Minimizing Internal Speedup for Performance Guaranteed Switches With Optical Fabrics

Bin Wu, *Member, IEEE*, Kwan L. Yeung, *Senior Member, IEEE*, Mounir Hamdi, *Member, IEEE*, and Xin Li

***Abstract*—We consider traffic scheduling in an $N \times N$ packet switch with an optical switch fabric, where the fabric requires a reconfiguration overhead to change its switch configurations. To provide 100% throughput with bounded packet delay, a speedup in the switch fabric is necessary to compensate for both the reconfiguration overhead and the inefficiency of the scheduling algorithm. In order to reduce the implementation cost of the switch, we aim at minimizing the required speedup for a given packet delay bound. Conventional Birkhoff–von Neumann traffic matrix decomposition requires $N^2 - 2N + 2$ configurations in the schedule, which lead to a very large packet delay bound. The existing DOUBLE algorithm requires a fixed number of only $2N$ configurations, but it cannot adjust its schedule according to different switch parameters. In this paper, we first design a generic approach to decompose a traffic matrix into an arbitrary number of N_S ($N^2 - 2N + 2 > N_S > N$) configurations. Then, by taking the reconfiguration overhead into account, we formulate a speedup function. Minimizing the speedup function results in an efficient scheduling algorithm ADAPT. We further observe that the algorithmic efficiency of ADAPT can be improved by better utilizing the switch bandwidth. This leads to a more efficient algorithm SRF (Scheduling Residue First). ADAPT and SRF can automatically adjust the number of configurations in a schedule according to different switch parameters. We show that both algorithms outperform the existing DOUBLE algorithm.**

***Index Terms*—Optical switch fabric, performance guaranteed switching, reconfiguration overhead, speedup, scheduling.**

I. Introduction

RECENT progress on optical switching technologies [1]–[4] has enabled the implementation of high-speed scalable switches with optical switch fabrics, as shown in Fig. 1. These switches can efficiently provide huge switching capacity as demanded by the backbone routers in the Internet. Since the input/output modules are connected with the central switch fabric by optical fibers, the system can be distributed over several standard telecommunication racks. This reduces the power consumption for each rack, and makes the resulting switch architecture more scalable.

Manuscript received November 07, 2006; revised August 10, 2007 and August 10, 2007; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor A. Fumagalli. First published July 25, 2008; current version published April 15, 2009.

B. Wu is with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON N2L 3G1, Canada (e-mail: b7wu@uwaterloo.ca).

K. L. Yeung is with the Department of Electrical and Electronic Engineering, The University of Hong Kong, Pokfulam, Hong Kong (e-mail: kyeung@eee.hku. hk).

M. Hamdi and X. Li are with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong (e-mail: hamdi@cse.ust.hk; lixin@cse.ust.hk).

Digital Object Identifier 10.1109/TNET.2008.926501

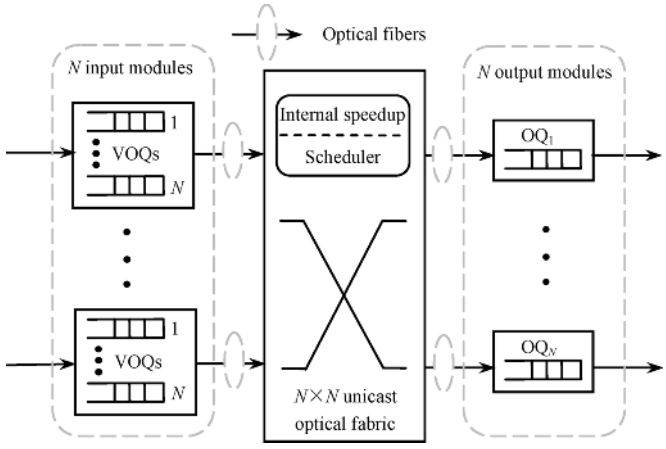

Fig. 1. Scalable switch with an optical switch fabric.

However, switches with optical fabrics suffer from a significant *reconfiguration overhead* when they update their configurations [5]. The reconfiguration overhead includes time needed for: 1) interconnection pattern update (ranging from 10 ns to several milliseconds for different optical switching technologies [1]–[4], [6]); 2) optical transceiver resynchronization (10–20 ns or higher [5]); and 3) extra clock margin to align optical signals from different input modules. With most fast optical switching technologies [1]–[4] available nowadays, the reconfiguration overhead is still more than one slot for a system with slotted time equal to 50 ns (64 bytes at 10 Gbps).

During the reconfiguration period, packet switching is prohibited. To achieve 100% throughput with bounded packet delay (or *performance guaranteed switching* [6]), the fabric has to transmit packets at an internal rate higher than the external line-rate, resulting in a *speedup*. The amount of speedup S is defined as the ratio of the internal packet transmission rate to the external line-rate. Speedup is directly associated with the implementation cost in practical switching systems. It concerns not only the internal optical transmission rate, but also the memory access time. In this paper, we focus on minimizing the speedup requirement for a given packet delay bound. The goal is to achieve a cost-effective solution while at the same time maintaining guaranteed QoS performance of the system.

Assume each switch reconfiguration takes an overhead of δ slots. Conventional slot-by-slot scheduling methods may severely cripple the performance of optical switches due to frequent reconfigurations. Hence, the scheduling rate has to be slowed down by holding each configuration for multiple time slots. Time slot assignment (TSA) [6]–[10] is a common approach to achieve this. Assume time is slotted and each time slot can accommodate one packet. The switch works in

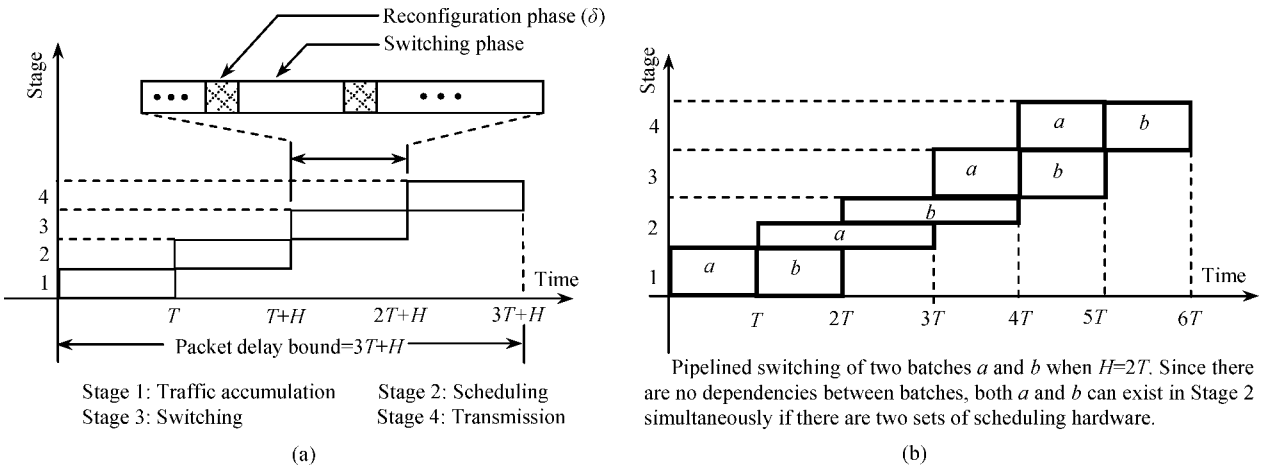

Fig. 2. Timing diagram for packet switching. (a) Pipelined switching and the packet delay bound. (b) An example of $H = 2T$.

a *pipelined* four-stage cycle: traffic accumulation, scheduling, switching and transmission, as shown in Fig. 2. Stage 1 is for traffic accumulation. Its duration T is a predefined system constant. Packets arrived within this duration form a *batch* which is stored in a traffic matrix $\boldsymbol{C}(\boldsymbol{T}) = \{c_{ij}\}$. Each entry c_{ij} denotes the number of packets arrived at input i and destined to output j. Assume the traffic has been regulated to be *admissible* before entering the switch, i.e., the entries in each line (row or column) of $\boldsymbol{C}(\boldsymbol{T})$ sum to at most T (defined as *maximum line sum* T). In Stage 2, the scheduling algorithm is executed in H time slots to compute a schedule consisting of (at most) N_S configurations for the accumulated traffic. Each configuration is given by a permutation matrix $\boldsymbol{P}_n = \{p_{ij}^{(n)}\}$ $(N_S \geq n \geq 1)$, where $p_{ij}^{(n)} = 1$ means that input port i is connected to output port j. A weight ϕ_n is assigned to each $\boldsymbol{P}_n$ and it denotes the number of slots that $\boldsymbol{P}_n$ should be kept for packet switching in Stage 3. In order to achieve 100% throughput, the set of N_S configurations must *cover* $\boldsymbol{C}(\boldsymbol{T})$, i.e., $\sum_{n=1}^{N_S} \phi_n p_{ij}^{(n)} \geq c_{ij}$ for any $i, j \in \{0, \ldots, N-1\}$. Because $\boldsymbol{C}(\boldsymbol{T})$ has N^2 entries and each configuration can cover at most N of them, the number of configurations N_S must be no less than the switch size N. Otherwise, the N_S configurations are not sufficient to cover every entry of $\boldsymbol{C}(\boldsymbol{T})$ [6], [8], [9]. In essence, this scheduling problem is equivalent to a traffic matrix decomposition problem, where the traffic matrix is decomposed into a set of N_S weighted configurations (or permutations). For optical switches, this decomposition is constrained by the reconfiguration overhead δ, and the scheduling algorithm needs to determine a proper number of configurations N_S to minimize speedup.

In Stage 3, the switch fabric is configured according to the N_S configurations and packets are switched to their designated output buffers. Without speedup, Stage 3 requires $\sum_{n=1}^{N_S} \phi_n + \delta N_S$ slots, where $\sum_{n=1}^{N_S} \phi_n$ is the total holding time for the N_S configurations and δN_S is the total amount of reconfiguration overhead. Since $\sum_{n=1}^{N_S} \phi_n + \delta N_S$ is generally larger than the traffic accumulation time T, speedup is needed to ensure that Stage 3 takes no more than T slots. During the holding time of a configuration, some input-output connections become idle (earlier than others) if their scheduled backlog packets are all sent. As a result, the schedule will contain empty slots [7] and this causes bandwidth loss, or algorithmic inefficiency. In general, this bandwidth loss increases with the holding time ϕ_n. But a short holding time implies frequent switch reconfigurations, or large hardware inefficiency (due to large δN_S). A good scheduling algorithm should compromise between hardware and algorithmic inefficiency, and achieve a balanced tradeoff to minimize the speedup requirement.

At a speedup of S, the slot duration for a single packet transmission in Stage 3 is shortened by S times. Then 100% throughput can be ensured if

$$\delta N_S + \frac{1}{S} \sum_{n=1}^{N_S} \phi_n \leq T. \tag{1}$$

The values of N_S and $\sum_{n=1}^{N_S} \phi_n$ in (1) are determined by the scheduling algorithm. Note that speedup can accelerate packet switching in Stage 3, but cannot reduce the total amount of reconfiguration overhead δN_S. Formula (1) also indicates that $T > \delta N_S (\geq \delta N)$ must be true for any feasible schedule.

Rearranging (1), we have the minimum required speedup as

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

where

$$S_{\text{reconfigure}} = \frac{T}{T - \delta N_S} \tag{3}$$

$$S_{\text{schedule}} = \frac{1}{T} \sum_{n=1}^{N_S} \phi_n. \tag{4}$$

$S_{\text{reconfigure}}$ compensates for hardware inefficiency caused by the N_S times of switch reconfigurations. S_{schedule} compensates for algorithmic inefficiency.

Without loss of generality, we define a flow as a series of packets coming from the same input port and destined to the same output port of the switch. Since packets in each flow follow first-in-first-out (FIFO) [11] order in passing through the switch, there is no packet out-of-order problem within the same flow. (But packets in different flows may interleave at the output buffers.) Stage 4 takes another T slots to dispatch the packets from the output buffers to the output lines in the FIFO order. Consider a packet arrived at the input buffer in the first

slot of Stage 1. It suffers a delay of T slots in Stage 1 for traffic accumulation (i.e., the worst-case accumulation delay), and another delay of H slots in Stage 2 for algorithm execution. In the worst case, this packet will experience a maximum delay of $2T$ slots in Stages 3 and 4 (assume it is sent onto the output line in the last slot of Stage 4). Taking all the four stages into account, the delay experienced by any packet at the switch is bounded by $3T + H$ slots as shown in Fig. 2(a). Note that the value of H depends on how the scheduling hardware is engineered. For $H \leq T$, a single set of scheduling hardware can schedule all incoming batches. For $H > T$, $\lceil H/T \rceil$ sets of scheduling hardware can be used in Stage 2 for pipelined scheduling of multiple batches at the same time. This is feasible because each batch is independently processed. Fig. 2(b) shows an example of $H = 2T$, where two sets of scheduling hardware are used.

Several TSA-based scheduling algorithms have been proposed to achieve performance guaranteed switching [6], [8], [9]. Some of them target at minimizing the packet delay bound using the minimum number of $N_S = N$ configurations in the schedule. These algorithms are called minimum-delay scheduling algorithms. Examples include MIN [6] and QLEF [8], [9]. They generally have low algorithmic efficiency and thus require a very large S_{schedule}. Other algorithms favor larger number of configurations to achieve higher algorithmic efficiency. For example, EXACT [6] adopts the classic Birkhoff–von Neumann decomposition [12]–[15] to generate $N_S = N^2 - 2N + 2$ configurations. It achieves the smallest $S_{\text{schedule}} = 1$, but results in a large packet delay bound, because the large value of N_S requires a very large traffic accumulation time T to ensure $T > \delta N_S$. DOUBLE [6] makes an efficient tradeoff between the two extremes by using $N_S = 2N$ configurations to achieve $S_{\text{schedule}} = 2$. To the best of our knowledge, all previous performance guaranteed scheduling algorithms can only produce schedules with one of the three fixed N_S values: N, $N^2 - 2N + 2$ or $2N$. Obviously that may not well suit switches with different values of switch parameters, namely, traffic accumulation time T, switch size N and the amount of reconfiguration overhead δ. It is interesting to investigate 1) what will happen if other N_S values are used; 2) which N_S value can minimize the overall speedup; and 3) how the switch parameters T, N, and δ can affect the required speedup.

The above questions can be answered by a *speedup function* $S = f(N_S)$ derived in this paper for $N^2-2N+2 > N_S > N$.[1] Minimizing the speedup function leads to the proposal of a novel ADAPT algorithm. ADAPT works by converting the traffic matrix $\boldsymbol{C}(\boldsymbol{T})$ into a weighted sum of a quotient matrix $\boldsymbol{Q}$ and a residue matrix $\boldsymbol{R}$. Then, $\boldsymbol{Q}$ is covered by $N_S - N$ configurations and $\boldsymbol{R}$ is covered by the other N configurations (detailed in Sections II and III). We further show that the performance of ADAPT can be enhanced by sending more packets in the $N_S - N$ configurations devoted to $\boldsymbol{Q}$. This leads to another algorithm SRF (Scheduling Residue First), which requires an even lower speedup than ADAPT. Both ADAPT and SRF outperform DOUBLE [6], because they always construct a schedule with a proper number of N_S configurations (instead of fixing $N_S = 2N$) to minimize speedup. In other words, both ADAPT and SRF can automatically adjust the schedule according to different switch parameters T, N, and δ. Note that ADAPT and SRF are based on a generic matrix decomposition approach proposed in this paper. This matrix decomposition technique may also find applications in other networks, such as SS/TDMA [16], [17], TWIN [18] and wireless sensor networks [19]. It can also be applied to *unconstrained switches* [6] (e.g., electronic switches) to reduce the number of configurations in the schedule (compared to Birkhoff–von Neumann decomposition) [14].

The rest of the paper is organized as follows. In Section II, we derive a generic approach to decompose a traffic matrix into N_S configurations ($N^2 - 2N + 2 > N_S > N$). Based on the traffic matrix decomposition, our speedup function $S = f(N_S)$ is formulated. ADAPT algorithm is designed in Section III and SRF algorithm is proposed in Section IV. Section V gives some discussions. The paper is concluded in Section VI.

II. Traffic Matrix Decomposition and Speedup Function

A. Traffic Matrix Decomposition

To generate a schedule consisting of at most N_S switch configurations, we use $T/(N_S - N)$ to divide each entry $c_{ij} \in \boldsymbol{C}(\boldsymbol{T})$. For simplicity, we first assume $T/(N_S - N)$ is an integer. The traffic matrix $\boldsymbol{C}(\boldsymbol{T})$ is then converted into a weighted sum of a *quotient matrix* $\boldsymbol{Q} = \{q_{ij}\}$ and a *residue matrix* $\boldsymbol{R} = \{r_{ij}\}$. That is,

$$\begin{aligned} \boldsymbol{C}(\boldsymbol{T}) &= \frac{T}{N_S - N} \times \boldsymbol{Q} + \boldsymbol{R}, \\ \text{where } q_{ij} &= \left\lfloor \frac{c_{ij}}{T/(N_S - N)} \right\rfloor \\ \text{and } r_{ij} &= c_{ij} - \frac{T}{N_S - N} \times q_{ij}. \end{aligned} \tag{5}$$

Since the maximum line sum of $\boldsymbol{C}(\boldsymbol{T})$ is T, we have

$$\sum_{i=0}^{N-1} c_{ij} \leq T \text{ and } \sum_{j=0}^{N-1} c_{ij} \leq T, \ \forall i, j \in \{0, \cdots N-1\}. \tag{6}$$

From (5) and (6), we get

$$\begin{aligned} \sum_{i=0}^{N-1} q_{ij} &= \sum_{i=0}^{N-1} \left\lfloor \frac{c_{ij}}{T/(N_S - N)} \right\rfloor \\ &\leq \left\lfloor \frac{\sum_{i=0}^{N-1} c_{ij}}{T/(N_S - N)} \right\rfloor \\ &\leq \left\lfloor \frac{T}{T/(N_S - N)} \right\rfloor \leq N_S - N \end{aligned} \tag{7}$$

and

$$\sum_{j=0}^{N-1} q_{ij} \leq N_S - N. \tag{8}$$

With (7) and (8), it is easy to see that the maximum line sum of $\boldsymbol{Q}$ is at most $N_S - N$. According to Lemma 1 below, the quotient matrix $\boldsymbol{Q}$ can be covered by $N_S - N$ configurations.

[1] We focus on this N_S range, because $N_S = N^2 - 2N + 2$ corresponds to Birkhoff–von Neumann decomposition, and $N_S = N$ corresponds to minimum-delay scheduling which is handled by other algorithms (such as QLEF [8], [9]).

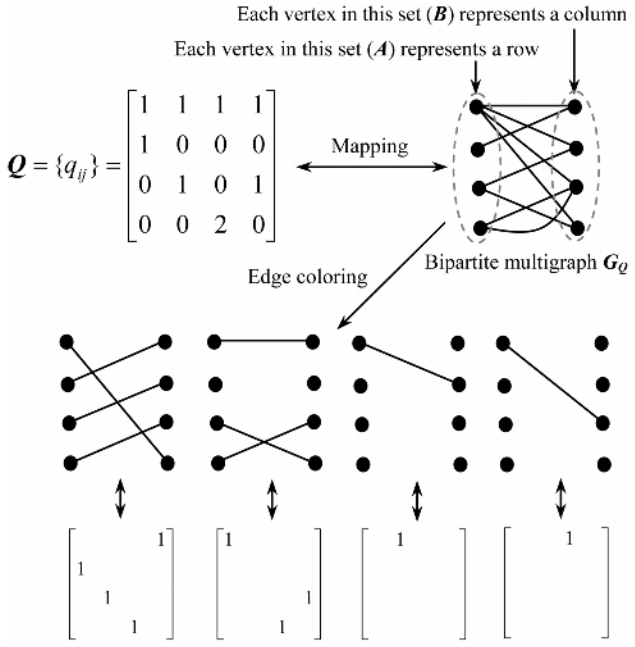

Fig. 3. Bipartite multigraph and edge coloring.

Lemma 1: An $N \times N$ matrix with maximum line sum $N_S - N$ can be covered by $N_S - N$ configurations, each with a weight of 1.

Proof: We first construct a bipartite multigraph [7], [20][21] $\boldsymbol{G}_Q$ from $\boldsymbol{Q}$, as illustrated by the example in Fig. 3. Rows and columns of $\boldsymbol{Q}$ translate to two sets of vertices $\boldsymbol{A}$ and $\boldsymbol{B}$ in $\boldsymbol{G}_Q$, and each entry q_{ij} translates to q_{ij} edges connecting vertices $i \in \boldsymbol{A}$ and $j \in \boldsymbol{B}$. Since the maximum vertex degree is at most $N_S - N$, $\boldsymbol{G}_Q$ can be edge-colored [20], [21] using $N_S - N$ colors, such that the edges incident on the same vertex have different colors. Each color is then mapped back to generate a configuration, where the edges in this color translate to $1s$ at the corresponding entries and all other entries are $0s$. As a result, the quotient matrix $\boldsymbol{Q}$ can be covered by the $N_S - N$ configurations obtained, with a weight of 1 for each configuration. ■

On the other hand, from (5) we have

$$r_{ij} < \frac{T}{N_S - N}, \forall r_{ij} \in \boldsymbol{R}. \tag{9}$$

Therefore, the residue matrix $\boldsymbol{R}$ can be covered by any N *non-overlapping configurations* with a weight $T/(N_S - N)$ for each. "Non-overlapping" means that any two of the N configurations do not cover the same entry of $\boldsymbol{R}$. Mathematically, these N non-overlapping configurations can add to an all-1 matrix. They can be chosen (or predefined) arbitrarily without any explicit computation.[2]

In summary, $\boldsymbol{C}(\boldsymbol{T})$ in (5) can be covered by N_S configurations. Among them, $N_S - N$ configurations are devoted to $\boldsymbol{Q}$ and the other N configurations are devoted to $\boldsymbol{R}$. Each configuration is equally weighted by $\phi_n = T/(N_S - N)$. This complies with our earlier assumption of using N_S configurations in the schedule.

[2]However, SRF in Section IV schedules $\boldsymbol{R}$ in a more careful way to improve the algorithmic efficiency.

B. Speedup Function

With the above traffic matrix decomposition, S_{schedule} in (4) can be formulated as follows:

$$S_{\text{schedule}} = \frac{1}{T}\sum_{n=1}^{N_S}\phi_n = \frac{1}{T} \times N_S \times \frac{T}{N_S - N} = 1 + \frac{N}{N_S - N}. \tag{10}$$

Note that S_{schedule} is further reduced in Section IV by (19).

If $T/(N_S - N)$ is not an integer, we can replace $T/(N_S - N)$ by $\lceil T/(N_S - N) \rceil$. From (10), this would increase S_{schedule} by at most N_S/T. When $T \gg N_S$, it can be ignored. For simplicity, we assume $T/(N_S - N)$ is an integer.

Note that T, N, and δ are predefined switch parameters. From (2) and (10), the overall speedup S can be expressed in N_S using the speedup function below:

$$S = f(N_S) = \frac{T}{T - \delta N_S}\left(1 + \frac{N}{N_S - N}\right) = \frac{TN_S}{(T - \delta N_S)(N_S - N)}. \tag{11}$$

The importance of the above speedup function can be summarized as follows: 1) it formulates how speedup S changes with the number of configurations N_S (in the range of $N^2 - 2N + 2 > N_S > N$); and 2) it allows us to study how the switch-dependent parameters T, N, and δ can affect speedup S.

C. An Example Based on DOUBLE

The traffic matrix decomposition in the existing DOUBLE algorithm [6] can be regarded as a special case of our proposed decomposition with $N_S = 2N$. As mentioned in Section I, DOUBLE uses $N_S = 2N$ to achieve $S_{\text{schedule}} = 2$. This is obtained by replacing N_S in (10) by $2N$. In DOUBLE, $c_{ij} \in \boldsymbol{C}(\boldsymbol{T})$ is divided by $T/(N_S - N) = T/N$ to get the quotient matrix $\boldsymbol{Q}$ and the residue matrix $\boldsymbol{R}$. Then, both $\boldsymbol{Q}$ and $\boldsymbol{R}$ are covered by N configurations. Particularly, the $N_S - N = N$ configurations devoted to $\boldsymbol{Q}$ are obtained from edge-coloring, and the other N non-overlapping configurations devoted to $\boldsymbol{R}$ can be chosen arbitrarily [6]. Each configuration is equally weighted by $\phi_n = T/(N_S - N) = T/N$. Fig. 4 gives an example of DOUBLE execution.

III. ADAPT ALGORITHM

A. ADAPT Algorithm

Based on the speedup function $S = f(N_S)$ in (11), we can design a scheduling algorithm to minimize the overall speedup S. Let

$$\frac{dS}{dN_S} = \frac{df(N_S)}{dN_S} = 0. \tag{12}$$

Solving (12) for N_S, we get

$$N_S = \sqrt{\frac{TN}{\delta}} = \lambda N \tag{13}$$

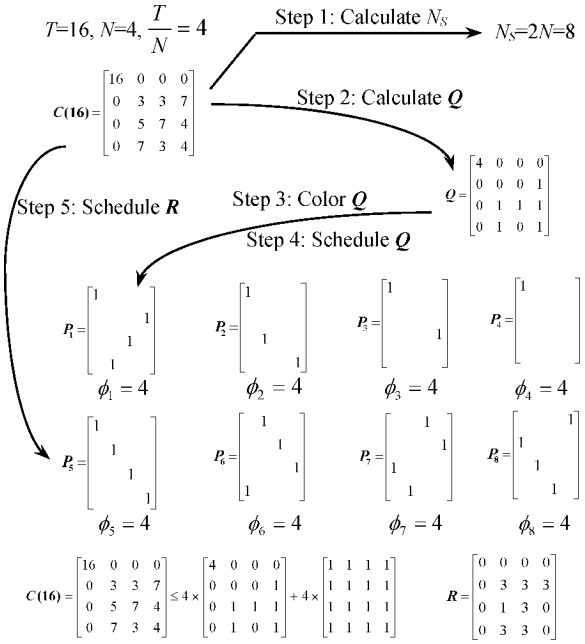

Fig. 4. Eample of DOUBLE execution. The all-1 matrix used to cover $\boldsymbol{R}$ equals to the sum of the N non-overlapping configurations $(\boldsymbol{P}_5 - \boldsymbol{P}_8)$.

where

$$\lambda = \sqrt{\frac{T}{\delta N}}. \tag{14}$$

Since $T > \delta N_S \geq \delta N$, we have $\lambda > 1$. λ is used to normalize the switch-dependent parameters T, N, and δ. Without this normalization, it is generally difficult to compare speedup requirement among switches with different parameters T, N, and δ. Otherwise, the comparison would be too complex, where each parameter should be considered one by one and others should be kept the same for a fair comparison. From (12)–(14), we can see that the overall speedup S is minimized if $N_S = \lambda N$ configurations are used for scheduling.

The above analysis leads to our ADAPT algorithm, as summarized in Fig. 5. The number of configurations N_S required by ADAPT is self-adjusted with switch parameters T, N, and δ using (13). The traffic matrix $\boldsymbol{C}(\boldsymbol{T})$ is then covered by the N_S configurations obtained from the decomposition in Section II-A. In practice, if λN and $T/(N_S - N)$ are not integers, we can set $N_S = \lfloor \lambda N \rfloor$ and $\phi_n = \lceil T/(N_S - N) \rceil$. Since $N_S = N$ is not allowed, if $\lfloor \lambda N \rfloor = N$, we can use $N_S = N + 1$ instead.

Without loss of generality, let S_{ADAPT} be the overall speedup required by ADAPT. Substituting (13) into (11), we have

$$S_{\mathrm{ADAPT}} = f(N_S)\Big|_{N_S=\lambda N} = \frac{TN_S}{(T-\delta N_S)(N_S - N)}\Big|_{N_S=\lambda N} = \frac{\lambda T}{(T-\delta\lambda N)(\lambda - 1)} = \left(\frac{\lambda}{\lambda - 1}\right)^2. \tag{15}$$

We can see that the minimized speedup (S_{ADAPT}) only depends on the value of λ. In other words, switches with the same value of λ require the same speedup. This theoretical insight cannot be easily seen without λ.

ADAPT ALGORITHM

Input:

An $N \times N$ traffic matrix $\boldsymbol{C}(\boldsymbol{T})=\{c_{ij}\}$ with maximum line sum no more than T, and the reconfiguration overhead δ.

Output:

At most N_S configurations $\boldsymbol{P}_1, \ldots, \boldsymbol{P}_{N_S}$ and the corresponding weights $\phi_1, \ldots, \phi_{N_S}$.

Step 1. Calculate N_S:

$$\lambda = \sqrt{\frac{T}{\delta N}} \quad \text{and} \quad N_S = \lfloor \lambda N \rfloor.$$

If $\lfloor \lambda N \rfloor = N$, set $N_S = N+1$ instead.

Step 2. Calculate the quotient matrix Q:

Construct an $N \times N$ matrix $\boldsymbol{Q}=\{q_{ij}\}$ such that

$$\boldsymbol{C}(\boldsymbol{T}) = \left\lceil \frac{T}{N_S - N} \right\rceil \times \boldsymbol{Q} + \boldsymbol{R} \quad \text{and} \quad q_{ij} = \left\lfloor \frac{c_{ij}}{\lceil T/(N_S - N) \rceil} \right\rfloor.$$

Step 3. Color Q:

Construct a bipartite multigraph $\boldsymbol{G}_{\boldsymbol{Q}}$ from $\boldsymbol{Q}$. Rows and columns of $\boldsymbol{Q}$ translate to two sets of vertices $\boldsymbol{A}$ and $\boldsymbol{B}$ in $\boldsymbol{G}_{\boldsymbol{Q}}$, and each entry $q_{ij} \in \boldsymbol{Q}$ translates to q_{ij} edges connecting vertices $i \in \boldsymbol{A}$ and $j \in \boldsymbol{B}$. Find a minimal edge-coloring of $\boldsymbol{G}_{\boldsymbol{Q}}$ to get at most $N_S - N$ colors, such that the edges incident on the same vertex have different colors. Set $1 \rightarrow n$.

Step 4. Schedule the quotient matrix Q:

For a specific color in the edge-coloring of $\boldsymbol{G}_{\boldsymbol{Q}}$, construct a configuration $\boldsymbol{P}_n$ from the edges in that color by setting the corresponding entries to 1 in $\boldsymbol{P}_n$ (all other entries in $\boldsymbol{P}_n$ are set to 0). Set the weight

$$\phi_n = \lceil T/(N_S - N) \rceil$$

and $n+1 \rightarrow n$. Repeat *Step 4* for each and all of the colors in the edge-coloring of $\boldsymbol{G}_{\boldsymbol{Q}}$.

Step 5. Schedule the residue matrix R:

Find any N non-overlapping configurations $\boldsymbol{P}_n$, $n \in \{N_S - N + 1, \ldots, N_S\}$ and set the following weight for each of the N configurations.

$$\phi_n = \lceil T/(N_S - N) \rceil.$$

Fig. 5. ADAPT algorithm.

B. Performance Comparison with DOUBLE

Let S_{DOUBLE} denote the overall speedup required by DOUBLE. Since DOUBLE uses $N_S = 2N$ configurations to achieve $S_{\mathrm{schedule}} = 2$, from (2) we have

$$S_{\mathrm{DOUBLE}} = \frac{T}{T - \delta N_S} S_{\mathrm{schedule}}\big|_{N_S=2N \text{ and } S_{\mathrm{schedule}}=2} = \frac{2T}{T - 2\delta N} = \frac{2\lambda^2}{\lambda^2 - 2}. \tag{16}$$

ADAPT and DOUBLE are feasible only if S_{ADAPT} and S_{DOUBLE} are positive. From (15) and (16), this requires $\lambda > 1$ for ADAPT and $\lambda > \sqrt{2}$ for DOUBLE. In other words, ADAPT can always generate a feasible schedule (because $\lambda > 1$ or $T > \delta N$ is always true), but DOUBLE is feasible only if $\lambda > \sqrt{2}$ or $T > 2\delta N$. In fact, if $T \leq 2\delta N$ in DOUBLE, the T time slots in the switching stage (Stage 3 in Fig. 2) will be fully occupied by the $2N$ times of switch reconfigurations, and no time can be left for packet switching. In this case, DOUBLE cannot generate a feasible schedule even if the speedup is infinite.

Recall that the packet delay bound is given by $3T + H$ time slots, and T is directly related to the QoS that the switch can provide. ADAPT can achieve a tighter packet delay bound than DOUBLE. For example, if we set $T = 2\delta N$, a delay bound of $6\delta N + H$ slots can be achieved by ADAPT at a proper speedup.

However, this is impossible in DOUBLE as the minimum delay bound it can provide must be larger than $6\delta N + H$ slots.

Even for the region where both DOUBLE and ADAPT are feasible (i.e., $\lambda > \sqrt{2}$), from (15) and (16), it is easy to prove that $S_{\text{ADAPT}} \leq S_{\text{DOUBLE}}$ is always true. Therefore, for the same set of switch parameters T, N, and δ, the overall speedup required by ADAPT is always smaller than that required by DOUBLE (except $\lambda = 2$ or $T = 4\delta N$, where $S_{\text{ADAPT}} = S_{\text{DOUBLE}} = 4$). For example, with $T = 16\delta N$, we have $S_{\text{ADAPT}} = 1.78$ and $S_{\text{DOUBLE}} = 2.29$.

The running time of ADAPT is dominated by edge-coloring of $(N_S - N) \times N = (\lambda - 1)N^2$ edges. This gives a time complexity of $O((\lambda - 1)N^2 \log N)$ for ADAPT.

IV. SRF Algorithm

In both DOUBLE and ADAPT, the N non-overlapping configurations devoted to the residue matrix $\boldsymbol{R}$ are chosen arbitrarily without explicit computation. In this section, we design an SRF (Scheduling Residue First) algorithm which schedules $\boldsymbol{R}$ more carefully to further reduce S_{schedule}. Since DOUBLE is a special case of ADAPT at $N_S = 2N$, for simplicity, we first design SRF based on DOUBLE.

A. Observation and Motivation

DOUBLE assigns an equal weight of T/N to each of its $2N$ configurations (see the example in Fig. 4). We observe that a schedule generated by DOUBLE may be inefficient. In particular, the bandwidth in the N configurations devoted to $\boldsymbol{Q}$ is generally not well utilized (due to the bandwidth loss). If such otherwise wasted bandwidth can be used to transmit some packets in $\boldsymbol{R}$, the remaining packets in $\boldsymbol{R}$ may be sent in a shorter amount of time. That is, some configurations devoted to $\boldsymbol{R}$ may require a weight less than T/N. Then the overall switch speedup requirement can be further reduced.

In DOUBLE, $\boldsymbol{C(T)} = [T/N] \times \boldsymbol{Q} + \boldsymbol{R}$ and $r_{ij} < T/N$ for any entry $r_{ij} \in \boldsymbol{R}$. If $r_{ij} > T/(2N)$, we call the entry an *LER* (large entry in $\boldsymbol{R}$). Otherwise it is an *SER* (small entry in $\boldsymbol{R}$). We have the following Lemma 2 (proved in Appendix I).

Lemma 2: In DOUBLE, if a line (row i or column j) of $\boldsymbol{R}$ contains k LERs, then in $\boldsymbol{Q}$ we have

$$\sum_{j=0}^{N-1} q_{ij} \leq N - \left\lceil \frac{k}{2} \right\rceil \text{ for row } i$$

or

$$\sum_{i=0}^{N-1} q_{ij} \leq N - \left\lceil \frac{k}{2} \right\rceil \text{ for column } j.$$

Fig. 6 gives an example based on the same $\boldsymbol{Q}$ and $\boldsymbol{R}$ as in Fig. 4. The third row of $\boldsymbol{R}$ contains $k = 1$ LERs ($> T/(2N) = 2$). Then the entries in the third row of $\boldsymbol{Q}$ sum to at most $N - \lceil k/2 \rceil = 3$.

Based on Lemma 2, we can move some packets from $\boldsymbol{R}$ to $\boldsymbol{Q}$, while still keeping the maximum line sum of $\boldsymbol{Q}$ no more than N. Note that all ϕ_n in DOUBLE equal to T/N. If a line in $\boldsymbol{R}$ contains k LERs, we can move half (i.e., $\lceil k/2 \rceil$) of them to $\boldsymbol{Q}$. This is achieved by setting the moved LER entries to 0 in $\boldsymbol{R}$, and at the same time increasing the corresponding entries in $\boldsymbol{Q}$ by 1. Fig. 6 shows this operation. We use $\boldsymbol{Q'}$ and $\boldsymbol{R'}$ to denote the updated $\boldsymbol{Q}$ and $\boldsymbol{R}$. In Fig. 6, because the maximum line sum of $\boldsymbol{Q'}$ is at most 4, $\boldsymbol{Q'}$ can still be covered by $N = 4$ configurations, each with the same original weight $\phi_n = T/N = 4$. Compared to the example in Fig. 4, more packets are scheduled in the N configurations devoted to the quotient matrix.

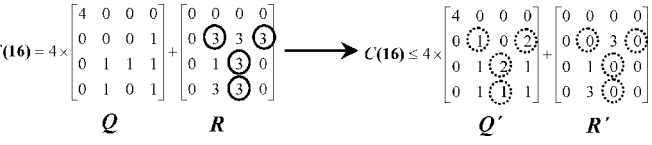

Fig. 6. Move the circled LERs from $\boldsymbol{R}$ to $\boldsymbol{Q}$ ($T = 16,\ N = 4$).

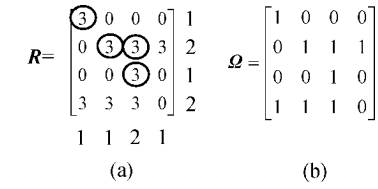

Fig. 7. Example of $\boldsymbol{R}$ and $\boldsymbol{\Omega}$.

Note that each line of $\boldsymbol{R}$ can contain at most N LERs. If half of them are moved to $\boldsymbol{Q}$ (without increasing its maximum line sum), then each line of $\boldsymbol{R'}$ will contain at most $N/2$ LERs (if N is even). Though we still need N non-overlapping configurations to cover $\boldsymbol{R'}$, it is possible to reduce the weight for some of them. Specifically, we may find half $(N/2)$ of the non-overlapping configurations, each with the original weight $\phi_n = T/N$, to cover all the remaining LERs in $\boldsymbol{R'}$. The other half of the non-overlapping configurations only need a *reduced* weight of $\phi_n = T/(2N)$ to cover the remaining SERs. If this can be achieved, then S_{schedule} can be reduced to

$$S_{\text{schedule}} = \frac{1}{T}\sum_{n=1}^{2N} \phi_n = \frac{1}{T} \times \left(\frac{T}{N} \times N + \frac{T}{N} \times \frac{N}{2} + \frac{T}{2N} \times \frac{N}{2} \right) = 1.75. \tag{17}$$

The above observation motivates us to explore a more efficient scheduling algorithm than DOUBLE. The key is to find a proper residue matrix $\boldsymbol{R'}$, such that $\boldsymbol{R'}$ contains at most $N/2$ LERs in *each* line, and *each* line sum of $\boldsymbol{Q'}$ is not larger than N. Generally, this is not easy. For example, we assume that all the non-zero entries $(3s)$ in Fig. 7(a) are LERs. The number next to each line of $\boldsymbol{R}$ gives the number of LERs that can be moved from this line to $\boldsymbol{Q}$, i.e., half of the number of LERs in this line, or $\lceil k/2 \rceil$. In Fig. 7(a), if the four circled entries are moved to $\boldsymbol{Q}$, then we cannot further move any other LERs without violating the quota of the corresponding lines. At this point, the last row of $\boldsymbol{R'}$ still contains 3 LERs where $3 > N/2 = 2$. This makes it impossible to cover the remaining LERs by $N/2$ configurations. For larger switch size N, it will be more difficult to figure out a proper set of LERs to move.

B. SRF Algorithm

For simplicity, we first consider even switch size N (an example for odd N is given in Appendix II). We map the residue matrix $\boldsymbol{R} = \{r_{ij}\}$ into a 0/1 *reference matrix* $\boldsymbol{\Omega} = \{\omega_{ij}\}$ such

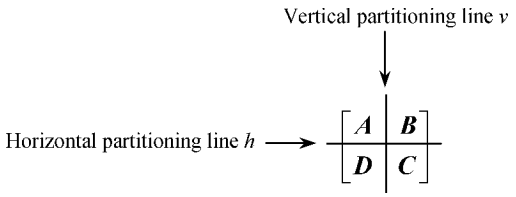

Fig. 8. Partition $\boldsymbol{\Omega}$ into four zones.

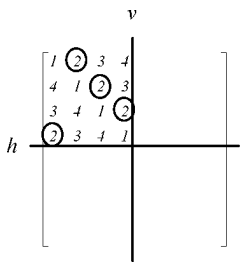

Fig. 9. $\boldsymbol{PPC}$s for an 8 × 8 matrix.

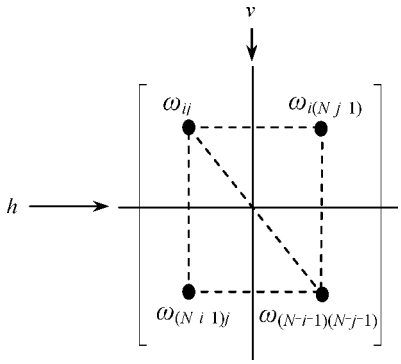

Fig. 10. Images.

that $\omega_{ij} = 1$ if r_{ij} is an LER and $\omega_{ij} = 0$ otherwise. Fig. 7(b) gives an example of $\boldsymbol{\Omega}$ for $\boldsymbol{R}$ in Fig. 7(a) (where all the $3s$ are LERs). A horizontal line h and a vertical line v are used to partition $\boldsymbol{\Omega}$ into four $(N/2) \times (N/2)$ zones $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ and $\boldsymbol{D}$ as shown in Fig. 8. The partitioning line v (or h) separates each row (or column) into two parts, and each part contains $N/2$ entries. In addition, we define a set of $N/2$ non-overlapping *predefined partial configurations* ($\boldsymbol{PPC}$s) in a cyclic manner to cover all the entries in zone $\boldsymbol{A}$. Specifically, if an entry (i, j)[3] in zone $\boldsymbol{A}$ is covered by $\boldsymbol{PPC_p}$ (where $N/2 \geq p \geq 1$), then

$$p = \left[\frac{N}{2} - (i - j)\right] \bmod \left(\frac{N}{2}\right) + 1, \quad \text{where } \frac{N}{2} > i \geq 0 \text{ and } \frac{N}{2} > j \geq 0. \quad (18)$$

From (18),[4] each $\boldsymbol{PPC_p}$ covers $N/2$ entries in zone $\boldsymbol{A}$. Fig. 9 gives an example where four non-overlapping $\boldsymbol{PPC}$s (i.e., $\boldsymbol{PPC_1}$–$\boldsymbol{PPC_4}$) are used to cover all the entries in zone $\boldsymbol{A}$ of an 8 × 8 matrix. For easy reading, we use an italic number at an entry to denote the index number p if $\boldsymbol{PPC_p}$ covers that entry. For example, the circled entries in Fig. 9 are covered by $\boldsymbol{PPC_2}$.

For an arbitrary entry $\omega_{ij} \in \boldsymbol{\Omega}$, we define its *line images* and *diagonal image* as shown in Fig. 10. Particularly, $\omega_{i(N-j-1)}$ and $\omega_{(N-i-1)j}$ are line images of ω_{ij}, because they are symmetrical to ω_{ij} with respect to the two partitioning lines v and h. $\omega_{(N-i-1)(N-j-1)}$ is the diagonal image as it is symmetrical to ω_{ij} with respect to the cross-point of v and h.

Without loss of generality, we consider $\boldsymbol{PPC_p}$. For *each* entry ω_{ij} (in zone $\boldsymbol{A}$) covered by $\boldsymbol{PPC_p}$, we find its line and diagonal images. The 4-tuple $\{\omega_{ij}, \omega_{i(N-j-1)}, \omega_{(N-i-1)j}, \omega_{(N-i-1)(N-j-1)}\}$ have 16 possible values (or combinations) as shown in Fig. 11. The tuples in Figs. 11(a)–11(l) are defined as *diagonal dominant* tuples (DD tuples), and the two circled diagonal entries in each DD tuple are defined as *dominant entries*. Each dominant entry in a DD tuple is no less than *both* of its line images. On the contrary, the non-DD tuples in Figs. 11(m)–11(p) do not have such a property (i.e., either pair of diagonal entries in these tuples are not diagonal dominant). Each non-DD tuple in Figs. 11(m)–11(n) is called a *column isomorphic tuple* because the two columns are exactly the same. Similarly, the non-DD tuples in Figs. 11(o)–11(p) are called *row isomorphic tuples*.

To cover the residue matrix using N non-overlapping configurations $\boldsymbol{P}_1 \sim \boldsymbol{P}_N$, we first initialize $\boldsymbol{P}_1 \sim \boldsymbol{P}_N$ to all-0 matrices. Then, based on the reference matrix $\boldsymbol{\Omega}$, each $\boldsymbol{PPC_p}(N/2 \geq p \geq 1)$ is sequentially examined to construct two configurations $\boldsymbol{P}_p$ and $\boldsymbol{P}_{p+N/2}$. Specifically, for each ω_{ij} covered by $\boldsymbol{PPC_p}$, we find its line and diagonal images to form a 4-tuple $\{\omega_{ij}, \omega_{i(N-j-1)}, \omega_{(N-i-1)j}, \omega_{(N-i-1)(N-j-1)}\}$. The corresponding entries $\{(i, j), (i, N-j-1), (N-i-1, j), (N-i-1, N-j-1)\}$ in $\boldsymbol{P}_p$ and $\boldsymbol{P}_{p+N/2}$ are set according to the three cases below.

Case 1: If the 4-tuple is a DD tuple, we set the two dominant entries to 1 in $\boldsymbol{P}_p$, and set the other two non-dominant entries to 1 in $\boldsymbol{P}_{p+N/2}$.

Case 2: If the 4-tuple is a row isomorphic tuple, we check whether there exist other row isomorphic tuples in the same row pair $\{i, N-i-1\}$ (which may occur when examining an earlier $\boldsymbol{PPC_t}$, $t < p$.) If no, we set either pair of the diagonal entries to 1 in $\boldsymbol{P}_p$, and set their line images to 1 in $\boldsymbol{P}_{p+N/2}$. For example, if the two entries (i, j) and $(N-i-1, N-j-1)$ are set to 1 in $\boldsymbol{P}_p$, then the entries $(i, N-j-1)$ and $(N-i-1, j)$ are set to 1 in $\boldsymbol{P}_{p+N/2}$. On the other hand, if some row isomorphic tuples occurred earlier in the same row pair, we set the entries in $\boldsymbol{P}_p$ and $\boldsymbol{P}_{p+N/2}$ according to the most recently occurred one. Assume that the most recent row isomorphic tuple in this row pair occurred in examining $\boldsymbol{PPC_t}$ (where $p > t$). Note that two configurations $\boldsymbol{P}_t$ and $\boldsymbol{P}_{t+N/2}$ have been obtained in examining $\boldsymbol{PPC_t}$. If $\boldsymbol{P}_t$ was set to cover a "1" in row i and a "0" in row $N-i-1$ of $\boldsymbol{\Omega}$, then we set $\boldsymbol{P}_p$ to cover a "1" in row $N-i-1$ and a "0" in row i, and vice versa. Fig. 12 gives an example. Assume that the two dash-circled entries have been set to 1 in $\boldsymbol{P}_t$. Then, the two solid-circled entries are set to 1 in $\boldsymbol{P}_p$. At the same time, their line images (i.e., the two entries in the triangles) are set to 1 in $\boldsymbol{P}_{p+N/2}$. The goal is to let $\boldsymbol{P}_t$ and $\boldsymbol{P}_p$ cover the $1s$ in row i and row $N-i-1$ of $\boldsymbol{\Omega}$ in an alternating manner. We call this the *butterfly mechanism*.

[3]For simplicity, we denote an entry by either its location in the matrix (e.g., entry (i, j)), or the matrix element at this location (e.g., entry ω_{ij}).

[4]Note that the $\boldsymbol{PPC}$s are not necessarily predefined in the cyclic manner as in (18). In fact, any set of $N/2$ non-overlapping permutation sub-matrices in zone $\boldsymbol{A}$ can be used as $\boldsymbol{PPC}$s. We use (18) only to facilitate the presentation.

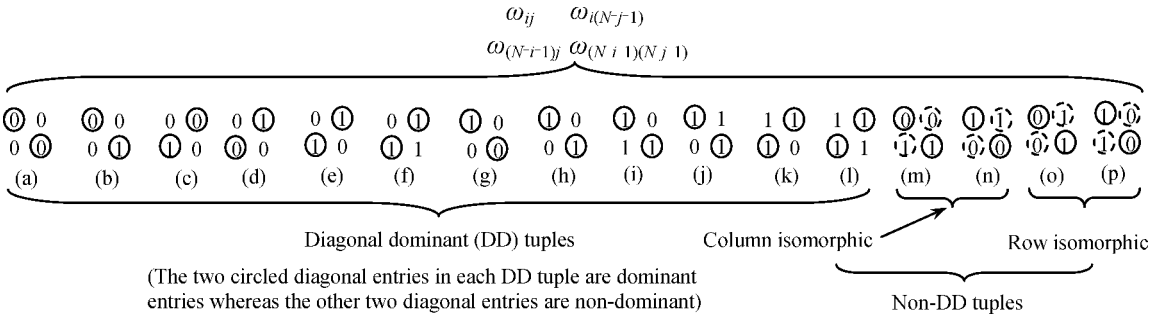

Fig. 11. Sixteen possible combinations of $\{\omega_{ij}, \omega_{i(N-j-1)}, \omega_{(N-i-1)j}, \omega_{(N-i-1)(N-j-1)}\}$.

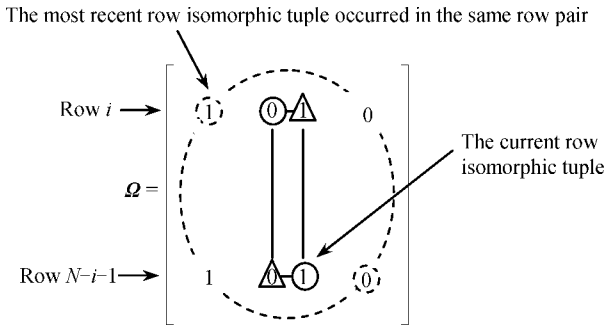

Fig. 12. Butterfly mechanism.

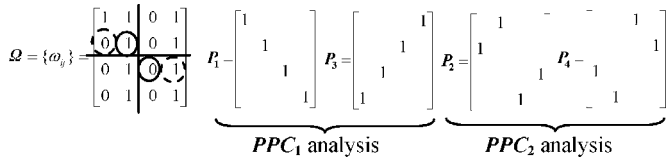

Fig. 13. Residue matrix scheduling based on Ω.

Case 3: If the 4-tuple is a column isomorphic tuple, we use the same mechanism as in Case 2 to set P_p and $P_{p+N/2}$, but we operate on the corresponding column pair instead of row pair.

A 4×4 example is given in Fig. 13. For simplicity, we only discuss the construction of P_1 and P_2. In examining PPC_1 (which covers entries (0, 0) and (1, 1) in zone A), $\{\omega_{00}, \omega_{33}\}$ are first identified as dominant entries in the 4-tuple $\{\omega_{00}, \omega_{03}, \omega_{30}, \omega_{33}\}$. Therefore, the two entries (0, 0) and (3, 3) are set to 1 in P_1. Then, we can see that $\{\omega_{11}, \omega_{12}, \omega_{21}, \omega_{22}\}$ is a row isomorphic tuple. Since no other row isomorphic tuples precede it so far, we simply set the two solid-circled entries (1, 1) and (2, 2) to 1 in P_1. In examining PPC_2 (which covers entries (0, 1) and (1, 0) in zone A), $\{\omega_{01}, \omega_{02}, \omega_{31}, \omega_{32}\}$ is also a row isomorphic tuple but it resides in a row pair different from that of $\{\omega_{11}, \omega_{12}, \omega_{21}, \omega_{22}\}$ (which occurred earlier in examining PPC_1). Since no other row isomorphic tuples precede it in the same row pair, we can set either pair of the diagonal entries to 1 in P_2. In Fig. 13, the two diagonal entries (0, 1) and (3, 2) are set to 1 in P_2. After that, the row isomorphic tuple $\{\omega_{10}, \omega_{13}, \omega_{20}, \omega_{23}\}$ is considered. Because another row isomorphic tuple $\{\omega_{11}, \omega_{12}, \omega_{21}, \omega_{22}\}$ precedes it in the same row pair, we need to set P_2 according to the butterfly mechanism. As a result, the two dash-circled entries (1, 0) and (2, 3) are set to 1 in P_2.

Obviously, the above process generates N non-overlapping configurations $P_1 \sim P_N$ to cover every entry of the residue matrix. This is because the entries covered by the PPCs in zone A and their corresponding images do not overlap each other. On the other hand, $P_1 \sim P_{N/2}$ usually cover more than half of $1s$ for *each and all* of the lines in Ω. This is because the dominant entries in each DD tuple are always covered by a configuration among $P_1 \sim P_{N/2}$. For non-DD tuples, the number of $1s$ covered by $P_1 \sim P_{N/2}$ and $P_{N/2+1} \sim P_N$ in each line of Ω are well balanced by the butterfly mechanism.

However, for a particular line in Ω, the number of 1 s covered by $P_1 \sim P_{N/2}$ (denoted by x) may be one less than that covered by $P_{N/2+1} \sim P_N$ (denoted by y). This is due to the odd number of isomorphic tuples in the corresponding line pair and the butterfly mechanism. Assume there are z isomorphic tuples in a particular row or column pair. For simplicity, we only consider these isomorphic tuples when counting x and y. If z is even, then x and y can be perfectly balanced by the butterfly mechanism, and thus $x = y$. On the other hand, if z is odd, then the butterfly mechanism leads to $|x - y| = 1$. Generally, if DD tuples are also taken into account, we have either $x \geq y$ or $x+1 = y$ for each and all of the lines in Ω. In other words, among all the $x + y$ $1s$ in each line of Ω, the $N/2$ configurations $P_1 \sim P_{N/2}$ always cover "more than half" and the other $N/2$ configurations $P_{N/2+1} \sim P_N$ always cover "less than half" of them (quotes are used here because the case $x + 1 = y$ is an exception).

Recall that ω_{ij} =1 means r_{ij} is an LER. Therefore, if $P_{N/2+1} \sim P_N$ cover some $1s$ in Ω, we can move the corresponding LERs from R to Q. This gives us R' and Q'. From Lemma 2, the maximum line sum of Q' will not exceed N. This is also true if $x + 1 = y$ for some lines in Ω. Note that k in Lemma 2 equals to $x + y$. If $x + 1 = y$, then k is odd and the roof function in Lemma 2 can handle this case. Consequently, all the remaining LERs in R' can be covered by $P_1 \sim P_{N/2}$ with a weight $\phi_n = T/N$ for each configuration. Besides, $P_1 \sim P_{N/2}$ may also cover some SERs in R'. For the remaining SERs in R' that are not covered by $P_1 \sim P_{N/2}$, they can be covered by $P_{N/2+1} \sim P_N$ with a reduced weight of $\phi_n = T/(2N)$. This leads to $S_{\text{schedule}} = 1.75$ as conjectured in (17), instead of $S_{\text{schedule}} = 2$ in DOUBLE.

The above discussion is based on DOUBLE algorithm, and DOUBLE is a special case of ADAPT at $N_S = 2N$. In general, we have the following theorem.

Theorem 1: To switch an admissible $N \times N$ traffic matrix $C(T)$ in N_S configurations (where $N^2 - 2N + 2 > N_S > N$

and $T > \delta N_S$), S_{schedule} in (19) is sufficient if N is even and T is a multiple of $N_S - N$.

$$S_{\text{schedule}} = 1 + \frac{3N}{4(N_S - N)}. \tag{19}$$

Proof: By using $T/(N_S - N)$ to divide each entry $c_{ij} \in \boldsymbol{C}(\boldsymbol{T})$, we convert $\boldsymbol{C}(\boldsymbol{T})$ into a weighted sum of $\boldsymbol{Q}$ and $\boldsymbol{R}$ as in (5). The maximum line sum of $\boldsymbol{Q}$ is at most $N_S - N$ (see (7) & (8)), and each entry $r_{ij} \in \boldsymbol{R}$ is not larger than $T/(N_S - N)$ [see (9)].

Define an arbitrary entry $r_{ij} \in \boldsymbol{R}$ as an LER if $r_{ij} > T/[2(N_S - N)]$. Similar to Lemma 2, we can prove that if a particular line of $\boldsymbol{R}$ contains k LERs, then the entries in the corresponding line of $\boldsymbol{Q}$ sum to at most $N_S - N - \lceil k/2 \rceil$. Accordingly, we can construct $\boldsymbol{R}'$ and $\boldsymbol{Q}'$ by moving a proper set of LERs from $\boldsymbol{R}$ to $\boldsymbol{Q}$, such that the maximum line sum of $\boldsymbol{Q}'$ is still not more than $N_S - N$, and each line of $\boldsymbol{R}'$ contains at most $N/2$ LERs. Then, $\boldsymbol{R}'$ can be covered by N non-overlapping configurations, with a weight $T/(N_S - N)$ for $N/2$ configurations and a reduced weight $T/[2(N_S - N)]$ for the other $N/2$ configurations. On the other hand, $\boldsymbol{Q}'$ can be covered by $N_S - N$ configurations with a weight $T/(N_S - N)$ for each, as discussed in Section II-A. Consequently, we have

$$\begin{aligned} S_{\text{schedule}} &= \frac{1}{T}\sum_{n=1}^{N_S} \phi_n \\ &= \frac{1}{T}\left[\frac{T}{(N_S - N)} \times (N_S - N) + \frac{T}{(N_S - N)} \right. \\ &\quad \left. \times \frac{N}{2} + \frac{T}{2(N_S - N)} \times \frac{N}{2}\right] \\ &= 1 + \frac{3N}{4(N_S - N)} \end{aligned}$$

■

Corollary 1: With S_{schedule} in (19), the overall speedup can be formulated by the speedup function $S = f(N_S)$ below:

$$S = f(N_S) = \frac{T\left(N_S - \frac{1}{4}N\right)}{(T - \delta N_S)(N_S - N)}. \tag{20}$$

Proof: Compared to S_{schedule} in (10), S_{schedule} in (19) is reduced. We can replace S_{schedule} in (2) by (19). This gives us a refined speedup function (still denoted by $S = f(N_S)$ for simplicity).

$$\begin{aligned} S &= S_{\text{reconfigure}} \times S_{\text{schedule}} \\ &= \frac{T}{T - \delta N_S}\left[1 + \frac{3N}{4(N_S - N)}\right] = \frac{T\left(N_S - \frac{1}{4}N\right)}{(T - \delta N_S)(N_S - N)} \end{aligned}$$

■

Corollary 2: To minimize the overall speedup, a schedule should consist of N_S configurations where N_S is formulated by

$$\begin{cases} N_S^{\text{real}} = \frac{1}{4}\left(N + \sqrt{\frac{12NT}{\delta} - 3N^2}\right) = \frac{N}{4}\left(1 + \sqrt{12\lambda^2 - 3}\right) \\ N_S = \left\lfloor N_S^{\text{real}} \right\rfloor, \quad \text{if } N^2 - 2N + 2 > N_S^{\text{real}} \geq N + 1 \\ N_S = N + 1, \quad \text{if } N + 1 > N_S^{\text{real}} > N. \end{cases} \tag{21}$$

Proof: Based on the refined speedup function $S = f(N_S)$ in (20), we can solve (12) for N_S to minimize the overall speedup S. The N_S value obtained is generally a real number (denoted by N_S^{real} in (21)). We can convert it into an integer using (21), where $\lambda = \sqrt{\frac{T}{\delta N}}$ is defined in (14) and $\lambda > 1$ must be true. Formula (21) also prevents $N_S = N$. This ensures that the traffic matrix decomposition can be carried out properly. ■

Based on the above analysis, SRF (Scheduling Residue First) algorithm is designed and is summarized in Fig. 14. SRF guarantees a better algorithmic efficiency than both DOUBLE and ADAPT, by taking residue matrix scheduling as a priori. Accordingly, SRF adopts a refined matrix decomposition which is slightly different from that given in Part A of Section II. Compared with ADAPT, SRF needs an extra $O(N^2)$ comparisons for residue matrix scheduling, but it still has the same time complexity of $O((\lambda - 1)N^2 \log N)$ as ADAPT.

Corollary 3: Let T be the traffic accumulation time and H be the execution time of the scheduling algorithm. The overall speedup S_{SRF} in (22) is sufficient to transmit an $N \times N$ admissible $\boldsymbol{C}(\boldsymbol{T})$ with a packet delay bound of $3T + H$ slots.

$$\begin{aligned} S_{\text{SRF}} &= \frac{2\lambda^2}{2\lambda^2 + 1 - \sqrt{12\lambda^2 - 3}} \\ \text{where } \lambda &= \sqrt{\frac{T}{\delta N}}. \end{aligned} \tag{22}$$

Proof: SRF algorithm in Fig. 14 can achieve performance guaranteed switching with a packet delay bound of $3T + H$ slots (see Fig. 2(a)). From (2), (3), (19) and (21), we can formulate the overall speedup required by SRF as below.

$$\begin{aligned} S_{\text{SRF}} &= S_{\text{reconfigure}} \times S_{\text{schedule}}|_{N_S = N_S^{\text{real}}} \\ &= \frac{T}{T - \delta N_S} S_{\text{schedule}}|_{N_S = N_S^{\text{real}}} \\ &= \frac{T}{T - \frac{\delta N}{4}\left(1 + \sqrt{12\lambda^2 - 3}\right)} \\ &\quad \times \left\{1 + \frac{3N}{4\left[\frac{N}{4}\left(1 + \sqrt{12\lambda^2 - 3}\right) - N\right]}\right\} \\ &= \frac{2\lambda^2}{2\lambda^2 + 1 - \sqrt{12\lambda^2 - 3}}. \end{aligned}$$

■

So far we have discussed SRF based on an even switch size N. SRF can be extended to odd N with minor modifications. Appendix II gives an example with a 9×9 reference matrix $\boldsymbol{\Omega}$.

V. Discussion

Fig. 15 compares the overall speedup of the three algorithms: S_{DOUBLE} in (16), S_{ADAPT} in (15), and S_{SRF} in (22). For simplicity, we focus on an optical switch with a given switch size N and a given reconfiguration overhead δ. From (14), varying λ now corresponds to varying the traffic accumulation time T, and thus the packet delay bound $3T + H$ (assume the algorithm's execution time H is constant). Accordingly, Fig. 15 shows the tradeoff between the overall speedup and the packet delay bound for the three algorithms. Since DOUBLE is a special case of ADAPT at $N_S = 2N$, in Fig. 15 we can see that

SRF ALGORITHM

Input:

An $N \times N$ traffic matrix $\boldsymbol{C(T)}=\{c_{ij}\}$ with maximum line sum no more than T, and the reconfiguration overhead δ.

Output:

At most N_S configurations $\boldsymbol{P}_1, \ldots, \boldsymbol{P}_{N_S}$ and the corresponding weights $\phi_1, \ldots, \phi_{N_S}$.

Step 1: Divide the entries in C(T):

Calculate N_S by (21). Use $\lceil T/(N_S-N) \rceil$ to divide each entry $c_{ij} \in \boldsymbol{C(T)}$ and separate $\boldsymbol{C(T)}$ into a quotient matrix $\boldsymbol{Q}=\{q_{ij}\}$ and a residue matrix $\boldsymbol{R}=\{r_{ij}\}$ as in (5).

Step 2: Schedule the residue matrix:

a) Define a reference matrix $\boldsymbol{\Omega}=\{\omega_{ij}\}$ based on $\boldsymbol{R}$, such that $\omega_{ij}=1$ if r_{ij} is an LER ($> \lceil T/(2 \times (N_S - N)) \rceil$) and $\omega_{ij}=0$ otherwise. Define a set of predefined partial configurations $\boldsymbol{PPC}_p$ ($N/2 \geq p \geq 1$) to cover every entry $\omega_{ij} \in \{\omega_{ij} \mid N/2>i, j \geq 0\}$, where the value of p is calculated from i and j according to (18). Initialize $\boldsymbol{P}_1 \sim \boldsymbol{P}_N$ to all-0 matrices. Set $1 \rightarrow p$.

b) Pick up an entry $\omega_{ij} \in \{\omega_{ij} \mid N/2>i, j \geq 0\}$ covered by $\boldsymbol{PPC}_p$, and construct a 4-tuple $\{\omega_{ij}, \omega_{i(N-j-1)}, \omega_{(N-i-1)j}, \omega_{(N-i-1)(N-j-1)}\}$.

- If the 4-tuple is a DD tuple in Figs. 11(a)–11(l), pick up the two circled dominant entries and set the corresponding entries to 1 in $\boldsymbol{P}_p$. Set the other two non-dominant entries to 1 in $\boldsymbol{P}_{p+N/2}$.
- If the 4-tuple is a row isomorphic tuple in Figs. 11(o)–11(p), check whether there exists another row isomorphic tuple which most recently occurred in the same row pair $\{i, N-i-1\}$. If no, choose any pair of the diagonal entries of the tuple and set the corresponding entries to 1 in $\boldsymbol{P}_p$. Set the other two diagonal entries to 1 in $\boldsymbol{P}_{p+N/2}$. Otherwise if the most recent row isomorphic tuple occurred in examining $\boldsymbol{PPC}_t$ ($p>t$), invoke the butterfly mechanism to set the corresponding entries in $\boldsymbol{P}_p$, such that $\boldsymbol{P}_p$ and $\boldsymbol{P}_t$ can cover the 1*s* in the row pair $\{i, N-i-1\}$ of $\boldsymbol{\Omega}$ in an alternating manner. If a pair of diagonal entries are set to 1 in $\boldsymbol{P}_p$, then set the other pair of diagonal entries to 1 in $\boldsymbol{P}_{p+N/2}$.
- If the 4-tuple is a column isomorphic tuple in Figs. 11(m)–11(n), use the same mechanism as in the row isomorphic case to set $\boldsymbol{P}_p$ and $\boldsymbol{P}_{p+N/2}$ but operate on the column pair $\{j, N-j-1\}$ instead of row pair $\{i, N-i-1\}$.

c) Repeat *Step 2b)* until all the entries $\omega_{ij} \in \{\omega_{ij} \mid N/2>i, j \geq 0\}$ covered by $\boldsymbol{PPC}_p$ are considered and the two configurations $\boldsymbol{P}_p$ and $\boldsymbol{P}_{p+N/2}$ are obtained.

d) Set $p+1 \rightarrow p$. If $p \leq N/2$, go to *Step 2b)*. Otherwise continue.

e) Set $\phi_1 \sim \phi_{N/2}$ to $\lceil T/(N_S-N) \rceil$ and $\phi_{N/2+1} \sim \phi_N$ to $\lceil T/(2 \times (N_S - N)) \rceil$. If some 1*s* in $\boldsymbol{\Omega}$ are covered by $\boldsymbol{P}_{N/2+1} \sim \boldsymbol{P}_N$, increase the corresponding entries in $\boldsymbol{Q}$ by 1. Denote the updated $\boldsymbol{Q}$ by $\boldsymbol{Q}'$.

Step 3: Schedule the quotient matrix:

a) Construct a bipartite multigraph $\boldsymbol{G}$ from $\boldsymbol{Q}'$. Rows and columns of $\boldsymbol{Q}'$ translate to two sets of vertices $\boldsymbol{A}$ and $\boldsymbol{B}$ in $\boldsymbol{G}$, and each entry $q'_{ij} \in \boldsymbol{Q}'$ translates to q'_{ij} edges connecting vertices $i \in \boldsymbol{A}$ and $j \in \boldsymbol{B}$. Find a minimal edge-coloring of $\boldsymbol{G}$ to get at most N_S-N colors, such that the edges incident on the same vertex have different colors. Set $N+1 \rightarrow n$.

b) For a specific color in the edge-coloring of $\boldsymbol{G}$, construct a configuration $\boldsymbol{P}_n$ from the edges in that color by setting the corresponding entries to 1 in $\boldsymbol{P}_n$ (all other entries in $\boldsymbol{P}_n$ are set to 0). Set $\phi_n = \lceil T/(N_S-N) \rceil$ and $n+1 \rightarrow n$. Repeat *step 3b)* for each color in the edge-coloring of $\boldsymbol{G}$.

Fig. 14. SRF algorithm for even switch size N.

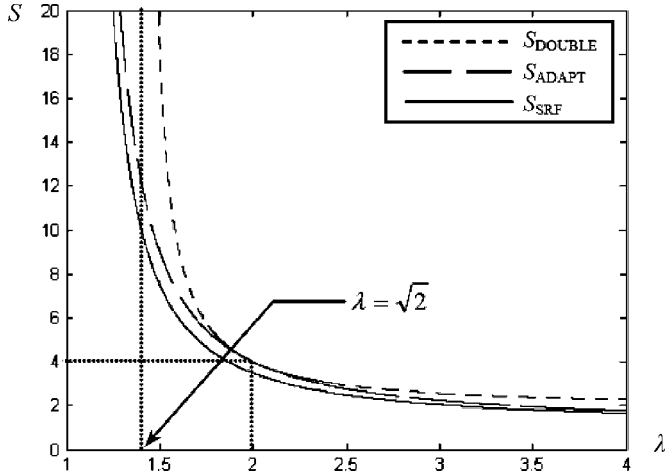

Fig. 15. Speedup comparison of DOUBLE, ADAPT and SRF.

$S_{\text{DOUBLE}} = S_{\text{ADAPT}} = 4$ at $\lambda = 2$. For other λ values, we have $S_{\text{ADAPT}} < S_{\text{DOUBLE}}$. At the same time, $S_{\text{SRF}} < S_{\text{ADAPT}}$ is always true. For example, at $\lambda = 1.5$, we have $S_{\text{DOUBLE}} = 18$, $S_{\text{ADAPT}} = 9$, and $S_{\text{SRF}} = 7.49$. SRF outperforms ADAPT and DOUBLE by 16.78% and 58.39% respectively. We can see that ADAPT and SRF dramatically cut down the required speedup for delay-sensitive traffic with small λ values. As λ increases, the speedup gap between DOUBLE and ADAPT/SRF diminishes.

ADAPT and SRF can also be used to minimize the required traffic accumulation time and the packet delay bound for a given speedup requirement. Assume we can only tolerate a speedup up to 3. Consider a 64×64 fast optical switch with a reconfiguration overhead of 100 ns [5]. Let the duration of a time slot be 50 ns (64 bytes at 10 Gbps). We have $\delta = 2$ slots. Further let $T_{\text{DOUBLE}}, T_{\text{ADAPT}}$ and T_{SRF} denote the traffic accumulation times required by DOUBLE, ADAPT and SRF under the given speedup $S = 3$, and $\lambda_{\text{DOUBLE}}, \lambda_{\text{ADAPT}}, \lambda_{\text{SRF}}$ denote their corresponding λ values. From (16), (15) and (22), we get $\lambda_{\text{DOUBLE}} = 2.45, \lambda_{\text{ADAPT}} = 2.37$ and $\lambda_{\text{SRF}} = 2.19$. From (14), we get $T_{\text{DOUBLE}} = 768$ slots (38.4 μs), $T_{\text{ADAPT}} = 719$ slots (35.95 μs) and $T_{\text{SRF}} = 614$ slots (30.7 μs). We can see that $T_{\text{DOUBLE}} > T_{\text{ADAPT}} > T_{\text{SRF}}$. Note that both speedup and packet delay bound take realistic values in this example (e.g., the packet delay bound for SRF is $3T_{\text{SRF}}+H = 92.1\ \mu\text{s}+H$, where H is the algorithm's execution time). Compared to DOUBLE, the required traffic accumulation time is cut down by 6.38% in ADAPT and by 20.05% in SRF.

From our discussion in Section I, $T > \delta N_S \geq \delta N$ (and thus $\lambda = \sqrt{T/\delta N} > 1$) must be ensured in any feasible schedule. Since DOUBLE enforces $N_S = 2N$, its traffic accumulation time T must be larger than $2\delta N$ ($T > \delta N_S = 2\delta N$). Therefore, DOUBLE is only feasible for $\lambda > \sqrt{2}$, as shown in Fig. 15. In comparison, both ADAPT and SRF are feasible for $\lambda > 1$. This is because $\lambda > 1$ (or $T > \delta N$) always ensures $T > \delta N_S$ in ADAPT and SRF. In ADAPT, the number of configurations in a schedule is optimized to be $N_S = \lfloor \lambda N \rfloor$, and thus

$$\delta N_S = \delta \lfloor \lambda N \rfloor = \delta \lfloor \sqrt{\frac{TN}{\delta}} \rfloor \leq \sqrt{T \times (\delta N)} < \sqrt{T^2} = T. \tag{23}$$

Similarly, with $\lambda > 1$ and N_S in (21) for SRF, $T > \delta N_S$ is ensured in SRF because

$$\delta N_S = \frac{\delta N}{4}(1 + \sqrt{12\lambda^2 - 3}) < \lambda^2 \delta N = T. \tag{24}$$

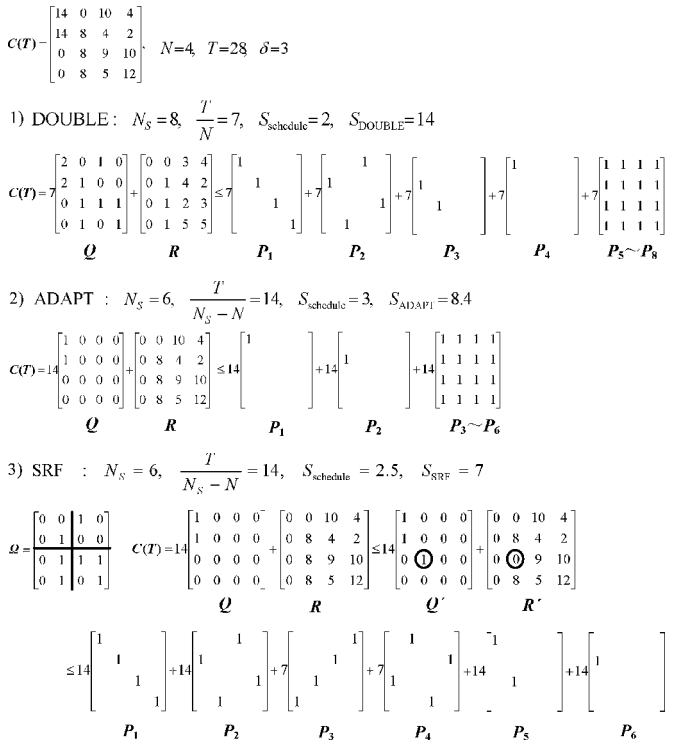

Fig. 16. Example of DOUBLE, ADAPT and SRF execution.

On the other hand, if we consider the feasibility region in terms of speedup S, then DOUBLE is only feasible for $S > 2$ (i.e., $S = S_{\text{reconfigure}} \times S_{\text{schedule}} = 2 \times S_{\text{reconfigure}} > 2$). In comparison, both ADAPT and SRF are feasible for $S > 1$.

Fig. 16 gives a simple example to compare the execution of DOUBLE, ADAPT and SRF. Although DOUBLE produces the smallest S_{schedule} (but the largest N_S), it requires the largest overall speedup of $S_{\text{DOUBLE}} = 14$, whereas ADAPT requires $S_{\text{ADAPT}} = 8.4$ and SRF only requires $S_{\text{SRF}} = 7$.

In this paper, we presented a generic approach to decompose a traffic matrix into an arbitrary number of N_S ($N^2-2N+2 > N_S > N$) configurations/permutations. For general applications in other networks, it may or may not have a constraint of reconfiguration overhead. If such a constraint exists, the corresponding system can be modeled as a constrained switch [6] (e.g., SS/TDMA [16], [17] and TWIN [18]), and ADAPT/SRF algorithms can be directly applied. If such a constraint does not exist, our generic matrix decomposition can still be applied. For example, computing a schedule for an electronic switch (which has negligible reconfiguration overhead) is difficult as switch size increases [6], [14], [24]. This is because the large number of $O(N^2)$ configurations in Birkhoff-von Neumann decomposition limits the scalability of the switch [14]. In this case, our generic matrix decomposition can be applied to generate a schedule with less number of configurations, at a cost of speedup.

It should be noted that $N_S > N$ is ensured in ADAPT and SRF because $\lambda > 1$. As mentioned in Section I, $N_S = N$ corresponds to minimum-delay scheduling, and it can be handled by QLEF algorithm [8], [9]. Also note that in this paper we focused on performance guaranteed switching with worst-case analysis. Average performance analysis is out of the scope of this paper, but can be handled by two existing greedy algorithms, GOPAL [22] and LIST [6], [23].

VI. Conclusion

The progress of optical switching technologies has enabled the implementation of high-speed packet switches with optical fabrics. Compared with conventional electronic switches, the reconfiguration overhead issue of optical switches must be properly addressed.

In this paper, we focused on designing scheduling algorithms for optical switches that provide performance guaranteed switching (100% throughput with bounded packet delay). We first designed a generic approach to decompose a traffic matrix into the sum of N_S weighted switch configurations (for $N^2-2N+2 > N_S > N$ where N is the switch size). We then took the reconfiguration overhead constraint of optical switches into account, and formulated a speedup function to capture the relationship between the speedup and the number of configurations N_S in a schedule. By minimizing the speedup function, an efficient scheduling algorithm ADAPT was designed to minimize the overall speedup for a given packet delay bound. Based on the observation that some packets can be moved from the residue matrix to the quotient matrix and thus the bandwidth utilization of the configurations can be improved, another algorithm SRF (Scheduling Residue First) was designed to achieve an even lower speedup. Both ADAPT and SRF algorithms can automatically adjust the schedule according to different switch parameters, and find a proper N_S value to minimize speedup. We also showed that ADAPT and SRF can be used to minimize packet delay bound under a given speedup requirement.

Appendix I
Correctness Proof of Lemma 2

Lemma 2: In DOUBLE, if a line (row i or column j) of $\boldsymbol{R}$ contains k LERs, then in $\boldsymbol{Q}$ we have

$$\sum_{j=0}^{N-1} q_{ij} \le N - \left\lceil \frac{k}{2} \right\rceil \text{ for row } i$$

or

$$\sum_{i=0}^{N-1} q_{ij} \le N - \left\lceil \frac{k}{2} \right\rceil \text{ for column } j.$$

Proof: After the entries in $\boldsymbol{C}(\boldsymbol{T})$ are divided by T/N, we have

$$\boldsymbol{C}(\boldsymbol{T}) = \frac{T}{N}\boldsymbol{Q} + \boldsymbol{R}$$
$$\text{or } c_{ij} = \frac{T}{N} q_{ij} + r_{ij}.$$

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

$$\sum_{j=0}^{N-1} c_{ij} = \frac{T}{N} \sum_{j=0}^{N-1} q_{ij} + \sum_{j=0}^{N-1} r_{ij} \le T$$

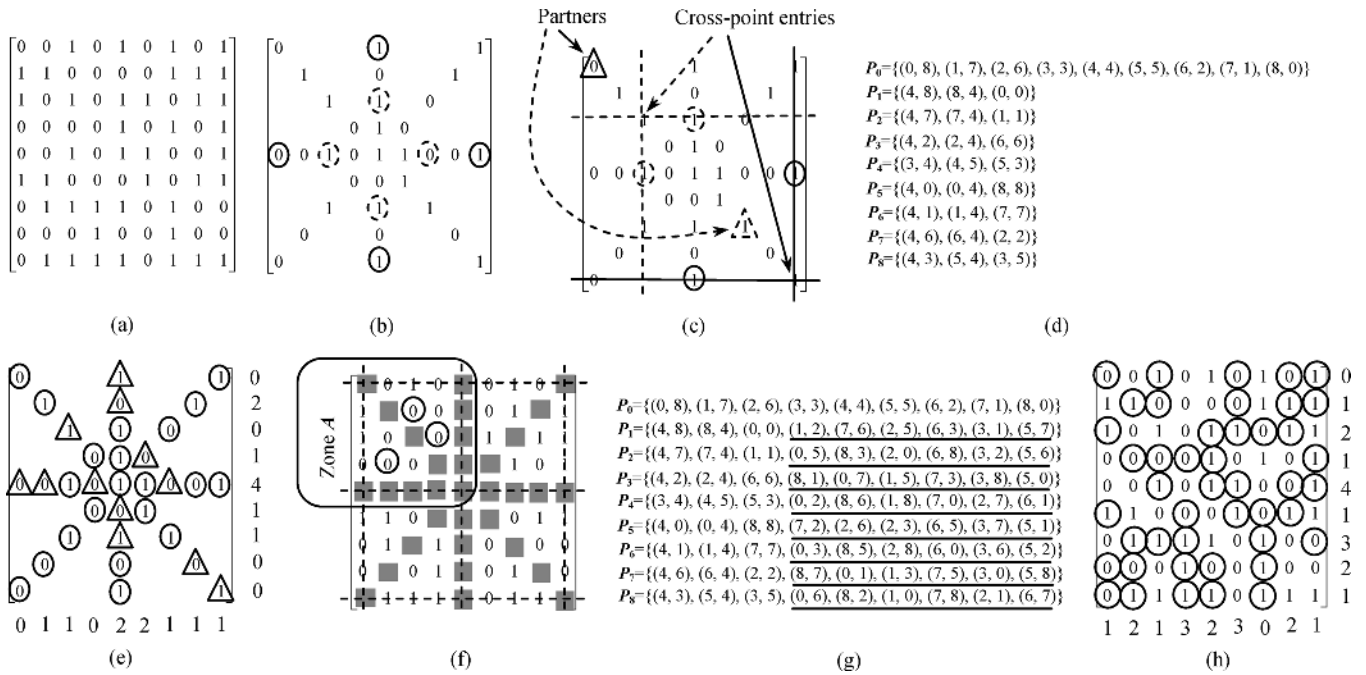

Fig. 17. SRF implementation for $N = 9$.

we have

$$\sum_{j=0}^{N-1} q_{ij} \le \frac{T - \sum_{j=0}^{N-1} r_{ij}}{\frac{T}{N}} < \frac{T - \frac{T}{2N} \times k}{\frac{T}{N}} = N - \frac{k}{2}.$$

Since $\sum_{j=0}^{N-1} q_{ij}$ is an integer, we then have

$$\sum_{j=0}^{N-1} q_{ij} \le N - \left\lceil \frac{k}{2} \right\rceil.$$

■

APPENDIX II
SRF EXTENSION FOR ODD SWITCH SIZE

To show how SRF can be extended to odd switch size N, we consider the 9×9 reference matrix $\boldsymbol{\Omega}$ in Fig. 17(a) (i.e., $N = 9$). We call row/column $4\,(= \lfloor N/2 \rfloor = \lfloor 9/2 \rfloor)$ the *middle row/column*. The N non-overlapping configurations devoted to $\boldsymbol{R}$ can be constructed with the following two steps.

Step 1: We first consider the entries in the middle row and the middle column of $\boldsymbol{\Omega}$, and the entries in the two matrix diagonals, as shown in Fig. 17(b). An extra 4-tuple (denoted by *x-tuple*) is defined as $\{\omega_{4t}, \omega_{t4}, \omega_{4(8-t)}, \omega_{(8-t)4}\}$ for $3 \ge t \ge 0$. For example, the solid-circled and dash-circled entries in Fig. 17(b) form two x-tuples, with $t = 0$ and $t = 2$ respectively.

In each x-tuple, a pair of entries resides in the middle row (row 4) and the other pair resides in the middle column (column 4). For each pair, if one entry is no less than the other entry, then we define it as a *dominant entry* and the other entry is non-dominant. For example, we can take entries (4, 8) and (8, 4) as two dominant entries in x-tuple $\{\omega_{40}, \omega_{04}, \omega_{48}, \omega_{84}\}$, whereas entries (4, 0) and (0, 4) are non-dominant. Then, based on the two dominant entries (4, 8) and (8, 4), we draw two solid lines as in Fig. 17(c) and find the entry at the cross-point of the two lines, which is defined as a *cross-point entry*. The diagonal image of the cross-point entry, as shown by the solid-triangle in Fig. 17(c), is defined as the *partner* of the two dominant entries (4, 8) and (8, 4). Similarly, for dominant entries (4, 2) and (2, 4) in x-tuple $\{\omega_{42}, \omega_{24}, \omega_{46}, \omega_{64}\}$, entry (2, 2) is the cross-point entry, and (6, 6) is the partner of the two dominant entries, as shown by the dashed part in Fig. 17(c).

For each possible x-tuple $\{\omega_{4t}, \omega_{t4}, \omega_{4(8-t)}, \omega_{(8-t)4}\}$ $(3 \ge t \ge 0)$, the two dominant entries and their partner are set to 1 in configuration $\boldsymbol{P}_{t+1}$. At the same time, the two non-dominant entries and the cross-point entry are set to 1 in $\boldsymbol{P}_{t+\lceil N/2 \rceil} = \boldsymbol{P}_{t+5}$. $\boldsymbol{P}_1 \sim \boldsymbol{P}_8$ in Fig. 17(d) give the result of this operation, where we use a set for each configuration to record the three entries that are set to 1. In Fig. 17(b), we remove all the entries recorded in $\boldsymbol{P}_1 \sim \boldsymbol{P}_8$ of Fig. 17(d), the remaining 9 entries are set to 1 in $\boldsymbol{P}_0$, as shown in Fig. 17(d).

At the end of Step 1, we can see that all the entries in Fig. 17(b) have been covered by the *partial* configurations $\boldsymbol{P}_0 \sim \boldsymbol{P}_8$ in Fig. 17(d). For each matrix line (row or column), let a be the number of $1s$ covered by $\boldsymbol{P}_0 \sim \boldsymbol{P}_4$, and b be the number of $1s$ covered by $\boldsymbol{P}_5 \sim \boldsymbol{P}_8$. In Fig. 17(e), we use circles and triangles to indicate the entries covered by $\boldsymbol{P}_0 \sim \boldsymbol{P}_4$ and $\boldsymbol{P}_5 \sim \boldsymbol{P}_8$, respectively. The number next to each matrix line in Fig. 17(e) gives the value of $a - b$ for this line. In the example, we can see that $a - b \ge 0$ for every line. In general, it is easy to prove that $a - b \ge -1$ is always true. It means that the partial configurations $\boldsymbol{P}_0 \sim \boldsymbol{P}_4$ in Fig. 17(d) usually cover more $1s$ than $\boldsymbol{P}_5 \sim \boldsymbol{P}_8$ for each line in Figs. 17(b) & (e). If this is not the case, then $\boldsymbol{P}_5 \sim \boldsymbol{P}_8$ can cover at most one more 1 for some lines. It is also not difficult to prove that, for a particular line pair (i.e., row pair $\{i, N-i-1\}$ or column pair $\{j, N-j-1\}$), at most one line (but not both) can have $a - b = -1$.

Step 2: In Fig. 17(f), we *shade* all the entries covered so far using filled rectangles. Then, we pick up a partial configuration $\boldsymbol{P}_{t+1}$ $(3 \geq t \geq 0)$ from Fig. 17(d) and use dash lines to *shadow* the rows and columns of its three entries. Assume $\boldsymbol{P}_1$ is chosen (i.e., $t = 0$). Fig. 17(f) shows the result after the shadowing operation. We can use maximum-size matching [25] to determine a $\boldsymbol{PPC}$ in zone $\boldsymbol{A}$, such that it contains the maximum number of not-yet-shadowed and not-yet-covered entries, each in a distinct row and column in zone $\boldsymbol{A}$. As shown in Fig. 17(f), the $\boldsymbol{PPC}$ found contains three circled entries. For each circled entry, we find its line and diagonal images to form a 4-tuple $\{\omega_{ij}, \omega_{i(N-j-1)}, \omega_{(N-i-1)j}, \omega_{(N-i-1)(N-j-1)}\}$. If it is a DD-tuple, the two dominant entries are set to 1 in $\boldsymbol{P}_{t+1}$, and the two non-dominant entries are set to 1 in $\boldsymbol{P}_{t+\lceil N/2 \rceil} = \boldsymbol{P}_{t+5}$. If it is an isomorphic tuple, we use the butterfly mechanism to set the entries in $\boldsymbol{P}_{t+1}$ and $\boldsymbol{P}_{t+5}$. If an entry is set to 1 in $\boldsymbol{P}_{t+1}$ or $\boldsymbol{P}_{t+5}$, we add it to the corresponding partial configuration in Fig. 17(g) (see the underlined entries), and shade this entry by a filled rectangle as we have done in Fig. 17(f) (because it has been covered).

It is important to note that we need to slightly modify the butterfly mechanism for odd switch size N. Recall that for even N, if an isomorphic tuple is the first one in a particular line pair, we can set the corresponding configurations according to either pair of its diagonal entries. However, for odd N, we have an initial state as shown in Fig. 17(e), where $a - b \neq 0$ for some matrix lines. It means that the number of 1 s covered by the partial configurations $\boldsymbol{P}_0 \sim \boldsymbol{P}_4$ and $\boldsymbol{P}_5 \sim \boldsymbol{P}_8$ in Fig. 17(d) may not be perfectly balanced for every line at the beginning of Step 2. If $a - b = -1$ for a particular line, we can treat it as if another preceding isomorphic tuple already exists in the corresponding line pair. Therefore, for the first isomorphic tuple in this line pair, we should take this initial state into account, and set the entries in the corresponding configurations to ensure $a - b \geq -1$ for both lines. After that, we can use the same butterfly mechanism as in Fig. 12 for all subsequent isomorphic tuples in this line pair.

At the end of Step 2, we remove all the dash lines in Fig. 17(f). Then, Step 2 is repeated for another $\boldsymbol{P}_{t+1}$ $(3 \geq t \geq 0)$ in Fig. 17(d), until all the configurations $\boldsymbol{P}_0 \sim \boldsymbol{P}_8$ are obtained, as shown in Fig. 17(g).

The entries covered by $\boldsymbol{P}_0 \sim \boldsymbol{P}_4$ [in Fig. 17(g)] are circled in Fig. 17(h). The number next to each matrix line in Fig. 17(h) gives the difference on the number of $1s$ covered by $\boldsymbol{P}_0 \sim \boldsymbol{P}_4$ and $\boldsymbol{P}_5 \sim \boldsymbol{P}_8$ for that line. Note that $\boldsymbol{P}_0 \sim \boldsymbol{P}_4$ cover more $1s$ than $\boldsymbol{P}_5 \sim \boldsymbol{P}_8$ for each line. Therefore, for those $1s$ covered by $\boldsymbol{P}_5 \sim \boldsymbol{P}_8$, we can move the corresponding LERs from $\boldsymbol{R}$ to $\boldsymbol{Q}$. Then, $\boldsymbol{P}_5 \sim \boldsymbol{P}_8$ can be weighted by a reduced weight of $T/[2(N_S - N)]$. In this 9×9 example, the set of configurations $\boldsymbol{P}_0 \sim \boldsymbol{P}_4$ contains one more configuration than $\boldsymbol{P}_5 \sim \boldsymbol{P}_8$. For large switch size N, this difference is trivial.

It is not difficult to extend the above approach to other odd switch size N. Note that maximum-size matching [25] in Step 2 is only used to find a predefined $\boldsymbol{PPC}$ pattern. It is not really required for online execution. Therefore the time complexity of SRF algorithm is still $O((\lambda - 1)N^2 \log N)$ for odd N.

ACKNOWLEDGMENT

The authors would like to thank the editor and the anonymous reviewers for their valuable comments and great contributions helping us to improve the presentation of this paper.

REFERENCES

[1] J. E. Fouquet *et al.*, "A compact, scalable cross-connect switch using total internal reflection due to thermally-generated bubbles," in *IEEE LEOS Annual Meeting*, Dec. 1998, pp. 169–170.
[2] L. Y. Lin, "Micromachined free-space matrix switches with sub-millisecond switching time for large-scale optical crossconnect," in *OFC'98 Tech. Dig.*, Feb. 1998, pp. 147–148.
[3] O. B. Spahn, C. Sullivan, J. Burkhart, C. Tigges, and E. Garcia, "GaAs-based microelectromechanical waveguide switch," in *Proc. 2000 IEEE/LEOS Intl. Conf. Optical MEMS*, Aug. 2000, pp. 41–42.
[4] A. J. Agranat, "Electroholographic wavelength selective crossconnect," in *1999 Dig. LEOS Summer Topical Meetings*, Jul. 1999, pp. 61–62.
[5] K. Kar, D. Stiliadis, T. V. Lakshman, and L. Tassiulas, "Scheduling algorithms for optical packet fabrics," *IEEE J. Sel. Areas Commun.*, vol. 21, no. 7, pp. 1143–1155, Sep. 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] X. Li and M. Hamdi, "On scheduling optical packet switches with reconfiguration delay," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 7, pp. 1156–1164, Sept. 2003.
[8] B. Wu and K. L. Yeung, "Traffic scheduling in non-blocking optical packet switches with minimum delay," in *Proc. IEEE GLOBECOM'05*, Dec. 2005, vol. 4, pp. 2041–2045.
[9] B. Wu and K. L. Yeung, "Minimum delay scheduling in scalable hybrid electronic/optical packet switches," in *Proc. IEEE GLOBECOM'06*, Dec. 2006.
[10] B. Wu, X. Wang, and K. L. Yeung, "Can we schedule traffic more efficiently in optical packet switches?," in *Proc. IEEE HPSR'06*, Jun. 2006, pp. 181–186.
[11] S. T. Chuang, A. Goel, N. McKeown, and B. Prabhakar, "Matching output queuing with a combined input output queued switch," in *Proc. IEEE INFOCOM'99*, Mar. 1999, vol. 3, pp. 1169–1178.
[12] G. Birkhoff, "Tres observaciones sobre el algebra lineal," *Univ. Nac. Tucumán Rev. Ser. A*, vol. 5, pp. 147–151, 1946.
[13] J. von Neumann, "A certain zero-sum two-person game equivalent to the optimal assignment problem," in *Contributions to the Theory of Games*. Princeton, NJ: Princeton Univ. Press, 1953, vol. 2, pp. 5–12.
[14] C. S. Chang, W. J. Chen, and H. Y. Huang, "Birkhoff-von Neumann input buffered crossbar switches," in *Proc. IEEE INFOCOM'00*, Mar. 2000, vol. 3, pp. 1614–1623.
[15] J. Li and N. Ansari, "Enhanced Birkhoff-von Neumann decomposition algorithm for input queued switches," *IEE Proc.-Commun.*, vol. 148, no. 6, pp. 339–342, Dec. 2001.
[16] T. Inukai, "An efficient SS/TDMA time slot assignment algorithm," *IEEE Trans. Commun.*, vol. COM-27, no. 10, pp. 1449–1455, 1979.
[17] Y. Ito, Y. Urano, T. Muratani, and M. Yamaguchi, "Analysis of a switch matrix for an SS/TDMA system," *Proc. IEEE*, vol. 65, no. 3, pp. 411–419, Mar. 1977.
[18] K. Ross, N. Bambos, K. Kumaran, I. Saniee, and I. Widjaja, "Scheduling bursts in time-domain wavelength interleaved networks," *IEEE J. Sel. Areas Commun.*, vol. 21, no. 9, pp. 1441–1451, Nov. 2003.
[19] H. Liu, P. Wan, C.-W. Yi, X. H. Jia, S. Makki, and N. Pissinou, "Maximal lifetime scheduling in sensor surveillance networks," in *Proc. IEEE INFOCOM'05*, Mar. 2005, vol. 4, pp. 2482–2491.
[20] R. Cole and J. Hopcroft, "On edge coloring bipartite graphs," *SIAM J. Comput.*, vol. 11, pp. 540–546, Aug. 1982.
[21] R. Diestel, *Graph Theory*, 2nd ed. New York: Springer-Verlag, 2000.
[22] S. Gopal and C. K. Wong, "Minimizing the number of switchings in a SS/TDMA system," *IEEE Trans. Commun.*, vol. COM-33, pp. 497–501, Jun. 1985.
[23] R. L. Graham, "Bounds on multiprocessing timing anomalies," *SIAM J. Appl. Mathemat.*, vol. 17, no. 2, pp. 416–429, Mar. 1969.
[24] E. Altman, Z. Liu, and R. Righter, "Scheduling of an input-queued switch to achieve maximal throughput," *Probabil. Eng. Inform. Sci.*, vol. 14, pp. 327–334, 2000.
[25] J. E. Hopcroft and R. M. Karp, "An $n^{5/2}$ algorithm for maximum matching in bipartite graphs," *Soc. Ind. Appl. Math. J. Comput.*, vol. 2, pp. 225–231, 1973.

Bin Wu (S'04–M'07) received the B.Eng. degree in electrical engineering from Zhe Jiang University, Hangzhou, China, in 1993, the M.Eng. degree in communication and information engineering from the University of Electronic Science and Technology of China, Chengdu, China, in 1996, and the Ph.D. degree in electrical and electronic engineering from The University of Hong Kong, Pokfulam, Hong Kong, in 2007.

From 1996 to 2001, he served as the department manager of TI-Huawei DSP co-lab at Huawei Tech. Co. Ltd., Shenzhen, China. He is currently a postdoctoral fellow at the University of Waterloo, Waterloo, ON, Canada, where he is involved in optical and wireless networking research.

Kwan L. Yeung (S'93–M'95–SM'99) received the B.Eng. and Ph.D. degrees in information engineering from The Chinese University of Hong Kong, Shatin, New Territories, Hong Kong, in 1992 and 1995, respectively.

He joined the Department of Electrical and Electronic Engineering, The University of Hong Kong, Pokfulam, Hong Kong, in July 2000, where he is currently an Associate Professor. His research interests include next-generation Internet, packet switch/router design, all-optical networks and wireless data networks.

Mounir Hamdi (S'89–M'90) received the B.S. degree in computer engineering (with distinction) from the University of Louisiana in 1985, and the M.S. and the Ph.D. degrees in electrical engineering from the University of Pittsburgh in 1987 and 1991, respectively.

He has been a faculty member in the Department of Computer Science at the Hong Kong University of Science and Technology since 1991, where he is now Professor of computer science and was the Director of the Computer Engineering Program which has around 350 undergraduate students for the past eight years. He is also Manager of the Computer Engineering Lab.

Dr. Hamdi is or has been on the Editorial Boards of IEEE TRANSACTIONS ON COMMUNICATIONS, *IEEE Communication Magazine*, *Computer Networks*, *Wireless Communications and Mobile Computing*, and *Parallel Computing*. He was a guest editor of *IEEE Communications Magazine*, guest editor-in-chief of two special issues of IEEE JOURNAL ON SELECTED AREAS OF COMMUNICATIONS, and a guest editor-in-chief of *Optical Networks Magazine*. He has been on the program committees of more than 80 international conferences and workshops, and has chaired more than five international conferences and workshops including the IEEE HPSR Workshop, IEEE GLOBECOM/ICC Optical networking workshop, the IEEE ICC High-speed Access Workshop, and the IEEE IPPS HiNets Workshop. He was the Chair of IEEE Communications Society Technical Committee on Transmissions, Access and Optical Systems, and Vice-Chair of the Optical Networking Technical Committee, as well as a member of the ComSoc technical activities council. He received the best paper award at the International Conference on Information and Networking in 1998. He also supervised the best Ph.D. paper award among all universities in Hong Kong in 2003. He received the best 10 lecturers award (through university-wide student voting for all university faculty held once a year), the distinguished engineering teaching appreciation award from the Hong Kong University of Science and Technology, and various grants targeted towards the improvement of teaching methodologies, delivery and technology. He is a member of the ACM.

Xin Li received the B.Eng. degree in computer science (with distinction) from the Xi'an Jiao Tong University, China, in 2000, and the Ph.D. degree in computer science from the Hong Kong University of Science and Technology in 2005.

Since fall 2005, she has been a Visiting Assistant Professor in the Department of Computer Science at the Hong Kong University of Science and Technology, where she is involved in major research projects as well as teaching undergraduate courses.