Title	Minimizing internal speedup for performance guaranteed optical packet switches
Author(s)	Wu, B; Yeung, KL
Citation	Globecom - Ieee Global Telecommunications Conference, 2004, v. 3, p. 1742-1746
Issued Date	2004
URL	http://hdl.handle.net/10722/45764
Rights	©2004 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

Minimizing Internal Speedup for Performance Guaranteed Optical Packet Switches

Bin Wu and Kwan L. Yeung
Dept. of Electrical and Electronic Engineering
The University of Hong Kong
Pokfulam, Hong Kong
E-mail: {binwu, kyeung}@eee.hku.hk

***Abstract*—Providing QoS guarantee for Internet services is very important. It evokes the issue that packet switches should provide guaranteed performance (i.e. 100% throughput with bounded worst-case delay). Optical switching technology is widely considered as an excellent solution for packet switches in future networks. However, to achieve guaranteed performance in optical packet switches, an internal speedup is required due to the existence of reconfiguration overhead. How to reduce the internal speedup is the main concern for making these switches practical. In this paper, we first derive the internal speedup S as a function of the number of switch configurations N_S and the reconfiguration overhead δ, or $S=f(N_S,\delta)$. We show that the recently proposed ADJUST algorithm is flawed. Based on the internal speedup function we derived, a new algorithm (ADAPTIVE), with time complexity of $O((\lambda-1)N^2\log N)$, is proposed to minimize S.**

Keywords- Optical packet switch, guaranteed scheduling, reconfiguration overhead, speedup.

I. INTRODUCTION

The idea of using optical switching technologies (such as MEMS mirror [1], thermal bubble [2] and waveguide [3]) in the design of packet switches/routers is widely accepted since the recent progress of these technologies brings about scalability, high rate, huge capacity and low power consumption features on economical bases. However, reconfiguring these optical packet switches requires relatively large time period (i.e. reconfiguration overhead) because some operations involved, such as mechanical settling and synchronization, are quite time-consuming.

Some optical switch architectures are introduced [4][5] to provide guaranteed performance (i.e. 100% throughput with bounded worst-case delay) while taking the reconfiguration overhead (denoted by δ) into account. In these architectures, an internal switch fabric speedup S is necessary to compensate for both the reconfiguration overhead δ and the inefficiency of the scheduling algorithm. We can express $S=S_{\min}S_{exact}$, where $S_{\min}$ is required to compensate for the inefficient scheduling, and S_{exact} is required to compensate for δ.

Two algorithms, DOUBLE [4] and ADJUST [5], are recently proposed to schedule traffic in an $N\times N$ optical switch. DOUBLE uses (at most) $2N$ configurations and generates a maximum total reconfiguration overhead of $2\delta N$. The resulting S_{exact} is given by $T/(T-2\delta N)$, where T is the traffic accumulating time (Please refer to the next section for details). To compensate for the inefficient scheduling, $S_{\min}=2$ is necessary. Thus the overall internal speedup required by DOUBLE is $S=S_{\min}S_{exact}=2T/(T-2\delta N)$. ADJUST uses a regulating factor λ (which is a function of δ) to self-adjust the algorithm under different system parameters. ADJUST claims a better performance than DOUBLE. However, we believe that ADJUST algorithm is flawed.

In this paper, we first formulate the internal speedup S as a function of the number of switch configurations N_S required and the reconfiguration overhead δ. We then prove that the recently proposed ADJUST algorithm is problematic. Based on the internal speedup function derived, we propose a new ADAPTIVE algorithm which gives a solution to minimize internal speedup for optical packet switches.

II. ARCHITECTURE

Fig. 1 shows a general architecture of optical packet switches.[1] An internal speedup S is applied to the switch core in order to compensate for δ and guarantee the scheduling performance of the switch.

The architecture works by periodically accumulating incoming traffic over a duration of T time slots to form an $N\times N$ cumulative traffic matrix $\boldsymbol{C(T)}$ and applying time slot assignment (TSA) method to determine a set of N_S configurations to deliver the collected packets. Fig. 2 shows the scheduling procedure in four stages. In the first stage, incoming packets are accumulated in the input buffers over T time slots to construct $\boldsymbol{C(T)}$. Its entry c_{ij} denotes the number of packets

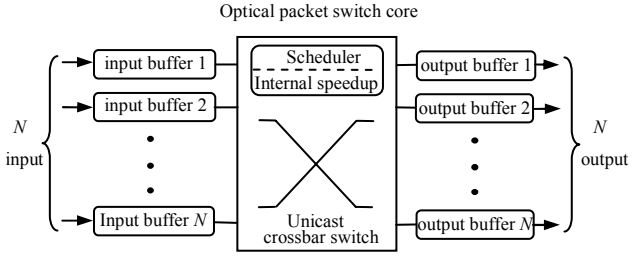

Fig. 1. Optical packet switch architecture

[1] We only consider unicast optical packet switches in this paper.

This research is supported by the Research Grant Council Earmarked Grant 7032/01E, Hong Kong.

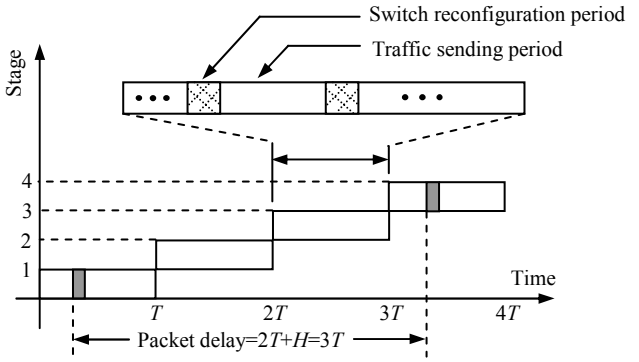

Fig. 2. Optical packet switch scheduling stages

received at input i whose destination is output j. Assume that each time slot can accommodate one packet and all the row sums/column sums of $\boldsymbol{C(T)}$ are not larger than T.

The switch fabric takes H time slots (Fig. 2 assumes $H=T$ for simplicity) in the second stage to generate N_S configurations $\boldsymbol{P_1}$, …, $\boldsymbol{P_{Ns}}$. Rows and columns of $\boldsymbol{P_n}$, $n \in \{1, \ldots, N_S\}$ represent input and output ports of the switch respectively. $\boldsymbol{P_n}$ has at most a single "1" element in each row and column, indicating connection of the two corresponding ports, and other entries are all zeros.

Then the switch is configured and reconfigured in the third stage according to these configurations. When two ports are connected, one packet can be delivered to the corresponding output buffer in each slot. An internal speedup S is required to ensure that this stage occupies only T time slots and all the packets in $\boldsymbol{C(T)}$ are transmitted during this stage. As a result, the length of unit slot time in this stage is shortened because of deploying internal speedup. In order to distinguish slots in this stage with time slots outside, we use a definition of "A-domain slots" to denote the slots produced by the scheduling algorithm (i.e. slots after internal speedup is applied). Each configuration $\boldsymbol{P_n}$ will hold for ϕ_n A-domain slots for packet transmission. The set of configurations $\boldsymbol{P_n}$, $n \in \{1, \ldots, N_S\}$ should cover $\boldsymbol{C(T)}$.[2] Let T_S be the total number of A-domain slots generated by the scheduling algorithm. We have $T_S = \sum_{n=1}^{N_S} \phi_n$.

Finally in the fourth stage packets are sent onto the output lines from the output buffers.

From Fig. 2, it is clear that the traffic delay is bounded to $2T+H$ time slots. Because δN_S time slots have to be assigned to reconfigure the switch for N_S times and thus only $T-\delta N_S$ time slots left for transmitting $\boldsymbol{C(T)}$ in the third stage, a speedup $S_{exact} = T/(T-\delta N_S)$ is necessary to compensate for δ.[3] Since the scheduling algorithm usually produces some empty slots (except EXACT algorithm which produces $N_S=N^2-2N+2$ configurations with $S_{\min}=1$ [4][6]), a minimum speedup given by $S_{\min}=T_S/T$ is required to compensate for the inefficient scheduling. The overall internal speedup S is thus given by

$$S = S_{exact}\, S_{\min} = \frac{T}{T-\delta N_S} S_{\min} = \frac{T_S}{T-\delta N_S}. \qquad (1)$$

III. Characteristics Over the Whole S_{MIN} Versus N_S Design Space

We decompose the cumulative traffic matrix $\boldsymbol{C(T)}$ into a coarse matrix $\boldsymbol{A}$ and a fine matrix $\boldsymbol{R}$ such that

$$\boldsymbol{C(T)} = \frac{T}{N+k}\boldsymbol{A} + \boldsymbol{R},\ a_{ij} = \left\lfloor \frac{c_{ij}}{T/(N+k)} \right\rfloor,\ \boldsymbol{R} = \boldsymbol{C(T)} - \frac{T}{N+k}\boldsymbol{A},$$

where k is an integer in the range of $N^2-4N+2>k>-N$, and a_{ij}, c_{ij} and r_{ij} represent the element (i,j) of $\boldsymbol{A}$, $\boldsymbol{C(T)}$ and $\boldsymbol{R}$ respectively. We first assume that T is a multiple of $N+k$. Since for any $i, j \in \{1, \ldots, N\}$, we have

$$\sum_{i=1}^{N} c_{ij} \le T \ \text{ and } \ \sum_{j=1}^{N} c_{ij} \le T,$$

there must be $c_{ij} \le T$. Thus

$$a_{ij} \le N+k \ \text{ and } \ r_{ij} < \frac{T}{N+k},$$

and the maximum row sum or column sum of $\boldsymbol{A}$ is bounded by $N+k$. In fact,

$$\sum_{i=1}^{N} a_{ij} = \sum_{i=1}^{N} \left\lfloor \frac{c_{ij}}{T/(N+k)} \right\rfloor \le \left\lfloor \frac{\sum_{i=1}^{N} c_{ij}}{T/(N+k)} \right\rfloor \le \left\lfloor \frac{T}{T/(N+k)} \right\rfloor \le N+k.$$

Construct the corresponding bipartite multigraph $\boldsymbol{G_A}$ from matrix $\boldsymbol{A}$. Because the maximum degree of $\boldsymbol{G_A}$ is $N+k$, $\boldsymbol{G_A}$ can be edge-colored in $N+k$ colors [7][8]. Each subset of edges corresponding to a specific color can be mapped back to form a switch configuration. As a result, the coarse matrix $\boldsymbol{A}$ can be covered by $N+k$ configurations with each weighted by 1. The fine matrix $\boldsymbol{R}$ does not need to be explicitly computed because all its elements are guaranteed to be less than $T/(N+k)$. Thus any N non-overlapping perfect matchings,[4] each weighted by $T/(N+k)$, can be used to cover it.

Consequently, any cumulative traffic matrix $\boldsymbol{C(T)}$ can be covered by $N_S=(N+k)+N=2N+k$ switch configurations, each weighted by $T/(N+k)$. $S_{\min}$ of this schedule is

$$S_{\min} = \frac{T_S}{T} = \frac{1}{T}\left[\frac{T}{N+k}(N+k) + \frac{T}{N+k}N\right] = 1 + \frac{N}{N+k}. \qquad (2)$$

Substituting k in (2) by $k=N_S-2N$ yields

$$S_{\min} = 1 + \frac{N}{N_S - N}, \text{ for } N^2-2N+2 > N_S > N. \qquad (3)$$

If T is not a multiple of $N+k$ (general case), we can use $\lceil T/(N+k) \rceil$ to replace $T/(N+k)$. This would increase $S_{\min}$ by at most $(2N+k)/T$. When $T \gg N$ and $T \gg k$, this term can be ignored. For simplicity, we only consider this situation in the

[2] Let $p^{(n)}_{ij}$ and c_{ij} represent the element (i,j) of $\boldsymbol{P_n}$ and $\boldsymbol{C(T)}$ respectively. An $N \times N$ traffic matrix $\boldsymbol{C(T)}$ is covered by a set of switch configurations $\boldsymbol{P_1}$, …, $\boldsymbol{P_{Ns}}$, each weighted by a non-negative integer ϕ_1, …, ϕ_{Ns}, if and only if $\sum_{n=1}^{N_S} \phi_n p^{(n)}_{ij} \ge c_{ij}$ for any $i,j \in \{1, \ldots, N\}$.

[3] Assume that $T > T_{\min} = \delta N_S$. All the parameters ($T$, H, and δ) are counted in time slots. However, T_S is counted in A-domain slots.

[4] $\boldsymbol{P_n}$, $n \in \{1, \ldots, N_S\}$ is called a perfect matching if it has N "1" elements. i.e. in the corresponding bipartite multigraph of $\boldsymbol{P_n}$, each vertex is incident on exactly one edge.

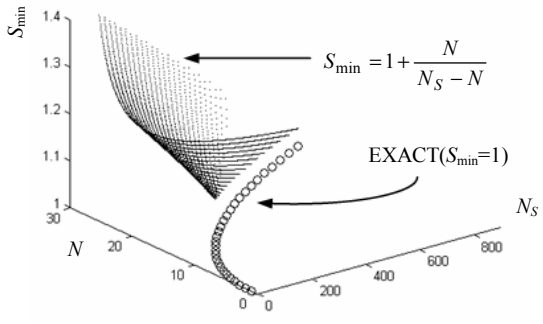

Fig. 3. Illustration of $S_{\min}$ for $N \leq 30$.

following discussion. Equation (3) is for $N^2-2N+2>N_S>N$. In fact, when $N_S=N$, we can use MIN algorithm [4] to schedule $\boldsymbol{C}(\boldsymbol{T})$. When $N_S=N^2-2N+2$, we can apply EXACT algorithm [4][6] so that $S_{\min}=1$ can be achieved.

Fig. 3 illustrates $S_{\min}$ in (3). From (1) and (3), the overall internal speedup S is given by

$$S = \frac{T}{T-\delta N_S} S_{\min} = \frac{T}{T-\delta N_S}\left(1+\frac{N}{N_S-N}\right).$$

If we take T and N as predefined system constants and study the relationship of S versus N_S and δ, S can be formulated as a function of N_S and δ as follows:

$$S = f(N_S,\delta) = \frac{T}{T-\delta N_S}\left(1+\frac{N}{N_S-N}\right) = \frac{TN_S}{(T-\delta N_S)(N_S-N)}. \quad (4)$$

IV. Problems in Adjust Algorithm

An "OSS (Optical Switch Scheduling) problem" is formulated in [5] with the goal of minimizing a covering cost, which is defined as

$$\text{cost} = \sum_{i=1}^{N_S} \phi_i + N_S\delta. \quad (5)$$

Based on this idea, an ADJUST algorithm is proposed, with

$$N_S = \left(\sqrt{T/\delta N}+1\right)N \text{ and } \text{cost}=T+\delta N+2\sqrt{\delta TN}. \quad (6)$$

According to [5], the overall internal speedup is given by

$$S = \frac{\text{cost}}{T} = 1+\frac{N\delta}{T}+2\sqrt{\frac{N\delta}{T}}. \quad (7)$$

However, the above equation (7) is flawed. Note that T_S is the total number of A-domain slots (i.e. slots produced by the scheduling algorithm), or

$$T_S = \sum_{i=1}^{N_S} \phi_i .$$

The number of A-domain slots T_S is counted after speedup S is applied, and $N_S\delta$ is counted before speedup S is applied. In these two different domains, unit time slot lengths are different. Consequently, minimizing the sum of these two values as in (5) *does not* mean that the actual total transmission time and the overall internal speedup S can be minimized. Thus, we conclude that the OSS problem defined in [5] is flawed, and the results produced by ADJUST algorithm are not accurate.

In fact, in order to find a self-adaptive algorithm to minimize the overall internal speedup S for systems with different values of δ, we can solve the following equation

$$\frac{\partial S}{\partial N_S} = 0, \quad (8)$$

where $S=f(N_S,\delta)$ is derived in (4). From (4) and (8), we have

$$N_S = \sqrt{\frac{TN}{\delta}} = \lambda N \text{ where } \lambda = \sqrt{\frac{T}{\delta N}}. \quad (9)$$

The above N_S is less than the number of configurations required by ADJUST in (6). In fact, this is the basic theory for ADAPTIVE algorithm to be discussed in Section V.

Let S_{ADAPT}, S_{ADJUST} and S_{DOUBLE} represent the overall internal speedup S for ADAPTIVE, ADJUST and DOUBLE. From (4) and (9), we have

$$S_{\text{ADAPT}} = f(N_S,\delta)\big|_{N_S=\lambda N} = \frac{TN_S}{(T-\delta N_S)(N_S-N)}\Big|_{N_S=\lambda N} = \frac{\lambda T}{(T-\delta\lambda N)(\lambda-1)}.$$

The internal speedup defined in (7) for ADJUST is obviously incorrect. According to (1) and the algorithm description in [5], we correct the internal speedup for ADJUST as follows:

$$S_{\text{ADJUST}} = \frac{T_S}{T-\delta N_S}\Big|_{N_S=(\lambda+1)N} = \frac{\sum_{i=1}^{N_S}\phi_i}{T-\delta N_S} = \frac{T+\frac{T}{\lambda}}{T-\delta N_S}\Big|_{N_S=(\lambda+1)N} = \frac{T+\sqrt{\delta NT}}{T-\sqrt{\delta NT}-\delta N}.$$

It is easy to prove by calculation that $S_{\text{ADAPT}}<S_{\text{ADJUST}}$ is always true. At the same time, note that

$$S_{\text{DOUBLE}} = \frac{T}{T-\delta N_S} S_{\min}\Big|_{N_S=2N,\,S_{\min}=2} = \frac{2T}{T-2\delta N}$$

is the internal speedup for DOUBLE algorithm. Because

$$T > T_{\min} = \delta N_S \geq \delta N \text{ and } \frac{1}{\lambda} = \sqrt{\frac{\delta N}{T}} < 1,$$

we have

$$S_{\text{ADAPT}} \leq S_{\text{DOUBLE}} \Leftrightarrow F_1(\lambda) = \left(\frac{2}{\lambda}-1\right)^2 \geq 0 \quad (10)$$

$$S_{\text{ADJUST}} \leq S_{\text{DOUBLE}} \Leftrightarrow F_2(\lambda) = \frac{2}{\lambda^3}-\frac{3}{\lambda}+1 \geq 0 \Leftrightarrow \lambda \geq \frac{1}{0.36} = 2.78 \quad (11)$$

$F_1(\lambda)$ and $F_2(\lambda)$ are plotted in Fig. 4. Comparing S of the three algorithms is mathematically equivalent to comparing $F_1(\lambda)$ and $F_2(\lambda)$ with 0. From (10) we conclude that S_{ADAPT} is always smaller than S_{DOUBLE} except when $\lambda=2$ ($\delta=T/(4N)$) for equality. From (11), we can see that S_{ADJUST} is not always smaller than S_{DOUBLE} as it was claimed in [5]. In fact, S_{ADJUST} is smaller than S_{DOUBLE} only when $\lambda>2.78$.

Finally, it is necessary to point out that Fig. 8 as well as some related discussion in [5] is impossible to be correct. The

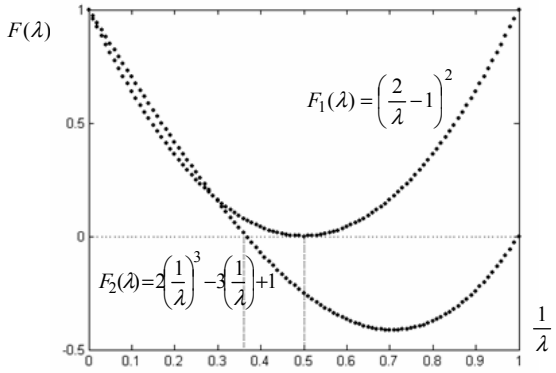

Fig. 4. Comparing S values of the three algorithms is mathematically equivalent to comparing $F_1(\lambda)$ and $F_2(\lambda)$ with 0.

first reason is that there must be $T>T_{\min}=\delta N_S \geq \delta N$ for any performance guaranteed algorithm. So it is impossible for δ to be larger than T/N (and thus δ/T should always be less than $1/N$). Secondly, for ADJUST algorithm, it is obvious that

$$N_S = (1+\lambda)N = \left(1+\sqrt{T/\delta N}\right)N > 2N .$$

Thus it is impossible for ADJUST to use less configurations than DOUBLE for arbitrary traffic matrix $\boldsymbol{C}(\boldsymbol{T})$.

V. Adaptive Algorithm

Given T, N and δ, according to (9), internal speedup S is minimized when

$$N_S = \lambda N = \sqrt{\frac{TN}{\delta}} .$$

ADAPTIVE algorithm

Step 1. Calculate the coarse matrix $\boldsymbol{A}$. Calculate the regulating factor λ and the number of configurations N_S as follows:

$$\lambda = \sqrt{\frac{T}{\delta N}} \quad \text{and} \quad N_S = \lfloor \lambda N \rfloor ,$$

Define an $N \times N$ matrix $\boldsymbol{A}$ such that

$$a_{ij} = \left\lfloor \frac{c_{ij}}{T/(N_S - N)} \right\rfloor .$$

Step 2. Color $\boldsymbol{A}$. Construct the bipartite multigraph $\boldsymbol{G_A}$ from $\boldsymbol{A}$. Find a minimal edge-coloring [7][8] of $\boldsymbol{G_A}$ to get at most $N_S - N$ colors. Set $n \leftarrow 1$.

Step 3. Schedule the coarse matrix $\boldsymbol{A}$. For a specific color in the edge-coloring of $\boldsymbol{G_A}$, construct a switch configuration $\boldsymbol{P_n}$ from the edges assigned to that color. Set

$$\phi_n = \lceil T/(N_S - N) \rceil$$

and $n \leftarrow n+1$. Repeat *Step 3* for each of the colors in $\boldsymbol{G_A}$.

Step 4. Schedule the fine matrix. Find any N non-overlapping perfect matchings $\boldsymbol{P_n}$, $n \in \{N_S - N+1, \ldots, N_S\}$ and set

$$\phi_n = \lceil T/(N_S - N) \rceil .$$

Based upon the above analysis, an ADAPTIVE algorithm is given above. This algorithm takes both δ and scheduling inefficiency into account. The algorithm is self-adaptive to different δ, T and N values. It can generate a suitable number of configurations to minimize the overall internal speedup.

Note that in our previous discussion in Section III, we use $N_S=(N+k)+N=2N+k$ switch configurations to cover $\boldsymbol{C}(\boldsymbol{T})$. i.e. $N_S - N = N+k$. Thus, ADAPTIVE is the same as the problem we discussed in Section III. From (4), (8), (9) and our previous discussion, it is easy to see that ADAPTIVE algorithm is correct. The algorithm's running time is dominated by edge-coloring of $(N_S - N)\times N = (\lambda - 1)N^2$ edges and thus the time complexity is $O((\lambda-1)N^2 \log N)$. Table I summarizes DOUBLE, ADJUST, and ADAPTIVE algorithms. Fig. 5 shows an example execution of these algorithms. The all-1-matrices in

TABLE I

Comparison of Double, Adjust and Adaptive

Algorithms	N_S	S	Time complexity
DOUBLE	$2N$	$S_{\text{DOUBLE}} = \frac{2T}{T - 2\delta N}$	$O(N^2 \log N)$
ADJUST	$(\lambda+1)N$	$S_{\text{ADJUST}} = \frac{T+\sqrt{\delta NT}}{T-\sqrt{\delta NT}-\delta N}$	$O(\lambda N^2 \log N)$
ADAPTIVE	$\lfloor \lambda N \rfloor$	$S_{\text{ADAPT}} = \frac{\lambda T}{(T-\delta\lambda N)(\lambda-1)}$	$O((\lambda-1)N^2 \log N)$

$$C(T) = \begin{bmatrix} 20 & 7 & 0 & 6 \\ 1 & 10 & 12 & 1 \\ 7 & 8 & 12 & 9 \\ 7 & 6 & 12 & 11 \end{bmatrix}, \quad N=4, \quad T=36, \quad \delta=4, \quad \lambda = \sqrt{T/\delta N} = 1.5$$

a) ADJUST algorithm

$$C(T) = 6\begin{bmatrix} 3 & 1 & 0 & 1 \\ 0 & 1 & 2 & 0 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 2 & 1 \end{bmatrix} + \begin{bmatrix} 2 & 1 & 0 & 0 \\ 1 & 4 & 0 & 1 \\ 1 & 2 & 0 & 3 \\ 1 & 0 & 0 & 5 \end{bmatrix} \leq 6\left(\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \right) + 6\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$N_S = (\lambda+1)N = 10, \quad N_S^* = 10, \quad T_S^* = 60, \quad S^* = \frac{T_S^*}{T - \delta N_S^*} = 15$$

b) DOUBLE algorithm

$$C(T) = 9\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} + \begin{bmatrix} 2 & 7 & 0 & 6 \\ 1 & 1 & 3 & 1 \\ 7 & 8 & 3 & 0 \\ 7 & 6 & 3 & 2 \end{bmatrix} \leq 9\left(\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \right) + 9\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$N_S = 2N = 8, \quad N_S^* = 7, \quad T_S^* = 63, \quad S^* = \frac{T_S^*}{T - \delta N_S^*} = 7.875 < 18 = \frac{2T}{T - 2\delta N} = S_{\text{DOUBLE}}$$

c) ADAPTIVE algorithm

$$C(T) = 18\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 2 & 7 & 0 & 6 \\ 1 & 10 & 12 & 1 \\ 7 & 8 & 12 & 9 \\ 7 & 6 & 12 & 11 \end{bmatrix} \leq 18\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + 18\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$N_S = \lfloor \lambda N \rfloor = 6, \quad N_S^* = 5, \quad T_S^* = 90, \quad S^* = \frac{T_S^*}{T - \delta N_S^*} = 5.625 < 9 = \frac{\lambda T}{(T - \delta\lambda N)(\lambda - 1)} = S_{\text{ADAPT}}$$

Fig. 5. Example execution of ADJUST, DOUBLE and ADAPTIVE

the example represent the fine part schedule for each algorithm. It can be covered by N=4 non-overlapping perfect matchings with corresponding weights. Let N_S represent the maximum number of configurations required by each algorithm. N_S^*, T_S^* and S^* are used to denote the actual number of configurations required, the total number of A-domain slots produced and the actual internal speedup required for the particular $\boldsymbol{C(T)}$ in each algorithm. However, for arbitrary traffic in a performance guaranteed system, the worst-case internal speedup S_{ADJUST}, S_{DOUBLE} and S_{ADAPT} have to be applied, and at the same time the worst-case traffic delay has to be designed according to N_S ($T>T_{\min}=\delta N_S$) instead of N_S^*.

It is obvious that ADJUST is disabled for the instance listed in Fig. 5. The reason is that ADJUST generates too many configurations and they are impossible to be accommodated in T time slots in the third stage ($T<\delta N_S$). Consequently ADJUST can not generate a performance guaranteed schedule for this situation. However, for the same instance, DOUBLE and ADAPTIVE work perfectly well. In fact, the same fault will never happen on ADAPTIVE because

$$\delta N_S = \delta\lfloor \lambda N \rfloor = \delta\left\lfloor \sqrt{\frac{TN}{\delta}} \right\rfloor \le \sqrt{T\times(\delta N)} < \sqrt{T\times T} = T \cdot$$

The example in Fig. 5 also indicates that ADAPTIVE outperforms DOUBLE because $S_{\text{ADAPT}}<S_{\text{DOUBLE}}$. For $\lambda<2$ (i.e. $T<4\delta N$), ADAPTIVE needs less N_S than DOUBLE and thus a smaller traffic delay can be achieved.

VI. Discussion

In the internal speedup function $S=f(N_S,\delta)$ in (4), we assume that T and N are predefined system constants. But obviously it is not a necessary condition. In fact, if we want to study how the values of T and N affect S, the internal speedup function in (4) can also be considered as a function of T and N. The $S=f(N_S,\delta)$ function presented in this paper gives an upper bound for overall internal speedup S. However, it is possible that there exists another polynomial time algorithm which can achieve a lower bound than the one we presented. We leave this problem as an open question for future study.

ADJUST mixes up the notions of A-domain slots and time slots outside (the third stage) in the architecture discussed in Section II. Consequently, the OSS problem defined in [5] is flawed and results based on it are not accurate. In some cases when DOUBLE and ADAPTIVE work well, ADJUST may be inapplicable. On the other hand, DOUBLE ignores the reciprocity of δ and scheduling inefficiency towards the goal of minimizing S. ADAPTIVE takes both effects (δ and scheduling inefficiency) into account. Based on minimizing the internal speedup function $S=f(N_S,\delta)$, ADAPTIVE achieves a better result than DOUBLE and ADJUST. The new algorithm can be applied to optical circuit switches and any other non-zero-overhead switches as well. For example, the SS/TDMA scheduling problem [9][10] is the same as the scheduling problem considered in this paper. Our results can be easily applied to those similar problems and systems.

Finally, this paper is for performance guaranteed scheduling and average case performance analysis is not included. In addition, it does not consider scheduling with arbitrary asynchronous reconfigurations, in which some connections remain static and continue to transmit packets while other connections are reconfigured. Scheduling a switch with arbitrary reconfigurations can be formulated as an *open shop* problem and some related discussion can be found in [4].

VII. Conclusion

Using optical technologies in the design of packet switches/routers is very attractive because of many of its benefits. In order to meet QoS requirements for Internet services, optical packet switches need to provide performance guaranteed switching. Thus, an internal speedup is required to compensate for the reconfiguration overhead and the inefficient scheduling of the algorithm. How to minimize the overall internal speedup is a key issue for making these optical packet switches practical.

We proved that the ADJUST algorithm proposed in [5] is not accurate. Based on the internal speedup function $S=f(N_S,\delta)$ derived in this paper, we proposed another new algorithm (ADAPTIVE) to minimize the internal speedup S. The algorithm generates an appropriate schedule and dynamically suits the system parameters δ, T and N for the best scheduling result. Consequently, the effects of reconfiguration overhead δ and scheduling inefficiency are well balanced by generating a suitable number of configurations N_S to achieve a minimized overall internal speedup S. The new algorithm outperforms DOUBLE and ADJUST since it requires a smaller S with time complexity of $O((\lambda-1)N^2\log N)$. It provides guaranteed performance (100% throughput with bounded delay of $2T+H$ time slots) for optical packet switches.

References

[1] A. Neukermans and R. Ramaswami, "MEMS technology for optical networking applications", *IEEE Commun. Mag.*, vol. 39, pp. 62-69, Jan. 2001.

[2] J.E Fouquet et. al, "A compact, scalable cross-connect switch using total internal reflection due to thermally-generated bubbles", in *IEEE LEOS Annual Meeting*, Orlando, FL, Dec. 1998, pp. 169-170.

[3] O. B. Spahn, C. Sullivan, J. Burkhart, C. Tigges, and E. Garcia "GaAs-based microelectromechanical waveguide switch", in *Proc. 2000 IEEE/LEOS Intl. Conf. on Optical MEMS*, Honolulu, HI, Aug. 2000, pp. 41-42.

[4] B. Towles and W. J. Dally, "Guaranteed scheduling for switches with configuration overhead", *IEEE/ACM Trans. Networking*, vol. 11, no. 5, pp. 835-847, Oct. 2003.

[5] Xin Li and Hamdi, M., "On scheduling optical packet switches with reconfiguration delay", *Selected Areas in Communications, IEEE Journal on* , vol. 21, issue 7, pp. 1156-1164, Sept. 2003.

[6] T. Inukai, "An efficient SS/TDMA time slot assignment algorithm", *IEEE Trans. Commun*, vol. COM-27, no. 10, pp. 1449-1455, 1979.

[7] R. Cole and J. Hopcroft, "On edge coloring bipartite graphs", *SIAM Journal on Computing*, vol. 11, pp. 540-546, Aug. 1982.

[8] R. Diestel, *Graph Theory*, 2nd ed. New York: Spring-Verlag, 2000.

[9] Y. Ito, Y. Urano, T. Muratani, and M. Yamaguchi, "Analysis of a switch matrix for an SS/TDMA system", *Proc. of the IEEE*, vol. 65, no. 3, pp. 411-419, 1977.

[10] S. Gopal and C. K. Wong, "Minimizing the number of switchings in an SS/TDMA system", *IEEE Trans. Commun*, vol. 33, pp. 497-501, Jun. 1985.