<table>
<thead>
<tr>
<th><strong>Title</strong></th>
<th>Can we schedule traffic more efficiently in optical packet switches?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Author(s)</strong></td>
<td>Wu, B; Wang, X; Yeung, KL</td>
</tr>
<tr>
<td><strong>Citation</strong></td>
<td>2006 Workshop On High Performance Switching And Routing, Hpsr 2006, 2006, p. 181-186</td>
</tr>
<tr>
<td><strong>Issued Date</strong></td>
<td>2006</td>
</tr>
<tr>
<td><strong>URL</strong></td>
<td><a href="http://hdl.handle.net/10722/45951">http://hdl.handle.net/10722/45951</a></td>
</tr>
<tr>
<td><strong>Rights</strong></td>
<td>This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.; ©2006 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.</td>
</tr>
</tbody>
</table>
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.

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], \( \delta \)-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 \( N_S = 2 \) configurations to schedule any legitimate traffic matrix, with a speedup of \( S_{\text{schedule}} = 2 \) (detailed in Section II). However, this algorithm does not consider the amount of reconfiguration overhead in its scheduling decision, and thus it is not optimized for switches with different reconfiguration overhead \( \delta \). 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
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 time T slots to construct an $N \times N$ traffic matrix $C(T) = [c_{ij}]$. Each entry $c_{ij}$ denotes the number of packets received at input $i$ and destined to output $j$. $C(T)$ 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 $C(T)$. The scheduling algorithm takes $H$ time slots in Stage 2 to generate $N_S$ configurations $P_{ns} = [p^{(n)}_{ij}]$, $n \in \{1, ..., N_S\}$, each weighted by $\phi_n$ to cover $C(T)$. “Cover” means that $\sum_{n=1}^{N_S} \phi_n p^{(n)}_{ij} \geq c_{ij}$ for any $i, j \in \{1, ..., N\}$. $P_n$ 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 $P_n$ for $\phi_n$ 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 $\delta N_S$ slots are used to reconfigure the switch for $N_S$ times in Stage 3, only $T-\delta N_S$ slots are left for transmitting $C(T)$. Since there are at most $T$ packets waiting at each input port for transmission, a speedup factor $S_{reconfig} = T/(T-\delta N_S)$ 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 $C(T)$. Therefore another speedup factor $S_{schedule} = 1/T \sum_{n=1}^{N_S} \phi_n$ 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

$$S = S_{reconfig} \times S_{schedule} = \frac{T}{T-\delta N_S} S_{schedule} = \frac{1}{\frac{T}{T-\delta N_S} \sum_{n=1}^{N_S} \phi_n}.$$  \hspace{1cm} (2)

B. Speedup-Delay Tradeoff and Scheduling Algorithms

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

$$C(T) = \frac{T}{N_S-N} \times Q + R.$$  \hspace{1cm} (3)

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

$$S_{schedule} = \frac{1}{\frac{T}{T-\delta N_S} \sum_{n=1}^{N_S} \phi_n} = \frac{T}{T-\delta N_S} \times \frac{\sum_{n=1}^{N_S} \phi_n}{N_S-N} = 1+\frac{N}{N_S-N}.$$  \hspace{1cm} (4)

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

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

$$\frac{\partial S}{\partial N_S} = 0.$$  \hspace{1cm} (5)

Therefore the schedule generated by **ADAPTIVE** is optimized with respect to the value of $N_S$.

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].
size $N$ is an even number.

A. Observation and Motivation

In DOUBLE, the traffic matrix $C(T)$ is decomposed as $C(T) = |T/N| \cdot Q + R$. For any $r_j \in R$, if $r_j > T/(2N)$, we call it an LER (large entry in $R$). Otherwise it is an SER (small entry in $R$). We have the following Lemma 1 (proved in Appendix A).

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

$$\sum_{j \in \mathcal{J}_i} q_{ij} \leq N - \left[ \frac{k}{2} \right]$$

for row $i$, or

$$\sum_{i \in \mathcal{I}_j} q_{ij} \leq N - \left[ \frac{k}{2} \right]$$

for column $j$.

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

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

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 $R$. From Lemma 1, “half” of these LERs in each line of $R$ can be moved to $Q$, while keeping the maximum line sum of $Q'$ not more than $N$. Therefore, it is reasonable to expect that each line of $R'$ 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 $\phi_w=|T/N|$, to cover all the remaining LERs in $R'$. At the same time, we may find another set of $N/2$ non-overlapping configurations, each with a reduced weight of $\phi_w=|T/(2N)|$, to cover the remaining SERs. If this can be done, then $S_{\text{schedule}}$ of DOUBLE can be reduced to

$$S_{\text{schedule}} \leq \frac{1}{T} \sum_{j=1}^{N} \sum_{i=1}^{T} q_{ij} = \frac{T}{N} \cdot N + \frac{T}{N} \cdot \frac{N}{2} + \frac{T}{N} \cdot \frac{N}{2} = 1.75$$

(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 $Q$, such that each line sum of $Q'$ does not exceed $N$, and $R'$ contains at most $N/2$ LERs in each line.

- Among the $N$ non-overlapping configurations used to cover $R'$, $N/2$ of them should cover all the remaining LERs in $R'$, 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 $Q$. This can be seen from the example in Fig. 5. Assume all the non-zero entries (3s) in $R$ are LERs. The number next to each line is the number of LERs that can be moved from this line to $Q$, which is obtained from $T/(2N)$ 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 ($\text{Predefined Configurations}$) 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. $PC_1 \sim PC_8$), as defined in Fig. 6, can be used to cover all the entries of an $8 \times 8$ matrix. Note that the number at each entry of this matrix denotes the particular $PC$ that covers this entry of $Q$. A carefully designed mechanism is necessary in order to move LERs properly.
entry. For easy reading, the entries covered by $PC_3$ 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/submatrices $A$, $B$, $C$, and $D$ as shown in Fig. 9 (and Fig. 10) in Appendix B. For each row/column, if the number of 1s in zone $A$ or $C$ is no less than that in zone $B$ or $D$, or is less by at most one, then this 0/1 matrix is called a DHH matrix. In other words, the two diagonal zones $(A$ and $C)$ of a DHH matrix contain at least “half” number of 1s 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 $PC$ and DHH matrix. We first convert the residue matrix $R = \{r_{ij}\}$ into a 0/1 indicator matrix $Q = \{q_{ij}\}$ such that $q_{ij} = 1$ if $r_{ij}$ is an LER and $q_{ij} = 0$ otherwise. The DHH conjecture given in Appendix B says that we can always find two permutation matrices $U$ and $V$ such that $\Omega = UQV$ is a DHH matrix. For the 8×8 matrix, assume that a DHH matrix $\Omega'$ is obtained by $\Omega = UQV$. Then, $PC_3 \sim PC_4$ defined in Fig. 6 (which span over the sub-matrices $A$ and $C$ as illustrated in Fig. 9) can cover “more than half” of the 1s for each row and column of $\Omega'$. The remaining 1s covered by $PC_3 \sim PC_4$ “correspond” to LERs that should be moved to $Q$. Since $\Omega'$ is obtained from $\Omega$ after some row/column permutations (i.e. $\Omega = UQV$), the 1s in $\Omega'$ 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 $Q$ from $R$. In Step 2, we find two
permutations $U$ and $V$ to permute $\Omega$ into a DHH matrix $\Omega'$. Then, Step 3 imposes $U$ and $V$ on the predefined configurations $PC_1 \sim PC_5$ and uses $A_i = U^{-1} (PC_i) V^{-1}$ (note $U^{-1} = U$ and $V^{-1} = V$ in this particular example) to generate eight configurations $A_1 \sim A_8$. For $A_1 \sim A_4$, if they cover some LERs in the original $R$, then these LERs (circled entries in Step 4) are to be moved to $\Omega$. Based on Lemma 1, we can set these LERs to 0s in $R$, and increase the corresponding entries in $\Omega$ by one. $\Omega'$ and $R'$ are thus obtained in Step 4. In Step 5, to cover $\Omega'$ we simply use the same edge-coloring algorithm [13] as in DOUBLE to determine configurations $P_1 \sim P_6$, each weighted by $T/N=4$. On the other hand, $R'$ can be covered by $A_5 \sim A_8$, with a weight of 4 for $A_5 \sim A_6$, and a reduced weight of $T/(2N)=2$ for $A_7 \sim A_8$. As a result, $S_{\text{schedule}}$ can be reduced to 1.75, as shown in Step 6. In fact, this 12.5% reduction in $S_{\text{schedule}}$ is independent of switch size $N$, as stipulated by (6). Note that there is a one-to-one mapping from entry to entry between $\Omega$ and $\Omega'$, which is determined by the linear transform $\Omega = UDV$. Therefore, the resulting $A_1 \sim A_8$ are non-overlapping, and they can cover every entry of $R'$.

IV. DISCUSSION

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

\[ S_{\text{schedule}}\leftarrow \frac{1}{T} \sum_{j=1}^{N} q_j = 1 + \frac{3}{4} \frac{N}{Ns/N}. \]  

(7)

For $Ns/N < 2$, 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 $U$ and $V$ are used to record the row/column permutations involved, and they can be constructed from a square unit matrix $E$ (i.e. an $N\times N$ matrix with $N$ 1s at its diagonal entries and all other entries are 0s). Therefore, it is not necessary to get $U^{-1}$ and $V^{-1}$ by algebraic calculations. Instead, we can also start from a square unit matrix $E$, and permute its lines in a reverse order to generate $U^{-1}$ and $V^{-1}$.

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 far as 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 $R$ contains $k$ LERs, then in $\Omega$ we have

\[ \sum_{j=1}^{N} q_j \leq N - \left\lfloor \frac{k}{2} \right\rfloor \text{ for row } i, \text{ or } \sum_{i=1}^{N} q_i \leq N - \left\lceil \frac{k}{2} \right\rceil \text{ for column } j. \]

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

\[ C(T) = \frac{T}{N} Q + R \quad \text{or} \quad c_{ij} = \frac{T}{N} q_{ij} + r_{ij}, \]  

(8)

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

\[ \sum_{j=1}^{N} c_{ij} = \frac{T}{N} \sum_{j=1}^{N} q_{ij} + \sum_{j=1}^{N} r_{ij} \leq T, \]  

(9)

we have

\[ \sum_{j=1}^{N} q_{ij} \leq \frac{T - \sum_{j=1}^{N} r_{ij}}{T/N} < \frac{T - T/2N}{T/N} = N - \frac{k}{2}. \]

(10)

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

\[ \sum_{j=1}^{N} q_{ij} \leq \left\lfloor \frac{k}{2} \right\rfloor. \]  

(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 1s in each part. If $|x-y|\leq 1$, we say that the 1s in the vector are halved by the line.

Fig. 8 gives some examples, where the 1s 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 $A$, $B$, $C$ and $D$ as in Fig. 9. For each row/column of the matrix, if the number of 1s in zone $A$ or $C$ is more than that in zone $B$ or $D$, or the 1s 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 $A$ and $C$ contain at least “half” number of 1s 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 1s in zone $D$ than that in zone $A$.

DHH conjecture: Given an arbitrary $N\times N$ 0/1 matrix $\Omega$, we
Fig. 8. Illustration of definition "halve".

\[
\begin{bmatrix}
1 & 0 & 1 & 0 \\
1 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 \\
0 & 0 & 1 & 1
\end{bmatrix}
\]

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.

\[
\begin{bmatrix}
1 & 0 & 0 & 1 \\
1 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 \\
0 & 1 & 0 & 1
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 0 & 0 & 1 \\
1 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 \\
0 & 1 & 0 & 1
\end{bmatrix}
\]

Initialize \( U \) and \( V \) as square unit matrices, then we also permute the second switching time for large-scale optical crossconnect, \( IEEE/LEOS Intl. Conf. on Optical MEMS, pp. 41-42, Aug. 2000. \)

REFERENCES


