Rate Quantization and the Speedup Required to Achieve 100% Throughput for Multicast over Crossbar Switches

C. Emre Koksal, *Member*

***Abstract*— The problem of providing quality of service guarantees for multicast traffic over crossbar switches has received a limited attention, despite the popularity of its counterpart for unicast traffic. Providing a 100% throughput to all admissible multicast traffic has been shown to be very difficult task and it requires a very high speedup in the switching fabric. In this paper, we introduce the concept of rate quantization and use rate quantization to show an analogy between packet scheduling in crossbar switches and circuit switching in three-stage Clos networks. We exploit the analogy to adopt circuit switching algorithms in wide sense and strict sense non-blocking Clos networks in order to construct non-blocking packet schedulers for unicast and multicast traffic. We illustrate a simple multicast non-blocking packet scheduler, for which a speedup of $6 \log n / \log \log n$ is sufficient to support 100% throughput for any admissible multicast traffic in an $n \times n$ crossbar switch. Moreover, we revisit some problems in unicast switch scheduling. We illustrate that the analogy provides useful perspectives and we give a simple proof for a well known result.**

***Index Terms*— Input queued switch, switch scheduling, multicast switch, multicast crossbar, rate quantization, Clos network**

I. Introduction

Applications requiring QoS support for *multicast traffic* remain important in small and large scale networks. The problem of providing quality of service guarantees for multicast traffic over crossbar switches has received a limited attention, despite the popularity of its counterpart for unicast traffic. One of the main reasons for this is the difficulty of the task. Indeed, it was shown in [1], [2] that the problem of "optimal scheduling" of multicast packets over a crossbar switch is NP-hard, as it is dual to the fractional weighted graph coloring problem and in [3] it was proved that the resource speedup necessary to achieve 100% throughput for all admissible multicast traffic grows unbounded with increasing switch size. These results hold even if the crossbar switch is *multicast capable*, i.e., it is capable of connecting an input to multiple outputs or if the crossbar switching fabric contains internal buffers at each crosspoint [4]. Furthermore, it is stated in [2], [3] that the numerical evaluation of the necessary speedup is prohibitive, as it is dual to the membership problem for the stable set polytope of a graph and no scaling law has been given as to how the speedup scales with the switch size. In this paper we present a simple algorithm to provide 100% throughput for all admissible multicast traffic and specify the sufficient speedup as a function of the switch size, to achieve 100% throughput.

In the development, our main tool will be an analogy between middle-stage switch configurations of three-stage Clos networks and schedules for a crossbar switch. A similar analogy was first exploited in [5] for certain set of TDMA schedulers. We construct the analogy for *frame based schedulers* in which the scheduler has the a priori knowledge of the matrix of real arrival rates, that is not necessarily supportable by (periodic) TDMA schedulers. We generalize the analogy to non-periodic switch schedulers, using the notion of rate quantization [6]. Note however that, as stated in [3], the set of all frame based schedulers is equivalent to the set of all *slot-by-slot schedulers* (no a priori information of the arrival rates), provided that slot-by-slot schedulers introduce an additional delay equal to the frame length. We make this statement precise here, also using rate quantization. We prove the main theorem of rate quantization, which was stated in [6] without a proof. Then we generalize rate quantization to multicast rate matrices. Ultimately, the Clos network analogy will be valid not only for frame based schedulers, but for all unicast and multicast schedulers. Indeed we illustrate that, once rate quantization is applied, the rest of the analogy is similar to the equivalence of space time space (STS) switching to time space time (TST) switching [7] for a fixed TDMA schedule.

We also discuss why achieving 100% throughput is difficult with multicast traffic in crossbar switches: We show that a Birkhoff decomposition based scheduling (see e.g., [8]) is not possible, since -unlike unicast scheduling- a traffic pattern is *not necessarily* sustainable, even if the matrix of arrival rates is within the convex hull of all possible matrices of multicast capable crossbar configurations. To further elaborate on the difficulty of multicast switch scheduling, we show that, if *fanout splitting* of multicast packets is not allowed, an extra speedup of 2 is necessary for 100% throughput. This is true even when the arrival rates are within the admissible region for mere unicast traffic.

Next, we discuss the properties of the frame scheduling analogous to non-blocking switching in Clos networks.We specifically focus on *strict* and *wide sense non-blocking* to reduce the complexity of circuit switching in Clos networks. For unicast traffic, strict sense non-blocking leads to an analogous scheduler based on *maximal matchings*, for which we give an $O(n^2)$ time complexity algorithm on an $n \times n$ crossbar

Can Emre Koksal is with the Department of Electrical and Computer Engineering, The Ohio State University, Columbus, OH (email: koksal@ece.osu.edu).

This work was supported by NSF grants 0635242 and 0916664. Portions of this work were presented at the 2008 IEEE International Workshop on Quality of Service (IWQoS 08), Enschede, The Netherlands.

switch. Also, exploiting the analogy for unicast traffic, we show that the result [9] that, *finding maximal matchings is sufficient to provide 100% throughput*[1] *with a speedup of 2*, becomes straightforward. For the multicast traffic we provide a simple non-blocking switch scheduling algorithm, which is analogous to an existing [10] wide sense non-blocking circuit switch for Clos networks. Combining it with rate quantization we show that, a speedup of $6 \log n / \log \log n$ is *sufficient* to achieve 100% throughput for all admissible multicast traffic. Even though this is only a (possibly tight) upper bound for the *necessary* speedup for 100% throughput (using any algorithm), the simplicity of the algorithm enables efficient distributed hardware implementations of the scheduler with a time complexity of $\mathrm{O}(n^2 \log n / \log \log n)$ per time slot, whereas finding the optimal schedule (with minimal speedup) is NP-hard. For both unicast and multicast schedulers, we illustrate tradeoffs between packet delay and necessary speedup. Since our main focus is exploring the necessary and sufficient resources to achieve 100% throughput, the delay-speedup tradeoff is the extent that we elaborate on the issue of delay.

The rest of the paper is organized as follows. After giving the model in Section II, we discuss rate quantization in Section III and state the basic theorems for rate quantization. We illustrate the analogy between crossbar switch schedulers and circuit switching Clos networks in Section IV. In Section V, we provide non-blocking unicast and multicast switch schedulers motivated by strict sense and wide sense non-blocking Clos networks. Finally, we summarize the results and discuss some other potential directions in Section VI.

II. Switch Model and Multicast Traffic

We consider the combined input and output queued (CIOQ) switch architecture with a single crossbar fabric. We assume that the crossbar fabric is *multicast capable*, i.e., an input can be connected to multiple outputs, but the inverse is not allowed. We call a given set of connections, a *switch configuration*.

We assume input and output links with identical capacities, and packets arriving over an input link are fragmented into fixed sized cells. We define a time slot as the time in which a cell can be transmitted over a link. In case an internal speedup, s, is used, up to s switch configurations can be set up in a time slot and hence up to s cells can be transferred to an output. Since more than 1 cell can be transferred to an output in a given time slot, output queueing as well as input queueing is necessary. Speedup $s = 1$ is also referred to as *no speedup*. We call the time in which a switch configuration remains active, a *schedule slot*. Hence a schedule slot is $1/s$ of a time slot.

Each cell arriving at an input queue has a *fanout set*, i.e., the set of the output links that the cell needs to be forwarded to. Unicast cells have a fanout set of unit cardinality. To avoid head of the line (HOL) blocking [11], we assume the presence of virtual output queueing (VOQ) at each input for every possible fanout set. As in [3], virtual output queueing at a per fanout set level is referred to as *multicast virtual output queueing* (MC-VOQ). In an $n \times n$ switch, for any given input, there exist $2^n - 1$ possible fanout sets. Due to this exponential growth, MC-VOQ has issues of scalability and consequently it is merely a theoretical tool used to investigate the limitations of input queued crossbar switches under multicast traffic.

A scheduler may choose not to place an arriving cell with a certain fanout set F directly to the associated MC-VOQ. It is also possible that it duplicates the cell and place one copy to the MC-VOQ with a fanout set F' and the other copy to the MC-VOQ with a fanout set F'' such that $F' \cup F'' = F$ and $F' \cap F'' = \emptyset$. Hence, these two copies are transferred to the corresponding outputs at possibly different times. This process is called *fanout splitting*.

We assume that the cell arrivals are rate ergodic and each MC-VOQ is associated with a certain cell arrival rate (before possible fanout splitting). For a given set of rates to be admissible, the total rate of cells arriving at each input link or destined to each output link cannot exceed 1 cell per time slot. Note that it may be possible that, after fanout splitting, the total rate of cells arriving at the MC-VOQs of an input exceed 1 cell per time slot.

Now let us consider frame based schedulers, which have the information of the cell arrival rates for each MC-VOQ. A frame is a (possibly non-periodic) collection of configurations. In an $n \times n$ switch, each configuration can be represented with an $n \times n$ *configuration matrix*, which has a single '1' in each column and all '0's otherwise. If the switch is not multicast enabled, then each configuration matrix is a permutation matrix.

First we focus on the case with only unicast cells. At each input there exist n virtual output queues, one for each output. These n^2 VOQ arrival rates can be written in the form of an $n \times n$ *rate matrix*[2]. In the context of packet switching, throughput is defined as the fraction of the capacity of the output links that can be utilized, if the input queues are completely backlogged (and hence the input links are fully utilized). Indeed, if all input queues are backlogged, then a switch is said to achieve 100% throughput if all output links can be fully utilized. It can be shown (see e.g., [8], [6]) that 100% throughput is achievable, if and only if R lies in the convex hull of the set of permutation matrices, i.e., there exists a frame, $\pi_1, \pi_2, \ldots, \pi_T$ of (containing possibly identical elements) of permutation matrices such that $R \leqslant \frac{1}{T} \sum_{k=1}^{T} \pi_k$ for some possibly infinite frame size T.

For multicast traffic, the *rate matrix* R is such that, R_{ij} is, as a fraction of the link capacity, the rate at which input i wants to be connected to output j. Hence it is possible that $\sum_{j=1}^{n} R_{ij} > 1$. For example, suppose in every time slot only input 1 receives cells each of which is to be broadcast to all outputs. Then $\sum_{j=1}^{n} R_{1j} = n$. On the other hand, for all output j, $\sum_{i=1}^{n} R_{ij} \leqslant 1$ under all possible admissible traffic matrices, since no output can be oversubscribed.

We can express a rate matrix R as a sum of $2^n - n$ matrices: $R = \sum_{m=1}^{2^n - n} R_m^{(\mathrm{f})}$. Here, one of the matrices, $R_1^{(\mathrm{f})}$ represents the rates for all unicast traffic (fanout set of unit cardinality)

[1] In fact [9] shows work conservation, which is stronger than achieving 100% throughput.

[2] This convention is also common in photonic switches (see e.g., [12], [13], [14]).

and the remaining $2^n - n - 1$ matrices represent the multicast rates (fanout set of cardinality $\geqslant 2$), one for each fanout set. For instance, if the rate of multicast cells arriving at input i, destined to outputs j and j' is r, then the matrix associated with the fanout set $\{j, j'\}$ has an r in locations (i, j) and (i, j'). We call this expansion of a multicast rate matrix R, the *fanout set expansion*.

One fundamental difference of the multicast traffic from the unicast traffic is that, in multicast, the existence of a frame, $c_1, c_2, \ldots, c_T$ of configuration matrices for which $R \leqslant \frac{1}{T}\sum_{k=1}^{T} c_k$ does not necessarily imply that 100% throughput is achievable. Consequently even with a multicast capable crossbar, some speedup $s > 1$ is necessary for 100% throughput. Following is an example.

Example 1: First consider the following unicast rate matrix:

$$R^{(u)} = \begin{bmatrix} 0.5 & 0 & 0.5 \\ 0.5 & 0.5 & 0 \\ 0 & 0.5 & 0.5 \end{bmatrix} = 0.5 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + 0.5 \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

Here, since there is only unicast traffic, fanout set expansion of $R^{(u)}$ is itself. Since $R^{(u)}$ can be written as the convex combination of two permutation matrices (i.e., unicast configuration matrices) with a weight 0.5, an equal time-share between the two associated configurations suffices to provide the desired service (and hence 100% throughput) for this traffic.

Next consider the following multicast rate matrix:

$$R^{(m)} = \begin{bmatrix} 0 & 0 & 0.5 \\ 0.5 & 0.5 & 0 \\ 0 & 0 & 0.5 \end{bmatrix} + \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

The first component of the above fanout set expansion contains the rates of the unicast cells. Here, there is a single class of multicast cells: half of the cells arriving at input 1 are multicast cells with a fanout set $\{1, 2\}$. Thus there is only one other component (for fanout set $\{1, 2\}$) other than the unicast component in the fanout set expansion. Even though the sum of the rates in the first row of R is 1.5, the total rate of the cells arriving at the first input is 1. Indeed, input links 1 and 2 are fully subscribed and no input or input or output link is oversubscribed under this traffic. Therefore to achieve 100% throughput with no speedup, at any point in time, input 2 must be connected to either output 1 or output 2, but not both, since all the cells are unicast at the second input. On the other hand input 1 needs to be connected to these two outputs simultaneously half the time to transfer multicast cells. This implies that these two outputs can be let free by the first input only half of the time as shown in Fig. 1, where the time period illustrated can be arbitrarily long. Consequently whenever the first input serves a multicast cell, input 2 must remain idle. However, since input 2 is fully utilized, some speedup is necessary.

The other alternative is the fanout splitting of the multicast cells. The total rate of cell arrivals at the VOQs of input 1 exceeds 1 after fanout splitting; hence without some speedup $s > 1$, it cannot be accommodated[3].

We conclude that without a speedup, R is not supportable, with or without fanout splitting. This is valid despite the fact that matrix R can be written as a convex combination,

$$R = 0.5 \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + 0.5 \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

of configurations matrices (R is in the convex hull of configuration matrices) for multicast enabled crossbar. This example illustrates that multicast scheduling problem can be more complicated than unicast scheduling. Even with a multicast capable crossbar, some speedup $s > 1$ is necessary for 100% throughput.

In this example, speedup $s = 1.5$ is necessary and sufficient for 100% throughput for the given traffic matrix. In [3] it was proved that the speedup necessary to achieve 100% throughput for all admissible multicast traffic grows unbounded with increasing switch size. It is also stated that "the numerical evaluation of the necessary speedup is prohibitive" and no scaling law has been given for the necessary speedup for 100% throughput. Also, finding the multicast schedule that works with the minimum necessary speedup is NP-hard as shown in [1], [2]. There are some obvious ways of simplifying multicast scheduling, such as ruling out fanout splitting. However, extra speedup is necessary to make up for the lost flexibility as shown in the following theorem. Note that the motivation of this theorem is *not* to calculate the sufficient speedup for 100% throughput without fanout splitting. It is merely to illustrate that, without fanout splitting, a larger speedup may be required to deliver 100% throughput under multicast traffic.

Theorem 1: If fanout splitting of multicast cells is not allowed in an $n \times n$ crossbar switch, then a speedup of $2 - \frac{1}{n}$ is necessary to achieve 100% throughput for a doubly stochastic multicast rate matrix R.

Before the proof of the theorem, note that, if fanout splitting is allowed, no speedup ($s = 1$) is required to support any doubly stochastic rate matrix. Indeed, with complete fanout splitting, every cell can be treated as a unicast cell. Since the row and column sums of this matrix is 1, 100% throughput is achievable by treating the problem as a unicast switch scheduling problem. Thus ruling out fanout splitting costs us some extra speedup or a reduced throughput at a fixed speedup.

Proof: Consider the following fanout set expansion:

$$R = \underbrace{\begin{bmatrix} 1/n & 1/n & \cdots & 1/n \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}}_{\text{unicast}} + \underbrace{\begin{bmatrix} 0 & 0 & \cdots & 0 \\ 1/n & 1/n & \cdots & 1/n \\ \vdots & \vdots & \ddots & \vdots \\ 1/n & 1/n & \cdots & 1/n \end{bmatrix}}_{\text{fanout set } \{1,\ldots,n\}}.$$

[3]Another issue with fanout splitting is that the order of the packets are not necessarily preserved as different packets of the same flow are transferred to the output through different MC-VOQs. To reorder packets, some queuing at the output is required. The analysis of these reordering queues is beyond the scope of this paper.

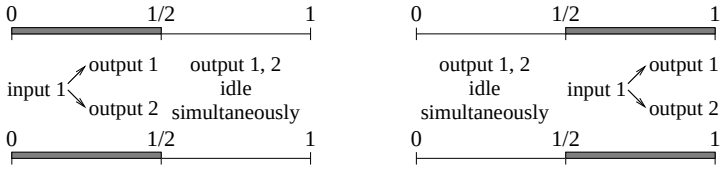


Fig. 1. No matter where the multicast flow is served, both outputs 1 and 2 will be idle simultaneously.

Here, the set of rates for which input 1 receives all unicast traffic with an equal rate of $1/n$ to every output. Every other input receives cells once every n time slots to be multicast to all outputs (broadcast). For all i, j, $R_{ij} = 1/n$ and consequently the overall rate matrix is doubly stochastic.

Since fanout splitting of broadcast cells is not allowed, one schedule slot must be occupied for each broadcast cell arriving at inputs 2 to n. Along with any broadcast cell, no unicast cell can be scheduled from any of the input 1 VOQs. Thus extra n schedule slots is necessary to accommodate input 1 traffic. As a result, to support this traffic, a total of $2n-1$ schedule slots is necessary in a span of n time slots, corresponding to a speedup of $2-\frac{1}{n}$, completing the proof.

III. Rate Quantization and Periodic Frame Schedulers

Every entry of a rate matrix R takes on any real non-negative value, as long as no input or output link is over-subscribed. Consequently a given frame scheduler can end up having a non-periodic schedule, $c_1, c_2, \ldots$ of infinitely many configuration matrices. We show that, using the concept of rate quantization, the frame schedule for a given R can be made periodic, even with some arbitrarily small speedup. We first introduce rate quantization for doubly stochastic unicast rate matrices, and then state the generalization to multicast rate matrices in a follow-up corollary. Note that the following theorem for unicast rate matrices was first introduced in [6], with no proof.

Theorem 2: Let R be an $n \times n$ doubly stochastic matrix and ε be a rational number, which can be written as $1/f$, where f is an integer. There exists an $n \times n$ matrix $Q = R' + U$, where

1) R' is a doubly-stochastic matrix with all entries integer multiples of ε;
2) for all $1 \leqslant i, j \leqslant n$, $U_{ij} = \varepsilon$, and $R_{ij} \leqslant Q_{ij} \leqslant R_{ij} + 2\varepsilon$.

Our proof is constructive: First, we introduce an algorithm to construct matrix R', (and thus the matrix Q) for a given R. Then we prove that the algorithm always ends up with the desired Q matrix. The details of the algorithm and the proof are given in the appendix. Here we give an example which illustrates the theorem, and at the same time provides intuition.

Example 2: Let $\varepsilon = 0.1$ and consider the following 3×3 doubly stochastic matrix:

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

$$\overset{(\varepsilon=0.1)}{\leqslant} \underbrace{\begin{bmatrix} 0.5 & 0.4 & 0.2 \\ 0.3 & 0.5 & 0.3 \\ 0.3 & 0.2 & 0.7 \end{bmatrix}}_{\text{rounded up}} \begin{matrix} \rightarrow & 1.1 \\ \rightarrow & 1.1 \\ \rightarrow & 1.2 \end{matrix} \quad (1)$$

$$\leqslant Q = \underbrace{\begin{bmatrix} 0.5 & 0.4 & 0.1 \\ 0.2 & 0.5 & 0.3 \\ 0.3 & 0.1 & 0.6 \end{bmatrix}}_{\text{doubly stochastic } R'} + \underbrace{\begin{bmatrix} 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 \end{bmatrix}}_{\text{constant matrix } U}. \quad (2)$$

Note that each entry of Q in (2) is within 2ε of the corresponding entry in R and the rows and the columns sum to $1 + \varepsilon n = 1.3$.

First we illustrate how rate quantization leads to a periodic switch schedule for unicast traffic, even for a speedup arbitrarily close to 1.

Corollary 1: For any doubly stochastic unicast rate matrix R and any given $\delta > 0$, there exist a sequence of $n + n/\delta$ permutation matrices, $c_1, c_2, \ldots, c_{n+\frac{n}{\delta}}$ such that

$$R \leqslant \frac{\delta}{n} \sum_{k=1}^{n+n/\delta} c_k.$$

Proof: The proof follows directly from Theorem 2. Choosing $\varepsilon = \delta/n$, one can find a matrix Q, whose rows and columns sum to $1 + \delta$ such that, for all $1 \leqslant i, j \leqslant n$, Q_{ij} is an integer multiple of $\varepsilon = \delta/n$ and $Q_{ij} \geqslant R_{ij}$. Thus, any sequence of unicast configuration matrices (i.e., permutation matrices) that supports rate matrix Q, will also support R. As shown in [8] and [6], the Birkhoff decomposition of Q terminates with $n/\delta + n$ permutation matrices (containing possibly identical elements) $c_1, c_2, \ldots, c_{n+\frac{n}{\delta}}$ each with a coefficient $\varepsilon = \delta/n$. We complete the proof noting that the first n/δ permutation matrices are due to R' and the remaining n of them[4] are due to the constant matrix U. Hence $R \leqslant Q = \frac{\delta}{n} \sum_{k=1}^{n+n/\delta} c_k$ and periodically repeating the sequence of these $n + \frac{n}{\delta}$ switch configurations with a period of $\frac{n}{\delta}$ time slots suffices to provide 100% throughput to any given set of admissible rates. The required speedup is $s = \left(n + \frac{n}{\delta}\right) / \frac{n}{\delta} = 1 + \delta$.

This corollary also illustrates the relationship between frame scheduling and cell scheduling. In particular, a cell scheduler capable of providing 100% throughput without speedup (e.g.,

[4]For instance, the identity matrix and $n-1$ cyclic shifts of it constitute a valid choice

based on maximum weight matching [15]) will choose a sequence of configurations $c'_1, \ldots, c'_{n/\delta}$ within the next n/δ time slots such that, for $\delta \ll 1$, $R_{ij} \approx \frac{1}{n/\delta} \sum_{k=1}^{n/\delta} c'_{k,ij}$ for all $1 \leqslant i, j \leqslant n$, since VOQ arrival processes are ergodic. Moreover, $Q_{ij} = \frac{1}{n/\delta} \sum_{k=1}^{n+n/\delta} c_k \leqslant R_{ij} + 2\delta$ for all $1 \leqslant i, j \leqslant n$, and hence, as $\delta \to 0$ the frame scheduler and the cell scheduler serve each VOQ at identical rates and achieve the same throughput. Note that this *asymptotic* equality between the schedules of the periodic frame scheduler and the cell scheduler has a *conceptual importance, rather than a practical one*. Indeed, the cell delay with the frame scheduler can be arbitrarily high, as $\delta \to \infty$.

We just showed that the 100% *throughput* can be achieved for a unicast rate matrix with a periodic frame scheduler for any speedup $s > 1$. For multicast rate matrices, quantization is slightly more complicated. To obtain a periodic schedule with a period $1/\varepsilon$, the rate of the unicast cells as well as every class of multicast cells need to be an integer multiple of ε. Therefore we quantize rates on a per MC-VOQ basis. The following theorem illustrates how per MC-VOQ rate quantization also leads to a periodic switch schedule for multicast traffic, for any speedup $s > 1$.

Theorem 3: Let the fanout set expansion for a multicast rate matrix R be given by $R = \sum_{m=1}^{2^n-n} R_m^{(\mathrm{f})}$. For any given $\delta > 0$, there exists a matrix Q such that

1) $\sum_{i=1}^{n} Q_{ij} \leqslant (1+\delta) \sum_{i=1}^{n} R_{ij}$ for all $1 \leqslant j \leqslant n$ and $\sum_{j=1}^{n} Q_{ij} \leqslant (1+\delta) \sum_{j=1}^{n} R_{ij}$ for all $1 \leqslant i \leqslant n$;
2) there exists a fanout expansion $Q = \sum_{m=1}^{2^n-n} Q_m^{(\mathrm{f})}$ such that, for all $1 \leqslant m \leqslant 2^n - n$, $Q_m^{(\mathrm{f})} \geqslant R_m^{(\mathrm{f})}$ and every entry of $Q_m^{(\mathrm{f})}$ is an integer multiple of $\delta/(n2^{n-1})$.

Proof: Here, we quantize the rate of every MC-VOQ. Let $\varepsilon = \delta/(n2^{n-1})$. For all $1 \leqslant m \leqslant 2^n - n$, every non-zero entry in $R_m^{(\mathrm{f})}$ is increased by a positive amount to the closest integer multiple of ε to form $Q_m^{(\mathrm{f})}$. Clearly, (2) is satisfied after the described quantization, since each entry of $Q_m^{(\mathrm{f})}$ is an integer multiple of ε and is no less than the corresponding entry of $R_m^{(\mathrm{f})}$. Now we need to show (1) for the new matrix $Q = \sum_{k=1}^{2^n-n} Q_m^{(\mathrm{f})}$. In the fanout expansion of R, there can be no more than 2^{n-1} matrices for which any given input-output pair (i, j) is non-zero. After quantization, each such entry is increased by an amount no more than δ. Since a fanout set has a maximum cardinality of n, each row i of matrix R is increased by a total of no more than $2^{n-1} \times n \times \varepsilon = \delta$. Hence $\sum_{j=1}^{n} Q_{ij} \leqslant (1+\delta) \sum_{j=1}^{n} R_{ij}$ for all $1 \leqslant i \leqslant n$. Repeating the same argument for each column j (we skip this for brevity), one can show both (1) and (2) can be satisfied simultaneously, applying rate quantization on each component of the fanout set expansion.

Note that, since the rate for every fanout set is made an integer multiple of some $\varepsilon > 0$, similar to the unicast scenario, the corresponding schedule of configuration matrices is periodic. We provide such a scheduler in Section V. The period of the schedule (and hence the cell delay) for a multicast rate matrix is, though, higher than that for a unicast rate matrix by a factor of 2^{n-1}.

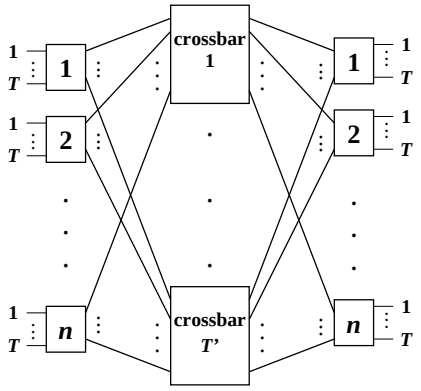


Fig. 2. Three stage Clos network.

In this section we showed that, with an arbitrarily small speedup $s = 1 + \delta$ used for rate quantization, there exists a frame scheduler that provides a periodic service of a period, inversely proportional to δ. We also argued that, along with that delay, long enough for "sufficient averaging" of the ergodic input traffic, one can find a cell scheduler which provides identical service rates as the frame scheduler. Periodic frame scheduling also leads to the Clos network analogy, which we study in the following section.

IV. Analogy between Crossbar Switch Schedulers and Clos Networks

A three-stage Clos network is a multistage switching architecture and it is specified using three parameters (T, n, T') as shown in Fig. 2. There are T' middle-stage crossbars of size $n \times n$ to connect n input stage switches of size $T \times T'$ to n output stage switches of size $T' \times T$. Clos networks have been traditionally used for circuit switching and a detailed treatment can be found in standard switching textbooks, e.g., [7].

Now consider a frame based scheduler with a schedule of period T'. Let the speedup be $s = T'/T$, so the switch goes through T' configuration matrices, $c_1, \ldots, c_{T'}$ within the frame of T time slots. The above scheduler is "time-analogous" to the *circuit switching* Clos network as illustrated in Fig. 3 for the case of unicast.

If input i of the crossbar switch requires to send k cells to output j within a frame of T slots, then in the associated Clos network, k of the input links of input stage switch i require to make circuit connections to k of the output links of the output stage switch j. If c_m has a '1' in position (i, j), then the mth middle-stage crossbar connects input stage switch i to output stage switch j. Consequently the middle-stage crossbars are set to have configurations $c_1, \ldots c_{T'}$. In a sense, the middle-stage crossbars of the analogous Clos network replicates the entire sequence of T' configurations of the original crossbar switch, from the top to the bottom in space. Note that configuration matrices can contain multicast connections as well, as illustrated in Fig. 4(a).

Any periodic frame schedule with a period T' corresponds to a *fixed circuit assignment* in the Clos network. The ratio of the number of middle-stage crossbars (i.e., number of output links of each input stage switch) to the number of input links

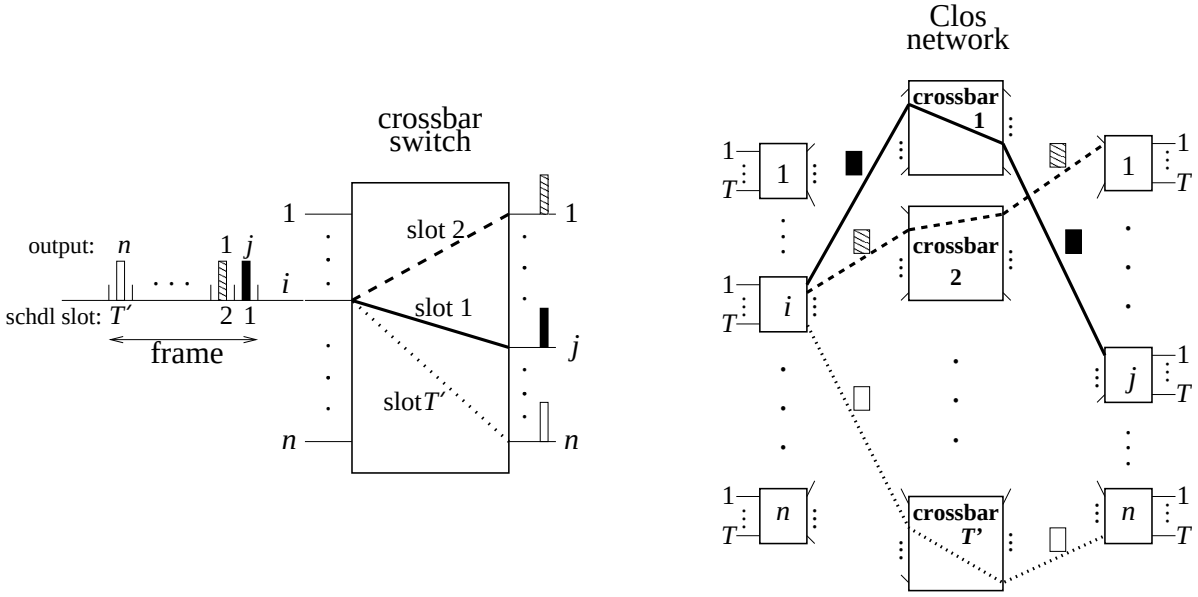


Fig. 3. The T' middle-stage crossbars of the Clos network goes through the entire frame of T' configurations of the crossbar switch schedule.

of each input stage switch is the speedup $s = T'/T$. The analogy is set for a periodic frame scheduler. For a non-periodic (infinitely long) schedule, one can still set the analogy after the application of rate quantization. As shown in the previous section, for any $\delta > 0$, one can choose $T = n2^{n-1}/\delta$ and $T' = (1+\delta)T$ to find the Clos network, analogous to the scheduler of the quantized matrix.

There is one more thing we need to describe to complete the analogy for the case of multicast. In the case of fanout splitting, different copies of a multicast cell are served over different configuration matrices within a frame. If fanout splitting is not allowed, each multicast cell can be served with only a single configuration over the frame. In the analogous Clos network, to replicate fanout splitting, multicast connections in input stage switches are used as illustrated in Fig. 4(b). If fanout splitting is not allowed, input stage switches are limited to only unicast configurations, since each cell (input link) can go through only one middle-stage crossbar. Note that Theorem 1 also shows that $2n-1$ middle-stage crossbars are necessary to support multicast circuit switching in a Clos network in which only point to point connections are allowed at the input and output stage switches.

Based on our analogy, instead of asking the question "what is the necessary speedup to support all admissible multicast traffic in a crossbar switch?" we ask "what is the necessary number of middle-stage crossbars to support multicast circuit switching in a Clos network?" The second question is also difficult and -to the best knowledge of the author- unanswered. However, things become simpler once we focus on different "grades" of *non-blocking Clos networks*, which lead to an interesting set of packet schedulers, as we shall discuss in the following section.

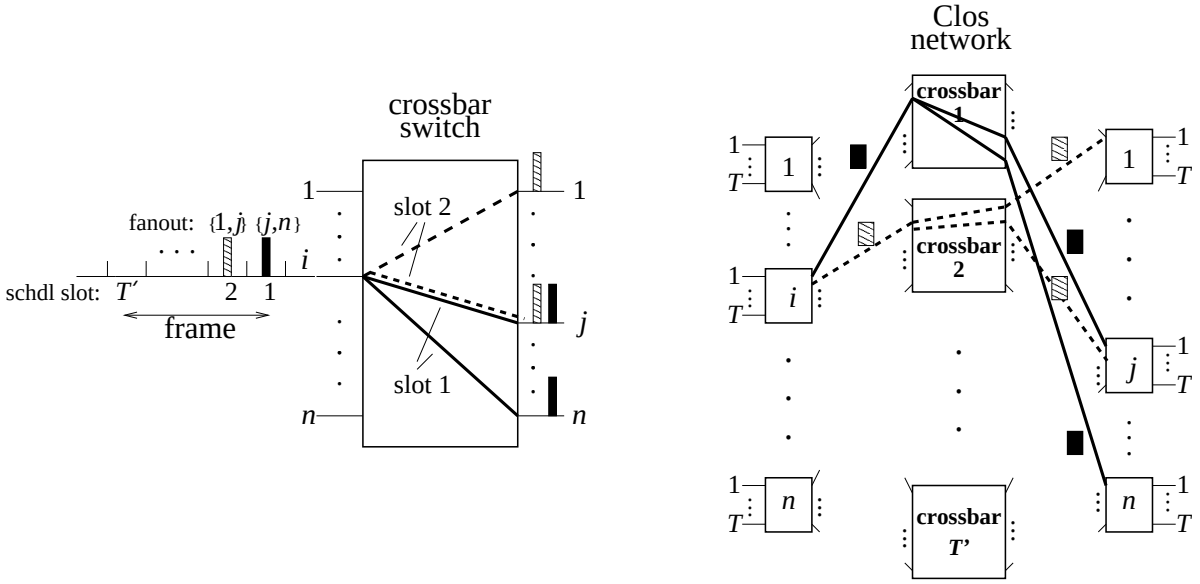
V. Non-Blocking Switch Scheduling for Multicast Traffic

In a switching network, blocking is the failure to satisfy certain set of connection requirements because of the absence of non-conflicting internal paths between the input links and the output links. A switching network is non-blocking if a connection can always be set up between any idle input and any idle output. There are multiple degrees of non-blocking.

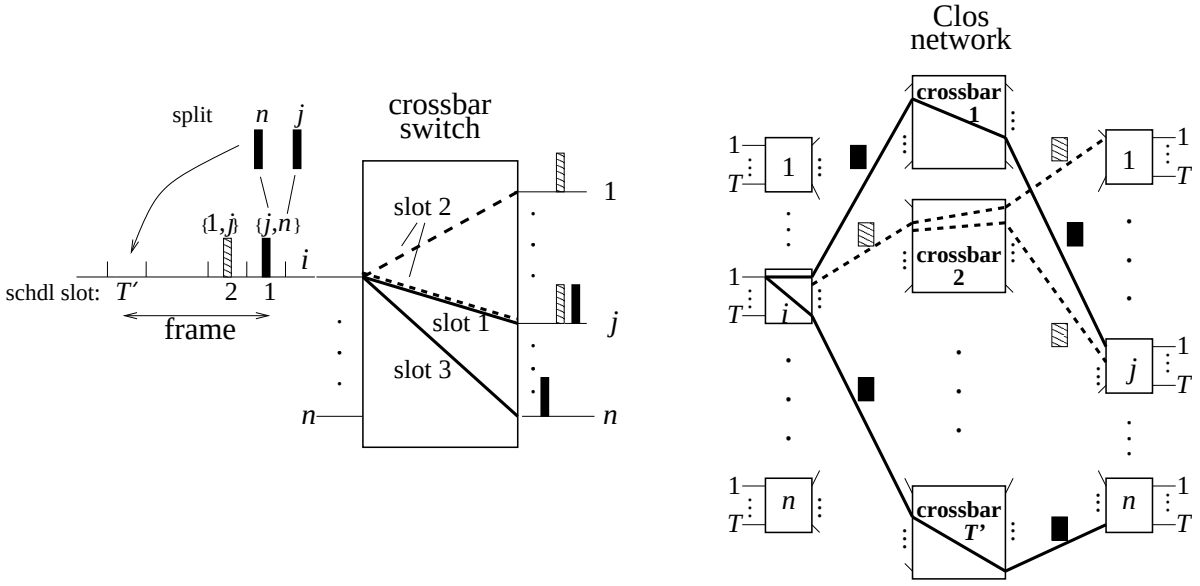
If a connection between an idle input-output pair cannot necessarily be established without rearranging the existing connections, the network is called rearrangeably non-blocking. For a Clos network to be rearrangeably non-blocking for unicast connections, the number of middle-stage crossbars need not be more than the number of input links per input stage switch, i.e., $T' = T$. This implies that no speedup is necessary for a frame scheduler (given a sufficiently high delay) to provide 100% throughput for any admissible unicast traffic. Indeed, Birkhoff von-Neumann switches [8] use a frame scheduler to achieve 100% throughput with no speedup. However, if a change occurs in some entries of R, a new schedule needs to be constructed. It is possible that the change cannot be accommodated with a minor modification in the schedule.

A network is strictly non-blocking if a connection between an idle input and an idle output can always be established, without the need for a rearrangement of the existing connections. If a (T, n, T') network is strictly non-blocking, then there exists a path between any idle input-output pair regardless of the existing configuration of the middle-stage crossbars. Thus, to satisfy an incoming connection request, a simple search for that middle-stage crossbar will be sufficient. It is well known [7] that, a three-stage Clos network, (T, n, T') is strictly non-blocking for unicast connections if and only if $T' \geqslant 2T - 1$.

The analogous scheduling interpretation of strictly non-



(a) No fanout splitting



(b) Fanout splitting

Fig. 4. In Fig. 4(a) the cell with the fanout set $\{j, n\}$ is sent in the first time slot without fanout splitting. In Fig. 4(b), the same cell is split and sent to outputs j and n at different times. In the associated Clos network, the input stage switches are not capable of multicast in the former scenario and are capable of multicast in the latter.

blocking is interesting, for unicast frame schedulers. First, consider a unicast rate matrix R such that each entry is an integer multiple of some $\varepsilon = 1/T$. There exists a periodic frame schedule, which repeats itself once every T schedule slots to provide 100% throughput for R. In the associated Clos network, the use of $T' = 2T - 1$ middle-stage crossbars "decouples" the circuit setup process for input stage switches, in that, each input stage switch is capable of finding paths to the output stage switches, independently of the paths that the other input stage switches set up.

Similarly, in the frame scheduler, the use of a schedule frame of size $2T - 1$ decouples the scheduling of cells at distinct inputs: For instance suppose an input link has a cell to be sent to output j. For a crossbar schedule, a strictly non-blocking associated Clos network guarantees that, among the frame of $2T - 1$ configuration matrices, the existence of a configuration matrix whose jth output is not already reserved by some other input link. Hence in order to achieve 100%

throughput, it suffices that each input independently makes an exhaustive search over the frame of configuration matrices for all the outputs that it possesses cells to send. This leads to a frame scheduler based on *maximal matching*s, as we will outline next. Note that the speedup necessary to quantize any given matrix to have entries, integer multiples of $\varepsilon = 1/T$ is $1+n/T$. Thus the total speedup necessary to achieve the above decoupling is $s = (2 - 1/T)(1 + n/T)$ for *any given* unicast rate matrix R. Note that the necessary speedup decreases as T is increased, at the expense of a longer frame and consequently a higher delay.

A. Unicast non-blocking switch scheduling algorithm

For a given unicast rate matrix R and some frame period $T > 1$, the scheduler goes through the following steps:

1) Apply rate quantization on R to obtain Q with all entries integer multiples of $1/T$ and the rows and the columns of Q sum to $1 + n/T$.
   Repeat (2)-(3) to construct configuration matrices $c_1, \ldots, c_{T'}$, until all entries of Q are set to 0. Initially $k := 1$ and all entries of $c_1, \ldots, c_{T'}$ are 0.
2) For the current configuration matrix c_k, each input i takes order (randomly or any arbitrary order) and chooses an output j that was not already claimed by another input $i' \neq i$ and also $Q_{ij} > 0$. If there are multiple such outputs, input i chooses one of them at random. Once the output j is picked, set $c_{k,ij} := 1$ and $Q_{ij} := Q_{ij} - 1/T$. An input, i, discards its turn to modify c_k, *only if* $Q_{ij} = 0$ for all j for which $c_{k,i'j} = 1$ for some $i' \neq i$.
3) Update $k := k + 1$.

This process leads to $T' \leqslant \lceil (2T - 1)(1 + n/T) \rceil$ *maximal matchings* represented by $c_1, \ldots, c_{T'}$ between the inputs and the outputs, since, there exists no c_k with all '0's in an entire row i and an entire column j, unless $Q_{ij} = 0$ at that point. Note that the non-blocking switch scheduling algorithm terminates with a Q matrix with all '0' entries. This is guaranteed by the fact that the Clos network, $(T(1 + n/T), n, (2T - 1)(1 + n/T))$, corresponding to our frame scheduler is strictly non-blocking. The significance of the result is that, we built a frame scheduler for a crossbar switch, based on the intuition we drew from the corresponding circuit switching problem in the Clos network.

The algorithm works for any frame size $T > 1$. As $T \to \infty$, the necessary speedup $s \to 2$ and the contribution of the portion of speedup required for quantization goes to 1. Moreover, as $T \to \infty$, the non-blocking switch scheduling algorithm becomes identical to a *cell scheduling* algorithm based on maximal matching. This proves a weaker version of a result, which was initially showed in [9]: for unicast input traffic, a speedup of 2 along with maximal matching is necessary and sufficient for work conservation (which is stronger than 100% throughput) as we just showed. Also an algorithm, LOOFA, with time complexity $O(n)$ is provided in [9] to obtain the desired maximal matchings. Note that, here, the tradeoff between the speedup and the delay is clear. The higher the frame size T, the delay increases proportionally, whereas the speedup goes down to 2.

Above, we described a sequential non-blocking switch scheduling algorithm, in which the configuration matrices are formed one by one. One can realize that this construction can be done online as the cells are transferred from the inputs and the outputs. To construct a configuration matrix, each input simply picks no more than a single entry that had not been claimed before. Including the search for that free output for each of the n inputs, the time complexity of the algorithm is $O(n^2)$ per schedule slot, and it can be implemented in a distributed fashion at different inputs.

Next, we state the *multicast non-blocking switch scheduling algorithm*. Then, we discuss the necessary speedup and complexity. Strictly non-blocking schedulers allow the possibility of full fanout splitting, which leads to a necessary speedup of n to achieve 100% throughput for all admissible multicast traffic. Consequently for multicast, we will exploit the notion of *wide sense non-blocking*[5]. Wide sense non-blocking in Clos networks is accompanied by a circuit switching algorithm to guarantee that an incoming connection to a free output link can be accommodated. In what follows we provide a scheduling algorithm, which is analogous to a circuit switching algorithm [10] for wide sense non-blocking Clos networks. We discuss the associated speedup later on.

B. Multicast non-blocking switch scheduling algorithm

The main difference between this algorithm and its unicast counterpart is the following. In the unicast version, each input takes turn to form configuration matrices in order, one by one. Here, again each input will take turns once every round, but the entire frame of T configuration matrices is constructed, not necessarily one by one, nor in any particular order. Instead, in every round, each input chooses an arbitrary fanout set for which it has non-zero rate of cells. Then, that input searches among the available configuration matrices for one, that can accommodate the highest number of connections for that particular fanout set. Note that, if there exists no matrix to accommodate the entire fanout set, only then fanout splitting is applied and different copies are served over different configuration matrices.

More precisely, for a given multicast rate matrix R and a given frame period $T > 1$ (we discuss how T is chosen later), the scheduler goes through the following steps:

1) Quantize all entries of every non-zero matrix $R_m^{(f)}$, $m = 1, \ldots, 2^n - n$ in the fanout set expansion of R to make them an integer multiple of some $1/T$. The quantized matrix Q satisfies $\sum_{i=1}^{n} Q_{ij} \leqslant 1 + n2^{n-1}/T$ for every column j.
   Repeat (2)-(3) to construct configuration matrices $c_1, \ldots c_{T'}$ (T' is unknown at this point), until all entries of Q_{ij} are set to 0. Initially all configuration matrices $c_1, \ldots c_{T'}$ contain all 0 entries.

[5]The analogous results to wide-sense non-blocking for unicast traffic include [16], [13] and [6]. In [6], it is shown that with some speedup $s > 1$, the frame delay can be reduced by a factor $O(n)$ for unicast traffic. Indeed, the delay is inversely proportional to $s - 1$, as the speedup s takes on values between 1 and 2.

2) Each input takes order (randomly or any arbitrary order). The current input, i, chooses an *arbitrary* element, $Q_m^{(\mathrm{f})}$, of the quantized fanout set expansion (possibly the unicast component) for which row i has a non-zero entry, i.e., $Q_{m,ij}^{(\mathrm{f})} \neq 0$ for some j. Let $\mathcal{F}(Q_m^{(\mathrm{f})})$ be the fanout set of that $Q_m^{(\mathrm{f})}$ and $F(c_k)$ be the set of outputs j, for which $c_{k,lj} = 0$ for *all* inputs l. Input i chooses the configuration matrix

$$c_k^* = \arg \max_{c_k \in \{c_1, \ldots c_{T'}\}} \left| \mathcal{F}(Q_m^{(\mathrm{f})}) \cap F(c_k) \right|. \qquad (3)$$

If multiple matrices solve (3), one of them is chosen randomly.

3) For all $j \in \mathcal{F}(Q_m^{(\mathrm{f})}) \cap F(c_k^*)$, input i sets $c_{k,ij}^* := 1$ and $Q_{m,ij}^{(\mathrm{f})} := Q_{m,ij}^{(\mathrm{f})} - 1/T$. Also, let the matrix $Q_{m'}^{(\mathrm{f})}$ be the component in the fanout set expansion for which $\mathcal{F}(Q_{m'}^{(\mathrm{f})}) = \mathcal{F}(Q_m^{(\mathrm{f})}) \setminus F(c_k^*)$. If $\mathcal{F}(Q_{m'}^{(\mathrm{f})}) \neq \emptyset$, it means fanout splitting has to be applied. To handle the remaining part of the fanout set that has been left unscheduled, set $Q_{m',ij}^{(\mathrm{f})} := Q_{m',ij}^{(\mathrm{f})} + 1/T$ for all $j \in \mathcal{F}(Q_{m'}^{(\mathrm{f})})$.

Even though multicast non-blocking switch scheduling algorithm forms the entire frame before scheduling, it can still be made online to work as a cell scheduler as follows. In every time slot, received cells at each input is placed in an appropriate MC-VOQ. Then, as in step (2), each input looks through all the configuration matrices that will be served within the next T' (unknown at this point) schedule slots in order to find the one that has the available set of outputs with the largest intersection set with the fanout set as in (3). That portion of the fanout set is scheduled to be transmitted with that matrix. The process is repeated for the remaining part of the fanout set, until the entire fanout set is covered for that cell. Therefore, any arriving cell is guaranteed to be transferred to its entire fanout set within the next T' schedule slots (corresponding to a worst case delay of T time slots) if T' is chosen appropriately. The choice of T' and T also determines the speedup $s = (1 + n2^{n-1}/T)(T'/T)$, where the first component of s is required for quantization and the second component is required for the algorithm. We make use of the following existing results in circuit switching in Clos networks to evaluate that second component of s.

Multicast non-blocking switch scheduling algorithm is analogous to the circuit switching algorithm for Clos networks, given in [17] and later in [10]. There, it shown that $T' = 3(T-1)\log n/\log\log n$ middle-stage crossbars is sufficient for the (T, n, T') Clos network to be non-blocking with the circuit switching counterpart of the above multicast non-blocking switch scheduling algorithm. Note that this value is a tight upper bound on the necessary speedup, since it is also shown that the necessary number of middle-stage crossbars grows as $\mathrm{O}(T \log n/\log\log n)$ with the switch size. Thus by choosing the frame size $T = \alpha n 2^{n-1}$, for some $\alpha > 1$, one can show that a speedup $s = 6\log n/\log\log n$ is sufficient for multicast non-blocking switch scheduling algorithm to accommodate any admissible multicast rate matrix. Consequently, for each arriving cell, the scheduler needs to search within the next $T' = 6T\log n/\log\log n$ schedule slots to find (3). In [4], which considers an internally buffered crossbar fabric, the sufficiency of the speedup $\Theta(\log n/\log\log n)$ is shown for a certain class of input traffic. With the analogy, we show that the same speedup scaling is sufficient, even for the unbuffered crossbar fabric.

The time complexity of the *software implementation* of multicast non-blocking switch scheduling algorithm is $\mathrm{O}(n^2T')$ per schedule slot: up to n cells are scheduled, and each search goes through up to possibly T' schedule matrices up to n times (since the cell may need to be fanout split up to n times). Each input needs to keep the schedule for every cell enqueued at its MC-VOQs. Since the delay is no more than T' for each cell, the required amount of memory *per cell* is no more than $\mathrm{O}(\log T')$. In order to keep the schedule for a frame of T cells, the total memory requirement for the switch schedule, including the possibility of fanout splitting is $\mathrm{O}(n^2T\log T')$.

Note however that, the "regularity" of the algorithm enables highly efficient, parallel *hardware implementations* as studied in [18] for circuit switching in Clos networks. There, authors divide the required operation of finding the desired middle-stage crossbar into some basic types of functions such as finding the cardinality of a set and comparison. A total of T' counters are used to keep the number of occupied output links of each middle-stage crossbar. These counters are connected to a parallel comparator array, which finds the minimum cardinality middle-stage crossbar among a candidate set. For each connection request between an input and an output stage switch, this process of counting and comparing to find the appropriate middle-stage crossbar is repeated for $\mathrm{O}(n\log n/\log\log n)$ times, requiring $\mathrm{O}(n\log n/\log\log n)$ gate propagations. If the same system is adopted for cell scheduling, it will have a time complexity of $\mathrm{O}(n^2\log n/\log\log n)$ per schedule slot (since n cells are scheduled in each schedule slot), much lower compared to the time complexity of the software implementation. However, the required amount of memory (hardware cost), $\mathrm{O}(T'(\log T + \log n))$, is slightly higher than that of the software implementation.

Lastly, we would like to mention that the implementation of the MC-VOQs is the main difficulty in realizing the algorithm in hardware or software. One way of implementing them is by using linked lists, i.e., all the cells arriving at an input are stored in a single buffer and each cell is assigned a pointer to keep track of its fanout set. In this implementation, each input requires a single buffer (as opposed to $2^n - 1$ separate ones), but the number of possible fanout sets that each input needs to keep track of is $2^n - 1$ per cell. This corresponds to an extra memory requirement of n bits per cell, which becomes impractical with increasing buffer sizes. Consequently, we would like to emphasize that the value of our scheduler is rather conceptual than it is practical.

Discussion: Unfortunately, parallel results do not exist for rearrangeably non-blocking multicast capable Clos networks. Using more complex schedulers compared to the given non-blocking switch scheduler, it may be possible to achieve an $s = \mathrm{o}(\log n/\log\log n)$ scaling for the required speedup. However, since for 100% throughput (for any scheduler) we know that the necessary speedup grows unboundedly as the size of the switch, it is not clear whether the extra effort for lowering the throughput is worth it. Indeed, we also know

that, achieving 100% throughput using the minimum speedup necessary is NP-hard, which is not the case with multicast non-blocking switch scheduling algorithm.

VI. Summary and Conclusions

In this paper we use the analogy between packet scheduling in crossbar switches and circuit switching in a three-stage Clos network to study a number of issues involving providing 100% throughput to all admissible unicast and multicast traffic over crossbar switches. To set up the analogy, we presented a theory of rate quantization.

We showed that for a crossbar switch of size $n \times n$, a speedup of $O(\log n / \log\log n)$ is sufficient to support 100% throughput for any admissible multicast traffic and we provided a scheduler for this task. For the scheduler, we exploited circuit switches for wide sense non-blocking Clos networks. The problem of multicast switch scheduling with minimum necessary speedup is NP-hard, whereas the time complexity of multicast non-blocking switch scheduling associated with the efficient hardware implementations of our scheduler is only $O(n^2 \log n / \log\log n)$ per time slot.

We also showed that if fanout splitting of multicast packets is not allowed, a speedup of 2 is necessary, even when the arrival rates are within the admissible region for unicast traffic, for which no speedup is necessary to provide 100% throughput. Thus, disabling the fanout splitting of multicast cells may not be an efficient solution for the complexity problem.

Based on the well-known equivalence between 3-stage Clos networks and frame-based scheduling, we revisited some problems in unicast switch scheduling. We illustrate that the well known result that "a speedup of 2 is necessary for 100% throughput for all admissible unicast traffic using maximal matching" becomes a straightforward by-product of the Clos network analogy using strict sense non-blocking. We believe the analogy between the scheduling problem in crossbar switches and non-blocking circuit assignment in Clos networks with wide sense non-blocking (see e.g., [19] for various theorems on wide sense non-blocking) can be insightful if one considers speedup values between 1 and 2.

Acknowledgment

I would like to thank Professor Robert G. Gallager and Dr. Charles E. Rohrs for all their support and many discussions on this work. I also thank the anonymous reviewers for their valuable feedback.

References

[1] M. Andrews, S. Khanna, and K. Kumaran, "Integrated Scheduling of Unicast and Multicast Traffic in an Input-Queued Switch," in *Proc. of the IEEE Infocom*, 1999.

[2] J. Sundararajan, S. Deb, and M. Medard, "Extending the birkhoff-von neumann switching strategy for multicast - on the use of optical splitting in switches," *IEEE Journal on Selected Areas in Communications*, vol. 25, pp. 36–50, Aug 2007.

[3] M. Marsan, A. Bianco, P. Giaccone, E. Leonardi, and F. Neri, "Multicast Traffic in Input-Queued Switches: Optimal Scheduling and Maximum Throughput," *IEEE Transactions on Networking*, vol. 11, pp. 465–476, 2003.

[4] P. Giaccone and E. Leonardi, "Asymptotic performance limits of switches with buffered crossbars," *IEEE Transactions on Information Theory*, vol. 54, pp. 595–607, Feb. 2008.

[5] A. Varma and S. Chalasani, "An Incremental Algorithm for TDM Switching Assignments in Satellite and Terrestrial Networks," *IEEE Journal on Selected Areas in Communications*, vol. 10, pp. 364–377, February 1992.

[6] C. Koksal, R. Gallager, and C. Rohrs, "Rate Quantization and Service Quality for Variable Rate Traffic over Single Crossbar Switches," in *Proc. of the IEEE Infocom*, (Hong Kong), Mar. 2004.

[7] J. Hui, *Switching and Traffi c Theory for Integrated Broadband Circuits*. Boston, MA: Kluwer Academic Publishers, 1990.

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

[9] P. Krishna, N. Patel, A. Charny, and R. Simcoe, "On the speedup required for work-conserving crossbar switches," *IEEE Journal on Selected Areas in Communications*, vol. 17, pp. 1057–1066, Jun 1999.

[10] Y. Yang and G. Masson, "The necessary conditions for clos type non-blocking multicast networks," *IEEE Trans. on Computers*, vol. 48, no. 11, pp. 1214–1227, 1999.

[11] M. Hluchyj, M. Karol, and S. Morgan, "Input versus output queueing on a space division switch," *IEEE Trans. on Communications*, vol. 35, pp. 1347–1356, Dec. 1987.

[12] I. Keslassy and M. Kodialam, "Scheduling schemes for delay graphs with applications to optical packet networks," in *in Proc. IEEE Work. High Perf. Switch. and Routing*, 2003.

[13] I. Keslassy, M. Kodialam, T. V. Lakshman, and D. Stiliadis, "On guaranteed smooth scheduling for input-queued switches," *IEEE/ACM Transactions on Networking*, vol. 13, pp. 1364–1375, December 2005.

[14] J.-F. P. Labourdette and A. S. Acampora, "Logically rearrangeable multihop lightwave networks," *IEEE Transactions on Communications*, vol. 39, pp. 1223–1230, August 1991.

[15] N. McKeown, A. Mekittikul, V. Anantharam, and J. Walrand, "Achieving 100% Throughput in an Input Queued Switch," *IEEE Transactions on Communications*, vol. 47, pp. 1260–1267, August 1999.

[16] B. Towles and W. Dally, "Guaranteed Scheduling for Switches with Configuration Overhead," in *Proc. of the IEEE Infocom*, (New York, NY), June 2002.

[17] Y. Yang and G. Masson, "Non-blocking broadcast switching networks," *IEEE Trans. on Computers*, vol. 40, no. 9, pp. 1005–1015, 1991.

[18] Y. Yang and G. M. Masson, "Fast Path Routing Techniques for Nonblocking Broadcast Networks," in *Proc. of the IEEE International Phoenix Conf. Computers and Comm.*, Apr. 1994.

[19] D. S. Kim and D. Z. Du, "Multirate multicast switching networks," *Elsevier Theoretical Computer Science*, vol. 261, pp. 241–251, June 2001.

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

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

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

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

Appendix I
Rate Quantization Algorithm and Proof of Theorem 2

Our algorithm generates matrix R' (and thus matrix Q) in two steps; in the first step ((1) in Example 2), a matrix, $\tilde{R}$, with all entries integer multiples of ε is constructed. Every entry of the original matrix is increased by some non-zero amount, so that they all become integer multiples of ε. The column sums of matrix $\tilde{R}$ are not necessarily identical.

In the second step ((2) in Example 2) sufficiently many entries of $\tilde{R}$ are reduced by ε to make the column sums of matrix R' equals 1. 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

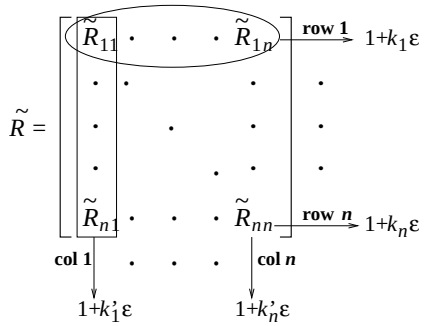


Fig. 5. The $n \times n$ matrix $\tilde{R}$ is illustrated. Each row i and column j sums to $1 + k_i\varepsilon$ and $1 + k'_j\varepsilon$ respectively, where k_i and k'_j are non-negative integers.

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 ε 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:

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

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 by ε no more than once.

Rate Quantization Algorithm: We first give the algorithm formally, and then a detailed explanation of each step follows.
Initial Values: Let k_m and k'_l be such that, $1+\varepsilon k_m$ and $1+\varepsilon k'_l$ are the mth row and lth column sum respectively, as illustrated in Fig. 5. 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) $R'_{ij} = \tilde{R}_{ij} - \varepsilon,\ k_i \rightarrow k_i - 1,\ k'_j \rightarrow k'_j - 1,\ E \rightarrow E - \{j\}$.
2) $i = \arg\max_{1 \leqslant l \leqslant n} k_l$

Setup: Given any ε, there exists a $\sigma_{ij},\ 0 < \sigma_{ij} \leqslant \varepsilon$ such that $R_{ij} + \sigma_{ij}$ is an integer multiple of ε 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 ε. By definition, 1 is also an integer multiple of ε, and thus, as illustrated in Fig. 5, we can represent the sum of the entries of the ith row and the jth columns $1+k_i\varepsilon$ and $1+k'_j\varepsilon$ respectively where k_i and k'_j are positive integers.

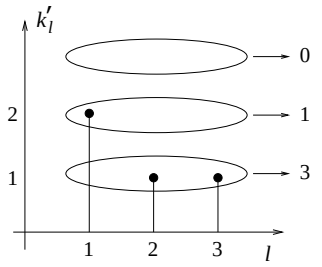


Fig. 6. Vector $\vec{k'} = [2\ 1\ 1]^T$, thus $\vec{q'} = [3\ 1\ 0]^T$.

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 ε 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 ε 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. We prove Theorem 2 constructively by proving that the described algorithm indeed ends up with matrix Q of the desired form. Note that 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 [20].

Lemma 1: Rate Quantization Algorithm successfully terminates with a matrix R' which is doubly stochastic.

Before we give the proof of the lemma, note that

$$k_i = \frac{1}{\varepsilon}\left(\sum_{j=1}^{n}(R'_{ij}) - 1\right), \text{ and } k'_j = \frac{1}{\varepsilon}\left(\sum_{i=1}^{n}(R'_{ij}) - 1\right).$$

We can represent k_i and k'_j as an entry of the vectors $\vec{k}$ and $\vec{k'}$ respectively. Let $q'_i,\ i \geqslant 1$ be the number of columns j, for which $k'_j \geqslant i$. For example, if $\vec{k'} = [2\ 1\ 1]^T$, then $\vec{q'} = [3\ 1\ 0]^T$ as illustrated in Fig. 6.

Proof: By induction. We shall first show that initially

$$q'_1 \geqslant k_i \tag{4}$$

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 ε) to make the row sum equal to 1. We will prove a more general version of (4):

$$\vec{k} \prec \vec{q'} \tag{5}$$

namely, the vector $\vec{k}$ is majorized by the vector $\vec{q'}$. For the definition see Appendix II or [21] for a complete treatment of majorization.

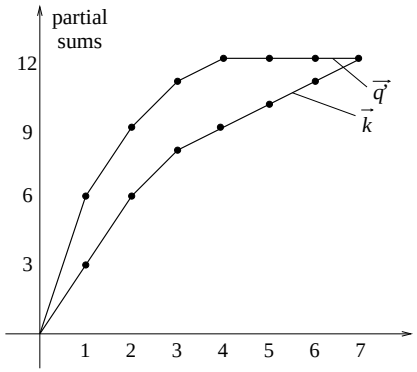


Fig. 7. Sample Lorentz curves for $\vec{q}'$ and $\vec{k}$ are illustrated. Since $\vec{q}' \succ \vec{k}$ initially, the partial sum curve of $\vec{q}'$, is above that of $\vec{k}$.

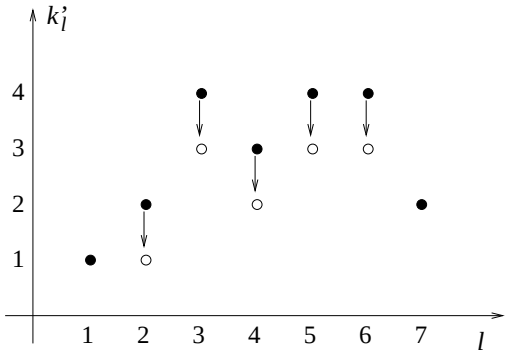


Fig. 8. If $k_i = 5$, the entries of the ith row that are decreased are illustrated above.

First we prove that (5) holds at the beginning of the algorithm. Recall that $\sigma = \tilde{R} - R$. Hence,

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

$\forall i, j$. Let the lth column vector of σ be $\vec{v}_l$ and thus $v_{l,j} = \sigma_{jl}$ and $\langle \vec{v}_l, \vec{e} \rangle = k_l'\varepsilon$, where $\vec{e} = [1 \cdots 1]^T$. From Kemperman's theorem [22], $\vec{v}_l$ is majorized by any vector for which k_l' entries are ε, and the other $n - k_l'$ entries are 0. Hence,

$$\vec{v}_l \prec [\underbrace{\varepsilon \cdots \varepsilon}_{k_l'} \ \underbrace{0 \cdots 0}_{n-k_l'}]^T \equiv \vec{v}_l^{\max} \tag{6}$$

Thus, the vector on the right side of (6) is the *maximal vector* (in the sense of majorization) of the set of vectors whose entries are between 0 and ε and $\langle \vec{v}, \vec{e} \rangle = k_l'\varepsilon$. Let us denote the maximal vector of the lth column vector by $\vec{v}_l^{\max}$.

Now, let us define a new matrix, $\frac{1}{\varepsilon}[\vec{v}_1^{\max} \cdots \vec{v}_n^{\max}]$, where each column is the maximal vector of the corresponding column of $\frac{1}{\varepsilon}\sigma$. Note that the vector of column sums for this new matrix is $\vec{k}'$, and thus the corresponding distribution will be $\vec{q}'$; 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., q_1'; $k_{\text{new},2}$ is the number of columns with $k_j' \geqslant 2$, i.e., q_2', and so on. More precisely, $k_{\text{new},i}$ is the number of columns j, for which $k_j' \geqslant i$. Thus,

$$k_{\text{new},i} = q_i' \tag{7}$$

But the vectors, $\vec{v}_l^{\max}$, $l \in \{1, \ldots, n\}$ are order symmetric (see [21] for the definition). Hence we get the desired result using Day's theorem [23]:

$$\vec{k}_{\text{new}} = \vec{q}' = \frac{1}{\varepsilon}\sum_{l=1}^{n} \vec{v}_l^{\max} \tag{8}$$

$$\succ \frac{1}{\varepsilon}\sum_{l=1}^{n} \vec{v}_l \tag{9}$$

$$= \vec{k} \tag{10}$$

We just showed that at the beginning of the algorithm, $\vec{q}' \succ \vec{k}$, and thus, $q_1 \geqslant k_i$, 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[6] of the two sequences are illustrated in Fig. 7. Such curves are called Lorentz curves and if, for two vectors, $\vec{v}^{\text{I}} \prec \vec{v}^{\text{II}}$, then the partial sum curve for $\vec{v}^{\text{II}}$ will always be above that of $\vec{v}^{\text{I}}$.

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 $\vec{k} \prec \vec{q}'$ 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 by-product, 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 (5) 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, $\vec{q}'$ 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 the 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 $\vec{k} \prec \vec{q}'$; therefore, there should be as many 0s in vector $\vec{q}'$ as there are in $\vec{k}$ (verified in Appendix II). Since there are at least $i-1$ 0s in $\vec{k}$, we have $q_{n-i+2}', \ldots, q_n' = 0$. At the beginning of the ith step, the entries of $\vec{q}'$ and $\vec{k}_{\downarrow}$ (the decreasing rearrangement of the entries of $\vec{k}$) can be listed as follows:

$$\begin{array}{cccccc} q_1' & \cdots & q_{r-1}' & q_r' & q_{r+1}' & \cdots \\ k_{M(i)} & \cdots & k_{M(i+r-2)} & k_{M(i+r-1)} & k_{M(i+r)} & \cdots \end{array}$$

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

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

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

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

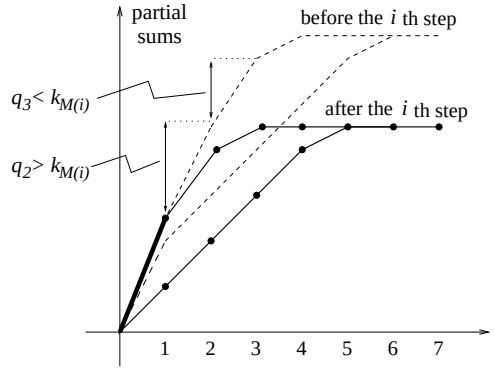


Fig. 9. In the ith step, $k_{M(i)}$ is removed (replaced with a 0) and the smallest entry of $\vec{q}'$ greater than or equal to $k_{M(i)}$ is reduced by $k_{M(i)} - q'_{r+1}$, and the following entry, q'_{r+1} is removed (replaced with a 0). The dashed curve is the initial curve, and the solid one is 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.

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 ε. Suppose this maximum column sum is $m\varepsilon$ 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 $\vec{q'}$ 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 q'_m, then after processing q'_m entries, q'_m will become 0 and $k_{M(i)} - q'_m$ entries will be left to be decreased at the row currently being processed. The algorithm will continue with the entries that have not been reduced before and with highest possible column sums. At this stage, the new value, $\hat{q}'_m$, of q'_m is 0 and the new value, $\hat{k}_{M(i)}$ of $k_{M(i)}$ is $k_{M(i)} - q'_m$. Note that q'_m potential entries have already been processed, and if $k_{M(i)}$ is greater than the second largest entry, q'_{m-1}, of $\vec{q'}$ then q'_{m-1} will be reduced to $\hat{q}'_{m-1} = q'_m$ but no further beyond that, since q'_m potential entries have already been processed. Similarly, each entry of $\vec{q}'$, which is smaller than $k_{M(i)}$ will be replaced with the next entry in order. Finally, the first entry, q'_r, in $\vec{q'}$ that is greater than $k_{M(i)}$ will be reduced by only $q'_r - k_{M(i)}$. Hence, after the ith row is processed, $\vec{q'}$ will have a 0 replacing q'_r, and a $\left[q'_{r+1} + \left(q'_r - k_{M(i)}\right)\right]$ replacing q'_{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. 8 assuming $\vec{k'} = [1\ 2\ 4\ 3\ 4\ 4\ 2]$, i.e., $\vec{q'} = [7\ 6\ 4\ 3\ 0\ 0\ 0]$ at the beginning of step i. If $k_{M(i)} = 5$, then at the end of step i, $\vec{q'} = [7\ 5\ 3\ 0\ 0\ 0\ 0]$. Notice that 6 is the smallest entry in $\vec{q'}$ 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, $\vec{k} \prec \vec{q}'$ 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 $\vec{q}'$ are illustrated in Figures 9. 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 $\vec{q}$ 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 $k_{M(i)} - q'_r$, remove q'_r, and attach the two separate parts. The two Lorentz curves intersect at 0 and at $\sum_l q'_l = \sum_l k_l$. Initially, these are the only two points they intersect, and the curve for $\vec{q}'$ 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. 9. Since the removed segment in $\vec{k}$ is to the left of the reduced segment of $\vec{q}'$, 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:

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

I

$$\begin{array}{ccc} & & \overbrace{}^{i-1} \\ q'_{r+2} & q'_{r+3} & \cdots \quad 0\cdots0 \\ k_{M(i+r+1)} & k_{M(i+r+2)} & \cdots \quad \underbrace{0\cdots0}_{i-1} \end{array}$$

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 $\vec{q}'$ will be unchanged up to q'_r. Thus, the partial sums will change in favor of $\vec{q'}$ by an extra $k_{M(i)}$ from the beginning all the way down to q'_r. This entry is replaced with $\left[q'_{r+1} + \left(q'_r - k_{M(i)}\right)\right]$, and the next entry, q'_{r+1} will be replaced with a 0 and removed from the second region. The total decrease in the partial sums of $\vec{q'}$ in the first region is $k_{M(i)}$. The extra $k_{M(i)}$ gained in favor of $\vec{q'}$ earlier by the removal of $k_{M(i)}$ from vector $\vec{k}$ is good enough to make up for this loss of $\vec{q}'$. The second region for both $\vec{q'}$ 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, (5) holds and $q'_1 \geqslant k_{M(i)}$, 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

$$0 = \sum_{i=1}^{n} k_i = \sum_{j=1}^{n} k'_j \tag{11}$$

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 ε.

Proof: The input matrix, $\tilde{R}$, of the algorithm already has all the entries integer multiples of ε. We complete the proof noting

that the change in each entry from $\tilde{R}$ to R' is an integer multiple of ε (either reduced by ε or left unchanged).

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

$$R'_{ij} \geqslant R_{ij} - \varepsilon, \quad 1 \leqslant i,j \leqslant n \tag{12}$$

Proof: Note that

$$\tilde{R}_{ij} \geqslant R_{ij}, \quad 1 \leqslant i,j \leqslant n \tag{13}$$

Since the algorithm reduces every entry by at most ε,

$$R'_{ij} \geqslant \tilde{R}_{ij} - \varepsilon, \quad 1 \leqslant i,j \leqslant n \tag{14}$$

Inequality (12) is immediate by (13) and (14).

Combining Lemma 1 and 3, we proved that R is the doubly stochastic matrix, all entries of which is an integer multiple of ε. Since U is defined as the constant matrix, composed of $U_{ij} = \varepsilon$ for all input-output pairs (i,j), all entries of matrix Q are integer multiples of ε and the rows and columns of Q sum to $1 + \varepsilon n$. Moreover, Lemma 4 shows that $Q_{ij} = R'_{ij} + \varepsilon \geqslant R_{ij}$ for all input-output pairs (i,j). Since during Rate Quantization Algorithm, no entry of R is increased by more than 2ε to construct matrix Q, we can write $R_{ij} \leqslant Q_{ij} \leqslant R_{ij} + 2\varepsilon$. Therefore, putting all three Lemmas, 1, 3 and 4, together, Theorem 2 is proved. In [20] we prove that Lemma 1 holds even if Rate Quantization 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.

APPENDIX II
BASIC DEFINITIONS IN MAJORIZATION

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

$$\sum_{i=1}^{k} x_{[i]} \leqslant \sum_{i=1}^{k} y_{[i]}, \ k = 1, \ldots, n-1 \tag{15}$$

$$\sum_{i=1}^{n} x_{[i]} = \sum_{i=1}^{n} y_{[i]} \tag{16}$$

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.

$$\left(\frac{1}{n}, \ldots, \frac{1}{n}\right) \prec \left(\frac{1}{n-1}, \ldots, \frac{1}{n-1}, 0\right) \prec \cdots \prec \left(\frac{1}{2}, \frac{1}{2}, 0, \ldots, 0\right) \prec (1, 0, \ldots, 0)$$

Suppose

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

Subtracting both sides of (15) from S, we get

$$\sum_{i=k+1}^{n} x_{[i]} \geqslant \sum_{i=k+1}^{n} y_{[i]}, \ k = 1, \ldots, n-1 \tag{17}$$

which is equivalent to (15). Hence, if $\vec{x}$ and $\vec{y}$ both have non-negative entries, there are at least as many 0s in $\vec{y}$ as in $\vec{x}$.

Can Emre Koksal received the B.S. degree in electrical engineering from the Middle East Technical University, Ankara, Turkey, in 1996, and the S.M. and Ph.D. degrees from the Massachusetts Institute of Technology (MIT), Cambridge, in 1998 and 2002, respectively, in electrical engineering and computer science. He was a Postdoctoral Fellow in the Networks and Mobile Systems Group in the Computer Science and Artificial Intelligence Laboratory, MIT, until 2003 and a Senior Researcher jointly in the Laboratory for Computer Communications and the Laboratory for Information Theory at EPFL, Switzerland, until 2006. Since then, he has been an Assistant Professor in the Electrical and Computer Engineering Department, Ohio State University, Columbus. His general areas of interest are wireless communication, computer networks, information theory, stochastic processes, and financial economics.