Non-Violation Set Scheduling for Two-Dimensional Optical MEMS Switches

Xin Li, *Member, IEEE,* Zhen Zhou, *Member, IEEE,* and Mounir Hamdi, *Senior Member, IEEE*

***Abstract*— Optical fabrics based on 2D MEMS suffer time-consuming reconfiguration delay. Traditional time slot assignment (TSA) and burst scheduling schemes do not solve the problem effectively. In this letter, we propose a novel non-violation set scheduling scheme which allows overlap traffic transmission of current schedule with the fabric setup for the next schedule. It makes the switch work as if no reconfiguration delay existed. A dynamic diagonal (DD) algorithm following this scheme shows superior performance. In addition, it runs at much lower time complexity and is readily implemented in hardware.**

***Index Terms*— Optical switch, MEMS, reconfiguration delay, scheduling.**

I. Introduction

OVER the past few years, there is increasing interest in using optical fabrics. Compared to its electronic counterparts, optical fabrics are cheaper and smaller in size. They provide more scalability, higher bit rate, and lower power consumption on economical bases. Current technologies for optical fabric include optical micro-electro-mechanical systems (MEMS), liquid crystal, bubble switches, thermo-optic, and so on [1].

Among them, silicon-based optical MEMS have turned out to be most attractive [2]. Take the popular two-dimensional (2D) MEMS for example. The basic switching elements are tiny mirrors with binary ON/OFF positions, which are arranged in a crossbar configuration (see Fig. 1). Switching is done by reflection of light. In general, if the (i, j) mirror is raised up (i.e., in the ON position), it directs light from the ith input fiber to the jth output fiber.

The time required for establishing a connection between inputs and outputs in 2D MEMS optical switches (or *reconfiguration delay*) takes hundreds of nanoseconds to a few milliseconds [1]. This includes mechanical settling (i.e., to raise up the mirrors), synchronization, and the like. The delay is around 10 to 10^5 time slots for a system with slotted time equals to 50 ns (64 bytes at 10 Gb/s). In this case, traditional slot-by-slot scheduling is no longer efficient. For example, even when each reconfiguration takes only one time slot, at least half of the bandwidth is wasted on setting up the fabric in between each transmission.

Obviously, the scheduling rate needs to be slowed down and each schedule holds for several slots. Time Slot Assignment (TSA) scheduling [3]–[6] is a common approach. However most TSA algorithms are quite unable to deal with the reconfiguration delay effectively; so they turn the research focus to alleviate the side effect of the delay to the system (e.g. they try to minimize the increased fabric speedup requirement). Other approaches, such as Burst Scheduling scheme [7], suffer from reconfiguration delay too.

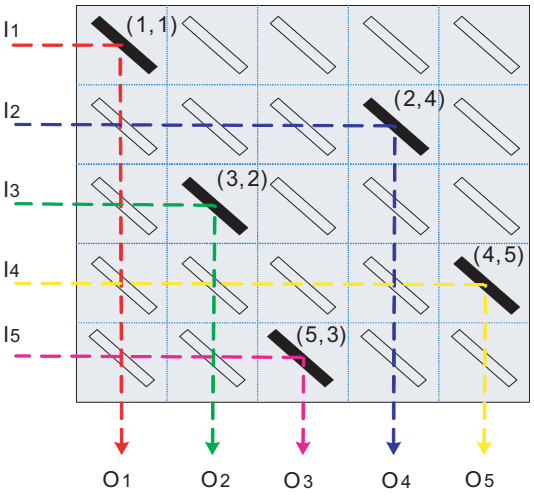

Fig. 1. A 5×5 MEMS optical switch with crossbar structure. The mirrors $(1, 1)$, $(2, 4)$, $(3, 2)$, $(4, 5)$ and $(5, 3)$ are in the ON state. Optical signals from inputs I_1, I_2, I_3, I_4, I_5 are switched to outputs O_1, O_4, O_2, O_5, O_3.

However, that does not mean we have to accept that scheduling schemes are helpless for reconfiguration delay. We observe that the special architecture of 2D MEMS allows us to overlap traffic transmission of the current schedule with the fabric setup for the next schedule, as long as the scheduling scheme guarantees no interference between them. By doing so, the fabric is setup concurrently with traffic transmission. Then the switch works as if no reconfiguration delay existed. In this letter, we introduce a novel non-violation set scheduling scheme which successfully hides the fabric reconfiguration time. A scheduling algorithm that follows this scheme, namely the Dynamic Diagonal (DD) algorithm, is proposed in Section IV.

II. Switch Model

The optical 2D MEMS fabric is modeled as a non-blocking crossbar which realizes any one-to-one mapping between inputs to outputs. For an $N \times N$ switch, such a mapping is described by a *switch configuration* (or *schedule*) $P_{N\times N}$; where P is a (0,1)-matrix with at most one nonzero element for each row and column (or *permutation matrix*). $p_{ij} = 1$

Manuscript received November 1, 2005. The associate editor coordinating the review of this letter and approving it for publication was Dr. Maode Ma. This work was supported in part by Hong Kong Research Grant Council (Grant Number: RGC HKUST6160/03E)

The authors are with the Department of Computer Science, the Hong Kong University of Science & Technology, Clear Water Bay, Hong Kong (e-mail: {lixin, cszz, hamdi}@cs.ust.hk).

Digital Object Identifier 10.1109/LCOMM.2006.04029.

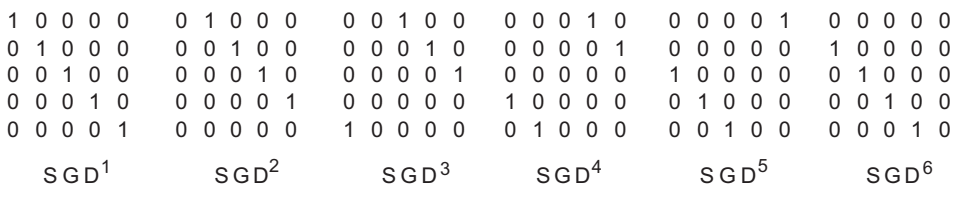

Fig. 2. A complete sequence of schedules of Static Generalized Diagonal (SGD) on a 5×5 switch.

means switch input i connects output j in schedule P. A fixed, nonzero reconfiguration delay δ is associated with the model. Time is slotted inside the switch and δ is in unit of time slots.

Packets are segmented into fixed size *cells* before entering the fabric. Traffic accumulates periodically every T slots. T is a predefined system constant. The accumulated traffic batch is stored in a *traffic matrix* $D_{N\times N}$. d_{ij} denotes the number of cells received in input i destined to output j during the accumulation stage. Traffic should be *admissible*; meaning that the line sum of D is no larger than T. That is, $\sum_{j=1}^{N} d_{ij} \leq T$ and $\sum_{i=1}^{N} d_{ij} \leq T$.

III. Non-Violation Set Scheduling Scheme

Although TSA scheduling schemes abandon attempts to deal with fabric reconfiguration time, we find a turn for the better in scheduling 2D MEMS switches. Take a 5×5 switch as shown in Fig. 1 for example. Current connections are between inputs I_1, I_2, I_3, I_4, I_5 and outputs O_1, O_4, O_2, O_5, O_3, respectively. The light paths of optical signals are shown in dotted lines. Obviously, all mirrors at the light paths should be strictly in the OFF position. However, for those not on the light paths (i.e. mirrors at $(1,2)$, $(2,5)$, $(3,3)$, and so on, denoted as *non-violation set of mirrors*), changing them from OFF to ON does not affect the current transmission. If the next schedule only uses the non-violation set of mirrors, the fabric may start to set up the schedule during the current transmission. Suppose each schedule is held for δ slots, then as soon as the current transmission ends, the next schedule has been set up and transmission can start right away. In this manner, the fabric works as if no reconfiguration delay existed.

A. Scheduling Scheme

The above observation is described more precisely in the following definition.

Definition 1 (Non-Violation Set): Given a schedule $P_{N\times N}$, its non-violation set $NV(P)_{N\times N}$ is an (0,1)-matrix such that

$$nv(P)_{ij} = \begin{cases} 1 & \text{if } \sum_{b=j+1}^{N} p_{ib} + \sum_{a=1}^{i-1} p_{aj} = 0 \\ 0 & \text{otherwise} \end{cases}$$

The non-violation set $NV(P)$ includes all mirrors that are not on any light path in schedule P. Notice that $P \subseteq NV(P)$, because ON mirrors for the current schedule can still be used in the next one without additional setup time.

If a scheduling scheme successfully produces two consecutive schedules P^k and P^{k+1}, where $P^{k+1} \subseteq NV(P^k)$; then the reconfiguration delay is invisible outside the switch. It is called a *non-violation scheduling scheme* in this letter. More specifically, given a traffic matrix D and delay δ, a non-violation set scheduling scheme finds a set of schedules (P^1,...,P^k, P^{k+1},...,P^K) which satisfies the following.

- Each schedule restricts itself on using switching elements that belong to the non-violation set of the previous schedule $P^{k+1} \subseteq NV(P^k)$.
- Each schedule holds for δ slots in order to overlap the packet transmission of P^k and fabric setup of P^{k+1}.
- They cover the traffic matrix; $\sum_{k=1}^{K} \delta P^k \geq D$.

Notice that two consecutive schedules P^k and P^{k+1} are not necessarily different.

B. Example: Static Generalized Diagonal Sequence

An example of non-violation schedules is the Static Generalized Diagonal (SGD) sequence. The SGD sequence for an $N \times N$ switch includes $N + 1$ schedules, SGD^1 to SGD^{N+1}. Generally speaking, input i sequentially connects to output $i, i+1, \ldots, N, \text{idle}, 1, \ldots, i-1$ in schedule SGD^1 to SGD^{N+1}. Therefore for $\forall k$, $SGD^{(k+1) \bmod N} \subseteq NV(SGD^k)$, and we deminish the effect of reconfiguration delay. Fig. 2 shows an example on a 5×5 switch.

IV. Dynamic Diagonal(DD) Algorithm

SGD sequence fairly serves every input-output connection and works perfectly under uniform traffic. However the rigid schedules cannot adapt to various traffic patterns. An enhanced non-violation set scheduling scheme, Dynamic Diagonal (DD) algorithm is proposed here.

Algorithm Dynamic Diagonal (DD)

Input:

Current schedule P^k and residue request matrix R

Output:

Next schedule P^{k+1}

Procedure:

1) Initialize empty schedule P^{k+1}. Set all outputs available, $Out_N = [1]_N$.
2) Find the non-violation set of current schedule $NV(P^k)$.
3) Screen out the connection requests using the non-violation set of mirrors, store the results in the non-violation request matrix NVR in which
$$nvr_{ij} = \begin{cases} 1 & \text{if } nv(P^k)_{ij} = 1 \text{ and } r_{ij} \neq 0 \\ 0 & \text{otherwise} \end{cases}$$
4) If NVR is empty, we fail to find a schedule which can overlap its setup with P^k's transmission. Setup of P^{k+1} has to wait until P^k finishes. δ extra delay is therefore suffered and P^{k+1} can use every mirror.
   if (NVR is empty)
   update NVR, **if** ($r_{ij} \neq 0$) $nvr_{ij} = 1$, **else** $nvr_{ij} = 0$
   set flag $overlap =$ false
5) Determine schedule P^{k+1}.

for (input $i = 1$ to N)
  for (output $j = 1$ to N)
    if $((nvr_{ij} = 1)$ and $(out_j = 1))$
      assign output j to input i, $p_{ij}^k = 1$, $out_j = 0$
6) Update residue request matrix.
   $R = (R - \delta P^{k+1})^+$
7) Setup fabric connections.
   if $(overlap)$ setup schedule P^{k+1} now
   else setup schedule P^{k+1} after δ slots

Given a traffic batch, the first schedule P^1 starts with $R = D$ and all switching elements are available. Starting from input 1, each input sequentially searches for outputs 1 to N, and connects to the first available output. If the traffic load is high and traffic matrix D contains no zero entries, P^1 is the same as the SGD^1 pattern. Then the connections are set up accordingly during $[1, \delta]$ slots. In $[\delta + 1, 2\delta]$ slots, traffic are switched out according to schedule P^1; meanwhile P^2 is determined and set up for $R = R - \delta P^1$. However, P^2 could be the same as P^1 if all diagonal connections have requests (i.e., $\forall i$, $r_{ii} \neq 0$). The connection pattern evolves only when some input finishes transmit all its cells for the current output. Then in the next schedule, it will try to connect to a new output and so on. The procedure repeats until R is empty.

Therefore, the DD algorithm dynamically adjusts the schedule according to the traffic demands. It deminishes the reconfiguration delay more effectively for any traffic pattern.

V. Performance Analysis

A. Time Complexity and Hardware Implementation

Each execution of the DD algorithm is dominated by step 2) to 5), for a total time complexity of $O(N^2)$. This is much more efficient than the TSA algorithms, such as EXACT $O(N^5)$ [3], K-T $O(N^4)$ [4] and DOUBLE $O(N^2 \log N)$ [5]. To the best of our knowledge, most TSA algorithms only run in software with slow scheduling speed. A significant advantage of DD algorithm is that it can be easily implemented in hardware. The scheduler uses an N^2-bit vector for a request matrix, an N^2-bit vector for the non-violation set, an N-bit vector to check the availability of the outputs and N decision registers storing the connection information for each input. Fig. 3 illustrates the design of the hardware at input i.

B. Simulation Results

The performance of the DD algorithm is tested on a 16×16 switch. Traffic matrix are randomly generated, with most row/column sums equal or close to T (a simulation of high traffic load, which is the real situation under which most switches works). Various system configurations with different reconfiguration delays and comparatively short, medium or long accumulation times are tested. The statistics shown in Table I are based on 100000 random samples.

Compared to TSA scheduling algorithms, the DD algorithm shows a promising performance on total transmission time needed to switch out the traffic. Take one of the best TSA algorithms, DOUBLE [5], for example. Its total transmission time equals to $2T + 2N\delta$, which is much larger than even the worst case of DD algorithm. Further more, TSA algorithms constrain

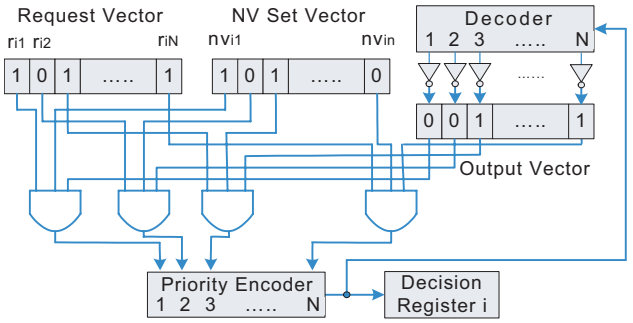

Fig. 3. Hardware implementation scheme for DD algorithm. For request and non-violation set vector, only bits related to input i are shown for simplicity. The priority encoder has highest priority 1 and lowest priority N. The selected output is saved in input i's decision register. Then the corresponding bit in output vector is updated to 0 (unavailable).

TABLE I
Performance of DD algorithm in 16×16 switch.

T	δ	Total Trans. Time				Num. of Extra Delay	
		DD			DOUBLE		
		AVG	SD	MAX	$(2T + 2N\delta)$	AVG	MAX
10	1	10.84	1.73	19	52	0.67	6
10	2	21.67	3.61	38	84	0.68	6
10	5	54.15	9.16	95	180	0.69	6
50	2	91.86	4.89	108	164	0.42	4
50	10	272.86	14.79	330	420	0.45	4
50	25	682.35	37.27	825	900	0.44	4
100	2	153.27	10.583	199	264	0.38	3
100	10	299.19	16.21	370	520	0.46	4
100	25	690.57	34.67	825	1000	0.43	4
100	50	1380.57	69.45	1650	1800	0.45	4

that the accumulation time T has to be larger than $N\delta$ (the least possible reconfiguration time required); while the DD algorithm fully releases that. Since T directly corresponds to the worst case delay bound [5], DD algorithm can provide a much better quality-of-service (QoS) guarantee. Although DD algorithm cannot fully avoid reconfiguration delay, our simulation results show on average less than one extra delay is suffered. It is interesting to note that the DD algorithm has a self-compensating capability: if the non-violation set contains very few switching elements for the current schedule, then the next schedule will have a much larger non-violation set to choose from.

References

[1] X. H. Ma and G. H. Kuo, "Optical switching technology comparison: optical MEMS vs. other technologies," *IEEE Commun. Mag.*, vol. 41, pp. S16–S23, Nov. 2003.
[2] P. D. Dobbelaere, K. Falta, S. Gloeckner, and S. Patra, "Digital MEMS for optical switching," *IEEE Commun. Mag.*, vol. 40, pp. 88–95, Mar. 2002.
[3] T. Inukai, "An efficient SS/TDMA time slot assignment algorithm," *IEEE Trans. Commun.*, vol. 27, pp. 1449–1455, Oct. 1979.
[4] I. S. Gopal and C. K. Wong, "Minimizing the number of switchings in an SS/TDMA system," *IEEE Trans. Commun.*, vol. 33, pp. 497–501, June 1985.
[5] B. Towles and W. J. Dally, "Guaranteed scheduling for switches with configuration overhead," *IEEE/ACM Trans. Networking*, vol. 11, pp. 835–847, Oct. 2003.
[6] X. Li and M. Hamdi, "On scheduling optical packet switches with reconfiguration delay," *IEEE J. Select. Areas Commun.*, vol. 21, pp. 1156–1164, Sept. 2003.
[7] K. Kar, D. Stiliadis, T. V. Lakshman, and L. Tassiulas, "Scheduling algorithms for optical packet fabrics," *IEEE J. Select. Areas Commun.*, vol. 21, pp. 1143–1155, Sept. 2003.