On Optimization of Optical Packet Switches with Reconfiguration Overhead

Bin Wu, *Student Member, IEEE,* and Kwan L. Yeung, Senior *Member, IEEE*

***Abstract*—Optical packet switching is one of the most promising technologies for carrying IP traffic over WDM optical networks. For optical packet switch (OPS) design, due to the reconfiguration overhead in the switch fabric, packet delay and speedup are two key factors to be considered. Existing scheduling algorithms, DOUBLE [4] and ADAPTIVE [5], make effective tradeoff between these two factors. In this paper, we show that the performance of both DOUBLE and ADAPTIVE, as well as their underlying OPS switch architecture, can be further optimized. Our proposed solutions are shown to effectively reduce both speedup and packet delay at the same time without incurring any extra cost.**

***Index Terms*—Optical packet switch (OPS), performance guaranteed scheduling, reconfiguration overhead, speedup.**

I. Introduction

Optical packet switching has been widely recognized as one of the most promising technologies for future *optical Internet.* At the same time, recent progress on optical switching technologies, such as MEMS mirror [1], thermal bubble [2] and waveguide [3], provide a solid groundwork for optical packet switch (OPS) implementation, whereas WDM (wavelength division multiplexing) offers an excellent transmission platform for carrying IP traffic. Consequently, OPS, the technology that can bring about many advantages, attracts more intensive attentions than ever.

However, one major implementation hurdle of OPS is the *reconfiguration overhead.* That is, OPS needs relatively large time period to change its cross-connection state. Two factors are mainly responsible for this result. First, OPS needs much longer time to set up the physical state for new connections compared with its electronic counterpart. Generally, the time spent on physical state setup can range from 10ns to several milliseconds depending on the optical switching technology adopted [4]. Second, system resynchronization also introduces considerable overhead. Since OPS usually makes use of passive crossbar switch fabric (which does not process signals), the transmitter and receiver devices must be resynchronized every time the switch fabric is reconfigured. Furthermore, the central scheduler has to synchronize all the ports as well as the crossbar, so that they can switch at the same time. Due to the latent variations in signal arriving times, the clock and its phase have to be aligned, and extra clock margins have to be considered in order to avoid data loss. All the above operations need time to be completed before switching, collectively resulting in a large switch reconfiguration overhead.

Consequently, four penalties appear in OPS: 1) incoming packets have to be buffered in either optical or electrical domain, waiting for the switch reconfiguration; 2) traffic is inevitably delayed; 3) scheduling algorithm is necessary; and 4) an internal speedup is required in order to achieve performance guaranteed switching (i.e. 100% throughput with bounded packet delay). Thus, how to reduce the internal speedup and the packet delay is the main concern.

Unfortunately, speedup and packet delay interact with each other in such a way that one goes down and the other one goes up [5]. Intuitively, minimizing the number of switch configurations required can minimize the packet delay, because each reconfiguration is associated with an overhead. However, this idea makes the scheduling algorithm to generate many empty slots, causing scheduling inefficiency and resulting in a large speedup requirement [4-6]. Therefore, some efficient algorithms, namely DOUBLE [4] and ADAPTIVE [5], are recently proposed to trade speedup for packet delay or vice versa.

In this paper, we aim at optimizing the design of DOUBLE [4] and ADAPTIVE [5], as well as their underlying OPS switch architecture. Our work is shown to improve almost all the aspects of the two existing algorithms. It reduces speedup, packet delay and the scheduling algorithms' *computational* complexity at the same time without incurring any extra cost.

The remaining part of this paper is organized as follows. In Section II, we review the existing OPS switch architecture [4-7]. In Section III, we briefly summarize the two existing scheduling algorithms, DOUBLE and ADAPTIVE. Then Sections IV and V optimize these two algorithms and their underlying OPS switch architecture respectively. Finally the paper is concluded in Section VI.

II. Existing OPS Switch Architecture

The existing OPS switch architecture consists of the OPS switch fabric (Fig. 1) and the corresponding scheduling procedure/pipeline (Fig. 2). Incoming packets are periodically

This research was supported by the Research Grants Council of Hong Kong under Grant No. 7032/01E.

Bin Wu and Kwan L. Yeung are with the Department of Electrical and Electronic Engineering, The University of Hong Kong, Pokfulam, Hong Kong (Tel: 852-2857-8493; Fax: 852-2559-8738; e-mail: {binwu, kyeung} @eee.hku.hk).

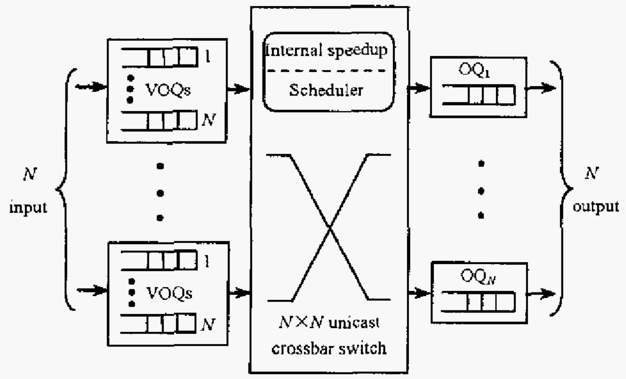


Fig. 1. An $N \times N$ unicast OPS switch fabric.

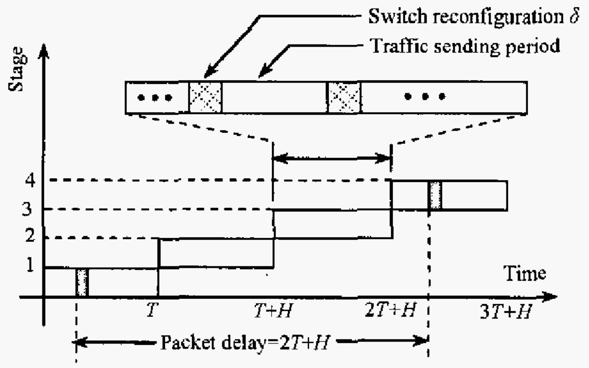


Fig. 2. Four-stage optical packet switch scheduling pipeline.

accumulated and TSA (time slot assignment) method is applied to determine a set of N_S configurations to deliver the collected packets. The scheduling procedure/pipeline in Fig. 2 is divided into four stages. In Stage 1, incoming packets are accumulated in the input VOQ buffers over T time slots to construct the $N \times N$ traffic matrix $C(T)$. Its entry c_{ij} denotes the number of packets received at input i and destined to output j. The switch fabric takes H time slots in Stage 2 to generate N_S configurations $P_1, \ldots, P_{N_S}$ to cover $C(T)$.[1] Assume that the switch is a unicast crossbar. Rows and columns of P_n, $n \in \{1, \ldots, N_S\}$ represent input and output ports of the switch respectively. The $N \times N$ matrix P_n has at most a single "1" in each line (row or column), indicating connection of the two corresponding ports. Other entries are all zeros. P_n is called a *perfect matching* if it has N "1" elements. Then in Stage 3 the switch is reconfigured according to these N_S configurations. We assume that each switch reconfiguration needs δ (*regular*) time slots, during which no packet can be transmitted across the switch. An overall internal speedup S is applied to ensure that this stage occupies only T time slots. After the speedup is applied, the unit slot time for packet transmission in Stage 3 is compressed/shortened by the speedup, and each P_n holds for ϕ_n *compressed* slots for packet transmission. Finally in Stage 4 packets are sent onto output lines from output OQ buffers.

From the tagged packet in Fig. 2, we can see that the packet delay is bounded to $2T+H$ time slots. Assume $T > \delta N_S$ and all the line sums of $C(T)$ are not larger than T. Since δN_S (regular) time slots must be used to reconfigure the switch for N_S times and thus only $T-\delta N_S$ slots left for transmitting $C(T)$ in Stage 3, a speedup factor $S_{\text{reconfigure}}=T/(T-\delta N_S)$ is necessary to compensate solely for δ. At the same time, the scheduling algorithm may produce some empty slots. Thus another speedup factor S_{schedule} is required to compensate for the inefficient scheduling. The overall speedup S is then given by [4-5]

$$S = S_{\text{reconfigure}} S_{\text{schedule}} = \frac{T S_{\text{schedule}}}{T - \delta N_S}. \quad (1)$$

[1] Let $p^{(n)}_{ij}$ and c_{ij} represent the element (i,j) of P_n and $C(T)$ respectively. An $N \times N$ traffic matrix $C(T)$ is covered by a set of switch configurations $P_1, \ldots, P_{N_S}$, each weighted by a non-negative integer $\phi_1, \ldots, \phi_{N_S}$, if and only if $\sum^{N_S}_{n=1} \phi_n p^{(n)}_{ij} \ge c_{ij}$ for any $i,j \in \{1, \ldots, N\}$.

III. Double and Adaptive Algorithms

DOUBLE and ADAPTIVE are *concisely* summarized in this section. Details can be found in [4-5].

Divide the traffic matrix $C(T)$ into a *coarse* matrix A and a *fine* matrix R as follows:

$$C(T) = \frac{T}{N_S - N} A + R, \; a_{ij} = \left\lfloor \frac{c_{ij}}{T/(N_S - N)} \right\rfloor, \quad (2)$$

where a_{ij} denotes the element (i,j) of A. N_S represents the number of configurations required to cover $C(T)$ and is in the range of $N^2-2N+2 > N_S > N$.[2] Because the maximum line sum of $C(T)$ is assumed to be not more than T, the maximum line sum of A is at most N_S-N. Thus the corresponding bipartite multigraph of A can be edge-colored in N_S-N colors [7-9]. These N_S-N colors can be mapped back to form N_S-N corresponding switch configurations P_n, $n \in \{1, \ldots, N_S-N\}$ to cover A. The fine matrix R *does not need* to be explicitly computed because all its elements are guaranteed to be less than $T/(N_S-N)$. Therefore any N configurations (from P_{N_S-N+1} to P_{N_S}) that *collectively* represent every entry of $C(T)$ and each weighted by $T/(N_S-N)$, can be used to cover the fine matrix R. Consequently, $C(T)$ can be covered by $(N_S-N)+N=N_S$ switch configurations, each equally weighted by $\phi_n=T/(N_S-N)$.[3] It means that the algorithm uses $N_S \times \phi_n = N_S \times T/(N_S-N)$ compressed slots to transmit (at most) T packets for each input port. As a result, S_{schedule} in (1) is given by

$$S_{\text{schedule}} = \frac{1}{T} N_S \left(\frac{T}{N_S - N} \right) = \frac{N_S}{N_S - N} = 1 + \frac{N}{N_S - N}. \quad (3)$$

DOUBLE [4] is a special case of our above discussion. It uses $N_S=2N$ configurations to cover $C(T)$ (N configurations to cover the coarse matrix A and the other N configurations to cover the fine matrix R), each weighted by $\phi_n=T/N$. It achieves $S_{\text{schedule}}=2$. ADAPTIVE [5] considers more general cases. Based on (1) and (3), it finds out the best N_S value to minimize the overall speedup S in (1), where

[2] $N_S=N^2-2N+2$ stands for EXACT algorithm [4, 10] and $N_S=N$ stands for MIN [4] and α-SCALE [6] algorithms.

[3] If $T/(N_S-N)$ is not an integer, use its ceiling integer as the substitute.

$$N_S = \lfloor \lambda N \rfloor = \left\lfloor \sqrt{\frac{TN}{\delta}} \right\rfloor \text{ and } \lambda = \sqrt{\frac{T}{\delta N}}. \qquad (4)$$

IV. Optimization on OPS Scheduling Algorithms

The performance of both DOUBLE [4] and ADAPTIVE [5] can be optimized. While sharing the same basic idea, the mechanisms discussed below apply to both algorithms with only minor modifications. For simplicity, we only take DOUBLE (which is the simpler case, i.e. N_S=2N) as the example for discussion. Then we give the corresponding results for ADAPTIVE at the end of this section.

A. Optimization on DOUBLE

We start from the following Lemma 1, which is basically the same as Lemma 2 in [7] (pp. 1159) and a proof can be found there.

Lemma 1: Any $N\times N$ matrix $\boldsymbol{M}$ with maximum line sum L_S can be covered by using L_S configuration matrices.

In DOUBLE, there are only two possible cases for the coarse matrix $\boldsymbol{A}$: 1) all of its lines sum to an integer less than N; 2) some of its lines sum to N. In case 1), all of the line sums of $\boldsymbol{A}$ are at most $N-1$. Thus, $\boldsymbol{A}$ can be covered by $N-1$ configurations according to Lemma 1; In case 2), some of the line sums of $\boldsymbol{A}$ equal to N. As a result, the corresponding lines of $\boldsymbol{C(T)}$ *do not have residues in the fine matrix* $\boldsymbol{R}$ because the maximum line sum of $\boldsymbol{C(T)}$ is T (refer to formula (2) and note that N_S=2N for DOUBLE). So, in the fine matrix $\boldsymbol{R}$, the corresponding rows or columns should be all zeros. We can refer to Fig. 3 for an example. In Fig. 3, T=16 and N=4. For the example execution of DOUBLE, the corresponding $\boldsymbol{A}$ and $\boldsymbol{R}$ matrices are

$$\boldsymbol{A} = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 2 & 0 & 1 \\ 0 & 1 & 0 & 2 \end{bmatrix}, \ \boldsymbol{R} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 3 & 1 \\ 0 & 0 & 3 & 1 \end{bmatrix}.$$

We can see that the first row, the first and the second columns of $\boldsymbol{R}$ are all zeros. This is because the corresponding lines in $\boldsymbol{A}$ sum to N=4. Since the coarse matrix $\boldsymbol{A}$ is multiplied by an integer T/N=4 in $\boldsymbol{C(T)}$'s decomposition and the maximum line sum of $\boldsymbol{C(T)}$ is T=16, there are no residues in these corresponding lines of $\boldsymbol{R}$. However, in the fine part schedule of DOUBLE, N configurations (that collectively represent *every entry* of an $N\times N$ matrix) are used to cover the fine matrix $\boldsymbol{R}$. For the above example, an all-1 matrix (equals to the sum of the N perfect matchings $\boldsymbol{P_5}$-$\boldsymbol{P_8}$) weighted by T/N=4 is used to cover $\boldsymbol{R}$. Obviously, for those lines of $\boldsymbol{A}$ whose line sums equal to N, slots are *worthlessly wasted* by DOUBLE's fine part schedule. In fact, we can make use of these wasted slots to reduce N_S.

Lemma 2: The maximum number of configurations (N_S) required in DOUBLE can be reduced by one (i.e. N_S=2$N-1$) by making use of the wasted slots in the fine part schedule.

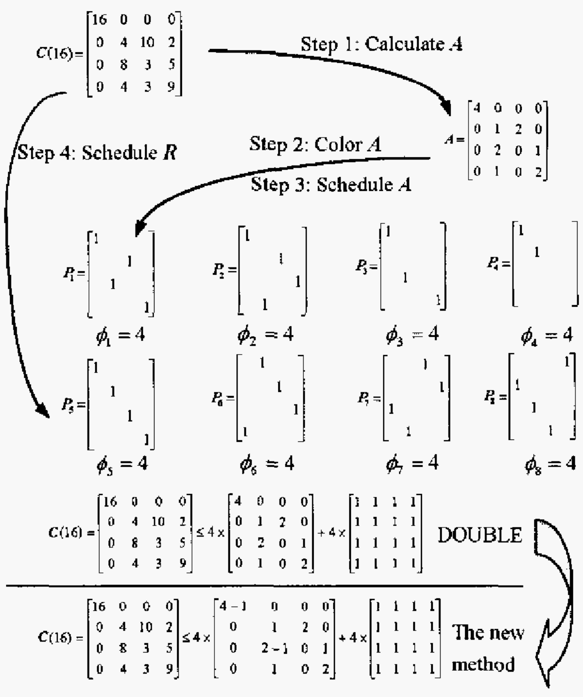
Proof: If the maximum line sum of $\boldsymbol{A}$ is less than N, then $\boldsymbol{A}$ can be covered by at most $N-1$ configurations. Otherwise if some lines in $\boldsymbol{A}$ sum to N, the corresponding lines in the fine matrix $\boldsymbol{R}$ should be all zeros, and the corresponding slots are wasted by DOUBLE's fine part schedule. To counteract this, we can "shift" some traffic from $\boldsymbol{A}$ to $\boldsymbol{R}$ before scheduling is carried out. This can be achieved by reducing any one of the positive elements in each of those lines of $\boldsymbol{A}$ (with sum to N) by 1, making the maximum line sum of the new coarse matrix to $N-1$ instead of N. This treatment will not cause any problem in terms of covering $\boldsymbol{C(T)}$, because all the weights of the 2N configurations in DOUBLE are equally set to ϕ_n=T/N and the deducted values in $\boldsymbol{A}$ can be compensated by the fine part schedule (by making use of the wasted slots, where the fine part schedule keeps the same as DOUBLE). Thus, according to Lemma 1, the new coarse matrix ($\boldsymbol{A}$ after deduction) can be covered by $N-1$ configurations. Taking the other N configurations to cover $\boldsymbol{R}$ into consideration, $\boldsymbol{C(T)}$ as a whole can be covered by N_S=($N-1$)+N=2$N-1$ configurations with each weighted by T/N.

#

Lemma 3: S_{schedule} (the speedup factor to compensate for the scheduling inefficiency) is reduced to $2-1/N$ under the new method, resulting in a reduced overall speedup of

$$S = \frac{\left(2-\frac{1}{N}\right)T}{T-(2N-1)\delta}. \qquad (5)$$

Proof: As a consequence of Lemma 2, the resulting schedule consists of N_S=2$N-1$ configurations with each weighted by T/N under the new method. Thus, the OPS switch transmits T packets for each input port in $(2N-1)\times(T/N)$ compressed slots.



$$C(16)=\begin{bmatrix}16&0&0&0\\0&4&10&2\\0&8&3&5\\0&4&3&9\end{bmatrix} \qquad A=\begin{bmatrix}4&0&0&0\\0&1&2&0\\0&2&0&1\\0&1&0&2\end{bmatrix}$$

$$\phi_1=4 \quad \phi_2=4 \quad \phi_3=4 \quad \phi_4=4 \quad \phi_5=4 \quad \phi_6=4 \quad \phi_7=4 \quad \phi_8=4$$

$$C(16)=\begin{bmatrix}16&0&0&0\\0&4&10&2\\0&8&3&5\\0&4&3&9\end{bmatrix}\le 4\times\begin{bmatrix}4&0&0&0\\0&1&2&0\\0&2&0&1\\0&1&0&2\end{bmatrix}+4\times\begin{bmatrix}1&1&1&1\\1&1&1&1\\1&1&1&1\\1&1&1&1\end{bmatrix}$$

$$C(16)=\begin{bmatrix}16&0&0&0\\0&4&10&2\\0&8&3&5\\0&4&3&9\end{bmatrix}\le 4\times\begin{bmatrix}4-1&0&0&0\\0&1&2&0\\0&2-1&0&1\\0&1&0&2\end{bmatrix}+4\times\begin{bmatrix}1&1&1&1\\1&1&1&1\\1&1&1&1\\1&1&1&1\end{bmatrix}$$

Fig. 3. An example of DOUBLE and its optimized counterpart.

Consequently, $S_{\text{schedule}}=(2N-1)\times(T/N)/T=2-1/N$. According to (1), the overall speedup S is reduced to

$$S=\frac{TS_{\text{schedule}}}{T-\delta N_S}=\frac{\left(2-\frac{1}{N}\right)T}{T-(2N-1)\delta}.$$

#

Lemma 4: The computational complexity of the algorithm is reduced from $O(N^2\log N)$ to $O(N(N-1)\log N)$.

Proof: Generally, the edge-coloring algorithm has a time complexity of $O(E\log V)$ [8], where E is the number of edges and V is the number of vertices in the bipartite multigraph. For DOUBLE, $E=O(N^2)$ and $V=O(N)$. For the new method, $E=O(N(N-1))$ and $V=O(N)$. Consequently, the computational complexity of the algorithm is reduced from $O(N^2\log N)$ to $O(N(N-1)\log N)$.

#

Theorem 1: The optimized DOUBLE algorithm can cover $\boldsymbol{C(T)}$ using $N_S=2N-1$ configurations, each weighted by T/N, with $S_{\text{schedule}}=2-1/N$. The computational complexity of the optimized algorithm is also reduced to $O(N(N-1)\log N)$.

Usually, reducing N_S means that the packet delay $2T+H$ can also be reduced. At the same time, making the algorithm execution simpler is helpful for achieving a smaller H.

B. *Optimization on ADAPTIVE*

Following the same argument as above, ADAPTIVE [5] can also be optimized. The detailed derivations are omitted and the result is given below.

Theorem 2: The optimized ADAPTIVE algorithm can cover $\boldsymbol{C(T)}$ using $\lfloor \lambda N \rfloor -1$ configurations (where λ is defined in (4)), each weighted by $\lceil T/(\lfloor \lambda N \rfloor - N)\rceil$. The following overall internal speedup is sufficient to schedule $\boldsymbol{C(T)}$ in T time slots:

$$S=\frac{(\lambda N-1)T}{\lceil T-(\lambda N-1)\delta\rceil(\lambda N-N)}. \tag{6}$$

V. Optimization on OPS Switch Architecture

From our previous discussion in Section III, we know that the N configurations (perfect matchings) used to cover the fine matrix $\boldsymbol{R}$ can be chosen freely as long as they can cover every entry of an $N\times N$ matrix. This is true for both DOUBLE and ADAPTIVE. In fact, these N perfect matchings do not need to be explicitly computed. That is, *they can be predetermined offline*. This situation further implies that, in Fig. 2, the OPS switch fabric does not need to wait the whole H time slots for algorithm execution before it enters Stage 3. If we can arrange these N predetermined perfect matchings to be configured at the beginning of Stage 3, then switching (Stage 3) and algorithm execution (Stage 2) can work in parallel and thus time can be saved. (Note that Fig. 2 only shows that Stages 2&3 work in series.) Consequently, for DOUBLE [4] and ADAPTIVE [5], these two stages can partially overlap to reduce the total packet delay. The key point is that, N out of the N_S configurations can be predetermined for transmitting at the beginning of Stage 3, so that even if the scheduler is still calculating for the other N_S-N configurations, packet transmission at the beginning of Stage 3 can be carried out in parallel.

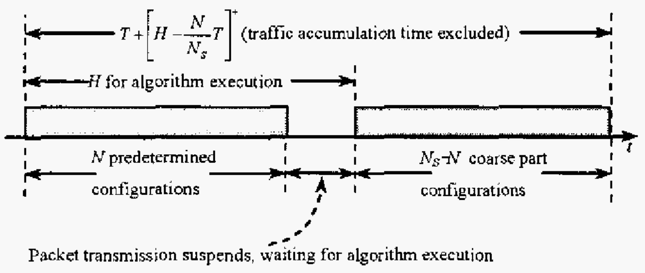

$T+\left[H-\frac{N}{N_S}T\right]^+$ (traffic accumulation time excluded)
H for algorithm execution
N predetermined configurations
N_S-N coarse part configurations
t
Packet transmission suspends, waiting for algorithm execution

Fig. 4. The time overlap between algorithm execution (Stage 2) and traffic sending (Stage 3) for $H>TN/N_S$. In this case, traffic sending suspends after the N predetermined configurations complete, waiting for the scheduling algorithm to compute the remaining N_S-N coarse part configurations.

Since the time duration of Stage 3 (in Fig. 2) is T and it consists of N_S times of reconfigurations, the N predetermined perfect matchings will last for TN/N_S (regular) time slots. Let D and T_O represent the total packet delay and the time overlap between Stages 2&3. We have

$$\begin{aligned}D&=T(\textit{traffic accumulation})+H(\textit{algorithm execution})\\&\quad+T(\textit{traffic sending})-T_O(\textit{time overlap between algorithm execution and traffic sending})\\&=2T+\left[H-\frac{N}{N_S}T\right]^+,\end{aligned} \tag{7}$$

where

$$\left[H-\frac{N}{N_S}T\right]^+=\max\left\{0,H-\frac{N}{N_S}T\right\}.$$

Equation (7) indicates that if we use a scheduler fast enough for OPS switch implementation, such that $H\leq TN/N_S$, then the packet delay D can be as small as $2T$ time slots.

Fig. 4 shows the time overlap between algorithm execution and traffic sending periods, in which the N predetermined perfect matchings are *actually* arranged at the beginning of Stage 2. That is, the OPS switch will use these N perfect matchings in parallel with algorithm execution as soon as it finishes traffic accumulation (Stage 1).[4] For $H>TN/N_S$, because the algorithm's running time is relatively long, the N_S-N coarse part configurations are not yet available right after the first N predetermined configurations complete. So, the packet transmission suspends until the algorithm execution is finished and the remaining N_S-N configurations are generated. This procedure is different from what is shown in Fig. 2, but it does not affect the conclusion that the accumulated traffic can be scheduled and transmitted in $T+[H-TN/N_S]^+$ time slots.

Without loss of generality, we assume $H>TN/N_S$ (the worst

[4] In fact, this is the simplest way to implement our pipeline idea. Note that in the new architecture, the definition of Stage 3 (traffic sending) is different from that in Fig. 2, whereas other stages are the same.

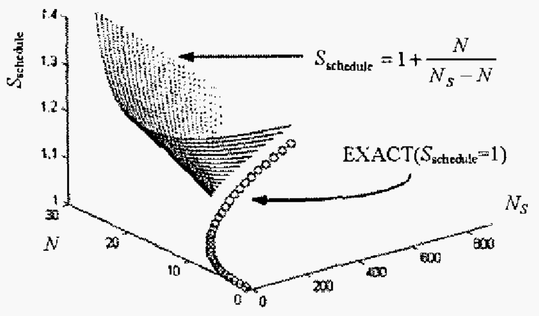


Fig. 5. Illustration of S_{schedule} for $N \le 30$.

case) in our subsequent discussion. For simplicity, we still use (3) to calculate S_{schedule} although it has already been optimized in Section IV.

From (3) we have

$$\frac{N}{N_S} = \frac{S_{\text{schedule}} - 1}{S_{\text{schedule}}}. \tag{8}$$

From (7) and (8), we get

$$D = 2T + \left[H - \frac{N}{N_S} T \right]^{+} = \frac{S_{\text{schedule}} + 1}{S_{\text{schedule}}} T + H. \tag{9}$$

The above equation (9) shows the relationship between S_{schedule} and D under the optimized OPS switch architecture. It is important to note that (7) and (9) are still subject to the constraint $T > \delta N_S$, as discussed in Section II. Our optimization on the OPS switch architecture makes use of the time overlap between Stages 2&3 (in Fig. 2), but this does not change the constraint, which indicates that T must be large enough to accommodate all the N_S configurations.

If we take DOUBLE [4] as an example, from (9) we can see that the packet delay is reduced to $D=1.5T+H$ because $S_{\text{schedule}}=2$. If we assume $H=T$ for this case, we can see that the new OPS switch architecture cuts down the packet delay by $[(2T+H)-(1.5T+H)]/(2T+H)=0.5T/(2T+H)$ =16.67%.

Generally, because $S_{\text{schedule}}>1$ (except EXACT algorithm [4, 10] which uses $N_S=N^2-2N+2$ configurations to achieve $S_{\text{schedule}}=1$), the packet delay formulated in (9) is always less than that of the architecture shown in Figs. 1&2 (which is $2T+H$). Obviously, this is because our proposed architecture makes use of the time overlap between algorithm execution and traffic sending. For general cases, the amount of time saved depends on the particular situation. To make this point clearer, S_{schedule} in (3) is plotted in Fig. 5. From the figure, we can see that a larger S_{schedule} corresponds to a smaller N_S. According to our previous discussion, a smaller N_S means that a greater portion of the T (regular) slots in Stage 3 of Fig. 2 (the old architecture) is occupied by the N predetermined perfect matchings. (Formula (8) also reflects the ratio.) This further indicates that the overlapped time period in Fig. 4 is longer and thus the new architecture can provide a greater saving in packet delay. On the contrary, a smaller S_{schedule} corresponds to a larger N_S and less saving.

Finally, it is necessary to point out that our optimization on OPS switch architecture is for DOUBLE [4] and ADAPTIVE [5]. It is infeasible for other algorithms such as MIN [4], α^i-SCALE [6] and EXACT [4][10].

VI. Conclusion

Optical packet switches (OPS) bring about scalability, high line rate, huge capacity and low power consumption features to communication networks on an economical base. It is very attractive for carrying IP traffic over WDM optical networks.

Due to the inherent reconfiguration overhead in OPS switches, speedup and packet delay are two key issues in terms of OPS implementation. Existing scheduling algorithms (DOUBLE [4] and ADAPTIVE [5]) make effective tradeoff between these two factors. In this paper, we optimized these two algorithms to use less switch configurations and lower speedup for performance guaranteed switching. The resulting algorithms' computational complexity and the packet delay are also reduced. Based on the characteristics of these two algorithms, we also modified the existing OPS switch architecture. The new switch architecture was shown to reduce the packet delay significantly. In addition, all the above performance gains are achieved without incurring any extra cost.

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", *IEEE LEOS Annual Meeting*, pp. 169-170, Dec. 1998.

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

[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] Bin Wu and Kwan L. Yeung, "Minimizing internal speedup for performance guaranteed optical packet switches", *IEEE GLOBECOM '04*, Vol. 3, pp. 1742-1746, 29 Nov.-3 Dec. 2004.

[6] Bin Wu and Kwan L. Yeung, "Scheduling optical packet switches with minimum number of configurations", *IEEE ICC '05*, Seoul South Korea, May 2005.

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

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

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

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