Rate Quantization and Service Quality for Variable Rate Traffic over Single Crossbar Switches

Can Emre Koksal, Robert G. Gallager and Charles E. Rohrs

Abstract— **We study the provision of deterministic rate guarantees over single crossbar switches. Birkhoff decomposition yields a general approach for this problem, but the required complexity can be very high and the quality of service can be unsatisfactory for practical traffic sources.**

We develop a method called *rate quantization* which converts the set of desired rates into a certain discrete set in such a way that the complexity and the quality of service guarantees can be greatly improved over a Birkhoff switch. The rate quantization algorithm we introduce works with any *resource speedup* greater than 1 and it satisfies certain fairness criteria.Moreover, rate quantization enables us to develop a Slepian-Duguid-like algorithm with which the switch can both adapt to dynamically varying traffic and simplify switch scheduling significantly.

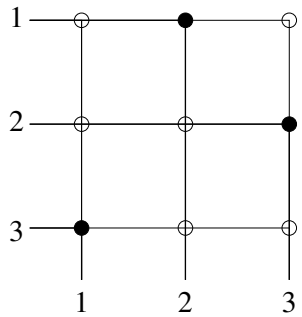

Fig. 1. A 3×3 crossbar fabric. Crosspoints are set to connect the lines to enable end to end connections. For instance, input 1 and output 2 are connected through the corresponding crosspoint.

I. Introduction

The core of a packet switch is composed of a switching fabric and input and output units. The function of the fabric is to set up connections between the input and output units. The function of the input units is to segment the incoming packets into equal size cells and provide buffering if necessary. The output units convert cells to packets with the requisite buffering.

The fabric is assumed to operate synchronously in the sense that time is segmented into *service time slots* and at most one cell from an input unit can be transferred to an output unit during a service time slot. The fabric is also assumed to be non-blocking, which means that a connection can always be made between any free input unit and free output unit in a service time slot.

The most popular non-blocking fabric is the crossbar. A crossbar fabric can be thought of as a set of lines, one for each input and each output, and a crosspoint that connects each input-output line pair as illustrated in Fig. 1. The most important limitation of a crossbar is the crossbar constraint which stipulates that in any service time slot at most one input can be matched with any given output and at most one output can be matched with any given input.

We assume that the input link to each input unit and the output link from each output unit is constrained to the same data rate. We define a *link time slot* as the required time for the data in a cell to travel over one of these links. A service time slot is not necessarily identical to a link time slot since in practice the crossbar may transfer $S \geqslant 1$ cells per link time slot. The factor S is the amount of resource speedup and need not be an integer.

In a packet switch, packets destined for the same output may arrive almost simultaneously at different inputs. Since only one cell per service time slot may be transferred to the output, the waiting packets and cells must be buffered somewhere in the switch. This form of congestion is unavoidable in a packet switch and dealing with it often represents the greatest source of complexity in the switch architecture.

A brute force approach to avoiding contention in an $N \times N$ switch is to use a speedup of N, i.e., to place each cell directly into its desired output queue immediately upon arrival. Such switches are known as output queued (OQ) switches since queueing is necessary only at the output. Output queueing has long been considered an ideal way of constructing packet switched devices because of its theoretical performance: Output queued switches can provide the QoS of a multiplexer for any traffic pattern (arrival process). However, with ever increasing link rates, it is simply no longer possible to find random access memories (RAMs) with sufficiently fast access times to build OQ switches.

Input queueing does not have the scaling limitations of output queueing. In the input queueing architecture, the fabric can run at a single line rate with one read and one write operation per incoming packet. However, an input buffered architecture also presents some technical difficulties due to the limitations of the fabrics, such as the crossbar constraint. An example for such a difficulty is *head of line* (HOL) blocking. HOL blocking is caused when the packets at the head of multiple input queues are all destined to the same output. Assuming FIFO queue discipline, only one of these packets

The authors are with the MIT Electrical Engineering and Computer Science Department, Cambridge, MA 02139, USA. E-mail:{emre, gallager, crohrs}@mit.edu

can be transmitted at a time; the others block their input queues even if later packets in these queues are destined to other outputs.

Karol et.al. [1] showed that with such FIFO queueing at each input, the throughput of a switch for large N is limited to 58.6% for uniform random arrivals of input traffic.

One way of eliminating HOL blocking is to change the queueing structure at the input. Instead of keeping all the packets in a FIFO queue, a separate queue can be maintained for each source-destination pair. This scheme is known as virtual output queueing (VOQ). This queueing scheme overcomes the HOL blocking associated with FIFO input queueing while keeping its scalability advantage.

Combined Input and Output Queueing is a good compromise between the performance and scalability of both output and input queued switches. For input queued switches, at most one packet needs to be delivered to an output port in one unit of time, and for an output queued switch, up to N packets need to be delivered to an output in the same amount of time. Using CIOQ, instead of choosing one of these two extreme choices, we can choose a reasonable value in between 1 and N. This can be achieved by having buffers at both the input and output ports. IQ and CIOQ switches are employed in many high speed switch architectures, e.g., [2] - [3].

One key factor in achieving high performance using VOQ switches is the scheduling algorithm that is responsible for the selection of packets to be transmitted from the input queues to the output queues in each service time slot. There have been two fundamentally different approaches to this scheduling problem depending on the level at which the scheduling is done: cell scheduling and rate reservation based scheduling.

In the approaches based on cell scheduling ([2]-[11]), the problem of finding the appropriate connections between the inputs and the outputs of a crossbar is posed as a matching problem in a bipartite graph. Every service time slot, a request graph is generated. A request graph is composed of the edges, one for each non-empty VOQ. A scheduling algorithm chooses which edges shall be used for transmission of cells in the service time slot. Edges are assigned priorities (possibly identical) that are functions of the state of the system. They are updated every link time slot and the new matching which maximizes some objective function is found subject to the crossbar constraint. The objective may be to achieve a stable marriage match ([2], [7]), to achieve a maximal match ([8], [11]) or to maximize the sum of edge weights which are in the connect graph ([4], [5]). Edge weights for each VOQ are usually chosen to be the queue size ([2]-[5]), the delay experienced by the packet at the head of the queue ([2], [5]) or in some cases edge weights can be identical.

Cell scheduling algorithms tend to be instantaneous in the sense that they look for some useful matching at each slot. Therefore, they adapt to some extent to dynamically varying traffic patterns. They rely on congestion avoidance and control to avoid buffer overflow, and most of them depend on the use of traffic shapers for fairness. Other than for admission control purposes, they do not require any *a priori* information about the arrival processes. It was shown ([5]) that 100% throughput can be achieved with VOQ switches for all possible cell arrival processes in which all the input links are fully utilized and no output link is oversubscribed. It has also been shown that with a speedup of 2, work conservation and certain delay guarantees can be achieved with a maximal matching algorithm ([8]).

Rate reservation based algorithms were originally proposed for circuit switches in traditional voice networks to provide constant bit rate (CBR) guarantees for voice traffic that is rather static in nature. In that case, rate is reserved for very long durations. This kind of switching is also known as *multirate* circuit switching if desired rates between input output pairs (I-O pairs) are picked from a set of possible rates (e.g., certain fractions of link capacity rather than $\{0, 1\}$ as in full circuit switching).

Rate reservation based scheduling does not necessarily lead to a work conserving service, and in fact, at the time a crossbar configuration is set, all the VOQs which are supposed to take advantage of the connection to send a cell through the crossbar may be empty. Despite this, the configuration is still set. This fact reflects the fundamental difference between rate reservation based scheduling and cell scheduling. Cell scheduling works slot by slot; rate reservation provides guarantees using *a priori* knowledge. A number of rate reservation based scheduling algorithms have been proposed to provide rate guarantees for multirate circuit switches.

The BATCH-TSA algorithm proposed in [12] is a rate reservation based scheduling algorithm that guarantees bounded service lag. The algorithm treats the switch as a time division multiple access (TDMA) network and the problem of providing rate guarantees is translated into the link time slot assignment problem. Each flow is first stored in a queue for a period, say T link time slots. If at most T cells arrive at each input and at most T cells are destined to each output in the first time period, it was shown that all these packets can be transmitted in the next period.

The idling weighted round robin (WRR) algorithm [13] is fundamentally the same as BATCH-TSA but the method in which the packets are scheduled within a frame is different. In both of these approaches, the frame size is initially chosen by the algorithm. A large frame size implies a large delay, while a small frame size implies the set of rates for which the switch can provide bounded delay is very limited.

A more general approach to rate reservation based scheduling is the Birkhoff-von Neumann decomposition ([14], [15]). This approach eliminates the problem of choosing the frame size and provides uniform service guarantees for all admissible traffic. However, the worst case delay can be very high with this approach. Therefore, a higher (possibly much higher) rate than the long term average rate of a bursty, delay sensitive traffic stream must be allocated in order to satisfy its delay requirement. In addition, the complexity of Birkhoff's decomposition algorithm is $O(N^{4.5})$ for an $N \times N$ switch, and since the decomposition must be complete before the first configuration can be set, this algorithm cannot run in parallel with scheduling the crossbar configurations. As a result, for dynamically varying traffic such as the multimedia traffic, Birkhoff-von Neumann approach may not be satisfactory.

In this paper, we introduce *rate quantization* with which a VOQ switch can provide deterministic service guarantees like

the traditional rate reservation based scheduling algorithms and at the same time can handle the dynamically changing nature of input traffic like cell scheduling algorithms. Basically, rate quantization converts the set of desired rates into a certain discrete set which can then be used as an input to fairly simple scheduling algorithms.

Some speedup is necessary to support quantized rates. Any speedup between 1 and 2 is acceptable, and the speedup is a function of the "grade" of quantization which directly affects the quality of service provided by the switch. The following can be initially noted about rate quantization and the performance of an $N \times N$ switch with rate quantization. These statements will be clarified later on.

- Rate quantization can be used with very simple schedulers to improve the worst case delay approximately by a factor of N compared to Birkhoff-von Neumann switches even with very simple schedulers. Indeed with a speedup S for a leaky bucket constrained source of rate r, this delay is $\frac{N}{(S-1)r}$ over that for an output queued switch. This delay is a constant factor lower than the delay for such a source with the maximal matching algorithm given in [8] and a speedup of 2. However, we emphasize that rate quantization provides this QoS with any speedup between 1 and 2.
- With rate quantization, the schedule generated by any scheduler is periodic. For a given set of rates the schedule repeats itself with a period of $\mathrm{O}(N)$ link time slots. Initially, there is an $\mathrm{O}(N^{2.5})$ complexity associated with the schedule construction. Once the schedule is generated, rate quantization makes it possible to make incremental schedule updates with $\mathrm{O}(N)$ complexity per update.

The rest of the paper is organized as follows. In Section II we give the problem model, definitions and some fundamental properties of reservation based scheduling. In Section III, we study *rate quantization*: what it is and how it is implemented. In Section IV, we present the impacts of rate quantization on rate reservation based scheduling algorithms for single crossbar switches. First, we give a worst case delay analysis, next we describe a simple algorithm by which incremental scheduling updates can be made with rate quantization. Finally, we give conclusions and possible extensions in Section V.

II. Problem Model

At a very high level, a simple version of the problem we focus on can be stated as follows. Suppose each input-output (I-O) pair (i, j) of a VOQ crossbar switch asks for a rate, R_{ij}, to be sustained over a duration of T time slots. Also, suppose that at the end of that duration, each I-O pair either asks for a new rate or chooses to keep the same rate to be sustained for another T time units. We study how these guarantees can be provided simultaneously over all I-O pairs and investigate different tradeoffs involved in such a dynamic system. Next we present the mathematical model. The model we present will be for a switch of size $N \times N$ but the generalization to an asymmetric ($N \times M, N \neq M$) switch is straightforward.

A. Definitions and Assumptions

We define a contract between an input-output (I-O) pair as a number of cells and an associated duration (in number of link time slots) in which these cells are to be transferred from the queues at the corresponding pair. Thus, a rate, R_{ij}, can be associated with the input-output pair (i, j) as the ratio of the number of cells to be transferred between the pair to the lifetime of the contract. The only constraint we have on the duration, T_{ij}, of a contract is that it must be an integer multiple of some *frame size*, T, which is a function of speedup that will be specified later in Section IV. We have no other restriction on durations and the number of cells to be transferred; and hence, R_{ij} takes on any value from the set $[0, 1]$. We assume the presence of an admission controller that makes sure that at any point in time $\sum_i R_{ij} \leqslant 1$ for all j and $\sum_j R_{ij} \leqslant 1$ hold simultaneously for all i. Succinctly, we can represent each R_{ij} as the (i, j) entry of a matrix, R. The two admission control inequalities imply that R is a doubly sub-stochastic matrix.

A switch is responsible for providing the desired number of service opportunities specified in the contract between each I-O pair. Consider an input queued switch (no speedup). If the lifetime of the contract for the I-O pair (i, j) is T_{ij}, then $T_{ij}R_{ij}$ of the T_{ij} configurations that the switch goes through in the next T_{ij} link time slots should connect input i to output j for the terms of the contract to be met. We call a schedule of configurations, a *switch schedule*.

Suppose, at time t for some switch schedule, the switch has gone through configurations such that input i and output j are connected for $D_{ij}(t)$ link time slots. We call the difference, $tR_{ij} - D_{ij}(t)$, the service lag for I-O pair (i, j) at time t.

Note that service lag is directly tied with delay[1]. Indeed, for a cell waiting at the head of a VOQ of an I-O pair,

$$\text{maximum cell delay} = \frac{\text{maximum service lag}}{\text{rate}} \qquad (1)$$

Let us restate the crossbar constraint without broadcast: In a single service time slot, no input can be connected to more than one output, and no output can be connected to more than one input. There corresponds a distinct configuration matrix for every feasible configuration. A configuration matrix has at most one 1 in each row and column. Thus, there corresponds a distinct permutation matrix for every feasible configuration in which no input and no output remains unmatched. Such a pairing of a switch configuration and configuration matrix is illustrated for a 3×3 crossbar in Fig. 2. There is a 1 in every position of the matrix where the crosspoint in the corresponding location of the crossbar is connected.

Next, we give a couple of theorems that provide some fundamental insight on rate reservation based scheduling algorithms.

B. Fundamentals of Rate Reservation Based Scheduling

First, we study contracts with infinite durations, i.e., each I-O pair keeps its rate forever, and present a theorem on rate

[1]There are two different types of delay at the input of a packet switch. The first is due to the randomness in the packet arrival process. This kind of delay is unavoidable and depends on how bursty the arrival processes are. The second is due to the imperfections of the schedulers used. We analyze the latter in this paper.

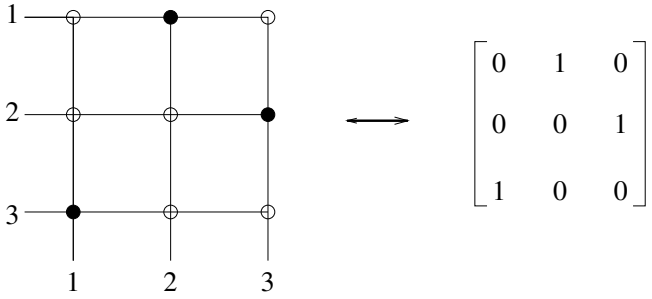

Fig. 2. There is a one to one correspondence between permutation matrices and crossbar configurations.

reservation based scheduling. Next, we focus on Birkhoff's decomposition and switch scheduling for contracts with infinite durations.

1) Long Term Contracts: For a contract with rate matrix, duration matrix pair, (R, T), let us define the corresponding infinite duration contract as the one with the same rate matrix and $T \to \infty$, for all pairs (i, j).

Definition 1: A contract is *supportable* if a schedule of permutation matrices exist that produces a service history, $D(t)$, for which the service lag remains upper bounded for all I-O pairs as $t \to \infty$.

If a schedule exists for which there is a certain point, t, in time such that $tR_{ij} - D_{ij}(t) \leqslant 0$ for all pairs (i, j), then we say the rate matrix R is *perfectly supportable* at time t.
If R is perfectly supportable at time t, then it is also perfectly supportable at times kt, $\forall k \in \mathbb{Z}^+$ since the switch can implement a periodic schedule which repeats itself every t seconds. Therefore perfect supportability implies supportability but the converse is not necessarily true.

Note that, supportability is defined only for infinite duration contracts, and determined by R alone. Given that R is supportable with the schedule $D(t), t > 0$,

$$\lim_{t\to\infty} \frac{D_{ij}(t)}{t} \geqslant R_{ij}$$

for all (i, j). In a crossbar switch, perfect support may not be possible for some supportable traffic. The following theorem gives a necessary and sufficient condition for supportability.

Theorem 1: A rate matrix R is supportable for an input queued switch if and only if it can be upper bounded entry-wise by a convex combination of configuration matrices.

Proof: First, let us prove the only if part. Suppose R is supportable; then there exists a scheduler and some $B < \infty$ such that $tR_{ij} - D_{ij}(t) \leqslant B$ for all pairs (i, j) and for all t. Hence,

$$\lim_{t\to\infty} [tR - D(t)] \leqslant B\vec{e}\vec{e}^T \tag{2}$$

where $\vec{e}$ is the N dimensional vector all of whose entries are 1, and $\vec{e}^T$ is its transpose. Note also that the inequality is entry-wise. Suppose $P(1), \ldots, P(t)$ are the corresponding configuration matrices for the the configurations that the crossbar goes through in $(0, t]$. Then,

$$D(t) = \sum_{l=1}^{t} P(l) \tag{3}$$

Since $D(t)$ is a sum of t configuration matrices and B is not a function of t, (2) can be written as,

$$R \leqslant \lim_{t\to\infty} \frac{1}{t} \sum_{\tau=1}^{t} P(\tau) \tag{4}$$

The limit exists because the set of feasible configuration matrices is finite.

Conversely, if there exists a set of non-negative coefficients, $\{\phi_1, \ldots, \phi_M\}$ for which $\sum \phi_i = 1$ such that

$$R \leqslant \sum_{l=1}^{M} \phi_l P_l \tag{5}$$

then it was shown in [14] that the service lag remains upper bounded at all times using a packetized processor sharing schedule of the configurations with non-zero coefficients in the decomposition. Next we study this closely.

2) Birkhoff's Decomposition: As shown in [14], for every feasible rate request matrix, a doubly stochastic matrix can be found whose entries are at least as large as their counterparts in the rate request matrix (von Neumann). Hence, Theorem 1 implies that for a set of rates to be supported by an input queued switch, there must exist a convex combination of permutation matrices such that each entry of the combination is at least as large as the rate between the corresponding I-O pair. Also from [14], for every doubly stochastic matrix, R, the following decomposition can be made:

$$\begin{aligned} R &= \sum_{i=1}^{K} \phi_i P_i \\ 1 &= \sum \phi_i \end{aligned} \tag{6}$$

where $K \leqslant (N-1)^2 + 1$. In view of (6), the fraction of time that the crossbar has to spend configured to the permutation matrix P_k is equal to ϕ_k. Since only one permutation matrix can be set each link time slot, a schedule of the corresponding configurations must be constructed according to the weights in the decomposition.

This idea was first introduced in [14] where the use of Packetized Generalized Processor Sharing (PGPS[2]) is proposed. Each permutation matrix is treated as a user in PGPS, and the weight of a matrix represents the desired rate of that user. Users are assumed to be backlogged all the time and the finishing times of the next unserved token for each user in the corresponding Generalized Processor Sharing (GPS) system is calculated starting at time 0.

It is illustrated in [19] that, with a fairly large class of schedulers, a maximum service lag of $O(N^2)$ is unavoidable for input queued switches. Namely, there exist some rate matrices for which the service lag can go as high as $O(N^2)$ service time slots for some I-O pairs at different points in time. To our knowledge, no scheduler that overcomes this $O(N^2)$ has been developed so far. Also for many rate matrices, it is not always possible to find certain points in time for which the service lag is small over all I-O pairs simultaneously.

[2]PGPS is introduced in [18]

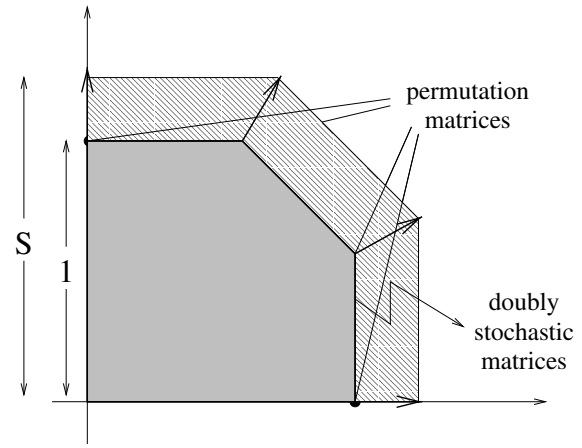

Fig. 3. Extreme points are expanded by a factor of S. The convex combination the expanded extreme points is a superset of the set of doubly stochastic matrices which constitute the set of admissible rates. Note that the set of admissible rates is unchanged.

A service lag of $O(N^2)$ may correspond to an undesirable delay of tens to hundreds of milliseconds for typical commercial packet switches. For instance, for a 256×256 ATM switch with lines of rate 155.5 Mbps, a cell delay of up to ~ 179 msecs can be experienced for some of the I-O pairs using the Birkhoff decomposition. With such delays, service contracts cannot be made for short time periods. For example, for the 256×256 switch, rate contracts need to be held for periods of order seconds for all the service contracts to be met reasonably closely.

Another issue is that, computationally, running Birkhoff's decomposition is fairly complex. Indeed, for an $N \times N$ switch, Birkhoff's decomposition has a computational complexity of $O(N^{4.5})$. Also, it is not possible to run the decomposition and set configurations simultaneously; the first crossbar configuration can be set only after the decomposition algorithm terminates. If the input traffic changes frequently, or contracts have short durations, PGPS with a plain Birkhoff decomposition approach may be infeasible. For example, this approach is ruled out for current multimedia applications whose traffic dynamics vary in time scales from microseconds to a few milliseconds.

In the next section, we will show how to use a small speedup both to cut the service lag to $\frac{N}{S-1}$ and to make the algorithms simpler.

III. Rate Quantization

In the preceding analyses, we assumed no speedup, i.e., only one cell can be transferred between an I-O pair per link time slot. In this section, we loosen this constraint and allow $S > 1$ cells to be transferred per link time slot. Thus, a service time slot is S^{-1} times a link time slot. The factor, S, represents the resource speedup[3]. Since more than one cell can be forwarded to the same output port per link time slot, the switches are combined input and output queued.

Since the crossbar can set up to S configurations per link time slot, we are no longer limited to the convex combination of configuration matrices. In fact, the region of the set of rates over which the switch can transfer cells from input side to the output side is no longer bounded by the convex hull of permutation matrices, but a positive linear combination. A supportable rate, R, with speedup S has the following form:

$$\begin{aligned} R &\leqslant \sum \phi_i P_i \\ S &= \sum \phi_i \end{aligned} \tag{7}$$

Note, however, that the region of admissible rates is still bounded by the convex hull of permutation matrices since the capacities of the input and output links are unchanged (1 cell per link time slot). The support set is, thus, a superset of the set of admissible rates (i.e., doubly stochastic matrices) as illustrated in Fig. 3. Therefore, we may transfer cells through the crossbar at a higher rate than they actually arrive. In this section we show how to divide this extra rate over the I-O pairs of a switch.

If we distribute the extra resource uniformly over each I-O pair, the worst case service lag and the worst case delay decreases proportional to S^{-1}. Hence, the service lag would still be $O(N^2)$, even though an improvement is observed for all the I-O pairs. But, the worst case service lag is not uniform over all the I-O pairs, and therefore, we do not want the improvement to be uniform. Thus, we should not assign the extra resource uniformly.

The following fact, presented in [20] and [14], gives us insight into a good way of distributing the extra resource. If there exists an integer f such that the matrix fR contains all integer-valued elements, then Birkhoff's decomposition terminates in f steps with permutation matrices (not necessarily distinct), all of which have the same coefficient, f^{-1}. This is illustrated for $f = 2$ in the following example:

$$\begin{bmatrix} 1/2 & 0 & 1/2 \\ 0 & 1/2 & 1/2 \\ 1/2 & 1/2 & 0 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} + \frac{1}{2}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

We will discuss how this affects the performance later. Before we proceed with the main theorem of this paper, we present an example which illustrates rate quantization on a single multiplexer (or an $N \times 1$ switch).

Example 1: Suppose a link sends 1 cell per second starting at time 0. The link is shared by 3 users with rates given by $R_1 = 0.53, R_2 = 0.17, R_3 = 0.3$ respectively. Let $D_i(t)$ be the (integer) number of cells served for user i by time t. We have

$$\begin{aligned} \sum_{i=1}^{3} D_i(t) &= \lfloor t \rfloor \\ &= \sum_{i=1}^{3} R_i \lfloor t \rfloor \end{aligned}$$

Since $D_i(t)$ is integer and $R_i\lfloor t \rfloor$ is non-integer for at least one i for each $t < 100$, the above equation shows that for each $t < 100$, $D_i(t) < R_i t$ for at least one i. Thus the service lag $R_i t - D_i(t)$ is positive for all $t < 100$ for some i.

Now suppose the link capacity is increased to 11 cells per 10 seconds. If the users are given 6, 2, and 3 service opportunities respectively in the first 10 seconds, then $D_i(10) \geqslant 10R_i$ for

[3]We assume that S can also be a non-integer number.

each i and the service lag is 0 for each user at each multiple of 10 seconds.

In the first scenario, if the users made a contract for some period of time less than 100 seconds, the link would not be able to meet the terms of at least one contract. With rate expansion, this period is cut down to 10 seconds. This enables the users to update the terms (e.g., the rate) of their contracts more frequently, and thus a larger set of sources (e.g., bursty sources whose statistics vary in shorter time scales) can be accommodated.

Next, we show how to divide some given extra capacity (speedup) among the I-O pairs to achieve a similar improvement in the case of an $N \times N$ crossbar switch.

Theorem 2: Let R be an $N \times N$ doubly stochastic matrix and s be a rational number which can be written as $\frac{1}{f}$ where f is an integer. There exists a (doubly super-stochastic) matrix, $Q = R' + U$, where R' is a doubly stochastic matrix with all the entries integer multiples of s, $U_{ij} = s$ and $Q_{ij} \geqslant R_{ij},\ \forall 1 \leqslant i, j \leqslant N$. Thus, all rows and columns of Q sum up to $S = 1 + sN$.

This theorem can be proved as follows[4]. Let us set $Q_{ij} = s\lceil R_{ij}/s \rceil$ if $R_{ij} > 0$ and $Q_{ij} = s$ if $R_{ij} = 0$.

It is easy to see that entries of Q are all integer multiples of s. Moreover, $R_{ij} \leqslant Q_{ij} \leqslant R_{ij} + s$ and $Q_{ij} \geqslant s$ for all (i, j). As such, the row sums and the column sums of Q are bounded above by $1 + sN$. Now we can use the von Neumann algorithm (see [23]) to convert it into a matrix with identical row sums and column sums, completing the proof.

In fact, the proof of this theorem not only shows that the desired matrix Q exists for any given R, but also gives us a way to construct that matrix. However, the von Neumann algorithm is inherently unfair in dividing the extra service among I-O pairs that share the same input or the same output. Indeed, it may be the case that one I-O pair gets all the extra sN service opportunities per unit time while the other I-O pairs that share the same input or the same output end up with no extra service at all. This may lead to a waste of resources and thus a faster queue build-up.

Next, we introduce an algorithm to construct matrix R', (and thus the matrix Q) for a given R in such a way that certain fairness criteria are met. Before we give the algorithm, we give an example which illustrates the theorem, and at the same time provides intuition about an algorithm. Consider the following 3×3 doubly stochastic matrix.

[4]We would like to thank the editors of INFOCOM 2004 for this version of the proof.

Example 2:

$$R = \begin{bmatrix} 0.48 & 0.35 & 0.17 \\ 0.29 & 0.49 & 0.22 \\ 0.23 & 0.16 & 0.61 \end{bmatrix}$$

$$\overset{(s=0.1)}{\leqslant} \underbrace{\begin{bmatrix} 0.5 & 0.4 & 0.2 \\ 0.3 & 0.5 & 0.3 \\ 0.3 & 0.2 & 0.7 \end{bmatrix}}_{\text{rounded up}} \begin{matrix} \rightarrow & 1.1 \\ \rightarrow & 1.1 \\ \rightarrow & 1.2 \end{matrix} \quad (8)$$

$$= \underbrace{\begin{bmatrix} 0.5 & 0.4 & 0.1 \\ 0.2 & 0.5 & 0.3 \\ 0.3 & 0.1 & 0.6 \end{bmatrix}}_{\text{doubly stochastic}} + \begin{bmatrix} 0 & 0 & 0.1 \\ 0.1 & 0 & 0 \\ 0 & 0.1 & 0.1 \end{bmatrix} \quad (9)$$

Our algorithm generates R' in two steps; in the first step, a matrix $\tilde{R}$ whose entries are integer multiples of s is constructed and in the second step $\tilde{R}$ is modified to get R'.

In the first step (8), every entry of the original matrix is increased by some non-zero amount, so that they all become integer multiples of s. Matrix $\tilde{R}$ is not necessarily some multiple of a doubly stochastic matrix.

In the second step R' is constructed. Matrix R' is a modification of $\tilde{R}$, where sufficiently many entries are reduced by s to make R' doubly stochastic. The challenging part of the algorithm is choosing which entries to reduce. To illustrate that this indeed is not a straightforward task, consider the above example and suppose we construct R' from $\tilde{R}$ starting with the first entry of the first row. Proceed with that row going through all the columns from left to right, reducing each entry by s if the sum of the entries of that column is greater than 1, until the first row sum becomes 1. Once the first row entries sum to 1, proceed with the second row and repeat the process. After completing the second row, we end up with the following matrix, whose third row is yet to be processed:

$$\begin{bmatrix} 0.4 & 0.4 & 0.2 \\ 0.3 & 0.4 & 0.3 \\ 0.3 & 0.2 & 0.7 \end{bmatrix} \begin{matrix} \rightarrow & 1 \\ \rightarrow & 1 \\ \rightarrow & 1.2 \end{matrix}$$
$$\begin{matrix} \downarrow & \downarrow & \downarrow \\ 1 & 1 & 1.2 \end{matrix}$$

As we proceed with the third row, the only entry that can be reduced is the final one, 0.7, since all the other column sums are already 1. However, it has to be reduced by 0.2 for the resulting matrix to be doubly stochastic. If we do so, we end up with $Q_{33} = R'_{33} + 0.1 = 0.6 < R_{33}$.

Hence, we cannot choose the entries to be processed arbitrarily, and must be more careful in constructing Q since each entry of $\tilde{R}$ can be reduced once.

Algorithm:

We first give the algorithm formally, and then a detailed explanation of each step follows.

Initial Values: Let k_m and k'_n be such that, $1 + sk_m$ and $1 + sk'_n$ are the mth row and nth column sum respectively, as illustrated in Fig. 4. Let $i = \arg\max_{1 \leqslant l \leqslant N} k_l$ and $R' = \tilde{R}$ Repeat (1)-(2) until $k_i = 0$ for all $i \leqslant N$.

1) Set $E = \{1, \ldots, N\}$. Repeat (a)-(b) until $k_i = 0$.
   a) $j = \arg\max_{j \in E} k'_j$
   b) $R'_{ij} = \tilde{R}_{ij} - s,\ k_i \rightarrow k_i - 1,\ k'_j \rightarrow k'_j - 1,\ E \rightarrow E - \{j\}$.

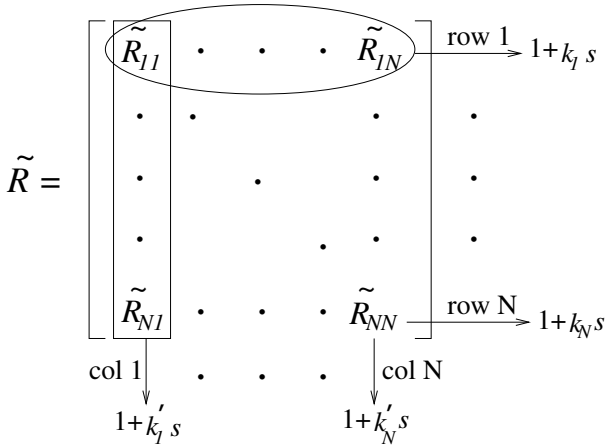

Fig. 4. The $N \times N$ matrix $\tilde{R}$ is illustrated. Each row i and and column j sum to $1 + k_i s$ and $1 + k'_j s$ respectively where k_i and k'_j are non-negative integers.

2) $i = \arg\max_{1 \leqslant l \leqslant N} k_l$

Setup: Given any s, there exists a $\sigma_{ij},\ 0 < \sigma_{ij} \leqslant s$ such that $R_{ij} + \sigma_{ij}$ is an integer multiple of s for all $1 \leqslant i, j \leqslant N$. Let σ be the matrix whose (i, j) entry is σ_{ij}. Define $\tilde{R} = R + \sigma$. All rows and columns of $\tilde{R}$ sum to integer multiples of s. By definition, 1 is also an integer multiple of s, and thus, as illustrated in Fig. 4, we can represent the sum of the entries of the ith row and the jth columns $1+k_i s$ and $1+k'_j s$ respectively where k_i and k'_j are positive integers.

In the iterative step, the algorithm scans $\tilde{R}$ row by row, starting with the row with maximum row sum $k_{\max}$, and determines whether the entry will remain unchanged or reduced by s before it is copied as the corresponding entry of the output matrix, R'. Each row is scanned starting from the entry with the largest column sum and continuing with entries of decreasing column sums. If both k_i and k'_j are positive for the current (i, j), that entry is reduced by s and otherwise it is copied directly as the corresponding entry of R'.

The described algorithm reduces the elements of each row of $\tilde{R}$ in the order of decreasing row sums. The proof of correctness for the algorithm can be found in Section .One might also randomize the procedure and work on a row randomly picked at every iteration. This modified algorithm and the proof of correctness for the modified algorithm can be found in [19].

Let us define the extra service for an I-O pair as the difference between the service provided and the service desired. When the algorithm terminates, the extra service awarded to any given pair of I-O pairs is within $2s$ of each other which is much smaller compared to sN, which was the case with the von Neumann algorithm as given in the proof of the theorem. This has some important implications on the performance of the switch in the steady operation regime which will be studied in Section IV-C.

Furthermore, suppose Q' is any matrix which has entries integer multiples of s and all rows and all columns sum to $1 + sN$. Then, any row or any column of the matrix $Q - \tilde{R}$ is majorized by the corresponding row or the column of the matrix, $Q' - \tilde{R}$. That is, after the rounding up stage, there is no "fairer[5]" way of constructing matrix Q than by our algorithm. This is a direct consequence of Kemperman's theorem ([24]) which can also be found in the appendix.

Given that s is the reciprocal of an integer, each row and column of $\tilde{R}$ sums to no more than $1 + s(N - 1)$ (1 is an integer multiple of s and each entry is increased by no more than s.) unless all the entries of a row are already integer multiples of s. Hence, matrix $\tilde{R} - R'$ has rows and columns each of which is composed of no more than $(N - 1)$ s's, and all 0's otherwise. Using von Neumann's algorithm this matrix can be processed to have identical row and column sums of $1 + s(N - 1)$. We can slightly modify the main theorem by replacing the constant matrix, U with this modified matrix so that all rows and columns of the resultant matrix, Q sum up to $S = 1 + s(N - 1)$ instead of $1 + sN$. However, we will use the constant matrix, $U_{ij} = s$ in constructing Q in what follows since it does not require extra processing to generate and the extra speedup necessary associated with using U is insignificant.

IV. Impacts of Rate Quantization

In this section, we show how to build a switch with a given speedup S using rate quantization. There are two regimes of operation.

Initially, we assume that R is doubly stochastic. The rate quantization algorithm is applied on this matrix and a schedule of configurations is constructed. But, as will be shown, this construction will run on line as the switch configurations are set. We call this regime the *schedule construction regime*.

Once the schedule is generated, the switch enters into the *steady operation regime*. In this regime, no schedule change is made unless a change in the quantized matrix occurs. We show that incremental schedule changes can be made with a fairly simple algorithm. In what follows, we describe both of these regimes.

A. Schedule Construction and Performance with Rate Quantization

Suppose, we initially have a doubly stochastic rate matrix, R. Given the parameter s, the rate quantization algorithm generates a doubly super-stochastic matrix Q such that $Q_{ij} \geqslant R_{ij}\ \forall i, j$. Thus, supporting the matrix Q is sufficient for supporting R. We showed in the last section that Q can be written as the sum of a doubly stochastic matrix, R', with entries that are integer multiples of s, and a constant matrix, U. If we define E to be the $N \times N$ constant matrix with all 1's, then, $U = sE$. Note that E can be written as a sum of N permutation matrices (e.g., the identity matrix and $N - 1$ cyclic shifts of the identity matrix). As a result, Q can be decomposed as follows.

$$Q = \sum_{i=1}^{\frac{1}{s}+N} sP_i \tag{10}$$

[5]Majorization is being used as a tool for measuring and comparing income distributions. Let $\vec{A}$ be the vector of incomes of the population of a country. Then, $\vec{A}$ is said to be fairer than all $\vec{A}'$ such that $\vec{A}' \succ \vec{A}$

where it may be the case that $P_l = P_m$ for some $l \neq m$. Note that:

1) The coefficients in (10) sum to $1 + sN$. Since each permutation matrix corresponds to a crossbar configuration, the crossbar must serve $1 + sN$ permutations per link time slot, i.e., it must transfer $1 + sN$ cells per link time slot[6]. Hence, a speedup of $S = 1 + sN$ is necessary and sufficient to support Q.
2) The coefficients in (10) are all identical. Thus, an equal amount of time must be spent on each configuration, and a scheduler can construct a periodic schedule that repeats itself every $\frac{1}{s} + N$ configurations. We call one period of configurations a *frame*.

We can take advantage of the simple structure of (10) when designing our scheduler. We will consider an *arbitrary order scheduler* (AOS) which serves the $\frac{1}{s} + N$ configurations in an arbitrary order. Let them be served in the order they are generated by the decomposition algorithm so that a configuration can be set up once constructed, rather than waiting for the entire decomposition to be complete.

Next, we derive an upper bound for the worst case service lag with an AOS scheduler. It was shown in [19] that this service lag is tight with a large set of schedulers[7].

It is clear that, due to the periodic nature of the schedule, the number of service opportunities provided to an I-O pair is at least as much as that desired by that user at times $t = k/s,\ k \in \mathbb{Z}^+$. Thus, Q (and R) is perfectly supportable at time $\frac{N}{S-1}$ by the crossbar switch with speedup S. Hence, a cell cannot get delayed at the head of its VOQ more than one frame time, i.e.,

$$\begin{aligned} \text{delay} &\leqslant \frac{1}{s} - 1 \\ &< \frac{N}{S-1} \end{aligned} \tag{11}$$

link time slots. Next, we analyze what happens in between these *perfect service* points and derive an upper bound on the service lag for the AOS. Recall that we defined $R_{ij}t - D_{ij}(t)$ as the service lag, where R_{ij} is the desired rate and $D_{ij}(t)$ is the number of service opportunities provided for the I-O pair (i, j) by time t.

Claim 1: Let $L(S)$ be the maximum service lag over all I-O pairs and all $t > 0$. The following holds for $L(S)$ with rate quantization along with a speedup of $S = 1 + sN$ and an AOS:

$$L(S) = \begin{cases} \frac{1}{4} N \frac{S}{S-1} & ,\ 1 < S \leqslant 2 \\ \frac{N}{S} & ,\ S > 2 \end{cases} \tag{12}$$

Proof: Because of the periodic nature of the services, it suffices to look at the first frame, $\left[0, \frac{1}{s}\right]$. Suppose an I-O pair, (i, j) with a quantized rate, Q_{ij}, is awarded all of its Q_{ij}/s service opportunities toward the end of each frame. In this scenario, I-O pair (i, j) gets no service in the first

[6]Note that $1+sN$ does not need to be an integer, it is just a factor by which the switch operates faster than line rates. For instance, if $1 + sN = 1.5$, then three cells every two link time slots can be transferred from the input to the output of the switch.

[7]Here we consider the set of schedulers that construct a schedule using the coefficients of the decomposition only. Namely, the schedule does not rely on rates of individual I-O pairs.

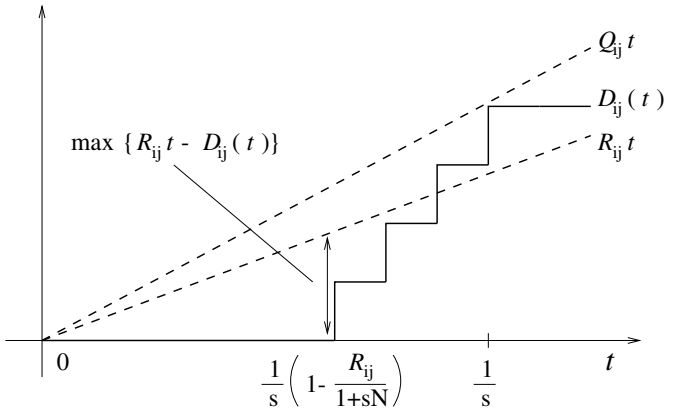

Fig. 5. The service curve for the I-O pair (i, j) which gets its service opportunities at the end of each frame.

$(1 - Q_{ij})\frac{1}{s} + N$ service time slots and is given service in all of the final $Q_{ij}\frac{1}{s}$ service time slots of each frame. This happens with AOS if the final $Q_{ij}\frac{1}{s}$ permutation matrices have $P_{\cdot,ij} = 1$ and all the others have a 0 in this location. It is clear that this leads to the worst possible service lag for this I-O pair. The desired and provided service curves are illustrated in Fig. 5 for this scenario. User (i, j) can be delayed by no more than

$$\begin{aligned} \left[(1 - Q_{ij})\frac{1}{s} + N\right] \frac{\frac{1}{s}}{\frac{1}{s} + N} &= \frac{1}{s}\left(1 - \frac{Q_{ij}}{1 + sN}\right) \\ &< \frac{1}{s}\left(1 - \frac{R_{ij}}{1 + sN}\right) \end{aligned}$$

link time slots, since $Q_{ij} > R_{ij}$. Thus,

$$R_{ij}t - D_{ij}(t) \leqslant \frac{R_{ij}}{s}\left(1 - \frac{R_{ij}}{1 + sN}\right) \tag{13}$$

Hence, the bound on the service lag varies with the desired rate. If we solve the constrained optimization problem to maximize the upper bound over all possible rates:

$$\max_{R_{ij} \in [0,1]} \frac{1}{s}\left(1 - \frac{R_{ij}}{1 + sN}\right) R_{ij}$$

we get,

$$R^*_{ij} = \min\left\{1, \frac{S}{2}\right\}$$

and when we substitute this into our objective function, we get the upper bound, $L(S)$, of (12).

Note that the bound derived for the service lag is tight for a PGPS scheduler as well as an AOS. Consider some input output pair, (i, j). Suppose, after rate quantization, $Q_{ij} \approx R_{ij}$ It is possible for all the matrices with $P_{\cdot,ij} = 1$ to be distinct and to get scheduled at the end of a frame of a PGPS schedule, and thus, the upper bound for the service lag is tight.

The maximum service lag is plotted in Fig. 6 as a function of the speedup. All the curves illustrate the bounds normalized with respect to the switch size, N. The ordinate should be multiplied by N for the actual service bound.

Note that speedup improves the provided service a great deal in the sense that the provided service follows the rate request much more closely when compared to the no speedup

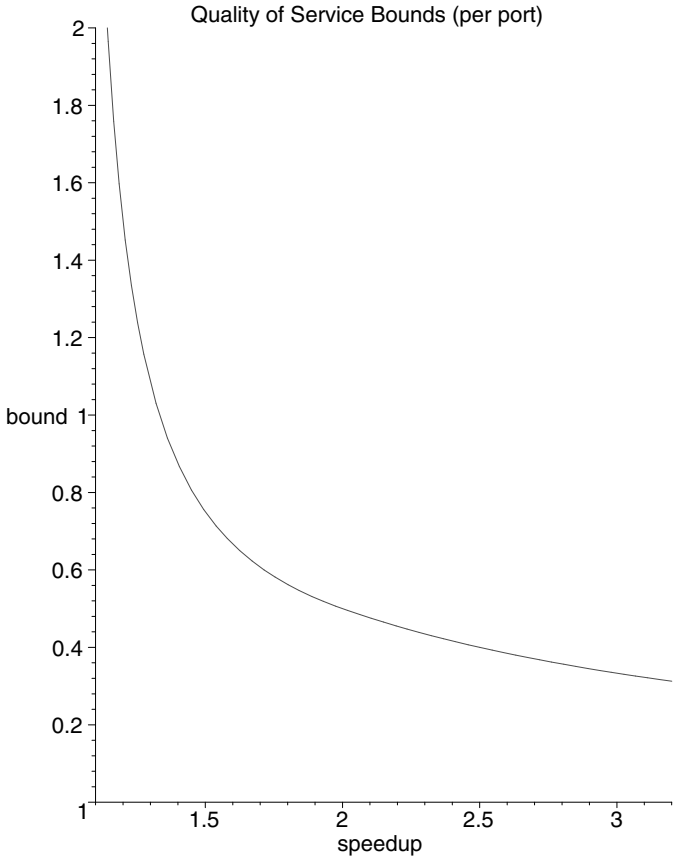

Fig. 6. The worst case normalized service lag as a function of the speedup for large N. Note that the origin corresponds to $S = 1$, hence no speedup. The ordinate should be multiplied by N for the actual service bound.

case. Indeed, for a speedup close to 2, the service lag does not exceed approximately $N/$speedup whereas it can go as high as $\mathrm{O}(N^2)$ without speedup. For example with a speedup of 2, the deviation is no more than $\frac{N}{2}$ link time slots. For a switch of size 256×256, this bound is 128 link time slots; without speedup it can be as high as 65,000 link time slots! The following example illustrates how much service guarantees may be improved even with reasonably small speedups.

Example 3: Suppose we have a 128×128 ATM switch whose links have a maximum capacity of 622 Mbps. The switch supports delay sensitive traffic with a maximum allowable delay of 50 μsec. Then, with rate quantization and AOS, the crossbar fabric should run with a speedup of,

$$\begin{aligned} \text{speedup} &\approx \frac{128 \times 53 \text{ bytes/packet} \times 8 \text{ bits/byte}}{622 \text{ Mbps} \times 50 \ \mu\text{sec}} \\ &= 1.745 \end{aligned}$$

B. Complexity of Schedule Construction

The complexity of Birkhoff's decomposition without quantization is $\mathrm{O}(N^{4.5})$ since it takes $\mathrm{O}(N^2)$ iterations of maximum matching each of complexity $\mathrm{O}(N^{2.5})$ are required (see [14]). Without rate quantization, the decomposition needs to be complete before the first crossbar configuration can be set up. With quantization, after generating the first permutation matrix of the decomposition, it can be scheduled at once, and as the corresponding cells are being transferred, the decomposition can continue.

The rate quantization algorithm has a computational complexity of $\mathrm{O}(N^2)$ and the complexity of schedule construction with quantization is $\mathrm{O}(N^{2.5})$ per service time slot, which is identical to the complexity associated with finding the maximum match.

Comparison of the initial schedule construction with plain Birkhoff approach and rate quantization with AOS is not straightforward since the nature of complexity for the two are quite different. In the Birkhoff algorithm, a huge one time cost is paid for schedule construction and also every time contracts are renewed. In the rate quantization algorithm, a smaller cost is paid for initial schedule construction. Moreover, it is not a one time cost but spread in time. In the following section, we present an $\mathrm{O}(N)$ algorithm using which the rate updates can be made without a need to construct a new schedule.

C. Steady Operation and Rate Updates

In the previous section we showed that rate quantization enables perfect supportability. Namely, there exist certain points in time where the service lag is 0 for all I-O pairs simultaneously. Hence, if the rates are kept unchanged at the end of a frame, the same schedule can be kept afterward.

Suppose, at the end of one such frame only one entry of the quantized matrix is changed. Then, a schedule update is necessary. A brute force approach is to construct a new schedule from scratch with the new quantized matrix. In this section, we illustrate a much simpler method for schedule updates.

This method is based on the Slepian-Duguid algorithm (originally developed for Clos networks; see [21] for an in depth treatment) with which rate updates can be made with minimal modification to the existing schedule in a simple and efficient way. We show that rate quantization is necessary for this approach to be successfully implemented, and discuss certain trade-offs. Our purpose is to accommodate rate updates of an I-O pair without changing the existing schedule of configurations significantly.

We assume that the rate matrix, R, is doubly stochastic at the beginning of the steady operation regime. In this case, an increase in rate of an I-O pair cannot be accommodated before some other I-O pairs (that share either the same input or the same output link as our I-O pair) reduce their rates. If an I-O pair reduces its rate so that its quantized rate is reduced by at least s, we can simply vacate that pair from at least one configuration matrix.

Now suppose the rate R_{ij} desired between the I-O pair (i, j) increased by some value such that the corresponding entry in the quantized matrix is increased by s. Further assume that the admission control inequalities still hold. Thus, before the increase, matrix R was doubly substochastic; moreover, the ith row and the jth column sum of the quantized matrix are both less than $1 + sN$. Hence, there must exist at least one configuration matrix, P_R, with all zeros in the ith row and a configuration matrix, P_C with all zeros in the jth column. If $P_R = P_C$, then the rate update can be made without a

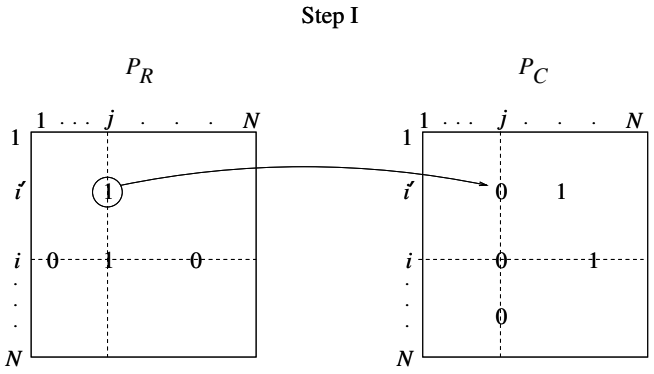

Fig. 7. Matrix P_R has two ones in column j, one in location (i, j) and the other in location (i', j). The latter is removed from P_R and inserted into P_C.

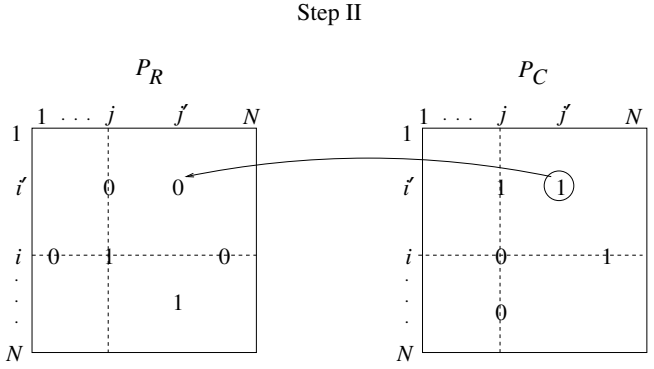

Fig. 8. If a one is found in row i' of matrix P_C, then it is removed from P_C and inserted into P_R.

need for any rearrangement of the existing connections by simply inserting a one in location (i, j) of the configuration matrix. Otherwise, i.e., if $P_R \neq P_C$, a rearrangement is necessary. In fact, the rearrangement process involves only the two configuration matrices P_R and P_C.

First, we insert a one in location (i, j) of P_R. Now, P_R has two ones in column j. One of them is in location (i, j) and let the other one be in location (i', j). The algorithm, then removes the latter from P_R and inserts it into P_C as the (i', j) entry, as illustrated in Fig.7. Next, the algorithm searches for another one in row i' of matrix P_C. If the search is not successful, then the rearrangement algorithm terminates. If a one is found, say in location (i', j'), then it is removed from P_C and inserted into P_R as illustrated in Fig. 8. Similarly, the algorithm searches for another one in the j'th column of P_R to be moved to the corresponding row in P_C. The algorithm terminates if it cannot be found. The algorithm continues in this manner until both matrices have at most one in each row and in each column.

The rearrangement algorithm takes at most $2N - 2$ steps since there are a total of $2N - 2$ rows and columns combined, other than row i and column j. We started the search process by placing a one in location (i, j) of matrix P_R. We could as well start it by placing a one as (i, j) entry of P_C and the procedure would still work. Indeed, Paull ([22]) proposed a slight modification to the Slepian Duguid algorithm. Using that approach we can modify our algorithm as follows: Instead of starting the algorithm with P_R or P_C alone, run two rearrangement algorithms in parallel. Since the sum of the number of rearrangements of the two algorithms can at most be $2N$ (since there are that many entries that contains a one), one of them lasts for no longer than N rearrangements. Using the configuration matrices produced by the shorter set of rearrangements reduces the maximum number of rearrangements down to a half ($N - 1$ rearrangements) of that of the original procedure.

Note that, even though up to $N - 1$ rearrangements are necessary to fulfill a request for an increase of s in some rate, these rearrangements involve only two configuration matrices. Hence, after the rearrangement process $\frac{1}{s} + N - 2$ configuration matrices will remain unchanged. The complexity of the rearrangement procedure is $O(N)$, which is an $O(N^{1.5})$ improvement over rerunning the decomposition each time an entry is changed. Only two configuration matrices are modified and the rearrangement process implies a schedule change for no more than $N - 1$ I-O pairs other than the one which asks for the rate increase. Similarly, this is a factor $O(N)$ improvement since, if we ran the entire decomposition after each rate change, the entire schedule might be changed.

It can be easily observed that rate quantization is necessary for the described rearrangement procedure to work. There are some trade-offs we need to take into consideration to choose the quantization parameter, s. The necessary speedup for quantization is $1 + sN$, which increases as s increases. On the other hand, in a system where desired rate changes are small[8], the frequency of rate updates will increase as s decreases. Also, if the change in the desired rate is high compared to s, the number of users that may be affected by the update will increase. Indeed, with each s increase in the quantized rate, the number of users that possibly need a schedule update increases by N.

Even though the steady operation regime is described for the case where all the rate requirements are known, it can be modified to be suitable for the scenario where the rates are not known apriori. For instance, the system may use the states of the VOQs to estimate the changes in rate requirements and make rate updates accordingly.

Finally, we note that in [19] show that a speedup of 2 (over the speedup necessary for rate quantization) is sufficient for strictly non-blocking switch schedules for unicast traffic over single crossbar switches. That is, schedule updates of different I-O pairs can be completely decoupled, i.e., any change in the rate of an I-O pair can be accommodated, without any need for rearrangement of the existing schedule of other I-O pairs. We proved this result by illustrating an isomorphism between single crossbar switch schedules and three stage Clos networks.

V. Conclusions

We presented a rate quantization algorithm which, along with some speedup, significantly improves the performance

[8]For instance, suppose the rates are updated according to the state of the VOQs. As the number of cells start to increase, the rates are increased accordingly, and vice versa. In such a system, the rate updates may be desired quite frequently and consequently, the amount of change may be small.

and practicality of reservation based scheduling algorithms. In many cases, rate quantization along with a small speedup cuts the worst case delay by a factor of $O(N)$. It simplifies switch scheduling a great deal. We also presented a Slepian-Duguid like algorithm of complexity $O(N)$ per rate update.

In some cases, speedup may be undesirable. If there are not long periods of time where links are fully utilized, it can be easily shown that rate quantization performs well without speedup. Indeed, if each link is utilized to no more than S^{-1} of its capacity, then the rate quantization algorithm can be used to achieve the same improvement without speedup as it gives with fully utilized links and a speedup of S. Also note that, in practice it would probably be more straightforward to simply set up contracts directly with a quantization of s.

We would like to state some possible extensions of this work. A number of results on each item can be found in [19].

- The idea of rate quantization can be extended and it can be used to provide rate guarantees over optical wavelength switches. This could integrate the optical and the electronic layers in an inter-operable and compatible manner. Rate quantization could also reduce the need for optical add drop multiplexers since with the proper choice of the quantization parameter, it can eliminate the need for sub-wavelength processing in an optical switch.
- The crossbar fabric is attractive since it is non-blocking and easy to manufacture. However as the size of a switch gets larger, coordination among all the ports becomes increasingly difficult and thus, many algorithms get very complex as the switch size grows. This compels us to look into distributed architectures with less coordination among different units of the architecture.
- Unlike unicast, the set of multicast rates are not supportable over single crossbar switches without speedup even if no input or output link is oversubscribed. In [19] we show that a speedup of $O(\log N)$ is necessary to support all admissible multicast rates over a single crossbar switch.

REFERENCES

[1] Hluchyi M. Karol M. and Morgan S., "Input Versus Output Queueing on a Space Division Switch," *IEEE Transactions on Communications*, vol. 35, 1987.

[2] Kam A. and Siu K. Y., "Linear Complexity Algorithms for QoS Support in Input-Queued Switches with No Speedup," *IEEE Journal on Selected Areas of Communications*, vol. 17, no. 6, pp. 1040–1056, 1999.

[3] Rohrs C. E. Magill R. and Stevenson R. L., "Revisiting Output Queued Switch Emulation by a Combined Input/Output Queued Switch," in *Allerton Conference*, 2002.

[4] Anantharam V. McKeown N. and Walrand J., "Achieving 100% Throughput in an Input Queued Switch," in *Proceedings of INFOCOM*, 1996.

[5] Anantharam V. McKeown N., Mekittikul A. and Walrand J., "Achieving 100% Throughput in an Input Queued Switch," *IEEE Transactions on Communications*, vol. 47, no. 8, pp. 1260–1267, 1999.

[6] McKeown N., "The iSLIP Scheduling Algorithm for Input Queued Switches," *IEEE/ACM Transactions on Networking*, vol. 7, no. 2, pp. 188–201, 1999.

[7] McKeown N. Chuang S. T., Goel A. and Prabhakar B., "Matching Output Queueing with a Combined Input/Output-Queued Switch," *IEEE Journal on Selected Areas of Communications*, vol. 17, no. 6, pp. 1030–1039, 1999.

[8] Charny A. Krishna P., Patel N. and Simcoe J., "On the Speedup Required for Work-Conserving Crossbar Switches," *IEEE Journal on Selected Areas of Communications*, vol. 17, no. 6, pp. 1057–1066, 1999.

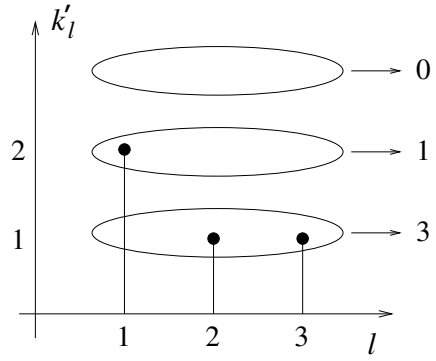

Fig. 9. Vector $\vec{k'} = [2\ 1\ 1]^T$, thus $\vec{n'} = [3\ 1\ 0]^T$.

[9] Prabhakar B. and McKeown N., "On the Speedup Required for Combined Input and Output Switch," in *Proceedings of INFOCOM*, 1999.

[10] Patel N. Charny A., Krishna P. and Simcoe R., "'Algorithms for Providing Bandwidth and Delay Guarantees in Input Buffered Crossbars with Speedup," in *Proceedings of INFOCOM*, 1998.

[11] Charny A., "Providing QoS Guarantees in Input Buffered Crossbar Switches with Speedup," 1998, PhD Dissertation.

[12] Weller T. and Hajek B., "Scheduling Nonuniform Traffic in a Packet Switching System with Small Propagation Delay," *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 813–823, 1997.

[13] McKeown N. Hung A., Kesidis G., "ATM Input Buffered Switches with Guaranteed Rate Property," in *Proceedings of IEEE ISCC*, 1998.

[14] Chen W. J. Chang C. S. and Huang H. Y., "'On Service Guarantees for Input Buffered Crossbar Switches: A Capacity Decomposition Approach by Birkhoff and von Neumann," in *IEEE IWQoS*, 1999.

[15] Chen W. J. Chang C. S. and Huang H. Y., "Birkhoff-von Neumann Input Buffered Crossbar Switches," in *Proceedings of INFOCOM*, 2000.

[16] Rohrs C. E. Magill R. and Stevenson R. L., "Output-queued Switch Emulation by Fabrics with Limited Memory," *To appear, IEEE-JSAC Special Issue on High Performance Optical/Electronic Switches/Routers for High Speed Internet*, 2003.

[17] Rohrs C. E. Magill R. and Stevenson R. L., "Output Queued Switch Emulation by a Buffered Crossbar Fabric," in *Allerton Conference*, 2002.

[18] Parekh A. K. and Gallager R. G., "A Generalized Processor Sharing Approach to Flow Control in Integrated Service Networks: the Single Node Case," *IEEE/ACM Transactions on Networking*, vol. 1, pp. 344–357, 1993.

[19] Koksal C. E., "Providing QoS over High Speed Electronic and Optical Networks," 2002, PhD Dissertation.

[20] Hall M., *Combinatorial Theory*, Blaisdell Publishing Company, 1967.

[21] Hui J. Y., *Switching and Traffic Theory for Integrated Broadband Circuits*, Kluwer Academic Publishers, Boston, MA, 1990.

[22] Paull M. C., "Reswitching of Connection Networks," *Bell Syst. Tech. J.*, vol. 41, pp. 833–855, 1962.

[23] Marshall A. W. and Olkin I., *Inequalities: Theory of Majorization and Its Applications*, Academic Press, New York, NY, 1979.

[24] Kemperman J. H. B., "Moment Problems for Sampling Without Replacement, I, II, III," *Nederl. Akad. Wetensch. Proc. Ser.*, vol. 76, pp. 149–188, 1973.

[25] Day P. W., "Rearrangement Inequalities," *Canad. J. Math.*, vol. 24, pp. 930–943, 1972.

APPENDIX

In this section, we prove a couple of lemmas to show that the rate quantization algorithm we introduced terminates with a matrix of the desired form.

Lemma 1: The algorithm successfully terminates with a matrix R' which is doubly stochastic.

Before we give the proof of the lemma, let us introduce some notation. Let

$$k_i = \frac{1}{s}\left(\sum_{j=1}^{N}(R'_{ij}) - 1\right)$$
$$k'_j = \frac{1}{s}\left(\sum_{i=1}^{N}(R'_{ij}) - 1\right)$$

We can represent k_i and k'_j as an entry of the vectors $\vec{k}$ and $\vec{k'}$ respectively. Let $n'_i,\ i \geqslant 1$ be the number of columns j, for which $k'_j \geqslant i$. For example, if $\vec{k'} = [2\ 1\ 1]^T$, then $\vec{n'} = [3\ 1\ 0]^T$ as illustrated in Fig. 9.

Proof: By induction. We shall first show that initially

$$n'_1 \geqslant k_i \tag{14}$$

for all $i,\ 1 \leqslant i \leqslant N$, Thus, for any $\vec{k}$ and for $i = \arg\max_{1\leqslant l\leqslant N} k_l$ which is the first row to be processed, the algorithm will always be able to find sufficient entries to reduce (by s) to make the row sum equal to 1. We will prove a more general version of (14):

$$\vec{k} \prec \vec{n'} \tag{15}$$

namely, the vector $\vec{k}$ is majorized by the vector $\vec{n'}$. For the definition see Section or [23] for a complete treatment of majorization.

First we prove that (15) holds at the beginning of the algorithm. Recall that $\sigma = \tilde{R} - R$. Hence,

$$\frac{1}{s}\sigma_{ij} \leqslant 1$$

$\forall i, j$. Let the lth column vector of σ be $\vec{v}_l$ and thus $v_{l,j} = \sigma_{jl}$ and $\langle \vec{v}_l, \vec{e}\rangle = k'_l s$, where $\vec{e} = [1 \cdots 1]^T$. From Kemperman's theorem ([24]), $\vec{v}_l$ is majorized by any vector for which k'_l entries are s, and the other $N - k'_l$ entries are 0. Hence,

$$\vec{v}_l \prec [\underbrace{s \cdots s}_{k'_l}\ \underbrace{0 \cdots 0}_{N-k'_l}]^T \equiv \vec{v}_l^{\max} \tag{16}$$

Thus, the vector on the right side of (16) is the *maximal vector* (in the sense of majorization) of the set of vectors whose entries are between 0 and s and $\langle \vec{v}, \vec{e}\rangle = k'_l s$. Let us denote the maximal vector of the lth column vector by $\vec{v}_l^{\max}$.

Now, let us define a new matrix, $\frac{1}{s}\,[\vec{v}_1^{\max} \cdots \vec{v}_N^{\max}]$, where each column is the maximal vector of the corresponding column of $\frac{1}{s}\sigma$. Note that the vector of column sums for this new matrix is $\vec{k'}$, and thus the corresponding distribution will be $\vec{n'}$; however, the row sums are not $\vec{k}$. Let the vector of row sums for our matrix be $\vec{k}_{\text{new}}$. Thus, $k_{\text{new},1}$ is the number of columns with $k'_j \geqslant 1$, i.e., n'_1; $k_{\text{new},2}$ is the number of columns with $k'_j \geqslant 2$, i.e., n'_2, and so on. More precisely, $k_{new,i}$ is the number of columns j, for which $k'_j \geqslant i$. Thus,

$$k_{new,i} = n'_i \tag{17}$$

But the vectors, $\vec{v}_l^{\max},\ l \in \{1, \ldots, N\}$ are order symmetric (see [23] for the definition). Hence we get the desired result

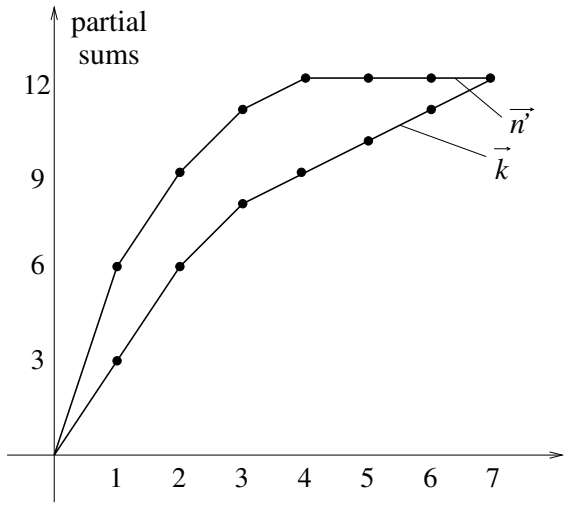

Fig. 10. Sample Lorentz curves for $\vec{n'}$ and $\vec{k}$ are illustrated. Since $\vec{n'} \succ \vec{k}$ initially, the partial sum curve of $\vec{n'}$, is above that of $\vec{k}$.

using Day's theorem ([25]):

$$\vec{k}_{new} = \vec{n}' = \frac{1}{s}\sum_{l=1}^{N}\vec{v}_l^{\max} \tag{18}$$
$$\succ \frac{1}{s}\sum_{l=1}^{N}\vec{v}_l \tag{19}$$
$$= \vec{k} \tag{20}$$

We just showed that at the beginning of the algorithm, $\vec{n'} \succ \vec{k}$, and thus, $n_1 \geqslant k_i$, for all $i \leqslant N$. That is, the first step of the algorithm can be executed successfully to make the first row sum to 1. The partial sums[9] of the two sequences are illustrated in Fig. 10. Such curves are called Lorentz curves and if, for two vectors, $\vec{v}^I \prec \vec{v}^{II}$, then the partial sum curve for $\vec{v}^{II}$ will always be above that of $\vec{v}^I$.

Next, we will prove that a similar majorization relation holds at the beginning of every step of the algorithm. We will use induction as follows. We have shown that $\vec{k} \prec \vec{n}'$ at the beginning of the first step. We now assume that it holds at the beginning of the ith step, $1 \leqslant i \leqslant N-1$ and show that it still holds at the end of the ith step. As a byproduct, we also show that the algorithm can successfully complete each step.

Suppose, the algorithm successfully constructed the first i rows of R'. We will show that (15) still holds at the beginning of the $(i+1)$st step, and the corresponding row of R' can be formed successfully.

First, let us focus on the two vectors, $\vec{n'}$ and $\vec{k}$ at the beginning of step i. At this point, $k_{M(1)}, \ldots, k_{M(i-1)} = 0$ where $M(q)$ is the qth entry in decreasing order from the largest in $\vec{k}$ at the beginning of tha algorithm (before any row is processed). The sum of the entries of the row that is currently being processed is $k_{M(i)}$. By the induction hypothesis, we assume $\vec{k} \prec \vec{n'}$; therefore, there should be as many 0s in vector $\vec{n'}$ as there are in $\vec{k}$ (verified in Section). Since there

[9]The mth partial sum of a vector, $\vec{v}$, is defined to be $\sum_{j=1}^{m} v_j$. Recall that $\vec{v}^I \prec \vec{v}^{II}$ if every partial sum of $\vec{v}^{II}$ is at least as great as that of $\vec{v}^I$

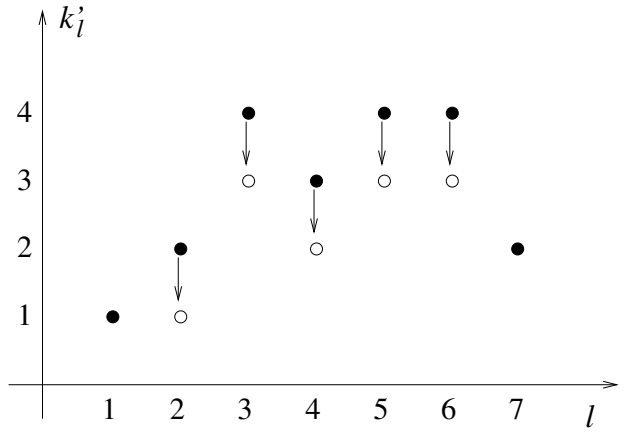

Fig. 11. If $k_i = 5$, the entries of the ith row that are decreased are illustrated above.

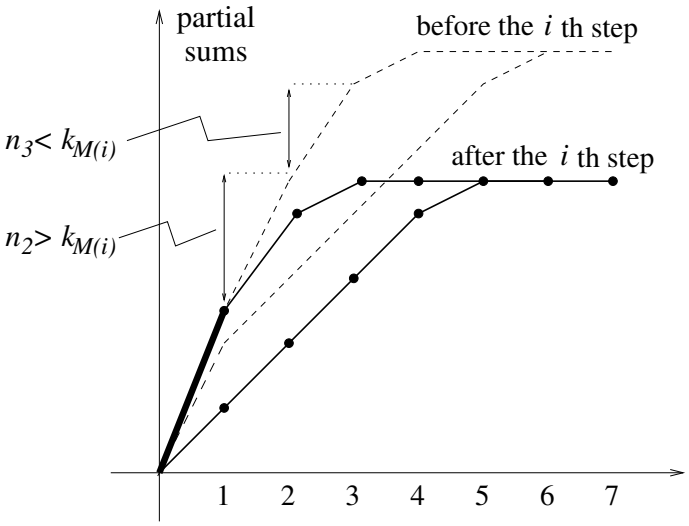

Fig. 12. In the ith step, $k_{M(i)}$ is removed (replaced with a 0) and the smallest entry of $\vec{n}'$ greater than or equal to $k_{M(i)}$ is reduced by $k_{M(i)} - n'_{r+1}$, and the following entry, n'_{r+1} is removed (replaced with a 0). The dashed curve is the initial curve, and the solid one is the one at the end of the ith step. The bold segment is the one which do not change. The distance between the two curves does not decrease at all.

are at least $i-1$ 0s in $\vec{k}$, we have $n'_{N-i+2}, \ldots, n'_N = 0$. At the beginning of the ith step, the entries of $\vec{n'}$ and $\vec{k}_{\downarrow}$ (the decreasing rearrangement of the entries of $\vec{k}$) can be listed as follows:

$$\begin{array}{cccccc} n'_1 & \cdots & n'_{r-1} & n'_r & n'_{r+1} & \cdots \\ k_{M(i)} & \cdots & k_{M(i+r-2)} & k_{M(i+r-1)} & k_{M(i+r)} & \cdots \end{array}$$

$$\begin{array}{ccc} & \overbrace{\quad}^{i-1} & \\ \cdots & 0 \cdots 0 & \\ \cdots & \underbrace{0 \cdots 0}_{i-1} & \end{array}$$

Since $\vec{k} \prec \vec{n'}$, there exists at least one entry in $\vec{n'}$ which is greater than or equal to $k_{M(i)}$. Let the smallest such entry be n'_r.

Lemma 2: At the end of ith step, the only change in $\vec{n'}$ is that the entries n'_r and n'_{r+1} will be replaced with $\left[n'_{r+1} + \left(n'_r - k_{M(i)}\right)\right]$ and a 0.

Proof: These two changes can be explained as follows. The algorithm will look into the current R' for the column with an entry which has not yet been reduced in step i and which has the maximum column sum, and reduce it by s. Suppose this maximum column sum is ms for some $m \in \mathbb{Z}^+$. This operation will reduce the number of columns, j, such that $k'_j = m$ by 1. Thus, the only change in $\vec{n'}$ will be in the smallest non-zero entry, n'_m, which will decrease by 1. If that entry is greater than 1, then there were multiple entries with the maximum column sum. The algorithm continues with these other entries. Hence, if the original value of $k_{M(i)}$ is greater than n'_m, then after processing n'_m entries, n'_m will become 0 and $k_{M(i)} - n'_m$ entries will be left to be decreased at the row currently being processed. The algorithm will go on with the entries that have not been reduced before and with highest possible column sums. At this stage, the new value, $\hat{n}'_m$, of n'_m is 0 and the new value, $\hat{k}_{M(i)}$ of $k_{M(i)}$ is $k_{M(i)} - n'_m$. Note that n'_m potential entries have already been processed, and if $k_{M(i)}$ is greater than the second largest entry, n'_{m-1}, of $\vec{n'}$ then n'_{m-1} will be reduced to $\hat{n}'_{m-1} = n'_m$ but no further beyond that, since n'_m potential entries have already been processed. Similarly, each entry of $\vec{n}'$, which is smaller than $k_{M(i)}$ will be replaced with the next entry in order. Finally, the first entry, n'_r, in $\vec{n'}$ that is greater than $k_{M(i)}$ will be reduced by only $n'_r - k_{M(i)}$. Hence, after the ith row is processed, $\vec{n'}$ will have a 0 replacing n'_r, and a $\left[n'_{r+1} + \left(n'_r - k_{M(i)}\right)\right]$ replacing n'_{r+1}. Note that at the end of the ith step, $\vec{k}_{\downarrow}$ will be the same except $k_{M(i)}$ will be replaced with a 0.

This process is illustrated in Fig. 11 assuming $\vec{k'} = [1\ 2\ 4\ 3\ 4\ 4\ 2]$, i.e., $\vec{n'} = [7\ 6\ 4\ 3\ 0\ 0\ 0]$ at the beginning of step i. If $k_{M(i)} = 5$, then at the end of step i, $\vec{n'} = [7\ 5\ 3\ 0\ 0\ 0\ 0]$. Notice that 6 is the smallest entry in $\vec{n'}$ greater than or equal to $k_{M(i)} = 5$. Hence, 6 and 4 are changed to $6 + (4 - 5) = 5$ and 0 respectively.

Now, we show that, $\vec{k} \prec \vec{n}'$ at the end of the ith step of the algorithm. But before that we present a graphical illustration of what happens in the ith step. The Lorentz curves of $\vec{k}$ and $\vec{n}'$ are illustrated in Figures 12. The entry, $k_{M(i)}$ is removed from $\vec{k}$. The new Lorentz curve for $\vec{k}$ can be sketched from the old one by just removing the first segment segment of the curve and attaching the rest of the curve to the origin as illustrated in the figure. The new Lorentz curve for $\vec{n}$ can similarly be sketched with some modification to the old one. The algorithm will find the segment with the smallest increment greater than $k_{M(i)}$. Then, it will reduce this increment by $k_{M(i)} - n'_r$, remove n'_r, and attach the two separate parts. The two Lorentz curves intersect at 0 and at $\sum_l n'_l = \sum_l k_l$. Initially, these are the only two points they intersect, and the curve for $\vec{n}'$ is always above the curve for $\vec{k}$, otherwise. We need to show that this is the case after the ith step. This can be easily observed from Fig. 12. Since the removed segment in $\vec{k}$ is to the left of the reduced segment of $\vec{n}'$, the distance between the two curves will only increase in between these modified segments, and remain the same outside this region at the end of the ith step. We can prove this statement as follows. There are two

regions we need to consider as shown in the following table:

$$\begin{array}{cccc|} n'_1 & \cdots & n'_r & n'_{r+1} \\ k_{M(i)} & \cdots & k_{M(i+r-1)} & k_{M(i+r)} \end{array}$$

I

$$\begin{array}{ccc} n'_{r+2} & n'_{r+3} & \cdots \\ k_{M(i+r+1)} & k_{M(i+r+2)} & \cdots \end{array} \quad \begin{array}{c} \overbrace{0\cdots 0}^{i-1} \\ \underbrace{0\cdots 0}_{i-1} \end{array}$$

II

At the end of the ith step, the partial sums of the two sequences are as follows. In region I, at the end of the ith step, $k_{M(i)}$ will be replaced with a 0 and it will no longer be in the second region. All the entries of $\vec{n}'$ will be unchanged up to n'_r. Thus, the partial sums will change in favor of $\vec{n'}$ by an extra $k_{M(i)}$ from the beginning all the way down to n'_r. This entry is replaced with $\left[n'_{r+1} + \left(n'_r - k_{M(i)}\right)\right]$, and the next entry, n'_{r+1} will be replaced with a 0 and removed from the second region. The total decrease in the partial sums of $\vec{n'}$ in the first region is $k_{M(i)}$. The extra $k_{M(i)}$ gained in favor of $\vec{n'}$ earlier by the removal of $k_{M(i)}$ from vector $\vec{k}$ is good enough to make up for this loss of $\vec{n}'$. The second region for both $\vec{n'}$ and $\vec{k}$ are expanded similarly, with the addition of a 0. This will not affect the partial sums, and hence the majorization is preserved.

Thus, we proved that at the beginning of each step, (15) holds and $n'_1 \geqslant k_{M(i)}$, for all $i \leqslant N$. Therefore, the algorithm will always be able to find the desired number of entries to reduce, and at the end of the algorithm, $k_i = 0$, for all $i \leqslant N$. But, since

$$0 = \sum_{i=1}^{N} k_i = \sum_{j=1}^{N} k'_j \tag{21}$$

and $k'_j \geqslant 0$, for all $j \leqslant N$, it is also true that $k'_j = 0$, for all $j \leqslant N$ completing the proof.

Lemma 3: Every entry of R' is an integer multiple of s.

Proof: The input matrix, $\tilde{R}$, of the algorithm already has all the entries integer multiples of s. We complete the proof noting that the change in each entry from $\tilde{R}$ ro R' is an integer multiple of s (either reduced by s or left unchanged).

Lemma 4: Every entry of R' is at least as great as its counter part in R decreased by s:

$$R'_{ij} \geqslant R_{ij} - s, \quad 1 \leqslant i, j \leqslant N \tag{22}$$

Proof: Note that

$$\tilde{R}_{ij} \geqslant R_{ij}, \quad 1 \leqslant i, j \leqslant N \tag{23}$$

Since the algorithm reduces every entry by at most s,

$$R'_{ij} \geqslant \tilde{R}_{ij} - s, \quad 1 \leqslant i, j \leqslant N \tag{24}$$

Inequality (22) is immediate by (23) and (24).

Putting all three Lemmas 1, 3 and 4, together, we completed the proof of correctness for the rate quantization algorithm, and thus our main theorem is also proved. In [19] we prove that Lemma 1 holds even if the algorithm processes the rows of matrix $\tilde{R}$ in an arbitrary order, rather than processing the one with the maximum row sum in each step. For any $\vec{x} = (x_1, \ldots, x_N) \in \Re^N$, let

$$x_{[1]} \geqslant \cdots \geqslant x_{[N]}$$

denote the components of $\vec{x}$ in decreasing order, and let

$$\vec{x}_{\downarrow} = \left(x_{[1]}, \ldots, x_{[N]}\right)$$

Definition 2: For $\vec{x}, \vec{y} \in \Re^N$, $\vec{x} \prec \vec{y}$ if the following two conditions hold:

$$\sum_{i=1}^{k} x_{[i]} \leqslant \sum_{i=1}^{k} y_{[i]}, \ k = 1, \ldots, N-1 \tag{25}$$

$$\sum_{i=1}^{N} x_{[i]} = \sum_{i=1}^{N} y_{[i]} \tag{26}$$

When $\vec{x} \prec \vec{y}$, $\vec{x}$ is said to be *majorized by* $\vec{y}$ (or $\vec{x}$ majorizes $\vec{y}$). This terminology was introduced by Hardy, Littlewood and Polya. The following is a trivial example of majorization.

$$\begin{aligned} \left(\frac{1}{N}, \ldots, \frac{1}{N}\right) &\prec \left(\frac{1}{N-1}, \ldots, \frac{1}{N-1}, 0\right) \prec \cdots \\ &\prec \left(\frac{1}{2}, \frac{1}{2}, 0, \ldots, 0\right) \prec (1, 0, \ldots, 0) \end{aligned}$$

Suppose

$$\sum_{i=1}^{N} x_{[i]} = \sum_{i=1}^{N} y_{[i]} = S$$

Subtracting both sides of (25) from S, we get

$$\sum_{i=k+1}^{S} x_{[i]} \geqslant \sum_{i=k+1}^{S} y_{[i]}, \ k = 1, \ldots, N-1 \tag{27}$$

which is equivalent to (25). Hence, if $\vec{x}$ and $\vec{y}$ both have non-negative entries, there are at least as many 0s in $\vec{y}$ as in $\vec{x}$.