DEADLINE-ORDERED

PARALLEL ITERATIVE MATCHING

WITH QOS GUARANTEE

BY
LUI HUNG NGAI

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF PHILOSOPHY
IN
INFORMATION ENGINEERING

AUGUST 2000

Acknowledgement

I would like to thank many people for supporting my effort during my M. Phil. study in CUHK. Without them, my M. Phil. research would not be such fruitful and rewarding.

First of all, I would like to express my gratitude towards my supervisor Prof. Tony T. Lee. During these two years, his criticism and advice inspires me. I am grateful to him for being a true teacher to me.

Specifically, I would like to thank Dr. Soung-Yue Liew and Mr. Man C. Chan for their valuable comments and continuous discussions on my research work and thesis contents. I wish to thank Dr. Philip P. T. To, Dr. Cathy W. C. Chan, Mr. Alan Yeung, Mr. Albert T. M. Lau, Mr. Danny, C. S. Yip, Mr. H. P. Sze for various kind of help. They are kind and being good companions to me.

Thanks to my family: Dad, Mon and my younger brother for supporting me during all these years. I love you all.

Finally, I thank God and my Lord Jesus Christ for allowing me remain wholesome and in good health in my life. Without Him, nothing is possible.

I believe that I shall see the goodness of the LORD in the land of the living!

Psalms 27:13(RSV)

摘要

未來，高速分組交換網絡將取代早期的電話線路交換網絡，而成爲新一代的通訊標準中介體。從統計學的角度來說，由於分組交換網絡佔有資源共享的好處，所以她可以擁有更高的線路頻寬的使用率。這也使得我們可以利用她來實現一個不僅能夠傳送資料，同時也能夠提供不同型態如即時多媒體應用等等的服務的電腦網絡。在不久的將來，我們很可能只需要一個積體服務網絡，就可以提供如電話會議，遙距教學，電子郵件，傳真電報以及即時多媒體等等的服務。然而另一方面，線路的傳輸速率以及使用者的人口都有戲劇性的增長。這一切因素將使得如何控制這樣的積體服務網絡成爲一個極富挑戰性的任務。

其中一個近幾年來相當熱門的題材是，如何在網絡中提供不同的連接以公平的服務。爲這，文獻上出現了許多不同的，甚至是相矛盾的方案。非常有趣的是，有許多方案都不約而同的提出了一個非常重要的網絡設計概念，那就是交通流程的演算法。流程演算法主要的功能是決定在交換網絡中，服務來自於不同連接的分組的次序。不同於從前的是，流程演算法不單是要爲各別連接找到一個無沖撞的通道，而且也必須提供有保證的資源分配，以及提供可預測的終端對終端的延遲和延遲變動的上限。

這份論文主要的貢獻是在於，我們提出了一個命名爲期限依序平行反復配對的平行演算法。這個演算法主要應用在輸入端緩沖型交換機，除了讓我們可以找到無沖撞的輸入輸出配對之外，也可以提供公平的服務次序給於所有的資訊流動。我們將展示我們的演算法比起其前身，平行反復配對演算法，更具有數項優點。比如，從模擬實驗中我們可觀察出，分組所經歷的最大延遲變小了。這是因爲每個連接所分配到的服務是與其所預訂的頻寬是相關的。同時，我們的演算法也保護了行爲良好的連接，以避免它們受到行爲不良的連接拖累。最後，我們還提出了一個結合了期限依序平行反復配對演算法以及靜態流程排定的演算法，以使得輸入端緩沖形態交換機能提供使用頻寬的保證。我們也展示了利用這個方案，交換機將可以在達致高使用率的同時，提供個別資訊流動所需求的服務品質。

Abstract

High-speed packet networks are becoming a standard medium of communication, replacing earlier schemes based on the telephone-type circuit-switched networks. Due to statistical multiplexing gain, packet-switching networks enjoy a high utilization of the link bandwidth. This enables us to build computer networks that not only transmit data, but also provide many different types of services like real-time multimedia applications. In the coming future, it is likely that a single integrated-services network will be used by different applications such as teleconferencing, distance education,e-mail, FAX and real-time multimedia. On the other hand, increase in link speed and number of users are dramatic. All these factors make the control of such integrated-services network a challenging task.

Specifically, providing fair services among different connections in networks have been a hot topic during the last several years. Many different, and sometimes conflicting approaches have been proposed in the literature. It is interesting to observe that many approaches require a correct design of a key network component—*traffic scheduling algorithm.* The main function of a scheduling algorithm is to determine the service order of packets that offered by an application in a switching network. Unlike the past ones, scheduling algorithms now

not only need to find a conflict-free of network connections, but also need to provide guaranteed allocation of network resources and give predictable bounds on end-to-end delays and/or delay variation.

In this thesis, our major contribution is a parallel algorithm, called DeadLine-ordered Parallel Iterative Matching (DLPIM), for finding conflict-free bipartite match and providing fair service order among flows in input-buffered switch. We show that our algorithm has several advantages over its previous version, the Parallel Iterative Matching (PIM). From our simulations, we illustrate that the maximum delay experienced by packets under our algorithm is smaller. The service delivered to each connections follows its bandwidth reservation. Our algorithm provides protection for well-behaved connections against mis-behaved flows. Finally, we suggest a scheme that is an incorporation of DLPIM with static scheduling algorithm to provide bandwidth guarantee in input-buffered switch. We demonstrate how our suggested scheme benefits the switch by guaranteeing individual flows their specified Quality-of-service (QoS) and at the same time, achieving a high system utilization.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the last decade, the developments of the telecommunication network is very fast. Because of the technological advanced, modern broadband digital networks can deliver large amount of data in much faster way. Switching speed and bandwidth capacity of telecommunication network have grown rapidly, following a linear or an exponentially curve. In the future, most of the communication networks will be operated over high-speed mediums and be run from several hundred Mbps to tens of Gbps. With such high speed and capacity, different kinds of applications can be supported. The introduction of Broadband Integrated Services Digital Network (B-ISDN) suggests that future networks not only need to support traditional telephone traffic and data communication, but also provide new types of multimedia services like Video-On-Demand(VOD), Video-Conference(VCF). In order to satisfy these services' requirements, maximizing bandwidth utilization and providing performance guarantees are two important issues need to achieve in designing a network.

When a network is set up, the bandwidth always exists no matter it is used

or not. The utilization of bandwidth benefits network providers in term of economic sense. On the other hand, users always want to have services with quality. Instead of just providing a low average delay, end-to-end delay bounds and delay jitter bounds become more and more important in determining whether or not customers are getting their QoS requirements [1] [2]. Asynchronous Transfer Mode(ATM), having the feature of traffic multiplexing and providing diverse QoS guarantee, becomes the core technique in building B-ISDN. The key to the success of ATM network deployment lies in the design of large-scale ATM switches to cope with heterogeneous bandwidth constraints and diverse QoS conditions. However, ATM network is connection oriented; almost all of the control algorithms at the network level, including call admission, access control, flow control, and congestion control proposed in the literature, are based on this fundamental assumption. However, the packet switching proposed for many ATM switching architectures is actually performing datagram routing within the switch fabric. This inconsistency between networking and the underlying switching may lead to inconsequential network layer protocols. In order to provide a coherent network management paradigm, service order of packets becomes important. A good scheduling algorithm shall service packets in a way that high throughput and bandwidth guarantee can be achieved. Different algorithms have been developed to reach this goal. In this thesis, we will address this issue and propose a scheduling algorithm to solve the problem. The remainder of the thesis will discuss it in details.

1.1 Thesis Overview

Here is the organization of this thesis:

- Chapter 2 puts our work in context by describing background knowledge and related work. We will discuss the scheduling problems in a switch and their solutions. Specifically, we will focus on fair queueing concept in output-buffered switch and the Parallel Iterative Matching algorithm (PIM) employed in input-buffered switch. These two scheduling algorithms are fundamental building blocks in our work.

- Chapter 3 presents the details of our algorithm - Deadline-ordered Parallel Iterative Matching (DLPIM). Also, in this chapter, we are going to show the improvement in delay bound of DLPIM over PIM through conducting simulations.

- Chapter 4 shows how to incorporate DLPIM with a static algorithm to provide bandwidth guarantee to connections.

- Chapter 5 concludes the findings in this thesis.

Chapter 2

Background & Related work

2.1 Scheduling problem in ATM switch

ATM has been accepted by TU-T and will be likely to become the universal platform for developing new switching techniques for B-ISDN in the future. A number of ATM switch architectures have been proposed in the literature, and many of them have been implemented as commercial products. Throughout the thesis, we will place our focus on the type of switches that is internally non-blocking. For internally non-blocking switching networks, since arrival of packets at the input ports is not cooperative, output contention will be resulted if more than one input port has packets destined for the same output. In each time slot, only one packet is allowed to transmit, those who lose in the contention resolution will either be discarded (loss system) or be stored in the buffer (lossless system). In a lossless system, we can place buffer either in the input side (input-buffered switch) or the output side (output-buffered switch). In order to schedule the transmission of packets stored, we need a mechanism to arrange the service order

of packets. Many service disciplines have been proposed for both input-buffered switch ([3] [4] [5] [6]) and output-buffered switch ([7] [8] [9]). In this chapter, we will briefly describe the concept of scheduling algorithm used in output-buffered switch and input-buffered switch.

2.2 Traffic Scheduling in output-buffered switch

In an $N \times N$ output-buffered switch as shown in fig 2.1, packets destined for the same output are transmitted from their source ports to their destination ports simultaneously. This is done by providing multi-path inside the switch or operating the internally nonblocking switch at a speed that is a multiple of the external link rate. Since packets arrived simultaneously are stored at the output side of the switch, the scheduling task of output-buffered switch becomes the problem of allocating the whole outgoing capacity to all backlogged sessions in a fair and effective manner. Figure 2.2 shows a scheduling scenario of an output port. It is clear that the output port act like a $N \times 1$ server which serves a total of N connections. In each time slot, the switch will select a packet among all the backlogged ones and transmit it to the outgoing link.

To tackle the scheduling task of output-buffered switch, many algorithms have been proposed. Some of them are work-conserving, while the others are non-work-conserving. With a working-conserving discipline, a server is never idle when there is at least one packet waiting for transmission. With a non-work-conserving discipline, each packet is assigned, either explicitly or implicitly, an eligibility time. Even when the server is idle, if no packets are eligible, none will be transmitted. In particular, we will study the principles of Fluid

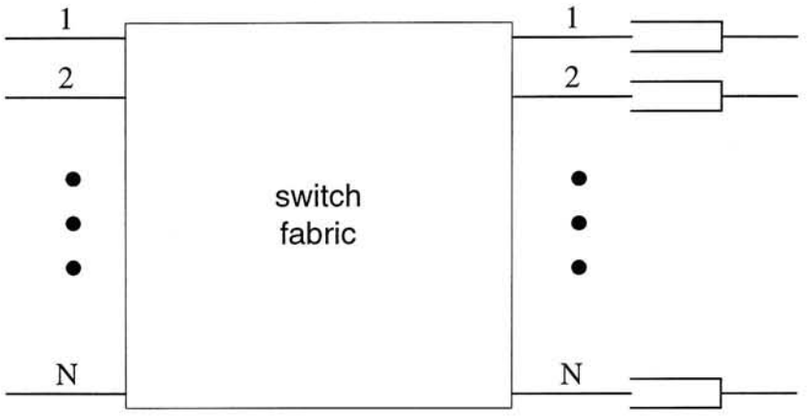

Figure 2.1: A $N \times N$ output-buffered switch

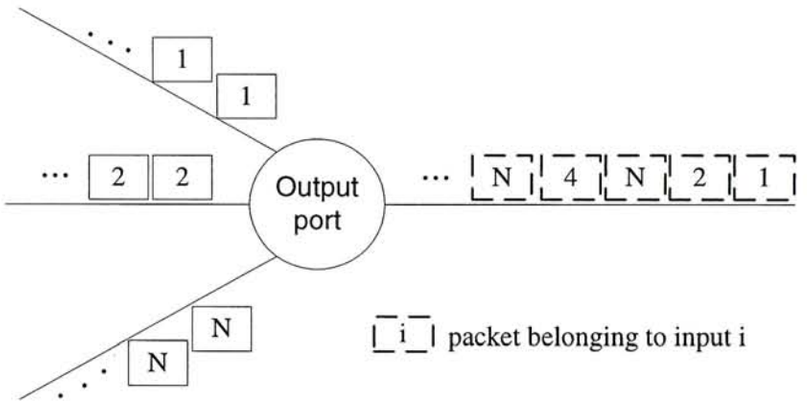

Figure 2.2: Scheduling of packet in the output port

Flow Queueing (FFQ) scheduling algorithms and briefly describe the related algorithms.

It is well-known that Fluid Flow Queueing (FFQ) or Generalized Process sharing (GPS) is fair in single server system because it allocates the whole capacity of the link to all backlogged connections. FFQ serves each connection proportional to its minimum rate (bandwidth) requirements. This algorithm is based on the ideal fluid-flow model, where we assume the server can serve all backlogged connections instantaneously and simultaneously. FFQ is a general

form of the head-of-line processor sharing service discipline (HOL-PS) [18]. With HOL-PS, there is a separate FIFO queue for each connection sharing the same link. During any time interval when there are exactly N non-empty queues, the server serves the N packets at the head of the queues simultaneously, each at a rate of one N^{th} of the link speed. Unlike HOL-PS that serves each link at the same rate, FFQ allows different connections to have different service shares. A FFQ is characterized by N positive real numbers, $\phi_1, \phi_2, \ldots, \phi_N$, each corresponding to the bandwidth requirement of one queue. At any time τ, the service rate for a non-empty queue i is exactly

$$\frac{\phi_i}{\sum_{j \in B(\tau)} \phi_j} C$$

where $B(\tau)$: the set of non-empty queues and C is the link speed.
Therefore, FFQ serves the non-empty queues in proportion to their service shares. FFQ is impractical as it assumes that the server can serve all connections with non-empty queues simultaneously and that the traffic is infinitely divisible.

Although FFQ service principle is perfectly fair, this ideal fluid-flow model is not practical to implement. However, we can emulate the FFQ server and then schedule the backlogged packets in accordance with the packet behaviour of the emulated FFQ server. These emulation algorithms, also known as Deadline-ordered service disciplines, like Virtual Clock [9], Fair Queueing (FQ) [7] and its Weighted version (WFQ) also called Packetized Generalized Processor Sharing (PGPS) [10], Self-Clocked Fair Queueing (SCFQ)[8] all use this similar sorted priority queueing mechanism to tackle the scheduling task. Under this mechanism, a state variable associated with each connection to keep track and enforce

its traffic. When a packet is arrived, the variable is updated according to the followings:

- The bandwidth reservation made for the connection during setup time.
- Traffic arrival history of this connection and/or other connection during transfer.

By considering the above, each packet is stamped with the state variable for the connection to which it belongs. This stamped value is used as a priority index value which determine the order of service. Different algorithms have different ways of calculating or updating the virtual time variable. Some of them may need extra information, for example, WF^2Q transmits packets according to their time stamp and the eligibility of packets.

The reason of assigning time-stamp to packets in backlogged connections is that we can easily serve packets without violating the fairness bound of other backlogged connections. Different service disciplines use different names for the state variable. In virtual Clock, the state variable is called auxiliary Virtual Clock ($auxVC$) while in WFQ, WF^2Q and SCFQ, we refer it as Virtual Finish time (F). In all cases, $auxVC$ and F serve as a priority indices of packets. The calculation of the state variable for different algorithms is similar. We will describe them (Virtual Clock, WFQ and SCFQ) later in this chapter.

WFQ and WF^2Q

WFQ is a packet policy that try to approximate the same Fluid Fair Queueing (FFQ) or Generalized Processor Sharing (GPS) policy. In a more realistic system, the server can only serves one connection in a particular time. Before

serving another packet, the transmission of previous one must be completed. Different disciplines use different approaches to approximate FFQ service in a packet system. The most well-known is the Weighted Fair Queueing (WFQ) [7], also known as Packetized Generalized Processor Sharing (PGPS) [10]. In WFQ, the scheduler assigns the departure time of a packet in the emulated FFQ server at the time stamp of that packet. When the server is ready to transmit a packet, it will choose the first packet that would be completed its service in the corresponding FFQ system. In other words, the packet with the smallest time stamp will be transmitted. Thus, the service order of all the packets can be easily found.

The state variable, the virtual finish time (F), in WFQ and WF^{2}Q is calculated according to the following:

$$F_j^k \leftarrow max\{V(a_j^k), F_j^{k-1}\} + \frac{L_j^k}{\phi_j} \tag{2.1}$$

where j and k correspond to the connection number and packet number;
$V(t)$ is the system virtual time (a measurement of system progress) at time t;
a_j^k is the arrival of k^{th} packet on connection j;
L_j^k is the packet length in bits;
ϕ_j is the bandwidth allocation of connection j.
The virtual time function $V(.)$ during any busy period $[t_1, t_2]$ is defined as follows:

$$\begin{aligned} V(t_1) &= 0 \\ \frac{\partial V(\tau)}{\partial \tau} &= \frac{1}{\sum_{j \in B_{FFQ}(\tau)} \phi_j} \quad \forall t_1 \leq \tau \leq t_2 \end{aligned}$$

with $B_{FFQ}(\tau)$ is the set of backlogged connections at time τ under the reference

FFQ system. Since in FFQ, when a connection j is backlogged at time τ, the server will serve the connection at a rate of $\frac{\partial V(\tau)}{\partial \tau}\phi_j C$.

Although WFQ provides certain approximation of FFQ emulation, there are some drawbacks of the algorithm. In some situations, WFQ does not perform very well (figure 2.3). In order to achieve more accurate emulation of FFQ, WF^{2}Q is introduced. Under WFQ scheme, at time τ, the system will transmit the first packet that would complete service in FFQ system if no additional packets were to arrive after time τ. While WFQ considers only the finish time of the packets, WF^{2}Q uses both the start times and finish times of packets in the FFQ system. In WF^{2}Q, we only choose to transmit packet that is in the set of packets that had started (and possibly finished) receiving service in the FFQ system at time τ, and select the one that would complete service first in the corresponding FFQ system. We will now show an example to illustrate the unfairness of WFQ and the improvement of WF^{2}Q over WFQ.

Figure 2.3 shows the difference between WFQ and WF^{2}Q. Here we assume the packet size is 1 and the link speed is also 1. The bandwidth allocation of each connection is shown in the figure. Connection 1 uses 50% of the link capacity, while connections 2 to 11 uses only 5%. In this example, connection 1 sends 11 back-to-back packets at time 0 while the other 10 connections send only one packet at time 0. If the server is FFQ, the first 10 packets in connection 1 will take 2 time units each to be transmitted. The last packet in connection 1 will take only 1 time units. For packets of other connections, each will take 20 time units. For the same arrival pattern, WFQ and WF^{2}Q will give a different service order. The 10^{th} packet of connection 1 has the FFQ finish time equal to 20, the same as that of packets of the other connection. Under WFQ, as the

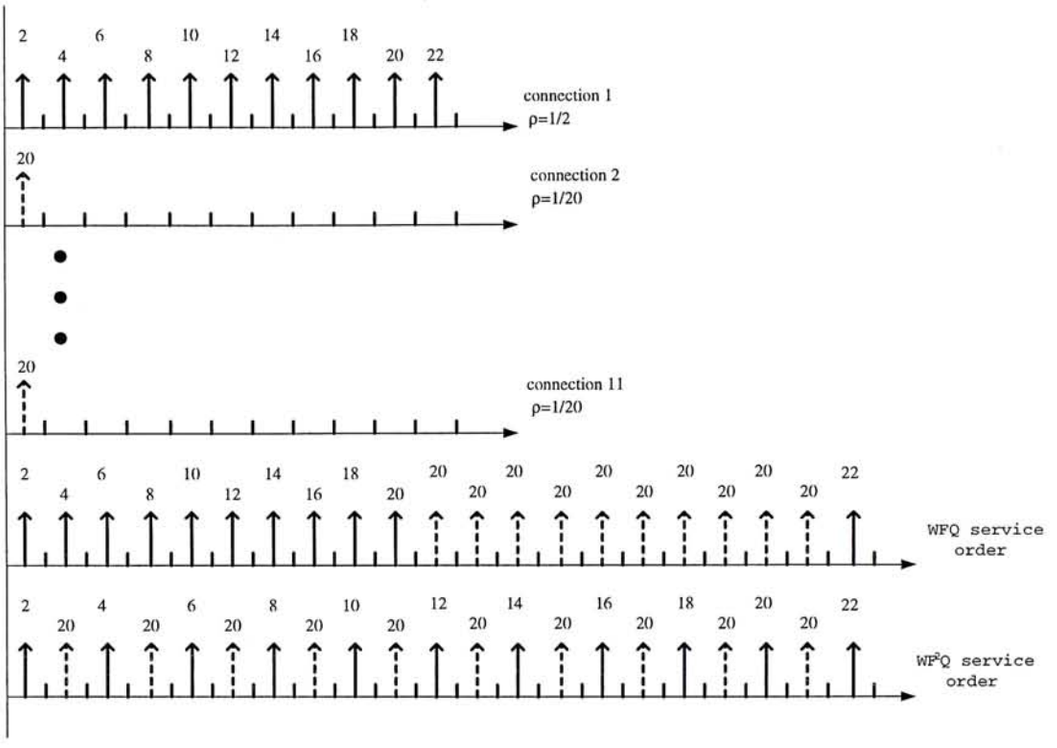

Figure 2.3: WFQ and WF^{2}Q

FFQ finish times of the first 10 packets on connection 1 are smaller than packets on the other connection, the server will serve 10 packets on connection 1 before serving packets from other connections. However, under WF^{2}Q, at time 0, all packets at the head of the queues have started service in the emulated FFQ system. Since the first packet on connection 1 has the smallest finish time, it is selected and transmitted. At time 1, the second packet arrives in connection 1. Although the packet of connection 1 has the smallest finish time, the packet has not started its service until time 2. So, it is not eligible for transmission at time 1. So, the server will then select a packet from the set of packets that have already started their service. In this case, this means the packets of connections $2, \ldots, 11$. Since all of them have the same finish time, the server will transmit

the packet of connection 2 assuming the tie-breaking rule of giving highest priority to the connection with smallest number. In time 2, the packet in connection 1 becomes eligible and is transmitted because of its smallest finish time. The rest of the service order is shown in figure 2.3.

Virtual Clock

Although WFQ and WF^{2}Q give an accurate approximation of a fair FFQ service, these emulations of a reference FFQ server is computationally expensive. So, other service schemes are suggested. Among them, the most simplest one is Virtual Clock [9]. This discipline aims at emulating the Time division Multiplexing (TDM) system. Each packet is allocated a virtual transmission time, which is the time at which the packet would have been transmitted were the server actually doing TDM. Packets are transmitted in the increasing order of virtual transmission times.

The Virtual Clock scheduler uses real time function to approach the virtual time function. That is, the scheduler assigns

$$V^{vc}(t) = t, \text{for } t \geq 0$$

The state variable of Virtual Clock is called auxiliary Virtual Clock ($auxVC$). The k^{th} packet from the backlogged session j is assigned a time stamp with a value equal to $auxVC$. The $auxVC$ is calculated using:

$$auxVC_j^k \leftarrow max\{a_j^k, auxVC_j^k\} + \frac{L_j^k}{\phi_j} \tag{2.2}$$

Figure 2.4 gives a simple example to illustrate how Virtual Clock works. In the example, there are three connections sharing the same output link. All three

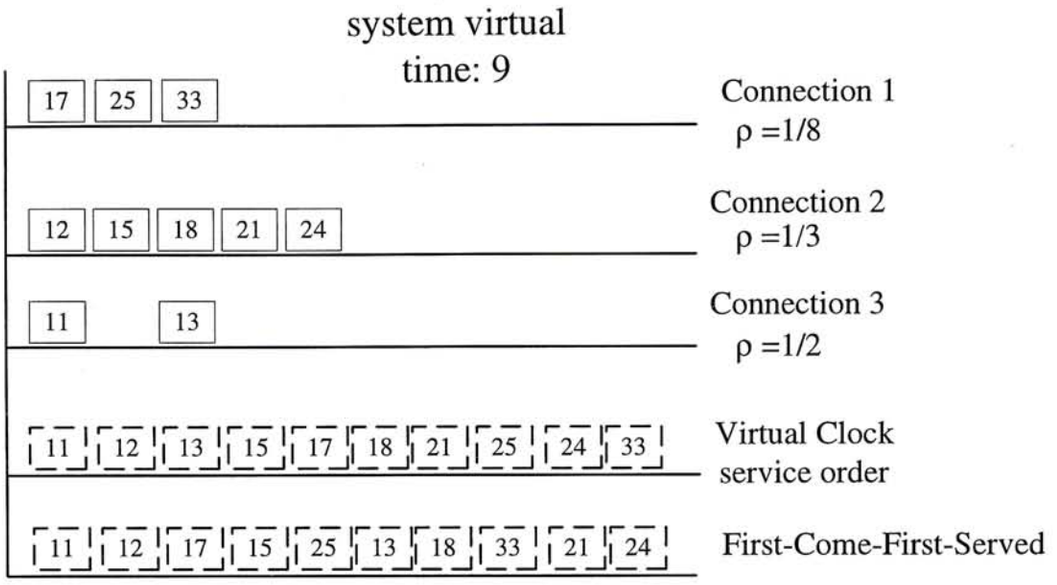

Figure 2.4: Virtual Clock scheduling

connections specify their traffic characteristics and reserve resources accordingly. Connections 1 has an average packet inter-arrival time of 8 time units ($\rho = \frac{1}{8}$), connections 2 and 3 have an average packet inter-arrival time of 3 and 2 time units respectively. In this example, the first packet of connection 1 is calculated by $max\,(9,0)+\left(\frac{1}{8}\right)^{-1}$. Here we assume all the packets are of the same size, and transmission of a packet takes one time unit. Hence, the bandwidth distribution of connections 1, 2 and 3 will be 12.5%, 33.33% and 50%. From figure 2.4, we can see the transmission pattern. Obviously, connections 1 and 2 send packets at higher rates than reserved, while connection 3 sends packets according to the specified traffic pattern. The final line in the figure shows the service order of packets when the service discipline is FCFS. In this case, even though connection 3 reserves more resources, the misbehavior of connection 1 and 2 affect its performance.

The Virtual Clock algorithm assigns each packet a virtual transmission time

based on the arrival pattern and the reservation of the connection to which the packet belongs. The fourth line shows the service order of the packets under Virtual Clock scheme. Notice that although connection 1 and 2 are sending packets at higher rates, the Virtual Clock algorithm ensures that each well-behaving connection(connection 3) gets good performance.

Self-Clocked Fair Queueing

The Virtual Clock service discipline is defined with reference to the static time system ($V^{VC}(t) = t, t \geq 0$) and time stamp of sessions are independent of each other. As a result, if a connection has been sending more packets than specified, it may be punished by Virtual Clock, regardless whether such behaviour affects the performance of other connections. Consider the scenario given in figure 2.5. In the beginning, packets of connection 1 arrive every time slot while connection 2 is idle. At time slot 20, the value of auxiliary Virtual Clock ($auxVC_1$) of connection 1 increases to 40. At time slot 20, packets of connection 2 arrives and the value of time stamp $auxVC_2$ assigned is 20 since $auxVC_2$ is zero. In this situation, packets of connection 1 are backlogged by Virtual Clock until the $auxVC_2$ of connection 2 increases up to 40. In this case, connection 1 is being punished by its misbehaviour in the past resulting in an unfair transmission.

To approach WFQ more accurately and avoid the above unfairness, [8] proposed a new emulation of WFQ called Self-Clocked Fair Queueing (SCFQ). Basically, the computation of time stamp under SCFQ is the same as Virtual Clock. Upon packet arrival, the scheduler assigns the time stamp F_j^k according

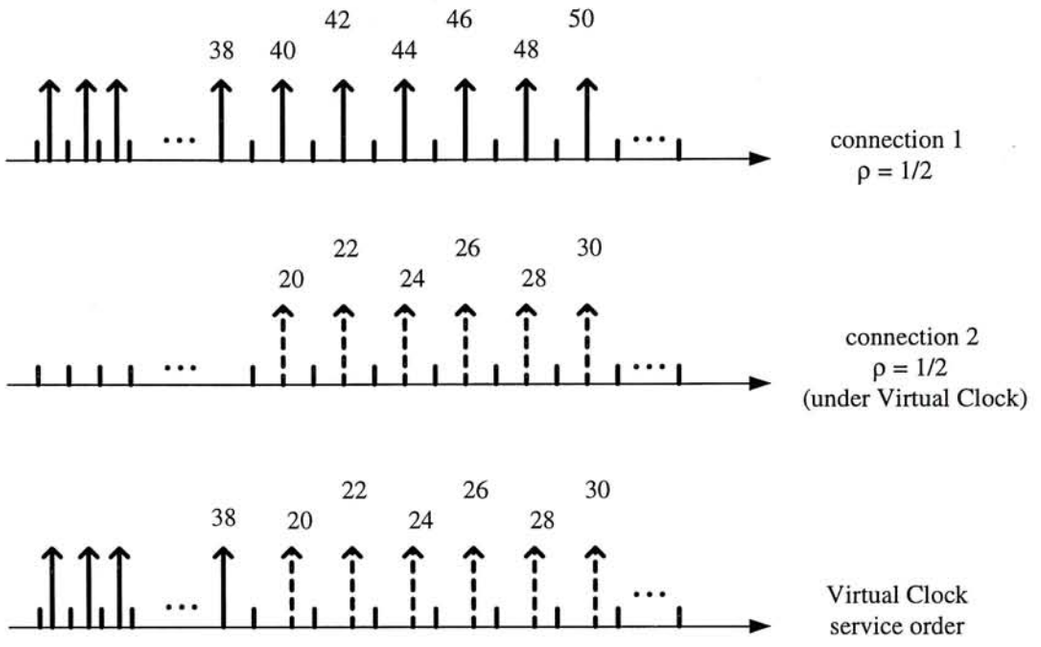

Figure 2.5: Unfair service order in Virtual Clock

to:

$$F_j^k \leftarrow max\{\hat{V}_i(a_j^k), F_j^{k-1}\} + \frac{L_j^k}{\phi_j} \tag{2.3}$$

where $\hat{V}_i(t)$ is the value of time stamp of the last departed packet
Unlike Virtual Clock, the time-stamp of that departed packet will be used as the new system virtual time.

Figure 2.6 shows the service order of packet under SCFQ in the same situation shown earlier. By applying the new update rule, we eliminate the credit gained when a connection is idle. Compare figures 2.6 and 2.5, although connection 2 is idle in the beginning, the new arrived packets are assigned with time stamps 40, 42, ... As a result, the packet service order of SCFQ is different from that of Virtual Clock. SCFQ avoids the unfairness introduced by the idle of connection and provides a more fair service order than Virtual Clock.

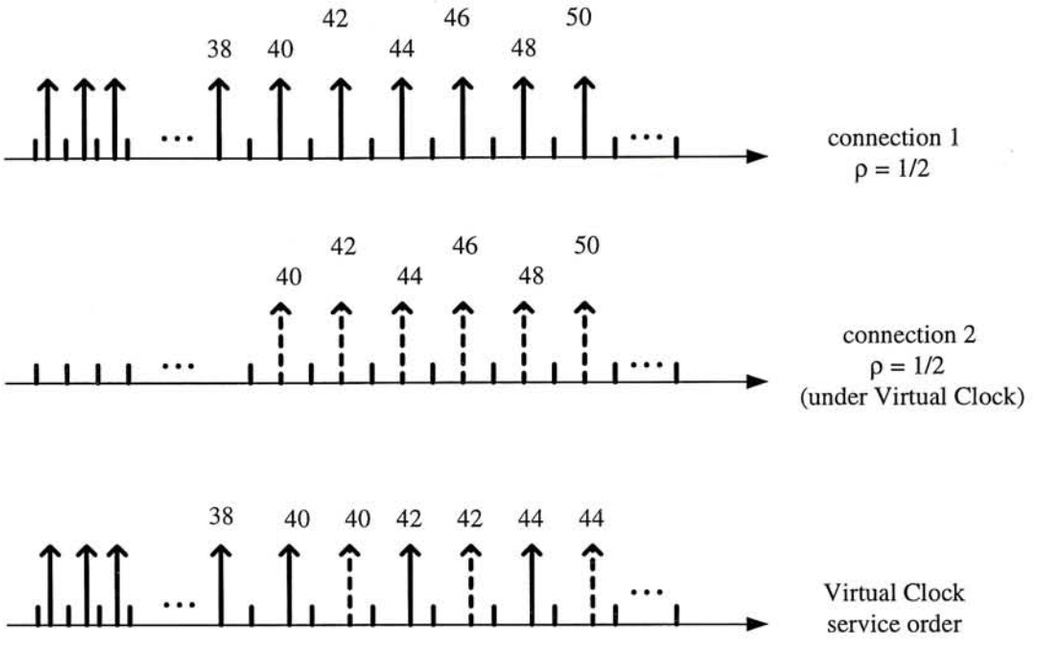

Figure 2.6: Service order under Self-Clocked Fair Queueing (SCFQ)

2.3 Traffic Scheduling in Input buffered Switch

In the past, a lot of attention in designing scheduling algorithms has been drawn to the output-buffered switch. The reason is because output-buffered switch can achieve 100% throughput with a proper designed scheduling algorithm. These scheduling algorithms like WFQ allow effective bandwidth allocation among calls and at the same time maintain a high utilization of switch bandwidth. However, the price of achieving these is high, the internal speed of the switch has to be increased to N (number of input ports) times the transmission rate of the external lines. Also, the scheduling algorithms used are usually computationally expensive. These two drawbacks of output-buffered switch limit its speed and scalability. On contrary, input-buffered switch with single First-In-First-Out(FIFO) queues is good for its simplicity. However, input-buffered

switch suffers from the problem of Head-Of-Line (HOL) blocking. When multiple packets are destined for the same output, only one of them is switched to the output while losing packets must wait at the head of the queue for retry in the coming time slots. As a result, the losing packets will forbid any packets behind them even though the packets being blocked may be switched in that time slot. Due to this HOL blocking problem, the maximum throughput of the generic switch of size N is limited to 58.6% for $N \rightarrow \infty$ [11]. This limitation makes input-buffered switch unacceptable for practical purpose.

Many techniques have been suggested to cope with the HOL blocking problem of the input-buffered switch. In the following, we will describe two fundamental methods to overcome this throughput limitation.

1. **Output Space Expansion:**

   By allowing more than one packet (say L packets) to reach an output simultaneously[12], the throughput of a switch can be increased. Several methods can be used to achieve simultaneous transmission of packets. We can speed up the internal switching fabric [13], or expand the output spaces by allocating multiple physical output ports to each logical output address[15], or by using multiple switching paths [14]. From [13], we can achieve 100% throughput in input-buffered switch if $L \geq 8$.

2. **Input look-ahead contention resolution[17]:**

   This method intends to increase the switch's throughput by reducing HOL blocking effect (Figure 2.7). During each time slot, the input still transmit only one packet to output. However, the packet being transmitted may not necessarily be the first packet in the FIFO queue. Each input can transmit

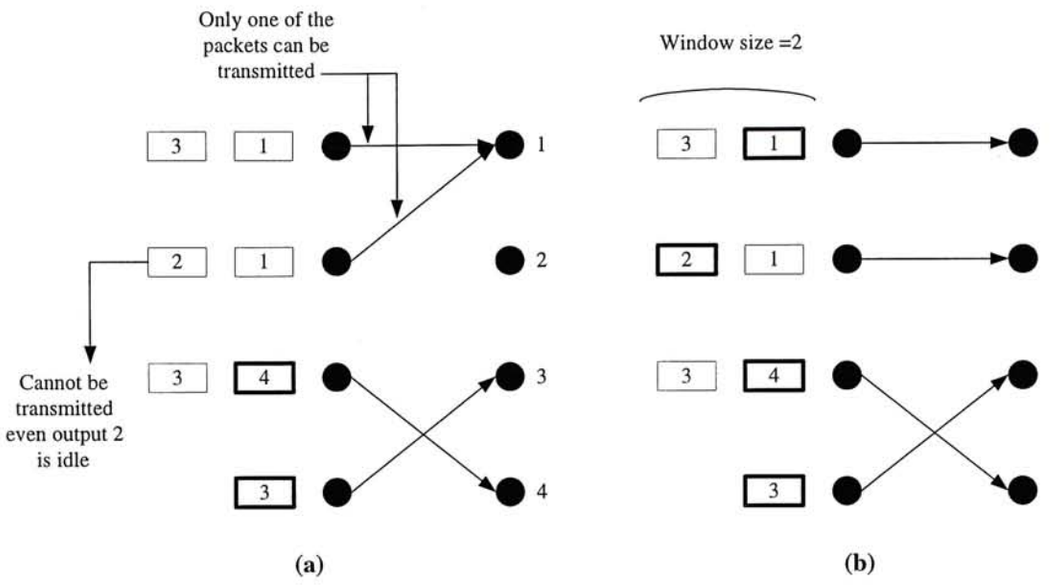

Figure 2.7: (a) HOL blocking problem (b) Look-ahead scheme ($w = 2$)

one packet among the first w packets (W: window size) at the head of the FIFO queue. Simulation and analysis show that this relaxation of packet selection actually improve the switch throughput. The large the window size, the better the throughput.

One of the most representative approach to scheduling in input-buffered switch is the Parallel Iterative Matching (PIM) [3]. It is well known that switch scheduling is simply an application of bipartite graph matching. During each time slot, Each output must be paired with at most one input that has a packet destined for that output. The main purpose of PIM is to find a maximal matching [1] in input-buffered switch. PIM uses parallelism, randomness, and iteration to accomplish this efficiently. There are 3 phases in this algorithm, namely Request, Grant and Accept. The scheduler will iterate the following steps:

[1] A Maximal matching is one for which pairings cannot be trivially added; each node is either matched or has no edge to an unmatched node.

1. **Request:**

   Each unmatched input sends a request to every output for which it has a backlogged packet. This notifies an output of all its potential partners.

2. **Grant:**

   If an unmatched output receives any requests, it chooses one randomly to grant. The output notifies each input whether its request was granted.

3. **Accept:**

   If an input receives any grants, it chooses one to accept and notifies that output.

Each of these steps occurs independently and in parallel at each input/output , there is no need of a centralized processor to do scheduling. At the end of each iteration, a legal matching of inputs to outputs is found. In PIM, more than one inputs can request an output and the grant phase will let the output choose one input among many. This ensures that an output will be paired up with at most one input. Similarly, in the accept phase, an input can accept an output among many grants ensuring it is paired up with at most one output.

After one iteration, we may have unmatched inputs with packets destined for unmatched outputs. An output whose grant is not accept may be able to pair up with an input whose request is not granted. In order to fill these "gap", we repeat the request, grant and accept phases, retaining the match found in the previous iteration. By PIM, HOL blocking problem is eliminated and the scheduling task is achieved.

Figure 2.8 illustrates one iteration of Parallel Iteration Matching. After this iteration, two connections have been set up and the match will be brought

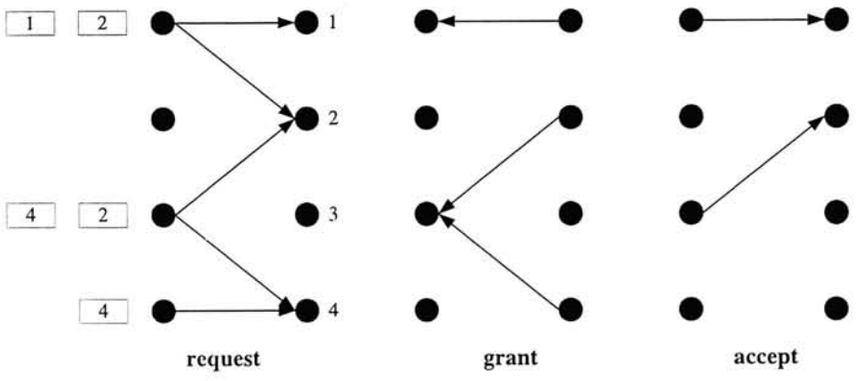

Figure 2.8: Parallel Iterative Matching: One Iteration

forward to next iteration. In the next iteration, only input 4 will request output 4. The request will be granted by output 4. In the accept phase, the grant issued by output 4 will be accepted. A connection between input 4 and output 4 will be set up. In this simple example, we see that how PIM can find a maximal matching of connection bipartite graph. We can iterate the algorithm until no further pairings can be found or after a fixed number of iterations. Then the result is used to setup the crossbar for the next time slot. From [3], study shows that with 4 iterations, 100% throughput of the input-buffered switch is achieved.

However, PIM only finds a matching in the connection bipartite graph and provides best-effort packet scheduling. The algorithm itself cannot provide fair or guaranteed service order to connections. Despite of this, PIM presents us with a good framework for tackling scheduling problem in input-buffered switch. The 3-phases communications of inputs and outputs gives a simple but also efficient protocol for performing matching between inputs and outputs. Much attention was drawn, and various modifications have been devised. The modifications are mostly done by changing the way of making decision in grant and accept phases. For instance, an alternative algorithm called Iterative Round Robin Matching

with slip (SLIP-IRRM) is proposed in [4]. SLIP-IRRM modifies the randomness of PIM. Instead of randomly grant a request and randomly accept a grant, It selects the packet to be transmitted through the use of round robin scheduler at each input and output port. Other derivatives of PIM include LRU and LQU [5, 16]. In the next chapter, we will present our enhancement to PIM. We adopt the fair queueing concept to PIM. We relinquish the use of random grants and accepts in the selection process and use fair queueing priority mechanism instead. By this modification, the resultant algorithm performs better and provides bandwidth guarantees to individual connections. We will elaborate the details of our algorithm in the next chapter.

Chapter 3

Deadline-ordered Parallel Iterative Matching (DLPIM)

3.1 Introduction

As mentioned previously,the ultimate goal of scheduling packets in high speed packet switches is to resolve the output conflicts of different packets. At the same time, the algorithm should achieve high throughput and distribute the switch bandwidth in a quick fair manner among different flows. We are going to achieve these requirements in our proposed algorithm and at the same time, make the algorithm simple and easily implemented.

In this chapter, we will present the defined problem and the switch model used. Then we introduce our algorithm – the Deadline-ordered Parallel Iterative Matching (DLPIM) – which performs packets scheduling in generic input-buffered switches. Finally, simulation results of our algorithm are presented in the end of this chapter.

3.2 Switch model

Consider a $N \times N$ internally non-blocking input-buffered switch, Figure 3.2. Inside the switch, there are N input ports and N output ports, an internally non-blocking switching fabric and a scheduler. Each input port contains some buffer to store arrived packets before they are transmitted. Traditionally, the switch employs First-In-First-Out (FIFO) queueing at each input port to buffer the backlogged packets. However, this scheduling policy limits the switch throughput to about 58.6% because of the Head-Of-Line (HOL) blocking problem.

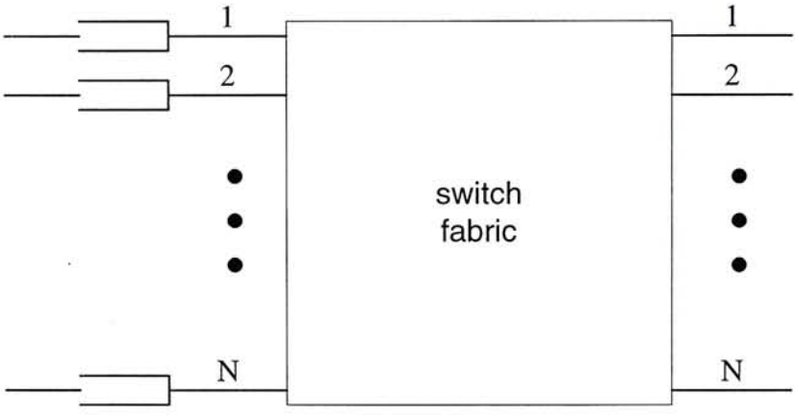

Figure 3.1: An $N \times N$ input-buffered switch

In order to increase the throughput of the switch, a special technique called Virtual Output Queueing (VOQ) is used to eliminate the HOL problem of input-buffered switches. Under VOQ, each input port maintains N logical queues, each of which is dedicated for one of the N output ports. Instead of considering only one packet in the FIFO queue, packets destining for different output ports are considered simultaneously during scheduling. Using VOQ, the problem of HOL blocking is eliminated.

The scheduler inside the switch is responsible for resolving conflicts among

packets destined for the same output port in different input ports. A good scheduler should be able to satisfy the performance guarantee to packets as specified in the setup time. Average delay, delay bound or other quantities can be used as measurement of performance. The scheduler is the unit where the scheduling algorithm is implemented. The decision made is then passed to the switching fabrics and the real connections among inputs and outputs are setup. The decision that made by the scheduler is what the main concern of our algorithm.

3.3 Deadline-ordered Parallel Iterative Matching (DLPIM)

3.3.1 Motivation

In the previous chapter, we have introduced the Parallel Iterative Matching (PIM) which is used to find a maximal matching of connections in input-buffered switch. There are three phases in PIM, namely Request, Grant and Accept phases. Figures 3.2 and 3.3 demonstrate the iterations of PIM in a particular time slot. In this example, there are six flows presented. The aggregate bandwidth allocation of each flow are listed as follows.

$$\lambda_0(0,0) = \frac{1}{8} \qquad \lambda_1(0,0) = \tfrac{1}{2} \qquad \lambda_0(0,1) = \frac{1}{4}$$
$$\lambda_0(1,0) = \frac{1}{3} \qquad \lambda_0(1,1) = \tfrac{1}{6} \qquad \lambda_1(1,1) = \frac{1}{5}$$

where $\lambda_k(i,j)$ is the k^{th} flow from input i to output j

After two iterations of PIM, a match pattern is found and packets can be transmitted according to the match pattern. However, from bandwidth allocation of flows, we see that the aggregate bandwidth $\lambda(0,0) = \lambda_0(0,0) + \lambda_1(0,0)$ is much greater than the aggregate bandwidth of input 0 to output 1 $\lambda(0,1)$. Instead of randomly grant a request or randomly accept a grant, inputs and outputs should issue their grants or accepts such that the rate of transmission of packets in each ports can be guaranteed. This is especially important if we want to provide Quality-of-Service (QoS) guarantee in switches.

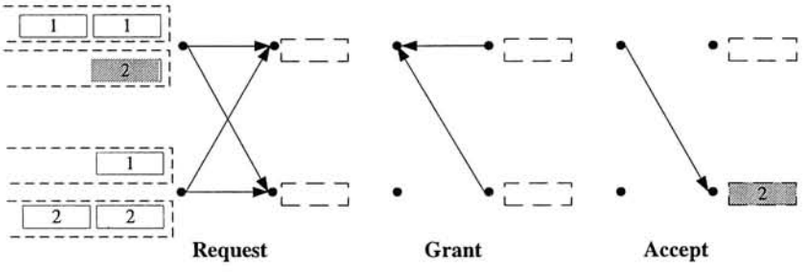

Figure 3.2: PIM: Time slot 1, 1st iteration

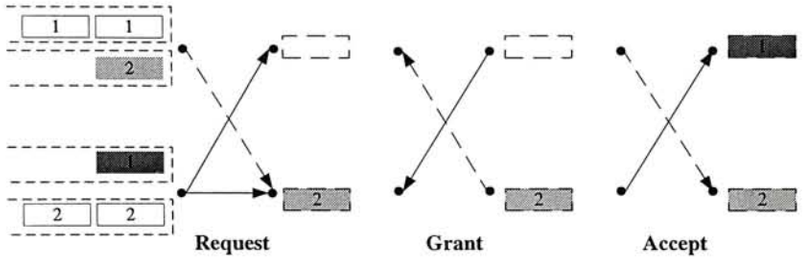

Figure 3.3: PIM: Time slot 1, 2nd iteration

3.3.2 Algorithm

In the following, we will introduce a new scheduling algorithm that contributes the major part of this thesis.

In this section, we will discuss the detail of the Deadline-ordered Parallel Iterative Matching (DLPIM). An example which makes use of the concept of Self-Clock Fair Queueing (SCFQ) will be presented to show the working principle of the DLPIM.

As mentioned before, in Parallel Iterative Matching (PIM), when there are multiple grants or acceptances, the corresponding input port or output port chooses one randomly. This results in a loose delay bound of flows and increases the maximum packet delay. In order to overcome the problem of PIM, we will use some mechanisms to determine how outputs grant a request or inputs accept a grant. One way to realize this is to use a deadline-ordered scheme with PIM.

In DLPIM, we preserve the concept of parallelism and iteration of the original PIM. Parallelism enables our algorithm to accomplish the scheduling task in a distributed manner. Each port does not require any interaction with other ports except the source or destination ports. Like PIM, each iteration of our algorithm consists of three phases: Request, Grant and Accept. At the beginning of each time slot, when new packets destined for certain output port j arrives at input port i, they are mapped to a logical flow queue $F_k(i,j)$ according to the flow k they belong to and each packet is assigned with a two-dimensional service tag $(I_{i,k}, O_j)$. Both service tags $I_{i,k}$ and O_j are calculated by the deadline-ordered scheme employed. Several schemes like VitualClock can be used in our algorithm. The time stamp $I_{i,k}$ is calculated according to the virtual time of the input port while O_j is calculated from the virtual time of the output port. After

assigning the service tag, we begin our iterations. The following operations of the three phases will be repeated for a fixed number of times or there is no unmatched ports:

Request: Every unmatched input port will request to every output port for which it has a unscheduled backlogged flow. During this phase, the input port should report which connections are backlogged to the destined output port.

Grant: If an output receives any requests, the output port will select the request with minimum service tag O_j and notifies the corresponding input port. If there are several packets having same value, the output port chooses one randomly.

Accept: If an input port receives only one grant, it will then choose the packet granted to transmit. On the other hand, if there are multiple grants, the input port will find the minimum service tag among all the granted connection and transmit the corresponding packet.

The Grant phase, as well as the Accept phase, of our algorithm is different from that of the original PIM. To grant a request, the output port must compares the output service tags O_j of requesting backlogged connections. The output port will then grant the request which has a minimum output service tag O_j packet.

Upon receiving multiple grants, our algorithm compares input service tags $I_{i,k}$ of the granted connections. Again, input port will accept the grant which acknowledges the connection with minimum input service tag $I_{i,k}$.

3.3.3 An example of DLPIM

In the following, we will present an example that employ our algorithm. In this example, we use Self-Clocked Fair Queueing (SCFQ) service discipline to calculate both input and output service tags. Consider a 2×2 input-buffered switch, there are six connections presented. The aggregate bandwidth allocation for each port is same as that in the example of PIM. Here we re-state bandwidth requirements for each flows. They are listed as follows.

$$\lambda_0(0,0) = \frac{1}{8} \qquad \lambda_1(0,0) = \tfrac{1}{2} \qquad \lambda_0(0,1) = \frac{1}{4}$$
$$\lambda_0(1,0) = \frac{1}{3} \qquad \lambda_0(1,1) = \tfrac{1}{6} \qquad \lambda_1(1,1) = \frac{1}{5}$$

Figure 3.4 and 3.5 show how our algorithm really works. The scenario is similar to the example showed in the beginning of this section. The scheduler still invokes 2 iterations in each time slot. Here we arbitrarily choose the value of virtual times of each switch port. The first phase of the first iteration is the same as that of the original PIM. All the input ports with backlogged packets will send the request signals to the destined output ports. During the grant phase, instead of randomly grant a request, each output port will compare the output service tag O_j of requesting packets and grant the packets with minimum service tag O_j. In the given example, both outputs 0 and 1 issue grants to input 0.

In the accept phase, upon receiving multiple grants from output ports, input port 0 compares the input service tags $I_j s$ of the granted connection. The input port will accept the granted connection which has a minimum input service tag packet. In this example, input port 0 will accept the grant issued from output port 0.

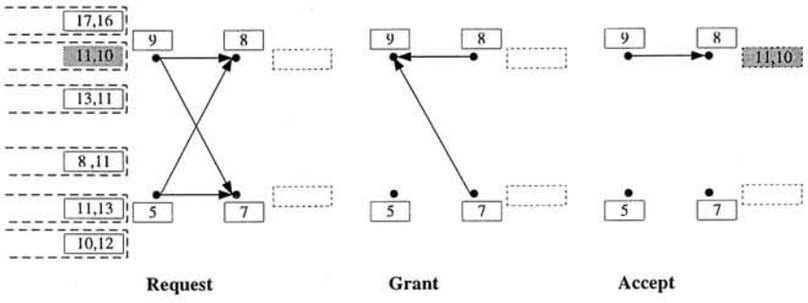

Figure 3.4: DLPIM: Time slot 1, 1st iteration

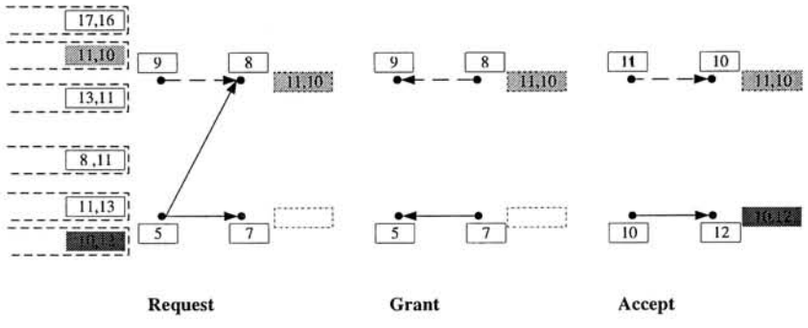

Figure 3.5: DLPIM: Time slot 1, 2nd iteration

In the second iteration, only input port 1 is unscheduled and it sends request to two output ports. Since only output port 1 is unmatched, it grants the request from input port 1. The input port 1 then accepts the grant and finishes the whole switching schedule for this time slot. After transmission of the packets, we update the system virtual time of both inputs and outputs.

We can see that the scheduling patterns of our algorithm is different from that of the original PIM. The major reason is that during the grant phase, upon receiving multiple requests, the output port no longer randomly chooses one request to grant. The output now compares the output service tags associated with packets at the head of each backlogged flow queue. Based on these service tags, the output issues the grant to the request with minimum output service tag. Similarly, in the accept phase, the input port now accepts the grant which

belongs to the flow with minimum input service tag. By time stamp mechanism, packets are transmitted according to their assigned transmission deadlines and eligibility times. This results in a more systematic and deterministic way of transmission of packets.

3.4 Simulation

In this section, we estimate the performance of PIM and DLPIM by simulating a 4×4 switch under these two algorithms. A 4×4 switch with uses Bernoulli packets arrival process is used in our simulation. In both simulations, four iterations are used for scheduling packets in each time slot. Each input port is 90% loaded with respect to the capacity of the port. For each input port, there is a flow destined for each output port. Our focus is on the packet delay bound of individual flow. We will compare the packet delay distribution of the flows under two algorithms. In the following, we focus on the delay of input port 0.

In the first simulation, each input port has same traffic load to every output port in the switch. The connection matrix is shown as follows.

$$\lambda = \begin{bmatrix} 0.225 & 0.225 & 0.225 & 0.225 \\ 0.225 & 0.225 & 0.225 & 0.225 \\ 0.225 & 0.225 & 0.225 & 0.225 \\ 0.225 & 0.225 & 0.225 & 0.225 \end{bmatrix}$$

where λ_{ij} is the flow from input i to output j

Figures 3.6 and 3.7 show the delay distribution of the transmitted packets. The maximum delay of the transmitted packets are recorded in table 3.1. Although both algorithms have similar performance which are in term of the average

packets delay (DLPIM: 4.341067, PIM: 4.789768), the maximum packet delay for them are different. Under our algorithm, the maximum delay bound of the transmitted packets is smaller than that of PIM. When we compare the figures 3.6 and 3.7, we can see that the tail probability of the packet delay for our algorithm is smaller than that of the original PIM. From table 3.1, the maximum packet delay under DLPIM is smaller than that of PIM. Thus, we can see that our algorithm have better control on the maximum packet delay bound than PIM.

Input\ Output	0	1	2	3
0	23	26	25	24
1	25	25	24	22
2	29	24	27	27
3	26	26	27	26

a)

Input\ Output	0	1	2	3
0	44	52	41	50
1	41	43	42	41
2	46	39	43	45
3	46	49	38	36

b)

Table 3.1: Max. packet delay under uniform load 0.9: a) DLPIM, b) PIM

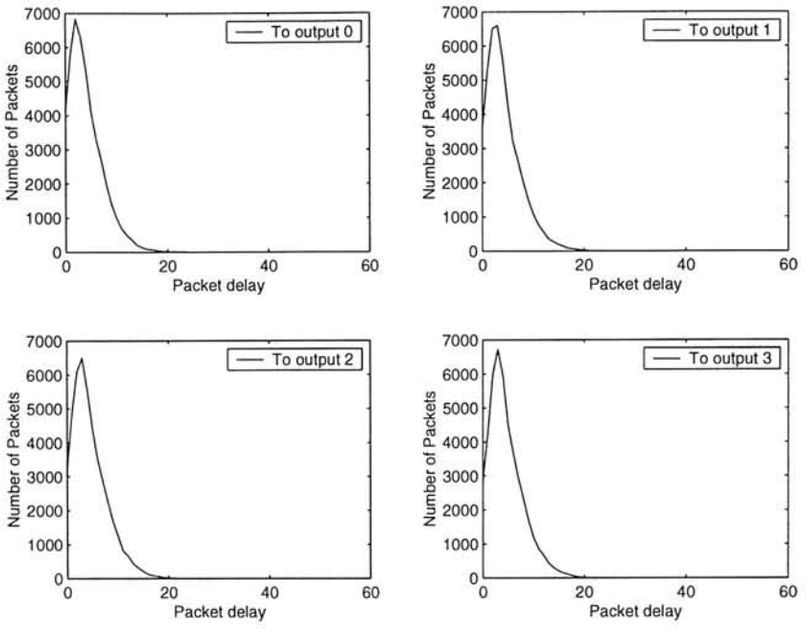

Figure 3.6: Packet delay distribution using DLPIM under uniform load 0.9

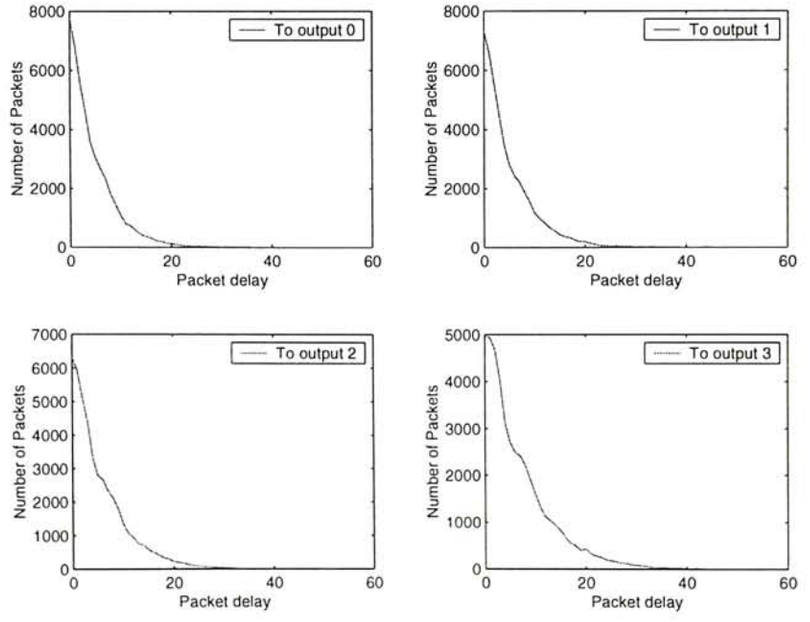

Figure 3.7: Packet delay distribution using PIM under uniform load 0.9

In the second simulation, the connection matrix is changed and shown as follows.

$$\lambda = \begin{bmatrix} 0.6 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.6 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.6 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.6 \end{bmatrix}$$

Now there is a main stream from input i to output i and the bandwidth allocation for the rest of the flow is much lower than the main one. Figures 3.8 and 3.9 show the delay distribution of the transmitted packets, while table 3.2 shows that maximum packet delay under DLPIM and PIM correspondingly.

From table 3.2, connections that have a high bandwidth allocation have smaller maximum delay under our new scheme. In PIM, the main stream packets experience very large delay fluctuation. This is easily seen from table 3.2 and figure 3.9. The variation in the packet delay reflects that PIM does not consider the bandwidth allocation of the connection. From the bandwidth requirement of flow $F_0(0,0)$, it is obvious that more packets will arrive in flow queue $F_0(0,0)$ and destined for output port 0. But due to the random grant and accept of PIM, more and more packets in flow queue $F_0(0,0)$ are backlogged. This results in the queue up of packets in the flow 0 and thus large packet delay is resulted.

Input\ Output	0	1	2	3
0	11	39	41	48
1	44	11	43	46
2	43	43	10	48
3	47	53	41	11

a)

Input\Output	0	1	2	3
0	150	37	34	40
1	44	3286	38	35
2	32	33	3944	39
3	43	50	45	778

b)

Table 3.2: Max. packet delay under asymmetric load: a) DLPIM, b) PIM

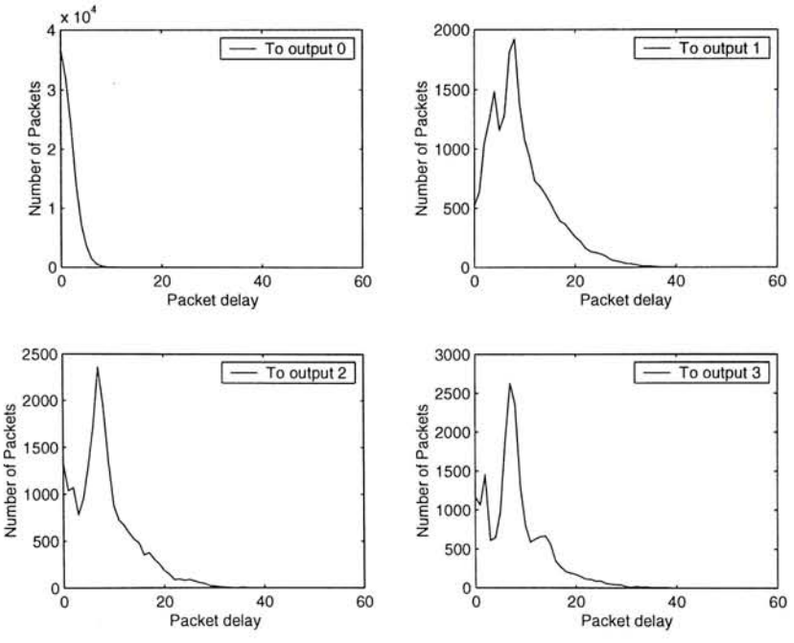

Figure 3.8: Delay distribution using DLPIM under asymmetric load

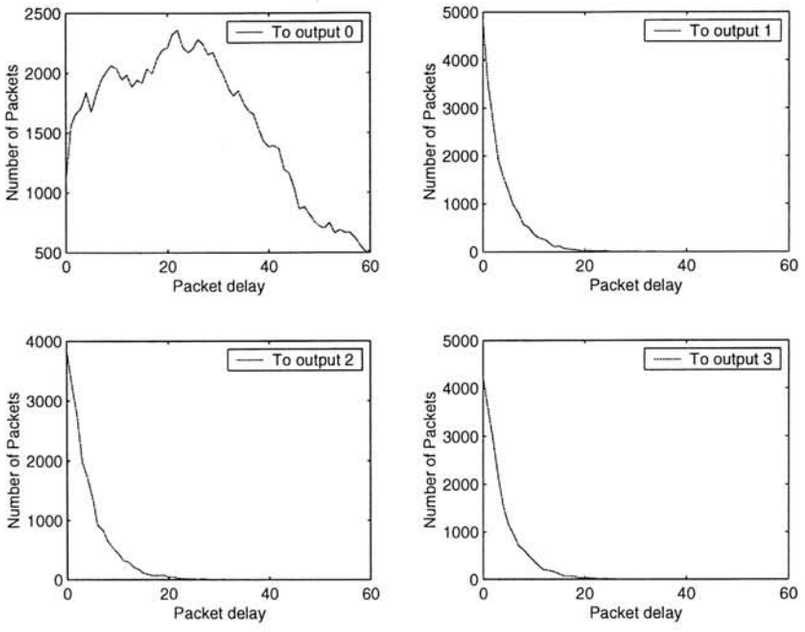

Figure 3.9: Delay distribution using PIM under asymmetric load

In our final simulation, we will simulate a situation that one connection session is misbehaved. A connection is misbehaved if the arrival rate of its packets is larger than the reserved rate specified in the setup time. The reserved bandwidth λ of each connection and the corresponding arrival rate $\lambda\prime$ used in the simulation are shown in the following matrix.

$$\lambda = \begin{bmatrix} 0.225 & 0.225 & 0.225 & 0.225 \\ 0.225 & 0.225 & 0.225 & 0.225 \\ 0.225 & 0.225 & 0.225 & 0.225 \\ 0.225 & 0.225 & 0.225 & 0.225 \end{bmatrix} \qquad \lambda\prime = \begin{bmatrix} 0.34 & 0.22 & 0.22 & 0.22 \\ 0.22 & 0.225 & 0.225 & 0.225 \\ 0.21 & 0.225 & 0.225 & 0.225 \\ 0.21 & 0.225 & 0.225 & 0.225 \end{bmatrix}$$

We can see that flow $F_0(0,0)$ is misbehaved, meaning that the packet arrival rate is greater than the reserved bandwidth of the flow. The delay distribution of the packets under two algorithms are shown in figures 3.10 and 3.11. The maximum packet delays of flows are listed in table 3.3.

The simulation results are similar to that of uniform traffic. The maximum packet delay using PIM is larger than that of DLPIM. Again, DLPIM performs better than PIM in term of packet delay bound. Moreover, if we compare the result with the result when there is no misbehaved flow $F_0(0,0)$ (see table 3.4, figure 3.12 and 3.13), we can see that the presence of the misbehaved flow in PIM actually increases the maximum delay of all the connections of input port 0. On the other hand, in the case of DLPIM, flows are well-shielded from negative effect created by the misbehaviour of flows in the input ports. From table 3.4, we see the maximum packet delay of flows under the DLPIM scheme is roughly the same in both situations.

Input\ Output	0	1	2	3
0	N/A	26	27	25
1	27	25	28	24
2	27	25	27	25
3	26	28	26	25

(a)

Input\Output	0	1	2	3
0	N/A	55	66	81
1	45	41	40	37
2	48	37	38	38
3	44	45	35	36

(b)

Table 3.3: Max. Delay with bad-behaviour: (a) DLPIM, (b) PIM

Input\ Output	0	1	2	3
0	N/A	22	24	23
1	24	23	27	22
2	21	24	27	24
3	27	27	25	26

(a)

Input\Output	0	1	2	3
0	N/A	39	36	50
1	36	45	40	40
2	42	36	43	47
3	37	67	38	40

(b)

Table 3.4: Max Delay without bad-behaviour: (a) DLPIM, (b) PIM

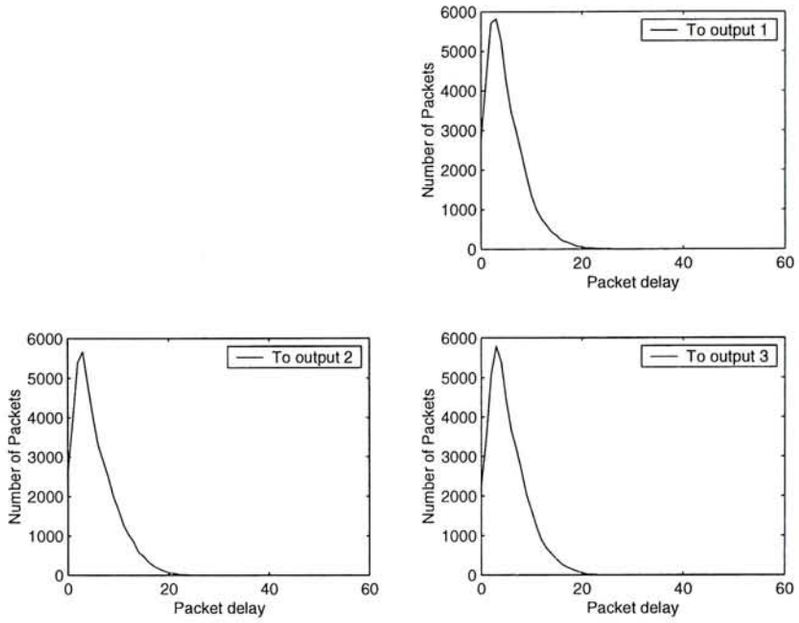

Figure 3.10: Delay distribution using DLPIM with mis-behaved flow $F_0(0,0)$

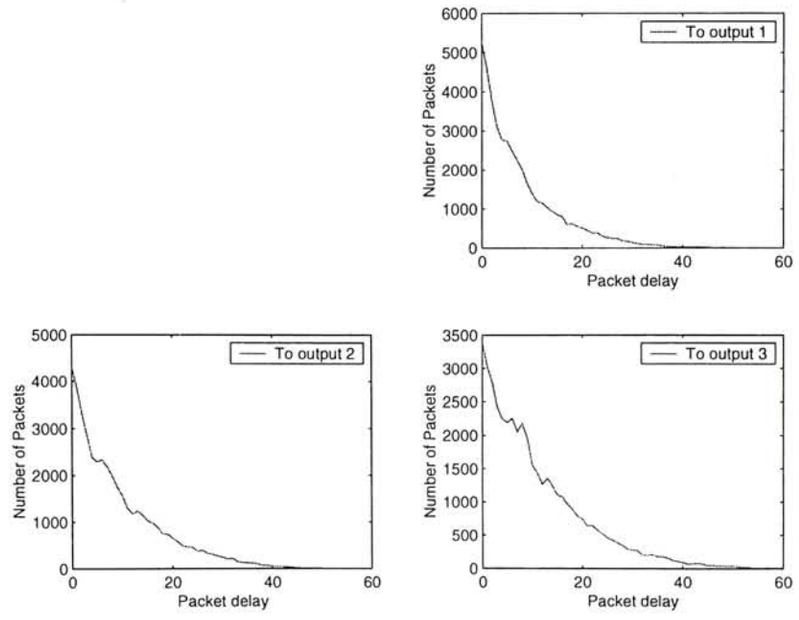

Figure 3.11: Delay distribution using PIM with mis-behaved flow $F_0(0,0)$

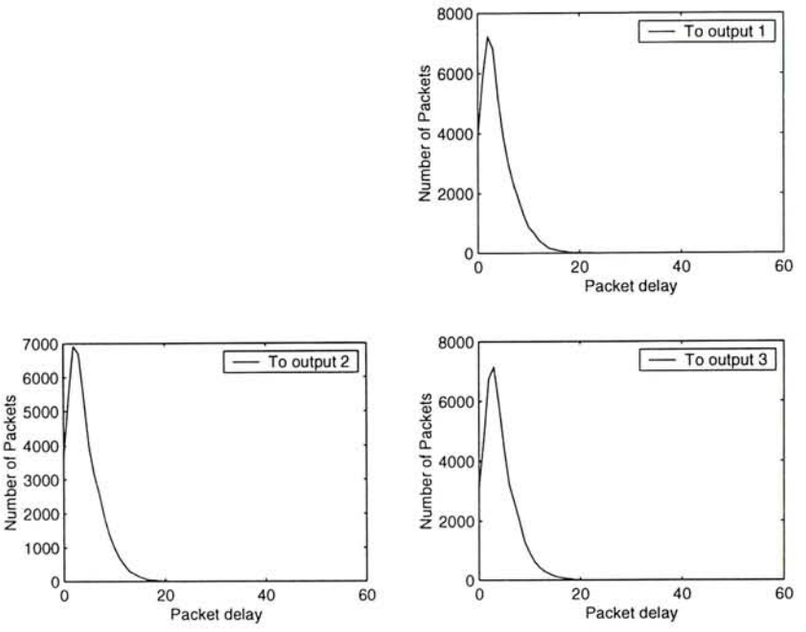

Figure 3.12: Delay distribution using DLPIM without mis-behaved flow

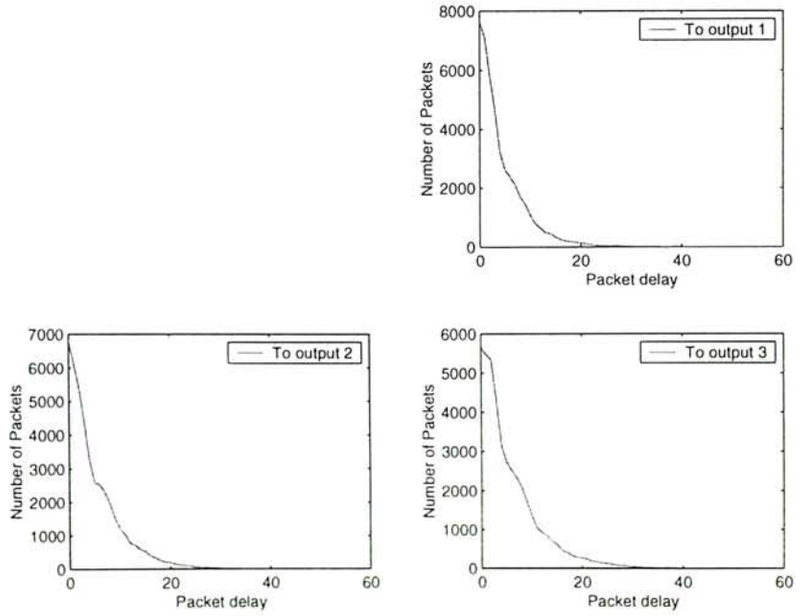

Figure 3.13: Delay distribution using PIM without mis-behaved flow

From the above simulation results, we obtained:

- If the traffic loading of each input is the same, DLPIM performs as good as PIM in term of average delay. However, under DLPIM, the maximum delay experienced by packets is small than that of PIM.

- If the traffic loading is asymmetric, like there are flows that require much more bandwidth than other flows, DLPIM performs better than PIM in minimizing the maximum delay of packets. Under PIM, the average delay of the flows increases with the increase of bandwidth. However, using our algorithm, the average delay is small. Similar to the first simulation, maximum delay experienced by packets is smaller under DLPIM than PIM.

- With the present of bad-behaved flows, DLPIM provides firewall protection among individual data flows. We show that our algorithm prevents misbehaving flows from interrupting normal service to others.

From the simulations, we show that unlike PIM, DLPIM schedules the transmission of packets according to the service tags assigned to packets. By adjusting the policy of service tags, we can easily change the quality of service provided to each flows. In our simulation, we use the bandwidth allocation of each flows to calculate the service tags of packets, as a result, packets belonging to different flows will experience different services in term of average delay. Using our algorithm, the switches does not just schedule packets, but also provides different level of services to individual flows. This is a remarkable advantage if we want to guarantee Quality-of-Service (QoS) of flows in switch.

Another improvement of DLPIM over the original PIM is that DLPIM provides guarantee of packets transmission of flows even under the present of misbehaved flows. This feature will increase the stability of the switches if the switches need to handle large amount of datagram traffic with unpredictable arrival pattern.

Chapter 4

DLPIM with static scheduling algorithm

4.1 Introduction

Providing Quality-of-Service in ATM networks is a hot topic in telecommunication. One way to achieve this goal is to operate the system using a static route assignment such as multirate circuit switching [22]. Under static routing scheme, a path is established for each connection such that the peak capacity is reserved along the path. In this way, we guarantee that the network can carry this connection at any time and provide QoS service to this connection. However, static routing schemes suffer from low utilization of the system because they do not exploit the full advantage of the statistical multiplexing gain offered by ATM networks. Another extreme is dynamic scheduling of packets, also call *cell switching*, such as matching proposed in [3]. Basically, dynamic routing schemes arrange the connection pattern every time slot according to the arrival

packets. In general, the system utilization of the dynamic schemes is higher than that of static scheme. One of the major drawback of dynamic schemes is that the computation cost of slot-by-slot paths assignment is too high. In each time slot, the switch need to collect and process the global traffic information in order to do path hunting on the fly. These operations severely limit the speed and size of the switch. Besides the computation cost, dynamic routing schemes fail to give the desired QoS requirements for each individual connection.

Many effort have been put in finding a scheduling algorithm that can give high system utilization and guarantee individual connection QoS. The scheduling algorithm should remain simple and effective. Recently, a new "path switching" [19] principle has been proposed to achieve optimal performance in scalable ATM switch design. Path switching is a quasi-static routing scheme which is a compromise of the dynamic and the static routing schemes. Path switching uses a predetermined periodical connection pattern to schedule the transmission of packets. In this chapter, we suggest another way to scheduling packet transmission so that we can guarantee the QoS requirement of connections and a high utilization is yielded. We will demonstrate how DLPIM can achieve this through the incorporation of static scheduling algorithm.

4.2 Static scheduling algorithm

In this section, we are going to describe the basic principle of static algorithm. Consider a $N \times N$ internally blocking switch, we specify the bandwidth requirement λ_{ij} between input i and output j using a bandwidth requirement matrix $B = [\lambda_{ij}]$ (eq.4.1) . In the following, we assume the capacity of links to be

normalized to 1, which is the maximum bandwidth allowed for connections.

$$B = \begin{pmatrix} \lambda_{0,0} & \lambda_{0,1} & \cdots & \lambda_{0,N-1} \\ \lambda_{1,0} & \lambda_{1,1} & \cdots & \lambda_{1,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{N-1,0} & \cdots & \cdots & \lambda_{N-1,N-1} \end{pmatrix} \tag{4.1}$$

with the stability condition:

$$\sum_{i=0}^{N-1} \lambda_{ij} \leq 1, \text{ for all i} \tag{4.2}$$

$$\sum_{j=0}^{N-1} \lambda_{ij} \leq 1, \text{ for all j} \tag{4.3}$$

Equations 4.2 and 4.3 ensures that the total bandwidth subscription of connections does not exceed the maximum capacity of the switch.

To realize the bandwidth requirement of the above matrix, we need to convert the above bandwidth requirement into the number of ATM packets transmitted. We divide the time axis into frames containing a fixed number of time slots (say F). In each frame, we can transmit at most F ATM packets. To guarantee the bandwidth requirement λ_{ij} of each connection, we need to calculate how many packets a connection can send within a frame time. It is done by multiplying each λ_{ij} with the frame size F, that is $c_{ij} = \lambda ij \times F$. To ensure individual connection gets its own bandwidth requirement, we will reserve $\lceil c_{ij} \rceil$ time slots. The following shows the computation of connection pattern.

$$C = B \times F$$

$$= \begin{pmatrix} \lambda_{0,0} & \lambda_{0,1} & \cdots & \lambda_{0,N-1} \\ \lambda_{1,0} & \lambda_{1,1} & \cdots & \lambda_{1,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{N-1,0} & \cdots & \cdots & \lambda_{N-1,N-1} \end{pmatrix} \times F$$

$$= \begin{pmatrix} c_{0,0} & c_{0,1} & \cdots & c_{0,N-1} \\ c_{1,0} & c_{1,1} & \cdots & c_{1,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ c_{N-1,0} & \cdots & \cdots & c_{N-1,N-1} \end{pmatrix}$$

Note that matrix $C = [c_{ij}]$ is known as capacity matrix. From equations 4.2 and 4.3, we deduce the following stability conditions 4.4 and 4.5:

$$\sum_{i=0}^{N-1} c_{ij} \leq F, \text{ for all i} \tag{4.4}$$

$$\sum_{j=0}^{N-1} c_{ij} \leq F, \text{ for all j} \tag{4.5}$$

Using this capacity matrix C, we can determine a conflict-free connection pattern within a frame. The connections are then set up and the pattern will varies from time slot to time slot. The whole schedule will be repeated every F time slots. The problem of determining that conflict-free pattern is well-addressed and has been described in [3, 19, 20, 23]. In the following, we use the bipartite graph edge coloring approach to find the connection pattern. For an $N \times N$ input-buffered switch, the relationship between the connection pattern and the capacity matrix is shown in figure 4.2.

In figure 4.2, we represent the inputs and outputs as two sets of nodes, namely I and O. If input i need to transmit c_{ij} packets to output j, then we draw c_{ij}

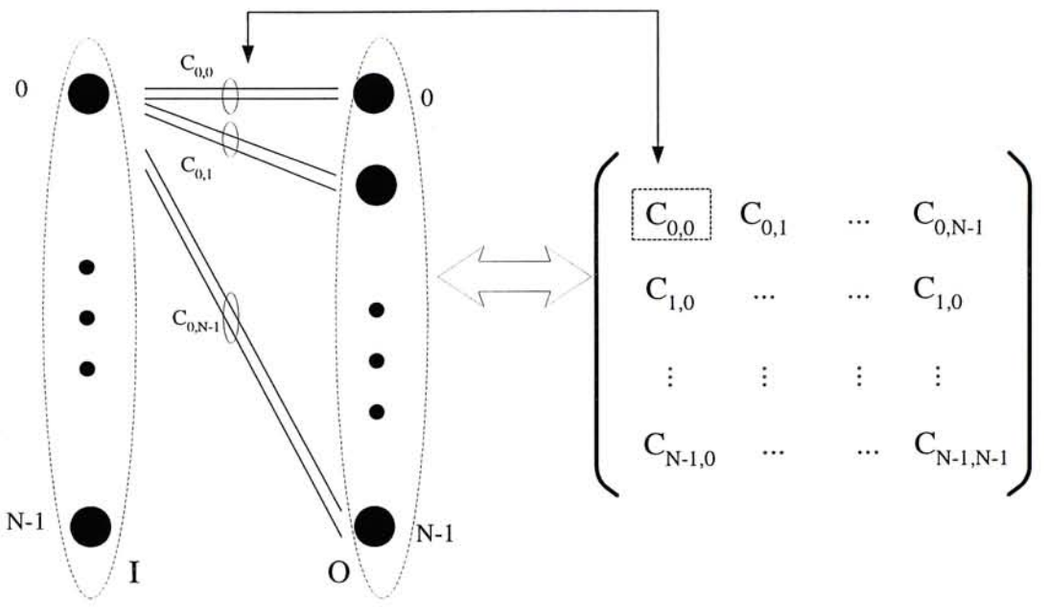

Figure 4.1: Bipartite Graph and the corresponding capacity matrix

edges from input i to output j. Using this representation, we obtain a bipartite graph which shows the connections of inputs and outputs in F time slot. Since in one time slot, we allow only one packet to be transmitted from one input to one output. In order to find the actual connection pattern in a particular time slot, we need to separate the graph into several graphs with each represents a connection pattern in a single time slot. The task of determining a conflict-free schedule is equivalent to the edge coloring problem of bipartite graph with F colors, where each color represents a time slot within a frame. For a regular bipartite graph of degree F, it has been proven that we can use F distinct colors to color all the edges. The resulting edge coloring is an one-to-one corresponding to a conflict-free schedule of the switch.

Figure 4.2 shows an example with $N = 2$, $F = 3$. The bandwidth requirement is given in the figure and the capacity matrix is calculated according to

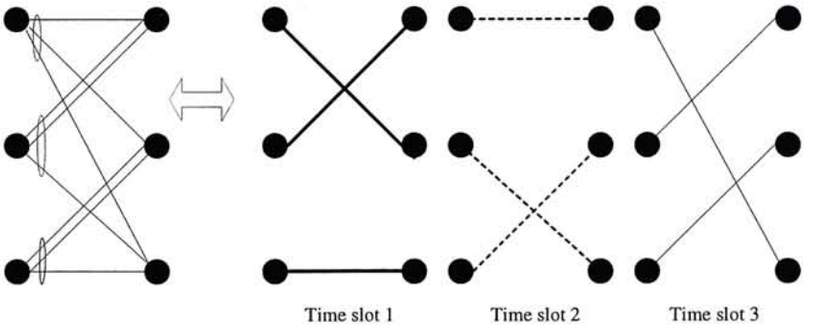

Figure 4.2: An example of edge-coloring problem

the following:

$$C = \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{2}{3} & 0 & \frac{1}{3} \\ 0 & \frac{2}{3} & \frac{1}{3} \end{pmatrix} \times 3$$

$$= \begin{pmatrix} 1 & 1 & 1 \\ 2 & 0 & 1 \\ 0 & 2 & 1 \end{pmatrix}$$

The capacity matrix shown above will then be convert to a bipartite graph with degree 3. We then use the edge-coloring technique and find the colored bipartite graph (figure 4.2). Each colored edge in the graph represents the connection in a particular time slot. This colored graph is then used to set up the real connections in the switch within a frame.

Although static scheduling algorithm provides QoS guarantee to connections, the major drawback of the algorithm is the inefficient bandwidth allocation. Consider the bandwidth requirement of connections given by the following matrix:

$$B = \begin{pmatrix} \frac{1}{8} & \frac{1}{2} & \frac{1}{5} \\ \frac{1}{3} & \frac{1}{3} & 0 \\ \frac{1}{3} & \frac{1}{10} & \frac{1}{2} \end{pmatrix}$$

and its corresponding capacity matrix with $F = 10$

$$C = \begin{pmatrix} 2 & 5 & 2 \\ 4 & 4 & 0 \\ 4 & 1 & 5 \end{pmatrix}$$

We note that in this connection matrix, the output links are not fully reserved by the inputs because

$$\sum_{i=0}^{2} \lambda_{ij} < 1, \text{ for all i}$$

$$\sum_{j=0}^{2} \lambda_{ij} < 1, \text{ for all j}$$

During a frame time, there are some empty time slots that no packets are transmitted. In static scheduling, connections requiring bandwidth will transmit their packets in the reserved time slots. However, we can make good use of these unreserved time slots to transmit extra packets. In this way, we can increase the utilization of the switch.

Another noteworthy point is that frame size determines the frame granularity. In the above matrix, $b_{0,0} = \frac{1}{8}$ and $F = 10$. Since the number of time slot allocated for each connection must be integer, we allocate 2 time slots for this connection although $\frac{1}{8} \times 10 = 1.6$. We reserve more time slots for the connection, despite the connection may not have packets to transmit. However, if we choose $F = 120$,

we obtain the new capacity matrix as:

$$C' = \begin{pmatrix} 15 & 60 & 24 \\ 40 & 40 & 0 \\ 40 & 12 & 60 \end{pmatrix}$$

and avoid the excessive reservation due to round-off of the capacity matrix. However, the drawback of larger frame size is the increase in the degree of corresponding bipartite graph. This increases the size of schedule that the scheduler need to keep track of and introduces more delay to packets.

4.3 DLPIM with static scheduling algorithm

As mentioned previously, static scheduling algorithms lacks of the flexibility to provide best-effort service. In each frame, connections are allowed to transmit packets in a predetermined periodical pattern. In general, static scheduling algorithm performs well when traffic belongs to constant bit rate (CBR) or variable bit rate (VBR) type. For data traffic belongs to unspecified bit rate (UBR) or available bit rate (ABR), it is difficult for static scheduling algorithm to perform well.

One of the advantages of ATM networks is the high system utilization given by statistical multiplexing gain. In many scenarios, both best-effort and bandwidth guaranteed service connections exists in the network. Many researches focus on finding solutions to deal with the issue of supporting those categories of traffic within the network. Now, we are going to introduce a method that makes use of both DLPIM and static scheduling algorithm to achieve our goal. Here are the details of our implementation:

During the set up time of a connection, we calculate the capacity matrix based on the bandwidth requirement of connections in the switch. Instead of reserving peak bandwidth requirement of connections, we only allocating minimum bandwidth required by connections. Upon an arrival of packets, we assign the service tag $(I_{i,k}, O_j)$ according to its source port i, flow k and its destination port j using deadline-ordered scheme. This time stamp assignment is the same as that in DLPIM. After that, we begin our iteration.

Request/Connect: In this phase, each input port will send a request / connect signal to an output port. If the input port has already been scheduled to transmit a packet to a specified output port, then the input port will send a connect signal indicating a connection must be set up between this input-output pair regardless of any request received by the output. On the other hand, if there is no any pre-determined connection, then the input port will send a request signal to every output port for which it has a unscheduled backlogged flow. During this phase, the input port should report which connections are backlogged to the destined output port.

Grant: If an output receives a connect signal, then the output will grant the corresponding input port and ignore any requests from other input ports. However, if no connect is received, then the output port will select the request with minimum service tag O_j and notifies the corresponding input port. If there are several packets having same value, the output port chooses one randomly.

Accept: If an input port receives only one grant, it will then choose

the packet granted to transmit. On the other hand, if there are multiple grants, the input port will find the minimum service tag among all the granted connection and transmit the corresponding packet.

In the above algorithm, it is clear that the Request and Grant phases of DLPIM are modified such that the static scheduling can be employed. The role of static scheduling is to reserve certain number of time slots within a frame to those connections requiring bandwidth guarantees. Then we use DLPIM mechanism to fill the granules in the frame. The modifications ensure that DLPIM will not affect the schedule pre-computed by static scheduling algorithm.

4.4 An example of DLPIM with static scheduling algorithm

Now we are going to see an example of how our scheme really works. Consider the following connections in a 3×3 switch:

$$\begin{array}{lll} \lambda_{0,0} = \dfrac{1}{4} & \lambda_{0,1} = \frac{3}{4} & \lambda_{0,2} = 0 \\ \lambda_{1,0} = 0 & \lambda_{1,1} = \frac{1}{4} & \lambda_{1,2} = \dfrac{1}{4} \\ \lambda_{2,0} = \dfrac{1}{2} & \lambda_{2,1} = 0 & \lambda_{2,2} = \dfrac{1}{4} \end{array}$$

Among all the connections, connections $\lambda_{1,2}, \lambda_{2,0}$ and $\lambda_{2,2}$ are ABR services with minimum bandwidth of $\frac{1}{4}$, $\frac{1}{2}$ and $\frac{1}{4}$, respectively. This means that arrival of the above connections may be higher than the one specified. According to the bandwidth requirement matrix, we calculate the capacity matrix (Eq. 4.6 with

$F = 4$) and obtain the connection patterns (figure 4.3) in each time slot.

$$B = \begin{pmatrix} \frac{1}{4} & \frac{3}{4} & 0 \\ 0 & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{2} & 0 & \frac{1}{4} \end{pmatrix}$$

$$C = \begin{pmatrix} \frac{1}{4} & \frac{3}{4} & 0 \\ 0 & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{2} & 0 & \frac{1}{4} \end{pmatrix} \times 4$$

$$C = \begin{pmatrix} 1 & 3 & 0 \\ 0 & 1 & 1 \\ 2 & 0 & 1 \end{pmatrix} \tag{4.6}$$

We note that major part of the schedule is done except in the last two time slot.

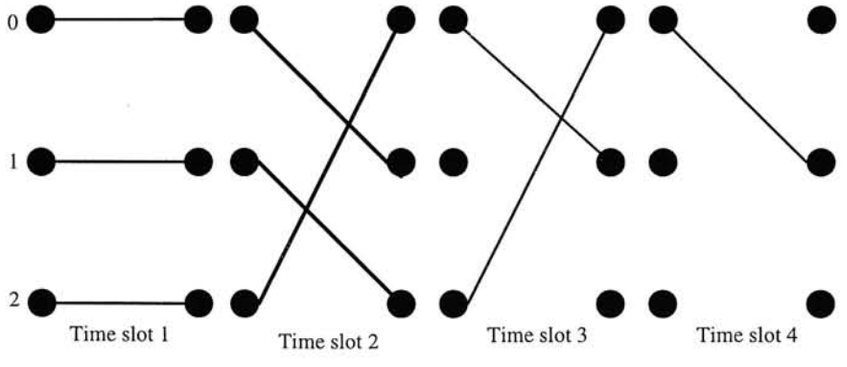

Figure 4.3: Connection patterns under static scheduling

From figure 4.3, input ports $1, 2$ and output port $0, 2$ are idle. Then we carry out DLPIM in time slot 3 and 4 to schedule the remaining schedule. Figure 4.4 shows the iteration of DLPIM at time slot 4 in this example. Similar operation is carried out in time slot 3 and results in a connection pattern in figure 4.5.

From this example, we see how DLPIM benefits the static scheduling algorithm. Compare figure 4.3 and 4.5, the "holes" between inputs and outputs in

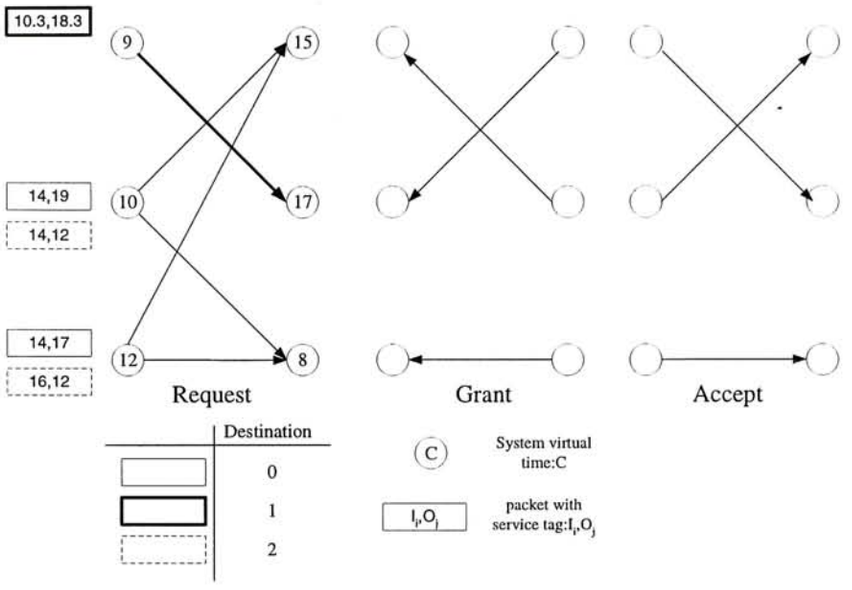

Figure 4.4: DLPIM in time slot 4

time slot 3 and 4 is filled. Under the new scheme, unoccupied time slots now can be used to transmit packets in a best-effort manner. On the other hand, static scheduling algorithm computes part of the schedule during connection set up time. This simplifies the remaining scheduling of packet transmission and reduces the size of corresponding bipartite matching of inputs and outputs. As a result, we can reduce the number of iterations used in DLPIM and decrease

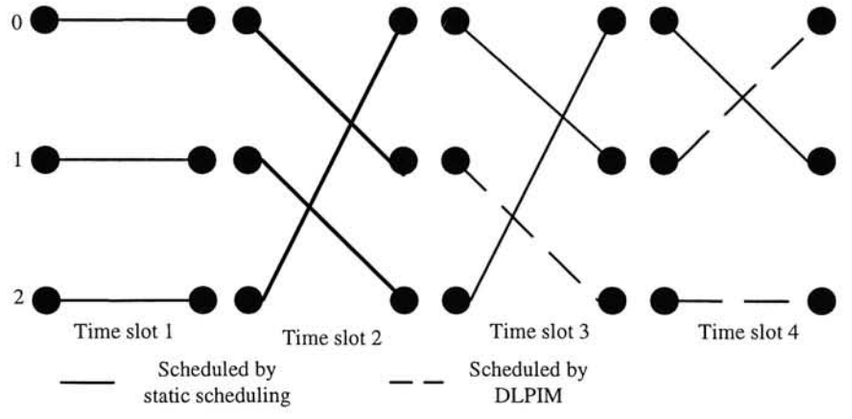

Figure 4.5: Connection patterns under static scheduling with DLPIM

the complexity of the scheduling algorithm. This incorporation guarantees each connection receives its required service, and at the same time, maintains a high system utilization .

Chapter 5

Conclusion

In this thesis, we propose a new scheduling algorithm, called DeadLine-ordered Parallel Iterative Matching (DLPIM), which performs packet scheduling in input-buffered switches. We present the detail of its working principle and illustrate its usages. DLPIM is based on the Parallel Iterative Matching (DLPIM) and employ fair queueing service order in the grant and accept phases. Instead of randomly grant or accept, we use a sorted priority queue mechanism to determine the service order of packets. Using our new scheme, maximum packet delay of flows decreases. In addition to the improvement on delay bound, DLPIM also provides a firewall protection of well-behaved flows against mis-behaved flows. These features can be seen in our simulation results. DLPIM not only solves the output contention of input-buffered switch, but also enables data delivery with a smaller delay bound, protection of flows against mis-behaved flows. These properties of DLPIM make the algorithm superior to PIM and are essential if we need to guarantee the QoS of flows.

Another issue investigated in this thesis is how to provide bandwidth guarantee to flows in ATM network. A simple and easy way to achieve this is to employ a static scheduling algorithm, which tries to find a frame schedule using the corresponding bandwidth requirement of individual flows. Static scheduling algorithm suffers from the problem of low utilization. To address this, we incorporate DLPIM with static scheduling scheme. By using this new scheme, we can provide the required bandwidth to connections by finding a frame schedule during setup time and maximize the utilization by finding a best-effort bipartite match between inputs and outputs. We illustrate how our algorithm combines with a static scheduling scheme to tackle the scheduling task in input-buffered switch in a simple and efficient way.

Bibliography

[1] Hui Zhang. "Service Disciplines for guaranteed performace service in packet-switching networks", *Invited paper,Proceedings of the IEEE Vol.83, No. 10, pp.1374-1396"*, October,1995.

[2] C. Partridge, *Gigabit Networking*, Addision-Wesley, 1994.

[3] T. E. Anderson, S. S. Owicki, J. B. Saxe, and C. P. Thacker. "High-speed Switch Scheduling for Local-area Networks", *ACM Transactions on Computer Systems, 11(4):319-352*, Nov., 1993.

[4] N. McKeown and J. Walrand. "Scheduling cells in an input-buffered switch", *Electronics Letters, 29(25):2174-2175*, Dec., 1994.

[5] A. Mekkittikul and N. McKeown. "A Practical Scheduling Algorithm to Achieve 100% Throughtput in Input-Queued Switches", *Proc. INFOCOM'98 Vol.2:792-799*, Mar.-Apr. 1998

[6] A. Hung, G. Kesidis, and N. McKweon. "ATM Input-Buffered Switches with the Guaranteed-Rate Property", *Proc. ISCC'98, 331-335*, Jun., 1998

[7] A. Demers, S. Keshav, and S. Shenker. "Analysis and simulation of a fair queueing algorithm", *Journal of internetworking Research and Experience"*,

pages 3-26 (Also in Proceedings of ACM SIGCOMM'89, pp3-12), October 1990.

[8] S. Golestani. "A self-clocked fair queueing scheme for broadband applications", *Proceedings of IEEE INFOCOM'94 pages 636-646*, Toronto, CA, June 1994.

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

[10] A. Parekh. "A generalized processor sharing approach to flow control in integrated services networks- the single node case", *Proceedings of INFOCOM'92"*, 1992.

[11] M. J. Karol, M. G. Hluchyj and S. P. Morgan, "Input versus output queueing on a space-division packet switch", *IEEE Trans. on Commun. Vol. COM-35, No. 12, pp. 1347-1356*, Dec. 1987.

[12] Y. S. Yeh, M. G. hluchyj and A. S. Acampora, "The Knockout Switch: A simple, modular architecture for high-performance packet switching", *IEEE J. Select. Areas Commun., Vol. SAC-5, No.8 pp. 1274-1283*, Oct. 1987.

[13] Y. Oie, M. Murata, K. Kubota and H. Miyahara, "Effect of speedup in nonblocking packet switch", *proceedings of IEEE ICC'89, pp. 410-414*, Jun. 1989.

[14] M. Littleworrd, "Sunshine: A broadband packet switch", *Bell Commun. Res. Tech Rep.*, Sept. 1987.

[15] S. C. Liew and K. W. Lu, "Comparison of buffering strategies for asymmetric packet switch modules", *IEEE J. Select. Areas Commun., Vol 9, No. 3 pp. 428-438*, Apr. 1991.

[16] N. McKeown, V. Anantharam, and J. Warland. "Achieving 100% Throughput in an Input-Queued Switch", *Proc. IEEE INFOCOM '96, 1:296-302*, 1996.

[17] M. G. Hluchyj and M. J. Karol, "Queueing in high-performance packet switching", *IEEE J. Select. Areas Commun., Vol 6, No.9 pp. 1587-1597*, Dec. 1988.

[18] L.Kleinrock. "Queueing Systems, Volume 2: Computer Applications", *Wiley*, 1976.

[19] T. T. Lee and C. H. Lam. "Path Switching - A Quasi-Static Routing Scheme for Large-Scale ATM Packet Switches", *IEEE JSAC, 15(5):914-924*, Jun. 1997.

[20] C. H. Lam. "Virtual Path Traffic Management of Cross-Path Switch", *PhD Thesis, Department of Information Engineering, The Chinese University of Hong Kong*, 1997.

[21] S. Y. Liew and T. T. Lee. "Bandwidth Assignment with QoS guarantee in a Class of Scalable ATM switches", *Proc. IEEE ICC'99. 3:1802-1806.* Jun. 1999.

[22] R. Melen and J. S. Turner, "Nonblocking multirate distribution networks", *IEEE Trans. Commun., Vol 41, pp. 362-369*, Feb. 1993.

[23] Joseph Y. Hui, "Switching and Traffic Theory for Integrated Broadband Networks", *Klwwer Academic*, 1990