A Load-Balanced Switch with an Arbitrary Number of Linecards

Isaac Keslassy, Shang-Tse Chuang, Nick McKeown
{keslassy, stchuang, nickm}@stanford.edu
Computer Systems Laboratory
Stanford University
Stanford, CA 94305-9030

***Abstract*— The load-balanced switch architecture is a pr ing way to scale router capacity. It requires no centralized s uler, requires no memory operating faster than the line-rat can be built using a fixed, optical mesh. In a recent paper v plained how to prevent packet mis-sequencing and provide throughput for all traffic patterns, and described the desig 100Tb/s router using technology available within three year there is one major problem with the load-balanced switc makes the basic mesh architecture impractical: Because th tical mesh must be uniform, the switch does not work whe or more linecards is missing or has failed. Instead we can passive optical switch architecture with MEMS switches th reconfigured only when linecards are added and deleted, all the router to function when any subset of linecards is presen working. In this paper we derive an expression for the nu of MEMS switches that are needed, and describe an algorit configure them. We prove that the algorithm will always correct configuration in polynomial time, and show examp its running time.**

I. Background

Our goal is to identify router architectures with predi throughput and scalable capacity. At the same time, we would like to identify architectures in which optical technology (for example optical switches and wavelength division multiplexing) can be used inside the router to increase capacity by reducing power consumption.

In a previous paper [1] we explained how to build a 100Tb/s Internet router with a single-rack switch fabric built from essentially zero-power passive optics, but without sacrificing throughput guarantees. Compared to routers available today, this is approximately 40 times more switching capacity than can be put in a single rack, with throughput guarantees that no commercial router can match today. The key to the scalability is the use of the *load-balanced switch*, first described by C-S. Chang *et al.* in [2]. In [1] we extended the basic architecture so that it has provably 100% throughput for any traffic pattern, and doesn't mis-sequence packets. It is scalable, has no central scheduler, is amenable to optics, and can simplify the switch fabric by replacing a frequently scheduled and reconfigured switch with a single, fixed, passive mesh of WDM channels.

This work was funded in part by the DARPA/MARCO Center for Circuits, Systems and Software, by the DARPA/MARCO Interconnect Focus Center, Cisco Systems, Texas Instruments, Stanford Networking Research Center, Stanford Photonics Research Center, and a Wakerly Stanford Graduate Fellowship.

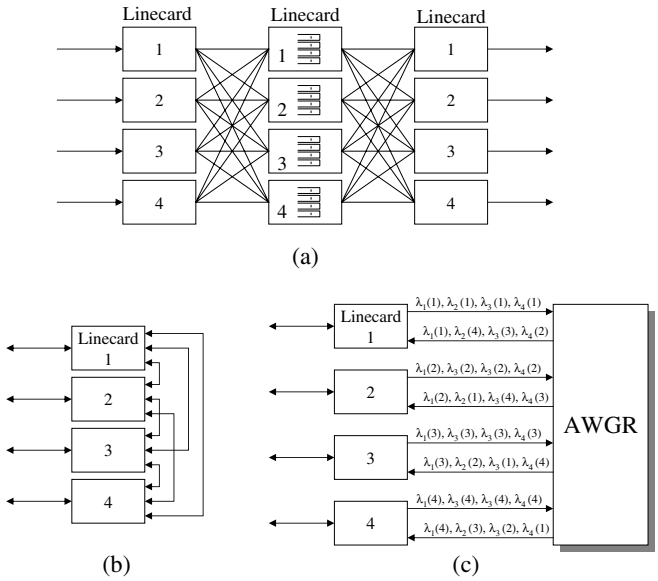

Fig. 1. Load-balanced router architecture

A load-balanced router based on an optical mesh is shown in Figure 1. Figure 1(a) shows the basic mesh architecture with $N = 4$ linecards interconnected by $2N^2$ links. Each linecard in the first stage is connected to each linecard in the center stage by a channel at rate R/N, where R is the linerate and N is the number of linecards. Likewise, each linecard in the center stage is connected to each linecard in the final stage by a channel at rate R/N. Essentially, the architecture consists of a single stage of buffers sandwiched by two identical stages of switching. The buffer at each center stage linecard input is partitioned into N separate FIFO queues, one per output (hence we call them virtual output queues, VOQs).

The operation of the two meshes is quite different from a normal single-stage packet switch. Instead of picking a switch configuration based on the occupancy of the queues, packets arriving at each input are spread uniformly over the center stage linecards. A packet arriving at time t to input linecard i is sent to linecard $[(i + t) \bmod N] + 1$; i.e. the mesh performs a cyclic shift, and each input is connected to each output exactly $\frac{1}{N}$-th of the time, regardless of the arriving traffic. The second stage mesh is identical; it services each VOQ at fixed rate R/N, regardless of its occupancy. Although they are identical, it helps

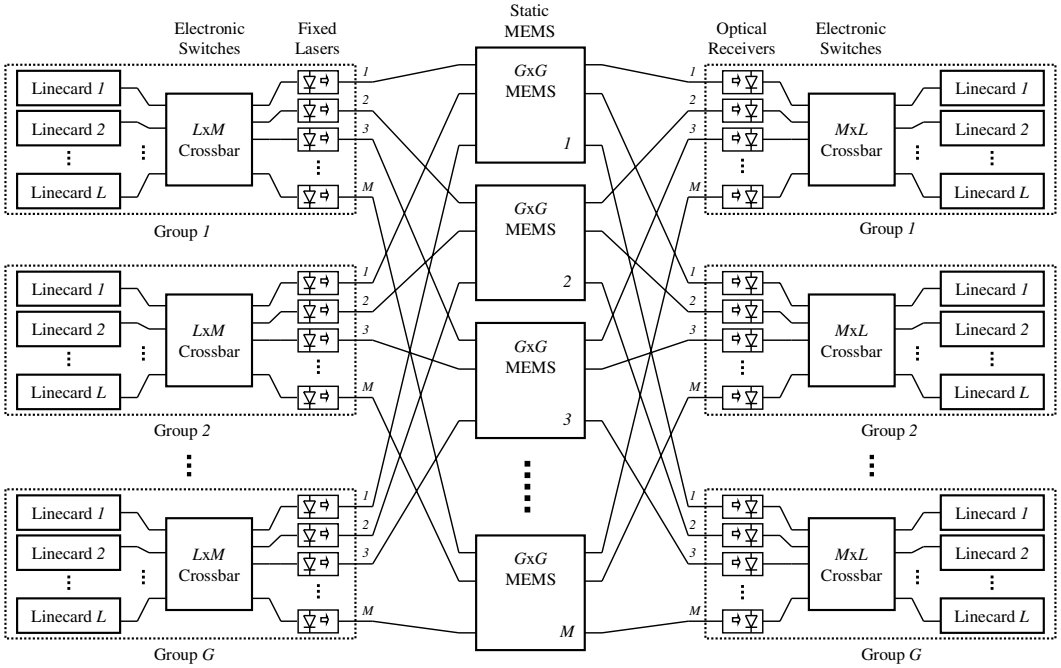

Fig. 2. Hybrid optical and electrical switch fabric.

to think of the two stages as performing different functions. The first stage is a load-balancer that spreads traffic over all the VOQs. The second stage serves each VOQ at a fixed rate. The packet is put into the VOQ at the center stage linecard according to its eventual output. Sometime later, the VOQ will be served by the second stage. The packet will then be transferred across the second switch to its output, from where it will depart the system.

Although Figure 1(a) appears to show $3N$ linecards (N for each stage), a real implementation would have N linecards, and each linecard would contain three logical parts. This means that the two meshes can be replaced by a single mesh running twice as fast, as shown in Figure 1(b). Every packet traverses the switch fabric twice: Once from the input linecard to a VOQ in the center stage linecard, then a second time from the VOQ to the output linecard. Finally, we can replace the mesh of N^2 fibers by $2N$ fibers and an arrayed-waveguide router (AWGR [3]), as shown in Figure 1(c). In this case, each input linecard uses WDM to multiplex a separate channel at rate $2R/N$ for each linecard onto a single fiber. The AWGR is a passive fixed device that permutes the channels so that each linecard receives a channel at rate $2R/N$ from each linecard.

II. Problem Statement

A. Background

Unfortunately, the load-balanced switch based on a passive mesh requires all linecards to be present and working. The load-balanced switch works by spreading packets over all linecards, and therefore needs to be aware of which linecards are present and which are not. If some linecards are missing, traffic must be spread equally over the remaining linecards.

This is a very real problem. Routers are often bought with a subset of linecards present to start with, and more are added as the network grows. Linecards fail and need to replaced, or are removed as the topology changes. In general, the router must operate when linecards are connected to arbitrary ports.

The switch fabric must therefore be able to scatter traffic uniformly over the linecards that are present. This means the switch needs to be reconfigured as linecards are added and removed, and we'll no longer be able to use a uniform fully-interconnected mesh. In [1] we described a hybrid electro-optical architecture that solves this problem, and will operate with any subset of linecards; it is shown in Figure 2. We encourage, and assume, that the reader is familiar with [1]. Previously, we haven't explained how to configure the switch, or even proved that it can spread traffic uniformly over all linecards. In this paper we describe an algorithm to do this, and prove that it will always find a valid configuration, so long as we have a sufficient number of MEMS switches.

B. Overview of problem

The architecture is arranged as G groups of L linecards. In the center, M statically configured $G \times G$ MEMS switches interconnect the G groups. The MEMS switches are reconfigured only when a linecard is added or removed. Each group of linecards spreads packets over the MEMS switches using an $L \times M$ electronic crossbar. Each output of the electronic crossbar is connected to a different MEMS switch over a dedicated fiber at a fixed wavelength (the lasers are not tunable). Packets from the MEMS switches are spread across the L linecards in a group by an $M \times L$ electronic crossbar.

When all linecards are present the operation is quite straightforward. For each linecard within a group, the electronic crossbar sends L consecutive packets to one laser. It then sends L consecutive packets to the next laser, and so on, cycling through all G lasers in turn. The first MEMS switch is statically configured to connect group g to group g; the second MEMS switch

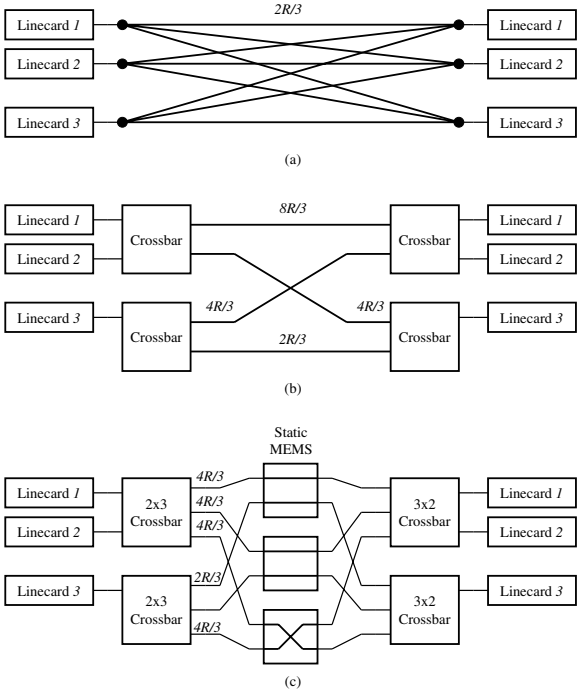

Fig. 3. Example of a hybrid switch architecture with three linecards in two groups. (a) A full linecard mesh logical view. (b) Group B is not fully populated, and so the rates between groups are different. (c) The configuration of MEMS switches to achieve the required rates.

connects group g to group $g+1$, and so on. This means that if a packet is sent from group g by laser k, it will be delivered to group $[(g+k-2) \bmod G]+1$. The electronic crossbar at the output receives L consecutive packets from each of the input groups. It spreads these packets so that one packet from each group goes to each of the L linecards. Hence, there is an equal rate path between every pair of linecards.[1]

Things get more complicated when there are fewer linecards present. We will illustrate the problem with a simple switch with just three linecards. Figure 3(a) shows three linecards connected as a full mesh; each linecard sends at rate $2R/3$ to every other linecard. Now partition the linecards into two groups, A and B, with two linecards in group A and one linecard in group B, as shown in Figure 3(b). We will determine how the electronic crossbars and MEMS switches are configured so that each pair of linecards is connected at rate $2R/3$. Group A needs to send at an aggregate rate of $8R/3$ to group A, and $4R/3$ to group B; group B needs to send at rate $4R/3$ to group A and $2R/3$ to group B. If we assume that each crossbar output can send at maximum rate $2R$, we require two outputs and two MEMS switches to connect group A to group A. We therefore need a total of three MEMS switches, two arranged in the straight configuration and one arranged in the cross configuration. The correct configurations of the MEMS switches are shown in Figure 3(c).

To spread packets uniformly over the linecards, we need to pick the static configuration for each MEMS switch, and the sequence of permutations followed by the electronic crossbars.

[1] Strictly speaking, this only works when $L \leq G$ since the numbers of MEMS switches needed between groups is equal to $\left\lceil L^2/(LG) \right\rceil = \lceil L/G \rceil$. If, say, $G=2$ and $L=3$, we need more MEMS switches; but the operation is similar.

We will do this by finding a fixed-length sequence of permutations for each $L \times M$ crossbar, then instruct each crossbar to cycle repeatedly through this sequence. Following the convention in circuit switching, we will call this sequence a *frame*.

Let us consider an example of how we might construct a frame. Consider the example in Figure 3 again. Since each linecard needs to spread its data uniformly over three output linecards, the frame will have three slots. In the frame, each linecard will send one packet to each of the three output linecards. A conflict occurs if, when a linecard sends a packet, the packet arrives at the output at the same time as another packet; or, if the packet collides with another packet in a MEMS switch. The algorithm that determines the frame needs to be aware of these conflicts.

In the remainder of this paper, we first formally describe the problem in Section III. In Section IV, we determine the minimum number of MEMS switches needed. Finally in Sections V-VII, we describe an algorithm that will correctly construct the frame.

III. Linecard Schedule Problem

We will assume throughout that there are G groups; group i contains L_i linecards, and the total number of linecards is:

$$N = \sum_{i=1}^{G} L_i.$$

We will assume that L_1, L_2, ..., L_G are fixed for a given linecard arrangement.

During every frame of N time-slots each sending linecard needs to be connected exactly once to each of the N receiving linecards. Similarly, each receiving linecard needs to be connected exactly once to each of the N sending linecards. Furthermore, in every time-slot, each sending linecard cannot connect to more than one receiving linecard, and vice-versa.

Put mathematically, if sending linecard i is connected to receiving linecard T_{ij} in time-slot j, then:

$$\begin{cases} T_{ij'} \neq T_{ij} & \text{for all } j' \neq j \\ T_{i'j} \neq T_{ij} & \text{for all } i' \neq i \\ T_{ij} \in \{1, ..., N\} & \text{for all } i, j \end{cases}$$

We'll call T the *linecard schedule*. T is a Latin square, i.e. the numbers from 1 to N appears exactly once in every row and every column. We will refer to a time-slot as a column.

For instance, let's assume that $L_1 = 3$, $L_2 = 2$, and $L_3 = 2$ (i.e., $G=3$, $N=7$). Then the following is a linecard schedule:

$$T = \left(\begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 2 & 3 & 4 & 5 & 6 & 7 & 1 \\ 3 & 4 & 5 & 6 & 7 & 1 & 2 \\ \hline 4 & 5 & 6 & 7 & 1 & 2 & 3 \\ 5 & 6 & 7 & 1 & 2 & 3 & 4 \\ \hline 6 & 7 & 1 & 2 & 3 & 4 & 5 \\ 7 & 1 & 2 & 3 & 4 & 5 & 6 \end{array} \right)$$

The last constraint arises from the use of MEMS switches in the hybrid optical-electrical switch fabric. Let L_i represent the number of linecards in group i. The rate needed between group i and group j is equal to

$$(L_i \cdot 2R) \cdot (L_j/N), \text{ where } 1 \leq i, j \leq G.$$

This is because the incoming traffic is spread uniformly over all N receiving linecards, and group j receives a portion (L_j/N) of this traffic. As assumed above, two groups can only communicate at a rate up to $2R$ through any single MEMS switch. Therefore, the minimum number of MEMS switches between group i and group j is:

$$\left\lceil \frac{L_i \cdot 2R \cdot L_j}{N} \cdot \frac{1}{2R} \right\rceil = \left\lceil \frac{L_i \cdot L_j}{N} \right\rceil.$$

We will call this the *MEMS constraint*.

Matrix T above doesn't meet the MEMS constraint because the maximum number of connections allowed between group 1 and group 1 at any time-slot is $\lceil \frac{3 \cdot 3}{7} \rceil = 2$. Similarly the second and third groups also don't meet the constraint.

IV. Number of MEMS Switches Needed for a Linecard Schedule

The following theorem shows how many MEMS switches are needed in order to build a linecard schedule that satisfies the MEMS constraint.

Theorem 1: We need at least

$$\alpha = \sum_{j=1}^{G} \left\lceil \frac{L \cdot L_j}{N} \right\rceil \leq L + G - 1$$

static MEMS switches in order to build a linecard schedule that satisfies the MEMS constraint, where $L = \max_i(L_i)$.

Proof: A MEMS switch can connect a sending group to at most one receiving group, and the minimum number of MEMS switches needed to connect sending group i to all receiving groups is:

$$\sum_{j=1}^{G} \left\lceil \frac{L_i \cdot L_j}{N} \right\rceil.$$

In particular, assume that the largest group has $L = \max_i(L_i)$ linecards. Then the total number of MEMS switches needed by the largest group to connect to all receiving groups is at least:

$$\alpha = \sum_{j=1}^{G} \left\lceil \frac{L \cdot L_j}{N} \right\rceil < \sum_{j=1}^{G} \left(\frac{L \cdot L_j}{N} + 1 \right) = L + G.$$

Because α, L and G are integers, $\alpha \leq L + G - 1$.

Hence we need at most $L + G - 1$ static MEMS switches to create a uniform mesh with any linecard arrangement. ■

In our example with $L_1 = 3$, $L_2 = 2$, and $L_3 = 2$,

$$\alpha = \left\lceil \frac{3 \cdot 3}{7} \right\rceil + \left\lceil \frac{3 \cdot 2}{7} \right\rceil + \left\lceil \frac{3 \cdot 2}{7} \right\rceil = 4.$$

It is clear that $\alpha \leq L + G - 1 = 5$. Using a different linecard configuration where $L = 3$ and $G = 3$, it is also possible to reach the upper bound, for instance with $L_1 = 3$, $L_2 = 3$, and $L_3 = 2$.

V. Valid Schedules

A. Linecard Schedule

In this section, we will find an algorithm that works with exactly α MEMS switches.

We will introduce different types of schedules to help clarify the presentation of the linecard schedule solution. The three new schedules are (1) linecard-to-linecard, (2) linecard-to-group, and (3) group-to-group schedules. As described in the definitions below, the first part of the schedule name represents whether the schedule determines the specific sending linecards or only the sending groups, and the second part of the name specifies whether the schedule determines the specific receiving linecards or only the receiving groups. For instance, a linecard-to-group schedule will determine which linecard will send to which receiving group in each column.

Definition 1: A *linecard-to-linecard* (L-L) schedule T is a matrix with N rows corresponding to the N sending linecards, N columns corresponding to the N time-slots of the frame, and one receiving linecard index per row-column intersection.

Note that a linecard-to-linecard schedule is the same as a linecard schedule.

Definition 2: An L-L schedule T is said to be *valid* iff a receiving linecard appears exactly once in every row and column of T, and at most $\left\lceil \frac{L_i \cdot L_j}{N} \right\rceil$ receiving linecards from group j are connected to sending linecards from group i in any column of T (MEMS constraint).

In other words, T is a valid L-L schedule if it is a Latin square satisfying the MEMS constraints. Here is an example of a L-L schedule which is valid.

$$T = \left(\begin{array}{ccccccc} 1 & 4 & 2 & 6 & 3 & 5 & 7 \\ 2 & 6 & 1 & 5 & 7 & 3 & 4 \\ 4 & 3 & 7 & 1 & 5 & 6 & 2 \\ \hline 5 & 1 & 3 & 4 & 2 & 7 & 6 \\ 7 & 5 & 4 & 3 & 6 & 2 & 1 \\ \hline 3 & 2 & 6 & 7 & 1 & 4 & 5 \\ 6 & 7 & 5 & 2 & 4 & 1 & 3 \end{array} \right)$$

Notice that the MEMS constraint used in Definition 2 applies to groups, not linecards. For instance, the example matrix T is not allowed to have more than two receiving linecards from the first group in the first three rows in any column. Therefore, in order to build L-L schedules, we cannot only consider the constraints on linecards, but also need to take into account the constraints on groups. The MEMS constraint makes the linecard schedule problem non-trivial.

We will show that it is possible to build a valid *group-to-group* schedule that only considers constraints on groups, and then successively build a valid *linecard-to-group* schedule and finally a valid *linecard-to-linecard* schedule which incorporates the constraints on linecards. We will define and provide examples for these schedules below.

B. Linecard-to-Group Schedule

Definition 3: A *linecard-to-group* (L-G) schedule U is a matrix with N rows corresponding to the N sending linecards, N columns corresponding to the N time-slots of the frame, and one letter per row-column intersection corresponding to the receiving group.

Definition 4: An L-G schedule U is said to be *valid* iff the i^{th} letter appears exactly L_i times in each row and each column, and at most $\left\lceil \frac{L_i \cdot L_j}{N} \right\rceil$ times in the linecards of group i in any column of U (MEMS constraint).

Here is an example of a valid L-G schedule.

$$U = \left(\begin{array}{ccccccc} A & B & A & C & A & B & C \\ A & C & A & B & C & A & B \\ B & A & C & A & B & C & A \\ \hline B & A & A & B & A & C & C \\ C & B & B & A & C & A & A \\ \hline A & A & C & C & A & B & B \\ C & C & B & A & B & A & A \end{array}\right)$$

Notice that matrix U is the same as matrix T except that the receiving linecard indices are replaced with the letters corresponding to the receiving linecard group.

C. Group-to-Group Schedule

Definition 5: A *group-to-group* (G-G) schedule V is a matrix with G rows corresponding to the G sending linecard groups, N columns corresponding to the N time-slots of the frame, and L_i letters per row-column intersection in row i.

Definition 6: A G-G schedule V is said to be *valid* iff the i^{th} letter appears exactly $L_i \cdot L_j$ times in each row j (corresponding to sending group j), L_i times in each column, and at most $\left\lceil \frac{L_i \cdot L_j}{N} \right\rceil$ times in any row-column intersection in row i (MEMS constraint).

Here is an example of a valid G-G schedule.

$$V = \left(\begin{array}{ccccccc} AAB & ABC & AAC & ABC & ABC & ABC & ABC \\ \hline BC & AB & AB & AB & AC & AC & AC \\ \hline AC & AC & BC & AC & AB & AB & AB \end{array}\right)$$

Notice that one can get matrix V by grouping together the rows corresponding to the same group in matrix U.

D. Schedule Equivalence Theorem

Given a valid L-L schedule, we can easily deduce a valid L-G schedule, and then a valid G-G schedule. However, it is not obvious how to create a valid L-L schedule from a valid G-G schedule. The following theorem, which is proved in the appendix, shows that we can.

Theorem 2: Consider the following three schedules:

(i) A valid linecard-to-linecard (L-L) schedule T

(ii) A valid linecard-to-group (L-G) schedule U

(iii) A valid group-to-group (G-G) schedule V

Given one schedule we can create the other two: (L-L)⇔(L-G)⇔(G-G).

In the next section, we will show how to construct a valid G-G schedule, hence proving that it is always possible to obtain a linecard schedule that satisfies the MEMS constraint.

VI. Constructing a Valid G-G Schedule

A. Algorithm for Constructing a Valid G-G Schedule

We will now construct an algorithm that recursively builds a valid group schedule time-slot after time-slot, for the N time-slots of the frame. We will then show in the appendix that the algorithm finds a valid solution, and that it has a polynomial complexity.

At the start:

Let t be the number of time-slots left to schedule after each iteration. At the start, $t = N$, since all the time-slots are unscheduled. Also, let $M \equiv M^t = M^N$ be the initial matrix of all the elements that need to be scheduled. Its rows represent the sending groups, its columns the receiving groups (letters "A", "B", ...). At the start, for all i, j, $M_{ij} = L_i \cdot L_j$, i.e. there are $L_i \cdot L_j$ connections to schedule from sending group i to receiving group j during the whole frame.

Iteratively:

For $t = N, N-1, ..., 1$, proceed as follows.

1) For each i, j, do the decomposition of M^t_{ij} in base t: $M^t_{ij} = P^t_{ij} \cdot t + Q^t_{ij}$ (i.e. $P^t = \left\lfloor \frac{1}{t} M^t \right\rfloor$, $Q^t = M^t - P^t \cdot t$). In this iteration, we will start by scheduling P^t, and then consider the remainder Q^t and schedule a part of it such that all the constraints are satisfied.
2) Define the vectors a^t and b^t such that

$$\begin{cases} a^t_i = \frac{\sum_{j'=1}^{G} Q^t_{ij'}}{t} & \text{for all } i \\ b^t_j = \frac{\sum_{i'=1}^{G} Q^t_{i'j}}{t} & \text{for all } j \end{cases}$$

   a^t and b^t are integer vectors (cf proof).
3) Find a 0-1 matrix $R^t \leq Q^t$ such that:

$$\begin{cases} \sum_{j'=1}^{G} R^t_{ij'} = a^t_i & \text{for all } i \\ \sum_{i'=1}^{G} R^t_{i'j} = b^t_j & \text{for all } j \\ R^t_{ij} \in \{0, 1\} & \text{for all } i, j \end{cases}$$

   The proof in the appendix shows that R^t exists (it uses graph theory for proof of existence, and the Ford-Fulkerson max-flow algorithm for building it).
4) Use the schedule $S^t = P^t + R^t$ for this time-slot. Update $M^{t-1} = M^t - S^t$.

B. Example

We build the matrix V given the schedules, S^t, provided in Table I. More specifically, S^t_{ij} represents the number of occurrences of the j^{th} letter in the i^{th} row in column $N - t + 1$ of matrix V. For instance, the schedule

$$S^7 = \left(\begin{array}{ccc} 2 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{array}\right)$$

helps us create the first column of V having two A's and one B in the first row, one B and one C in the second row, and one A and one C in the last row. S^6 will determine the second column, S^5 will determine the third column, and so on. The resulting matrix is

$$V = \left(\begin{array}{ccccccc} AAB & ABC & AAC & ABC & ABC & ABC & ABC \\ \hline BC & AB & AB & AB & AC & AC & AC \\ \hline AC & AC & BC & AC & AB & AB & AB \end{array}\right)$$

TABLE I

EXAMPLE OF APPLICATION OF THE ALGORITHM

$$M^7 = \begin{pmatrix} 9 & 6 & 6 \\ 6 & 4 & 4 \\ 6 & 4 & 4 \end{pmatrix}, P^7 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, Q^7 = \begin{pmatrix} 2 & 6 & 6 \\ 6 & 4 & 4 \\ 6 & 4 & 4 \end{pmatrix}, R^7 = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}, S^7 = \begin{pmatrix} 2 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}.$$

$$M^6 = \begin{pmatrix} 7 & 5 & 6 \\ 6 & 3 & 3 \\ 6 & 4 & 3 \end{pmatrix}, P^6 = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, Q^6 = \begin{pmatrix} 1 & 5 & 0 \\ 0 & 3 & 3 \\ 6 & 4 & 3 \end{pmatrix}, R^6 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}, S^6 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}.$$

$$M^5 = \begin{pmatrix} 6 & 4 & 5 \\ 5 & 2 & 3 \\ 4 & 4 & 2 \end{pmatrix}, P^5 = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, Q^5 = \begin{pmatrix} 1 & 4 & 0 \\ 0 & 2 & 3 \\ 4 & 4 & 2 \end{pmatrix}, R^5 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}, S^5 = \begin{pmatrix} 2 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}.$$

$$M^4 = \begin{pmatrix} 4 & 4 & 4 \\ 4 & 1 & 3 \\ 4 & 3 & 1 \end{pmatrix}, P^4 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}, Q^4 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 3 \\ 0 & 3 & 1 \end{pmatrix}, R^4 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, S^4 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}.$$

Finally, for t=3,2,1: $M^t = \begin{pmatrix} t & t & t \\ t & 0 & t \\ t & t & 0 \end{pmatrix} = t \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}, R^t = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$ and $S^t = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$.

VII. VALID L-L SCHEDULE

A. From a Valid G-G Schedule to a Valid L-G Schedule

We will now construct an algorithm that successively builds a valid L-G schedule given a valid G-G schedule, and then a valid L-L schedule given a valid L-G schedule. This algorithm will be used in the appendix to prove Theorem 2. In this section, we will transform the valid G-G schedule described in Section V into a valid L-G schedule.

For each $1 \leq j \leq G$, consider row j in V. In our example, the first row is:

$$(\ AAB \quad ABC \quad AAC \quad ABC \quad ABC \quad ABC \quad ABC\ )$$

We want to subdivide each row j into L_j sub-rows, corresponding to the subdivision of each sending group j into L_j sending linecards, thus forming a valid L-G schedule.

First, each letter has $L_i \cdot L_j$ occurrences in any given row of V. Arbitrarily divide them into L_i subscripted letters ("sub-letters") of L_j elements. In our example, we transform the letters of V into N arbitrarily assigned sub-letters $(A_1, A_2, A_3, B_1, B_2, C_1, C_2)$. For instance, since A appears 9 times in the first row, we replace the $A's$ arbitrarily with 3 A_1's, 3 A_2's and 3 A_3's:

$(A_1A_1B_1;\ A_1B_1C_1;\ A_2A_2C_1;\ A_2B_1C_1;\ A_3B_2C_2;\ A_3B_2C_2;\ A_3B_2C_2)$

In row j of matrix V, each of the N sub-letters has L_j occurrences, and each of the N columns has L_j elements. Let's form a new matrix that has sub-letters as inputs and columns as outputs. In this new matrix, all columns and all rows have L_j elements. In our example, the new matrix for the first row of V is :

	$col.1$	$col.2$	$col.3$	$col.4$	$col.5$	$col.6$	$col.7$
A_1	2	1	0	0	0	0	0
A_2	0	0	2	1	0	0	0
A_3	0	0	0	0	1	1	1
B_1	1	1	0	1	0	0	0
B_2	0	0	0	0	1	1	1
C_1	0	1	1	1	0	0	0
C_2	0	0	0	0	1	1	1

We can now apply the Birkhoff-von Neumann decomposition theorem to this matrix, by decomposing it into a sum of L_j permutations [4], [5].[2] We obtain L_j permutations. By reading column after column, each of these permutations gives a sequence of sub-letters that corresponds to a row of the desired L-G schedule. Therefore, the L_j permutations yield the L_j rows of the L-G schedule corresponding to group j. In our example, the first permutation could be:

	$col.1$	$col.2$	$col.3$	$col.4$	$col.5$	$col.6$	$col.7$
A_1	1	0	0	0	0	0	0
A_2	0	0	1	0	0	0	0
A_3	0	0	0	0	1	0	0
B_1	0	1	0	0	0	0	0
B_2	0	0	0	0	0	1	0
C_1	0	0	0	1	0	0	0
C_2	0	0	0	0	0	0	1

,

yielding the first row of:

$$\begin{pmatrix} A_1 & B_1 & A_2 & C_1 & A_3 & B_2 & C_2 \\ A_1 & C_1 & A_2 & B_1 & C_2 & A_3 & B_2 \\ B_1 & A_1 & C_1 & A_2 & B_2 & C_2 & A_3 \end{pmatrix}$$

We finally replace each sub-letter by the corresponding letter, and get the valid L-G schedule. Upon examination of the algorithm, it is clear that we only permute letters within the same

[2]Because all elements are integers we could use graph-coloring instead [6][7][8][9].

column of the same sending group, thus yielding a valid L-G schedule. In our example, the resulting L-G schedule is:

$$U = \left(\begin{array}{ccccccc} A & B & A & C & A & B & C \\ A & C & A & B & C & A & B \\ B & A & C & A & B & C & A \\ \hline B & A & A & B & A & C & C \\ C & B & B & A & C & A & A \\ \hline A & A & C & C & A & B & B \\ C & C & B & A & B & A & A \end{array}\right)$$

B. From a Valid L-G Schedule to a Valid L-L Schedule

In the previous section, we constructed a valid L-G schedule given a valid G-G schedule. In this section, we will transform the valid L-G schedule into a valid L-L schedule.

We apply the Birkhoff-von Neumann theorem (or graph-coloring) for each letter. First, we replace each *A* with a "1", and every other letter with a "0". For our example, we get:

$$\left(\begin{array}{ccccccc} A & 0 & A & 0 & A & 0 & 0 \\ A & 0 & A & 0 & 0 & A & 0 \\ 0 & A & 0 & A & 0 & 0 & A \\ \hline 0 & A & A & 0 & A & 0 & 0 \\ 0 & 0 & 0 & A & 0 & A & A \\ \hline A & A & 0 & 0 & A & 0 & 0 \\ 0 & 0 & 0 & A & 0 & A & A \end{array}\right)$$

$$\rightarrow \left(\begin{array}{ccccccc} 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ \hline 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ \hline 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{array}\right)$$

We then decompose the above matrix into the sum of L_i different permutations, such that the l^{th} permutation will indicate at which times linecard l is scheduled. Since there are exactly L_1 ones (corresponding to the L_1 *A*'s) in each row and column, this is possible by Birkhoff-von Neumann. In our example, we can decompose the above matrix into the sum of three permutations:

$$\left(\begin{array}{ccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{array}\right) + \left(\begin{array}{ccccccc} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \hline 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{array}\right)$$

$$+ \left(\begin{array}{ccccccc} 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ \hline 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right)$$

Applying this method to each letter, we then create a valid L-L schedule, with exactly one occurrence of each receiving linecard index in each row and column. In our example, we get the following valid L-L schedule, hence concluding the construction process:

$$T = \left(\begin{array}{ccccccc} 1 & 4 & 2 & 6 & 3 & 5 & 7 \\ 2 & 6 & 1 & 5 & 7 & 3 & 4 \\ 4 & 3 & 7 & 1 & 5 & 6 & 2 \\ \hline 5 & 1 & 3 & 4 & 2 & 7 & 6 \\ 7 & 5 & 4 & 3 & 6 & 2 & 1 \\ \hline 3 & 2 & 6 & 7 & 1 & 4 & 5 \\ 6 & 7 & 5 & 2 & 4 & 1 & 3 \end{array}\right)$$

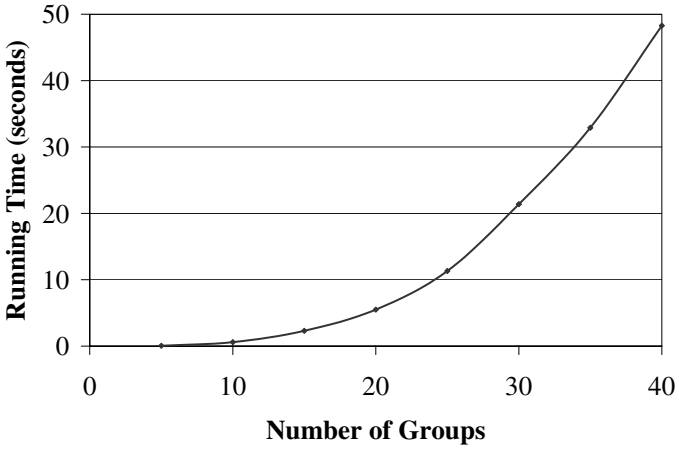

Fig. 4. Running time of our implementation of the algorithm. Times are averaged over 100 runs for each value of G.

VIII. Practical Considerations

To estimate how long the algorithm takes to run in practice, a program was implemented using the 'C' programming language (source code and detailed description available at [10]). Given an arrangement of linecards and groups, the program finds a valid G-G schedule to configure the MEMS switches and electronic crossbars.

We ran the program on a Pentium III operating at 1GHz, with different values of G, N and L, up to maximum values $N = 640$, $L = 16$ and $G = 40$. In each successive run, the placement of linecards in the rack was picked uniformly at random. The running times, averaged over 100 runs per value of G, are shown in Figure 4.

Our implementation runs too slowly to pick a new configuration in real-time when a linecard is added, removed or fails. The typical requirement would be that the router be up and running again with 50ms of a change, whereas with $N = 640$ the algorithm took up to 50 seconds to complete. Although our implementation is not ideally optimized, it is unlikely to run fast enough to make real-time decisions.

In practice, we could run the algorithm in advance and store the results. When a certain group of linecards is present, we can pre-calculate all the configurations that differ by one or two linecards, or one or two groups (e.g. if a whole rack is powered down, or fails). Alternatively, but less likely, the algorithm could be implemented to run very quickly in a custom ASIC. The algorithm lends itself to fine-level parallelism, especially in the parallel Birkhoff-von Neumann decompositions, and would run a lot faster than in software.

IX. Conclusions

In this paper, we showed how it is possible to reconfigure the load-balanced switch when one or more linecards is missing or has failed. We found that using the architecture described in [1], we need at most $L + G - 1$ static MEMS switches in order to build a linecard schedule that satisfies the MEMS constraint, where G is the number of groups and L is the maximum number of linecards per group. We then described a polynomial-time algorithm to configure the packet transmissions and the

MEMS switches, and proved that it is not only necessary, but also sufficient to use α static MEMS. This was done based on edge coloring algorithms in a regular bipartite graph and the Ford-Fulkerson max-flow algorithm.

X. Acknowledgements

The authors would like to thank Professors Balaji Prahbakar and Yinyu Ye for useful discussions; and Mingjie Lin for implementing the algorithm used to generate results in Figure 4.

References

[1] Isaac Keslassy, Shang-Tse Chuang, Kyoungsik Yu, David Miller, Mark Horowitz, Olav Solgaard, Nick McKeown, "Scaling Internet routers using optics," *ACM SIGCOMM 2003*, Karlsruhe, Germany, Sep. 2003.

[2] C.S. Chang, D.S. Lee and Y.S. Jou, "Load balanced Birkhoff-von Neumann switches, part I: one-stage buffering," *IEEE HPSR '01*, Dallas, May 2001.

[3] P. Bernasconi, C. Doerr, C. Dragone, M. Capuzzo, E. Laskowski and A. Paunescu, "Large N x N waveguide grating routers," *Journal of Lightwave Technology*, Vol. 18, No. 7, pp. 985-991, July 2000.

[4] C.S. Chang, J.W. Chen, and H.Y. Huang, "On service guarantees for input-buffered crossbar switches: a capacity decomposition approach by Birkhoff and Von Neumann," *IEEE IWQoS, London*, 1999.

[5] G. D. Birkhoff, "Tres observaciones sobre el algebra lineal," *Universidad Nacional de Tucuman Revista*, Serie A, vol. 5, pp. 147-151, 1946.

[6] R. Cole, K. Ost and S. Schirra, "Edge-coloring bipartite multigraphs in O(E log D) time," *Combinatorica*, vol. 21, pp. 5-12, 2001.

[7] R. Cole, K. Ost and S. Schirra, "Edge-coloring bipartite multigraphs in O(E log D) time," *New York University Technical Report NYU-TR1999-792*, New York, Sep. 1999.

[8] A. Schrijver, "Bipartite edge-coloring in O(Δm) time," *SIAM J. Comput.*, vol. 28, pp. 841-846, 1999.

[9] N. Alon, "A simple algorithm for edge-coloring bipartite multigraphs," *Information Processing Letters*, vol. 85, issue 6, pp. 301-302, March 2003.

[10] Switch configuration algorithm, available at http://yuba.stanford.edu/or/SwitchConfig.c

[11] A. Schrijver, "A course in combinatorial optimization," available at http://www.cwi.nl/~lex/files/dict.ps, Feb. 2003.

[12] L.R. Ford and D.R. Fulkerson, *Flows in Networks*, Princeton University Press, 1962.

Appendix I
Proof for Theorem 2

First, as shown in Section V, it is easy to successively build a valid L-G schedule from a valid L-L schedule and a valid G-G schedule from a valid L-G schedule.

On the other hand, Section VII shows the algorithm which successively constructs a valid L-G schedule from a valid G-G schedule and a valid L-L schedule from a valid L-G schedule. This is true as long as we can decompose the matrix into a sum of permutations (for example, using a Birkhoff-von Neumann decomposition or graph-coloring), which we can always do when the sums on each row and each column are equal [4], [5].

Appendix II
Proofs for the Construction of the Valid G-G Schedule

Proof: Assume that for a given t,

$$\begin{cases} \sum_{j'=1}^{G} M_{ij'}^t = L_i t & \text{for all } i \\ \sum_{i'=1}^{G} M_{i'j}^t = L_j t & \text{for all } j \\ 0 \le M_{ij}^t \le \left\lceil \frac{L_i \cdot L_j}{N} \right\rceil t & \text{for all } i,j \end{cases}$$

This is obviously true for $t = N$ by definition of M^t. We will prove at the end of the proof that if we assume it for t, it's also true for $t-1$.

First, since $Q^t = M^t - P^t$, using the definition of P^t we get:

$$\begin{cases} (\sum_{j'=1}^{G} Q_{ij'}^t) \bmod t = 0 & \text{for all } i \\ (\sum_{i'=1}^{G} Q_{i'j}^t) \bmod t = 0 & \text{for all } j \\ 0 \le Q_{ij}^t \le t-1 & \text{for all } i,j \end{cases}$$

Therefore, a^t and b^t are integer vectors.

Second, from the definition of a^t and b^t, they both have the same sum - call it σ.

Third, let's prove that there exists a matrix R^t satisfying the conditions above. In order to prove this, we use exercise 3.13 in the paper by A. Schrijver [11]. It tells us that R^t exists iff for each subset s_1 of the rows and for each subset s_2 of the columns, $\sigma + |E(s_1,s_2)| \ge \sum_{i\in s_1} a_i + \sum_{j\in s_2} b_j$, where $|E(s_1,s_2)|$ denotes the number of non-zero elements Q_{ij}^t with $i \in s_1, j \in s_2$. But we know that

$$\sum_{i\in s_1, j\in s_2} Q_{ij}^t = \sum_{i\in s_1} Q_{ij}^t - \sum_{i\in s_1, j\in {s_2}^C} Q_{ij}^t$$

$$\sum_{i\in s_1, j\in s_2} Q_{ij}^t \ge t\sum_{i\in s_1} a_i - \sum_{j\in {s_2}^C} Q_{ij}^t$$

$$\sum_{i\in s_1, j\in s_2} Q_{ij}^t \ge t\sum_{i\in s_1} a_i - t\sum_{j\in {s_2}^C} b_j$$

In addition, since $t - 1 \ge Q_{ij}^t$,

$$\sum_{i\in s_1, j\in s_2} t\delta_{Q_{ij}^t \ge 0} \ge \sum_{i\in s_1, j\in s_2} Q_{ij}^t,$$

therefore

$$\sum_{i\in s_1, j\in s_2} t\delta_{Q_{ij}^t \ge 0} \ge t\sum_{i\in s_1} a_i - t\sum_{j\in {s_2}^C} b_j.$$

Since $|E(s_1,s_2)| = \sum_{i\in s_1, j\in s_2} \delta_{Q_{ij}^t \ge 0}$,

$$|E(s_1,s_2)| \ge \sum_{i\in s_1} a_i - \sum_{j\in {s_2}^C} b_j = \sum_{i\in s_1} a_i + \sum_{j\in s_2} b_j - \sigma.$$

Thus R^t exists. Note that it is possible to construct it in polynomial time using Ford-Fulkerson (see next section).

Fourth, let's show that the resulting schedule S^t will satisfy the MEMS constraint, i.e. for all i, j,

$$S_{ij}^t \le \left\lceil \frac{L_i \cdot L_j}{N} \right\rceil.$$

We know that

$$S_{ij}^t = P_{ij}^t + R_{ij}^t = \left\lfloor \frac{1}{t} M_{ij}^t \right\rfloor + R_{ij}^t,$$

$$R_{ij}^t \le Q_{ij}^t = M_{ij}^t - t\left\lfloor \frac{1}{t} M_{ij}^t \right\rfloor,$$

$$R_{ij}^t \le 1,$$

and

$$M_{ij}^t \leq \left\lceil \frac{L_i \cdot L_j}{N} \right\rceil t.$$

Distinguish two cases regarding this last inequa equality is an equality,

$$M_{ij}^t = \left\lceil \frac{L_i \cdot L_j}{N} \right\rceil t,$$

so $R_{ij}^t \leq Q_{ij}^t = 0$ and $S_{ij}^t = M_{ij}^t/t = \left\lceil \frac{L_i \cdot L_j}{N} \right.$ this inequality is strict,

$$M_{ij}^t < \left\lceil \frac{L_i \cdot L_j}{N} \right\rceil t,$$

so there exists some $\epsilon > 0$ such that

$$M_{ij}^t/t = \left\lceil \frac{L_i \cdot L_j}{N} \right\rceil - \epsilon = (\left\lceil \frac{L_i \cdot L_j}{N} \right\rceil - 1)$$

Thus,

$$\left\lfloor \frac{1}{t} M_{ij}^t \right\rfloor = \left\lceil \frac{L_i \cdot L_j}{N} \right\rceil - 1 + \lfloor 1 - \epsilon \rfloor \leq \left\lceil \frac{L_i}{N} \right\rceil - 1$$

and

$$S_{ij}^t \leq \left\lfloor \frac{1}{t} M_{ij}^t \right\rfloor + R_{ij}^t \leq \left\lfloor \frac{1}{t} M_{ij}^t \right\rfloor + 1 \leq \left\lceil \frac{L_i \cdot L_j}{N} \right\rceil.$$

Note that in both cases

$$S_{ij}^t \leq \left\lceil \frac{L_i \cdot L_j}{N} \right\rceil.$$

Fifth, let's complete the recursion hypothesis and show that:

$$\begin{cases} \text{(i) } \sum_{j'=1}^{G} M_{ij'}^{t-1} = L_i(t-1) & \text{for all } i \\ \text{(ii) } \sum_{i'=1}^{G} M_{i'j}^{t-1} = L_j(t-1) & \text{for all } j \\ \text{(iii) } 0 \leq M_{ij}^{t-1} \leq \left\lceil \frac{L_i \cdot L_j}{N} \right\rceil (t-1) & \text{for all } i, j \end{cases}$$

We'll use the assumptions on M stated at the start of the proof, and the definition $M^{t-1} = M^t - S^t$.

Let's first prove (i) by showing that

$$\sum_{j'=1}^{G} S_{ij'}^t = L_i$$

(the proof for (ii) is similar). By definition,

$$\sum_{j'=1}^{G} S_{ij'}^t = \sum_{j'=1}^{G} (P_{ij'}^t + R_{ij'}^t),$$

and

$$\sum_{j'=1}^{G} R_{ij'}^t = a_i = (\sum_{j'=1}^{G} Q_{ij'}^t)/t,$$

thus

$$\sum_{j'=1}^{G} S_{ij'}^t = \sum_{j'=1}^{G} (tP_{ij'}^t + Q_{ij'}^t)/t = \sum_{j'=1}^{G} M_{ij'}^t/t = L_i.$$

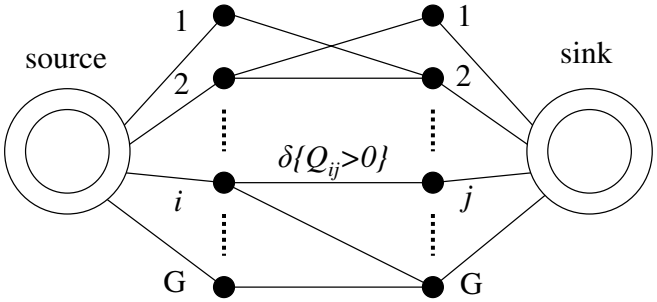

Fig. 5. Illustration of Ford-Fulkerson Construction.

Let's now prove (iii). We know that $M_{ij}^t \leq \left\lceil \frac{L_i \cdot L_j}{N} \right\rceil t$, therefore $P_{ij}^t \leq \left\lceil \frac{L_i \cdot L_j}{N} \right\rceil$. Also we can decompose M_{ij}^t in base t as $M_{ij}^t = P_{ij}^t t + Q_{ij}^t$, and

$$\begin{aligned} M_{ij}^{t-1} &= M_{ij}^t - S_{ij}^t \\ &= (P_{ij}^t t + Q_{ij}^t) - (P_{ij}^t + R_{ij}^t) \\ &= P_{ij}^t (t-1) + (Q_{ij}^t - R_{ij}^t). \end{aligned}$$

Distinguish two cases. In the case where

$$P_{ij}^t = \left\lceil \frac{L_i \cdot L_j}{N} \right\rceil,$$

then

$$M_{ij}^t = P_{ij}^t t, \;\; Q_{ij}^t = 0, \;\; R_{ij}^t = 0,$$

thus

$$M_{ij}^{t-1} = P_{ij}^t (t-1) = M_{ij}^t \frac{t-1}{t} \leq \left\lceil \frac{L_i L_j}{N} \right\rceil (t-1).$$

Otherwise, in the case where

$$P_{ij}^t \leq \left\lceil \frac{L_i \cdot L_j}{N} \right\rceil - 1,$$

we have

$$\begin{aligned} M_{ij}^{t-1} &= P_{ij}^t (t-1) + (Q_{ij}^t - R_{ij}^t) \\ &\leq (\left\lceil \frac{L_i \cdot L_j}{N} \right\rceil - 1)(t-1) + Q_{ij}^t \\ &\leq \left\lceil \frac{L_i \cdot L_j}{N} \right\rceil (t-1) - (t-1) + (t-1) \\ &= \left\lceil \frac{L_i L_j}{N} \right\rceil (t-1), \end{aligned}$$

because $Q_{ij}^t \leq t-1$ as shown before. Hence (iii) is correct in both cases and the three properties are proven by recurrence.

Finally, note that all parts of the algorithm are done in polynomial time, and thus the algorithm is also in polynomial time. ■

APPENDIX III
FORD-FULKERSON ALGORITHM

As explained in the section before, it is possible to use Ford-Fulkerson's max-flow algorithm [12] in order to construct R^t, and therefore the schedules S^t. More specifically, as illustrated in Figure 5, construct the network as follows. There is one

source, G inputs, G outputs, and one sink. The source is connected to each input i with capacity a_i. Each input i is connected to each output j with capacity $\delta_{Q_{ij} \geq 1}$, i.e. capacity 1 if $Q_{ij} \geq 1$ and 0 otherwise. Finally, each output j is connected to the sink with capacity b_j. Since capacities are integer, the resulting flows will also be integers, and will thus yield a correct matrix R^t.