Can We Schedule Traffic More Efficiently in Optical Packet Switches?

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

***Abstract*—We consider traffic scheduling in non-blocking electronic-buffered optical packet switches (OPS) with bounded packet delay. Due to the reconfiguration overhead of the switch fabric, the two commonly used optimization objectives, minimizing packet delay and minimizing switch speedup, conflict with each other. Intelligent scheduling algorithms have been designed to provide tradeoff between these two objectives. In this paper, we propose a more efficient approach to schedule OPS traffic, resulting in significantly reduced speedup and/or packet delay. However, our approach is based on a very interesting conjecture, which has not been strictly proved so far. We would like to put forward this conjecture as an open question, and call for a proof or disproof.**

***Index Terms*—Conjecture, optical packet switch (OPS), performance guaranteed switching, scheduling.**

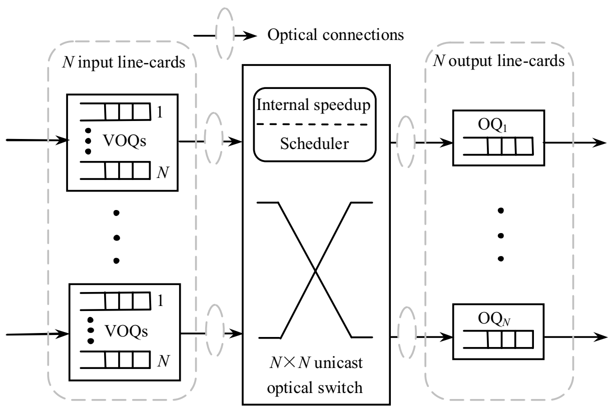


Fig. 1. A scalable high speed optical packet switch.

I. Introduction

RECENT progress on optical switching technologies [1-4] has enabled the implementation of electronic-buffered optical packet switches (OPS) as shown in Fig. 1. The core of this architecture is the optical switch fabric, which can efficiently provide huge switching capacity as demanded by the backbone routers in the Internet. Since optical connections (i.e. optical fibers) are used to interconnect the input/output line-cards with the central switch fabric, the input/output line-cards can be distributed into several racks, which may locate at hundreds of meters away from each other. As a result, power consumption in each rack can be reduced, and the switch becomes more scalable.

On the other hand, the optical switch fabric usually needs some guard time to change its inter-connection pattern from one to another, and to synchronize the signals arriving at the input ports [5]. This guard time is called *reconfiguration overhead*. During this period, no packet can be transmitted across the switch fabric. Accordingly, packet transmission rate in the switch fabric must be faster than the external line-rate (i.e. a *speedup* is required in the switch fabric) in order to achieve *performance guaranteed switching* (i.e. 100% throughput with bounded packet delay) [6-12]. It is shown in [6, 8] that minimizing speedup and minimizing delay are two conflicting goals, where a higher speedup gives a shorter packet delay and vice versa.

Based on switch architectures similar to Fig. 1, several algorithms have been recently proposed [6-12] to schedule OPS traffic with guaranteed switching performance. Among them, MIN [6], α^i-SCALE [9] and QLEF [10] aim primarily at minimizing the packet delay, whereas reducing speedup is a secondary objective. DOUBLE [6] is the first algorithm that allows the tradeoff between speedup and delay. Let N denote the switch size and N_S be the (maximum) number of switch configurations required for scheduling. DOUBLE needs no more than N_S=2N configurations to schedule any *legitimate* traffic matrix, with a speedup of S_{schedule}=2 (detailed in Section II). However, this algorithm does not consider the amount of reconfiguration overhead δ in its scheduling decision, and thus it is not optimized for switches with different δ. Besides, N_S=2N only represents a single point in the solution space [6, 8], and the characteristics for other N_S values are not studied in [6]. To address those issues, ADAPTIVE [8] is proposed. It is shown [8] that DOUBLE can be regarded as a special case of ADAPTIVE at N_S=2N.

In this paper, we explore the possibility of beating the performance of DOUBLE and ADAPTIVE. We show that this can be achieved based on a very interesting conjecture. We put forward this conjecture as an open question, and hope that a proof or disproof can be found soon.

The rest of the paper is organized as follows. In Section II, we review the generic scheduling procedure [6-12] and the existing scheduling algorithms DOUBLE and ADAPTIVE. In

This work is supported by Hong Kong Research Grant Council Earmarked Grant HKU 7032/01E.

Bin Wu, Xin Wang 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, xinwang, kyeung}@eee.hku.hk).

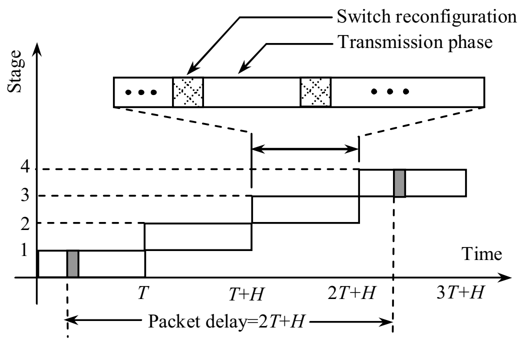


Fig. 2. Optical packet switch scheduling stages.

Section III, we propose an approach to improve the scheduling efficiency. Section IV gives some further discussion, and we conclude the paper in Section V.

II. Traffic Scheduling and Speedup-Delay Tradeoff

A. Scheduling Procedure

The generic four-stage scheduling procedure as shown in Fig. 2 is followed. In Stage 1, incoming packets are periodically accumulated in the input buffers over T time slots to construct an $N\times N$ traffic matrix $\boldsymbol{C(T)}=\{c_{ij}\}$. Each entry c_{ij} denotes the number of packets received at input i and destined to output j. $\boldsymbol{C(T)}$ is *legitimate* if each of its line sums (i.e. row sum or column sum) is no larger than T. Throughout the paper, we only consider legitimate $\boldsymbol{C(T)}$. The scheduling algorithm takes H time slots in Stage 2 to generate N_S configurations $\boldsymbol{P_n}=\{p^{(n)}_{ij}\}$, $n\in\{1, \ldots, N_S\}$, each weighted by ϕ_n, to cover $\boldsymbol{C(T)}$. "Cover" means that $\sum^{N_S}_{n=1}\phi_n p^{(n)}_{ij}\geq c_{ij}$ for any $i,j\in\{1, \ldots, N\}$. $\boldsymbol{P_n}$ is an $N\times N$ permutation matrix with at most a single "1" in each line (row or column). $p^{(n)}_{ij}=1$ indicates that a packet can be sent from input i to output j in one slot; $p^{(n)}_{ij}=0$ otherwise. In Stage 3, the switch fabric is reconfigured according to the N_S configurations obtained in Stage 2. An internal switch fabric speedup S is applied, resulting in compressed/shortened time slots, to ensure that this stage occupies only T (regular) slots. The fabric holds each configuration $\boldsymbol{P_n}$ for ϕ_n *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. 2, we can see that the bounded delay of any packet is $2T+H$ slots. Because δN_S slots are used to reconfigure the switch for N_S times in Stage 3, only $T-\delta N_S$ slots are left for transmitting $\boldsymbol{C(T)}$. Since there are at most T packets waiting at each input port for transmission, a speedup factor $S_{\text{reconfigure}}=T/(T-\delta N_S)$ is necessary to compensate for the idle time caused by reconfigurations. At the same time, the scheduling algorithm may produce some empty slots (i.e. underutilize the bandwidth provided by the configurations [6-12]). As a result, more than T compressed slots are usually needed in Stage 3 to transmit $\boldsymbol{C(T)}$. Therefore another speedup factor

$$S_{\text{schedule}}=\frac{1}{T}\sum_{n=1}^{N_S}\phi_n \tag{1}$$

is required to compensate for the inefficient scheduling. In fact, S_{schedule} denotes the efficiency of the scheduling algorithm adopted. A smaller S_{schedule} indicates a more efficient scheduler (with less empty slots in the schedule). The overall internal speedup S is then given by

$$S=S_{\text{reconfigure}}\times S_{\text{schedule}}=\frac{T}{T-\delta N_S}S_{\text{schedule}}=\frac{1}{T-\delta N_S}\sum_{n=1}^{N_S}\phi_n\,. \tag{2}$$

B. Speedup-Delay Tradeoff and Scheduling Algorithms

For a given $\boldsymbol{C(T)}$, we divide it by $T/(N_S-N)$ [1] to get a quotient matrix $\boldsymbol{Q}=\{q_{ij}\}$and a residue matrix $\boldsymbol{R}=\{r_{ij}\}$:

$$\boldsymbol{C(T)}=\frac{T}{N_S-N}\times\boldsymbol{Q}+\boldsymbol{R}\,. \tag{3}$$

Since each line of $\boldsymbol{C(T)}$ sums to at most T, the maximum line sum of $\boldsymbol{Q}$ is at most N_S-N. So we can apply edge-coloring [13] to the bipartite multigraph of $\boldsymbol{Q}$, and get N_S-N configurations to cover $\boldsymbol{Q}$ [6-8]. On the other hand, all the entries in $\boldsymbol{R}$ are not larger than $T/(N_S-N)$. So, $\boldsymbol{R}$ can be covered by *any* N non-overlapping configurations (i.e. any two of them do not cover the same entry), with each weighted by $T/(N_S-N)$. All in all, $\boldsymbol{C(T)}$ can be covered by $(N_S-N)+N=N_S$ configurations, each weighted by $\phi_n=T/(N_S-N)$. From (1), S_{schedule} can be found.

$$S_{\text{schedule}}=\frac{1}{T}\sum_{n=1}^{N_S}\phi_n=\frac{1}{T}\times\frac{T}{N_S-N}\times N_S=1+\frac{N}{N_S-N}\,. \tag{4}$$

The above formula (4) is referred to as *speedup function* in [8]. In essence, it depicts the tradeoff relationship between the speedup (S_{schedule} due to the inefficient scheduling) and the delay (in terms of N_S). Recall that the bounded delay of any packet is $2T+H$ slots and $T>\delta N_S$. Therefore the minimum achievable delay is given by $2\delta N_S+H$.

DOUBLE [6] requires $N_S=2N$ configurations to cover $\boldsymbol{C(T)}$. This is obtained by replacing N_S in (3) by $2N$ to get $\boldsymbol{C(T)}=[T/N]\times\boldsymbol{Q}+\boldsymbol{R}$. N configurations are required to cover $\boldsymbol{Q}$ and $\boldsymbol{R}$ respectively, and each configuration is equally weighted by $\phi_n=T/N$. From (4), DOUBLE achieves $S_{\text{schedule}}=2$. Fig. 3 gives an example of DOUBLE execution.

Unlike DOUBLE, ADAPTIVE [8] substitutes (4) into (2), and minimizes the overall speedup S by solving

$$\frac{\partial S}{\partial N_S}=0\,. \tag{5}$$

Therefore the schedule generated by ADAPTIVE is optimized with respect to the value of δ.

III. Improving Scheduling Efficiency

In this section, we aim at achieving a better scheduling efficiency than DOUBLE and ADAPTIVE. Since DOUBLE is a special case of ADAPTIVE at $N_S=2N$, for simplicity, we only focus on DOUBLE below. We further assume that the switch

[1] If $T/(N_S-N)$ is not an integer, use $\lceil T/(N_S-N)\rceil$ as the substitute [8].

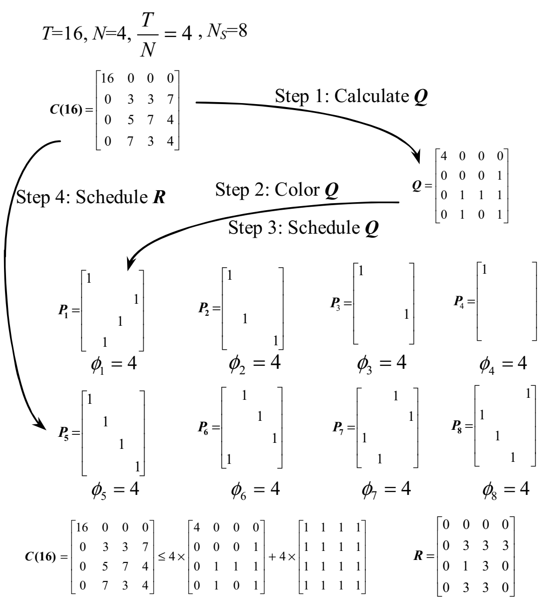


Fig. 3. An example of DOUBLE execution. The all-1 matrix used to cover $\boldsymbol{R}$ equals to the sum of the N non-overlapping configurations ($\boldsymbol{P}_5$-$\boldsymbol{P}_8$).

$$\boldsymbol{C}(16)=4\times\begin{bmatrix}4&0&0&0\\0&0&0&1\\0&1&1&1\\0&1&0&1\end{bmatrix}+\begin{bmatrix}0&0&0&0\\0&3&3&3\\0&1&3&0\\0&3&3&0\end{bmatrix}\longrightarrow \boldsymbol{C}(16)\le 4\times\begin{bmatrix}4&0&0&0\\0&\textcircled{1}&0&\textcircled{2}\\0&1&\textcircled{2}&1\\0&1&\textcircled{1}&1\end{bmatrix}+\begin{bmatrix}0&0&0&0\\0&\textcircled{0}&3&\textcircled{0}\\0&1&\textcircled{0}&0\\0&3&\textcircled{0}&0\end{bmatrix}$$

Fig. 4. Move the circled LERs from $\boldsymbol{R}$ to $\boldsymbol{Q}$.

size N is an even number.

A. Observation and Motivation

In DOUBLE, the traffic matrix $\boldsymbol{C(T)}$ is decomposed as $\boldsymbol{C(T)}=[T/N]\times\boldsymbol{Q}+\boldsymbol{R}$. For any $r_{ij}\in\boldsymbol{R}$, if $r_{ij}>T/(2N)$, we call it an *LER* (large entry in $\boldsymbol{R}$). Otherwise it is an *SER* (small entry in $\boldsymbol{R}$). We have the following Lemma 1 (proved in Appendix A).

Lemma 1: In DOUBLE, if a particular line (row i or column j) of $\boldsymbol{R}$ contains k LERs, then in $\boldsymbol{Q}$ we have

$$\sum_{j=1}^{N} q_{ij}\le N-\left\lceil\frac{k}{2}\right\rceil \text{ for row } i, \text{ or } \sum_{i=1}^{N} q_{ij}\le N-\left\lceil\frac{k}{2}\right\rceil \text{ for column } j.$$

For example, the second row of $\boldsymbol{R}$ in Fig. 3 contains k=3 LERs ($>T/(2N)$=2). Then the entries in the second row of $\boldsymbol{Q}$ sum to at most $N-\lceil k/2\rceil$=2.

Based on Lemma 1, we can *move* some packets from $\boldsymbol{R}$ to $\boldsymbol{Q}$, while keeping the maximum line sum of $\boldsymbol{Q}$ not more than N. Note that each configuration in DOUBLE is equally weighted by $\phi_n=T/N$. Without loss of generality, if row i of $\boldsymbol{R}$ contains k LERs, we can move *at most* $\lceil k/2\rceil$ of these LERs to $\boldsymbol{Q}$ by setting them to 0*s* in $\boldsymbol{R}$, and at the same time increasing the corresponding entries in $\boldsymbol{Q}$ by one. Fig. 4 shows an example based on the $\boldsymbol{Q}$ and $\boldsymbol{R}$ in Fig. 3. We use $\boldsymbol{Q}'$ and $\boldsymbol{R}'$ to denote the modified $\boldsymbol{Q}$ and $\boldsymbol{R}$. We can see that $\boldsymbol{Q}'$ can still be covered by N configurations with a weight $\phi_n=T/N$ each. So we can pack more packets in the N configurations used to cover the quotient matrix (than DOUBLE). On the other hand, since some LERs are moved to $\boldsymbol{Q}$, it may not be necessary to use another N *equally weighted* configurations (with $\phi_n=T/N$) to cover $\boldsymbol{R}'$.

The above observation motivates us to explore a more efficient scheduling algorithm than DOUBLE. In fact, there may be at most N LERs in each line of the original $\boldsymbol{R}$. From Lemma 1, "half" of these LERs in each line of $\boldsymbol{R}$ can be moved to $\boldsymbol{Q}$, while keeping the maximum line sum of $\boldsymbol{Q}'$ not more than N. Therefore, it is reasonable to expect that each line of $\boldsymbol{R}'$ contains at most N/2 LERs after packet moving. As a result, we may be able to find N/2 non-overlapping configurations, each weighted by $\phi_n=T/N$, to cover all the remaining LERs in $\boldsymbol{R}'$. At the same time, we may find another set of N/2 non-overlapping configurations, each with a *reduced* weight of $\phi_n=T/(2N)$, to cover the remaining SERs. If this can be done, then S_{schedule} of DOUBLE can be reduced to

$$S_{\text{schedule}}=\frac{1}{T}\sum_{n=1}^{2N}\phi_n=\frac{1}{T}\times\left(\frac{T}{N}\times N+\frac{T}{N}\times\frac{N}{2}+\frac{T}{2N}\times\frac{N}{2}\right)=1.75 \qquad (6)$$

B. Issues

To achieve the goal mentioned above, we need to solve the following issues.

- Determine the set of LERs to be moved to $\boldsymbol{Q}$, such that *each* line sum of $\boldsymbol{Q}'$ does not exceed N, and $\boldsymbol{R}'$ contains at most N/2 LERs in *each* line.
- Among the N non-overlapping configurations used to cover $\boldsymbol{R}'$, N/2 of them should cover all the remaining LERs in $\boldsymbol{R}'$, and the other N/2 configurations should cover all the SERs not yet covered by the first N/2 configurations.

Generally, it is not easy to determine the set of LERs that should be moved to $\boldsymbol{Q}$. This can be seen from the example in Fig. 5. Assume all the non-zero entries (3*s*) in $\boldsymbol{R}$ are LERs. The number next to each line is the number of LERs that can be moved from this line to $\boldsymbol{Q}$, which is obtained from $\lceil k/2\rceil$ based on Lemma 1. If the four circled entries are moved, then we cannot further move any other LERs without violating the quota of the corresponding line. At this point, the last row still contains more than N/2 LERs. For larger switch size N, it will be more difficult to figure out a proper set of LERs to move.

C. Methodology

We first define two important notions, ***PC****s* (*Predefined Configurations*) and *DHH matrix*. For an $N\times N$ matrix, we can use N predefined non-overlapping configurations (or ***PC****s*) to cover all of its entries. As an example, eight non-overlapping ***PC****s* (i.e. $\boldsymbol{PC}_1\sim\boldsymbol{PC}_8$), as defined in Fig. 6, can be used to cover all the entries of an 8×8 matrix. Note that the number at each entry of this matrix denotes the particular ***PC*** that covers this

$$\boldsymbol{R}=\begin{array}{c}\begin{bmatrix}\textcircled{3}&0&0&0\\0&\textcircled{3}&\textcircled{3}&3\\0&0&\textcircled{3}&0\\3&3&3&0\end{bmatrix}\begin{array}{c}1\\2\\1\\2\end{array}\\ \begin{array}{cccc}1&1&2&1\end{array}\quad\end{array}$$

Fig. 5. A carefully designed mechanism is necessary in order to move LERs properly.

$$\left[\begin{array}{cccc|cccc}5&8&7&6&2&\textcircled{3}&4&1\\6&5&8&7&\textcircled{3}&4&1&2\\7&6&5&8&4&1&2&\textcircled{3}\\8&7&6&5&1&2&\textcircled{3}&4\\\hline 4&\textcircled{3}&2&1&5&6&7&8\\\textcircled{3}&2&1&4&8&5&6&7\\2&1&4&\textcircled{3}&7&8&5&6\\1&4&\textcircled{3}&2&6&7&8&5\end{array}\right]$$

Fig. 6. A possible set of predefined configurations for an 8×8 matrix.

entry. For easy reading, the entries covered by $\boldsymbol{PC}_3$ are circled in Fig. 6.

Given an arbitrary 0/1 matrix, we can use two lines to partition it into four $(N/2)\times(N/2)$ zones/sub-matrices $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ and $\boldsymbol{D}$ as shown in Fig. 9 (and Fig. 10) in Appendix B. For each row/column, if the number of 1*s* in zone $\boldsymbol{A}$ or $\boldsymbol{C}$ is no less than that in zone $\boldsymbol{B}$ or $\boldsymbol{D}$, *or is less by at most one*, then this 0/1 matrix is called a *DHH matrix*. In other words, the two diagonal zones ($\boldsymbol{A}$ and $\boldsymbol{C}$) of a DHH matrix contain at least "half" number of 1*s for each row and column*. (Please refer to Appendix B for a more rigorous definition.)

Our approach in improving the scheduling efficiency (i.e. minimizing S_{schedule}) is based on the concepts of $\boldsymbol{PC}$ and DHH matrix. We first convert the residue matrix $\boldsymbol{R}=\{r_{ij}\}$ into a 0/1 indicator matrix $\boldsymbol{\Omega}=\{\omega_{ij}\}$ such that $\omega_{ij}=1$ if r_{ij} is an LER and $\omega_{ij}=0$ otherwise. The *DHH conjecture* given in Appendix B says that we can always find two permutation matrices $\boldsymbol{U}$ and $\boldsymbol{V}$, such that $\boldsymbol{\Omega}'=\boldsymbol{U\Omega V}$ is a DHH matrix. For the 8×8 matrix, assume that a DHH matrix $\boldsymbol{\Omega}'$ is obtained by $\boldsymbol{\Omega}'=\boldsymbol{U\Omega V}$. Then, $\boldsymbol{PC}_5\sim\boldsymbol{PC}_8$ defined in Fig. 6 (which span over the sub-matrices $\boldsymbol{A}$ and $\boldsymbol{C}$ as illustrated in Fig. 9) can cover "more than half" of the 1*s* for *each row and column* of $\boldsymbol{\Omega}'$. The remaining 1*s* covered by $\boldsymbol{PC}_1\sim\boldsymbol{PC}_4$ "correspond" to LERs that should be moved to $\boldsymbol{Q}$. Since $\boldsymbol{\Omega}'$ is obtained from $\boldsymbol{\Omega}$ after some row/column permutations (i.e. $\boldsymbol{\Omega}'=\boldsymbol{U\Omega V}$), the 1*s* in $\boldsymbol{\Omega}'$ do not directly match the original LER entries. Therefore, we need to invoke an inverse transform to get our desired configurations.

Fig. 7 gives an example, where the execution steps are indexed by the numbers in the dashed circles. In Step 1, we construct the indicator matrix $\boldsymbol{\Omega}$ from $\boldsymbol{R}$. In Step 2, we find two

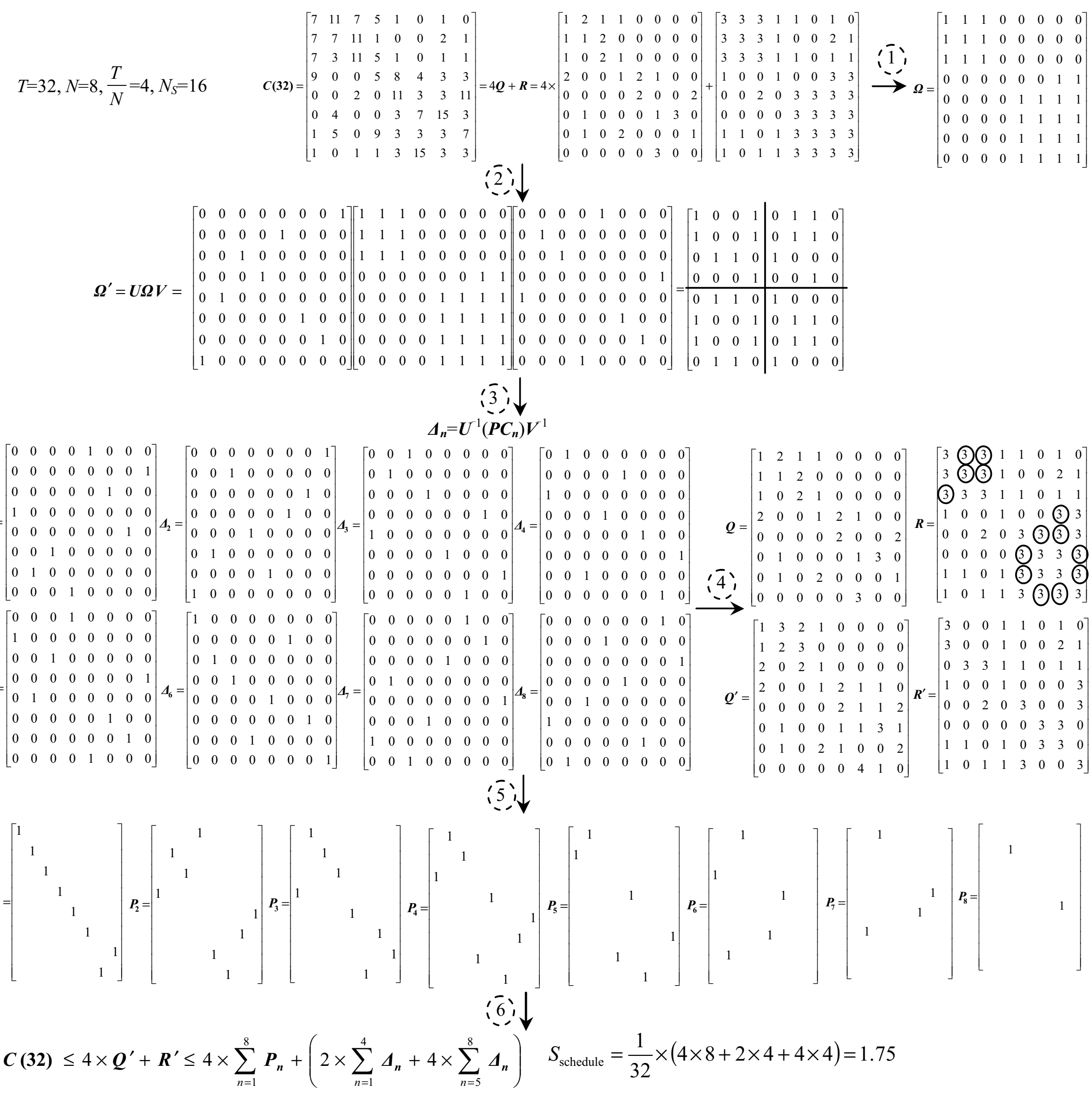


Fig. 7. An illustrative example of our proposed approach.

permutations $\boldsymbol{U}$ and $\boldsymbol{V}$ to permute $\boldsymbol{\Omega}$ into a DHH matrix $\boldsymbol{\Omega}'$. Then, Step 3 imposes $\boldsymbol{U}$ and $\boldsymbol{V}$ on the predefined configurations $\boldsymbol{PC}_1 \sim \boldsymbol{PC}_8$ and uses $\boldsymbol{\Delta}_n=\boldsymbol{U}^{-1}(\boldsymbol{PC}_n)\boldsymbol{V}^{-1}$ (note $\boldsymbol{U}^{-1}=\boldsymbol{U}$ and $\boldsymbol{V}^{-1}=\boldsymbol{V}$ in this particular example) to generate eight configurations $\boldsymbol{\Delta}_1 \sim \boldsymbol{\Delta}_8$. For $\boldsymbol{\Delta}_1 \sim \boldsymbol{\Delta}_4$, if they cover some LERs in the original $\boldsymbol{R}$, then these LERs (circled entries in Step 4) are to be moved to $\boldsymbol{Q}$. Based on Lemma 1, we can set these LERs to 0*s* in $\boldsymbol{R}$, and increase the corresponding entries in $\boldsymbol{Q}$ by one. $\boldsymbol{Q}'$ and $\boldsymbol{R}'$ are thus obtained in Step 4. In Step 5, to cover $\boldsymbol{Q}'$ we simply use the same edge-coloring algorithm [13] as in DOUBLE to determine configurations $\boldsymbol{P}_1 \sim \boldsymbol{P}_8$, each weighted by $T/N=4$. On the other hand, $\boldsymbol{R}'$ can be covered by $\boldsymbol{\Delta}_1 \sim \boldsymbol{\Delta}_8$, with a weight of 4 for $\boldsymbol{\Delta}_5 \sim \boldsymbol{\Delta}_8$, and a reduced weight of $T/(2N)=2$ for $\boldsymbol{\Delta}_1 \sim \boldsymbol{\Delta}_4$. As a result, S_{schedule} can be reduced to 1.75, as shown in Step 6. In fact, this 12.5% reduction in S_{schedule} is independent of switch size N, as stipulated by (6). Note that there is a one-to-one mapping from entry to entry between $\boldsymbol{\Omega}$ and $\boldsymbol{\Omega}'$, which is determined by the linear transform $\boldsymbol{\Omega}'=\boldsymbol{U\Omega V}$. Therefore, the resulting $\boldsymbol{\Delta}_1 \sim \boldsymbol{\Delta}_8$ are non-overlapping, and they can cover every entry of $\boldsymbol{R}'$.

IV. Discussion

If we define LERs as $r_{ij}>T/[2\times(N_S-N)]$ (instead of $r_{ij}>T/(2N)$ for DOUBLE) and decompose $\boldsymbol{C(T)}$ as discussed in Part B of Section II, then the speedup function (4) used in ADAPTIVE [8] can also be reduced to

$$S_{\text{schedule}}=\frac{1}{T}\sum_{n=1}^{N_S}\phi_n=1+\frac{3}{4}\times\frac{N}{N_S-N}. \qquad (7)$$

For $N_S<2N$, we can achieve a greater gain (than 12.5%) over the original ADAPTIVE algorithm. Since speedup and packet delay can trade one for another, this also means that packet delay can be smaller than that given in [8] if the same speedup is applied to the OPS switch.

From Fig. 11 in Appendix B, we know that $\boldsymbol{U}$ and $\boldsymbol{V}$ are used to record the row/column permutations involved, and they can be constructed from a *square unit matrix* $\boldsymbol{E}$ (i.e. an $N\times N$ matrix with N 1*s* at its diagonal entries and all other entries are 0*s*). Therefore, it is not necessary to get $\boldsymbol{U}^{-1}$ and $\boldsymbol{V}^{-1}$ by algebraic calculations. Instead, we can also start from a square unit matrix $\boldsymbol{E}$, and permute its lines *in a reverse order* to generate $\boldsymbol{U}^{1}$ and $\boldsymbol{V}^{-1}$.

Finally, it is important to note that our proposed approach is based on the DHH conjecture. We have tried to prove it for a very long time but without luck. We also cannot find a single counterexample by checking extensive samples using computer programs. Many mathematicians including the authors of [14] have reviewed our conjecture. As for now, the problem is still open.

V. Conclusion

Due to the reconfiguration overhead, speedup and packet delay are two main issues for traffic scheduling in optical packet switches (OPS). In this paper, we proposed a new approach to improve the scheduling efficiency in OPS with guaranteed switching performance. Compared with the existing scheduling algorithms, our approach can significantly reduce speedup and/or packet delay. However, the proposed approach is based on the DHH conjecture given in this paper. We call for a proof or disproof for this conjecture.

Appendix A Correctness Proof of Lemma 1

Lemma 1: In DOUBLE, if a particular line (row i or column j) of $\boldsymbol{R}$ contains k LERs, then in $\boldsymbol{Q}$ we have

$$\sum_{j=1}^{N} q_{ij} \le N-\left\lceil \frac{k}{2} \right\rceil \text{ for row } i, \textit{ or } \sum_{i=1}^{N} q_{ij} \le N-\left\lceil \frac{k}{2} \right\rceil \text{ for column } j.$$

Proof: After $\boldsymbol{C(T)}$ is divided by T/N, we have

$$\boldsymbol{C(T)}=\frac{T}{N}\boldsymbol{Q}+\boldsymbol{R} \quad \textit{or} \quad c_{ij}=\frac{T}{N}q_{ij}+r_{ij}, \qquad (8)$$

Without loss of generality, we assume that row i of $\boldsymbol{R}$ contains k LERs. Because

$$\sum_{j=1}^{N} c_{ij}=\frac{T}{N}\sum_{j=1}^{N} q_{ij}+\sum_{j=1}^{N} r_{ij} \le T, \qquad (9)$$

we have

$$\sum_{j=1}^{N} q_{ij} \le \frac{T-\sum_{j=1}^{N} r_{ij}}{\frac{T}{N}} < \frac{T-\frac{T}{2N}\times k}{\frac{T}{N}} = N-\frac{k}{2}. \qquad (10)$$

Since $\sum_{j=1}^{N} q_{ij}$ is an integer, we then have

$$\sum_{j=1}^{N} q_{ij} \le N-\left\lceil \frac{k}{2} \right\rceil. \qquad (11)$$

Appendix B DHH Conjecture

In this appendix, we assume that the size N of the matrices/ vectors is an even number.

Definition 1 (halve): Given an arbitrary 0/1 vector (row or column whose entries are either 0 or 1), use a line to separate it into two equal parts as shown in Fig. 8. Let x and y denote the number of 1*s* in each part. If $|x-y|\le 1$, we say that the 1*s* in the vector are halved by the line.

Fig. 8 gives some examples, where the 1*s* are halved in (a) and (c), but not in (b) and (d).

Definition 2 (DHH matrix): Given an arbitrary $N\times N$ 0/1 matrix, use two lines to partition it into four $(N/2)\times(N/2)$ zones/ sub-matrices $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ and $\boldsymbol{D}$ as in Fig. 9. For *each* row/column of the matrix, if the number of 1*s* in zone $\boldsymbol{A}$ or $\boldsymbol{C}$ is more than that in zone $\boldsymbol{B}$ or $\boldsymbol{D}$, or the 1*s* in this row/column are halved by one of the two lines, then this matrix is called a DHH matrix (it means that the diagonal half-size sub-matrices $\boldsymbol{A}$ and $\boldsymbol{C}$ contain at least "half" number of 1*s for each row and column*).

The matrix in Fig. 10a is a DHH matrix, whereas the matrix in Fig. 10b is not, because its second column has two more 1*s* in zone $\boldsymbol{D}$ than that in zone $\boldsymbol{A}$.

DHH conjecture: Given an *arbitrary* $N\times N$ 0/1 matrix $\boldsymbol{\Omega}$, we

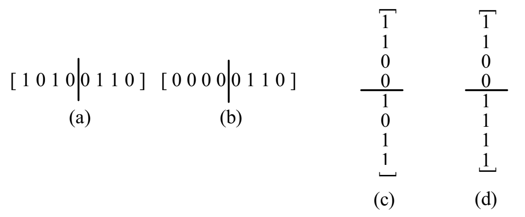


Fig. 8. Illustration of definition "halve".

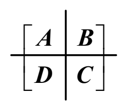


Fig. 9. Sub-matrices.

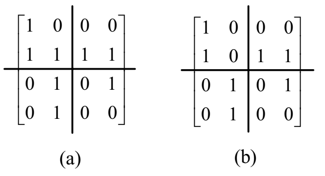
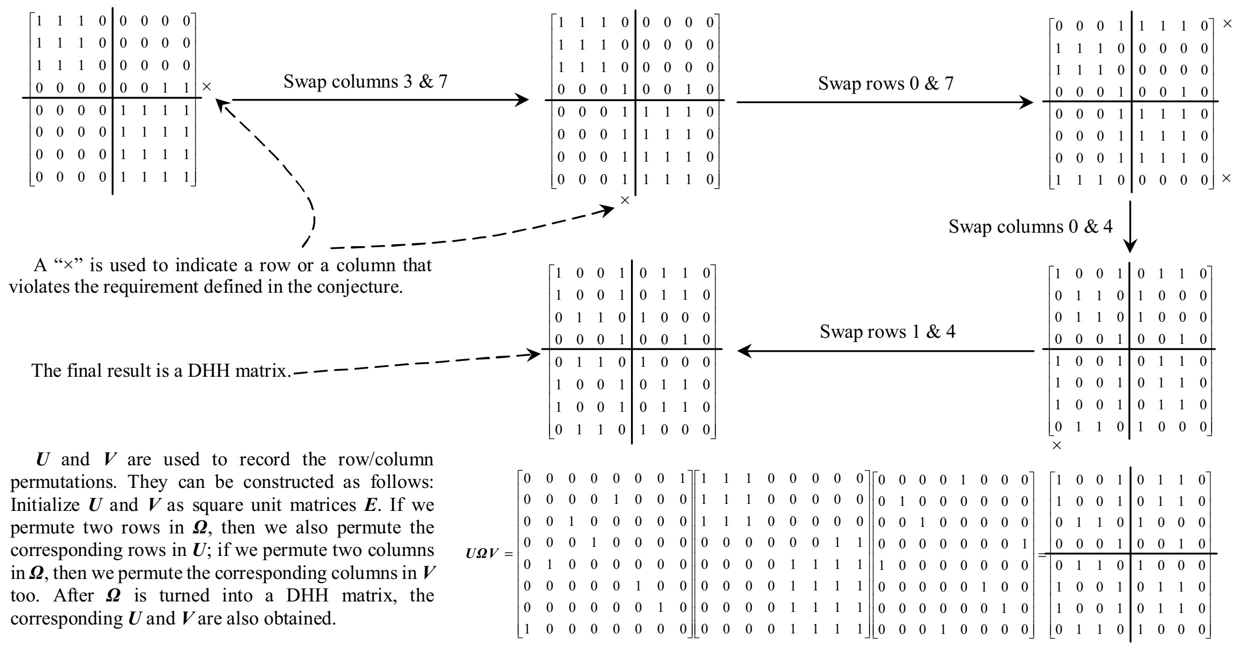


Fig. 10. (a) is a DHH matrix, and (b) is not.



Swap columns 3 & 7
Swap rows 0 & 7
Swap columns 0 & 4
Swap rows 1 & 4
A "×" is used to indicate a row or a column that violates the requirement defined in the conjecture.
The final result is a DHH matrix.
U and V are used to record the row/column permutations. They can be constructed as follows: Initialize U and V as square unit matrices E. If we permute two rows in Ω, then we also permute the corresponding rows in U; if we permute two columns in Ω, then we permute the corresponding columns in V too. After Ω is turned into a DHH matrix, the corresponding U and V are also obtained.
UΩV =

Fig. 11. Permute a matrix $\boldsymbol{\Omega}$ into a DHH matrix (the rows/columns are numbered from 0 to 7).

can permute its rows or columns[2] for a limited number of times, such that $\boldsymbol{\Omega}$ can be turned into a DHH matrix. In other words, there exist two permutation matrices $\boldsymbol{U}$ and $\boldsymbol{V}$, such that $\boldsymbol{U\Omega V}$ is a DHH matrix.

For example, if we swap the first row and the last row in Fig. 10b, then the resulting matrix is a DHH matrix. Fig. 11 gives a more complex example for $\boldsymbol{\Omega}$ in Fig. 7.

References

[1] 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.

[2] L. Y. Lin, "Micromachined free-space matrix switches with submilli-second switching time for large-scale optical crossconnect", *OFC'98 Tech. Digest*, pp. 147-148, Feb. 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] A. J. Agranat, "Electroholographic wavelength selective crossconnect", *1999 Digest of the LEOS Summer Topical Meetings*, pp. 61-62, Jul. 1999.

[5] K. Kar, D. Stiliadis, T. V. Lakshman, and L. Tassiulas, "Scheduling algorithms for optical packet fabrics", *IEEE Journal on Selected Areas in Communications*, vol. 21, issue 7, pp. 1143-1155, Sept. 2003.

[6] 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.

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

[9] Bin Wu and Kwan L. Yeung, "Scheduling optical packet switches with minimum number of configurations", *IEEE ICC '05*, vol. 3, pp. 1830-1835, May 2005.

[10] Bin Wu and Kwan L. Yeung, "Traffic scheduling in non-blocking optical packet switches with minimum delay", *IEEE GLOBECOM '05*, vol. 4, pp. 2041-2045, Dec. 2005.

[11] Bin Wu, Kwan L. Yeung and V. O. K. Li, "Two-layer parallel switching: A practical and survivable design for performance guaranteed optical packet switches", *IEEE GLOBECOM '05*, vol. 4, pp. 1905-1909, Dec. 2005.

[12] Bin Wu and Kwan L. Yeung, "On optimization of optical packet switches with reconfiguration overhead", *IEEE HPSR '05*, pp. 217-221, May 2005.

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

[14] R. A. Brualdi and H. J. Ryser, "Combinatorial matrix theory", Cambridge University Press, 1991.

[2] i.e., swap its rows or columns. Note that a row can only be swapped with another row, and a column can only be swapped with another column.