# Reconfiguration Issues in a Quasi-Static Packet Switch

BY

MAN WAI-HUNG

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

THE CHINESE UNIVERSITY OF HONG KONG
JULY 2003

The Chinese University of Hong Kong holds the copyright of this thesis. Any person(s) intending to use a part or whole of the materials in the thesis in a proposed publication must seek copyright release from the Dean of the Graduate School.

香港中文大學
大學
圖書館系統
28 APR 2004
UNIVERSITY
LIBRARY SYSTEM
THE CHINESE UNIVERSITY OF HONG KONG

# Acknowledgement

It is with my deepest appreciation that I acknowledge my supervisor, Prof. Tony T. Lee, for his guidance and supervision over the last two years. He has pointed me to the right direction in research with his deep insight, and his encouragement has carried me through the most frustrating moments in my work. Nothing in this thesis would be possible without his invaluable advice. I must thank him for providing me with constant encouragement and constructive criticism.

I would like to take this opportunity to thank Dr. S.Y. Liew, who has given me valuable advice at the beginning of my research work. The discussion with him has been very useful and generated a lot of ideas.

Also I would like to express my thanks to all my friends at Broadband Communication Laboratory, for the friendship and support I received from them. The days with Mr. Mui Sze Wai, Mr. Deng Yun, Mr. Choy Man Ting, Mr. Lam Lui Fuk, Ms. Tam Wing Lam, Mr. Kong Chun Wai and Ms. Ip Hoi Man shall

become an unforgettable memory in my heart. The two years with them will certainly enrich my life in the forth-coming years.

Finally, may I dedicate this thesis to my parents, brothers and friends who have loved and supported me at all times.

# 撮要

路徑選擇策略主要可分爲靜態路徑選擇及動態路徑選擇。由於靜態路徑選擇會犧牲交換機本身的吞吐量，而動態選擇所需的處理器運算也很高，因此兩者均未能符合建構大型交換機的條件。路徑交換策略是一准靜態路徑選擇策略。它通過周期性地改變三級 Clos 網絡實現了靜態和動態路徑選擇策略的綜合。可是要實現建構大型交換機，我們還需要處理建立連接、釋放連接以及特發交通等的問題。總括來說，就是要把路徑按實際需要的分配給各路徑，同時避免動態路徑選擇所引致的高運算。在本篇論文中，我們研究了一個簡單的路徑交換策略。籍著以路徑選擇策略爲基礎，改變當中部份路徑以達致建立連接、釋放連接，處理特發交通等目的。

# Abstract

Routing schemes can be classified into static and dynamic routing conventionally. Static routing, while providing service guarantees to individual calls, sacrifices the utilization of system. Dynamic routing, on the other hand, makes up for this deficiency. However, the price paid is in terms of the computational complexity. It is this complexity that makes dynamic routing scheme unsuitable for high-speed packet switches. In this paper we propose a simple yet efficient algorithm for switches which uses quasi-static routing schemes. By making use of the algorithm, the switch can decide to do the complex route recalculation under certain conditions(e.g. fixed intervals or performance falling under threshold), while using the reconfiguration algorithm as a temporary measure to cater for the requirement change in between the recalculations. Besides, the reconfiguration algorithm can also be used to tackle the sudden burst into the inputs by allocating more capacity to a virtual path.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

As the world wide web (WWW) continues to grow, the enormous traffic generated in the communications network has presented challenges on the existing infrastructure. The next generation network would be required to carry traffic with various characteristics. Thus many researches have focused on building a single network infrastructure which is able to support multi-service traffic. The core, and challenging, issue here is to develop a switching system which could satisfy a diverse Quality of Service (QoS) requirements. As essential as QoS is that the switching system should also be capable of scaling up to very large size and supporting multicasting.

## 1.1 General Types of Switch Architecture

A switching fabric is generally made up from many switching elements. Switching elements are generally smaller in size, and could be implemented on a single board – although this is not necessarily a strict rule. A switching fabric could mostly be used for three functions, header translation, switching and buffering (or contention resolution). In this section we introduce four general types of switching architectures, input-buffered switch, output-buffered switch, crossbar-based switch and shared buffer memory switch.

### 1.1.1 Input-Buffered Switch

An input-buffered switch consists of a space switch with a buffer on every input to prevent contention (Figure 1.1). The buffers are controlled so that no contention arises within the switch. A problem with this type of switching fabric is known as "head-of-line" (HOL) blocking. This happens when a packet is prevented from reaching a free output because other packets which are ahead of it in the buffer cannot be transmitted over the switch due to contention. The maximum throughput of this type of switches is limited to 58% [22], assuming the

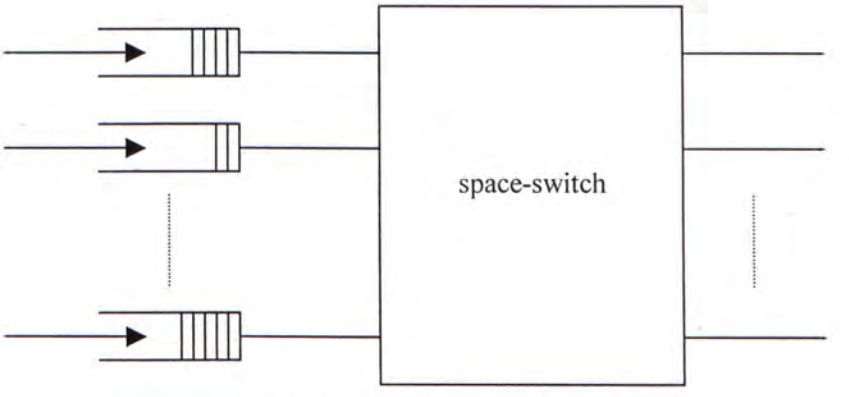



Figure 1.1: An input-buffered switch

number of inputs and outputs tend to infinity and uniform loading.

A three-phase algorithm, also known as request-acknowledge-transmit [23], is often used to resolve contention in an input-buffered switch. The first two phases constitute an overhead, because no packet is transmitted. To relieve HOL blocking, we can iterate through the request-acknowledge phases using the look-ahead scheme [24] to allow packets not at the first of input queues be transmitted. The price paid here is higher contention-resolution overhead, since the request-acknowledgement phases is repeatedly done.

### 1.1.2 Output-buffered switch

Output-buffered switch is similar to input-buffered switch, but with the buffers located on the output side. Output-buffered switch does not have HOL blocking, but the internal bandwidth of the interconnect must be increased to allow multiple packets to pass through at the same time. Many packet scheduling algorithms have been proposed using output-buffered switch for this advantage [25] [26] [27], but output-buffered switch also suffers from an inherent drawback of having to operate at $N$ times the line rate in a $N$-port switch. This attributes to a serious limitation in scaling up output-buffered switches, and is the reason why many researchers stick to input-buffered switches despite the effect of HOL blocking.

### 1.1.3 Crossbar-based switch

A crossbar-based switch consists of a crossbar arrangement with a buffer connecting an input to an output at each crosspoint. In a $N \times N$ switch, we have $N^2$ buffers. Only packets destined to the correct address are passed into the buffer. A typical example of crossbar-based switch is the multi-stage self routing (MSSR) switching element [28]. The increase in the amount of memory used in this type of switches is quite dramatic, scaling up with $N$. Thus crossbar-based switches are

generally not a feasible choice except for small switches.

### 1.1.4 Shared buffer memory switch

In a shared buffer memory switch, the switch keeps a central memory. Incoming packets are read into different queues in the memory according to their outputs. The queues share the central memory, and are allowed to grow as long as the total sum of the queue sizes does not exceed the total memory size. For the reason of a shared memory pool, shared buffer memory switches outperform output-buffered switches for the same amount of memory given.

Figure 1.2 shows the architecture of a shared buffer memory switch. The memory actually contains $N$ linked lists, each of which stores the packet destined to the output port particular output port. The main disadvantage of shared buffer memory switch is its complex control, where the switch is required to keep the linked lists in place. Shared buffer memory switches, like input-buffered switches, also exhibit HOL blocking.

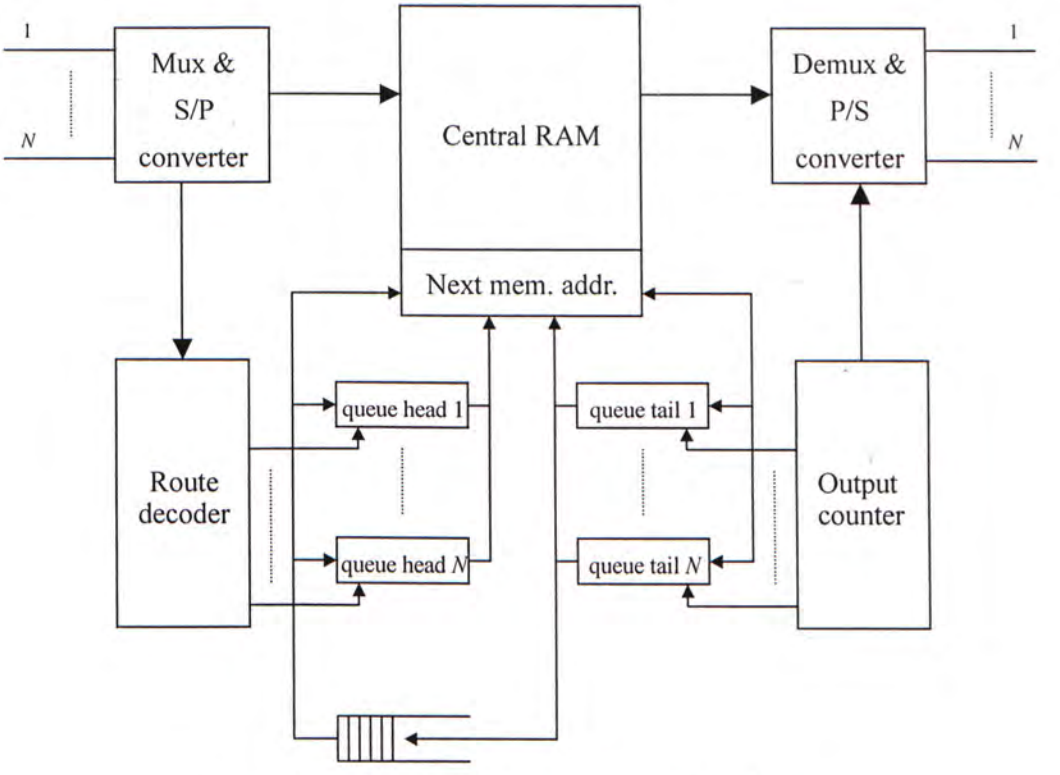



Figure 1.2: A shared buffer memory switch architecture

## 1.2 From Clos Network to Cross-Path Switch

Proposed by C. Clos in 1953, Clos Network [1] is the most important class of interconnection network ever studied. A three-stage Clos Network consists of

three stages of switch modules. Each module in one stage is connected to every module of the next stage. The three-stage Clos Network is rearrangeably non-blocking [2]. By rearranging the existing connections, Clos Network is able to realize any permutations of inputs and outputs.

Figure 1.3 shows a $N \times N$ Clos Network. The first stage consists of $k = N/n$ input modules, each of dimension $n \times m$. There are $m$ central modules, each of dimension $k \times k$. In the third stage, there are again $k$ output modules of dimension $m \times n$.

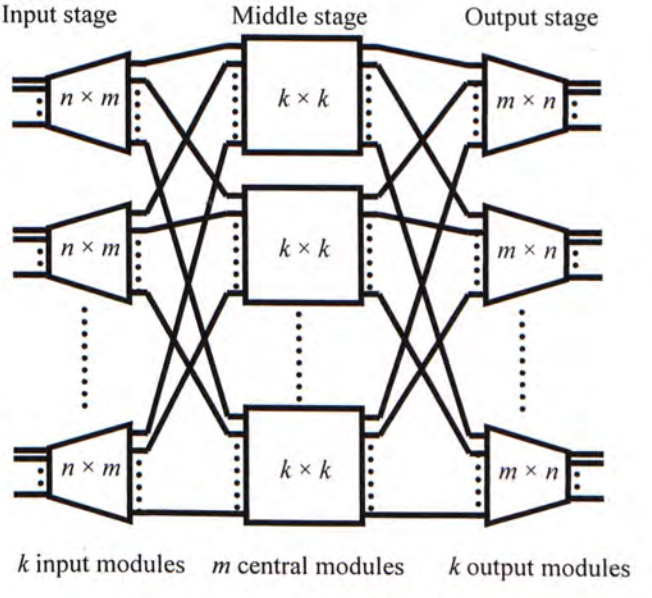



Figure 1.3: A $N \times N$ Clos Network

The original proposal of Clos Network was dedicated to circuit-switching systems, in which a path from an input to an output is established and owned by a connection during the time of conversation. Ever since Asynchronous Transfer Mode (ATM) [19] becomes prevalent as the underlying technology enabling broadband integrated services digital network (B-ISDN), many researches have been done making use of the Clos Network as the model of large scale ATM switch architecture. The major problem of the adoption of Clos Network falls in the path and bandwidth assignment for each connection request. For this purpose two different approaches have been developed. The first is the static routing scheme, such as multirate circuit switching [6], in which a path is established for each connection such that the peak capacity is reserved along the path and it is sufficient to carry the connection at any time. The second is dynamic routing scheme, such as straight matching [7], where the connection pattern is computed on a slot-by-slot basics.

Neither of these schemes serve good for developing a large-scale ATM switching fabric. The static routing scheme, though being able to guarantee the bandwidth for each individual connection, fails to achieve high system utilization because it does not exploit the full advantage of statistical multiplexing gain. On

the other hand, dynamic routing scheme requires a dynamic route assignment. Route assignment within the network is performed by means of control algorithm. In [3], a looping algorithm is proposed for this purpose in Benes Network, which is a Clos Network composed entirely of 2 x 2 switching elements. Some other parallel control algorithms include [4] and [5]. However, these routing algorithms are not feasible in packet switching systems since they require a relatively long period to execute.

The proposal of cross-path switch is motivated under these circumstances. Cross-path switch is built on a 3-stage Clos Network with the path switching [8] algorithm. Path-switching uses a quasi-static routing scheme, which is a compromise of the static and dynamic routing schemes. The routing of path switching is based on the concept of virtual path within the Clos Network. A virtual path is said to comprise of all virtual circuits interconnecting any incoming port and any outgoing port on a particular pair of modules. The scheduling of path switching consists of two steps, the capacity allocation and the route assignment. The capacity allocation is to find the capacity $C_{ij} > \lambda_{ij}$ for each virtual path $(i,j)$, where $\lambda_{ij}$ denotes the requested capacity of the virtual path at call setup. This step can be carried out by optimizing some objective function subject

to $\sum_i C_{ij} = \sum_j C_{ij} = m$. The objective function is chosen with accordance to the stochastic characteristic of the traffic on virtual paths and the quality of services requirements of calls.

In the route assignment stage, the capacity matrix is converted into a finite number, *F*, of regular bipartite multigraphs based on time-space interleaving principle. Each bipartite multigraph represents a particular connection pattern of central modules in the Clos Network. Edge coloring is performed to assign *m* distinct colors to *m* edges of each node such that no two adjacent edges have the same color. It is well known that a regular bipartite multigraph with degree *m* is *m*-colorable [20][21]. Each color corresponds to a central module, and the color assigned to an edge corresponds to a connection in the respective virtual path (see Figure 1.4).

To construct the cross-path switch, any nonblocking, self-routing switch fabric can be used as the building block of the first two stages. The switch modules of the third stage must be capable of resolving output port contentions, and either the Batcher-banyan network (STARLITE) with extended outputs [9] or knockout switch [10] can be employed for this purpose. Path switching is

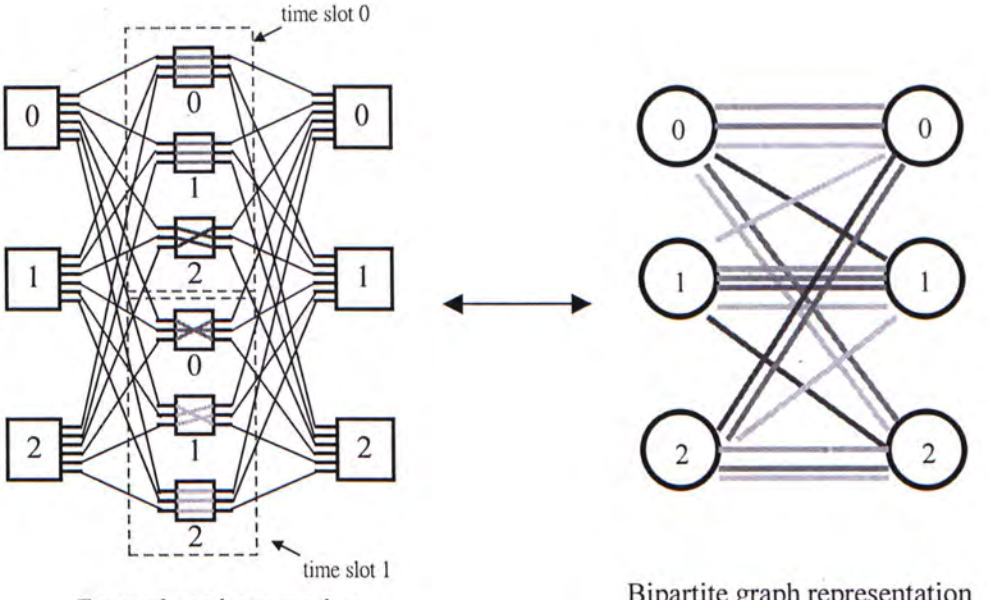



Figure 1.4: Bipartite graph representatation

completely distributed. Global information is still required in the calculation of the routes, but it is not on a slot-by-slot basis. The routing tables calculated can be stored in the local memory of input modules, and be updated only if the traffic matrix changes significantly and the switch performance becomes unacceptable.

## 1.3 Motivation and Organization

Cross-path switch, as we have seen, is a good candidate for building next generation switching fabric. As introduced above, the routing tables are updated only if the switch performance becomes unacceptable. An obvious issue here is if update is required too often, then cross-path switch falls into the same drawback as other packet switches with dynamic routing schemes, when the recalculation takes too much time to be desirable. Note that even with a few call setups, the switch can become unstable since the incoming traffic rate is larger than the service rate. In this thesis, we are going to study this issue and develop an adaptive algorithm which will cater for changes in the traffic matrix.

In Chapter 2 we develop a tools for our algorithm – a one-step route reconfiguration procedure in Clos Network. While looping algorithms can be used to achieve similar purpose, we show that by restricting ourselves to a sub-optimal rearrangement we could actually reallocate a central module to the virtual path in need in one step. This gives the advantage of being simple in the core of the algorithm. In Chapter 3, we develop the frame-based reconfiguration scheme for cross-path switch. The scheme makes use of the one-step reconfiguration

procedure described in Chapter 2, and extra information on the needs of the virtual paths is obtained through two matrices. These two matrices are updated when packets pass through the central switch, thus the algorithm does not rely on external information to calculate the route reallocation. Simulation results are also given in this Chapter. In Chapter 4, we study a more general analysis on frame-based reconfiguration schemes. In particular, we try to establish the maximum packet delay increase resulted by adopting any kind of frame-based reconfiguration algorithms. Chapter 5 concludes our work and provides further research possibilities.

# Chapter 2

# Route Reconfiguration in Clos Network

In this chapter, we will discuss the route reconfiguration issues in Clos Network. The original development of a rearrangeably non-blocking Clos Network requires the number of rearrangements in the connection matrix up to the order of the sizes of the input and output modules. Here, we establish a simple scheme to rearrange the connection matrix in one step, with the tradeoff of being a sub-optimal rearrangement. We will also show the condition under which this scheme can be applied. Since cross-path switch is actually a path-switched Clos Network, the development in this chapter will be useful when we proceed to the frame-based reconfiguration scheme in the following chapters.

## 2.1 Connection Matrix in Clos Network

A Clos Network is characterized by five independent parameters. We have $r_1$, $r_2$ and $r_3$ modules in stages 1, 2 and 3 respectively. The dimensions of the modules are $n_1 \times r_2$, $r_1 \times r_3$ and $r_2 \times n_3$ respectively. For a symmetric switch the number of inputs is equal to the number of outputs, and thus $N = n_1 r_1 = n_3 r_3$, and there are only four independent parameters. The connections of a 3-stage Clos Network can be described in the form of a connection matrix. Figure 2.1 shows the connection state of a switch, and the respective connection matrix. Row $i$ corresponds to first-stage module $i$ and column $j$ corresponds to third-stage module $j$. Entry $(i,j)$ is associated with the middle-stage modules. The figure shows that input module 2 is being switched to output module 4 via three central modules, $A$, $B$ and $C$, therefore entry (2,4) in the connection matrix becomes $\{A,B,C\}$. In other words, entry $(i,j)$ contains the symbols or labels of the middle-stage modules that are used to connect module $i$ in stage 1 to module $j$ in stage 3.

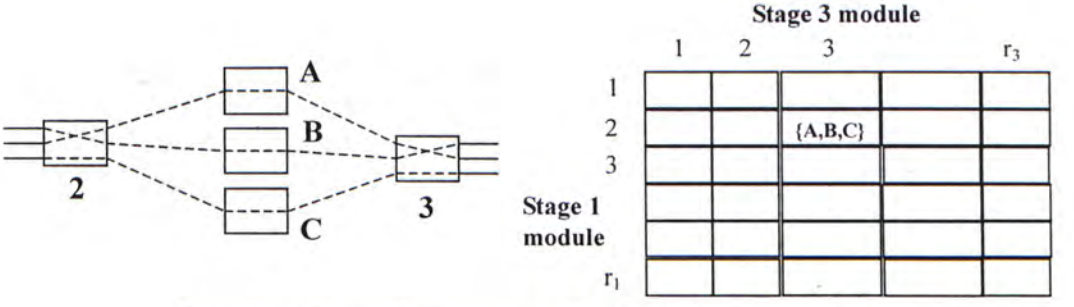



Figure 2.1: Connection pattern and its respective connection matrix

**Definition 2.1 (Conditions of a Legitimate Connection Matrix)** *Let $S_A$ and $S_B$ be the sets of symbols in any row A and column B respectively. Define a legitimate connection matrix as one which satisfies the following conditions:*

1. *Each row can have at most $n_1$ symbols:*

$$|S_A| \leq n_1 \tag{2.1}$$

2. *Each column can have at most $n_3$ symbols:*

$$|S_B| \leq n_3 \tag{2.2}$$

3. *The symbols in each row or each column must be distinct. There can be at most $r_2$ symbols in each row or column.*

$$|S_A|, |S_B| \leq r_2 \tag{2.3}$$

Without making impact to our theme of discussion we ignore the number of inputs $n_1$ and outputs $n_3$ in the following. (2.1) and (2.2) are assumed to be satisfied unless otherwise specified. Notice that a legitimate connection matrix may not fully utilize the capacity of the switch since the central modules may or may not be allocated. In practice the central modules are allocated to some virtual paths even if they are not used. Thus we further restrict ourselves by defining the full-utilization connection matrix as follows.

**Definition 2.2 (Full-utilization connection matrix)** *Let $S_A$ and $S_B$ be the sets of symbols in any row A and column B respectively. A full-utilization connection matrix is defined as one which contains exactly $r_2$ symbols in each row or column. Also, these $r_2$ symbols must be distinct.*

$$i.e. \quad |S_A|, |S_B| = r_2 \tag{2.4}$$

The definition of a full-utilization connection matrix largely reduces the possible set of connection matrix in a Clos Network. As we shall see this definition also introduces difficulty in rearranging central modules in a Clos Network.

## 2.2 Rearranging Central Modules in Clos Network

Suppose that we have a rearrangeably non-blocking Clos Network. An intuitive problem is how could we rearrange the central modules such that all the calls can be satisfied. Figure 2.2(a) shows part of a 4 × 4 connection matrix. The concerned central modules, C and D, are being allocated to different virtual paths. However, since C has been allocated to row 3 while D has been allocated to column 3, the virtual path (3,3) is being deprived of the access of either C or D. Assume that we try to allocate D to (3,3). Thus there is a contention between (1,3) and (3,3) for module D. By allocating module C to (1,3) we resolve the contention between (1,3) and (3,3), but raise another contention between (1,3) and (1,4) for module C. By a similar token we can traverse the chain until we finally have no contention. The resultant connection matrix is shown in Figure 2.2(b). Alternatively we could first allocate C to (3,3). This results in the connection matrix shown in Figure 2.2(c). The number of rearrangements starting with module C or D is different, one of which results in fewer steps.

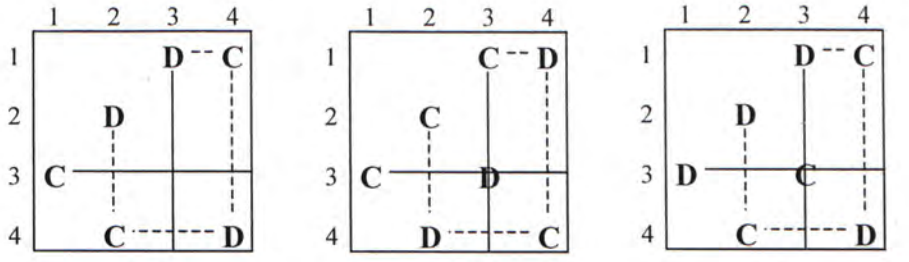



Figure 2.2: Two different ways to rearrange in a 4 x 4 connection matrix

Using this rearrangement technique it is guaranteed that every virtual path in a 3-stage Clos network could be allocated with enough central modules as long as (2.4) is satisfied. The rearrangement complexity, unfortunately, scales up with the number of input and output modules. In [1] it is shown that the number of rearrangements is at most $min(r_1,r_3)$ - 1. As $r_1$ and $r_3$ increase when we build a large switching fabric, this rearrangement technique becomes inappropriate for its purpose.

## 2.3 Changing the Connection Matrix

The section above shows the difficulty in rearranging central modules in Clos Network. In reality, the connection matrix will change under three conditions:

1. A new call being set up.
2. An existing call being released.
3. The incoming traffic having an incoming traffic different from the rate it requested at call setup.

Condition (3) can actually be viewed as a combination of conditions (1) and (2). Thus what we have to deal with is connection setup and release. Now assume that we ignore the issue with system utilization for a moment, so we could save our trouble from having to deal with connection release (because the existing connections are adequate for the requirements of the virtual paths). Using the conventional rearrangement technique as described in the previous section, we conclude the procedure for setting up a new connection as follows, assuming $(i,j)$ be the virtual path requesting the new connection.

1. Check the connection matrix such that

   $$number\ of\ central\ modules\ unused + capacity\ of\ new\ request \leq m$$

   , for row $i$ and column $j$ in the connection matrix.
2. If (1) is not satisfied, the connection is rejected. Otherwise accept the connection and proceed to (3).
3. For each token requested, find an unused central module in either row $i$ or column $j$. Perform chain rearrangements as shown in Section 2.2.

## 2.4 One-Step Route Reconfiguration

In this section we develop a rearrangement technique which allows us to rearrange central modules in a full-utilization connection matrix in one step. The proposed technique is concerned about the virtual path which requests new central module only, and does it at the cost of other virtual paths. By this we mean that some virtual paths will be sacrificed in order to fulfill a request. Therefore we have a sub-optimal reconfiguration scheme instead of an optimal one from Section 2.2. The benefit in return is a lower complexity in the reconfiguration procedure.

Assuming that we are working with a 3-stage Clos Network. Also assume that we have, by some means, determined that virtual path $(i_1,j_1)$ contains surplus capacity while $(i_2,j_2)$ is in lack of capacity. Thus the objective is to reallocate a central module from $(i_1,j_1)$ to $(i_2,j_2)$, which we refer to donor and receiver below. Let $A$ be a central module allocated to $(i_1,j_1)$ but not to $(i_2,j_2)$ so that it is available for rearrangement. We also assume that every central module should be allocated to some virtual path, as stated in Equation (2.4).

**Theorem 2.1 (One-step reconfiguration condition)** *Given that we want to rearrange one central module from virtual path* $(i_1,j_1)$ *to* $(i_2,j_2)$. *The reconfiguration can be completed in one step* (*rearrangement*) *if and only if* $i_1 = i_2$ *or* $j_1 = j_2$.

*Proof*: Let $A$ be the central module to be reallocated.

$\Leftarrow$) We are going to show that if $i_1 \neq i_2$ and $j_1 \neq j_2$, then the reconfiguration takes more than one step to complete. In this case, central module $A$ is allocated to $(i_1,j_1)$, so $A$ does not appear in $(i_2,j_1)$ or $(i_1,j_2)$ of the connection matrix. However, since all central modules are allocated to some virtual paths in the connection

matrix, $A$ should be allocated to some $(i_0,j_2)$ and $(i_2,j_0)$, where $i_0 \neq i_1, i_2$ and $j_0 \neq j_1, j_2$. Therefore, when $A$ is reallocated from $(i_1,j_1)$ to $(i_2,j_2)$, conflicting allocation exists in row $i_2$ ( $(i_2,j_2)$ and $(i_2,j_0)$ both containing $A$ ) and column $j_2$ ( $(i_2,j_2)$ and $(i_0,j_2)$ both containing $A$ ). Thus we need at least two more steps to complete the reconfiguration.

$\Rightarrow$) In the following we describe the details of a one-step reconfiguration. Clearly, module $A$ has to be reallocated from $(i_1,j_1)$ to $(i_2,j_2)$. We also have from above that $i_1 = i_2$ or $j_1 = j_2$. Without loss of generality we assume $i_1 = i_2$. By the property of full utilization module $A$ should have also be allocated to $(i_0,j_2)$ as well. When the reconfiguration occurs, we have conflicting allocation in $(i_0,j_2)$ and $(i_2,j_2)$. This is countered by removing the allocation of $A$ in $(i_0,j_2)$. Thus the virtual path $(i_0,j_2)$ becomes the victim in this one-step reconfiguration. Now module $A$ is not allocated on either row $i_0$ or column $j_1$ in the connection matrix. We complete the reconfiguration by allocating $A$ to $(i_0, j_1)$.

□

To illustrate the theorem, let us consider an example. Figure 2.3(a) shows a 4 x 4 capacity matrix with 3 central modules *A*, *B* and *C*. Now suppose that we want to allocate one central module to (3,2). If (4,1) is going to donate the central module *C*, then the connection in Figure 2.3(b) results. Observe that there are contentions for central module C in both row 3 and column 1. (3,1) and (3,2) both contend for module C in the row, and (2,2) and (3,2) contend for the same module in the column. Thus we need 2 more rearrangements to arrive at Figure 2.3(c).

Indeed, we could compare the reconfiguration procedure if we had chosen (2,2) to donate the central module. Again, central module C is being rearranged. What makes it simple is we could instantly find the owner of module C in row 3, in this case being (3,1), and declare it as the victim. Thus the rearrangements required are putting C in (3,2) and (2,1), while removing them from (2,2) and (3,1). The resultant capacity matrix is shown on Figure 2.3(d).

| | | | |
|---|---|---|---|
| | A,B | | C |
| B | C | | A |
| A,C | | B | |
| | | A,C | B |

| | | | |
|---|---|---|---|
| | A,B | | |
| B | C | | A |
| A,C | C | B | |
| | | A,C | B |

| | | | |
|---|---|---|---|
| C | A,B | | |
| B | | | A,C |
| A | C | B | |
| | | A,C | B |

| | | | |
|---|---|---|---|
| | A,B | | C |
| B,C | | | A |
| A | C | B | |
| | | A,C | B |

Figure 2.3: An example of one-step route reconfiguration

## 2.5 Closing Remarks

In this chapter, we have established an algorithm allowing us to rearrange one central module from a virtual path to another in one step. The algorithm fulfills the requirement of the receiver virtual path, while sacrificing others to attain the purpose. The reduction in complexity comes in two ways. First, only one rearrangement has to be performed. Second, the donor virtual path has to be in the same row or column as the receiver virtual path in order that one-step reconfiguration can be done. This largely reduces the number of virtual paths to consider as the switch size increases.

# Chapter 3

# Frame-Based Reconfiguration Scheme in Cross-Path Switch

In previous chapter we showed how we can perform rearrangement for a Clos Network in order to reallocate a central module from a donor virtual path to a receiver virtual path. The algorithm assumes that the donor and receiver are found by some means. In this chapter we proceed to apply the one-step reconfiguration to a cross-path switch. In particular, we will establish a way to trace the 'progress' of each virtual path in the connection matrix. Using this information, we will then determine the donor and receiver and presents the complete picture of the reconfiguration scheme in cross-path switch. The last section of this chapter gives the results of our simulation.

## 3.1 Route Assignment in Cross-Path Switch

In this section, we define the parameters used to characterize the route assignment in a cross-path switch. A cross-path switch employs a quasi-static routing scheme, and therefore comprises of time slots repeating itself in a periodic manner. The route assignment of the time slots is pre-determined from the requirements of the virtual paths. Thus the requirements of virtual paths are satisfied every frame. So Cross-path switch is said to satisfy the performance guarantees in the long run.

### 3.1.1 Requirement Matrix and Capacity Matrix

In a 3-stage Clos Network of size $k \times m \times k$, where $k$ is the number of input/output modules and $m$ is the number of central modules, we have a total of $k \times k$ virtual paths. Each of these virtual paths requests certain capacity from the switch. Thus we can represent the requirements by a requirement matrix of size $k \times m \times k$.

After the calculation of some sort of route assignment algorithm like the path-switching algorithm, the switch will decide to allocate central modules to the virtual paths. This allocation can be any number, but the switch usually allocates in a way that the requirements are satisfied. Under this condition, the $(i,j)$-*th*

element in the capacity matrix is required to be larger than or equal to the *(i,j)-th* element in the requirement matrix. In a quasi-static routing scheme, each frame is divided into time slots. The switch repeats the route pattern every frame. So the condition becomes that the total sum of capacity is greater than or equal to the total sum of requirement over a number of time slots.

We denote the **requirement matrix** and the **capacity matrix** as ***R*** and ***C*** respectively. $R_{ij}$ and $C_{ij}$ denotes the corresponding *(i,j)-th* element in the matrices. Intuitively the following conditions are satisfied:

$$\sum_i R_{ij} \le m \qquad j = 0, 1, 2, \ldots N\text{-}1 \tag{3.1}$$

$$\sum_i R_{ij} \le m \qquad i = 0, 1, 2, \ldots N\text{-}1 \tag{3.2}$$

$$\sum_i C_{ij} = m \qquad j = 0, 1, 2, \ldots N\text{-}1 \tag{3.3}$$

$$\sum_j C_{ij} = m \qquad i = 0, 1, 2, \ldots N\text{-}1 \tag{3.4}$$

Equations (3.1) and (3.2) are direct result of the capacity restriction in a Clos Network. Equations (3.3) and (3.4) comes from Equation (2.4), requiring that all central modules be allocated.

### 3.1.2 Allocation Vector

An **Allocation Vector** $V$ is the vector containing the connection matrices of every time slot of a frame. The connection matrices state the actual central modules allocated to the virtual paths in a particular time slot. Figure 3.1 shows a 3 *x* 4 *x* 3 cross-path switch with frame size $F = 2$. The requirement matrix $\boldsymbol{R}$ and capacity matrix $\boldsymbol{C}$ are shown, with the allocation vector $V$ and the actual route assignment in time slots 1 and 2

$$\boldsymbol{R} = \begin{bmatrix} 0 & 3 & 3 \\ 1 & 3 & 3 \\ 3 & 1 & 1 \end{bmatrix} \quad \boldsymbol{C} = \begin{bmatrix} 0 & 4 & 4 \\ 2 & 3 & 3 \\ 6 & 1 & 1 \end{bmatrix}$$

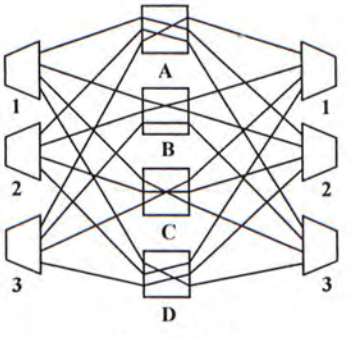
$V =$

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | | A, B | C, D |
| 1 | | C, D | A, B |
| 2 | A, B C, D | | |

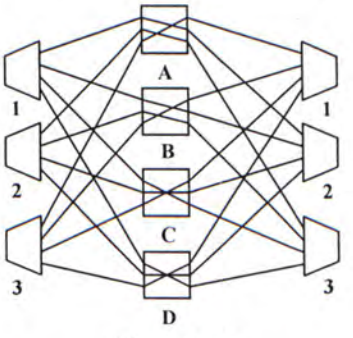
**Time slot 1**

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | | A, B | C, D |
| 1 | B, D | C | A |
| 2 | A, C | D | B |

**Time slot 2**




**Time slot 1**




**Time slot 2**

Figure 3.1: Requirement matrix $\boldsymbol{R}$, Capacity matrix $\boldsymbol{C}$, Allocation vector $V$ and switch connection

## 3.2 Progress Tracing in Cross-Path Switch

Dynamic routing algorithm requires the present information of every inputs and outputs to calculate the route connection, and therefore suffers from complexity issues inherently. One of the approaches to solve this problem is to use previous information instead of present information. By doing this we assume that virtual paths are more likely to be busy if they are presently busy. To gather previous information we mark certain indicators as packets pass through the central switch, thus we do not have to refer to the input and output modules. The information could be used to estimate the requirement of the virtual paths in the coming slot.

Here we introduce two indicator matrices in cross-path switch to aid us in tracing the requirements of the virtual paths. We refer to these two matrices as $\boldsymbol{M_1}$ and $\boldsymbol{M_2}$. $\boldsymbol{M_1}$ and $\boldsymbol{M_2}$ are $k \times k$ matrices, where $k$ is the number of input and output modules. $M_1(i,j)$ denotes the $(i,j)th$ element in the matrix $\boldsymbol{M_1}$. $M_1(i,j)$ is incremented by $1/R_{ij}$ for each packet going from input module $i$ to output module $j$ through the central switch. Likewise $M_2(i,j)$ is incremented by $1/C_{ij}$ for each transmitted packet through $(i,j)$.

Thus the value of $M_1(i,j)$ and $M_2(i,j)$ are calculated as

$$M_1(i,j) = \frac{\#of\ packets transmitted}{R_{ij}} \tag{3.5}$$

$$M_2(i,j) = \frac{\#of\ packets transmitted}{C_{ij}} \tag{3.6}$$

To see how $\boldsymbol{M_1}$ and $\boldsymbol{M_2}$ works we shall look at two cases. The first case is when a virtual path transmits at its requested rate ($R_{ij}$). Over $m$ frames

$$M_1(i,j) = m$$

$$M_2(i,j) = m\left(\frac{R_{ij}}{C_{ij}}\right) = m\rho$$

The second case is when a virtual path transmits at saturation, after m frames

$$M_1(i,j) = m\left(\frac{C_{ij}}{R_{ij}}\right) = m/\rho$$

$$M_2(i,j) = m$$

Thus by reviewing $\boldsymbol{M_1}$ and $\boldsymbol{M_2}$ we could estimate the requirement of the virtual path in the coming slot. Route update can then be done at the end of a frame based on the values of $\boldsymbol{M_1}$ and $\boldsymbol{M_2}$. In the next section, we continue to develop the methodology to choose a donor and receiver.

## 3.3 Implementing Frame-Based Reconfiguration

In this section we propose a frame-based reconfiguration scheme for cross-path switch. Without getting into the detailed discussion we will first outline the algorithm. The following sub-section gives more details in each step of the algorithm.

**General flow of the reconfiguration algorithm:**

1. Calculate the running sum of the mark matrices $\boldsymbol{M_1}$ and $\boldsymbol{M_2}$ over $m$ frames.
2. Find a receiver virtual path.
3. If (2) exists, proceed to find a donor virtual path. If (2) does not exists, proceed to the next frame and go back to (1).
4. If both donor and receiver exist, reconfigure a central module from the donor to the receiver using one-step reconfiguration, otherwise proceed to the next frame and go back to (1).
5. Proceed to the next frame and go back to (1).

### 3.3.1 Recognizing Receiver Virtual Path

Without referring to peripheral information from the input and output modules, we are going to find hints to allocate capacity to virtual paths which are in need of them. As in many packet scheduling algorithms, such as Virtual Clock[11], Fair Queueing (FQ) [12] and its weighted version (WFQ) also called Packetized Generalized Processor Sharing (PGPS) [13], Self-Clocked Fair Queueing (SCFQ) [14], and Worst-case Fair Weighted Fair Queueing ($WF^2Q$) [15], the processor will choose to satisfies the requirements of all the channels first. After the requirements are satisfied, the remaining of capacity is allocated in a proportional manner to the channels.

We apply the same principle here. A backlogged virtual path is a virtual path which has unserved packets awaiting. Therefore, we allocate the unused capacity to facilitate these backlogged virtual paths. As a result, a backlogged virtual path takes over the capacity of other virtual paths which do not need them. Due to this principle, busy virtual paths will take away the capacity of idle virtual paths. If later, the idle virtual paths becomes busy again, we can resume their capacity. So we have two types of receiver candidates. The latter has a priority higher than the

former one because the capacity allocated is smaller than its requirement. Virtual paths which are under-allocated but suffer from backlogging should be dealt with first.

In determining the receiver virtual path out of all backlogged virtual paths, we look for the one with the longest queue size. By doing so we have better ability to deal with bursts at the cost of making it more vulnerable to malicious attacks.

With these observations we conclude the methodology of finding a receiver as follows:

1. Find all backlogged virtual paths.
2. Among these virtual paths, choose those with $R_{ij} > C_{ij}$ such that $C_{ij}/R_{ij}$ is minimum. This minimum virtual path is deemed as the receiver. Resolve randomly if there is more than one.
3. If (2) does not exist, choose the virtual path with the longest queue length from (1).

### 3.3.2 Finding Donor Virtual Path

Virtual paths which hold up capacities while idle contributes to a low system utilization. Reconfigurations are meant to reallocate these capacities to others

which are in need of them. Idle virtual paths are characterized by having # of packets transmitted < $mC_{ij}$, so

$$\frac{\#\ of\ packets\ transmitted}{C_{ij}} = M_2(i,j) < m \tag{3.7}$$

On the other hand, we could still find a donor even if none of the virtual paths is idle. Virtual paths being given more tokens than they requested, which we refer to as greedy virtual paths, should be deprived of the extra tokens when others are in need of them.

Greedy virtual paths, by definition, have # of packets transmitted > $mR_{ij}$, so

$$\frac{\#\ of\ packets\ transmitted}{R_{ij}} = M_1(i,j) > m \tag{3.8}$$

Now we can summarize the procedure of finding donor virtual path as follows, assuming ($i_0$,$j_0$) be the selected receiver: Note that the donor should be in the same row or column as the receiver, as we have shown in Chapter 2.

1. Find $min(M_2(i,j))$ where $M_2(i,j) < m$ and $i = i_0$ or $j = j_0$ in the connection matrix.
2. If (1) does not exist, find $max(M_1(i,j))$ where $M_1(i,j) > m$ and $i = i_0$ or $j = j_0$ in the connection matrix.

## 3.4 Simulation Results

To evaluate the performance of the proposed reconfiguration scheme we choose to perform a simulation. This section concludes the results from the simulation, and gives us further insight in the effectiveness of the proposed scheme.

In our simulation we have a switch of size 3 $x$ 4 $x$ 3. The 3 $x$ 3 = 9 virtual paths request a random amount of capacity $R_{ij}$ from the switch, and the switch will only accept the call under condition (3.1) and (3.2). Incoming packets are generated to the inputs of the virtual paths as Poisson process with rate $R_{ij}$. The packets pass through a token bucket, and get queued at the input modules.

### 3.4.1 Fixed Requirement Matrix

Table 3.1 shows the simulation results of frame-based reconfiguration scheme under fixed requirement matrix. The virtual paths have a fixed requirement matrix, but with different initial capacity matrix. The frame size $F$ of the switch is 3, and m = 3. The first major column shows an ideal initial capacity matrix. In this case, since the switch performs ideally and a low mean delay (1.62) is observed. On the other hand, starting with the same ideal capacity matrix, reconfiguration scheme

will actually hurt the overall system performance (mean delay = 1.80). Fortunately, the achievable mean delay is close to the ideal value, and shows that reconfiguration scheme can meet an acceptable level. In the second major column, we randomly choose an initial capacity matrix such that the requirements are still satisfied. Since the capacity matrix is no longer ideal, we observe a serious performance degradation (mean delay = 3.68). However, in this case, reconfiguration scheme could help us to rectify the capacity allocation. The encouraging result is that out scheme does not only improve the performance, but also pushes it close to the ideal case (mean delay = 1.85).

| | Ideal initial cap. Matrix | | Random initial cap. Matrix | |
|---|---|---|---|---|
| Reconfigure? | Yes | No | Yes | No |
| Req. matrix | $\begin{bmatrix}3&1&2\\2&2&2\\2&4&3\end{bmatrix}$ | $\begin{bmatrix}3&1&2\\2&2&2\\2&4&3\end{bmatrix}$ | $\begin{bmatrix}3&1&2\\2&2&2\\2&4&3\end{bmatrix}$ | $\begin{bmatrix}3&1&2\\2&2&2\\2&4&3\end{bmatrix}$ |
| Initial cap. matrix | $\begin{bmatrix}6&3&3\\3&4&5\\3&5&4\end{bmatrix}$ | $\begin{bmatrix}6&3&3\\3&4&5\\3&5&4\end{bmatrix}$ | $\begin{bmatrix}7&2&3\\3&5&4\\2&5&5\end{bmatrix}$ | $\begin{bmatrix}7&2&3\\3&5&4\\2&5&5\end{bmatrix}$ |
| Final cap. matrix | $\begin{bmatrix}4&4&4\\4&4&4\\4&4&4\end{bmatrix}$ | $\begin{bmatrix}6&3&3\\3&4&5\\3&5&4\end{bmatrix}$ | $\begin{bmatrix}7&2&3\\3&6&3\\2&4&6\end{bmatrix}$ | $\begin{bmatrix}7&2&3\\3&5&4\\2&5&5\end{bmatrix}$ |
| Mean delay | 1.80 | 1.62 | 1.85 | 3.68 |

Table 3.1: Performance of reconfiguration algorithm under fixed traffic requirements

### 3.4.2 Time-Varying Requirement Matrix

Under time-varying requirement matrix, if the capacity matrix remains unchanged, then obviously some virtual paths will become unstable because eventually we will have $R_{ij} > C_{ij}$. This scenario gives the same situation as if connections are set up or released. Basically, the capacity matrix should change to reflect the varying need of the virtual paths.

Figure 3.2 shows a graph of average delay against input load under time-varying requirement matrix. The requirement matrix is randomly created every 100 frames, and is allowed to run under reconfiguration algorithm. Observe that the ideal uniform traffic curve actually represents a lower bound on the system performance, since we can simply allocate the capacity evenly to all virtual paths and no other allocation will out-perform it. With reconfiguration scheme applied, though the requirement matrix varies, the scheme is able to bring the mean delay close to the ideal case (while not optimal). Comparing to the case without reconfiguration, the mean delay gradually shoots up as the input load increases. The figure shows that the reconfiguration algorithm can be used under the situations that connections are to be set up or released in the virtual paths. This saves the complexity of having to recalculate the capacity allocation.

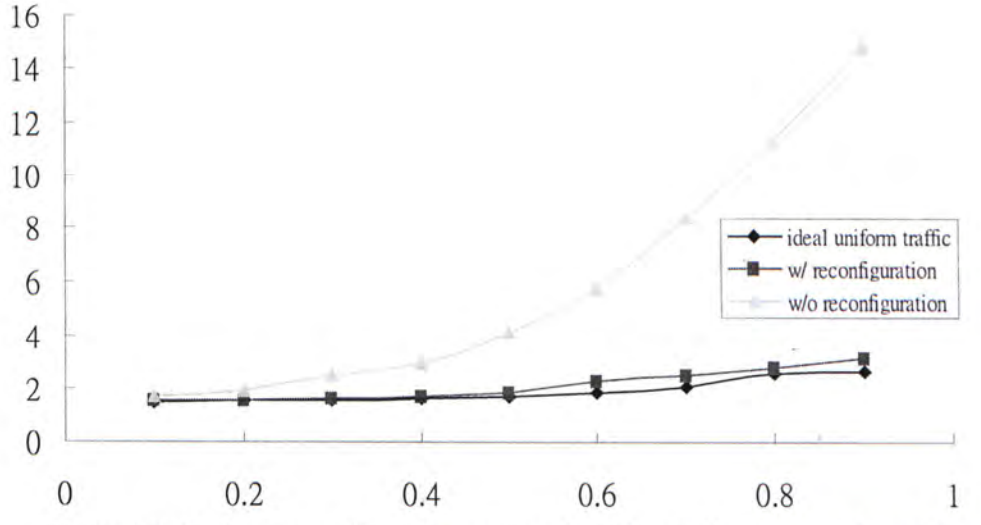



Figure 3.2: Performance of reconfiguration algorithm under time-varying traffic

## 3.5 Deadlocks in reconfigurations

Using one-step reconfiguration, we are able to gain simplicity in rearranging a central module from a donor to a receiver. However, it is also obvious that the rearrangement hurts others by introducing a victim virtual path. This victim path, regardless of being too busy or idle, takes one less token than it is used to.

There are cases where reconfiguration algorithms fall into deadlocks. It results in the choice of donor and receiver swapping back and forth. This can be recognized

from the decrease in system performance. Consider the example shown in Figure 3.3. In Figure 3.3(a) we see that virtual path (1,1) is in need of new tokens, since it requested a capacity of 1 but given zero token. Virtual path (2,1) becomes the donor, and by default (1,2) becomes the victim and (2,2) is the benefited virtual path. The resulting route assignment is shown on Figure 3.3(b). During the next frame, virtual path (1,2) becomes the receiver (in the previous frame it was the victim), virtual path (2,2) becomes donor, (1,2) becomes the victim and (2,1) becomes the benefited virtual path. The resulting configuration gives back the route assignment in Figure 3.3(a).

It would seem that this is the most severe barrier in the adoption of frame-based reconfiguration schemes. But the fact is it only happens when the switch is fully loaded. Therefore 2 approaches to avoid these deadlocks are:

1. Avoid full utilization in the switch.
2. Observe the mean delay (maximum packet delay). Recalculate the route assignment using path-switching algorithm when it reaches a certain threshold.

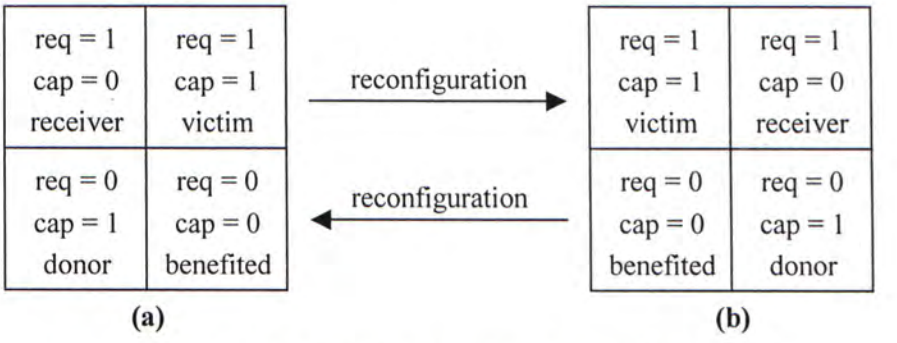



Figure 3.3: An example of deadlock in reconfigurations

## 3.6 Closing Remarks

In this chapter we employed the one-step route reconfiguration that we developed in chapter 2 in cross-path switch. This leads to our frame-based reconfiguration scheme. Though being sub-optimal compared to path-switching calculated assignment, we recognize immediately the advantages of doing so. First, variations of incoming traffic, induced by call setup, call release or traffic bursts, are catered to without recalculation of route assignment. Second, the system utilization is increased as idle virtual paths donate their central modules to the busy paths. We also showed by simulation that the reconfiguration scheme is efficient under both static and time-varying input traffic. To further improve the

usefulness of frame-based reconfiguration scheme, the path-switching algorithm is used jointly with the reconfiguration scheme. Route assignment is calculated periodically based on the path-switching algorithm to give an optimal allocation, while the reconfiguration scheme is used to support the requirements between recalculations. In some cases, frame-based reconfiguration schemes can make unfavourable reconfigurations and fall into instability. This is countered by either not fully-loading the switch, or by recalculating the route assignment with path-switching algorithm when the system performance falls below a certain threshold.

# Chapter 4

# Performance and Delay Tradeoff in Frame-Based Reconfiguration Scheme

The core issue of any reconfiguration scheme is to reallocate unused capacity from idle virtual paths to busy virtual paths. Thus we raise the system utilization and robustness against burstiness by reconfiguring the central modules. This does not come for free. The cost is the increase in maximal packet delay. By considering a simplified model, we will derive in this section the increase in the maximum packet delay by employing frame-based reconfiguration scheme, then we will introduce some parameters to the reconfiguration scheme so that we can bound the maximum delay increase to a value of our choice.

## 4.1 Service Curve and Cross-Path Switch

Before we begin with our discussion we will first characterize the service constraint. Suppose that session $i$ traffic is fed to the network which may be shared by other sessions. Let $R^i_{out}(\tau,t)$ be the amount of session $i$ traffic output from it during the interval $(\tau,t)$. The service constraint to each session is defined as follows.

**Definition 4.1 (Service Constraint)** *Let $S_i(\cdot)$ be a non-decreasing non-negative function, where $S_i(0) = 0$. The network element guarantees session i a service curve of $S_i$ if for any t, there exists maximum $\tau \le t$ such that*

$$R^i_{out}(\tau,t) \ge S_i(t-\tau), \tag{4.1}$$

*given that there is no session i packet stored in the network element at time $\tau$.*

In other words, a service curve gives the minimum amount of service provided by a network element to a session when it is busy. From the definition we observe that the service curve $S_i$ is always at the right side of the traffic output $R^i_{out}$.

Figure 4.1 shows an example of a service curve $S_i$ for session $i$ in its busy period. It is a straight line with slope $r_i$, which is the reserved service rate of the session. Stiliadis and Varma introduced in [16] a general class of packet scheduling servers, called Latency-Rate ($\mathcal{LR}$) servers. $\mathcal{LR}$ servers are characterized by two parameters, namely the latency $\theta$ and the allocated rate $r_i$. The latency of a $\mathcal{LR}$ server is the worst-case delay as seen by a packet of session $i$. Thus $\mathcal{LR}$ servers provides service guarantees of $(\theta,r_i)$. Examples of LR servers include Weighted Fair Queueing [12], VirtualClock [11], Self-Clocked Fair Queueing [14], Weighted Round Robin [17], Deficit Round Robin [18], exhibit this property and therefore belong to this class. Cross-path switch also belongs to the class of LR servers and therefore guarantees a session of service $(\theta,r_i)$.

## 4.2 Service Curve of Cross-Path Switch under Reconfiguration

When reconfiguration scheme is applied to a cross-path switch, a virtual path (session) $i$ can be deprived of all its capacity under long duration of idle period. If it becomes backlogged in a later time, the reconfiguration scheme will reconfigure

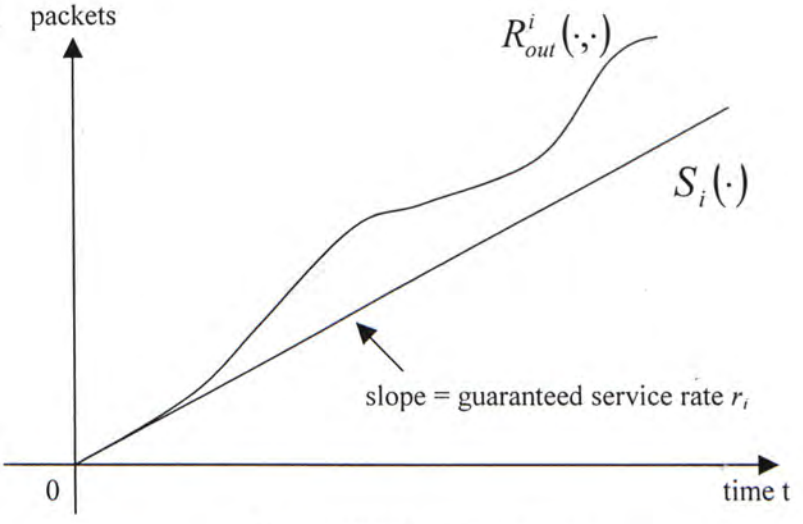



Figure 4.1: A service curve

central modules to it until the requirement of the virtual path is satisfied. Figure 4.2 shows an example of the service curve under reconfiguration scheme. The virtual path is guaranteed a service $(\theta,r_i)$ when no reconfiguration is applied. However, as reconfiguration scheme is used, the session could start from zero capacity. When the virtual path becomes busy, central modules are gradually allocated to this virtual path until it attains the requested rate. Thus the slope increases until it equals that without reconfiguration.

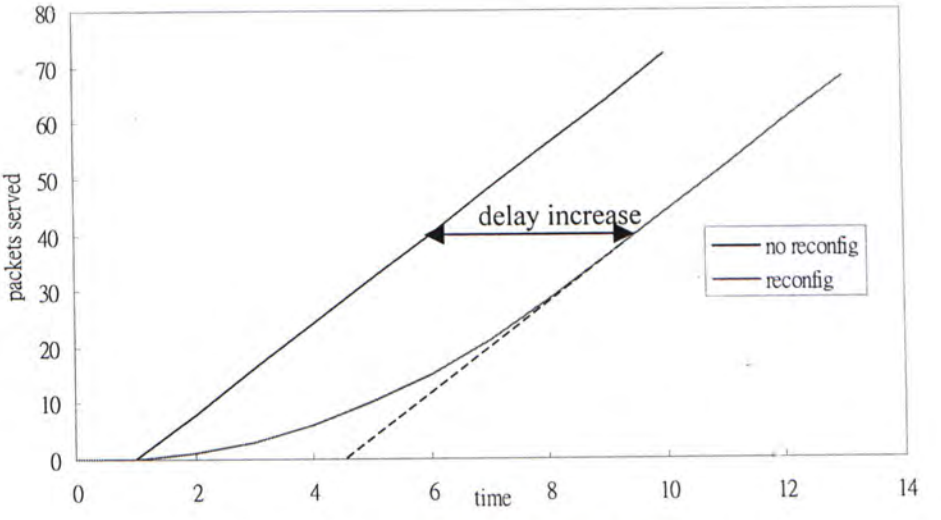



Figure 4.2: Service curve of cross-path switch

Note that the horizontal distance between the curves represents the delay increase due to reconfiguration being applied. The delay increase attains it maximum value when the slope equals, i.e. when all reconfigurations are done. The delay increase remains constant thereafter as long as the virtual path does not get capacity more than it requested.

## 4.3 Impact of Reconfiguration Algorithms to Maximum Delay Increase

Reconfiguration algorithms improve the utilization of the switch. At the same time the switch achieves better average delay. However, since central modules are being reconfigured from idle virtual paths to busy virtual paths, the inability to immediately reallocate back the central modules to the virtual paths which were previously idle induces an increase in the maximum delay of cross-path switch. In this section we first show the maximum delay increase induced by reconfiguration algorithms. In the second part we add constraints to our reconfiguration algorithm to bound the delay to an arbitrary value.

**Theorem 4.1 (Maximum Delay Increase of Reconfiguration Algorithms)** The frame-based reconfiguration algorithm induces a maximum delay increase of

$$\frac{1}{2}(Fm+k)+k \tag{4.2}$$

where $F$ is the frame size, $m$ is the number of central modules and $k$ is the number of input/output modules.

*Proof*: To simplify the model, first assume that only 2 virtual paths are sharing the

resource, and the virtual path that we are concerned with has requested $C$ time slots per frame. The frame size is $F$ and the switch size is $k$ x $m$ x $k$. The worst-case delay increase happens when our virtual path is being taken away with all the capacity that it was allocated, and start to transmit at its requested rate.

Consider the *n-th* packet being served after the virtual path goes busy. When no reconfiguration is introduced the virtual path transmit at a rate of $C$, and thus the *n-th* packet is served in the $\lceil n/C \rceil$ frame after the latency of the concerned server.

On the other hand, if a virtual path is being stripped of all its capacity, then it starts to serve zero packet in the first frame, one packet in the second frame and so on. Thus the number of packets served over $d$ frames

$$= \begin{cases} 0+1+2+...+(d-1) & ,d \le C \\ 0+1+2+...+C+C(d-(C+1)) & ,d > C \end{cases} \tag{4.3}$$

We focus on the case $d > C$ because it is when the maximum delay occurs from the observation in Section 4.2. To proceed, note that the *n-th* packet departs when the number of packets served $\geq n$.

$$\begin{aligned} \textit{i.e. } & 0+1+2+\ldots+C+C\left(d-\left(C+1\right)\right)\geq n \\ & \frac{1}{2}C\left(C+1\right)+C\left(d-\left(C+1\right)\right)\geq n \\ & d\geq \frac{n}{C}+\frac{C+1}{2} \end{aligned} \tag{4.4}$$

The delay increase is given by *D(n)*, where

$$\begin{aligned} D\left(n\right) &= d-\left\lceil \frac{n}{C} \right\rceil \\ &\leq \left\lceil \frac{n}{C}+\frac{C+1}{2} \right\rceil-\left\lceil \frac{n}{C} \right\rceil \\ &\leq \left\lceil \frac{C+1}{2} \right\rceil \end{aligned} \tag{4.5}$$

As the assumption suggests this only counts the competition between 2 virtual paths. In reality, there are $k$ virtual paths competing the same resource ($Fm$ central modules), where $k$ is the number of input/output modules of the switch.

Assuming that a correct decision is made for every reconfiguration, such that the route allocation converges.

Thus the worst-case delay

$$
\begin{aligned}
&= \left\lceil \frac{C_1 + 1}{2} \right\rceil + \left\lceil \frac{C_2 + 1}{2} \right\rceil + \ldots + \left\lceil \frac{C_k + 1}{2} \right\rceil \\
&\le \left( \frac{C_1 + 1}{2} \right) + 1 + \left( \frac{C_2 + 1}{2} \right) + 1 + \ldots + \left( \frac{C_k + 1}{2} \right) + 1 \\
&= \frac{1}{2}\left[ (C_1 + C_2 + \ldots + C_k) + k \right] + k \\
&\le \frac{1}{2}(Fm + k) + k
\end{aligned}
\tag{4.6}
$$

□

This worst-case happens when we have all but one virtual path being idle for some time, causing the reconfiguration algorithm to allocate all central modules to the busy path eventually. And then all in a sudden, these idle virtual paths begin to transmit simultaneously, and our concerned virtual path is being reallocated with its requested capacity at the very last.

The delay increase, as shown above, scales up as the switch size increases. It is not desired for the purpose of providing QoS. In many cases, we would like to bound the maximum delay increase to a value of our choice.

In the study of bounding the maximum delay increase, it can be observed that the extra delay by employing the reconfiguration algorithm comes from the inability to recover the requested capacity for a virtual path after the path has given up the capacity. Thus, the delay increment can be restricted in two ways:

1. By not allowing any particular virtual path to give up too much of its capacity. In other words, we require $C/R > \alpha$, where $0 \leq \alpha < 1$.
2. By speeding up the reallocation rate when a need arises. This can be done by reallocating multiple central modules in one reconfiguration. That is, we could reallocate $b$ central modules from the chosen donor to the receiver in every reconfiguration, where $b > 0$.

**Theorem 4.2 (Maximum Delay Increase of Reconfiguration Algorithms under Constraints $\alpha$, $b$)** The frame-based reconfiguration algorithm, when applied under the conditions $C/R > \alpha$ at any time and reconfigured in batch size of $b$, attains a maximum delay increase of

$$\frac{1}{2b}\left[(1-\alpha)^2 Fm + (1-\alpha)kb\right] + k \tag{4.7}$$

*Proof*: We follow the same line of derivation as shown in Theorem 4.1. Let $\beta$ be the greatest integer such that

$$C - \beta b > \alpha C \tag{4.8}$$

Equation (4.8) describes our reconfiguration controls, which reallocate $b$ central modules in one shot and require $C/R > \alpha$. Notice that an idle virtual path can, in the worst case, recover the requested capacity in $\beta$ frames.

Consider again at frame $d$, if $d > \beta$, # of packets served

$$\begin{aligned} &= (C - \beta b) + (C - (\beta - 1)b) + \ldots + C + C(d - (\beta + 1)) \\ &= \frac{1}{2}(C + C - \beta b)(\beta + 1) + C(d - (\beta + 1)) \end{aligned} \tag{4.9}$$

Again, the *n-th* packet departs if

$$\begin{aligned} &\frac{1}{2}(C + C - \beta b)(\beta + 1) + C(d - (\beta + 1)) \geq n \\ &d \geq \frac{n}{C} + \frac{1}{2C}[2C(\beta + 1) - (2C - \beta b)(\beta + 1)] \\ &= \frac{n}{C} + \frac{1}{2C}\beta b(\beta + 1) \end{aligned} \tag{4.10}$$

So the delay of the *n-th* packet, *D(n)* is given by

$$\begin{aligned} D(n) &= d - \left\lceil \frac{n}{C} \right\rceil \\ &\le \left\lceil \frac{n}{C} + \frac{1}{2C}\beta b(\beta+1) \right\rceil - \left\lceil \frac{n}{C} \right\rceil \\ &\le \left\lceil \frac{1}{2C}\beta b(\beta+1) \right\rceil \end{aligned} \tag{4.11}$$

From (4.8),

$$\beta = \left\lfloor \frac{C-\alpha C}{b} \right\rfloor < \frac{C-\alpha C}{b} \tag{4.12}$$

Substituting (4.12) into (4.11) we have

$$\begin{aligned} D(n) &\le \left\lceil \frac{1}{2C}\left(\frac{C-\alpha C}{b}\right) b\left(\left(\frac{C-\alpha C}{b}\right)+1\right) \right\rceil \\ &= \left\lceil \frac{1}{2b}(1-\alpha)(C(1-\alpha)+b) \right\rceil \end{aligned} \tag{4.13}$$

The worst-case delay, when considering $k$ virtual paths competing the same resource, becomes

$$\begin{aligned}
&\left\lceil \frac{1}{2b}(1-\alpha)(C_1(1-\alpha)+b) \right\rceil + \left\lceil \frac{1}{2b}(1-\alpha)(C_2(1-\alpha)+b) \right\rceil + \ldots + \left\lceil \frac{1}{2b}(1-\alpha)(C_k(1-\alpha)+b) \right\rceil \\
&\leq \frac{1}{2b}(1-\alpha)[(C_0+C_1+\ldots+C_k)(1-\alpha)+kb]+k \\
&= \frac{1}{2b}\left[(1-\alpha)^2 Fm+(1-\alpha)kb\right]+k
\end{aligned} \tag{4.14}$$

□

We double-check our derivation by substituting $\alpha = 0$ and $b = 1$, which reduces to the case when there is no reconfiguration control. After substitution we have the worst-case delay given by

$$\frac{1}{2}(Fm+k)+k$$

,which agrees with our result in Theorem 4.1.

## 4.4 Numerical Example

To illustrate our results, let us consider a cross-path switch with size 32 $x$ 48 $x$ 32. To satisfy the rearrangeably non-blocking condition of Clos Network, the number of central modules must be greater than or equal to the number of input/output ports at the first stage/third stage modules. Thus in effect we could have 32 $x$ 48 = 1536 input/output ports and the switch we are considering is a 1536 x 1536 switch. The frame size is set to 3. Using the results in the previous section, we obtain Figure 4.3. Notice that since there are 32 input modules, and only one module will be taken care of in one cycle, the minimum achievable delay is 32. $\alpha$ can actually be viewed as the proportional of "unreconfigurable" central modules in the switch. As $\alpha$ increases, we choose to make more "static tokens" to the virtual paths. Thus the value of maximum delay decreases. On the other hand, if we raise the value of $b$, we use less time to rearrange the "dynamic tokens". In a similar manner we shorten the maximum delay.

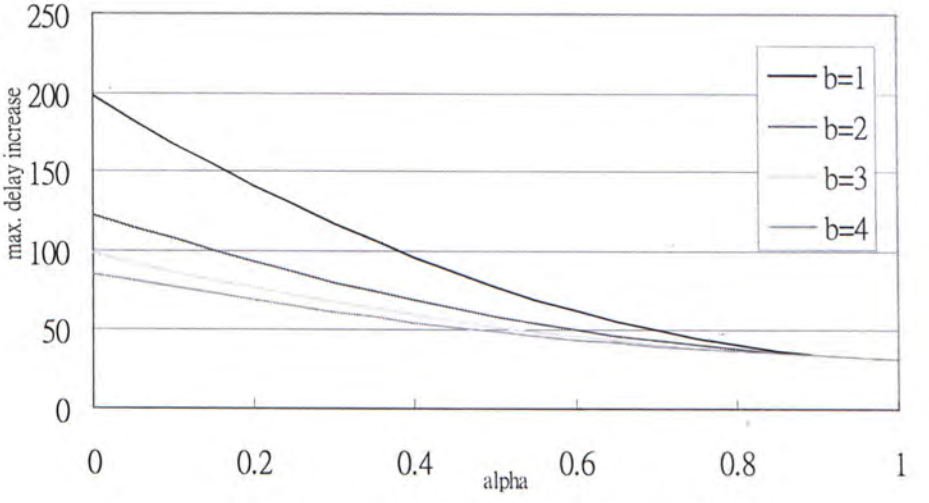



Figure 4.3: max. delay increase v.s. alpha

## 4.5 Closing Remarks

In this chapter we established, in general case, the maximum delay increase induced by employing the class of reconfiguration schemes in cross-path switch. The derivation does not concern how the actual reconfiguration is performed. Instead, for any reconfiguration scheme which (1) reconfigures in every frame, (2) reconfigure only one central module in every reconfiguration, (3) chooses the correct module to reconfigure, the derived bound holds. We also developed two

constraints in which we could bound the maximum delay increase. The derived bounds can serve as a guidance when we develop any type of frame-based reconfiguration scheme in cross-path switch.

# Chapter 5

# Conclusions and Future Research

In this thesis, we have developed a frame-based reconfiguration scheme for cross-path switch and established the maximum delay increase in adopting such kind of frame-based algorithms. A rather surprising result from our work is we actually do not have to start with a decent route configuration and yet we can still take the reconfiguration procedure and the results will still be fine. We also saw that there are cases where the algorithm does not converge. This is recognized from the increase of the mean delay of packets, and can be viewed as time when a total recalculation of the path-switching algorithm is required.

Reconfiguration schemes are meant to reallocate capacities from idle virtual paths to busy virtual paths, therefore as we could see, with the adoption of our algorithm, the average delay is actually lowered, while the maximum delay is

increased. Therefore, although we observed maximum delay increase from our simulation and analysis, the algorithm actually gives the switch higher utilization, and thus lower delay, in most cases. Sometimes, with very adverse situations, when we have no knowledge of the incoming traffic, we would like to be more pessimistic on the delay bounds. Under such condition we could impose some controls on the reconfiguration scheme, such that the maximum packet delay increase is bounded to a value of our choice. This makes the reconfiguration algorithm very viable for application in next generation switches as being able to support diverse QoS requirement in the switching element.

## 5.1 Suggestions for Future Research

The path-switching algorithm calculates the route assignment by optimizing certain objective function under the full-utilization capacity constraints. Therefore, the objective function is actually closely related to the route reconfiguration. In this thesis, we assumed that we have no knowledge of this objective function, and instead used the traffic we saw on the fly to estimate the forth-coming traffic pattern. Indeed, it is possible that we can further push the performance of the

reconfiguration scheme by taking the objective function into consideration as well.

Multicast QoS guarantee is another issue in cross-path switch. The discussion in this thesis is based on individual virtual paths. Thus, while QoS can still be guaranteed in some ways, multicasting virtual paths are not considered. It would be interesting to see the reconfiguration schemes being extended to include multicast traffic in it.

# Bibliography

[1] C. Clos, "A Study of Non-Blocking Switching Networks," *Bell Sys. Tech. Jour.*, Mar. 1953, pp. 406-424.

[2] V.E. Benes, "On Rearrageable Three-Stage Connection Networks," *Bell Sys. Tech. Jour.*, Sept. 1962, pp.1481-1492.

[3] D.C. Opferman and N.T. Tsao-Wu, "On a Class of Rearrangeable Switching Networks Part I: Control Algorithm", *Bell Sys. Tech. Jour.*, May-Jun. 1971, pp. 1579-1600.

[4] D. Nassimi and S. Sahni, "Parallel Algorithms to Set Up the Bnes Permutation Network," *IEEE Trans. Comput.*, Vol. C-31, Feb. 1982, pp. 148-154.

[5] T.T. Lee and S.Y. Liew, "Parallel Routing Algorithms for Clos Network", *Proc. IEEE INFOCOM'96*, 1996, pp. 279-286.

[6] R. Melen and J.S. Turner, "Nonblocking multirate distribution networks",

*IEEE Trans. Commun.*, vol. 41, pp. 362-269, February 1993.

[7] K.Y. Eng, M.J. Karol, and Y.S. Yeh, "A growable packet (ATM) switch architecture: Design principles and applications", *Proc. IEEE GLOBECOM'89.*

[8] T.T. Lee and C.H. Lam, "Path Switching – A Quasi-Static Routing Scheme for Large-Scale ATM Packet Switches," *IEEE Journ. Select. Areas Commun*, vol. 15, pp. 914-924, June 1997.

[9] A. Huang and S. Knauer, "Starlite: a wideband digital switch," *in Pro. GLOBECOM'84*, pp. 121-125, November 1984.

[10] Y. S. Yeh, M. G. Hluchyj, and A. S. Acampora, "The knockout switch: a simple architecture for high-performance packet switching," *IEEE J. Select. Areas Commun.*, col. SAC-5, pp. 1274-1283, October 1987.

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

[12] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," *Journal of Internetworking Research and Experience*, pp. 3-36, October 1990.

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

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

[15] J.C.R. Bennett and H. Zhang, "WF2Q: Worst-case fair weighted fair queueing," in *Proceedings of IEEE INFOCOM'96*, pp. 120-128, San Francisco, CA, March 1996.

[16] D. Stiliadis and A. Varma, "Latency-rate servers: A general model for analysis of traffic scheduling algorithms," in *IEEE/ACM Trans. Networking*, vol. 6, no. 5, pp. 611-624, October 1988.

[17] M. Katevenis, S. Sidiropoulos, and C. Courcoubetis, "Weighted round-robin cell multiplexing in a general-purpose ATM switch chip," *IEEE J. Select. Areas Commun.*, vol. 9, pp. 1265-1279, October 1991.

[18] M. Shreedhar and G. Varghese, "Efficient fair que3ueing using deficit round robin," *IEEE/ACM Trans. Networking*, vol. 4, no. 3, pp. 378-385, June 1996.

[19] P. Gonet, P. Adams, and J. P. Courdreuse, "Asynchronous Time-Division

Switching: The Way to Flexible Broadband Communication Networks," *Proc. IEEE 1986 Int. Zurich Sem. Digital Commun.*, Zurich, Switzerland, March 1986.

[20] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays·Trees·Hypercubes*. Los Altos, CA: Morgan Kaufmann, 1992.

[21] R. J. Wilson, *Introduction to Graph Theory*. New York: Academic, 1972.

[22] M. J. Karol, M. G. Hluchyj, S. P. Morgan, "Input Versus Output Queueing in a Space-Division Packet Switch", *IEEE Transaction on Communications*, vol. 35, no. 12, December 1987, pp. 1347-1356.

[23] J. Hui, E. Arthurs, "A Broadband Packet Switch for Integrated Transport", *IEEE Journal on Selected Areas in Communications*, vol. 5, no. 8, October 1987, pp. 1264-1273.

[24] N. Arakawa, A. Noiri, H. Inoue, "ATM Switch for Multi-Media Switching System", *13th International Switching Symposium*, Stockholm, Sweden, May 27-June 1, 1990, paper A7#2, vol. V, pp. 9-14.

[25] S. Keshav, "On the efficient implementation of fair queueing", *Internetworking: Research and Experience*, September 1991, vol. 2, no. 3,

pp. 157-173.

[26] F. P. Kelly, "Effective bandwidths at muliclass queues", *Queueing Systems Theory and Applications*, October 1991, vol. 9, no. 1-2, pp. 5-15.

[27] A. Demers, S. Keshav, S. Shenker, "Analysis and simulation of a fair queueing algorithm", *Internetworking: Research and Experience*, September 1990, vol. 1, no. 1, pp. 3-26.

[28] K. Hajikano, T. Nomura, K. Murakami, "ATM Switching Technologies", *FUJITSU Sci. Tech.*, vol. 28, no. 3, June 1992, pp. 132-140.

[29] M.C. Chan, "Providing Quality of Service Guarantees in Cross-path packet switch"

CUHK Libraries
004076616