99398 | 78 | 11.0655 |
| 3 | 33 | 164 | 164 | 0 | 0 | 5.96341 | 100 | 12.0498 |
| 3 | 34 | 166 | 166 | 0 | 0 | 6.56627 | 76 | 10.7271 |
| 3 | 35 | 171 | 171 | 0 | 0 | 7.77778 | 116 | 14.3513 |
| 3 | 36 | 167 | 167 | 0 | 0 | 7.04192 | 83 | 12.1068 |
| 3 | 37 | 163 | 163 | 0 | 0 | 6.37423 | 114 | 12.6772 |
| 3 | 38 | 163 | 163 | 0 | 0 | 5.25767 | 86 | 9.66897 |
| 3 | 39 | 165 | 165 | 0 | 0 | 7.93939 | 123 | 15.8981 |
| 3 | 40 | 170 | 170 | 0 | 0 | 5.92353 | 76 | 10.4426 |
| 3 | 41 | 165 | 165 | 0 | 0 | 8.92121 | 119 | 16.3221 |
| 3 | 42 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 43 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 44 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 45 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 46 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 47 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 48 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 49 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 50 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 51 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 52 | 10 | 10 | 0 | 0 | 1.9 | 6 | 1.71594 |

# Appendix D
# Software Simulation Results – Contd.

**Variable-sized packets**
44 60
250 15
552 10
1500 10
1000 5
Constant source traffic pdf
135 100
Least best-effort traffic pdf
600 20
800 20
1000 20
1200 20
1400 20
Output link rate = 320 bytes/ms
**WFQ scheduler**
Total traffic load at node 1 = 28.8986 %
Total traffic load at node 2 = 58.5099 %
Total traffic load at node 3 = 91.3006 %
Node Parameters

| Node | Session | Entered | Total | Lost | MinDelay | AvgDelay | MaxDelay | SD |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2999 | 2999 | 0 | 0 | 1.52518 | 13 | 1.74551 |
| 1 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 13 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 14 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 15 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 17 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 18 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 19 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 20 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Node | Session | Entered | Total | Lost | MinDelay | AvgDelay | MaxDelay | SD |
|---|---|---|---|---|---|---|---|---|
| 1 | 21 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 22 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 23 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 25 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 26 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 27 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 28 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 29 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 30 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 31 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 32 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 34 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 35 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 36 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 37 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 38 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 39 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 40 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 41 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 42 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 43 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 44 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 45 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 46 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 47 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 48 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 49 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 50 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 51 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 52 | 10 | 10 | 0 | 0 | 1.5 | 3 | 1.2693 |
| 2 | 1 | 2999 | 2999 | 0 | 0 | 2.1884 | 9 | 2.05701 |
| 2 | 2 | 111 | 111 | 0 | 0 | 11.87387 | 15 | 12.7173 |
| 2 | 3 | 108 | 108 | 0 | 0 | 12.25926 | 27 | 13.5295 |
| 2 | 4 | 111 | 111 | 0 | 0 | 11.83784 | 19 | 12.3508 |
| 2 | 5 | 111 | 111 | 0 | 0 | 12.07207 | 34 | 17.8209 |
| 2 | 6 | 109 | 109 | 0 | 0 | 12.81651 | 24 | 13.043 |
| 2 | 7 | 109 | 109 | 0 | 0 | 12.21101 | 33 | 13.545 |
| 2 | 8 | 108 | 108 | 0 | 0 | 11.74074 | 16 | 12.2531 |
| 2 | 9 | 111 | 111 | 0 | 0 | 12.28829 | 23 | 12.8555 |
| 2 | 10 | 110 | 110 | 0 | 0 | 12.05455 | 23 | 12.8334 |
| 2 | 11 | 110 | 110 | 0 | 0 | 12.54545 | 29 | 18.1062 |
| 2 | 12 | 110 | 110 | 0 | 0 | 12.46364 | 27 | 13.7142 |
| 2 | 13 | 109 | 109 | 0 | 0 | 12.24771 | 20 | 12.655 |
| 2 | 14 | 110 | 110 | 0 | 0 | 12.37273 | 22 | 12.7327 |
| 2 | 15 | 110 | 110 | 0 | 0 | 12.37273 | 23 | 13.5038 |

| Node | Session | Entered | Total | Lost | MinDelay | AvgDelay | MaxDelay | SD |
|---|---|---|---|---|---|---|---|---|
| 2 | 16 | 113 | 113 | 0 | 0 | 12.86726 | 38 | 14.8735 |
| 2 | 17 | 108 | 108 | 0 | 0 | 12.10185 | 19 | 12.1823 |
| 2 | 18 | 110 | 110 | 0 | 0 | 12.47273 | 20 | 22.8339 |
| 2 | 19 | 110 | 110 | 0 | 0 | 12.72727 | 29 | 13.2511 |
| 2 | 20 | 109 | 109 | 0 | 0 | 12.77982 | 15 | 12.8755 |
| 2 | 21 | 110 | 110 | 0 | 0 | 13.25455 | 34 | 14.4644 |
| 2 | 22 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 23 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 25 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 26 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 27 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 28 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 29 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 30 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 31 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 32 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 34 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 35 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 36 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 37 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 38 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 39 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 40 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 41 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 42 | 74 | 74 | 0 | 0 | 13.05405 | 49 | 14.5186 |
| 2 | 43 | 74 | 74 | 0 | 0 | 12.47297 | 67 | 13.3666 |
| 2 | 44 | 74 | 74 | 0 | 0 | 13.32432 | 42 | 14.3437 |
| 2 | 45 | 74 | 74 | 0 | 0 | 14.12162 | 60 | 14.8855 |
| 2 | 46 | 74 | 74 | 0 | 1 | 15.71622 | 56 | 16.6245 |
| 2 | 47 | 74 | 74 | 0 | 1 | 16.28378 | 63 | 17.3912 |
| 2 | 48 | 74 | 74 | 0 | 1 | 16.40541 | 46 | 16.6545 |
| 2 | 49 | 74 | 74 | 0 | 1 | 16.02703 | 68 | 20.2915 |
| 2 | 50 | 74 | 74 | 0 | 2 | 26.94595 | 51 | 27.4535 |
| 2 | 51 | 74 | 74 | 0 | 2 | 26.93243 | 58 | 26.5143 |
| 2 | 52 | 10 | 10 | 0 | 0 | 52.7 | 96 | 41.4089 |
| 3 | 1 | 2999 | 2999 | 0 | 0 | 3.24141 | 51 | 2.3661 |
| 3 | 2 | 111 | 111 | 0 | 0 | 29.4955 | 564 | 65.6994 |
| 3 | 3 | 108 | 108 | 0 | 0 | 26.23148 | 156 | 32.8097 |
| 3 | 4 | 111 | 111 | 0 | 0 | 27.38739 | 130 | 55.1591 |
| 3 | 5 | 111 | 111 | 0 | 0 | 35.0991 | 297 | 68.352 |
| 3 | 6 | 109 | 109 | 0 | 0 | 34.6147 | 266 | 53.4093 |
| 3 | 7 | 109 | 109 | 0 | 0 | 29 | 240 | 40.052 |
| 3 | 8 | 108 | 108 | 0 | 0 | 25.5 | 108 | 40.8778 |
| 3 | 9 | 111 | 111 | 0 | 0 | 26.91892 | 237 | 33.1615 |
| 3 | 10 | 110 | 110 | 0 | 0 | 27.78182 | 226 | 38.3022 |

| Node | Session | Entered | Total | Lost | MinDelay | AvgDelay | MaxDelay | SD |
|---|---|---|---|---|---|---|---|---|
| 3 | 11 | 110 | 110 | 0 | 0 | 27.14545 | 274 | 48.7028 |
| 3 | 12 | 110 | 110 | 0 | 0 | 31.0364 | 242 | 54.2969 |
| 3 | 13 | 109 | 109 | 0 | 0 | 29.61468 | 310 | 65.9303 |
| 3 | 14 | 110 | 110 | 0 | 0 | 32.4364 | 273 | 50.2664 |
| 3 | 15 | 110 | 110 | 0 | 0 | 29.30909 | 286 | 64.5187 |
| 3 | 16 | 113 | 113 | 0 | 0 | 33.7522 | 333 | 77.4625 |
| 3 | 17 | 108 | 108 | 0 | 0 | 27.60185 | 202 | 58.7323 |
| 3 | 18 | 110 | 110 | 0 | 0 | 26.68182 | 233 | 86.2476 |
| 3 | 19 | 110 | 110 | 0 | 0 | 27.47273 | 155 | 73.5133 |
| 3 | 20 | 109 | 109 | 0 | 0 | 24.73394 | 93 | 55.9309 |
| 3 | 21 | 110 | 110 | 0 | 0 | 30.8636 | 197 | 65.166 |
| 3 | 22 | 168 | 168 | 0 | 0 | 23.57143 | 103 | 54.4566 |
| 3 | 23 | 164 | 164 | 0 | 0 | 23.59756 | 108 | 45.2281 |
| 3 | 24 | 167 | 167 | 0 | 0 | 24.10778 | 91 | 37.0367 |
| 3 | 25 | 167 | 167 | 0 | 0 | 23.69461 | 82 | 35.056 |
| 3 | 26 | 164 | 164 | 0 | 0 | 23.45732 | 88 | 34.8058 |
| 3 | 27 | 163 | 163 | 0 | 0 | 24.57669 | 90 | 36.3482 |
| 3 | 28 | 161 | 161 | 0 | 0 | 23.43478 | 67 | 33.2837 |
| 3 | 29 | 168 | 168 | 0 | 0 | 24.85119 | 99 | 46.6542 |
| 3 | 30 | 168 | 168 | 0 | 0 | 25.04167 | 83 | 37.4172 |
| 3 | 31 | 170 | 170 | 0 | 0 | 23.62941 | 93 | 34.0552 |
| 3 | 32 | 168 | 168 | 0 | 0 | 24.1131 | 86 | 34.9745 |
| 3 | 33 | 168 | 168 | 0 | 0 | 24.58333 | 97 | 37.0293 |
| 3 | 34 | 170 | 170 | 0 | 0 | 24.81176 | 83 | 45.1272 |
| 3 | 35 | 166 | 166 | 0 | 0 | 24.70482 | 94 | 55.8625 |
| 3 | 36 | 169 | 169 | 0 | 0 | 23.72189 | 91 | 43.9197 |
| 3 | 37 | 165 | 165 | 0 | 0 | 24.05455 | 80 | 44.5252 |
| 3 | 38 | 166 | 166 | 0 | 0 | 24.57831 | 108 | 46.2025 |
| 3 | 39 | 169 | 169 | 0 | 0 | 24.1716 | 103 | 44.4608 |
| 3 | 40 | 168 | 168 | 0 | 0 | 23.49405 | 92 | 43.9568 |
| 3 | 41 | 169 | 169 | 0 | 0 | 23.99408 | 102 | 55.7228 |
| 3 | 42 | 74 | 74 | 0 | 0 | 28.52703 | 199 | 61.6249 |
| 3 | 43 | 74 | 74 | 0 | 0 | 27.16216 | 164 | 44.2873 |
| 3 | 44 | 74 | 74 | 0 | 0 | 27.18919 | 177 | 46.6672 |
| 3 | 45 | 74 | 74 | 0 | 0 | 30.0676 | 201 | 83.3164 |
| 3 | 46 | 74 | 74 | 0 | 0 | 31.2432 | 232 | 76.0804 |
| 3 | 47 | 74 | 74 | 0 | 0 | 30.027 | 215 | 53.4727 |
| 3 | 48 | 74 | 74 | 0 | 0 | 40 | 364 | 73.3005 |
| 3 | 49 | 74 | 74 | 0 | 0 | 26.2973 | 121 | 42.5952 |
| 3 | 50 | 74 | 74 | 0 | 0 | 32.1216 | 229 | 57.5658 |
| 3 | 51 | 74 | 74 | 0 | 0 | 28.06757 | 127 | 54.6478 |
| 3 | 52 | 10 | 10 | 0 | 0 | 22.4 | 270 | 91.8973 |

**$WF^2Q+$ scheduler**

Total traffic load at node 1 = 28.8986 %

Total traffic load at node 2 = 58.5099 %

Total traffic load at node 3 = 91.3006 %

**Node Parameters**

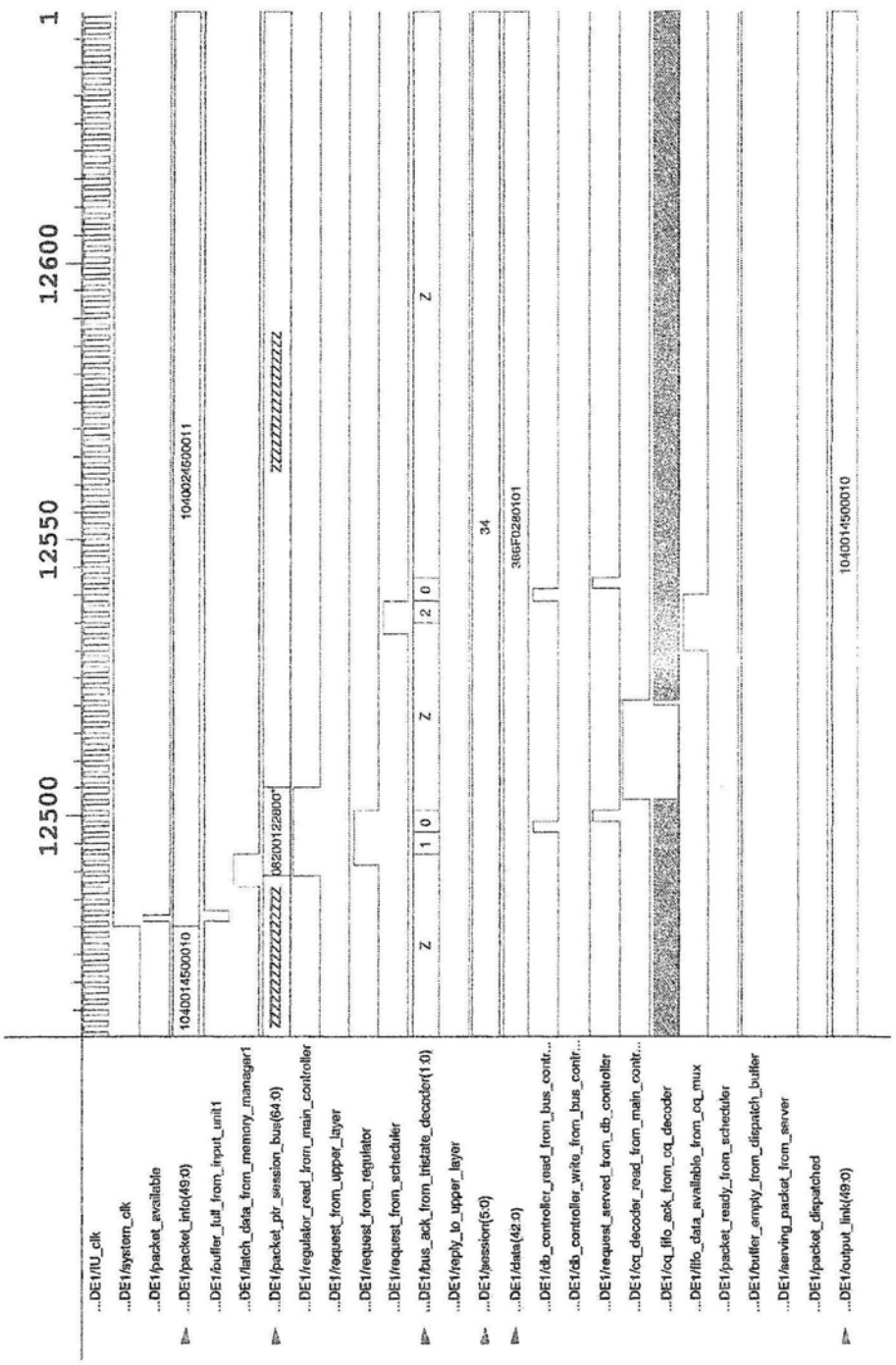
| Node | Session | Entered | Total | Lost | MinDelay | AvgDelay | MaxDelay | SD |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2999 | 2999 | 0 | 0 | 1.52518 | 13 | 1.74551 |
| 1 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 13 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 14 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 15 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 17 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 18 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 19 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 20 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 21 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 22 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 23 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 25 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 26 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 27 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 28 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 29 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 30 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 31 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 32 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 34 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 35 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 36 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 37 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 38 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 39 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 40 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 41 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| Node | Session | Entered | Total | Lost | MinDelay | AvgDelay | MaxDelay | SD |
|---|---|---|---|---|---|---|---|---|
| 1 | 42 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 43 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 44 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 45 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 46 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 47 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 48 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 49 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 50 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 51 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 52 | 10 | 10 | 0 | 0 | 1.4 | 3 | 1.18322 |
| 2 | 1 | 2999 | 2999 | 0 | 0 | 3.20473 | 25 | 3.70701 |
| 2 | 2 | 111 | 111 | 0 | 0 | 1.74775 | 12 | 2.17014 |
| 2 | 3 | 108 | 108 | 0 | 0 | 2.64815 | 27 | 3.79708 |
| 2 | 4 | 111 | 111 | 0 | 0 | 1.88288 | 16 | 2.35145 |
| 2 | 5 | 111 | 111 | 0 | 0 | 1.90991 | 29 | 3.28728 |
| 2 | 6 | 109 | 109 | 0 | 0 | 2.50459 | 24 | 2.90967 |
| 2 | 7 | 109 | 109 | 0 | 0 | 2.04587 | 29 | 3.23039 |
| 2 | 8 | 108 | 108 | 0 | 0 | 2.17593 | 38 | 4.21819 |
| 2 | 9 | 111 | 111 | 0 | 0 | 2.14414 | 22 | 2.71788 |
| 2 | 10 | 110 | 110 | 0 | 0 | 2.35455 | 30 | 3.58149 |
| 2 | 11 | 110 | 110 | 0 | 0 | 2.03636 | 22 | 2.50551 |
| 2 | 12 | 110 | 110 | 0 | 0 | 2.87273 | 42 | 5.08596 |
| 2 | 13 | 109 | 109 | 0 | 0 | 2.12844 | 10 | 2.10316 |
| 2 | 14 | 110 | 110 | 0 | 0 | 2.52727 | 20 | 2.66347 |
| 2 | 15 | 110 | 110 | 0 | 0 | 2.64545 | 42 | 5.15111 |
| 2 | 16 | 113 | 113 | 0 | 0 | 2.58407 | 29 | 3.84379 |
| 2 | 17 | 108 | 108 | 0 | 0 | 2.08333 | 18 | 2.09899 |
| 2 | 18 | 110 | 110 | 0 | 0 | 2.33636 | 16 | 2.40892 |
| 2 | 19 | 110 | 110 | 0 | 0 | 3.25455 | 27 | 4.08177 |
| 2 | 20 | 109 | 109 | 0 | 0 | 2.56881 | 15 | 2.48997 |
| 2 | 21 | 110 | 110 | 0 | 0 | 2.89091 | 23 | 3.23853 |
| 2 | 22 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 23 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 25 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 26 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 27 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 28 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 29 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 30 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 31 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 32 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 34 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 35 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 36 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

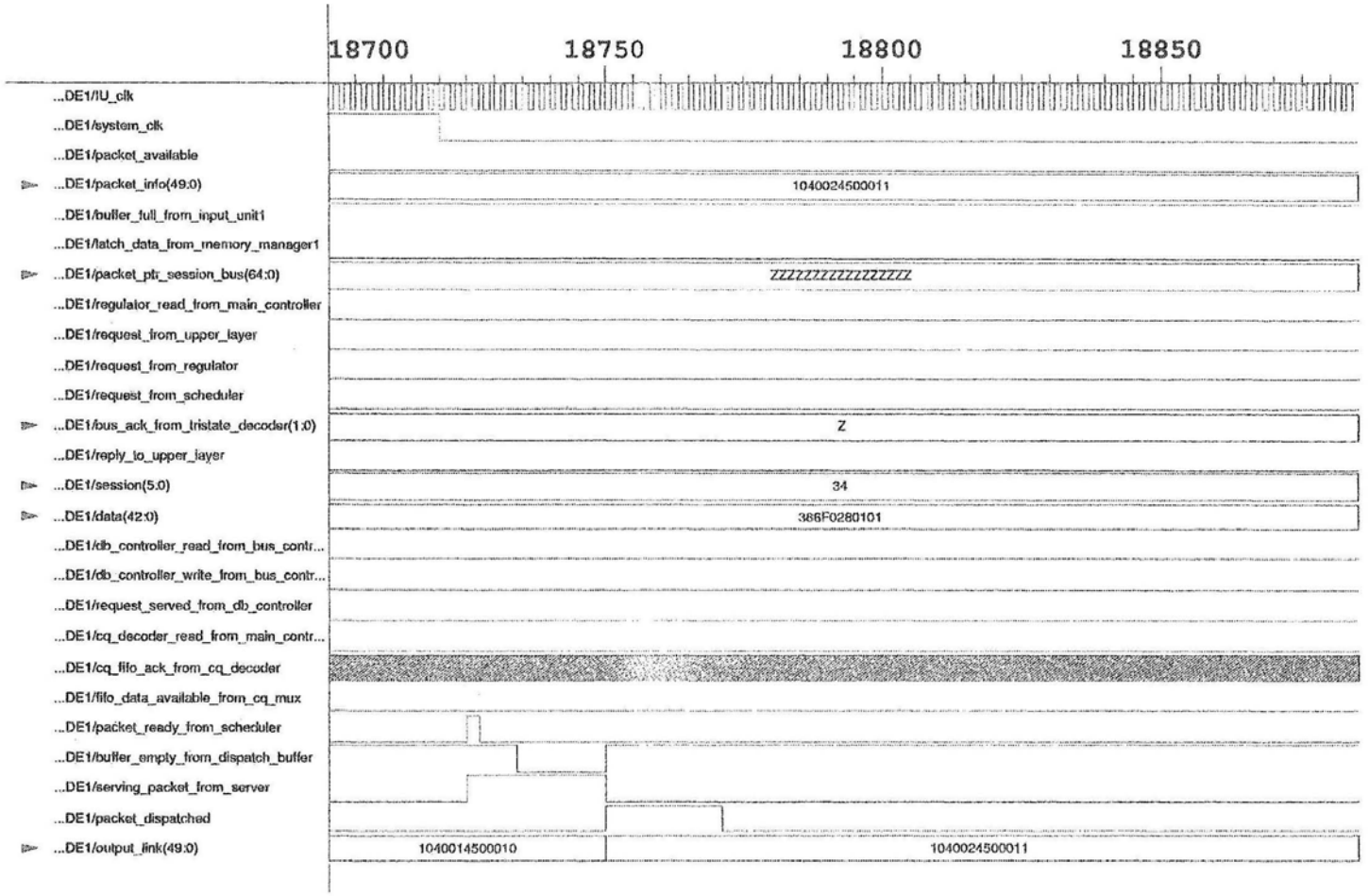
| Node | Session | Entered | Total | Lost | MinDelay | AvgDelay | MaxDelay | SD |
|---|---|---|---|---|---|---|---|---|
| 2 | 37 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 38 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 39 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 40 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 41 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 42 | 74 | 74 | 0 | 0 | 2.72973 | 29 | 4.11861 |
| 2 | 43 | 74 | 74 | 0 | 0 | 2.44595 | 24 | 3.11857 |
| 2 | 44 | 74 | 74 | 0 | 0 | 3 | 22 | 3.28342 |
| 2 | 45 | 74 | 74 | 0 | 0 | 4.02703 | 24 | 4.15107 |
| 2 | 46 | 74 | 74 | 0 | 1 | 5.58108 | 29 | 5.78117 |
| 2 | 47 | 74 | 74 | 0 | 1 | 6.12162 | 32 | 5.95188 |
| 2 | 48 | 74 | 74 | 0 | 1 | 5.86486 | 26 | 5.14082 |
| 2 | 49 | 74 | 74 | 0 | 1 | 5.41892 | 36 | 5.40854 |
| 2 | 50 | 74 | 74 | 0 | 2 | 6.39189 | 33 | 5.90761 |
| 2 | 51 | 74 | 74 | 0 | 2 | 6.60811 | 33 | 6.3344 |
| 2 | 52 | 10 | 10 | 0 | 0 | 2.9 | 6 | 2.01384 |
| 3 | 1 | 2999 | 2999 | 0 | 0 | 9.62287 | 54 | 8.91575 |
| 3 | 2 | 111 | 111 | 0 | 0 | 15.1532 | 240 | 38.8567 |
| 3 | 3 | 108 | 108 | 0 | 0 | 10.0093 | 167 | 21.7944 |
| 3 | 4 | 111 | 111 | 0 | 0 | 10.3514 | 194 | 24.8564 |
| 3 | 5 | 111 | 111 | 0 | 0 | 8.7027 | 152 | 20.5335 |
| 3 | 6 | 109 | 109 | 0 | 0 | 12.6239 | 177 | 24.7063 |
| 3 | 7 | 109 | 109 | 0 | 0 | 14.2844 | 161 | 27.7493 |
| 3 | 8 | 108 | 108 | 0 | 0 | 7.41667 | 85 | 15.2082 |
| 3 | 9 | 111 | 111 | 0 | 0 | 10.1622 | 122 | 19.6204 |
| 3 | 10 | 110 | 110 | 0 | 0 | 14.6273 | 225 | 35.6284 |
| 3 | 11 | 110 | 110 | 0 | 0 | 8.39091 | 164 | 20.536 |
| 3 | 12 | 110 | 110 | 0 | 0 | 10.2091 | 103 | 18.5074 |
| 3 | 13 | 109 | 109 | 0 | 0 | 10.4128 | 184 | 24.2798 |
| 3 | 14 | 110 | 110 | 0 | 0 | 17.4455 | 250 | 42.0692 |
| 3 | 15 | 110 | 110 | 0 | 0 | 14.0455 | 162 | 29.3769 |
| 3 | 16 | 113 | 113 | 0 | 0 | 14.6372 | 157 | 33.21 |
| 3 | 17 | 108 | 108 | 0 | 0 | 6.85185 | 94 | 13.0361 |
| 3 | 18 | 110 | 110 | 0 | 0 | 7.55455 | 79 | 14.0661 |
| 3 | 19 | 110 | 110 | 0 | 0 | 10.7636 | 129 | 19.2144 |
| 3 | 20 | 109 | 109 | 0 | 0 | 6.26606 | 99 | 12.6257 |
| 3 | 21 | 110 | 110 | 0 | 0 | 15.4636 | 190 | 33.8084 |
| 3 | 22 | 168 | 168 | 0 | 0 | 12.1786 | 162 | 27.8515 |
| 3 | 23 | 164 | 164 | 0 | 0 | 7.45732 | 151 | 17.1937 |
| 3 | 24 | 167 | 167 | 0 | 0 | 8.35329 | 104 | 16.127 |
| 3 | 25 | 167 | 167 | 0 | 0 | 11.7126 | 135 | 24.2615 |
| 3 | 26 | 164 | 164 | 0 | 0 | 8.07927 | 110 | 17.0404 |
| 3 | 27 | 163 | 163 | 0 | 0 | 18.4172 | 235 | 43.6005 |
| 3 | 28 | 161 | 161 | 0 | 0 | 14.1988 | 253 | 34.3033 |
| 3 | 29 | 168 | 168 | 0 | 0 | 12.2202 | 124 | 23.0952 |
| 3 | 30 | 168 | 168 | 0 | 0 | 15.8631 | 229 | 33.0788 |
| 3 | 31 | 170 | 170 | 0 | 0 | 10.5706 | 139 | 20.9214 |

| Node | Session | Entered | Total | Lost | MinDelay | AvgDelay | MaxDelay | SD |
|---|---|---|---|---|---|---|---|---|
| 3 | 32 | 168 | 168 | 0 | 0 | 12.744 | 154 | 26.7467 |
| 3 | 33 | 168 | 168 | 0 | 0 | 10.4405 | 136 | 22.3689 |
| 3 | 34 | 170 | 170 | 0 | 0 | 9.80588 | 124 | 18.634 |
| 3 | 35 | 166 | 166 | 0 | 0 | 15.8614 | 170 | 31.5794 |
| 3 | 36 | 169 | 169 | 0 | 0 | 9.28994 | 140 | 19.7114 |
| 3 | 37 | 165 | 165 | 0 | 0 | 10.9939 | 159 | 27.5445 |
| 3 | 38 | 166 | 166 | 0 | 0 | 11.4639 | 141 | 21.7097 |
| 3 | 39 | 169 | 169 | 0 | 0 | 10.2722 | 114 | 18.8775 |
| 3 | 40 | 168 | 168 | 0 | 0 | 9.0119 | 128 | 19.1944 |
| 3 | 41 | 169 | 169 | 0 | 0 | 9.74556 | 106 | 18.0556 |
| 3 | 42 | 74 | 74 | 0 | 0 | 7.86486 | 118 | 17.466 |
| 3 | 43 | 74 | 74 | 0 | 0 | 7.86486 | 118 | 16.898 |
| 3 | 44 | 74 | 74 | 0 | 0 | 5.71622 | 68 | 9.64026 |
| 3 | 45 | 74 | 74 | 0 | 0 | 6.66216 | 68 | 10.0461 |
| 3 | 46 | 74 | 74 | 0 | 0 | 8.66216 | 154 | 20.2655 |
| 3 | 47 | 74 | 74 | 0 | 0 | 8.91892 | 73 | 14.5713 |
| 3 | 48 | 74 | 74 | 0 | 0 | 8.18919 | 104 | 16.6352 |
| 3 | 49 | 74 | 74 | 0 | 0 | 5.67568 | 48 | 8.7381 |
| 3 | 50 | 74 | 74 | 0 | 0 | 11.3378 | 155 | 24.7769 |
| 3 | 51 | 74 | 74 | 0 | 0 | 8.77027 | 115 | 18.611 |
| 3 | 52 | 10 | 10 | 0 | 0 | 2.5 | 8 | 1.90029 |

# Appendix E
# Hardware Simulation Results

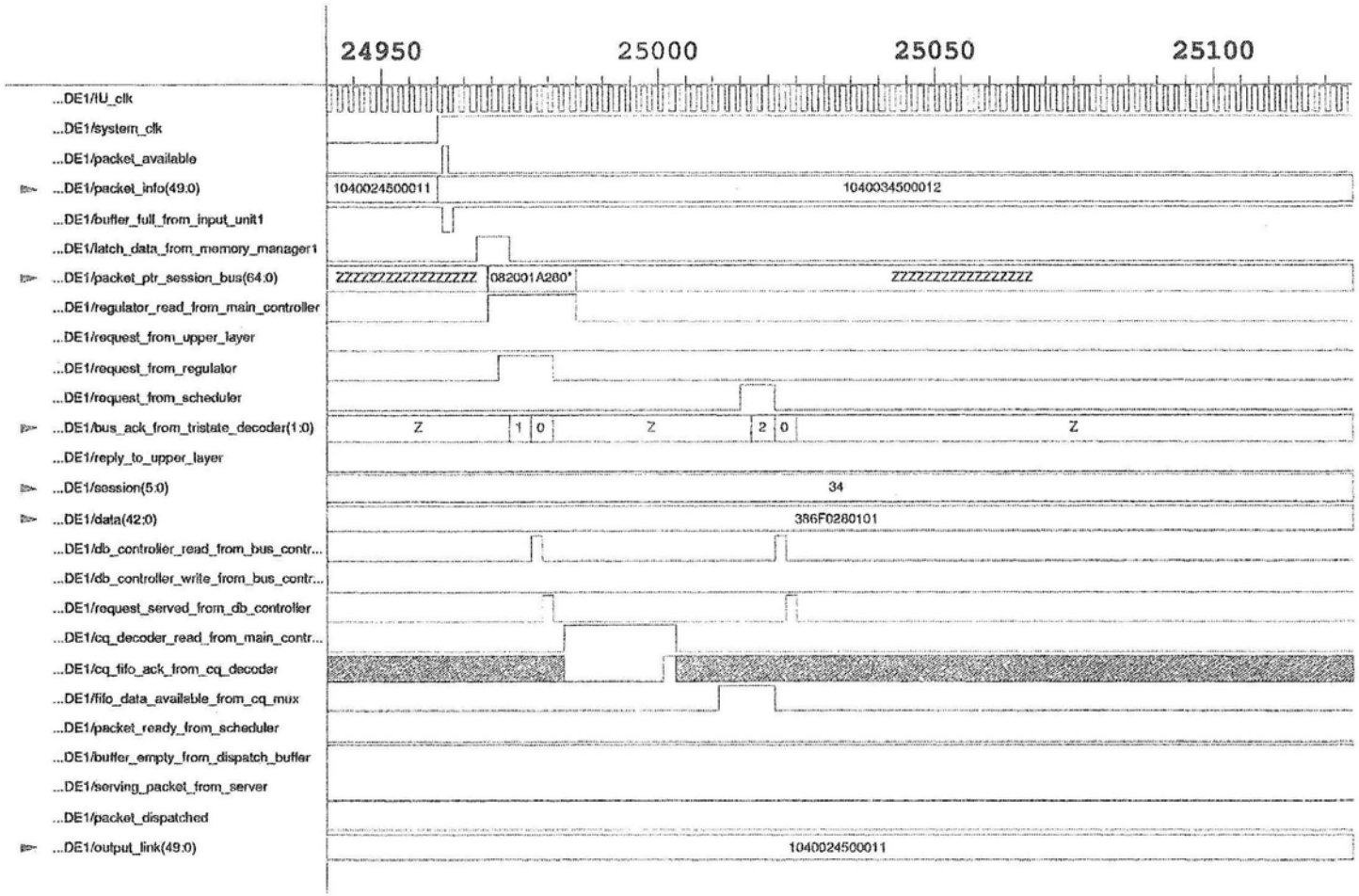



Second packet arrival




s/users.cs.study/mnt/padmini/padmini_code_7/NWC_SCHEDULER_TESTBENCH.cheetah.883
22/5/2003 22:20:9 Page 1,1 of 1,1

Second packet dispatched




s/users.cs.study/mnt/padmini/padmini_code_7/NWC_SCHEDULER_TESTBENCH.cheetah.883
22/5/2003 22:21:5 Page 1,1 of 1,1

Third packet arrival