Fast Scheduling for Optical Packet Switches with Minimum Configurations

Zhen Zhou, Xin Li and Mounir Hamdi
Department of Computer Science
Hong Kong University of Science and Technology
Clear Water Bay, Hong Kong
Email: {cszz, lixin, hamdi}@cs.ust.hk

Abstract—**Scheduling optical packet switches with minimum configurations has been proven to be NP-complete. Moreover, an optimal scheduling algorithm will produce as many as $\Theta(\log N)$ empty slots in each switching per time slot for an $N \times N$ optical switch. The best known algorithm approximates the optimal solution in $O(N^{3.5})$ time. In this paper, we propose an algorithm called DNC with minimal time complexity $\Theta(N^2)$ based on divide-and-conquer paradigm. The number of empty slots created by our algorithm is upper bounded by $\Theta(N)$. However, simulation results indicate that it is approximately $O(\log N)$ on average. Therefore, our algorithm is more practical and beats the previous ones when optical switches have large reconfiguration overhead.**

I. Introduction

Optical fabrics based on micro-electromechanical system (MEMS) mirror [1], thermal bubble [2], waveguide [3], and similar technologies [4] have been developed to meet the explosion of the internet traffic demands. They further provide potential benefits including scalability, high bit rate, and low power consumption on economical bases. However, reconfiguring the fabric connections for optical switches are more time-consuming than their electronic counterparts, due to mechanical settling and synchronization.

A common approach to diminish the effect of large reconfiguration delays is to periodically accumulates incoming traffic before constructing a schedule that delivers the packets to the output. Certain schedule consists of a number of switching patterns. Each holds for a period of time slots. Because the duration of both accumulating and transmitting packets shall be the same for stability reasons, transmission speedup is required in order to compensate not only fabric reconfiguration delays but also inefficient use of time slots caused by scheduling algorithms. Nevertheless, optimizing scheduling cost for the optical switch scheduling (OSS) problem has been proven to be NP-complete [5]. This suggests looking for fast approximation algorithms that come close to the optimum solution in polynomial time.

As the reconfiguration delay is very large compared to the slotted time, it is always desirable to minimize the number of switchings. There are several algorithms proposed. For example, K-TRANSPONDERS [6] uses the least number of switchings but is impractical due to its large time complexity. Towles and Dally invented a better algorithm MIN in [7] with guaranteed performance bound. However, the time complexity of MIN is still considered to be quite large. This is because traditional methods employ bipartite matching algorithms or edge coloring algorithms as subroutines. Therefore, in order to accelerate scheduling algorithms for OSS problem, one has to look into some other paradigms.

This work was supported in part by a grant from Hong Kong Research Grant Council (Grant Number: RGC HKUST6160/03E).

In this paper, we devise a novel algorithm (DNC) based on simple heuristic, namely divide-and-conquer. It turns out to be a fast algorithm at the cost of large asymptotic worst case bound on wastage. Our algorithm works in time proportional to the size of the switch, which is the minimal amount one can get. On the other hand, although there are more wasted time slots produced by DNC in the worst case, our simulation results provide the evidence that the expected number is kept at low level.

In the remaining of the paper, we will first introduce all technical aspects of OSS problem and its related researches in Section II. The description of our scheduling algorithm is given in Section III, followed by its performance analysis in Section IV. The simulation data and discussion is presented in Section V. Finally, we draw our conclusion in Section VI.

II. Preliminaries

In an $N \times N$ optical switch, the incoming traffic rates is denoted as an $N \times N$ matrix $\Lambda = [\lambda_{i,j}]_{N\times N}$. The traffic is said to be *admissible* if and only if no input port or output port is oversubscribed. In other words, $\sum_i \lambda_{i,j} \leq 1$ and $\sum_j \lambda_{i,j} \leq 1$. Moreover, there is at most one packet received from any input port or dispatched to any output port in one time slot.

An optical switch works in a three-phase cycle as shown in Figure 1. The three phases in turn are accumulating packets, scheduling, and transmitting packets. Let T be a pre-defined system parameter. After the accumulating phase of T time slots, we obtain a *demand matrix* $D = [d_{i,j}]_{N\times N}$ in which any row sum and column sum should not exceed the accumulated port capacity under admissible traffic. That is,

$$\sum_i d_{i,j} \leq T \text{ and } \sum_j d_{i,j} \leq T, \quad (T > N). \qquad (1)$$

We will refer Equation (1) as *admissible traffic condition* in our later discussion.

The switch does the scheduling task in the second phase and switches the packets to output lines in the third phase. In such a manner, pipelining is allowed and we are guaranteed to have the worst case switching delay bounded by the sum of the time spent in the three phases. As shown in Figure 1, we assume the duration of all three phases is T for the ease of analysis although they could be different. However, it is critical that the third phase never takes longer than the batch time T because otherwise the optical switch is not stable. In this model, the switching delay is at most $3T$, because packets that arrive in the first phase are guaranteed to departure after the third phase.

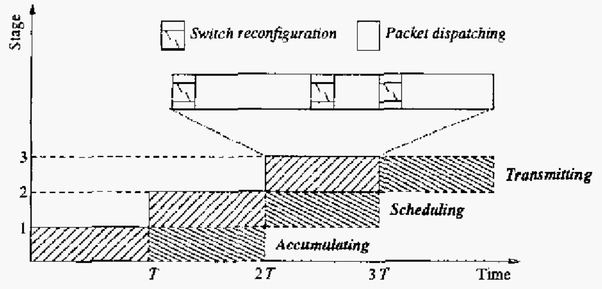
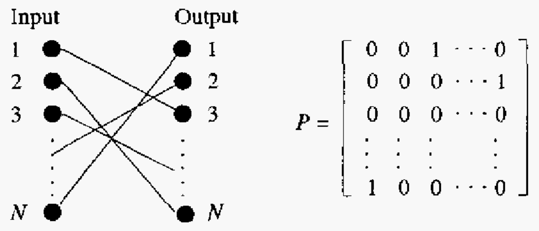


Fig. 1. Three phases of optical switching and pipelining.

Optical switches set up a bipartite matching (shown in Figure 2) between inputs and outputs in order to dispatch packets in one time slot. This elementary unicast operation can be modelled as generating a *(partial) permutation matrix* (i.e. *switching*) $P = [p_{i,j}]_{N \times N}$ which is a 0-1 matrix with at most one "1" on each row or column. An "1" on ith row jth column indicates that input i shall connect with output j in the current switching. In general, we want a scheduling algorithm that produces N_s switchings $P^{(k)}$ $(1 \leq k \leq N_s)$, each lasts for ϕ_k time slots, that *covers* the demand matrix. This constraint can be written as $\sum_k \phi_k p_{i,j}^{(k)} \geq d_{i,j}$, where $p_{i,j}^{(k)}$ is an entry in matrix $P^{(k)}$.



$$P = \begin{bmatrix} 0 & 0 & 1 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

Fig. 2. Example of bipartite matching (left) and permutation matrix (right).

Each time the optical switch starts up a new connection, it introduces a *reconfiguration overhead* δ. Assuming that T is larger than $T_{min} = N_s\delta$, in order to transmit all packets in T time slots, speedup is required both to cover the reconfiguration overhead and to compensate for empty slots left by the scheduling algorithm. Since T_{min} time slots for reconfiguring the optical switch is substantial, only $T - T_{min}$ time slots are available for actually transmitting packets. Therefore, the speedup required by the schedule is

$$S = \frac{\sum_{k=1}^{N_s} \phi_k}{T - T_{min}}. \tag{2}$$

Given that δ is an unavoidable and dominating factor, minimum number of switchings is desirable in most cases. Let r_i and c_j denote the number of nonzero entries on row i and column j respectively. Gopal and Wong have shown that $N_s = \max\{\max_i\{r_i\}, \max\{c_j\}\}$ is necessarily the minimum number and indeed achievable in $O(N^4)$ time [6]. Consequently, $N_s = N$ is sufficient and then the denominator in (2) is fixed. Our objective becomes

$$\min T_s = \sum_{k=1}^{N_s} \phi_k. \tag{3}$$

A side-effect comes together with minimizing the number of switchings. We are expected to see a lot of empty slots. By constructing an adversary input, Towles and Dally has proven the following lemma in [7].

Lemma 1: Any scheduling algorithm attempting to use only N switchings will cause at least $\Theta(T \log N)$ empty slots in each switching.

Beside that, They also shown that their algorithm MIN achieved this lower bound in $O(N^{3.5})$ time.

The above mentioned approaches use exactly N switchings, and are in fact *non-preemptive scheduling* algorithms. That is, all the packets from input i to output j must be delivered during one switching configuration. Whereas in *preemptive scheduling*, they can be split and covered by several permutation matrices if necessary. However, it will generally introduce more than N switchings, up to a number of $(N^2 - 2N + 2)$ [9]. Preemptive scheduling on optical switches is inherently difficult: even approximating it within a factor of 7/6 is NP-hard [8]. Some of the preemptive scheduling algorithms can be found in [5], [7]-[10].

III. Algorithm DNC

Before we formally introduce our algorithm, let us first present the idea with some notations defined. First, let's assume $N = 2^n$, where n is an integer. Note that this is a common situation in actual hardware implementation. Consider any $2^i \times 2^i$ demand matrix, denoted as D_i $(i \geq 1)$. We can divide it into four parts, each of size $2^{i-1} \times 2^{i-1}$, as shown in Figure 3. In counter-clockwise order, we denote

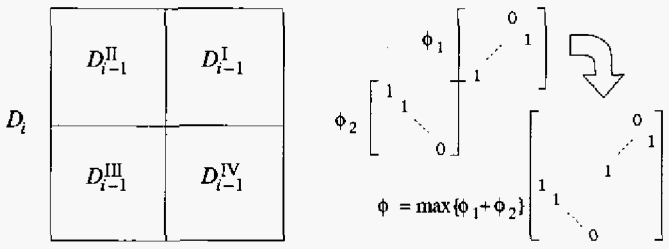


Fig. 3. Divide a $2^i \times 2^i$ matrix into 4 parts (left) and an example of combining two switchings in D_{i-1}^{I} and D_{i-1}^{III} (right).

them D_{i-1}^{I}, D_{i-1}^{II}, D_{i-1}^{III}, and D_{i-1}^{IV}. These sub-matrices can be regarded as subproblems and be further divided until the size reaches 1×1.

In a bottom-up manner, we are able to obtain the switchings of all subproblems. That is, for each subproblem, we obtain 2^{i-1} switching configurations associated with weights $\phi_1, \dots, \phi_{2^{i-1}}$, respectively. Observe that the solutions of any two sub-matrices along the diagonal can be combined without any column or row confliction. This is because the union of two disjoint permutation matrices is still a permutation matrix. Therefore, the 2^i switchings of D^{I}_{i-1} and D^{III}_{i-1} can be packed into 2^{i-1} double-sized switchings. One instance of such combine operations is shown in Figure 3. We let the larger weight of the two combined switchings become the new weight of its union. The intuition is that packing two switchings with roughly same weights will result in little wastage. Therefore, we first sort the switchings according to their weights for both D^{I}_{i-1} and D^{III}_{i-1}. Then we process the combination in the sorted order. After that we obtain 2^{i-1} switchings for D_i. Symmetrically, the other 2^{i-1} switchings and weights can be obtained by combining D^{II}_{i-1} and D^{IV}_{i-1}. As a result, we construct the 2^i switchings and weights for D_i, which are then available to form the schedules for D_{i+1}.

Next, we provide the pseudo-code of our algorithm named DIVIDE-AND-CONQUER (DNC) below.

```
Algorithm DNC($D_n$)
Input:
    $N \times N$ non-negative integer matrix $D_n$, where $N = 2^n$.
Output:
    A set of configuration matrices $P^{(1)}, \dots, P^{(N)}$ and corresponding non-negative integer weights $\phi_1, \dots, \phi_N$.
Procedure:
    If $n = 0$, assign the single element to form $P^{(1)}$ and its value to be $\phi_1$, and then return.
    Otherwise,
    1) Divide:
        Partition $D_n$ into four disjoint $2^{n-1} \times 2^{n-1}$ sub-matrices of following:
            $D^{\text{I}}_{n-1} = \{d_{rc}\}, \quad 1 \le r \le \frac{N}{2}$ and $\frac{N}{2}+1 \le c \le N;$
            $D^{\text{II}}_{n-1} = \{d_{rc}\}, \quad 1 \le r, c \le \frac{N}{2};$
            $D^{\text{III}}_{n-1} = \{d_{rc}\}, \quad \frac{N}{2}+1 \le r \le N$ and $1 \le c \le \frac{N}{2};$
            $D^{\text{IV}}_{n-1} = \{d_{rc}\}, \quad \frac{N}{2}+1 \le r, c \le N.$
    2) Conquer:
        Recursively call DNC($D^q_{n-1}$), $q \in \{\text{I}, \text{II}, \text{III}, \text{IV}\}$.
    3) Combine:
        a) Sort the weights in non-increasing order for each sub-matrix, such that $\phi_k(D^q_{n-1})$ is the $k$th largest weight in $D^q_{n-1}$, where $q \in \{\text{I}, \text{II}, \text{III}, \text{IV}\}$.
        b) Merge the results from the recursive call according to the sorted order and construct the weights and switchings, i.e.,
            $\phi_k \leftarrow \max\{\phi_k(D^{\text{I}}_{n-1}), \phi_k(D^{\text{III}}_{n-1})\}$
            $\phi_{\frac{N}{2}+k} \leftarrow \max\{\phi_k(D^{\text{II}}_{n-1}), \phi_k(D^{\text{IV}}_{n-1})\}$
            $P^{(k)} \leftarrow P^{(k)}(D^{\text{I}}_{n-1}) \cup P^{(k)}(D^{\text{III}}_{n-1})$
            $P^{(\frac{N}{2}+k)} \leftarrow P^{(k)}(D^{\text{II}}_{n-1}) \cup P^{(k)}(D^{\text{IV}}_{n-1})$
    4) Output the result.
```

We demonstrate our algorithm on the following simple example. Consider the following 4×4 demand matrix

$$\begin{bmatrix} 22 & 10 & 5 & 17 \\ 16 & 27 & 16 & 10 \\ 21 & 28 & 8 & 5 \\ 20 & 0 & 21 & 0 \end{bmatrix} \Rightarrow \begin{matrix} \begin{bmatrix} 22 & 10 \\ 16 & 27 \end{bmatrix} & \begin{bmatrix} 5 & 17 \\ 16 & 10 \end{bmatrix} \\ \begin{bmatrix} 21 & 28 \\ 20 & 0 \end{bmatrix} & \begin{bmatrix} 8 & 5 \\ 21 & 0 \end{bmatrix} \end{matrix},$$

which has been partitioned into four 2×2 matrices. After further partitioning, all matrices are 1×1, and can be scheduled trivially. As an example, the upper-left matrix D^{II}_1 is decomposed as

$$\begin{bmatrix} 22 & 10 \\ 16 & 27 \end{bmatrix} \Rightarrow \begin{matrix} [22] & [10] \\ [16] & [27] \end{matrix}$$

The resulting four 1×1 matrices are immediately available for combining. It yields

$$P^{(1)}(D^{\text{II}}_1) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \text{ and } P^{(2)}(D^{\text{II}}_1) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},$$

with weights $\phi_1(D^{\text{II}}_1) = 16$, $\phi_2(D^{\text{II}}_1) = 27$. Similarly, we have

$$P^{(1)}(D^{\text{IV}}_1) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \text{ and } P^{(2)}(D^{\text{IV}}_1) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},$$

with weights $\phi_1(D^{\text{IV}}_1) = 21$, $\phi_2(D^{\text{IV}}_1) = 8$ for D^{IV}_1. Arriving at the *combine* step, we first sort the weights. After sorting, $P^{(1)}(D^{\text{II}}_1)$ and $P^{(2)}(D^{\text{II}}_1)$ swap their places, while $P^{(1)}(D^{\text{IV}}_1)$ and $P^{(2)}(D^{\text{IV}}_1)$ remain the same. Then merging the four switchings yields

$$P^{(3)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \text{ and } P^{(4)} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

with weights $\phi_3 = 27$, $\phi_4 = 16$. We can perform similar operations on the other diagonal, too.

The correctness proof of DNC involves two parts. First we shall prove that exactly N permutation matrices are used for scheduling $N \times N$ demand matrix. This can be shown inductively by the arguments presented in the second paragraph of this section. Secondly, we shall prove that the resulting N switchings cover the demand matrix. Recall that we choose the larger weight as the new weight after merging two sub-matrices. Therefore, it must be larger than all entries that are covered by the two sub-matrices. By induction, it is easy to see that the ultimate weight is not smaller than any entry that is covered by the corresponding switching. This condition holds for every switching DNC produced. In summary, DNC creates exactly N switchings that cover the demand matrix.

IV. Performance Analysis

The performance of our algorithm is analyzed in two aspects. We will first give a tight bound for its time complexity. Then we show the efficiency of our algorithm in terms of the number of empty slots it produces.

A. Time Complexity

Since DNC follows standard divide-and-conquer paradigm, we can write its time complexity $X(N)$ as

$$X(N) = \begin{cases} \Theta(1) & \text{if } N = 1, \\ 4X(\frac{N}{2}) + O(N \log N) & \text{otherwise.} \end{cases}$$

In the above recurrence relation, $4X(\frac{N}{2})$ comes from four recursive calls in the *conquer* step. The assignments of new weights and switchings are of $O(1)$ time each and are dominated by the sorting operation. So $O(N \log N)$ is the worst-case time complexity of the *combine* step. The time complexity of other operations is at most $O(N)$, thus omitted.

By Master Theorem [11], $X(N) = \Theta(N^2)$. This indicates that the overall time complexity of the algorithm is proportional to the number of elements in the demand matrix. One can hardly find an other algorithm that runs faster than DNC asymptotically.

B. Efficiency

Due to the difference of the weights of two sub-matrices, every *combine* step will cause a few empty slots. In this subsection, we will show an asymptotic bound on the number of empty slots for our algorithm.

Consider the scenario depicted in Figure 3. The total number of empty slots in D_i, denoted by $E(D_i)$, is coming from six sources. They are

- empty slots from four sub-matrices, i.e., $E(D^q_{i-1})$ where $q \in \{\text{I}, \text{II}, \text{III}, \text{IV}\}$;
- empty slots from the merging operations in the *combine* step, i.e.,

$$f_{i-1} = 2^{i-1} \sum_{1 \le k \le 2^{i-1}} \Big(\left|\phi_k(D^{\text{I}}_{i-1}) - \phi_k(D^{\text{III}}_{i-1})\right| + \left|\phi_k(D^{\text{II}}_{i-1}) - \phi_k(D^{\text{IV}}_{i-1})\right| \Big). \quad (4)$$

  We multiply every difference by 2^{i-1} because all 2^{i-1} elements in the switching with smaller weight contribute empty slots.

Sum them up, we have

$$E(D_i) = \sum_{q \in \{\text{I},\text{II},\text{III},\text{IV}\}} E(D^q_{i-1}) + f_{i-1}. \quad (5)$$

It is again a recurrence relation. Hence, we can derive $E(D_n)$ by Master Theorem.

However, it is not trivial to estimate f_i for every possible i. We are only able to show the asymptotic bound of $E(D_n)$, from another perspective. In the worst case, each switching configuration has associated weight T, but covers only one nonzero entry (which has to have value T) of the demand matrix D. In this case, it is easy to see that each switching wastes $O(NT)$ time slots in the worst case. Notice that this worst case bound is valid for any OSS algorithm that attempts to use minimum configurations.

It is possible to construct a malicious adversary matrix that forces DNC actually reach the worst case bound. We give such an input as follows. We construct the adversary matrix D_{adv} by setting all $d_{j,2j-1} = T$ for $1 \le j \le \frac{N+1}{2}$ and all others entries zero. Portion of such a matrix is shown below.

$$D_{adv} = \begin{bmatrix} T & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & T & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & T & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

The adversary matrix clearly fulfill the admissible traffic condition as its row sums and column sums are no larger than T. DNC will produce $N/2$ switchings for D_{adv}, thus create $N(NT-1)/2$ empty slots in total. While on the other hand, the optimal algorithm will produce only one switching with $NT/2$ empty slots. In this case, DNC produces $T_s = \Theta(NT)$.

The worst case bound on the number of the empty slots seems to be very large compared to the lower bound $\Theta(T \log N)$ mentioned in Lemma 1. However, the adversary input also hint that it is really unusual to have such kind of demand matrices. This suggests further exploration on the average number of empty slots produced by DNC.

V. Simulation and Discussion

The purpose of this simulation is to experiment the actual performance of DNC in terms of its efficiency. Details of our methodology and outputs will be discussed in this section.

In our first experiment, we fix the size of the switch to be $N = 2^{10}$. The input demand matrices have random entries. The values of T is also random and $T \gg 10N$. Some of matrices are sparse and the others are dense, reflecting the fact that network traffics are by nature various. After running DNC on 10^4 such random matrices, we normalize the resulting T_s's with respect to T. The statistical distribution of these T_s/T's is shown in Figure 4.

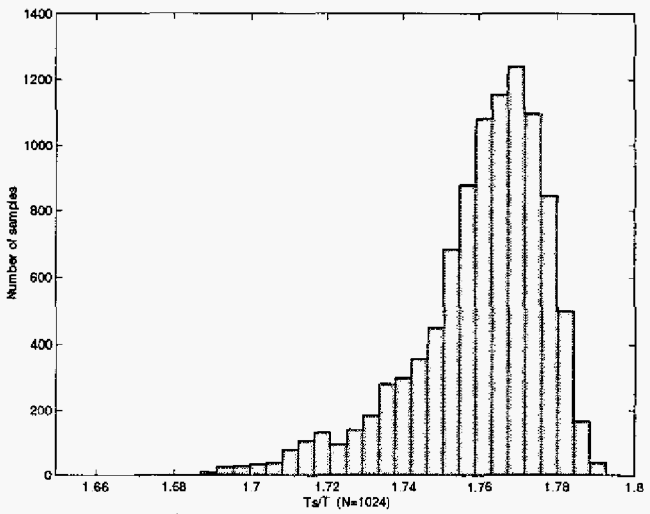


Fig. 4. Statistical distribution of T_s/T over 10^4 demand matrices by DNC on a $2^{10} \times 2^{10}$ switch.

The arithmetic mean of these 10^4 sample executions is about 1.7597, with standard deviation around 0.0179. From the figure, we also see that the majority of the outputs vary from 1.72 to 1.79, which form approximately a Gaussian distribution. This implies that DNC runs quite stable on a large set of inputs.

Our second experiment tests how T_s changes on various N values. The size of the switches ranges from 2^1 to 2^{12}. The input demand matrices are generated at random as before. We have seen from the previous experiment that arithmetic mean (MEAN) and standard deviation (STD) essentially capture the distribution of T_s/T. It is not necessary to present our result in full detail. Therefore for each N value, we collect the statistical data over 10^4 DNC executions, summarized in Table I. In the same table, we also compare the growth of T_s/T with that of $\log_2 N$.

TABLE I
COMPARISON BETWEEN T_s/T AND $\log_2 N$, ($N = 2^1, \ldots, 2^{12}$)

$n = \log_2 N$	MEAN	STD	$(T_s/T - 1) : \log_2 N$
1	1.0000	0	0
2	1.1065	0.0000	0.0533
3	1.2632	0.0000	0.0877
4	1.3973	0.0000	0.0993
5	1.4474	0.0019	0.0895
6	1.5117	0.0032	0.0853
7	1.6022	0.0115	0.0860
8	1.6725	0.0271	0.0842
9	1.7184	0.0121	0.0798
10	1.7595	0.0110	0.0760
11	1.7912	0.0171	0.0719
12	1.8162	0.0262	0.0680

When N becomes larger and larger, the accuracy of our statistical estimation shall drop because the sample space is not sufficiently large to cover all possibilities. Even though, it is clear that T_s/T grows more or less linearly with respect to $\log_2 N$ as implied from the last column of Table I. It is a supportive evidence to our conjecture that DNC produces on average $O(\log N)$ empty slots. It also confirms us with the intuition that DNC performs badly with very low probability, only on those artificial adversary inputs.

VI. Conclusion

Along with the fast development of Internet, optical switching technologies are becoming attractive for its huge capability and scalability. The disadvantage of optical switching comes from large reconfiguration overhead due to the technology constraints. Because of the NP-completeness of the OSS problem, few scheduling algorithms have been both fast and efficient. The DNC algorithm proposed in this paper provides an alternative that runs in minimal amount of time with bounded inefficiency in the worst case. It also guarantees that the number of configurations required is minimum, which also means minimum reconfiguration overhead.

By conducting simulations on large number of random inputs, we observe that the average wastage of time slots is about $O(\log N)$. It would be an interesting question for future study to show whether or not it is a coincidence. Moreover, since we applied a totally different paradigm for OSS problem other than bipartite matching or edge coloring, future researches could also look for better heuristics that solve the problem. Another open question is whether divide-and-conquer scheme could be improve the performance of preemptive scheduling algorithms that allow $N_s > N$.

Acknowledgment

The authors would like to thank Yajun Wang and Yan Zhang for their useful discussion and anonymous referees for their valuable comments.

References

[1] P. B. Chu, S. S. Lee and S. Park. MEMS: The Path to Large Optical Crossconnects *IEEE Communications Magazine*, vol. 40, Issue 3, pp. 80-87, Mar. 2002.

[2] J. E. Fouquet, S. Venkatesh, M. Troll, D. Chen, H. F. Wong, and P. W. Barth. A compact, scalable cross-connect switch using total internal reflection due to thermally-generated bubbles. In *Proceedings of Lasers and Electro-Optics Society Annual Meeting*, pp. 169 – 170, 1998.

[3] O. B. Spahn, C. Sullivan, J. Burkhart, C. Tigges, and E. Garcia. GaAs-based microelectromechanical waveguide switches. In *Proceeding of IEEE/LEOS International Conference on Optical MEMS*, pp. 41 – 42, 2000.

[4] X. H. Ma and G. H. Kuo. Optical switching technology comparison: optical MEMS vs. other technologies *IEEE Communications Magazine*, vol. 41, Issue 11, pp. S16-S23, Nov. 2003.

[5] X. Li and M. Hamdi. On scheduling optical packet switches with reconfiguration delay. *IEEE Journal on Selected Areas in Communications*, vol. 21, pp. 1156 – 1164, 2003.

[6] I. S. Gopal and C. K. Wong. Minimizing the number of switchings in an SS/TDMA system. *IEEE Transactions on Communications*, vol. 33, pp. 497 – 501, 1985.

[7] B. Towles and W. J. Dally. Guaranteed scheduling for switches with configuration overhead. *IEEE/ACM Transactions on Networking*, vol. 11, pp. 835 – 847, 2003.

[8] P. Crescenzi, X. Deng, and C. H. Papadimitriou. On approximating a scheduling problem. *Journal of Combinatorial Optimization*, vol. 5, pp. 287 – 297, 2001.

[9] T. Inukai. An efficient SS/TDMA time slot assignment algorithm. *IEEE Transactions on Communications*, vol. 27, pp. 1449 – 1455, 1979.

[10] L. K. Yeung. Efficient time slot assignment algorithms for tdm hierarchical and non-hierarchical switching systems. *IEEE Transactions on Communications*, vol. 49, pp. 351 – 359, 2001.

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Chapter 4, *Introduction to Algorithms*, 2nd Edition. MIT Press, 2001.