Scheduling Optical Packet Switches with Minimum Number of Configurations

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*—In order to achieve the minimum traffic delay in a performance guaranteed optical packet switch (OPS) with reconfiguration overhead, the switch fabric has to use the minimum number of configurations (i.e. N configurations where N is the switch size) for traffic scheduling. This requires a very high speedup in the switch fabric to compensate for the loss in scheduling efficiency. The high speedup requirement makes the idea of using N configurations (to schedule the traffic) impractical under current technology. In this paper, we propose a new scheduling algorithm called α^i-SCALE to lower the speedup required. Compared with the existing MIN algorithm [5], α^i-SCALE succeeds in pushing the speedup bound (i.e. worst-case speedup requirement) to a much lower level. For example, when N=200, the speedup bound required to compensate the loss in scheduling efficiency is 30.75 for MIN, whereas 23.45 is sufficient for our α^i-SCALE.**

Keywords-Optical packet switch(OPS); speedup; performance guaranteed scheduling; reconfiguration overhead.

I. INTRODUCTION

The rapid progress on IP and WDM research has resulted in a coalescence of these two technologies, leading to strong and wide interests in optical packet switches (OPS). OPS can offer many advantages at relatively low cost, such as scalability, high bandwidth utilization, high line-rate and low power consumption. Despite of the recent achievements on optical switching technologies [1-3], a major implementation hurdle of OPS is its relatively large *reconfiguration overhead*, which is the amount of *idle* time required to change the OPS configuration state because of some time-consuming operations involved, such as mechanical settling and synchronization.

Lying in the core of an optical packet switch is the packet scheduling algorithm. Following the approach of *batch-based time slot assignment* (*TSA*), many efficient algorithms, namely, EXACT [5,7], DOUBLE [5], MIN [5] and ADAPTIVE [6], are designed for packet scheduling. With the batch-based TSA, incoming packets are periodically (say, every T time slots, i.e. batch size = T) accumulated at the input ports of an OPS to form a traffic matrix $\boldsymbol{C}(\boldsymbol{T})$. Then a scheduling algorithm (such as any one above) is used to determine a set of switch configurations for forwarding the collected packets to the output ports. If all the packets in $\boldsymbol{C}(\boldsymbol{T})$ can be forwarded to their corresponding output ports within a bounded (i.e. worst-case) delay, the resulting OPS is called a *performance guaranteed OPS*, and the corresponding algorithm is called a *performance guaranteed scheduling algorithm*. Notably, existing algorithms, EXACT, DOUBLE, MIN and ADAPTIVE, all fall into this category. They differ in requiring different number of configurations to schedule the traffic matrix.

To realize a performance guaranteed OPS, the OPS switch fabric, which is responsible for the actual delivery of packets from input ports to output ports, must operate at a higher speed than each individual input/output line. This speedup is used to compensate for the idle time due to switch reconfiguration, and the possible loss of scheduling efficiency due to a particular scheduler implementation [4-6] (also refer to Section II). Therefore, the worst-case speedup requirement of a performance guaranteed OPS (i.e. speedup bound) depends on the scheduling algorithm adopted.

Because each reconfiguration is associated with an overhead, scheduling OPS traffic with *minimum* number of configurations can minimize traffic delay. For performance guaranteed OPS, N (where N is the switch size) is the minimum number of configurations required. This is because an $N\times N$ traffic matrix $\boldsymbol{C}(\boldsymbol{T})$ has N^2 entries, and each configuration can cover at most N of them [5]. So, at least N configurations are needed. On the other hand, as pointed out in [5,6], using less number of configurations makes the scheduling more inefficient (i.e. the packet transmission in each configuration cannot fully utilize the available switch bandwidth), and thus requires a higher switch fabric speedup than algorithms using more configurations.

Among all the proposed performance guaranteed scheduling algorithms [5-7], MIN [5] has the unique advantage of providing the minimum bounded traffic delay because it requires only N configurations for any traffic matrix $\boldsymbol{C}(\boldsymbol{T})$. But, as discussed above, the speedup required by MIN is extremely high, and seems to be prohibitive under current technology.

Obviously, scheduling OPS traffic with the minimum number of N configurations can be practical only if some new scheduling algorithm with lower speedup bound is available, or the current difficulties of high speedup implementation are overcome. In this paper, we put our effort on designing a more efficient algorithm than MIN. The new scheduling algorithm we proposed is called α^i-SCALE. We show that for small and medium size optical packet switches (N<100), α^i-SCALE complements the performance of MIN, requiring a lower speedup in roughly half of the switch size range. When switch size is large, e.g. N=200, the speedup bound required to compensate for the inefficient scheduling is 30.75 for MIN,

This work was supported by Competitive Earmarked Research Grant HKU 7048/02E.

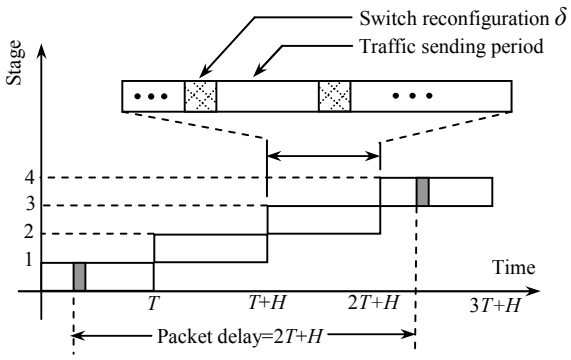

Fig. 1. Optical packet switch scheduling stages.

whereas 23.45 is sufficient for our α^i-SCALE. Besides, α^i-SCALE provides a new matrix decomposition method using N permutation matrices, which can be useful for future research in this area.

II. ARCHITECTURE

The same OPS switch architecture as in [4-6] is assumed in our work. In this architecture, batch-based TSA approach is applied to determine a set of N configurations to deliver the collected packets. Fig. 1 shows the scheduling procedure in four stages. In Stage 1, incoming packets are accumulated in the input buffers over T time slots to construct the traffic matrix $\boldsymbol{C(T)}$. Each entry c_{ij} of $\boldsymbol{C(T)}$ denotes the number of packets received at input i and destined to output j. Assume all the line sums (either row or column sum) of $\boldsymbol{C(T)}$ are not larger than T. The scheduling algorithm takes H time slots in Stage 2 to generate N configurations $\boldsymbol{P_1}$, …, $\boldsymbol{P_N}$ to cover [1] $\boldsymbol{C(T)}$. Configuration $\boldsymbol{P_k}=\{p^{(k)}_{ij}\}$ is an $N\times N$ matrix with at most a single "1" in each line (row or column). $p^{(k)}_{ij}=1$ indicates that a packet can be sent from input i to output j; $p^{(k)}_{ij}=0$ otherwise. $\boldsymbol{P_k}$ is called a *perfect matching* if it has exactly N "1" elements. In Stage 3, the switch fabric is reconfigured according to these N configurations. An internal speedup S is applied to ensure that this stage occupies only T *regular* slots. After the speedup is applied, the switch fabric holds each $\boldsymbol{P_k}$ for ϕ_k *compressed* slots for packet transmission. Finally in Stage 4 packets are sent onto the output lines from output buffers (in T slots).

From the tagged packet in Fig. 1, we can see that the bounded delay of any packet is $2T+H$ slots. Assume each switch reconfiguration takes δ regular slots and $T>T_{\min}=\delta N$. Since δN slots must be used to reconfigure the switch for N times, only $T-\delta N$ slots are left for transmitting $\boldsymbol{C(T)}$ in Stage 3. So, a speedup factor denoted by $S_{\text{reconfigure}}=T/(T-\delta N)$ is necessary to compensate solely for the idle time caused by reconfiguration. At the same time, the scheduling algorithm may produce many empty slots (i.e. underutilize the bandwidth provided by the configuration). Thus another speedup factor, $S_{\text{schedule}}=(1/T)\sum_{k=1}^{N}\phi_k$, is required to compensate solely for the inefficient scheduling. The overall internal speedup S is then given by $S=S_{\text{reconfigure}}\times S_{\text{schedule}}=TS_{\text{schedule}}/(T-\delta N)$ [5-6].

[1] $\boldsymbol{C(T)}$ is covered by configurations $\boldsymbol{P_1}$, …, $\boldsymbol{P_K}$, each weighted by a non-negative integer ϕ_1, …, ϕ_K, if and only if $\sum_{k=1}^{K}\phi_k p^{(k)}_{ij}\ge c_{ij}$ for any $i,j\in\{1,\ldots,N\}$. Note that $\boldsymbol{P_k}=\{p^{(k)}_{ij}\}$.

III. ALGORITHM

A. General Idea

α^i-SCALE takes a similar framework as MIN [5] but differs in its underlying design principle. Fig. 2 summarizes α^i-SCALE which is detailed in Parts B&C. The idea is to schedule large entries in $\boldsymbol{C(T)}$ first. The execution of α^i-SCALE consists of an inner-loop iteration (Steps 4-7) embedded inside an outer-loop iteration (Steps 2-8). In the i'th outer-loop iteration, α^i-SCALE uses a threshold T/α^i to identify large entries in $\boldsymbol{C(T)}$, where α is a *real number*. Note that when entering the i'th outer-loop iteration, the algorithm has scheduled all the entries larger than T/α^{i-1} in previous steps and thus they have been converted to zeros in $\boldsymbol{C(T)}$. In the i'th outer-loop iteration, α^i-SCALE selects large entries that satisfy $T/\alpha^{i-1}\ge c_{ij}>T/\alpha^i$ for scheduling.

After that, α^i-SCALE performs an edge-coloring [8] based on the selected large entries. Then, each color is scheduled by using two configurations in the inner-loop iteration, where each configuration covers half of the edges of the particular color. In addition, α^i-SCALE determines at most $N/4$ configurations in its outer-loop iterations (Steps 2-8), and leaves the task of determining the remaining configurations (for small entry scheduling) to Step 9. The above two mechanisms are taken to ensure that the N configurations can be properly constructed as *non-overlapping perfect matchings* (no any two of them cover the same entry of $\boldsymbol{C(T)}$).[2]

The key issue of α^i-SCALE is to determine a most suitable (α,m) pair (to be discussed more later) for any given switch size N, so that the resulting speedup factor S_{schedule} can be minimized. We define a *scale function*

$$f(N,i)=\alpha^i\Big|_{\alpha \text{ best suits } N \text{ and minimizes } S_{\text{schedule}}}.$$

Based on this scale function, we first calculate the best values of (α,m) for any given switch size N, and then substitute them into α^i-SCALE shown in Fig. 2 to schedule $\boldsymbol{C(T)}$. (α,m) can be found *offline*, as discussed in Part C. They are constants for any specific N. The online execution of α^i-SCALE is dominated by N maximum-size matchings, resulting in a total time complexity of $O(N^{3.5})$.

B. Design Principle

We first define

$$\gamma(i)=\lceil\alpha^i-1\rceil \text{ and } \omega(i)=\left\lfloor\frac{T}{\alpha^{i-1}}\right\rfloor. \quad (1)$$

Since all the line sums of $\boldsymbol{C(T)}$ are not larger than T, and

$$\gamma(i)+1=\lceil\alpha^i-1\rceil+1\ge\lceil\alpha^i\rceil\ge\alpha^i,$$

then in each line of $\boldsymbol{C(T)}$, there can be at most $\gamma(i)$ unscheduled entries greater than the threshold T/α^i in the i-th outer-loop iteration. These unscheduled large entries are indicated by "1"s in a matrix $\boldsymbol{L}$ for *each* loop (other entries are zeros), and the corresponding bipartite unigraph [4] $\boldsymbol{G_L}$ is edge-colored.

[2] This is guaranteed because, for an r-regular bipartite graph $\boldsymbol{G}=(\boldsymbol{X}\cup\boldsymbol{Y},\boldsymbol{E})$ with $|\boldsymbol{X}|=|\boldsymbol{Y}|=n$ and $r>3n/4$, any partial matching $\boldsymbol{M}$ of $\boldsymbol{G}$ with $|\boldsymbol{M}|\le n/2$ is a subset of a perfect matching of $\boldsymbol{G}$. Please also refer to Theorem 8 in [5].

According to the classical König theorem [9], $\boldsymbol{G_L}$ can be edge-colored in $\gamma(i)$ colors. Then each inner-loop iteration schedules one color by using two non-overlapping perfect matchings. Each perfect matching is weighted by $\omega(i)$ and it schedules half of the edges of the color. Since all the entries c_{ij} in $\boldsymbol{C(T)}$ are integers, those entries greater than $\omega(i)$ must have been scheduled in previous iterations. As a result, the weight $\omega(i)$ is sufficient for the i-th iteration. Consequently, the i-th outer-loop iteration determines $2\gamma(i)$ configurations and introduces $2\gamma(i)\omega(i)$ weight. After each configuration is determined, it is removed from an indicator matrix $\boldsymbol{B}$ (which is initialized to an all-1-matrix). Large entries that have already been scheduled in previous steps are also removed from $\boldsymbol{C(T)}$ (by referring to the indicator matrix $\boldsymbol{B}$). Note that this procedure terminates before the total number of configurations that have been previously determined (N_m) exceeds $N/4$ (i.e. $N_m<N/4$) in order to guarantee that non-overlapping perfect matchings can always be found in $\boldsymbol{B}$. Assume that m is the number of outer-loop iterations required to generate these N_m configurations. After m iterations, the remaining small entries in $\boldsymbol{C(T)}$ are scheduled by using another $N-N_m$ configurations with a fixed weight of $\omega(m+1)$. Each of these $N-N_m$ configurations can be extracted (by performing maximum-size matching) and then deducted from $\boldsymbol{B}$. This operation is guaranteed to be valid because $\boldsymbol{B}$ is always a *regular* matrix. Let S^E_{schedule} represent the S_{schedule} value produced *exactly* by the algorithm. We have

$$S^E_{\text{schedule}} = \frac{1}{T}\left(\sum_{i=1}^{m} 2\gamma(i)\omega(i) + (N-N_m)\omega(m+1)\right)$$

$$= \frac{1}{T}\left(\sum_{i=1}^{m} 2\left\lceil \alpha^i - 1\right\rceil \times \left\lfloor \frac{T}{\alpha^{i-1}} \right\rfloor + (N-N_m)\left\lfloor \frac{T}{\alpha^m} \right\rfloor\right). \quad (2)$$

Since minimizing S^E_{schedule} in (2) appears to be a mixed-integer *nonlinear* optimization problem and is obviously intractable, we apply an approximation method here. Specifically, we use an approximation $S^A_{\text{schedule}} \approx S^E_{\text{schedule}}$, where

$$S^A_{\text{schedule}} = \sum_{i=1}^{m} 2(\alpha^i - 1)\frac{1}{\alpha^{i-1}} + \left(\frac{3}{4}N+1\right)\frac{1}{\alpha^m} = 2\alpha m - \frac{2\alpha(\alpha^m-1)}{(\alpha-1)\alpha^m} + \left(\frac{3}{4}N+1\right)\frac{1}{\alpha^m}. \quad (3)$$

We can prove that the following inequality is true:[3]

$$\left|S^E_{\text{schedule}} - S^A_{\text{schedule}}\right| < \frac{3\alpha - 1}{\alpha - 1}. \quad (4)$$

As we will discuss later, the typical value of α is $\alpha \approx 2.5$. As a result we have $|S^E_{\text{schedule}} - S^A_{\text{schedule}}| < 4.4$. Inequality (4) guarantees that our approximation S^A_{schedule} is close enough to the exact S^E_{schedule}. Consequently, if we minimize S^A_{schedule}, S^E_{schedule} is also *roughly* minimized.

According to the basic idea of the algorithm, the following condition has to be satisfied in order to guarantee the existence of non-overlapping perfect matchings in the indicator matrix $\boldsymbol{B}$:

$$N_m = \sum_{i=1}^{m} 2\gamma(i) = \sum_{i=1}^{m} 2\left\lceil \alpha^i - 1\right\rceil < \frac{N}{4}. \quad (5)$$

[3] See Appendix A. Assume that $T>N$, $N/T+2m/\alpha^m \le 1$, and use the boundary condition (6) to determine (α,m) in α^i-SCALE.

Particularly, α^i-SCALE uses the following equation as its boundary condition (constraint):

$$\sum_{i=1}^{m} 2\alpha^i = \frac{N}{4} - 1. \quad (6)$$

According to Lemma 1 in Appendix A, when using (6) as the boundary condition, we have

$$\frac{N}{4} - 1 - 2m \le N_m < \frac{N}{4}. \quad (7)$$

Thus not only (5) is satisfied, but also N_m is guaranteed to be close enough to $N/4$ in α^i-SCALE. This ensures that large entries in $\boldsymbol{C(T)}$ are sufficiently analyzed and scheduled.

Up to this point, the flow of α^i-SCALE can be concisely summarized as follows. Under the constraint of the boundary condition (6), we find an (α,m) pair to minimize our objective function S^A_{schedule} in (3). α can be a fraction but m has to be an integer (because m is the number of iterations). At the same time, (α,m) pair is expected to be dynamically optimized for different switch size N (so as to approximately minimize

α^i-SCALE algorithm online part

Step 1. Initialization: Create an $N\times N$ all-1 matrix $\boldsymbol{B}=\{b_{st}\}$. Get (α,m) pair from α^i-SCALE offline calculation. Use $\boldsymbol{C}=\{c_{st}\}$ to denote the traffic matrix. Set i=1, *scale*=α and *count*=1.
Step 2. Select large entries: In an $N\times N$ all-0 matrix $\boldsymbol{L}=\{l_{st}\}$, set l_{st}=1 if $c_{st}>T/scale$ and b_{st}=1.
Step 3. Edge-coloring: Construct a bipartite unigraph $\boldsymbol{G_L}$ from $\boldsymbol{L}$. Edge-color $\boldsymbol{G_L}$ into $\gamma(i)$ colors where $\gamma(i)$ can also be calculated from (1). Set the color identifier k=1.
Step 4. Partition edges: For color k, equally divide its edges into two sets $\boldsymbol{E_a}$ and $\boldsymbol{E_b}$. If the total number of edges of color k is an odd number, $\boldsymbol{E_a}$ can have one more edge than $\boldsymbol{E_b}$.
Step 5. Schedule E_a: For each edge in $\boldsymbol{E_a}$, shadow its corresponding lines (row and column) in $\boldsymbol{B}$. Find a maximum-size matching $\boldsymbol{M_B}$ in the remaining un-shadowed sub-matrix of $\boldsymbol{B}$. Then un-shadow all the lines of $\boldsymbol{B}$. The matching $\boldsymbol{M_B}$ combines with all the edges in $\boldsymbol{E_a}$ to form a perfect matching $\boldsymbol{P}_i$. Set $\boldsymbol{P}_i$'s weight as $\omega(i)$ defined in (1). Set $\boldsymbol{B}-\boldsymbol{P}_i\rightarrow\boldsymbol{B}$ and $i+1\rightarrow i$. Set c_{st}=0 in $\boldsymbol{C}$ if b_{st}=0.
Step 6. Schedule E_b: Repeat *Step 5* for $\boldsymbol{E_b}$.
Step 7. Loop over colors: Set $k+1\rightarrow k$. Loop to *Step 4* until all the $\gamma(i)$ colors are scheduled.
Step 8. Outer-loop iteration: Set $scale\times\alpha\rightarrow scale$ and $count+1\rightarrow count$. Loop to *Step 2* until $count>m$.
Step 9. Schedule small entries in C: Repeat this step to sequentially extract the remaining $N-N_m$ perfect matchings (by performing maximum-size matching) from the indicator matrix $\boldsymbol{B}$, where N_m is the total number of configurations determined in *Steps 1-8*. After each perfect matching is extracted, deduct it from $\boldsymbol{B}$ and set the constant $\omega(m+1)$ as its weight.

α^i-SCALE algorithm offline part: (α,m) searching procedure

1) Search for α's approximate value α:* Search for α^*. α^* is the value that minimizes S^A_{schedule} in (8).
2) Calculate m's approximate value m:* Apply α^* to (9) to calculate m^*.
3) Determine m: Set m=ROUND(m^*), where ROUND( ) represents the function of taking the nearest integer.
4) Determine α: Substitute m in (10) by the value found in *3)* and solve the equation for α.

Fig. 2. α^i-SCALE algorithm.

S^E_{schedule} in (2)). The values of (α,m) can then be substituted into the online part of α^i-SCALE in Fig. 2 to schedule $\boldsymbol{C}(\boldsymbol{T})$.

C. Determining (α,m) Pair

We now consider how to determine the suitable (α,m) pair for a specific N. There are many possible methods. Aiming at providing a complete solution for α^i-SCALE, we suggest the (α,m) searching procedure as listed in Fig. 2. It adopts the following formulas (8)-(10). The correctness proof is given in Appendix B.

$$\begin{cases} S^A_{\text{schedule}} = 2\alpha\dfrac{\lg[(N+4)\alpha-(N-4)]-\lg 8\alpha}{\lg\alpha} - \dfrac{2\alpha(N-4)(\alpha-1)}{(N+4)\alpha^2-2N\alpha+N-4} \\ \qquad\qquad + \dfrac{6N\alpha+8\alpha}{(N+4)\alpha-(N-4)} \qquad (8) \\ m = \dfrac{\lg[(N+4)\alpha-(N-4)]-\lg 8\alpha}{\lg\alpha} \qquad (9) \\ \dfrac{2\alpha(\alpha^m-1)}{\alpha-1} = \dfrac{N}{4}-1 \qquad (10) \end{cases}$$

Fig. 3 plots S^A_{schedule} in (8). We can see that S^A_{schedule} is usually minimized around $\alpha\approx 2.5$ instead of $\alpha=2$ (note that MIN [5] uses 2^i as the threshold in the i'th iteration to select large entries). It is also very important to note that the performance of $\alpha=2$ in Fig. 3 does not stand for the performance of MIN. The reason is that, all the curves in Fig. 3 satisfy (7), which indicates that the large entries in $\boldsymbol{C}(\boldsymbol{T})$ are scheduled fine enough with as many configurations as possible ($N/4-1-2m\le N_m<N/4$). For MIN, usually the first m outer-loop iterations generate less configurations and leave more configurations to be weighted by the constant weight. It may increase S_{schedule}. Intuitively, MIN does not guarantee its N_m value to be as close to $N/4$ as α^i-SCALE does. This is because MIN uses a fixed threshold 2^i which cannot self-adjust according to the switch size N. So its schedule for large entries is usually not fine enough and the S_{schedule} performance may be worse than that (shown as $\alpha=2$) in Fig. 3. In the worst case, we can show that MIN may lead to a bias of $2(2^{m+1}-1)$ configurations less than $N/4$.

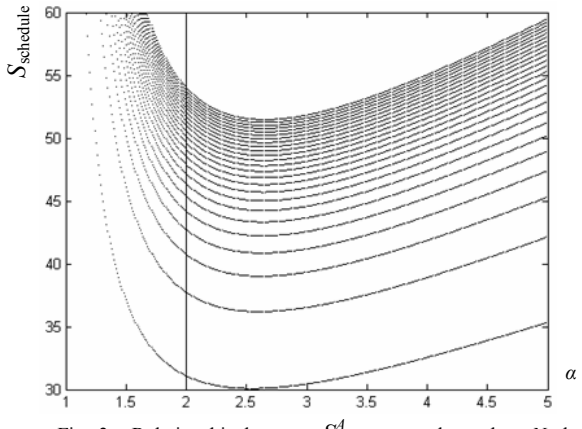

Fig. 3. Relationship between S^A_{schedule} and α when N changes from 200 to 5000 in a step of 200.

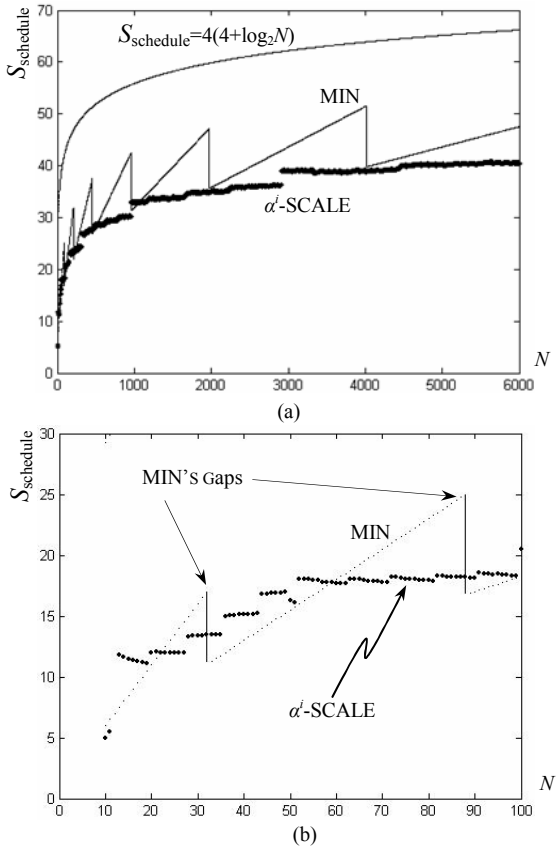

Fig. 4. Performance comparison for MIN and α^i-SCALE.

IV. Performance Analysis

Fig. 4 shows the S_{schedule} bounds due to the inefficient scheduling for MIN and α^i-SCALE. The bound for MIN in the figure is derived and plotted by strictly following the steps of MIN in [5]. [4] It is clear that the original bound $S_{\text{schedule}}=4(4+\log_2 N)$ in [5] is too conservative (and thus inaccurate) to represent MIN's performance. For example, when N=460, S_{schedule}=26.94 is sufficient for MIN. However, the bound from [5] is $S_{\text{schedule}}=4(4+\log_2 N)$=51.38.

From Fig. 4a, it is also obvious that α^i-SCALE (shown in the broad-brush curve) outperforms MIN in general. For example, when N=200, S_{schedule} needed by MIN is $2+6\times(1/2)+14\times(1/4)+(1/8)\times(200-2-6-14)=30.75$. However, for the same switch size, because (α,m)=(2.5,3), S_{schedule} needed by α^i-SCALE is just $4+12\times(1/2.5)+30\times(1/2.5^2)+(1/2.5^3)\times(200-4-12-30)=23.45$.

The above performance difference is due to the dynamic scale function $f(N,i)=\alpha^i$ adopted by α^i-SCALE. α^i-SCALE

[4] The i-th outer-loop iteration generates $2(2^i-1)$ configurations with each weighted by $T/2^{i-1}$. The boundary condition is $N_m<N/4$. The remaining $N-N_m$ configurations are weighted by a constant $T/2^m$, where m is the number of outer-loop iterations needed to determine the first N_m configurations. For example, when N=460 ($N/4$=115), m=5 iterations are needed and 2, 6, 14, 30, 62 configurations are determined in the first 5 iterations respectively. Thus $S_{\text{schedule}}=(1/T)\times[\sum_{i=1}^{m}2(2^i-1)T/2^{i-1}+(460-2-6-14-30-62)T/2^5]$=26.94. The effects of roof and floor functions are ignored for simplicity.

guarantees that N_m (the total number of configurations used to schedule large entries) is close enough to its maximum possible value of $N/4$ (i.e. $N/4-1-2m \le N_m < N/4$). In comparison, MIN does not have such a guarantee. In the above example, N_m for α^i-SCALE is 4+12+30=46 ($49-2\times3 \le N_m < 50$), but MIN uses only 2+6+14=22 configurations, which is much smaller than $N/4=50$. This increases the scheduling inefficiency of MIN.

Fig. 4b shows the performance of MIN and α^i-SCALE for small N. In this range, although α^i-SCALE is less advantageous, it does complement MIN's performance. Since α^i-SCALE involves approximation, this explains its poorer-than-MIN performance in half of the switch size range studied.

V. Conclusion

In this paper, we showed that the speedup bound $S_{\text{schedule}} = 4(4+\log_2 N)$ given in [5] does not accurately represent the performance of MIN algorithm. We recalculated the actual performance from MIN and got a much lower S_{schedule}. A new α^i-SCALE algorithm was proposed for performance guaranteed OPS scheduling, which also uses the minimum number (N) of configurations to minimize traffic delay. By employing a dynamic scale function, the new algorithm is optimized for different switch sizes. Our results showed that α^i-SCALE pushes the speedup bound to an even lower level in general, while for small and medium size OPS it effectively complements the performance of MIN.

Future work may take three possible directions. The first one is based on the same framework presented in this paper. The goal is to find another better scale function or some discrete series to analyze the traffic matrix. The second one is to devise some new algorithms which totally abandon the framework of MIN and α^i-SCALE. The third direction is to enhance the OPS architecture discussed in Section II. For example, we can consider a parallel switching architecture, in which some switching layers work in transmission phase while others are reconfigured. This extra space diversity can help to overcome the difficulties involved in realizing speedup in time domain.

Appendix A

Correctness Proof of Inequality (4)

Lemma 1:

If formula (6) is used as the boundary condition for α^i-SCALE, N_m must satisfy the following inequality:

$$\frac{N}{4}-1-2m \le N_m < \frac{N}{4}.$$

Proof:

Because

$$\sum_{i=1}^{m} 2(\alpha^i - 1) \le \left(N_m = \sum_{i=1}^{m} 2\lceil \alpha^i - 1 \rceil \right) \le \sum_{i=1}^{m} 2[(\alpha^i - 1) + 1] = \sum_{i=1}^{m} 2\alpha^i , \quad (11)$$

According to (11) and (6) we have

$$\frac{N}{4} - 1 - 2m \le N_m \le \frac{N}{4} - 1 . \quad (12)$$

That is

$$\frac{N}{4}-1-2m \le N_m < \frac{N}{4}.$$

□

We define

$$\Omega = \left(\sum_{i=1}^{m} 2\lceil \alpha^i - 1 \rceil \times \frac{1}{\alpha^{i-1}} + (N - N_m)\frac{1}{\alpha^m} \right), \quad (13)$$

Because

$$N_m = \sum_{i=1}^{m} 2\lceil \alpha^i - 1 \rceil,$$

from (13) and (2) we have

$$\Omega - \frac{N}{T} = \Omega - \frac{1}{T}\left(\sum_{i=1}^{m} 2\lceil \alpha^i - 1 \rceil + N - N_m \right) \le S^E_{\text{schedule}} \le \Omega . \quad (14)$$

According to (13) and (3), there exists

$$\left| \Omega - S^A_{\text{schedule}} \right| = \left| \left[\sum_{i=1}^{m} \frac{2}{\alpha^{i-1}} \left(\lceil \alpha^i - 1 \rceil - (\alpha^i - 1) \right) \right] + \frac{1}{\alpha^m}\left(\frac{N}{4} - 1 - N_m \right) \right| . \quad (15)$$

From (12) we know that both terms on the right hand side of (15) are non-negative. According to (12) and (15) we have

$$\left| \Omega - S^A_{\text{schedule}} \right| \le \sum_{i=1}^{m} \frac{2}{\alpha^{i-1}} + \frac{2m}{\alpha^m} = 2\frac{(\alpha^m - 1)\alpha}{\alpha^m(\alpha - 1)} + \frac{2m}{\alpha^m} < \frac{2\alpha}{\alpha - 1} + \frac{2m}{\alpha^m} . \quad (16)$$

From (14) and (16) we get

$$\left| S^E_{\text{schedule}} - S^A_{\text{schedule}} \right| = \left| (S^E_{\text{schedule}} - \Omega) + (\Omega - S^A_{\text{schedule}}) \right|$$

$$\le \left| S^E_{\text{schedule}} - \Omega \right| + \left| \Omega - S^A_{\text{schedule}} \right| < \frac{N}{T} + \frac{2\alpha}{\alpha - 1} + \frac{2m}{\alpha^m} .$$

Let $T > N$. Usually $2m/\alpha^m$ is a very small value. Further assume that $N/T + 2m/\alpha^m \le 1$. We can see that the inequality (4), as rewritten below, is true for any α and N.

$$\left| S^E_{\text{schedule}} - S^A_{\text{schedule}} \right| < 1 + \frac{2\alpha}{\alpha - 1} = \frac{3\alpha - 1}{\alpha - 1} .$$

Appendix B

Correctness Proof of (α,m) Searching Procedure

S^A_{schedule}, α, m and N are linked together by a complex function (3). It is difficult to entirely substitute α by m in S^A_{schedule} expression. However, substituting m by α is quite easy. Thus, we first substitute m by α in S^A_{schedule} expression (3) to reduce the number of variables and to get (8), in which α is the sole variable for any specific N. Taking minimizing S^A_{schedule} as our goal, we can find a solution α^* for (8) by searching calculation. According to our previous analysis, this value of α^* also approximately minimizes S^E_{schedule} in (2). But, after we get α^*, the corresponding m^* calculated from (9) is usually not an integer. However, the number of iterations m has to be an integer. Thus we let m be the nearest integer of m^* and calculate α again, but this time we use the boundary condition (10) (equivalent to (6)) to calculate α in order to guarantee that the algorithm generates as many configurations as possible in the first m outer-loop iterations (This is ensured by (7)). At this point, we get (α,m). This (α,m) pair allows the algorithm to

generate $N/4-1-2m \le N_m < N/4$ configurations in the first m outer-loop iterations and approximately minimizes S^A_{schedule} and S^E_{schedule} (because $\alpha \approx \alpha^*$).

According to the boundary condition (6), we have

$$\sum_{i=1}^{m} 2\alpha^i = \frac{2\alpha(\alpha^m - 1)}{\alpha - 1} = \frac{N}{4} - 1 .$$

The above equation is actually identical with (10). From this condition, it is easy to see that

$$\alpha^m = \frac{(N+4)\alpha - (N-4)}{8\alpha} . \quad (17)$$

This directly leads to (9). i.e.

$$m = \log_\alpha \frac{(N+4)\alpha-(N-4)}{8\alpha} = \frac{\lg[(N+4)\alpha-(N-4)]-\lg 8\alpha}{\lg\alpha} .$$

Substituting m and α^m in (3) by (9) and (17), we get

$$S^A_{\text{schedule}} = 2\alpha m - \frac{2\alpha(\alpha^m - 1)}{(\alpha-1)\alpha^m} + \left(\frac{3}{4}N + 1\right)\frac{1}{\alpha^m}$$

$$= 2\alpha\frac{\lg[(N+4)\alpha-(N-4)]-\lg 8\alpha}{\lg\alpha} - \frac{2\alpha(N-4)(\alpha-1)}{(N+4)\alpha^2-2N\alpha+N-4} + \frac{6N\alpha+8\alpha}{(N+4)\alpha-(N-4)} .$$

Thus (8) is correct. Consequently, the group of formulas used in α^i-SCALE (i.e. (8)-(10)) is correct. In fact, because (10) is equivalent to the boundary condition (6), according to Lemma 1 in Appendix A, N_m of α^i-SCALE is always smaller than $N/4$ but greater than or equal to $N/4-1-2m$. α^i-SCALE takes this as an advantage to analyze large entries in $\boldsymbol{C(T)}$ with sufficient granularity, and to generate a fine schedule.

It is obvious that (8) depends only on α (consider N as a system constant). We can find an α^* to minimize S^A_{schedule} and estimate the corresponding m^* using (9). However, m^* is usually a fraction. So, we have to take m=ROUND(m^*) and calculate the new α value again accordingly. In order to satisfy the boundary condition (6), we use its counterpart (10) to calculate α. This ensures that $N/4-1-2m \le N_m < N/4$.

A further analysis indicates that the following restriction in Lemma 2 exists for α^i-SCALE algorithm:

Lemma 2:

For any switch size N and $\alpha \ge 2$, any (α, m) pair in α^i-SCALE has to satisfy the following inequality:

$$\left(\frac{N-4}{8}\right)^{\frac{1}{m}} > \alpha > \left(\frac{N-4}{16}\right)^{\frac{1}{m}} .$$

Proof:

From the boundary condition (6), we know that

$$\sum_{i=1}^{m} 2\alpha^i = \frac{N}{4} - 1 = \frac{N-4}{4} \text{ and } \sum_{i=1}^{m} \alpha^i = \frac{N-4}{8} .$$

Thus

$$\alpha^m < \frac{N-4}{8} . \quad (18)$$

At the same time, for any $\alpha \ge 2$, there exists $\sum_{i=1}^{m-1}\alpha^i < \alpha^m$, thus

$$2\alpha^m > \sum_{i=1}^{m-1}\alpha^i + \alpha^m = \sum_{i=1}^{m}\alpha^i = \frac{1}{2}\left(\frac{N}{4} - 1\right) .$$

That is

$$\alpha^m > \frac{N-4}{16} . \quad (19)$$

Combining (18) and (19), we have

$$\left(\frac{N-4}{8}\right)^{\frac{1}{m}} > \alpha > \left(\frac{N-4}{16}\right)^{\frac{1}{m}} . \quad (20)$$

□

The above formula (20) holds for both before ROUND(m^*) and after ROUND(m^*). Let

$$\alpha_{\max} = \left(\frac{N-4}{8}\right)^{\frac{1}{m}} \text{ and } \alpha_{\min} = \left(\frac{N-4}{16}\right)^{\frac{1}{m}} ,$$

We have

$$\frac{\alpha_{\max}}{\alpha_{\min}} = 2^{\frac{1}{m}} \approx 1 . \quad (21)$$

For ROUND( ) function, it can change m's value by at most 0.5 and thus $1/m$ changes only a little. As a result, $\alpha_{\max}$ and $\alpha_{\min}$ will be quite stable. In this case, formula (21) indicates that our final α is close enough to α^*, that is $\alpha \approx \alpha^*$. Consequently, (α, m) pair can still approximately minimize S^A_{schedule} and S^E_{schedule}.

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] 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.

[5] 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.

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

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

[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.