Guaranteed Scheduling for Switches with Configuration Overhead

Brian Towles and William J. Dally

***Abstract*— In this paper we present three algorithms that provide performance guarantees for scheduling switches, such as optical switches, with configuration overhead. Each algorithm emulates an unconstrained (zero overhead) switch by accumulating a batch of configuration requests and generating a corresponding schedule for a constrained switch. Speedup is required both to cover the configuration overhead of the switch and to compensate for empty slots left by the scheduling algorithm. Scheduling algorithms are characterized by the number of configurations, N_s, they require to *cover* a batch of requests, and the speedup required to compensate for empty slots, S_{min}. We show that a well known exact matching algorithm, EXACT, leaves no empty slots (i.e. $S_{\text{min}} = 1$), but requires $N_s \approx N^2$ configurations for an N-port switch leading to high overhead or large batches and hence high delay. We present two new algorithms that reduce the number of configurations required substantially. MIN covers a batch of requests in the minimum possible number of configurations, $N_s = N$, but at the expense of many empty slots, $S_{\text{min}} \approx 4 \log_2 N$. DOUBLE strikes a balance, requiring twice as many configurations, $N_s = 2N$, while reducing the number of empty slots so that $S_{\text{min}} = 2$. We show that DOUBLE offers the lowest required speedup to emulate an unconstrained switch across a wide range of port count and delay.**

I. Introduction

Optical switches based on MEMs mirrors, tunable elements, bubble switches, and similar technologies [1][2][3][4][5] have been developed to meet the exponentially increasing demand for switch bandwidth and port count. These optical switching technologies offer high bandwidth in an economical manner. Switches built with this technology, however, require significant time to reconfigure due to mechanical settling, synchronization, and other factors. These configuration overheads range from milliseconds for bubble and free-space MEMs switches [2][3], to 10μs for MEMs waveguide switches [4], and as little as 10ns for electroholographic techniques [5]. With typical cell sizes on the order of 50ns (64 bytes at 10Gb/s), these switches take from 0.2 to 20,000 cell times to reconfigure. Efficiently scheduling such optical switches requires algorithms that take this configuration overhead into account and optimize the resulting schedule.

Algorithms and architectures for unconstrained (zero overhead) switches often rely on the fact that switches are *stateless*: any configuration can be presented each *slot time* with no difference in switch behavior. The configuration overhead of optical switches introduces state: a reconfiguration overhead is experienced if the switch configuration differs from the previous slot's configuration.

This paper develops an architecture and algorithms for using a constrained switch to exactly emulate the behavior of a unconstrained switch with a fixed delay. As long as the system employing the switch can tolerate the fixed delay, the emulation architecture can directly replace an unconstrained switch. In essence, emulation decouples the constraints of non-zero switching overhead from the classic switch scheduling problem. This allows designers to use optical signaling and switching directly with existing architectures and scheduling algorithms. Unlike previous algorithms that perform best-effort scheduling of constrained switches [6][7][8], the algorithms we present give guaranteed performance.

The emulation architecture operates by accumulating a batch of T switch requests and then mapping this batch onto a set of $N_s < T$ switch configurations. Reducing the number of configurations reduces the time spent reconfiguring the switches and hence reduces the delay required for emulation. However, there is a tradeoff as aggressive reduction in the number of configurations can lead to a large number of empty slots and hence require a large speedup.

We explore three algorithms that span this design space of number of the configurations versus the number of empty slots. At one end of the design space, a well-known exact decomposition algorithm, EXACT [7], generates a schedule with no empty slots but requires $N_s \approx N^2$ configurations (where N is the number of ports) and therefore a very high delay. At the other extreme, we introduce a new algorithm, MIN, that generates a minimum number of configurations, $N_s = N$, but leaves most slots empty and requires a switch speedup of $\Theta(\log N)$. We balance delay and speedup with another new algorithm, DOUBLE, that requires twice the minimum number of configurations, $N_s = 2N$, but leaves at most half of the slots empty, thus requiring a switch speedup of 2.

We compare the overhead of these three algorithms across the space of switch size N and delay T. Our results show that DOUBLE offers the lowest overhead of the three algorithms across a wide portion of this space. EXACT offers better performance only for low port count or high delay, and MIN offers better performance only for very low delays. Viewed another way, for a fixed overhead, DOUBLE requires much lower delay for emulation than EXACT at the expense of a speedup of two. For example, for a $N = 128$ port MEMS switch with a configuration time of 10μs, EXACT requires a minimum delay of 320ms while DOUBLE can operate with a delay of 5ms.

The remainder of this paper explores the design of algorithms

B. Towles and W.J. Dally are with the Computer Systems Laboratory in the Department of Electrical Engineering, Stanford University. This work has been supported by an NSF Graduate Fellowship with supplement from Stanford University and under the MARCO Interconnect Focus Research Center. E-mail: {btowles,billd}@cva.stanford.edu

that provide service guarantees for switches with configuration overhead in more detail. Section 2 introduces our notation along with a simple switch model used throughout the paper. The emulation architecture is detailed in Section 3. Section 4 introduces the three algorithms and discusses their performance guarantees. Section 5 compares the three algorithms in terms of overhead and delay as a function of switch ports. Related work is discussed in Section 6. Finally, conclusions are drawn in Section 7. Correctness proofs for the new algorithms are included in an appendix.

II. Preliminaries

Below is a list of symbols used throughout the paper:

$a_{i,j}$	element (i, j) of matrix A
C	cumulative request matrix, the sum of the switch configurations requested over a period of time; the rows and columns sum to the number of configurations requested
$C(T)$	C where rows and columns sum to T
δ	switching overhead in slot times
H	batch scheduling time in slot times (rounded up to the nearest integral number of batch times T to allow pipelining)
N	number of switch ports
N_s	number of switchings per batch
ϕ	switch configuration interval (weight)
S	internal speedup of switch
P	switch configuration / permutation matrix
T	batch size in slot times

This paper deals with scheduling of a crossbar switch that can realize any one-to-one (unicast) mapping of inputs to outputs. Such a mapping is described by a *switch configuration* P, where P is a permutation matrix; when an element $p_{i,j}$ is one, input i is connected to output j for that configuration. Multicast traffic is not considered. Time is slotted and a new configuration may be provided to the crossbar each *slot time*.

Unlike typical electronic switches, the model also associates a fixed, non-zero *switching overhead* δ with each *switching event* (any change in the switch configuration). The fixed switching overhead is intended to capture all effects, such as mechanical settling times and synchronization overhead, that temporarily prevent transmission as a switching element is reconfigured. An *unconstrained* switch has $\delta = 0$, whereas a *constrained* switch has $\delta > 0$. We express δ in units of slot times.

Finally, a matrix A is *covered* by a set of switch configurations $P(1), \ldots, P(N_s)$ and corresponding weights $\phi(1), \ldots, \phi(N_s)$ if

$$\sum_{k=1}^{N_s} \phi(k) p_{i,j}(k) \geq a_{i,j}, \quad \forall i, j \in \{1, \ldots, N\}.$$

In the case of equality for all i and j, the switch configurations *exactly cover* A.

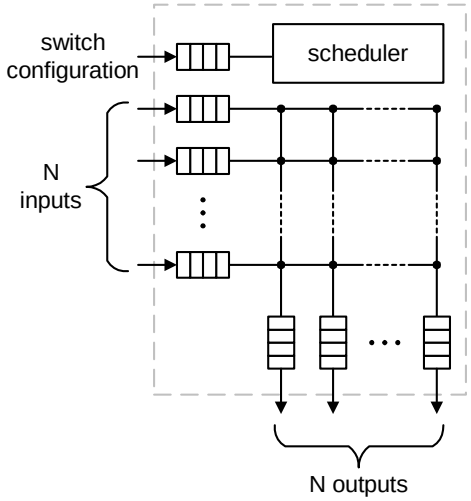


Fig. 1. Emulation architecture

III. Architecture

We emulate an unconstrained switch using a constrained crossbar with input and output queues (Figure 1) where the constrained crossbar has *speedup* S to compensate for its switching overhead δ. The dashed boundary represents the standard unconstrained interface: N inputs, N outputs, and a configuration input. The speedup S refers to the ratio of the internal line rate to the input line rate. The input and output queues enable this rate mismatch by physically decoupling the internal and external lines.

A. Emulation Approach

The scheduler in Figure 1 performs pipelined batch scheduling in four phases. In the first phase, a batch is created by accumulating the requested configurations $P(t)$ over an interval T such that

$$C(T) = \sum_{t=n}^{n+T-1} P(t).$$

Later phases may reorder the data, so incoming data is tagged with its arrival time allowing the original order to be restored.

The second phase finds N_s switch configurations $P'(1), \ldots, P'(N_s)$ and weights $\phi(1), \ldots, \phi(N_s)$ that cover $C(T)$.

In the third phase, the switch is configured in the sequence $P'(1), \ldots, P'(N_s)$ with each configuration held for $\phi(1), \ldots, \phi(N_s)$ slot times. If the switch configurations cover C, all traffic accumulated in the first phase can traverse the switch during the third phase. Guaranteeing that all the traffic *does* traverse the switch ensures no data element stays in the input queues for more than $T + H$ slot times. After traversal, data is reordered as it is stored in the output buffers.

Finally, the fourth phase simply sends the data from the output buffers onto the output lines in the same order it entered the switch. As shown by the arc between an arriving data element and its departure from the switch (Figure 2), this relationship implies a delay bound of $2T + H$ when H slot times are reserved for the second phase scheduling. Therefore the outputs

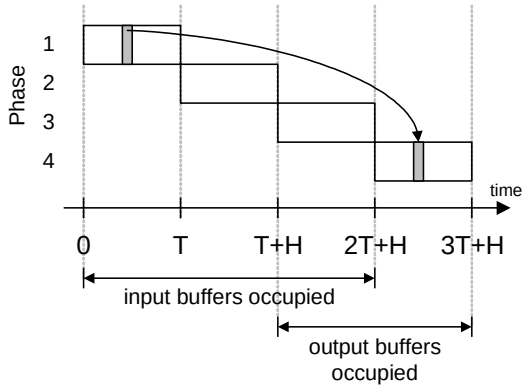


Fig. 2. Batch timeline

exactly emulate the behavior of a corresponding unconstrained switch plus the fixed delay of $2T + H$.

As expected, the amount of storage required in the architecture grows linearly with T. The pipeline diagram (Figure 2) indicates the buffer occupancy required for a single batch. Let L indicate the number of bits sent to a single input port during a slot time. Considering one port, a batch is held for $2T + H$ slot times in the input buffers and since a new batch is started each T slot times, enough buffers for $(2T + H)L$ bits of data are required in the input stage. Similarly, data is held for $2T$ slot times in the output stage, requiring $2TL$ bits of buffering. So, considering all ports, the architecture needs $(4T + H)LN$ bits of buffering total.

B. Emulation Requirements

To compensate for the overhead of switch configuration δ and slots left empty by the scheduling algorithm, the emulation architecture must operate with a speedup S that depends on the batch size T as illustrated in Figure 3. S is selected to ensure that $C(T)$ can be completely transmitted during the third phase of the emulation algorithm. The time spent on configuration overhead during each batch of N_s configurations is $T_{\text{min}} = \delta N_s$, the left asymptote of the S versus T curve. This leaves time $T - T_{\text{min}}$ to send T slots of data. If the scheduling algorithm exactly filled each of the slots with data, the speedup required would be $S_{\text{exact}} = T/(T - T_{\text{min}})$.

Not all algorithms completely fill the slots, however. So the total number of slots used by a scheduling algorithm T_s can be greater than T in general. Thus, the speedup required to compensate solely for these empty slots is $S_{\text{min}} = T_s/T$, which gives the bottom asymptote of the S versus T curve. Viewed another way, the fraction of slots filled by the scheduling algorithm is $1/S_{\text{min}}$. So, for example, if half the slots are filled with data, an additional speedup of $S_{\text{min}} = 2$ is required beyond the speedup S_{exact} necessary to compensate for switching overhead.

Multiplying these two speedups gives the total speedup required for a particular batch size T:

$$S = \frac{S_{\text{min}} T}{T - T_{\text{min}}} = \frac{S_{\text{min}} T}{T - \delta N_s}, \quad T > T_{\text{min}}.$$

This relationship can be also rewritten to give the batch size

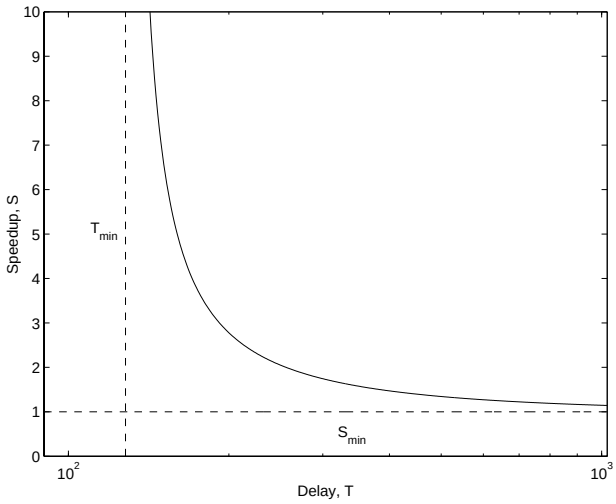


Fig. 3. Speedup required for emulation ($N_s = 128$, $\delta = 1$)

required for a particular speedup S,

$$T = \frac{S T_{\text{min}}}{S - S_{\text{min}}} = \frac{\delta N_s S}{S - S_{\text{min}}}, \quad S > S_{\text{min}}.$$

IV. Scheduling Algorithms

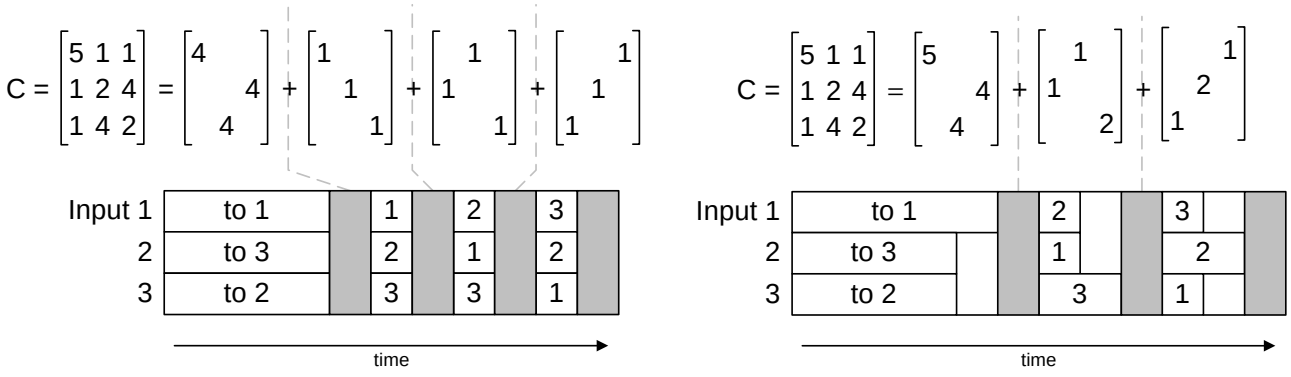
The scheduling task is a time-slot assignment problem. Given an input-output request matrix C, assign a switch traversal time for each element in C so that the total transmission time is minimized. Emulation also requires *guarantees* about the performance of scheduling algorithms. That is, for any matrix C and switching overhead δ, the worst-case transmission time required for a scheduling algorithm must be bounded.

This section presents several approaches for achieving guaranteed performance and examines the tradeoff between the number of switch configurations used to cover the matrix and the number of empty slots left by the algorithm. An example of this tradeoff is illustrated in Figure 4. First, a request matrix C is decomposed into 4 switch configurations that exactly cover C (Figure 4a). The accompanying time-slot assignment diagram shows the connection of inputs (shown vertically) to particular outputs, denoted by slot labels. The shaded segments show the switching time required between different configurations. An alternative decomposition of C gives only 3 switchings, but the corresponding time-slot assignment contains empty slot times (Figure 4b). Since each configuration is held for the maximum time of all the elements contained within it, some slots are left unused. From this simple example, it should be clear that fewer switchings require less overhead time, but at the potential cost of leaving slots empty during switch traversal. This tradeoff is quantified in the following subsections.

A. Exact covering

A well-known decomposition of any matrix $C(T)$ [7][9] exactly covers the matrix in at most $N_s = N^2 - 2N + 2$ switch configurations.

Theorem 1: $N_s = N^2 - 2N + 2$ switch configurations and positive integer weights $\phi(1), \ldots, \phi(N_s)$ are necessary and sufficient to exactly cover any $N \times N$ matrix $C(T)$.



(a) Decomposition into 4 configurations

(b) Decomposition into 3 configurations

Fig. 4. Tradeoff between fewer switch configurations and empty slots

Proof: As noted in [10] (page 36), necessity is proved in [11] and sufficiency in [9]. □

Several algorithms are suggested in [7] to realize the lower bound on the number of configurations required. These algorithms include optimizations to improve the average number of configurations, but we consider a simple algorithm that only meets the bound of $N_s = N^2 - 2N + 2$ configurations. Such an algorithm performs $O(N^2)$ maximum-size matchings, which have a time complexity of $O(N^{2.5})$. Therefore, the overall time complexity of an algorithm to exactly cover C is $O(N^{4.5})$.

Given the bound on the number of switch configurations, the total amount of switching overhead can be determined, and therefore the required speedup can be calculated.

Corollary 1: A speedup of

$$\frac{T}{T - \delta(N^2 - 2N + 2)}$$

is sufficient to schedule $C(T)$ in T slot times.

Proof: This follows directly from the number of switchings $N_s = N^2 - 2N + 2$ and the minimum speedup of $S_{\min} = 1$ required for an exact covering. □

Note that the minimum fixed delay experienced by the switch, $T_{\min} = \delta(N^2 - 2N + 1)$, must grow at least with the square of the number of ports on the switch. This implies the amount of storage must also grow with $O(N^2)$. In a system where the bandwidth between input-output pairs is expensive relative to the cost of providing the storage and tolerating the fixed delay, exact covering is an attractive approach. Since $N^2 - 2N + 1$ configurations are necessary to exactly cover C, further reducing the number of switchings would introduce empty slots leading to a waste of the expensive bandwidth. Alternatively, in systems with inexpensive bandwidth a designer may be willing to trade that bandwidth for a smaller fixed delay and less storage requirements.

B. Minimum switchings

While $O(N^2)$ configurations are necessary to exactly cover C, it is possible to cover any C with as few as N configurations. This is clearly the minimum number of configurations as C has N^2 non-zero entries in general and each configuration covers at most N of these entries. However, the use of fewer configurations introduces empty slots which must be overcome with speedup. In this section, we show that cost of these empty slots can be quite significant: for $N_s = N$, $S_{\min}$ is $\Theta(\log N)$.

$$A = \begin{bmatrix} T' & & & & & & \\ & T'/2 & T'/2 & & & & \\ & T'/2 & T'/2 & & & & \\ & & & T'/3 & T'/3 & T'/3 & \\ & & & T'/3 & T'/3 & T'/3 & \\ & & & T'/3 & T'/3 & T'/3 & \\ & & & & & & \ddots \end{bmatrix}$$

Fig. 5. Portion of adversarial matrix

Theorem 2: To transmit a general cumulative schedule matrix $C(T)$ in N switch configurations, $S_{\min}$ must be at least $\Omega(\log N)$ for $T > N$.

Proof: An adversarial matrix C is constructed by the following algorithm[1]:

Step 1. Initialization. Create two $N \times N$ matrices, A and B. Initialize all entries of A to zero and all entries of B to one. Set $i \leftarrow 1$ and $j \leftarrow 1$.

Step 2. Build A. Fill the submatrix

$$A(i : i + j - 1, i : i + j - 1) = T'/j,$$

where $T' = T - N$. Set $i \leftarrow i + j$ and $j \leftarrow j + 1$. If $i + j > N$ go to Step 3, otherwise repeat Step 2.

Step 3. Create C. Set $C \leftarrow A + B$ and $I \leftarrow i$.

From the construction of A (Figure 5), it is clear that each row and column sums to T'. The rows and columns of B each sum to N, and therefore each row and column sum of C is T.

The addition of the B matrix to A guarantees that there are no non-zero elements in C. Since all N^2 elements are covered in N switch configurations, each configuration must cover N unique elements. This implies that each element is included in exactly one switch configuration. For any scheduling algorithm that covers C, a switch configuration $P(1)$ will

[1] For clarity, this proof assumes that all parameters are such that the elements of C are integers. However, the same result holds if the elements of the constructed C are all rounded down to the nearest integer.

contain the $T' + 1$ entry (element $c_{1,1}$). Two switch configurations are required to cover all the $T'/2 + 1$ entries, so at least one of the entries will be in a configuration $P(2)$, where $P(2) \neq P(1)$. Likewise, one $T'/3 + 1$ entry will be in $P(3)$, where $P(3) \neq P(2)$ and $P(3) \neq P(1)$. This argument continues for I of the switch configurations. Since the time required for a switch configuration is the maximum of all elements in that configuration, switching $P(1)$ through $P(I)$ requires at least

$$(T' + 1) + (T'/2 + 1) + \ldots + (T'/I + 1) > T' \ln I.$$

From the above algorithm, I is the largest integer such that $\sum_{i=1}^{I} i \leq N$, or

$$I = \lfloor (\sqrt{1 + 8N} - 1)/2 \rfloor > \sqrt{2N} - 3/2.$$

Substituting yields the total number of time slots required,

$$(T - N) \ln(I) > (T - N) \ln(\sqrt{2N} - 3/2).$$

Therefore an $S_{\min}$ of at least $\Omega(\log N)$ is required. □

This result shows that regardless of the algorithm used, scheduling C so that there are only N switch configurations requires $S_{\min} = \Omega(\log N)$ in general. A simple algorithm MIN (Algorithm 1) shows this bound on the minimum speedup is also sufficient[2]. The algorithm's running time is dominated by N maximum size matchings, for a total time complexity of $O(N^{3.5})$.

The MIN algorithm operates by identifying the largest, unscheduled elements of C in Step 2 and then schedules these elements in Steps 3–4. At the beginning of each outer loop iteration (Step 2), unscheduled elements of C greater than the threshold T/d are indicated in A. Because the rows (columns) of C sum to T, there can be at most $d-1$ elements greater than or equal to this threshold in each row (column). Then, at least one element per column of A is removed in the inner loop (Step 4) and by performing the inner loop $d-1$ times, all the elements in C above the threshold are scheduled.

Before the inner loop begins, the bipartite multigraph G_A corresponding to A is edge-colored. This ensures no two elements in the same row or column have the same color, which requires at most $d - 1$ colors by the classical result of König. Then, each iteration of the inner loop handles one of these colors at a time.

During the inner loop, all the edges of a particular color are partitioned into two half-size subsets to guarantee they are all scheduled as part of a perfect matching[3]. By only considering half the edges at time, it is guaranteed that a perfect matching that contains these edges exists in G_A as long as the degree of G_A is more than $3N/4$, which is ensured by the loop condition in Step 5. This is formally proved in the Appendix.

For each iteration of the outer loop, $2(d - 1)$ schedules are created each iteration, each with weight $2T/d$. Therefore, the total weight per iteration is approximately $4T$. Also, since d is doubled each loop iteration, the total number of iterations is proportional to $\log N$. Step 6 schedules the remaining elements of C with a fixed weight, but since these elements are small, their contribution does not affect the overall logarithmic behavior of the algorithm.

[2]The algorithm and analysis presented assume $N \geq 8$ for simplicity. Cases where $N < 8$ can be handled by slightly modifying Steps 4–5.

[3]A perfect matching is a subset of edges such that each vertex is incident with exactly one edge in that subset.

Algorithm 1 Minimum switchings (MIN)

Step 1. Initialization. Create an $N \times N$ indicator matrix B with all entries set to one. Set $d \leftarrow 2$ and $k \leftarrow 1$.

Step 2. Identify large elements. Define the $N \times N$ matrix A such that

$$a_{i,j} = \begin{cases} 1 & \text{if } c_{i,j} > T/d \text{ and } b_{i,j} = 1 \\ 0 & \text{otherwise} \end{cases}$$

Step 3. Color. Construct the bipartite graph G_A from A (zero entries do not have a corresponding edge). Perform a minimal edge coloring of G_A.

Step 4. Schedule. Set $c \leftarrow 1$.

Step 4a. Partition edges. Let the matching M_c be the subset of edges in G_A assigned to color c. Take any subset of edges $E_a \subseteq M_c$, such that $|E_a| = \lceil |M_c|/2 \rceil$. Then $E_b \leftarrow M_c - E_a$.

Step 4b. Schedule E_a. Construct the bipartite graph $G_B = (E_B, V_B)$ from B. Remove edges from E_a which have been previously scheduled by setting $E_a \leftarrow E_a \cap E_B$. Then, for each edge in E_a, remove the corresponding edge, that edge's endpoints, and edges incident to those endpoints from G_B. Find the maximum-size matching M_B on the remaining vertices and edges of G_B. Construct the configuration $P(i)$ from the combination of the two matchings $M_c \cup M_B$ and set the weight $\phi(i) \leftarrow \lfloor 2T/d \rfloor$. Set $B \leftarrow B - P(i)$ and $i \leftarrow i + 1$.

Step 4c. Schedule E_b. Repeat the procedure of Step 4b, but for the edges of E_b instead of E_a.

Step 4d. Loop over colors. Set $c \leftarrow c+1$. If $c \leq d-1$, then go to Step 4a. Otherwise continue to Step 5.

Step 5. Loop. Set $d \leftarrow 2d$. If $(i - 1) + 2(d - 1) \leq N/4$, then go to Step 2. Otherwise continue to Step 6.

Step 6. Finish. Construct the bipartite graph G_B from B. Perform a maximum-size matching on G_B and produce the switch schedule $P(i)$. Set $\phi(i) \leftarrow \lfloor 2T/d \rfloor$, $B \leftarrow B - P(i)$, and $i \leftarrow i + 1$. Repeat Step 6 until there are no non-zero elements remaining in B.

Figure 6 shows an execution of the MIN algorithm for a matrix with $N = 32$ and $T = 32$. For simplicity, only a portion of the matrices and the first several steps are illustrated. In the first iteration of the example, $d = 2$ and the first cutoff is $T/d = 16$. All entries > 16 are considered for scheduling and indicated in A. For the first iteration, A requires only $d-1 = 1$ color in Step 3. Then, during Step 4a, the non-zero entries of A are partitioned into two subsets E_a (circled) and E_b (not circled). The elements of E_a are a subset of a perfect matching found in Step 4b, which is used as schedule $P(1)$ with weight $\phi(1) = 2T/d = 32$. Similarly, the elements of E_b are scheduled in Step 4c. After both steps, B is shown with zero entries corresponding to the scheduled elements of C.

The outer loop is repeated for $d = 4$ and all unscheduled elements in C greater than $T/d = 8$ are indicated in A. Again,

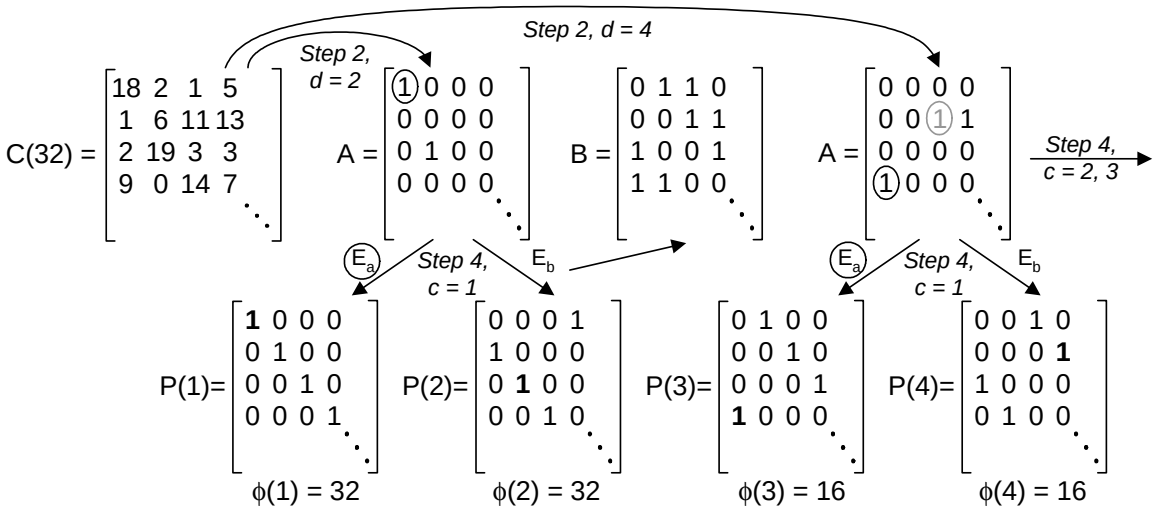


Fig. 6. Example execution of MIN ($N = 32$, $T = 32$). Only a portion of all steps is shown.

A is colored using $d-1=3$ colors. Schedules $P(3)$ and $P(4)$ correspond to the first color, while the remaining colors (shown in gray) are used for schedules $P(5)$ through $P(8)$. In Step 5, $(i-1)+2(d-1) = 8+14 = 22$ is greater than $N/4 = 8$, so the algorithm goes to Step 6 and creates the remaining schedules.

The general operation of MIN is verified in the Appendix.

Theorem 3: To cover a general cumulative schedule matrix $C(T)$ with N switch configurations $S_{\min} = 4T(4+\log_2 N)$ is sufficient.

Proof: Let m be the number of iterations of the outer loop of MIN (Steps 2–5). m is the largest integer such that

$$2\sum_{i=1}^{m} 2^i - 1 = 2(2^{m+1} - m - 2) \leq N/4.$$

Using m, the total weight of the schedules produced from Steps 2–5 is then

$$2\sum_{i=1}^{m}(2^i-1)\lfloor 2T/2^i \rfloor < 4Tm.$$

And the total weight produced during Step 6 is

$$(N-2(2^{m+1}-m-2))\lfloor 2T/2^{m+1}\rfloor \leq \frac{2T(N+2m+4)}{2^{m+1}} - 4T.$$

By conservatively estimating m as $\lfloor log_2(N/16)\rfloor$, a bound on the total weight is then

$$4T\lfloor log_2(N/16)\rfloor + \frac{2T(N+2\lfloor log_2(N/16)\rfloor+4)}{2^{\lfloor log_2(N/16)\rfloor+1}} - 4T.$$

Through further simplification, this expression can be bounded by $4T(4+\log_2 N)$. Therefore the minimum speedup is sufficient. □

Corollary 2: A speedup of

$$\frac{4T(4+\log_2 N)}{T-\delta N}$$

is sufficient to schedule $C(T)$ in T slot times.

Proof: This follows directly from the number of switchings $N_s = N$ and the minimum speedup of $S_{\min} = 4(4+\log_2 N)$ required for MIN. □

So, while successfully reducing the number of configurations to the minimum possible, the amount of speedup required to support this few switchings grows with $\log N$. This could be an effective tradeoff for switches with inexpensive bandwidth or a small number of ports. However, for larger switches, the required speedup factor could be too expensive. In this case, a more attractive alternative may be to use a near minimum number of configurations.

C. Near minimum switchings

As described in the previous section, using the minimal number of switchings requires a speedup of at least $\log N$. In this section we show that by allowing $2N$ switchings, the minimum speedup $S_{\min}$ can be reduced to approximately two. Most importantly, the minimum speedup is no longer a function of N. This approach has the advantage of the EXACT algorithm, a small, constant speedup, combined with a number of switchings that grows linearly with N. The DOUBLE algorithm (Algorithm 2) produces schedules with these properties in $O(N^2 \log N)$ time using the edge-coloring algorithm of [12].

DOUBLE works by separating C into *coarse* and *fine* matrices and devotes N configurations to each. The algorithm first generates the coarse matrix A by dividing the elements of C by T/N and taking the floor. The rows and columns of A sum to at most N, thus the corresponding bipartite multigraph can be edge-colored in N colors. Each subset of edges assigned to a particular color forms a matching, which is weighted by $\lceil T/N \rceil$. The fine matrix for C does not need to be explicitly computed because its elements are guaranteed to be less than $\lceil T/N \rceil$. Thus any N configurations that collectively represent every entry of C, each weighted by $\lceil T/N \rceil$, can be used to cover the fine portion.

An example execution of DOUBLE is shown in Figure 7. The algorithm begins by creating the coarse matrix A by dividing each element in C by T/N and taking the floor. So,

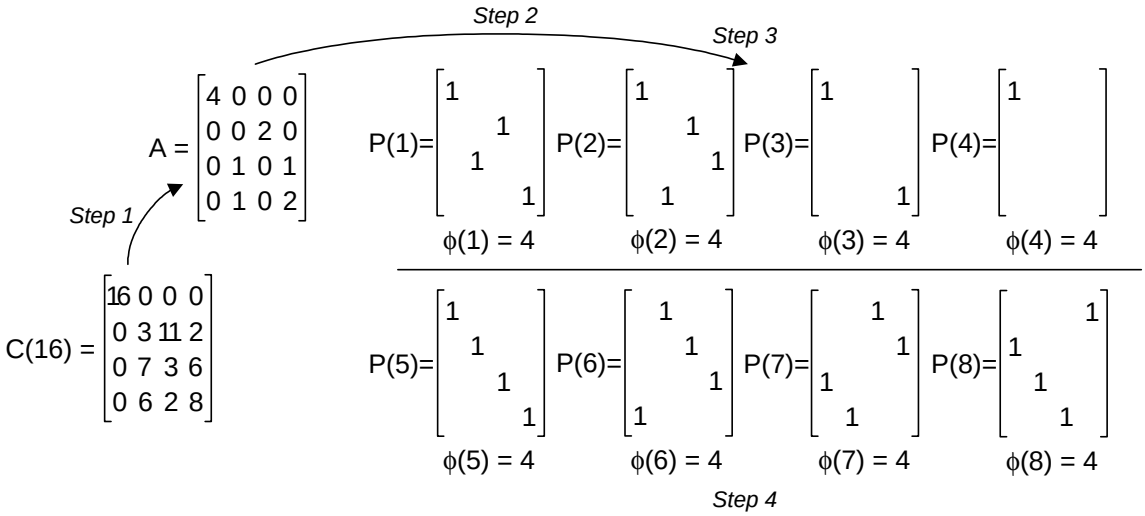


Fig. 7. Example execution of DOUBLE ($N = 4, T = 16$)

Algorithm 2 Near minimum switchings (DOUBLE)

Step 1. Split C. Define an $N \times N$ matrix A such that

$$a_{i,j} = \left\lfloor \frac{c_{i,j}}{T/N} \right\rfloor.$$

Step 2. Color A. Construct the bipartite multigraph G_A from A (the number of edges between vertices is equal to the value of the corresponding entry of A). Find a minimal edge-coloring of A. Set $i \leftarrow 1$.

Step 3. Schedule coarse. For a specific color in the edge-coloring of G_A, construct a switch configuration $P(i)$ from the edges assigned that color. Set $\phi(i) \leftarrow \lceil T/N \rceil$ and $i \leftarrow i + 1$. Repeat Step 3 for the each of the colors in G_A.

Step 4. Schedule fine. Find any N non-overlapping switch schedules $P(N+1), \ldots, P(2N)$ and set $\phi(N+1), \ldots, \phi(2N)$ to $\lceil T/N \rceil$.

in the example, entry $(1,1)$ of A contains $\lfloor 16/(T/N) \rfloor = 16/(16/4) = 4$. The resulting matrix A has row and column sums ≤ 4, ensuring that it can be edge colored with 4 colors (Step 2). Then, the edges assigned to each color are converted to schedules in Step 3. For example, $P(1)$ corresponds to the subset of edges assigned to color 1 during Step 2. Also, some of the schedules may not be complete permutations because the row and column sums of A are less than N, such as $P(3)$ and $P(4)$, but it is still guaranteed that all the elements of A are covered. In general, Step 3 creates at most N matchings with weight $\lceil T/N \rceil$, for a total weight of approximately T.

Step 4 picks 4 non-overlapping schedules, $P(4)$ through $P(8)$, and each is assigned a weight of $\lceil T/N \rceil = 4$. In general, Step 4 creates the same total weight as Step 3: approximately T. Therefore the total weight to schedule $C(T)$ using DOUBLE is approximately $2T$ and $S_{\min} = 2$. The general operation of DOUBLE is verified in the Appendix.

The required speedup is now simply derived from the weights assigned by DOUBLE.

Theorem 4: To transmit a general cumulative schedule matrix $C(T)$ in $2N$ switch configurations $S_{\min} = 2$ is sufficient when T is a multiple of N.

Proof: DOUBLE produces $2N$ switch configurations, each with a weight of $\lceil T/N \rceil$. Summing these weights,

$$2N \left\lceil \frac{T}{N} \right\rceil = 2T.$$

Therefore the minimum speedup is sufficient. □

Corollary 3: A speedup of

$$\frac{2T}{T - 2\delta N}$$

is sufficient to schedule $C(T)$ in T slot times when T is a multiple of N.

Proof: This follows directly from the number of switchings $N_s = 2N$ and the minimum speedup of $S_{\min} = 2$ required for DOUBLE. □

V. Discussion

The previous section detailed three algorithms for unconstrained switch emulation. Given these algorithms, which is the most appropriate for a particular system? The answer depends on the relative costs of bandwidth, delay, storage, and the switching overhead δ in the system.

If the system designer is insensitive to delay and storage requirements, but considers bandwidth expensive, then the EXACT algorithm is most likely an appropriate design choice. However, exact scheduling can lead to large delays, even with feasible system parameters. For example, consider a 128 port switch with 10 Gbit/sec input lines and a 64 byte slot (slot time of 50ns). Fast MEMS mirror switches are used, which have a switching time of $\delta = 200$ or 10μs [4]. For exact matching, $T_{\min} = \delta N_s$ is approximately 3.2 million slot times or 160ms, which makes the minimum fixed delay $2T + H$ equal to 320ms plus the scheduling time. This delay is obviously unacceptable for many switching applications.

The minimum switching algorithm MIN greatly reduces the fixed delay over the exact algorithm, but at the cost of increased

speedup. In our example 128 port switch, MIN reduces the minimum fixed delay to 2.5ms, but requires a minimum speedup of $4(4 + \log_2 N) = 44$.

DOUBLE provides a balance between the two other algorithms. For the 128 port switch, a minimum fixed delay of 5ms and minimum speedup of 2 are necessary. So, compared to the exact algorithm, a speedup of 2 reduces the fixed delay by a factor of 128. Alternatively, DOUBLE allows a switching overhead δ that is 128 times greater than the exact algorithm for the same fixed delay. Assuming there is a cost benefit in slower switches, the potential savings from using slower switches may more than offset the cost required to provide a speedup of 2. A summary of the costs for each scheduling algorithm is shown in Table I.

The tradeoffs between the different scheduling algorithms are represented graphically in Figure 8. Figure 8a shows a "phase diagram" indicating which algorithm gives the minimum speedup S for particular values of T and N. The regions partitioned by the lines represent the parameters for which the labeled algorithm provides the smallest speedup. So, for small values of T, the MIN algorithm has the smallest speedup because it is the only algorithm for which $T > T_{\text{min}}$. Soon after T is large enough for DOUBLE to be used, it becomes the algorithm of choice and likewise for the exact algorithm. For the example of $N = 128$ and $\delta = 200$, DOUBLE becomes preferred at approximately $T =$ 53,200 slot times and EXACT provides the lowest speedup at $T =$ 6,400,000 slot times (marked as circles in Figure 8a). A similar graph is shown in Figure 8b for the minimum delay T given S and N. As the speedup passes 2, DOUBLE becomes the favored algorithm, and at $4(4 + \log_2 N)$, MIN is preferred. In the example, DOUBLE provides the smallest delay at just beyond $S = 2$ and MIN at $S = 54.4$ (marked in Figure 8b).

VI. Related Work

The time-slot assignment problem has received significant attention in the context of scheduling satellite-switched time-division multiple access (SS/TDMA) systems. Notably, algorithms to find exact decomposition of a matrix C in a minimum number of switch configurations are described in [7]. The idea of using only N switch configurations was introduced in [8], where the authors proved the problem of finding the minimum length schedule for a particular matrix C to be NP-complete. They also introduced a heuristic algorithm to create the schedules. The SS/TDMA scheduling problem is the same as the scheduling problem considered in this paper. However, making an analogy to packet routing, existing algorithms provide "best-effort" schedules, where the goal is to minimize the average schedule length. We demonstrated new algorithms that solve the same scheduling problem, but have provable worst-case guarantees necessary for emulation.

More recently, similar problems have been considered in wavelength-division multiplexing (WDM) systems. Both [13] and [14] provide heuristic algorithms for scheduling transmissions in star networks given a number of tunable receivers and transmitters with non-zero tuning latencies. Optimal all-to-all transmission schedules for the star networks are considered in [15]. The problems addressed by these researchers are more broad in that multiple transmitters and receivers per input are used, but again schedules are chosen to minimize the average length, not to provide a bounded worst-case.

The impact of constrained switches on packet switch scheduling has also been addressed. The work of [16] develops an architecture and several algorithms to guarantee throughput and delay given a larger data envelope and therefore fewer logical switch configurations. This work complements our approach in that we develop techniques to implement a given number of logical switch configurations in fewer physical configurations, thus reducing the speedup requirements of the switch.

Also, as noted in both [17] and [18], the task of computing a schedule for an *unconstrained* switch is becoming a more difficult problem as switch sizes scale. Both of these papers provide solutions to this problem centered around decomposing a traffic matrix C into permutation matrices and show that the resulting switch is stable. The algorithms presented in this paper could readily be applied to this problem, extending the work of [17] and [18] to switches with non-zero switching overhead. [18] also notes that the exact scheduling algorithm's requirement of $O(N^2)$ switch configurations limits scalability and proposes a multi-stage network to solve the problem. In this case in particular, the DOUBLE algorithm could also provide scalability for a speedup of 2.

Finally, this paper only considers schedules in which the reconfiguration of all the ports is simultaneous. However, many optical switching technologies can have some connections that remain static and continue to transmit cells while other connections are reconfigured. This decoupled version of the scheduling problem is equivalent to the open shop scheduling problem with setup times removed. For switches with three or more ports, this problem is NP-hard [19]. Some results exist for small instances of open shops with setup times (see [20] for example), but finding algorithms with worst-case bounds on schedule length for a larger number of ports is left as future work.

VII. Conclusion

Optical switching technologies are becoming an attractive alternative to electronic switches as the demand for switch bandwidth and port count increase exponentially. However, many of these optical technologies have a large switching overhead — requiring from nanoseconds to milliseconds to reconfigure. Efficient scheduling of these constrained switches requires algorithms that consider this overhead.

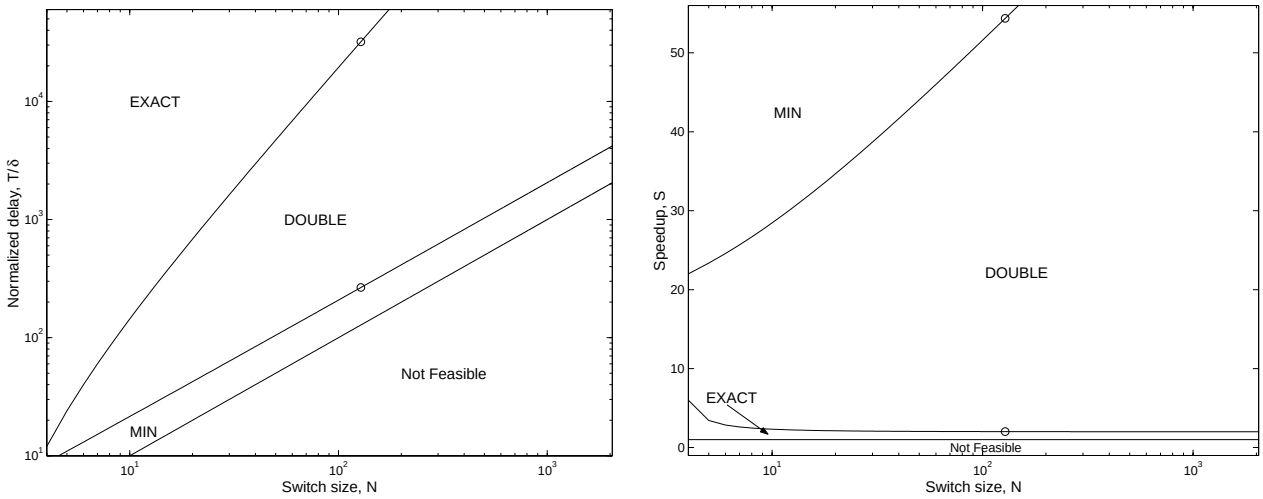
We proposed an architecture and algorithms that allow a constrained switch to exactly emulate an unconstrained switch within a fixed delay. This decouples the task of accounting for configuration overhead from the traditional switch scheduling problem. Constrained switches can then be used directly in designs that can tolerate the fixed delay.

Providing emulation requires scheduling algorithms that have guaranteed bounds on the length of their schedules. We analyzed the speedup and delay required for emulation using three bounded algorithms across a range of port sizes N and batch sizes T. The EXACT algorithm provides the lowest speedup requirement, but is only attractive for very large batch sizes, which are needed to amortize the cost of its quadratic

TABLE I

SUMMARY OF ALGORITHM COSTS (ALSO SHOWN FOR $N = 128, \delta = 200$)

	EXACT	MIN	DOUBLE
N_s	$N^2 - 2N + 2$ (16,130)	N (128)	$2N$ (256)
$S_{\min}$	1 (1)	$4(4 + \log_2 N)$ (44)	2 (2)
$T_{\min}$	$\delta(N^2 - 2N + 2)$ (3,226,000)	δN (25,600)	$2\delta N$ (51,200)



(a) Minimum speedup (b) Minimum delay

Fig. 8. Algorithm phase diagrams over the design space

number of configurations, or very low port counts. We developed the MIN algorithm to use the minimum number of switchings $N_s = N$, but the speedup required was shown to be prohibitive, $\Theta(\log N)$. As a result, MIN is only attractive for small batch sizes, where it is the only algorithm that will work. Alternatively, our DOUBLE algorithm balances a small number of switchings $N_s = 2N$ with a constant speedup of 2. DOUBLE offers the minimum required speedup across a wide range of N and T. The resulting family of algorithms provide a range of speedup versus delay tradeoffs, making emulation feasible over a large design space.

The work presented here raises many interesting questions for future study. The three algorithms we have presented represent three points in the space of N_s versus $R_{\min}$. It is interesting to ask what happens at other points. As we increase N_s from $2N$ to N^2 how rapidly does $R_{\min}$ fall from 2 to 1? Can a constant $R_{\min}$ be achieved for an N_s less than $2N$? Also, reconsidering the switch scheduling problem as an open shop (Section VI) may allow a reduction in empty slots for a given number of configurations. Finally, we only consider guaranteed traffic, but characterization of average-case performance of the scheduling algorithms would be useful for switches that can fill extra slots with best-effort traffic.

ACKNOWLEDGMENTS

The authors would like to thank Balaji Prabhakar for his initial discussions on this problem and Peter Glynn and Gideon Weiss for their assistance with open shops. The authors also thank the reviewers and members of Concurrent VLSI Architecture group for their helpful comments and suggestions.

REFERENCES

[1] A. Neukermans and R. Ramaswami, "MEMS technology for optical networking applications," *IEEE Commun. Mag.*, pp. 62–69, January 2001.

[2] 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*, Orlando, FL, 1988, pp. 169–170.

[3] L.Y. Lin, "Micromachined free-space matrix switches with submillisecond switching time for large-scale optical crossconnect," in *OFC Tech. Digest*, 1998, pp. 147–148.

[4] O.B. Spahn et. al, "GaAs-based microelectromechanical waveguide switch," in *Proc. 2000 IEEE/LEOS Intl. Conf. on Optical MEMS*, 2000, pp. 41–42.

[5] A.J. Agranat, "Electroholographic wavelength selective crossconnect," in *1999 Digest of the LEOS Summer Topical Meetings*, 1999, pp. 61–62.

[6] Y. Ito, Y. Urano, T. Muratani, and M. Yamaguchi, "Analysis of a switch matrix for an SS/TDMA system," *Proc. of the IEEE*, vol. 65, no. 3, pp. 411–419, 1977.

[7] T. Inukai, "An efficient SS/TDMA time slot assignment algorithm," *IEEE Trans. Commun.*, vol. COM-27, no. 10, pp. 1449–1455, 1979.

[8] S. Gopal and C. K. Wong, "Minimizing the number of switchings in a SS/TDMA system," *IEEE Trans. Commun.*, vol. 33, pp. 497–501, June 1985.

[9] D.M. Johnson, A.L. Dulmage, and N.S. Mendelsohn, "On an algorithm of G. Birkhoff concerning doubly stochastic matrices," *Canad. Math. Bull.*, vol. 3, pp. 237–242, 1960.

[10] L. Lovász and M.D. Plummer, *Matching Theory*, Elsevier Science Publishers, Amsterdam, The Netherlands, 1986.

[11] A.J. Hoffman and H.W. Wielandt, "The variation of the spectrum of a normal matrix," *Duke Math. J.*, vol. 20, pp. 37–39, 1953.

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

[13] M. Chen and T.-S. Yum, "A conflict-free protocol for optical WDMA networks," in *Proc. GLOBECOM'91*, 1991, pp. 1276–1281.
[14] A. Ganz and Y. Gao, "A time-wavelength assignment algorithm for a WDM star network," in *Proc. INFOCOM'92*, 1992, pp. 2144–2150.
[15] G.R. Pieris and G.H. Sasaki, "Scheduling transmissions in WDM broadcast-and-select networks," *IEEE/ACM Trans. Networking*, vol. 2, no. 2, pp. 105–110, April 1994.
[16] K. Kar et. al, "Reduced complexity input buffered switches," in *Proc. Hot Interconnects VIII*, 2000, pp. 13–20.
[17] E. Altman, Z. Liu, and R. Righter, "Scheduling of an input-queued switch to achieve maximal throughput," *Probability in the Engineering and Informational Sciences*, vol. 14, pp. 327–334, 2000.
[18] C.S. Chang, W.J. Chen, and H.Y. Huang, "Birkhoff-von Neumann input buffered crossbar switches," in *Proc. INFOCOM'00*, 2000, pp. 1614–1623.
[19] T. Gonzalez and S. Sahni, "Open shop scheduling to minimize finish time," *Journal of the ACM*, vol. 23, pp. 665–679, 1976.
[20] V.A. Strusevich, "Two-machine open-shop scheduling problem with setup, processing, and removal times separated," *Computers and Operations Research*, vol. 20, pp. 567–611, 1993.

APPENDIX

Two classical results from graph theory are used in the following sections.

Theorem 5: (Hall) For a bipartite graph $G = (X \cup Y, E)$, a perfect matching exists if and only if for all non-empty $S \subseteq X$, $|S| \leq |\mathcal{N}(S)|$ where $\mathcal{N}(S)$ is the set of vertices adjacent to S.

Theorem 6: (König) There exists an edge-coloring of any bipartite multigraph with a maximum degree of Δ which uses Δ colors.

A. Correctness of MIN

For simplicity, the MIN algorithm is presented for $N \geq 8$ and for this proof of correctness we also assume N is even.

Theorem 7: For a bipartite graph $G = (X \cup Y, E)$ with $|X| = |Y| = n$, there always exists a perfect matching in G if its minimum degree is greater than $n/2$.

Proof: Assume no perfect matching exists in the graph. Then by Hall's Theorem, there must exist a non-empty $S \subseteq X$ such that $|S| > |\mathcal{N}(S)|$. Since the minimum degree of G is greater than $n/2$, then $|S| > |\mathcal{N}(S)| > n/2$. Also, $|X - S| < n/2$.

By definition of $\mathcal{N}(S)$ there are no edges between $Y - \mathcal{N}(S)$ and S. Therefore, for any vertex $i \in (Y - \mathcal{N}(S))$, $\mathcal{N}(i) \subseteq (X - S)$. This implies $|\mathcal{N}(i)| \leq |X - S| < n/2$, which is a contradiction because the degree of i is greater than $n/2$. Therefore G contains a perfect matching. □

Theorem 8: For a k-regular bipartite graph $G = (X \cup Y, E)$ with $|X| = |Y| = n$ and $k > 3n/4$, any partial matching M of G with $|M| \leq n/2$ is a subset of a perfect matching of G.

Proof: Construct a copy of G in G'. For each edge in M, remove the edge, its endpoints, and edges incident to those endpoints from G'. This leaves $2(n-|M|)$ vertices in G'. Also, each removal reduces the degree of the remaining vertices of G' by at most one. Therefore, the minimum degree of the remaining vertices of G' is at least $k - |M|$.

By Theorem 7, there is a perfect matching in G' if $k-|M| > (n-|M|)/2$, or rewriting, that $|M| < 2k - n$. From the Theorem statement, $|M| \leq n/2 = 2(3n/4) - n < 2k - n$, so there is a perfect matching M' in G'. If a vertex in G was not covered in the partial matching M, it was included in G' and must be covered in the perfect matching M'. Therefore $M \cup M'$ is a perfect matching of G and M is a subset of this matching. □

Now the correctness of MIN can be examined step-by-step. Step 1 simply initializes the algorithm. Step 2 identifies all edges greater than T/d that have yet to be scheduled. The row (column) sums of A are less than d. Otherwise, the corresponding row (column) of C would be greater than $(T/d)d = T$, which is a contradiction because the row (column) sums of C are at most T. The graph G_A constructed in Step 3 has a maximum degree of at most $d-1$ because the row (column) sums of A are less than d. Then, by König's Theorem, G_A can always be edge colored with $d - 1$ colors.

Now that all the edges have been identified in Step 2 and colored in Step 3, Step 4 loops over $d-1$ colors, which is sufficient to visit each of the colors assigned to G_A. In Step 4a, half of the edges of a particular color are used as a partial matching in B. Since N is assumed to be even, $\lceil |M_c|/2 \rceil$ is at most $N/2$. By Theorem 8, Step 4a finds a perfect matching of G_B that includes E_a if G_B is k-regular with $k > 3N/4$. Regularity is enforced by the fact that only perfect matchings are removed from B throughout the algorithm. The condition on k is verified below. Also, it is possible that some of the edges in E_a were scheduled, and hence removed, since the coloring in Step 3. This is handled by simply removing these edges from E_a, which can only reduce $|E_a|$, ensuring the conditions of Theorem 8 still hold. Again, since N is even, there are at most $N/2$ edges remaining in M_c for Step 4b, so another perfect matching can be found. Therefore, Steps 4a and 4b together ensure that all the edges in G_A assigned to a particular color will be scheduled. Since this process is repeated over all the colors, all the edges in G_A will be scheduled during Step 4.

Once Step 5 is reached, all the entries greater that T/d have been scheduled during Step 4. So, during the next iteration, no entry will be greater than $2T/d$ (d has been updated in Step 5), which ensures the weight assigned to the schedules during Steps 4a, 4b, and 6 are sufficient to cover the corresponding elements of C. Also, since $2(d-1)$ additional schedules are produced in each loop, the loop condition during Step 5 ensures the above constraint on k is met. Finally, the Step 6 extracts the remaining perfect matchings from B, which are guaranteed to exist because G_B is regular.

B. Correctness of DOUBLE

The row (column) sums of A, created in Step 1, are at most N:

$$\sum_j a_{i,j} = \sum_j \left\lfloor \frac{c_{i,j}}{T/N} \right\rfloor \leq \frac{\sum_j c_{i,j}}{T/N} = N.$$

So, by König's Theorem, the edge-coloring produced during Step 2 uses at most N colors. Step 3 then produces at most N schedules, using all the edges in A exactly once. Finally, Step 4 covers every entry uniformly using N more schedules, for a total of at most $2N$ schedules. Any entry (i, j) is covered $a_{i,j}$ times in Step 3 and once more in Step 4:

$$(a_{i,j}+1)\lceil T/N \rceil = (\left\lfloor \frac{c_{i,j}}{T/N} \right\rfloor + 1)\lceil T/N \rceil \geq \frac{c_{i,j}}{T/N}(T/N) = c_{i,j}.$$

So, the schedules produced by DOUBLE cover every element of C.