

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

Can Emre Koksal, *Member*

Abstract— We show an isomorphism between packet scheduling in crossbar switches and circuit switching in three-stage Clos networks, using the concept of rate quantization. We use the analogy for a crossbar switch of size $n \times n$ to construct a simple packet scheduler of complexity $O(n \log n)$ based on maximal matching. We show that, with this simple scheduler, a speedup of $O(\log n / \log \log n)$ is necessary to support 100% throughput for any admissible multicast traffic. If fanout splitting of multicast packets is not allowed, we show that an extra speedup of 2 is necessary, even when the arrival rates are within the admissible region for mere unicast traffic. Also 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.

I. INTRODUCTION

Applications requiring QoS support for *multicast traffic* remain to be 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] that “optimal scheduling” of multicast packets is NP-hard over a crossbar switch and in [2] 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 when the crossbar switch is *multicast capable*, i.e., it is capable of connecting an input to multiple outputs. It is also stated in [2] that the numerical evaluation of the necessary speedup is prohibitive and no scaling law has been given as to how the speedup scales with the switch size. In this paper¹ we present a simple algorithm to provide 100% throughput for all admissible multicast traffic and specify a scaling law for the necessary speedup to achieve 100% throughput.

Our main tool in the development 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 [4] 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 arrival (admissible) rates. Note however that,

as stated in [2], 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, using the notion of rate quantization [5]. We also prove the main theorem of [5], which was stated there without a proof. Ultimately, the Clos network analogy will be valid not only for frame based schedulers, but for all schedulers.

After giving the model in Section II, we discuss rate quantization in Section III and state the basic theorem for rate quantization. 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 [6] as we illustrate in Section IV. We also discuss why achieving 100% throughput is difficult with multicast traffic in crossbar networks. We show that a Birkhoff decomposition based scheduling (see e.g., [7]) 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. Moreover, we show that, if *fanout splitting* of multicast packets is not allowed, even when the arrival rates are within the admissible region for mere unicast traffic, a speedup of 2 is necessary for 100% throughput.

Then we discuss the properties of the frame scheduling analogous to non-blocking switching in Clos networks in Section V. We specifically focus on *strict sense non-blocking*, which solves the problem of complexity of circuit switching and hence the complexity of scheduling for both unicast and multicast traffic. Indeed, we show that the analogous of circuit switching in a strict sense non-blocking Clos network is *maximal matching* based scheduling in the crossbar switch. Consequently the result [8] for unicast traffic that, maximal matching is sufficient to provide 100% throughput² with a speedup of 2, becomes straightforward. Then we evaluate the speedup necessary for the support of multicast traffic with maximal matching to be $O(\log n / \log \log n)$ using an analogous result [9] for multicast circuit switching in Clos networks. This is a -possibly tight- upper bound for the minimum speedup necessary for 100% throughput (using any algorithm). Note however that, the associated complexity of maximal matching for multicast is $O(n \log n)$, whereas

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

¹Part of these results were presented in the 5 page “short position paper” [3] in IWQoS 2008.

²In fact [8] shows work conservation, which is stronger than 100% throughput.

scheduling complexity is NP-hard for optimal scheduling. 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 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. In a given time slot up to s cells can be transferred to an output and in that case at least $s - 1$ of them must be stored at the corresponding output queue. 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. We assume links with identical capacities and packets arriving over an input link are fragmented into fixed sized cells.

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 [10], we assume the presence of virtual output queueing (VOQ) at each input for every possible fanout set. Further, virtual output queueing at a per fanout set level is referred to as *multicast virtual output queueing* (MC-VOQ) in [2]. Note that in an $n \times n$ switch, for a 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 IQ 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 a 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 different times. We call this process *fanout splitting*.

We assume that the cell arrivals are rate ergodic and each MC-VOQ is associated with a certain cell arrival rate (after 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 VOQs of an input exceed 1 cell per time slot.

Now we 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.

Let us first focus on the case with only unicast cells. In this scenario, at an input, there are 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* R . It can be shown (see e.g., [7], [5]) 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, $\pi_1, \pi_2, \dots, \pi_T$ of (containing possibly identical elements) permutation matrices such that $R \leq \frac{1}{T} \sum_{k=1}^T \pi_k$ for some possibly infinite 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 instance 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 , $\sum_{i=1}^n R_{ij} \leq 1$, since no output can be oversubscribed.

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

Example 1: Consider the following rate matrix:

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

The first and the second component contain the rates of the unicast cells and the rate of the multicast cells respectively. Thus half of the cells arriving at input 1 are multicast cells with a fanout set of $\{1, 2\}$ and the other half are unicasts with the destination 3. 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, no input or output link is oversubscribed under this traffic. In fact input links 1 and 2 are fully subscribed. Therefore to achieve 100% throughput without any 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. With fanout splitting, the total rate of cell arrivals at the VOQs of input 1 exceeds 1; consequently some speedup is necessary to accommodate them. We conclude that without a speedup, R is not supportable, with or without fanout splitting.

This example illustrates that multicast scheduling problem is much more complicated than unicast scheduling. Even with a multicast capable crossbar, some speedup is necessary for 100% throughput. This is valid despite the fact that matrix R

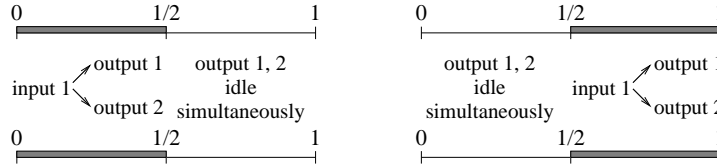


Fig. 1. No matter where the multicast flow is served, both outputs 1 and 2 will be idle simultaneously.

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 valid configurations matrices (R is in the convex hull of configuration matrices) for multicast enabled crossbar.

In this example, speedup $s = 1.5$ is necessary and sufficient for 100% throughput for the given traffic matrix. In [2] 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]. There are 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.

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 support multicast traffic for which the rate matrix R is a doubly stochastic matrix.

Before the proof of the theorem, note that, if fanout splitting is allowed, no speedup ($s = 1$) is required to support a doubly stochastic rate matrix. A complete fanout splitting is sufficient to achieve 100% throughput. Thus ruling out fanout splitting costs us some extra speedup or reduced throughput at a fixed speedup.

Proof: Consider 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 a broadcast cell, no unicast cell can be scheduled from any of the input 1 VOQs. Thus extra n scheduling 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.

Even without the possibility of fanout splitting, it is still not obvious how to build a switch scheduler for multicast traffic. Instead, we directly focus on an existing simple scheduler. We show that a speedup of $O(\log n / \log \log n)$ along with

maximal matching ($O(n \log n)$ complexity) is sufficient for 100% throughput.

III. RATE QUANTIZATION

Note that each entry of the rate matrix R can be any real positive number, as long as no input or output link is oversubscribed. Consequently a given frame scheduler can end up having a non-repetitive schedule, c_1, c_2, \dots of infinitely many configuration matrices. Now we present the notion of rate quantization (first introduced in [5]) and prove (not given in [5]) the following theorem in the appendix.

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

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 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 about the construction of the algorithm.

Example 2: We consider a 3×3 doubly stochastic matrix R . Let $\varepsilon = 0.1$.

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

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

$$\leq \underbrace{\begin{bmatrix} 0.5 & 0.4 & 0.1 \\ 0.2 & 0.5 & 0.3 \\ 0.3 & 0.1 & 0.6 \end{bmatrix}}_{\text{doubly stochastic}} + \begin{bmatrix} 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 \end{bmatrix}. \quad (2)$$

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

There are two main implications of rate quantization. The first one is that, with rate quantization, one can find a periodic switch schedule for any given R subject to a small speedup: First consider unicast traffic. Using Theorem 2, we know that there exists a matrix Q with rows and columns sum to $1 + \delta$

such that Q_{ij} is an integer multiple of $\varepsilon = \delta/n$ for all $1 \leq i, j \leq n$. As shown in [7] and [5], one can find a sequence $c_1, c_2, \dots, c_{n+n/\delta}$ consisting of possibly identical elements such that

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

Consequently, 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. Clearly this requires a speedup of $s = 1 + \delta$.

This example also illustrates the equality of frame scheduling and cell scheduling along with a certain delay. In particular, if cells are queued initially for $\frac{n}{\delta}$ time slots, a cell scheduler capable of providing 100% throughput without speedup (e.g., the maximum weight matching [11]) will choose a sequence of configurations $c'_1, \dots, c'_{n/\delta}$ within the next n/δ time slots such that, for $\delta \ll 1$, $R \approx \frac{1}{n/\delta} \sum_{k=1}^{n/\delta} c'_k \approx \frac{1}{n+n/\delta} \sum_{k=1}^{n+n/\delta} c_k$ since the VOQ arrival processes are ergodic. As $\delta \rightarrow 0$, the frame scheduler and the cell scheduler serve each VOQ at identical rates.

For multicast, quantization is slightly more complicated. Since up to $2^n - 1$ different types of cells (in terms of their fanout sets) arrive at each input, it is not clear how an increase in the rate after quantization will be split between them. For instance suppose $R_{ij} = 0.23$ for some i, j pair, where half of these cells are unicast and the other half are multicast with a fanout set $\{j, j'\}$, for some $j' \neq j$. In this case, with a choice of $\varepsilon = 0.1$, it is not possible to divide the extra service between the two flows such that the service rate for each MC-VOQ is increased to an integer multiple of 0.1. With this, it is not necessarily possible to find a periodic switch schedule with a period $n/0.1$.

Therefore, we quantize rates on a per MC-VOQ basis this time so that the rate of each MC-VOQ is increased to an integer multiple of $\frac{\delta}{n(2^{n-1}-1)}$. Consequently each entry of the quantized matrix R' is also an integer multiple of $\frac{\delta}{n(2^{n-1}-1)}$ as well. Since there are $2^{n-1} - 1$ MC-VOQs whose fanout set include a given input, each entry, R'_{ij} , can be as high as δ/n higher than the associated entry, R_{ij} , of the original matrix. Thus the sum of each column of Q is upper bounded by $1 + \delta$. Consequently with an extra speedup of $1 + \delta$ over the necessary speedup for 100% throughput (which we do not know yet), a periodic schedule can be found to support the desired set of multicast rates. Note that, for multicast the period of the schedule is higher with a factor of $2^{n-1}/n$ compared to the associated schedule period for unicast traffic. Consequently as $\delta \rightarrow 0$, for every multicast frame scheduler of period $2^{n-1}/\delta$, there corresponds a cell scheduler that provides identical service rates subject to an initial delay of $2^{n-1}/\delta$ time slots.

In summary, we showed that, given a speedup $1 + \delta$, for any given (multicast) rate matrix R , one can find a periodic frame schedule with period inversely proportional to δ , where δ can be chosen arbitrarily small. Moreover, along with a delay long enough (inversely proportional to δ) for ‘‘sufficient averaging’’ of the input traffic, one can find a cell scheduler

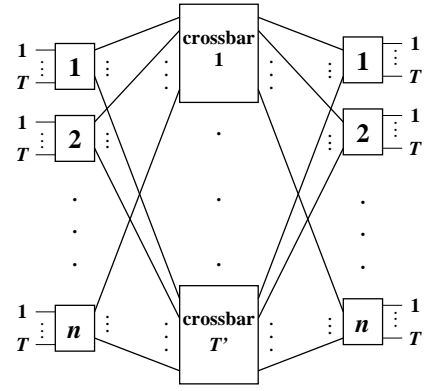


Fig. 2. Three stage Clos network.

which provides identical service rates as the frame scheduler.

The other implication of rate quantization is the Clos network analogy, which we study in the following section.

IV. CROSSBAR SWITCH SCHEDULERS AND CLOS NETWORKS

A three-stage Clos network 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 $T' \times T$.

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, $c_1, \dots, c_{T'}$ within the frame of T cell slots. The above scheduler is ‘‘time-analogous’’ to the following *circuit switching* Clos network:

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 m th middle-stage crossbar connects input stage switch i to output stage switch j . This is illustrated in Fig. 3. Consequently the middle stage crossbars are set to have configurations $c_1, \dots, c_{T'}$. In a sense the middle-stage crossbars of the analogous Clos network replicates the entire sequence of T' configurations of the crossbar switch in space from the top to the bottom.

Any frame consisting of T' configurations corresponds to a *fixed circuit assignment* in the Clos network. The ratio of the number of middle stage crossbars to the number of input links of each input stage switch is the speedup $s = T'/T$. The analogy can be made for any given rate matrix R , since, as described in Section III, for an arbitrarily low speedup of $s = 1 + \delta$, we can form a frame of finite size T (proportional to δ^{-1}) to provide 100% throughput for any given R arbitrarily closely. Note that in the Clos network, the size of the input and output stage switches grows with T .

There is one more thing we need to describe to complete the analogy for the case of multicast. In crossbar switching, in the case of fanout splitting, different copies of a multicast cell is served over different configuration matrices within a frame. If fanout splitting is not allowed, each multicast cell can be

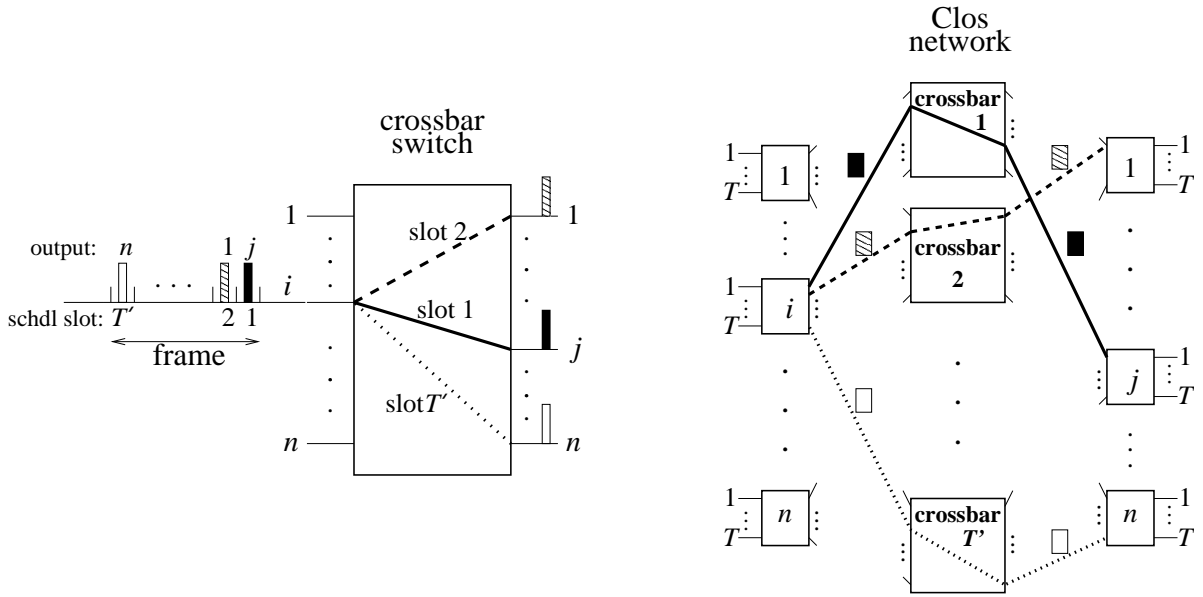


Fig. 3. The T' middle stage crossbars of the Clos network goes through the entire frame of T' configurations of the crossbar switch schedule.

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. If fanout splitting is not allowed, input stage switches are only capable of unicast connections 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 is 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 switches 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 strict sense non-blocking in Clos networks and it corresponds to an interesting set of schedulers based on maximal matching in switch scheduling as we discuss in the following section.

V. NON-BLOCKING SWITCH SCHEDULING AND MULTICAST SUPPORT

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 rearrangably non-blocking. For a Clos network to be rearrangably 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 [7] 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. 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 [6] that, a three-stage Clos network, (T, n, T') is strictly non-blocking for unicast connections if and only if $T' \geq 2T - 1$.

The analogous scheduling interpretation of strictly non-blocking is interesting. At each input, there exists a frame of T unicast cells to be sent to over of the output links. For a given T large enough for sufficient averaging of the arrival process, a speedup of $2 - \frac{1}{T}$ decouples the scheduling of cells at distinct inputs: For instance suppose an input has a cell to be sent to output j . It can search over the existing $2T - 1$ configuration matrices for one, whose j th output is not already reserved by some input. Each input can take turns in completely assigning their cells to one of the scheduling matrices and at the end of the process, it is guaranteed that every single cell will be assigned to a matrix.

Alternatively, one can construct the $T' = 2T - 1$ configuration matrices one by one as follows. For each matrix, every input (takes order and) chooses one cell destined to an output for which another cell (from another input) is not already

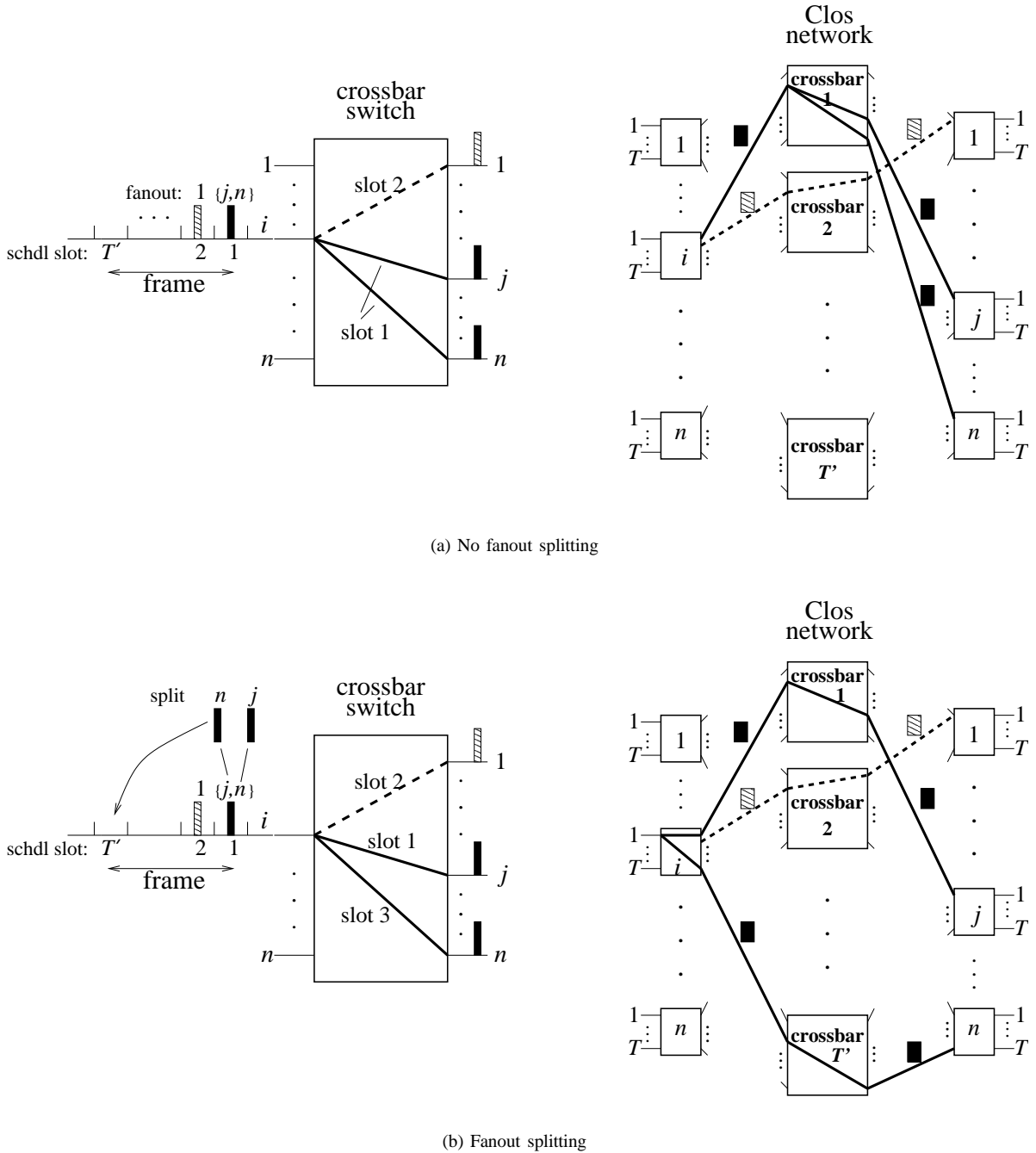


Fig. 4. In Fig. 4(a) the cell with the fanout set $\{j, n\}$ is sent in the first time slot without fanout splitting. In Fig. 4(b), the same cell is split and sent to outputs j and n at different times. In the associated Clos network the input stage switches are not capable of multicast in the former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

destined to. An input discards a matrix, only if it has no cell for the available (not reserved by other inputs) outputs of the current matrix being constructed. This process leads to $2T - 1$ maximal matchings between inputs and the outputs since, for each matrix, no input and output both of which are idle are left unmatched if there exists a cell demanding that connection. Using the analogy, we just proved a result, which was initially showed in [8]: Since $T' = 2T - 1$ middle-stage crossbars is necessary and sufficient for strictly non-

blocking Clos networks, a speedup of $\frac{T'}{T} = 2 - \frac{1}{T}$ is necessary and sufficient for 100% throughput for unicast traffic with maximal matching. Another important thing to note is that the complexity of maximal matching is $O(N)$ per time slot for unicast traffic.

In complexity, there is no difference between multicast cells and unicast cells for the construction of a single configuration matrix using maximal matching. To construct each configuration matrix, every input takes turn to assign a cell to be

scheduled for a transmission. This time there are multicast cells as well as unicast cells. Now suppose at an input there exists a multicast cell with a fanout set F ; however only the set of outputs $F' \subset F$ is available for the matrix currently being constructed. Then the fanout set is split into two sets, F' and $F \setminus F'$ and $|F'|$ copies of the cell is multicast to the set of outputs F' and the remaining part is placed in the corresponding MC-VOQ to be scheduled in another matrix.

In Theorem 1 of [9], it is shown that there exists a connection request pattern in a (T, n, T') Clos network such that an incoming feasible connection request cannot be met unless $T' \geq \Theta(T \log n / \log \log n)$. Consequently, in an $n \times n$ crossbar switch, a speedup of $s = O(\log n / \log \log n)$ is necessary to achieve 100% throughput for multicast traffic with maximal matching. The number of configuration matrices constructed per time slot is proportional to s , the scheduling complexity is $O(sn) \approx O(n \log n)$ per time slot.

Unfortunately, parallel results do not exist for rearrangeably non-blocking multicast capable Clos networks. Therefore, $s = O(\log n / \log \log n)$ is only an upper bound for the minimum necessary speedup, s_n , for 100% throughput for all admissible multicast traffic. However at a speedup s_n , the switch scheduling problem has been shown to be NP-hard, which is obviously not the case with maximal matching.

Finally note that there exists another degree of non-blocking between rearrangeably non-blocking and strictly non-blocking: If the necessity for rearranging the existing connections can be avoided using some algorithm, the Clos network is called wide sense non-blocking. As one would expect, the number of middle stage crossbars to support unicast traffic with wide-sense non-blocking is between T and $2T$. The analogous results to wide-sense non-blocking in crossbar scheduling include [12], [13] and [5]. In [5], it is shown that with some speedup $s > 1$, the complexity of frame scheduling algorithms can be reduced with a factor $O(n)$ for unicast traffic. Moreover, the complexity is inversely proportional to the speedup, as the speedup takes on values between 1 and 2. Again, to the best knowledge of the author, similar results do not exist for multicast traffic.

VI. SUMMARY AND CONCLUSIONS

In this paper we use the analogy between cell scheduling in crossbar switches and circuit switching in a three-stage Clos network to study a number of issues involving multicast support over crossbar switches.

We showed that for a crossbar switch of size $n \times n$, using maximal matching, a speedup of $O(\log n / \log \log n)$ is necessary to support 100% throughput for any admissible multicast traffic. Maximal matching is appealing for multicast traffic, because of its simplicity: The problem of multicast switch scheduling with minimum necessary speedup is NP-hard, whereas the complexity of switch scheduling associated with maximal matching for multicast traffic is only $O(n \log n)$ 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.

Thus, disabling the fanout splitting of multicast cells may not be an efficient solution for the complexity problem.

Also we revisit 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 with 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., [14] for various theorems on wide sense non-blocking) is insightful if one considers speedup values between 1 and 2.

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] M.A. 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.
- [3] C.E. Koksal, “On the Speedup Required to Achieve 100% Throughput for Multicast over Crossbar Switches,” in *Proceedings of IEEE IWQoS*, 2008.
- [4] Anujan Varma and Suresh 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.
- [5] C.E. Koksal, R.G. 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.
- [6] Hui J. Y., *Switching and Traffic Theory for Integrated Broadband Circuits*, Kluwer Academic Publishers, Boston, MA, 1990.
- [7] W.J. Chen, C.S. Chang, and H.Y. Huang, “On Service Guarantees for Input Buffered Crossbar Switches: A Capacity Decomposition Approach by Birkhoff and von Neumann,” in *Proceedings of IEEE IWQoS*, 1999.
- [8] P. Krishna, N.S. Patel, A. Charny, and R.J. Simcoe, “On the speedup required for work-conserving crossbar switches,” *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 6, pp. 1057–1066, Jun 1999.
- [9] Y. Yang and G.M. Masson, “The necessary conditions for clos type non-blocking multicast networks,” *IEEE Trans. on Computers*, vol. 48, no. 11, pp. 1214–1227, 1999.
- [10] M.G. Hluchyj, M.J. Karol, and S. Morgan, “Input versus output queueing on a space division switch,” *IEEE Trans. on Communications*, vol. 35, pp. 1347–1356, Dec. 1987.
- [11] 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.
- [12] B. Towles and W.J. Dally, “Guaranteed Scheduling for Switches with Configuration Overhead,” in *Proc. of the IEEE Infocom*, New York, NY, 2002.
- [13] I. Keslassy, Kodialam Murali, T.V. Lakshman, and D. Stiliadis, “On Guaranteed Smooth Scheduling for Input-Queued Switches,” in *Proc. of the IEEE Infocom*, San Fransisco, CA, 2003.
- [14] Dongsoo S. Kim and Ding Zhu Du, “Multirate multicast switching networks,” *Elsevier Theoretical Computer Science*, vol. 261, pp. 241–251, 2001.
- [15] Koksal C. E., “Providing QoS over High Speed Electronic and Optical Networks,” 2002, PhD Dissertation.
- [16] Marshall A. W. and Olkin I., *Inequalities: Theory of Majorization and Its Applications*, Academic Press, New York, NY, 1979.
- [17] Kemperman J. H. B., “Moment Problems for Sampling Without Replacement, I, II, III,” *Nederl. Akad. Wetensch. Proc. Ser.*, vol. 76, pp. 149–188, 1973.
- [18] Day P. W., “Rearrangement Inequalities,” *Canad. J. Math.*, vol. 24, pp. 930–943, 1972.

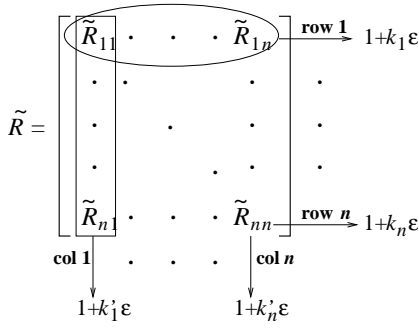


Fig. 5. The $n \times n$ matrix \tilde{R} is illustrated. Each row i and column j sums to $1 + k_i \varepsilon$ and $1 + k'_j \varepsilon$ respectively, where k_i and k'_j are non-negative integers.

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 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:

$$\begin{array}{ccc} \left[\begin{array}{ccc} 0.4 & 0.4 & 0.2 \\ 0.3 & 0.4 & 0.3 \\ 0.3 & 0.2 & 0.7 \end{array} \right] & \begin{array}{l} \rightarrow 1 \\ \rightarrow 1 \\ \rightarrow 1.2 \end{array} \\ \downarrow \quad \downarrow \quad \downarrow & \\ 1 & 1 & 1.2 \end{array}$$

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.

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 m th row and l th column sum respectively, as illustrated in Fig. 5. Let $i = \arg \max_{1 \leq l \leq n} k_l$ and $R' = \tilde{R}$

Repeat (1)-(2) until $k_i = 0$ for all $i \leq n$.

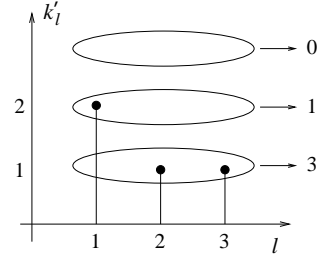


Fig. 6. Vector $\vec{k}' = [2 \ 1 \ 1]^T$, thus $\vec{q}' = [3 \ 1 \ 0]^T$.

- 1) Set $E = \{1, \dots, n\}$. 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) $i = \arg \max_{1 \leq l \leq n} k_l$

Setup: Given any ε , there exists a σ_{ij} , $0 < \sigma_{ij} \leq \varepsilon$ such that $R_{ij} + \sigma_{ij}$ is an integer multiple of ε for all $1 \leq i, j \leq n$. 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 i th row and the j th columns $1 + k_i \varepsilon$ and $1 + k'_j \varepsilon$ 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 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 [15].

Lemma 1: The 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 , $i \geq 1$ be the number of columns j , for which $k'_j \geq i$. 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

$$q'_1 \geq k_i \quad (3)$$

for all i , $1 \leq i \leq n$. Thus, for any \vec{k} and for $i = \arg \max_{1 \leq l \leq n} k_l$ 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 (3):

$$\vec{k} \prec \vec{q}' \quad (4)$$

namely, the vector \vec{k} is majorized by the vector \vec{q}' . For the definition see Section II or [16] for a complete treatment of majorization.

First we prove that (4) holds at the beginning of the algorithm. Recall that $\sigma = \tilde{R} - R$. Hence,

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

$\forall i, j$. Let the l th 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 $\vec{e} = [1 \dots 1]^T$. From Kemperman's theorem [17], \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,

$$\vec{v}_l \prec \left[\underbrace{\varepsilon \dots \varepsilon}_{k'_l} \underbrace{0 \dots 0}_{n-k'_l} \right]^T \equiv \vec{v}_l^{\max} \quad (5)$$

Thus, the vector on the right side of (5) 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 l th column vector by \vec{v}_l^{\max} .

Now, let us define a new matrix, $\frac{1}{\varepsilon} [\vec{v}_1^{\max} \dots \vec{v}_n^{\max}]$, 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}_{new} . Thus, $k_{\text{new},1}$ is the number of columns with $k'_j \geq 1$, i.e., q'_1 ; $k_{\text{new},2}$ is the number of columns with $k'_j \geq 2$, i.e., q'_2 , and so on. More precisely, $k_{\text{new},i}$ is the number of columns j , for which $k'_j \geq i$. Thus,

$$k_{\text{new},i} = q'_i \quad (6)$$

But the vectors, \vec{v}_l^{\max} , $l \in \{1, \dots, n\}$ are order symmetric (see [16] for the definition). Hence we get the desired result using Day's theorem [18]:

$$\vec{k}_{\text{new}} = \vec{q}' = \frac{1}{\varepsilon} \sum_{l=1}^n \vec{v}_l^{\max} \quad (7)$$

$$\succ \frac{1}{\varepsilon} \sum_{l=1}^n \vec{v}_l \quad (8)$$

$$= \vec{k} \quad (9)$$

We just showed that at the beginning of the algorithm, $\vec{q}' \succ \vec{k}$, and thus, $q_1 \geq k_1$, for all $i \leq n$. That is, the first step of the algorithm can be executed successfully to make the first row sum to 1. The partial sums³ of the two sequences are illustrated in Fig. 7. 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 .

³The m th 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 the corresponding partial sum of \vec{v}^I .

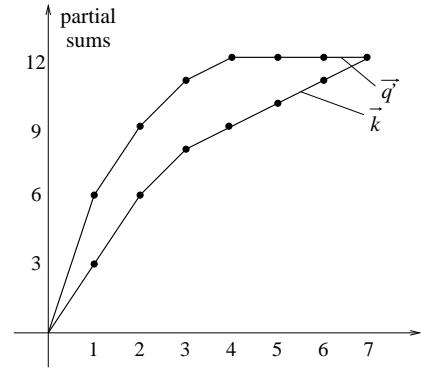


Fig. 7. Sample Lorentz curves for \vec{q}' and \vec{k} are illustrated. Since $\vec{q}' \succ \vec{k}$ initially, the partial sum curve of \vec{q}' , is above that of \vec{k} .

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 i th step, $1 \leq i \leq n - 1$ and show that it still holds at the end of the i th 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 (4) 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, $k_{M(1)}, \dots, k_{M(i-1)} = 0$, where $M(q)$ is the q th 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 $q'_{n-i+2}, \dots, q'_n = 0$. At the beginning of the i th step, the entries of \vec{q}' and \vec{k} (the decreasing rearrangement of the entries of \vec{k}) can be listed as follows:

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

$$\begin{array}{c} \dots \\ \dots \\ \dots \end{array} \begin{array}{c} \underbrace{0 \dots 0}_{i-1} \\ \underbrace{0 \dots 0}_{i-1} \end{array}$$

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 i th step, the only change in \vec{q}' is that the entries q'_r and q'_{r+1} will be replaced with $[q'_{r+1} + (q'_r - k_{M(i)})]$ 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 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

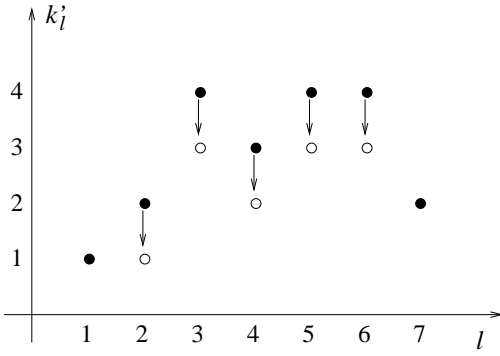


Fig. 8. If $k_i = 5$, the entries of the i th row that are decreased are illustrated above.

$k'_j = m$ by 1. Thus, the only change in \vec{q}' will be in the smallest non-zero entry, q'_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 i th row is processed, \vec{q}' will have a 0 replacing q'_r , and a $[q'_{r+1} + (q'_r - k_{M(i)})]$ replacing q'_{r+1} . Note that at the end of the i th step, \vec{k}_\perp will be the same except $k_{M(i)}$ will be replaced with a 0.

This process is illustrated in Fig. 8 assuming $\vec{k}^i = [1\ 2\ 4\ 3\ 4\ 4\ 2]$, i.e., $\vec{q}^i = [7\ 6\ 4\ 3\ 0\ 0\ 0]$ at the beginning of step i . If $k_{M(i)} = 5$, then at the end of step i , $\vec{q}^i = [7\ 5\ 3\ 0\ 0\ 0\ 0]$. Notice that 6 is the smallest entry in \vec{q}^i 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 i th step of the algorithm. But before that we present a graphical illustration of what happens in the i th 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

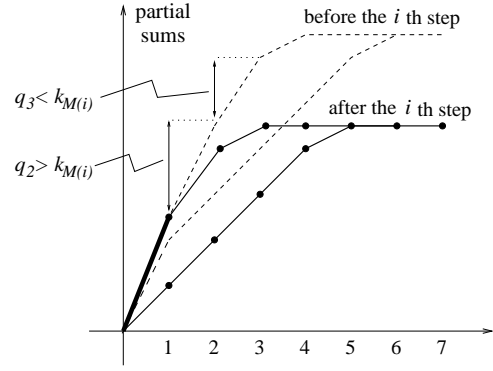


Fig. 9. In the i th step, $k_{M(i)}$ is removed (replaced with a 0) and the smallest entry of \vec{q}' greater than or equal to $k_{M(i)}$ is reduced by $k_{M(i)} - q'_{r+1}$, and the following entry, q'_{r+1} is removed (replaced with a 0). The dashed curve is the initial curve, and the solid one is the one at the end of the i th step. The bold segment is the one which do not change. The distance between the two curves does not decrease at all.

always above the curve for \vec{k} , otherwise. We need to show that this is the case after the i th 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 i th step. We can prove this statement as follows. There are two regions we need to consider as shown in the following table:

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

$$\begin{array}{cccc} q'_{r+2} & q'_{r+3} & \cdots & \underbrace{0 \cdots 0}_{i-1} \\ k_{M(i+r+1)} & k_{M(i+r+2)} & \cdots & \underbrace{0 \cdots 0}_{i-1} \end{array} \quad \text{II}$$

At the end of the i th step, the partial sums of the two sequences are as follows. In region I, at the end of the i th 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 $[q'_{r+1} + (q'_r - k_{M(i)})]$, 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, (4) holds and $q'_1 \geq k_{M(i)}$, for all $i \leq n$. 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 $i \leq n$. But, since

$$0 = \sum_{i=1}^n k_i = \sum_{j=1}^n k'_j \quad (10)$$

and $k'_j \geq 0$, for all $j \leq n$, it is also true that $k'_j = 0$, for all $j \leq n$ 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 counterpart in R decreased by ε :

$$R'_{ij} \geq R_{ij} - \varepsilon, \quad 1 \leq i, j \leq n \quad (11)$$

Proof: Note that

$$\tilde{R}_{ij} \geq R_{ij}, \quad 1 \leq i, j \leq n \quad (12)$$

Since the algorithm reduces every entry by at most ε ,

$$R'_{ij} \geq \tilde{R}_{ij} - \varepsilon, \quad 1 \leq i, j \leq n \quad (13)$$

Inequality (11) is immediate by (12) and (13).

Putting all three Lemmas, 1, 3 and 4, together, we completed the proof of correctness for the rate quantization algorithm, and thus Theorem 2 is also proved. In [15] 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.

APPENDIX II

BASIC DEFINITIONS IN MAJORIZATION

For any $\vec{x} = (x_1, \dots, x_n) \in \mathfrak{R}^n$, let

$$x_{[1]} \geq \dots \geq x_{[n]}$$

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

$$\vec{x}_{\downarrow} = (x_{[1]}, \dots, x_{[n]})$$

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

$$\sum_{i=1}^k x_{[i]} \leq \sum_{i=1}^k y_{[i]}, \quad k = 1, \dots, n-1 \quad (14)$$

$$\sum_{i=1}^n x_{[i]} = \sum_{i=1}^n y_{[i]} \quad (15)$$

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.

$$\begin{aligned} \left(\frac{1}{n}, \dots, \frac{1}{n}\right) &\prec \left(\frac{1}{n-1}, \dots, \frac{1}{n-1}, 0\right) \prec \dots \\ &\prec \left(\frac{1}{2}, \frac{1}{2}, 0, \dots, 0\right) \prec (1, 0, \dots, 0) \end{aligned}$$

Suppose

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

Subtracting both sides of (14) from S , we get

$$\sum_{i=k+1}^S x_{[i]} \geq \sum_{i=k+1}^S y_{[i]}, \quad k = 1, \dots, n-1 \quad (16)$$

which is equivalent to (14). 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} .