# On Guaranteed Smooth Scheduling For Input-Queued Switches

I. Keslassy* Murali Kodialam† T. V. Lakshman† Dimitrios Stiliadis†

* Computer Systems Laboratory
Stanford University
Stanford, CA 94305-9030
keslassy@stanford.edu

† Bell Laboratories
101 Crawfords Corner Road
Holmdel, NJ 07733
{muralik, lakshman, stiliadi}@bell-labs.com

***Abstract*— Input-queued switches are used extensively in the design of high-speed routers. As switch speeds and sizes increase, the design of the switch scheduler becomes a primary challenge, because the time interval for the matching computations needed for determining switch configurations becomes very small. Possible alternatives in scheduler design include increasing the scheduling interval by using envelopes [1], and using a frame-based scheduler that guarantees fixed rates between input-output pairs. However, both these alternatives have significant jitter drawbacks: the jitter increases with the envelope size in the first alternative, and previously-known methods do not guarantee tight jitter bounds in the second.**

**In this paper, we propose a hybrid approach to switch scheduling. Traffic with tight jitter constraints is first scheduled using a frame-based scheduler that achieves low jitter bounds. Jitter-insensitive traffic is later scheduled using an envelope-based scheduler. The main contribution of this paper is a scheduler design for generating low-jitter schedules. The scheduler uses a rate matrix decomposition designed for low jitter and different from the minimum-bandwidth Birkhoff-Von Neumann (BV) decomposition. In addition to generating low-jitter schedules, this decomposition yields fewer switch configuration matrices ($O(n)$) than the BV decomposition ($O(n^2)$), and so uses far less high-speed switch memory. We develop an efficient algorithm for decomposing the rate matrix and for scheduling the permutation matrices. We prove that our low-jitter algorithm has an $O(log\ n)$ factor bound on its bandwidth consumption in comparison to the minimum-bandwidth BV decomposition. Experimentally, we find that the bandwidth increase in practice is much lower than the theoretical bound. We also prove several related performance bounds for our scheduler. Finally, we propose a practical bandwidth-guaranteed algorithm, and show how our findings could even be extended to systems with large tuning time.**

## I. Introduction

Most high-speed core routers today use input-queueing with a crossbar switch fabric, virtual output queues and fixed-size cells At each time-slot, a scheduler finds a matching between the inputs and the outputs, and configures the crossbar according to this matching. Many heuristic algorithms have been proposed for finding this matching [2], [3], [4]. However, the need for routers with more ports and faster line rates makes these algorithms difficult to scale. For instance, with line rates of 40Gbps (OC768) and 64-byte cells, an algorithm would have to compute a matching every 12.8 ns, while the current highest-capacity commercially available centralized scheduler takes about 50 ns per matching. At least four alternatives have been proposed in the literature in order to decrease the frequency of the matching computation. First, by increasing the cell size through the use of envelopes [1]. Second, by using pipelining [5]. Third, by using several times the same matching, possibly with slight changes [6], [7].

In this paper, we will be focusing on a fourth alternative: frame-based scheduling. In this approach, at the beginning of each frame period (for instance, every 100 time-slots), the scheduler is assigned a *guaranteed rate table (GRT)*, which is a list of rate requirements. Then, it computes a list of matchings called the *schedule table*, such that during the following frame period, the schedule table will provide any input/output pair with at least as many services as the GRT would guarantee. Finally, the switch fabric transfers the input-queued cells according to the matchings of the schedule table.

In order to compute the schedule table, most of the algorithms in the literature are based on the Birkhoff Von-Neumann (BV) decomposition [8], [9], [10], [11]. As explained in section 2, this BV algorithm minimizes the bandwidth requirement for the schedule and provides bandwidth guarantees.

It can be noted that this frame-based scheduling approach applies to several systems other than input-queued routers. First, it is used in SONET all-optical circuit-switches, which switch cells in circuit-based frames by using delay lines [12]. Second, it applies to the satellite TDMA scheduling problem [13]. Third, it is of interest in time-slotted wireless FDMA systems where any station can communicate with any other according to a TDM schedule often established by a centralized scheduler. And finally, it is also useful in star-based WDM broadcast-and-select optical systems with tunable transmitters and fixed receivers (or fixed transmitters and tunable receivers), where a centralized scheduler assigns a specific wavelength to a specific station for any time-slot [14]. Therefore, although this paper will exclusively consider this problem in the context of input-queued router scheduling, most of the findings extend to these other systems.

For input-queued routers, this frame-based approach has several advantages. First, it solves the scheduling-frequency scalability issue mentioned above, which is one of the main bottlenecks in designing faster core routers. Second, it can be adapted to implement SONET, ATM, DiffServ, MPLS, and most (virtual) circuit- and frame- based schemes, which are predominant in the Internet core with which any Internet core router needs to interact. Third, it is one of the only input-queued policies that provably guarantee 100% throughput for both a router in isolation [9], [10], [11] and a network of routers [16], and therefore guarantees to the operators that

* This work was done while the author was with Bell Labs.

the lines they expensively laid will be fully utilized. Fourth, the method for computing the frame weights is often flexible. Therefore, this flexibility can be used to provide specific bandwidth guarantees for any high-priority input/output pair, which could correspond to a specific customer. This flexibility can also be used to ensure fairness among flows, capping for instance the bandwidth usage of aggressive users. Finally, this method provides equal guarantees to uniform as well as non-uniform traffic, contrarily to many scheduling heuristics based on uniform traffic.

However, frame-based scheduling suffers from an important drawback: it often results in large cell delays and large delay variability (jitter). This is especially true for an algorithm such as BV, which does not take delays into account. The objective of this paper is to define, study and approximate a class of *smooth* frame-based scheduling, which minimizes the jitter resulting from the schedule table.

For instance, consider the following simplistic example. Assume that for every frame period of 100 time-slots, we need to schedule the following GRT:

$$GRT = \begin{pmatrix} 50 & 50 \\ 0 & 50 \end{pmatrix}.$$

In this GRT, input 1 needs to send 50 cells to output 1 and 50 cells to output 2. Input 2 only needs to send 50 cells to output 2. Call $M_1$ the matching connecting input 1 to output 1 and input 2 to output 2, and call $M_2$ the matching connecting input 1 to output 2 and input 2 to output 1. A straightforward implementation of BV, as explained later, would lead to a schedule table first implementing 50 times $M_1$, then implementing 50 times $M_2$, and so on periodically. However, the objective of this paper is to obtain instead a smooth scheduling, which implements $M_1$, then $M_2$, again $M_1$, then $M_2$, and so on.

There are several reasons to desire this smoother scheduling. First, as noted above, this smooth scheduling would reduce the delay variability of the traffic, which is one of the main objections made to frame-based scheduling. Also, not only is this a good property, but guaranteeing a given amount of bandwidth to low-jitter traffic is a requirement of DiffServ for Expedited Forwarding traffic [17]. Therefore, if a router is to implement DiffServ, it has to find a method for implementing such guarantees, even though the jitter-sensitive traffic may only represent a small part of the total traffic. More generally, a typical implementation in core routers would first schedule traffic with tight jitter constraints using smooth frame-based scheduling, and then schedule the remainder bandwidth for jitter-insensitive traffic using an envelope-based scheduler [1].

Second, this smooth scheduling leads to less burstiness. For instance, in the example above, instead of receiving batches of 50 cells followed by 50 empty slots, output 2 will receive exactly one cell every two time-slots. Therefore, even though the throughput of the router itself is the same, a smoother scheduling may increase the throughput of a network of routers considered as a whole, because the routers down the line will receive a more regular traffic, thus allowing for better multiplexing effects. This is especially true for TCP traffic, for which a bursty scheduling may affect the round-trip time estimation and incur additional losses [18].

Third, a smoother scheduling results in better short-term fairness among flows, from a delay, jitter and bandwidth point-of-view.

Fourth, suppose that in the example above, the capacity of the second transmitter is only half of the line rate. The transmitter wouldn't then be able to use a BV scheduling, because it would be required to send at twice its capacity during half the time (send 50 packets in 50 slots, while it can only send 25 packets during this time). Therefore, smooth scheduling facilitates different per-port transmission (or reception) speeds. This is useful in heterogenous optical networks, with transmitters (and receivers) having different capacities. It is also useful in a wireless slotted FDMA system, in which the capacity may vary in time and depend on the position of the stations.

Finally, and perhaps most significantly, a smoother scheduling reduces the amount of buffering needed in the system. First, because a better multiplexing effect reduces the buffering needed in the following routers on the cell path, as explained above. But also because the instantaneous outgoing rates are better suited to the incoming rates. For instance, consider the second input in the example. If the incoming packets always come at the same rate, equivalent to 50% of the line rate, then input 2 will receive and buffer 25 cells during the 50 time-slots in which it is idle. With a smoother traffic, it wouldn't have to buffer more than one cell. While the effects of this second point are negligible in routers, they are crucial for optical, wireless and satellite systems, in which the buffers (or the optical delay-lines) incur significant power consumption, buffer (or delay-line) management complexity, and purchase cost.

Finally, in addition to finding a *smooth* scheduling algorithm, our objective is also to find a *practical* scheduling algorithm. First, because the BV algorithm needs $O(n^2)$ permutations [9], these are difficult to store on a single chip when $n$ is large (for $n = 512$, each permutation takes $n \cdot \log_2(n) = 4,608$ bits, hence we would need to store up to 150 Mbytes). Second, the BV algorithm complexity is in $O(n^{4.5})$ [9], and thus it is difficult to implement at high speeds.

The rest of the paper is organized as follows: We first outline the decomposition problem in Section II, provide a heuristic algorithm for solving this problem, and prove efficiency bounds for this algorithm. Then, in Section III, we explain how to use the decomposition algorithm in order to compute a schedule table that achieves low jitter. Section IV is devoted to experimental results. Section V develops a practical algorithm for bandwidth-guaranteed traffic, and finally section VI extends these results to long-tuning systems.

## II. Decomposition of the Guaranteed Rate Table

We consider an $n \times n$ input-output buffered packet switch with virtual output queueing. In each time slot, a central arbiter matches each output to at most one input. We will assume

that for the portion of the traffic that has bandwidth and jitter requirements, we have some knowledge of the rates required for each input/output port pair. Within the Internet architecture this knowledge is supplied either through a bandwidth broker or through MPLS signaling.

We are concerned only with the scheduling of the bandwidth guaranteed low jitter traffic that is specified by a *(Guaranteed) Rate Matrix*. Best effort traffic can be scheduled using any of the matching algorithms available in the literature.

Let $P$ represent the per-port capacity of the fabric. Let the $n \times n$ non-null matrix $R = [r_{ij}]$ represent the Rate Matrix where $r_{ij}$ represents the rate required from input $i$ to output $j$. Let $M$ represent the maximum row or column sum of the matrix $R$. Using standard techniques [9], it is easy to augment $R$ into a matrix $R' = [r'_{ij}]$ where all the row and column sums of $R'$ are $M$. For the rest of this paper we will assume that all the row sums and column sums of $R$ are $M$. When $M$ is not given, it is assumed that $M = 1$ - and in this case, $R$ is called a *Doubly Stochastic Matrix*. Finally, note that the rate matrix can be scheduled (without jitter constraints) if and only if the aggregate amount of traffic going to (respectively coming from) any port is less than the port capacity, i.e. $M \leq P$.

### A. Birkhoff Von-Neumann Decomposition

The basic idea of the Guaranteed Rate Tables is that once a rate matrix is provided, it is decomposed into schedule tables. Each schedule table is either a permutation or a partial permutation matrix (i.e. a $0-1$ matrix whose row sums and column sums are at most one without it being a permutation matrix). The standard approach to creating the schedule tables from the rate matrix relies on the following result due to Birkhoff and Von-Neumann [9].

*Theorem 1:* Any doubly stochastic matrix can be written as a convex combination of permutation matrices.

Therefore the BV decomposition of the rate matrix $R$ is to generate a set of permutation matrices (schedule tables) $Y^k$ for $k = 1, 2, \ldots, K$ such that

$$R = \sum_{k=1}^{K} \alpha_k Y^k.$$

In other words $r_{ij} = \sum_{i=1}^{K} \alpha_k Y_{ij}^k$. We refer to $\sum_{k=1}^{K} \alpha_k$ as the bandwidth requirement of the schedule tables generated by the BV decomposition. After scaling the results of Theorem 1 by a factor of $M$, we can see that the value of $\sum_{i=1}^{K} \alpha_k = M$, where $M$ is the row and column sum of the rate matrix $R$. Thus, the BV decomposition minimizes the bandwidth requirement. The number of matrices in the BV decomposition is $O(n^2)$ and the running time of the algorithm is $O(n^{4.5})$. More details and references on the decomposition algorithm can be found in Chang et al. ([9], [10]). The permutation (switching) matrices are scheduled across the switch using some Weighted Round Robin (WRR) scheme. Chang et al. [9] gives a scheduling algorithm and provides bounds on its performance. This method that is based on the BV decomposition is reasonable in the case that there are no jitter constraints. The BV decomposition, however, results in poor jitter performance especially when there is a large number of ports in the switch.

### B. The Low Jitter Decomposition

In the BV decomposition of the rate matrix $R$, a given entry $r_{ij}$ is striped across several permutation matrices. Therefore, independently of the type of algorithm used to schedule the permutation matrices, there is no control on when individual entries in the rate matrix will be scheduled. It is possible to derive bounds on the jitter [9], but it is not possible to ensure that the jitter is low. The bounds on the jitter for the traffic between input port $i$ and output port $j$ depends on the number of matrices in the decomposition that $r_{ij}$ is striped across and also on the number of matrices in the decomposition. Since both these factors increase with the number of ports in the switch, the jitter problem becomes severe when the number of ports is large. We formulate an alternate decomposition which we term Low Jitter (LJ) Decomposition. As in the case of the BV decomposition, we decompose the rate matrix into a combination of permutation matrices with the additional restriction that each non-zero entry in the rate matrix appears in only one of the permutation matrices. Let $X^k, k = 1, 2, \ldots K$ be the set of permutation matrices that form the LJD. Associated with each matrix $X^k$ in the decomposition is a rate $m_k$ which represents the bandwidth requirement for the switching matrix $X^k$.

$$\sum_k m_k x_{ij}^k \geq r_{ij} \quad \forall i, j \tag{1}$$

$$\sum_k x_{ij}^k = 1 \quad \forall i, j \tag{2}$$

$$\sum_i x_{ij}^k \leq 1 \quad \forall j, k \tag{3}$$

$$\sum_j x_{ij}^k \leq 1 \quad \forall i, k \tag{4}$$

$$x_{ij}^k \in \{0, 1\} \quad \forall i, j, k. \tag{5}$$

Constraints (3)(4) and (5) specify that $X^k$ is a partial permutation matrix. Constraint (1) specifies that the weighted sum of these permutation matrices should be greater than the rate matrix. Constraint (2) enforces that each entry in the rate matrix belongs to precisely one element in the decomposition. Note, that unlike the BV decomposition which uses only (full) permutation matrices, LJ decomposition splits the rate matrix into partial permutation matrices. Using the scheduling algorithm developed in the next section, we show that the above set of constraints are sufficient to guarantee low jitter. It is possible to construct specific examples where constraint (2) is not necessary to guarantee low jitter. We will comment on this in the last section of the paper. The bandwidth requirement for the schedule is $\sum_{i=1}^{K} m_k$. Therefore the objective of the LJ decomposition is to solve the following integer programming problem (ILJD):

$$D = \min \sum_{k=1}^{K} m_k$$

Subject to the constraints $(1)(2)(3)(4)(5)$ defined above. Since the BV decomposition solves the above problem without constraint (2), and the schedule length of BV is at most $M$, then $D \geq M$. Therefore there will be rate matrices that are schedulable by the BV decomposition but not by the LJ decomposition. This will especially be true if the amount of low jitter guaranteed traffic is a large fraction of the total switch traffic. In the applications that we consider the *low-jitter traffic is a relatively small fraction of the total switch traffic*. This is a valid assumption within the context of the EF class, that is expected to occupy less than 50% of the total bandwidth. In our experiments we varied the low-jitter traffic load from 10% to 60% and all the rate matrices were schedulable with LJD. As in the case of BV decomposition, the LJ decomposition of the rate matrix is not unique. Unlike the BV decomposition, the integer programming problem ILJD is NP-hard. The proof of the theorem below is a simple variant of a proof due to Rendl [19]. (In [19] the number of matrices in the decomposition is restricted to $n$, which is not the case here.)

*Theorem 2:* The problem ILJD is NP hard.

Several flavors of matrix decomposition problems have been studied extensively in the TDMA scheduling literature [20], [21], [22], [23]. As in our case, the two most important considerations in TDMA scheduling (especially for satellite scheduling) are to minimize the number of matrices in the decomposition and the total bandwidth needed to support the decomposition. However, the TDMA literature does not deal with low-jitter implementation of the rate matrix decomposition. Since the ILJD problem is NP-hard, our objective is to derive lower bounds on solutions to this problem, and then to use these lower bounds in order to get intuition and motivate a heuristic for solving the problem and implementing a low-jitter decomposition. We first prove a lemma on vector-ordering, and then use this lemma in order to find a lower bound on $D$. Note that proofs are placed in the appendix for ease of reading.

*Lemma 3:* Consider two vectors $(v_1, v_2, \ldots, v_p)$ and $(w_1, w_2, \ldots, w_p)$ where $v_1 \leq v_2 \leq \ldots \leq v_p$. Then a permutation $\sigma$ of the set $\{1, 2, \ldots, n\}$ that minimizes

$$\sum_{i=1}^{p} \max\{v_i, w_{\sigma(i)}\}$$

is a permutation where

$$w_{\sigma(1)} \leq w_{\sigma(2)} \leq \ldots \leq w_{\sigma(p)}.$$

*Theorem 4:* Let $R$ be a rate matrix. Sort each column of the matrix $R$ in descending order to get the matrix $R'$. Compute the maximum $g_i$ of each row $i$ of the matrix $R'$. Then $\sum_{i=1}^{n} g_i$ is a lower bound on $D$. Similarly sort each row of $R$ in descending order to get the matrix $R''$. Compute the maximum $h_j$ of each column $j$ of $R''$. Then $\sum_{j=1}^{n} h_j$ is a lower bound on $D$. Hence $D \geq \max(\sum_{i=1}^{n} g_i, \sum_{j=1}^{n} h_j)$.

The higher of the two lower bounds computed above is a lower bound on $D$ and is tighter than $M$. Let's illustrate the bounds of this theorem on the $4 \times 4$ doubly-stochastic rate matrix $R$ introduced in [9].

$$R = \begin{bmatrix} 0.38 & 0 & 0.22 & 0.40 \\ 0.11 & 0.24 & 0.60 & 0.05 \\ 0 & 0.53 & 0.14 & 0.33 \\ 0.51 & 0.23 & 0.04 & 0.22 \end{bmatrix}$$

First we sort each column of $R$ in descending order to obtain matrix $R'$ as per Theorem 4.

$$R' = \begin{bmatrix} 0.51 & 0.53 & 0.60 & 0.40 \\ 0.38 & 0.24 & 0.22 & 0.33 \\ 0.11 & 0.23 & 0.14 & 0.22 \\ 0 & 0 & 0.04 & 0.05 \end{bmatrix}$$

We find the sum of the maximum of each row to get $\sum_{i=1}^{n} g_i = 0.60 + 0.38 + 0.23 + 0.05 = 1.26$. Now we sort each row to obtain matrix $R''$.

$$R'' = \begin{bmatrix} 0.40 & 0.38 & 0.22 & 0 \\ 0.60 & 0.24 & 0.11 & 0.05 \\ 0.53 & 0.33 & 0.14 & 0 \\ 0.51 & 0.23 & 0.22 & 0.04 \end{bmatrix}$$

We find the sum of the maximum of each column to get $\sum_{j=1}^{n} h_j = 0.60 + 0.38 + 0.22 + 0.05 = 1.25$. Therefore the lower bound for this problem is 1.26. As a comparison, we solved the integer programming problem (ILJD) using the CPLEX optimization program. The ILJD optimization yields:

$$R = 0.60 \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} + 0.05 \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$+0.33 \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} + 0.38 \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Therefore the optimal LJ decomposition has a value $D = 0.60 + 0.38 + 0.33 + 0.05 = 1.36$. We can check $1.26 \leq D$.

Consider now a particular matrix $X^k$ in the LJ decomposition. The rate $m_k$ associated with this matrix is the largest entry in $R$ that is covered by this matrix. Therefore the total amount of bandwidth can be optimized by covering entries of roughly equal size with the same matrix. Of course, the entries should not share the same row or column in order to be in a decomposition. This is the basis of the Greedy Low-Jitter Decomposition (GLJD) which is a heuristic to solve the integer programming problem ILJD.

### C. The Greedy Low-Jitter Decomposition

The GLJD algorithm is defined as follows. First, the elements in $R$ are sorted in descending order and put in an ordered list $L$. Note that each element in $L$ corresponds to some element $R_{ij}$ in the matrix $R$. Two elements in $L$ are defined to be *non-conflicting* if these two elements do not belong to the same row or column in $R$. Then, we fill up each of the decomposition sub-matrices by traversing the list from the top and picking non-conflicting elements greedily. Once an element is picked to be inserted into the decomposition sub-matrix, the element is deleted from $L$. Once the end of

GREEDY_LOW_JITTER_DECOMPOSITION

1: Sort the entries in $R$ is descending order to create list $L$.
2: $\forall m \in L$, if $L(m) = r_{ij}$ then set $\rho(m) = i$ and $\kappa(m) = j$.
3: $k = 1$.
4: While $L \neq \emptyset$
    Set $m_k = 0$ and $l = 1$
    Set $C[j] = 0 \ \ \forall j = 1, 2, \ldots, n$
    While (Not end of list $L$)
        If ($C[\rho[l]] = 0$ and $C[\kappa(l)] = 0$)
            $X^k_{\rho(l)\kappa(l)} = 1$
            $C[\rho(l)] = C[\kappa(l)] = 1$
            If $(L(l) > m_k) \ \ m_k = L(l)$
            Eliminate entry $l$ from list $L$
        End If
        $l \leftarrow l + 1$
    End While
    $k \leftarrow k + 1$
End While
5: Output the matrices $X^k$ and the corresponding weights $m_k$.

Fig. 1. Description of the Greedy Low Jitter Decomposition Algorithm.

the list is reached, then a new decomposition sub-matrix is initiated. This process is repeated until all the elements of $L$ are inserted into some sub-matrix. It is easy to see that the worst case running time of the algorithm is $O(n^3)$. The description of the GLJD algorithm is shown in Figure 1. For the numerical example above, the GLJD algorithm generates the following decomposition.

$$R \leq \quad 0.60 \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad +0.38 \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$+0.23 \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \quad +0.22 \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$+0.05 \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

GLJD yields a schedule time of $0.60 + 0.38 + 0.23 + 0.22 + 0.05 = 1.48$ (as expected, it is greater than the optimal ILJD schedule time of $D = 1.36$ and the BV schedule time of 1). Therefore if the low-jitter traffic load is less than $\frac{1}{1.48} = 0.68$ then the LJ heuristic can guarantee a full capacity to the traffic. This is useful because, in practical situations, we expect the low-jitter traffic to not be a sizable portion of the traffic across the switch. Also, it can be noted that the bandwidth is not necessarily wasted just because the LJ heuristic has a higher value than the BV decomposition. Indeed, any bandwidth not used by the low-jitter traffic class can then carry best-effort traffic.

### D. Guarantees on the GLJD Algorithm

The primary objective of the GLJD algorithm is to enable traffic scheduling with low-jitter guarantees. However, the following theorems prove that it also provides additional guarantees, with upper bounds on the number of partial permutations needed, the bandwidth required for the scheduling and the competitive ratio with respect to the optimal algorithm.

For the sake of simplicity, in this section, we will always assume that $M = 1$, and thus a rate matrix $R$ has to be doubly stochastic with line and column sum 1. The results will then be easy to generalize to any given $M$. The following theorem provides an upper bound on the number of partial permutations used by GLJD. It derives from a more general result on greedy on-line edge coloring [24], [13].

*Theorem 5:* Let $K$ be the number of partial permutation matrices needed in the GLJD algorithm. Then

$$K \leq 2n - 1.$$

We have just found that the GLJD algorithm provides a guarantee on the number of matrices that it will use. We are now interested in the worst-case bandwidth that it will require. More generally, define the length of the schedule provided by an algorithm $ALG$ for a rate matrix $R$ as $T_{ALG}(R) = \sum_{k=1}^{K} m_k$. Then, define the worst-case bandwidth requirement of algorithm $ALG$ as: $BW(ALG) = \max_R(T_{ALG}(R))$, where the maximum is taken over the set of rate matrices $R$. Then, the next theorem guarantees that $BW(GLJD) = O(\log n)$.

*Theorem 6:* Let $n \geq 2$ and let $H$ represent the harmonic series ($H_n = \sum_{k=1}^{n} 1/k$). Then

$$BW(GLJD) \leq 2H_n - 1 \leq 2\ln(n) + 1.$$

How good is the upper bound provided by this theorem? Theorem 6 states that GLJD guarantees at least $\frac{1}{2H_n - 1}$ of the bandwidth to low-jitter traffic. Therefore, GLJD guarantees at least $11.8\%$ of the bandwidth to low-jitter traffic when $n = 64$, and $7.1\%$ when $n = 1024$.

However, during simulations, the bandwidth generally guaranteed to low-jitter traffic was around $60\%$ with $n = 64$, as described later. One may thus wonder if $BW(GLJD)$ really grows as $\Theta(\log_n)$ and reaches this worst-case bound. The objective of the next theorem is to show that $BW(GLJD)$ indeed grows as $\Theta(\log_n)$. In addition, this theorem will prove that for *any* algorithm $ALG$ which satisfies the conditions of the integer programming problem, $BW(ALG)$ grows at least as fast as $\Theta(\log_n)$. In particular, if ILJD is an optimal such algorithm, i.e. $D(R) = T_{ILJD}(R)$ for any rate matrix $R$, then the bandwidth-guarantee competitive ratio $\frac{BW(GLJD)}{BW(ILJD)}$ is bounded.

*Theorem 7:* For any $n \geq 3$, there exists at least one rate matrix $R_n$ with the following lower bound on the length of its optimal LJ decomposition:

$$D(R_n) \geq \frac{\ln(n+2)}{2\ln 2} - \frac{1}{2}.$$

*Corollary 8:* $BW(GLJD) = \Theta(\log n)$ and $BW(ILJD) = \Theta(\log n)$, hence $\frac{BW(GLJD)}{BW(ILJD)} = \Theta(1)$.

Is this bound meaningful? The fact that the bandwidth-guarantee competitive ratio of GLJD has a constant upper bound is indeed meaningful. This result implies that the bandwidth guarantee of any other algorithm satisfying the low-jitter decomposition would not better the bandwidth guarantee of GLJD by much as the scheduling complexity increases with $n$, contrarily to what one would have intuitively thought when considering the $\Theta(\log n)$ behavior of $BW(GLJD)$. However, note that the bound on the competitive ratio is not tight, and that the following theorems will prove in particular that this ratio is at most 2.

We are now interested in the performance of the GLJD algorithm with any given rate matrix $R$, rather than with its worst-case rate matrix. The following theorems will slightly improve the lower bound on $D(R) = T_{ILJD}(R)$ provided by Theorem 4, and then prove that for any rate matrix $R$, $\frac{T_{GLJD}(R)}{T_{ILJD}(R)} \leq 2 - \frac{1}{n}$. This means that for *any* matrix, the bandwidth required by GLJD is less than twice the bandwidth required by the optimal low-jitter decomposition. A consequence of this result is that the bandwidth-guarantee competitive ratio is also less than $2 - \frac{1}{n}$.

*Theorem 9:* For any given schedule, assume without loss of generality that $m_1 \geq m_2 \geq ... \geq m_K$. Then for any $k \in \{1, \ldots, n\}$,

$$m_k \geq \max(g_k, h_k), \text{ and} \tag{6}$$

$$D(R) = T_{ILJD}(R) \geq \sum_{k=1}^{n} \max(g_k, h_k). \tag{7}$$

*Theorem 10:* For any rate matrix $R$,

$$\frac{T_{GLJD}(R)}{T_{ILJD}(R)} \leq 2 - \frac{1}{n}.$$

*Corollary 11:* The bandwidth-guarantee competitive ratio of the GLJD algorithm is upper-bounded by $2 - \frac{1}{n}$.

Itis interesting to note that it is possible to derive a doubly-stochastic bipartite version of the worst-case matching in [24] in order to prove that this bound of 2 is actually tight for large $n$.

We now outline the scheduling algorithm that will be used to schedule the matrices generated by the LJ decomposition.

## III. Scheduling the LJ Decomposition

Since $r_{ij}$ is the desired rate from input port $i$ to output port $j$, we ideally want the time slots when $i$ is matched to $j$ to be spaced $\frac{1}{r_{ij}}$ apart, i.e., the points $0, \frac{1}{r_{ij}}, \frac{2}{r_{ij}}, \frac{3}{r_{ij}}, \ldots$. Given that there are multiple connections that share the same bandwidth, this is not possible in general. We settle for a slightly degraded jitter performance. We call a connection *Low Jitter* if we have exactly one match for this connection in each of the time intervals $\left[\frac{m}{r_{ij}}, \frac{m+1}{r_{ij}}\right], m = 0, 1, 2, \ldots$. At the end of the LJ decomposition, we have $K$ (perhaps partial) permutation matrices. Let $m_k$ be the bandwidth for the switching matrix $X^k$ and let $D = \sum_{k=1}^{K} m_k$. We assume that $D < P$, where $P$ is the switch speed. We define $\phi_k = \frac{m_k}{P}$, where $\phi_k$ represents the fraction of timeslots that should use the schedule table (switching matrix) $X^k$. We assume that $\sum_{k=1}^{K} \phi_k < 1$. We want to schedule matrix $k$ at rate $\phi_k$. Since each port pair belongs in exactly one matrix in the decomposition, it is possible to control the jitter for each port pair individually. Let $T_k^j$ represent the time slot in which the schedule table $k$ is scheduled for the $j^{\text{th}}$ time. For low jitter, we want

$$\frac{j}{\phi_k} \leq T_k^j \leq \frac{j+1}{\phi_k}.$$

We compute as follows the start time $S_k^j$ and the finish time $F_j^k$ for the $j^{\text{th}}$ schedule of table $k$:

$$S_k^j = \frac{j}{\phi_k} \qquad F_k^j = \frac{j+1}{\phi_k}.$$

In the implementation of the scheduling algorithm we define iteratively $A_k$ and $B_k$, respectively the current start time and the current finish time for schedule table $k$, as shown in Figure 2.

LOW_JITTER_SCHEDULER

1: Lat $A_k$ denote the current start time and $B_k$ the current finish time for class k.
2: Set $A_k = 0$ and $B_k = \frac{1}{\phi_k}$.
3: Let $t$ represent the current time slot.
4: Let $l = \text{Arg}\min_{k:A_k \leq t} B_k$.
5: If $l \neq \emptyset$ schedule $X^l$ in timeslot $t$.
   - Set $A_l \leftarrow B_l$ and $B_l = B_l + \frac{1}{\phi_l}$.
   - Set $t \leftarrow t + 1$ and go to Step 3.
6: If $l = \emptyset$ then set $t \leftarrow t + 1$ and go to Step 3.

Fig. 2. Description of Low Jitter Scheduling Algorithm.

At the beginning of time slot $t$, schedule table $k$ is defined to be eligible if $S_k \leq t$ and ineligible otherwise. The scheduling algorithm works as follows: At the beginning of each time slot, among all the eligible schedule tables, the table with the smallest finishing time is picked for scheduling. Let $l$ represent this table. The start time and the finish time for schedule table $l$ is updated. This process is then repeated. If no class is eligible at the beginning of a given time slot, then the inputs are matched to the outputs by some algorithm that optimizes the performance of best effort traffic. Best effort traffic also uses the slots where guaranteed jitter traffic is not available to take its allotted slot or in the case where the schedule table is a partial permutation table. The following theorem is a special case of the general result in [25]. The proof of the result in our case is straightforward and is proved directly.

*Theorem 12:* If $\sum_{k=1}^{K} \phi_k < 1$, then for all classes $k$, the LJS algorithm results in

$$\frac{j}{\phi_k} \leq T_k^j \leq \frac{j+1}{\phi_k} + 1.$$

Therefore by the above theorem, all connections are (almost) low jitter. Note that some connections may miss their allotted range by one time slot. Under the moderate loading

conditions that we consider, this seems to happen very rarely. In the experimental section, we compare the LJ decomposition with the scheduling algorithm described above with the BV decomposition and the scheduling algorithm outlined in Chang et al. [9]. The main difference between the scheduling algorithm described above and the one in [9] is that the scheduling algorithm in [9] does not use starting times.We ran the BV decomposition algorithm with our scheduling algorithm and the results for BV are not qualitatively different from the ones shown in the next section.

## IV. Experimental Results

The experimental results are shown to illustrate the advantages in jitter performance of the LJD. In the simulations, we consider $64 \times 64$ and $128 \times 128$ switches in the experiments, and assume that the switches process 100 slots per second. It is also assumed that the guaranteed rate for each input-output pair is independently and uniformly distributed between 0 and some upper bound $u$. The value of $u$ determines the mean guaranteed rate traffic load in the switch. We adjust the rate to ensure that all row and column sums are the same. If the row and column sums are different for different rows and columns, then the performance of the algorithms are qualitatively similar to the results for the balanced case. We measure the following quantities:

*Jitter Performance:* For a given input-output pair $(i, j)$, the LJ decomposition will result in one match in each interval $\left[\frac{m}{r_{ij}}, \frac{m+1}{r_{ij}}\right]$, for $m = 0, 1, 2, \ldots$ In the case of the BV decomposition some intervals will get multiple matches and others will get none. In Figure 3 we plot the interval number versus the number of matches produced by the different decompositions. Results are shown for a particular input-output port combination, but other input-output ports had similar results. The results show a very poor jitter performance for the BV decomposition. There are no matches for almost 20 intervals between 40 and 60, and almost 12 matches in the interval 39. On the contrary, GLJD results in one match per interval as predicted by Theorem 12. For BV, performance was even worse with the $128 \times 128$ switch, not shown here.

*Speedup Ratio:* This represents the ratio of the bandwidth requirement of the GLJD heuristic to the bandwidth requirement of BV. Recall that the bandwidth requirement of BV decomposition is the row sum $M$ of the rate matrix. We show the result for 10 experiments in Figure 4. In our experiments we varied the low-jitter traffic load from 10% to 60%, and in all these cases the schedule table obtained by the GLJD was feasible. In other words, the throughput requirement was not higher than the switch capacity. As stated earlier, in practice, we expect the low-jitter traffic load to be less than 50% of the switch traffic.

*Optimality Gap:* We compute the ratio of $T_{GLJD}$ to the lower bound $D = T_{ILJD}$ generated by applying Lemma 3. This ratio tracks the performance of GLJD to the optimal solution. The result is shown in Figure 5. Note that in all the cases, the heuristic is within 10% of the optimal solution.

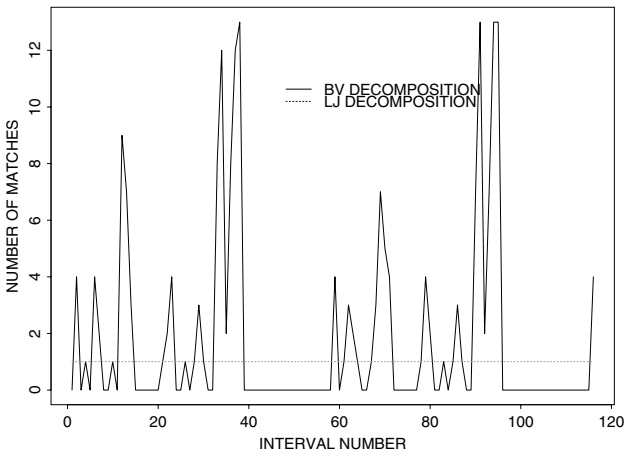


Fig. 3. Number of Matches per Interval for a $64 \times 64$ Switch

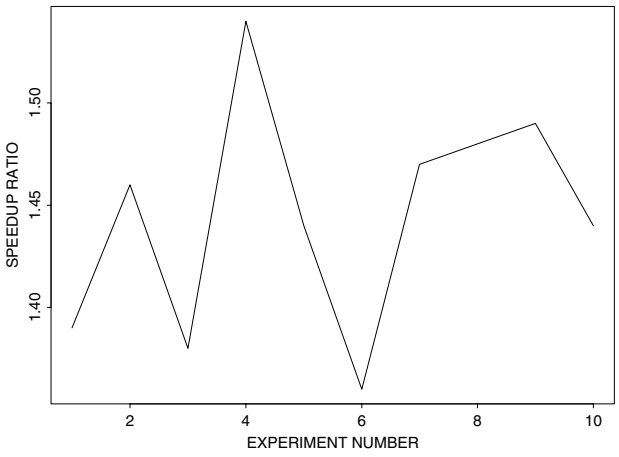


Fig. 4. Speedup Ratio for a $64 \times 64$ switch

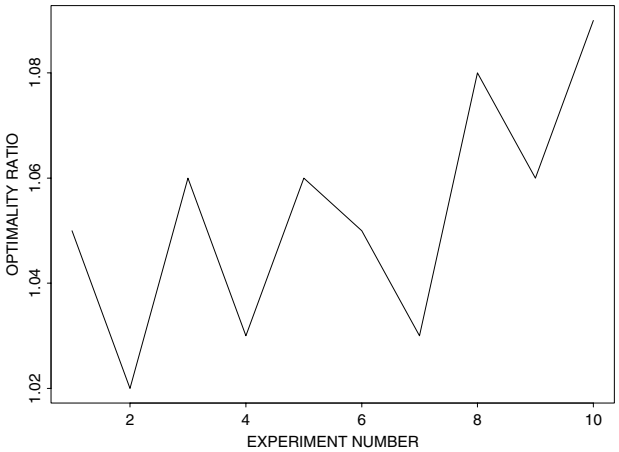


Fig. 5. Optimality Ratio for a $64 \times 64$ switch

TABLE I

NUMBER OF MATRICES IN THE DECOMPOSITION

| Trial | BV $64 \times 64$ Switch | LJ $64 \times 64$ Switch | BV $128 \times 128$ Switch | LJ $128 \times 128$ Switch |
|---|---|---|---|---|
| 1 | 854 | 68 | 1591 | 131 |
| 2 | 797 | 69 | 1575 | 134 |
| 3 | 878 | 69 | 1534 | 129 |
| 4 | 812 | 70 | 1612 | 137 |
| 5 | 878 | 67 | 1567 | 129 |

TABLE II

RUNNING TIMES OF THE ALGORITHM

| Trial | LJ $64 \times 64$ Switch (Secs) | BV $64 \times 64$ Switch (Secs) | LJ $128 \times 128$ Switch (Secs) | BV $128 \times 128$ Switch (Secs) |
|---|---|---|---|---|
| 1 | 0.05 | 5.42 | 0.32 | 59.7 |
| 2 | 0.04 | 5.37 | 0.35 | 58.7 |
| 3 | 0.04 | 5.22 | 0.41 | 56.4 |
| 4 | 0.05 | 5.68 | 0.37 | 57.2 |
| 5 | 0.06 | 5.44 | 0.35 | 51.7 |

*Number of Matrices in the Decomposition:* We compare the number of matrices in BV and GLJD. Lower number of matrices results in easier scheduling algorithms. The results are shown for a $64 \times 64$ and $128 \times 128$ switch in Table I . With $n = 64$, BV typically generates about 850 matrices versus about 70 matrices for GLJD. Similarly, BV results in about 1600 matrices and GLJD results in about 135 matrices for $n = 128$. Therefore the size of the schedule table is typically an order of magnitude less for GLJD with respect to BV.

*Total Computation Time:* We compare the total computation time to run both the BV and the LJ decompositions in Table II . The runs were done on a 1 Ghz Pentium processor. All run times are in seconds of system time. For the BV decomposition we run a fast maximum flow algorithm at each step in order to determine the next switching matrix in the decomposition. Note that although both these computation times could be minimized in hardware implementations, we believe that the LJ decomposition (using GLJD) will still be significantly faster.

## V. A PRACTICAL ALGORITHM FOR BANDWIDTH-GUARANTEED TRAFFIC

Assume that a router designer is to build a frame-based scheduler for bandwidth-guaranteed traffic with little or no jitter constraint. What algorithm should this designer use?

In a mathematical formulation, the designer needs to consider a rate matrix $R$ with real (floating point) coefficients, and find an algorithm that would guarantee a schedule to $R$ with small speed-up ($O(1)$), a number of permutations small enough to be stored on a single chip (say $O(n)$), and a complexity small enough to be implemented at core-router speeds.

For solving this problem, we have considered many algorithms proposed in the literature. Most of them are based on a common divisor in $R$ and thus require too many permutations, because the common divisor in router implementations might be as small as the floating point precision. Thus, this excludes algorithms related to call routing in Clos networks [26], [27], to openshop scheduling [28], to heuristic row bottleneck minimization in optical WDMA star networks [29] and in TDM switching [30], and to bipartite graph matching [13]. Alternative algorithms which do not require a common divisor are often too impractical because they require optimal edge coloring in bipartite graphs, such as BV [9] and a traffic rate approximation scheme proposed in [31].

Our approach is inspired by this last scheme. However, contrarily to this scheme, we will use maximal matchings [13] instead of optimal edge coloring. Maximal matchings are faster, but provide less guarantees.

We thus propose the following *iterative maximal frame-scheduling*. First, define an integer $d$, which will be a common divisor in the coarse matrix, such that $d = \Theta(n)$ - for instance, $d = n$, $d = 4n$, $d = 2^{\lceil \log_2(n) \rceil}$, and so on.

Then, given the rates $r_{ij}$, compute $u_{ij} = \lfloor d \cdot r_{ij} \rfloor$ and $\epsilon_{ij} = r_{ij} - \frac{u_{ij}}{d}$. Note that when $d$ is a power of 2 and numbers are represented in a binary form, this computation simply truncates the first digits of $r_{ij}$. $U_1 = U = [u_{ij}]$, the *coarse approximation matrix*, is a nonnegative-integer matrix with row and column sum of at most $d$. $\epsilon = [\epsilon_{ij}] = R - \frac{U}{d}$, the *remainder matrix*, is such that for all $\{i, j\}$, $0 \leq \epsilon_{ij} \leq \frac{1}{d}$.

Then, apply a maximal matching algorithm to $U_1$, yielding (partial) permutation $\sigma_1$, and compute $U_2 = U_1 - \sigma_1$. Apply again a maximal matching to $U_2$, yielding $\sigma_2$, and compute $U_3 = U_2 - \sigma_2$. Continue likewise $2d-1$ times, until obtaining $2d-1$ (partial) permutations $\{\sigma_1, ..., \sigma_{2d-1}\}$.

Finally, for $1 \leq k \leq n$, call $\pi_k$ the permutation that connects any input $i$ to output $i+k$ mod $n$, and consider the set $S = \{\sigma_1, ..., \sigma_{2d-1}, \pi_1, ..., \pi_n\}$. Then the following theorem proves that a schedule table using each of the permutations in $S$ exactly once in a cycle of $d$ time-slots satisfies the rate matrix requirements.

*Theorem 13:* At the end of the algorithm defined above, $\frac{\sum_{k=1}^{2d-1} \sigma_k}{d} \geq \frac{U}{d}$, $\frac{\sum_{k=1}^{n} \pi_k}{d} \geq \epsilon$, and therefore, $\frac{\sum_{k=1}^{2d-1} \sigma_k + \sum_{k=1}^{n} \pi_k}{d} \geq R$.

Assume that $d = \Theta(n)$. The resulting decomposition satisfies our objectives. First, it uses at most $2d + n - 1 = O(n)$ permutations in the schedule table, thus these permutations will typically hold on a chip. Second, it needs a finite speed-up of $2 + \frac{n-1}{d} = O(1)$ in order to guarantee 100% throughput. Finally, it takes a computation time of $O(n^3)$ with a centralized scheduler, and $O(n)$ with a distributed scheduling, as shown below. It can also be noted that the jitter guarantee is in $O(n)$ (the length of the schedule), hence this is an alternative for traffic not needing the extremely low jitter implemented with GLJD.

Also, all the maximal algorithms used in current routers can be extended to frame scheduling, and in particular as well those based on pipelining such as WFA [2] as those based on a request-grant-accept arbitration type such as iSlip [4]. This is interesting for legacy issues. The pipelined version of WFA

seems especially suited for such an iterative algorithm, since crosspoint $(i, j)$ can start working on iteration $k + 1$ of the decomposition algorithm at the same time as crosspoints $(i, j+1)$ and $(i + 1, j)$ start working on iteration $k$ [2]. Therefore, for any given crosspoint, there are $2d - 1$ iterations, with at most one computation and two transmissions of information per iteration. Hence, the distributed algorithm runs in $O(d) = O(n)$ time.

Finally, from a practical point of view, the main advantage of such a scheme is that it can easily support incremental rate updates without computing a whole new schedule again. For instance, assume that $r_{ij}^1$ is updated into $r_{ij}^2$. Compute the change in the integer coarse matrix $\delta = u_{ij}^2 - u_{ij}^1$. If $\delta = 0$, do nothing. If $\delta < 0$, remove $\delta$ elements in position $(i, j)$ from the set $\{\sigma_k\}$. If $\delta > 0$, add $\delta$ elements in position $(i, j)$ to the set $\{\sigma_k\}$ (possible because the set contains $2d - 1$ elements [13]). Hence, with a correct data structure, an update takes $O(n)$ time. Interestingly enough, this feature is related to the ease for establishing and removing calls in a non-blocking Clos network with $2d - 1$ middle-stages.

## VI. A Last Word on Bursty Decomposition Algorithms

A surprising side-effect of the greedy smooth scheduling provided in this paper is that it can actually be helpful when looking for the *least* smooth scheduling possible. For instance, consider an input-queued router using an optical switch fabric. Note that this fabric could be passive (for instance, based on an arrayed waveguide grating with tunable lasers [32]) as well as active (for instance, based on micro-mirrors [33]). Similarly, consider a star-based WDM broadcast-and-select optical system [14]. In those two cases, the tuning time from one channel to another takes a major share of the frame schedule time [31]. Thus, the main issue is not anymore the bandwidth available, but rather the number of tuning times in a given schedule: the designer will prefer a schedule with very few permutations, and thus few tunings, even if it implies an increased burstiness.

The primary objective is therefore to minimize as much as possible the number of permutations in a schedule, in order to minimize the number of tuning times, and a secondary objective is to minimize the bandwidth taken by those permutations. Towles and Dally [31] propose a heuristic algorithm for this problem called MIN. MIN minimizes the number of permutations used, and therefore uses at most $n$ permutations. However, MIN needs a speed-up of at least $\Theta(\log n)$ in order to schedule any rate matrix. Also, MIN needs to perform several times edge coloring in a bipartite graph, and is therefore difficult to implement in a core router. GLJD could thus be a practical algorithm for approximating MIN. First, GLJD needs at most $2n - 1$ permutations (Theorem 5), and in simulations the number of permutations is shown to be very close to $n$. Second, GLJD also needs a speed-up of at most $\Theta(\log n)$, with simulations results close to a speed-up of $1.5$. And finally, GLJD was shown to have a low implementation complexity.

## VII. Conclusion

In this paper we considered the problem of scheduling guaranteed-bandwidth low-jitter traffic in input-queued switches. We proposed to combine the GLJD algorithm and a Low Jitter Scheduler, and found experimentally that they are both both practical and efficient. We provided several performance bounds on these algorithms, with respect to the number of permutations as well as the schedule time. We proposed a practical algorithm for scheduling the remaining jitter-insensitive bandwidth-guaranteed traffic, and found that it supports incremental updates. Finally, we showed how our findings could even be extended to non-smooth systems with large tuning time.

## References


[1] K. Kar, T. V. Lakshman, D. Stiliadis, L. Tassiulas, "Reduced complexity input buffered switches," *Proceedings of Hot Interconnects VIII*, Palo Alto, Aug. 2000.

[2] Y. Tamir and H.C. Chi, "Symmetric crossbar arbiters for VLSI communication switches," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 1, pp. 13-27, 1993.

[3] T.E. Anderson, S.S. Owicki, J.B. Saxe, and C.P. Thacker, "High speed switch scheduling for local area networks," *ACM Transactions on Computer Systems*, vol. 11, no. 4, pp. 319–52, Nov. 1993.

[4] N. McKeown, M. Izzard, A. Mekkittikul, B. Ellersick, and M. Horowitz, "The Tiny Tera: A packet switch core," in *Proceedings of Hot Interconnects V*, August 1996.

[5] D. Shah and M. Kopikare, "Delay bounds for approximate maximum weight matching algorithms for input queued switches," *Proceedings of IEEE Infocom '02*, New York, NY, June 2002.

[6] L. Tassiulas, "Linear complexity algorithms for maximum throughput in radio networks and input queued switches," *Proceedings of IEEE Infocom '98*, vol. 2, pp. 533-539, 1998.

[7] P. Giaccone, B. Prabhakar, and D. Shah, "Towards simple, high-performance schedulers for high-aggregate bandwidth switches," *Proceedings of IEEE Infocom '02*, New York, NY, June 2002.

[8] A. Hung, G. Kesidis, and N. McKeown, "ATM input-buffered switches with guaranteed rate property," *IEEE ISCC, Athens*, 1998.

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

[10] C.S. Chang, J.W. Chen, and H.Y. Huang, "Birkhoff-Von Neumann input-buffered crossbar switches," *Proceedings of IEEE INFOCOM '00*, Tel Aviv, Israel, pp. 1614-1623, 2000.

[11] E. Altman, Z. Liu and R. Righter, "Scheduling of an input-queued switch to achieve maximal throughput," *Probability in the Engineering and Informational Sciences*, vol. 14, pp. 327334, 2000.

[12] R.S. Tucker, W.D. Zhong, "Photonic packet switching: an overview," *IEICE Transactions on Communications*, vol. E82-B, no. 2, pp. 254-264, Feb. 1999.

[13] T. Weller and B. Hajek, "Scheduling nonuniform traffic in a packet-switching system with small propagation delay", *IEEE Transactions on Networking*, vol. 5, pp. 813-823, Dec. 1997.

[14] V. Sivaraman and G.N. Rouskas, "A reservation protocol for broadcast WDM networks and stability analysis," *Computer Networks*, vol. 32, no. 2, pp. 211-227, Feb. 2000.

[15] J. Lin and N. Ansari, "Enhanced Birkhoff-von Neumann Decomposition Algorithm for Input Queued Switches," *IEE Proceedings Communications*, vol. 148, no. 6, pp. 339-342, 2001.

[16] M. Ajmone Marsan, P. Giaccone, E. Leonardi, and F. Neri, "Local scheduling policies in networks of packet switches,"' *unpublished*, 2002.

[17] A. Charny et. al., "An Expedited Forwarding PHB," draft-ietf-diffserv-rfc2598bis-02.txt, Internet Engineering Task Force.

[18] B. Sikdar, K.S. Vastola and S. Kalyanaraman, "On reducing the degree of self-similarity in network traffic," *unpublished*, 2002.

[19] F. Rendl, "On the complexity of decomposing matrices arising in satellite communication," *Operations Research Letters*, vol. 4, May 1985.

[20] G. Bongiovanni, D. Coppersmith, and C.K. Wong, "An optimal time slot assignment for an SS/TDMA system with variable number of transponders," *IEEE Transactions on Communications*, vol. 29, May 1981.
[21] E. Balas and P.R. Landweer, "Traffic assignment in communication satellites," *Operations Research Letters*, vol. 2, November 1983.
[22] I. Gopal, and C.K. Wong, "Minimizing the number of switchings in an SS/TDMA system", *IEEE Transactions on Communications*, vol. 33, June 1985.
[23] C.A. Pomalaza-Raez, "A note on efficient SS/TDMA assignment algorithms," *IEEE Transactions on Communications*, vol. 36, September 1988.
[24] A. Bar-Noy, R. Motwani, and J. Naor, "The greedy algorithm is optimal for on-line edge coloring," *Information Processing Letters*, vol. 44, no. 5, pp. 251-253, 1992.
[25] N.R. Figueria and J. Pasquale, "An upper bound on delay for the virtual clock service discipline," *IEEE Transactions on Networking*, vol. 3, August 1995.
[26] I. Gragopoulos and F.-N. Pavlidou, "A new evaluation criterion for Clos- and Benes-type rearrangeable switching networks", *IEEE Transactions on Communications*, vol. 45, no. 1, pp. 119-126, Jan. 1997.
[27] H. Lee, F. Hwang and J. Carpinelli, "A new matrix decomposition algorithm for rearrangeable Clos interconnection networks", *IEEE Transactions on Communications*, vol. 44, no. 11, pp. 1572-1578, Nov. 1996.
[28] A. Ganz and Y. Gao, "Efficient algorithms for SS/TDMA scheduling", *IEEE Transactions on Communications*, vol. 40, no. 6, pp. 1367-1374, Aug. 1992.
[29] K.L. Chen, "A conflict-free protocol for optical WDMA networks", *Proceedings of IEEE Globecom '91*, pp.1276-1281, 1991.
[30] K.L. Yeung, "Efficient time slot assignment algorithms for TDM hierarchical and nonhierarchical switching systems", *IEEE Transactions on Communications*, vol. 49, no. 2, pp. 351-359, Feb. 2001.
[31] B. Towles and W.J. Dally, "Guaranteed scheduling for switches with configuration overhead," *Proceedings of IEEE Infocom '02*, New York, NY, June 2002.
[32] J. Gripp et. al., "Demonstration of a 1.2 Tb/s optical packet switch fabric based on 40 Gb/s burst-mode clock-data-recovery, fast tunable lasers, and a high-performance NxN AWG," *ECOC '01*, vol. 6, 2001.
[33] Special issue on optical MEMS, *IEEE Journal on Selected Topics in Quantum Electronics*, vol. 8, no. 1 , Jan.-Feb. 2002.

## APPENDIX

**Proof for Lemma 3:** Let $\sigma$ be an optimal permutation. For some $k < l$ let $w_{\sigma(k)} \geq w_{\sigma(l)}$. Let $v_l = \max\{w_{\sigma(k)}, v_l\}$. By interchanging $w_{\sigma(k)}$ and $w_{\sigma(l)}$ note that the objective function value is non increasing. The same result can be shown if $w_{\sigma(k)} = \max\{w_{\sigma(k)}, v_l\}$.

**Proof for Theorem 4:** We show the result for the case where the rows are sorted. The proof for the column sorting is identical. Assume that the decomposition has $n$ matrices. It is easy to show that the lower bound is valid if there are more than $n$ matrices in the decomposition. Consider the first row of the matrix $R$. Assume without loss of generality that the columns of $R$ are permuted so that the first row is sorted in descending order. Each entry in the first row has to belong to a different matrix in the decomposition. Each entry in the second row has to belong to a different matrix in the decomposition. If we ignore the constraint that each column in each decomposition matrix has to have exactly one entry, then the minimum amount of bandwidth that we need for the first two rows, by Lemma 3 is given when the second row is in sorted order. It is a lower bound because we ignore one of the constraints. This argument can be repeated in order to show the result.

**Proof for Theorem 6:** Let $R$ be a rate matrix, and assume that $r_{ij}$ is an element of $R$ that still remains to be scheduled after $k$ iterations of the GLJD algorithm. Consider the set of all elements in $L$ that belong to the same row or column as $r_{ij}$ and have at least the same weight. Then, during the first $k$ iterations, the algorithm picked at least $k$ elements from this set (by construction of this greedy maximal algorithm, see also [13]).Thus, according to the pigeonhole principle, the algorithm picked at least $\lceil \frac{k}{2} \rceil$ of these elements either in row $i$ or in column $j$ - without loss of generality, let's assume that it was in row $i$. Therefore, on row $i$, there were initially at least $1 + \lceil \frac{k}{2} \rceil$ elements with a rate of at least $r_{ij}$ (counting $r_{ij}$ itself). Hence, since the row sum is 1, $r_{ij} \leq \frac{1}{1+\lceil \frac{k}{2} \rceil}$. We thus know that after $k$ iterations, all remaining elements have a rate of at most $\frac{1}{1+\lceil \frac{k}{2} \rceil}$. This implies that the weight of the partial permutation picked at the next iteration will have the same upper bound ($m_{k+1} \leq \frac{1}{1+\lceil \frac{k}{2} \rceil}$). Using $m_1 = \max_{i,j}(r_{ij}) \leq 1$ and $K \leq 2n-1$ (Theorem 5), we get

$$\begin{aligned} \sum_{k=1}^{K} m_k &\leq 1 + \sum_{k=1}^{K-1} \frac{1}{1+\lceil \frac{k}{2} \rceil} \\ &\leq 1 + \frac{1}{2} + \frac{1}{2} + \frac{1}{3} + \frac{1}{3} + \cdots + \frac{1}{n} + \frac{1}{n}, \end{aligned}$$

i.e. $T_{GLJD}(R) \leq 2H_n - 1$. Using the inequality $\frac{1}{k} \leq \int_{k-1}^{k} (\frac{1}{x})\, dx = \ln(k) - \ln(k-1)$ for $k \geq 2$, one gets $H_k \leq 1 + \ln(k)$ for $k \geq 2$, hence the result of the theorem.

Note that this proof shows that for if $r_{ij}$ is unscheduled after $k \geq 1$ iterations, then $r_{ij}$ is at least the $(1 + \lceil \frac{k}{2} \rceil)^{th}$ element of either its row or column. Thus a finer approximation is $m_{k+1} \leq \max(g_{1+\lceil \frac{k}{2} \rceil}, h_{1+\lceil \frac{k}{2} \rceil})$, where $g$ and $h$ are as defined in Theorem 4. Hence $\sum_{k=1}^{K} m_k \leq \max(g_1, h_1) + 2\sum_{k=1}^{2n-2} \max(g_{1+\lceil \frac{k}{2} \rceil}, h_{1+\lceil \frac{k}{2} \rceil}) = g_1 + 2\sum_{k=2}^{n} \max(g_k, h_k)$ (because $g_1 = h_1$). We can then get the result of the theorem by using the upper bound $max(g_k, h_k) \leq \frac{1}{k}$ for $k \geq 1$.

**Proof for Theorem 7:** For any $n \geq 3$, define $k$ such that $2^{k+1} - 1 \leq n < 2^{k+2} - 1$. Let $R_n$ be the following bloc-diagonal doubly-stochastic $n \times n$ matrix:

$$R_n = \begin{pmatrix} A_1 & 0 & \dots & \dots & \dots & \dots & 0 \\ 0 & A_2 & 0 & \dots & \dots & \dots & 0 \\ \vdots & 0 & \ddots & 0 & \dots & \dots & 0 \\ \vdots & \vdots & 0 & A_k & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & 0 & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix},$$

where for any $k' \geq 1$, $A_{k'}$ is the uniform doubly stochastic $2^{k'} \times 2^{k'}$ matrix of sum 1: $A_{k'} = \left[\frac{1}{2^{k'}}\right]_{ij}$. $R_n$ is an $n \times n$ doubly stochastic rate matrix (as in [31]). Theorem 4 provides the following lower bound on the length of the optimal LJ decomposition of $R_n$: $D(R_n) \geq 1 + \frac{1}{2} + \frac{2}{4} + \cdots + \frac{2^{k-1}}{2^k} = \frac{k+1}{2}$. Since $2^{k+2} - 2 \geq n$, $k \geq \log_2(n+2) - 2$, and thus $D(R_n) \geq \frac{\log_2(n+2)-1}{2}$.

**Proof for Corollary 8:** For $n \geq 3$, $\frac{\ln(n+2)}{2\ln 2} - \frac{1}{2} \leq BW(ILJD) \leq BW(GLJD) \leq 2\ln(n) + 1$.

**Proof for Theorem 9:** Let's prove that $m_k \geq g_k$ for a given $k \in \{1, ..., n\}$. By definition of $g_k$, there is an element $r_{ij}$ in the matrix $R$ such that $r_{ij} = g_k$, and $r_{ij}$ has the $k^{th}$ weight of its column $j$. In other words, in column $j$, there are at least $k$ elements with the same weight as $r_{ij}$. Since two elements from the same column can't be scheduled at the same time, by the pigeonhole principle, at least one of these $k$ elements does not belong to the first $k-1$ permutations - say it is $r_{i'j}$. $r_{i'j}$ thus belongs to some permutation $k'$ such that $k \leq k' \leq K$. But then $m_k \geq m_{k'} \geq r_{i'j} \geq r_{ij} = g_k$. Similarly, $m_k \geq h_k$.

**Proof for Theorem 10:** For $k \in \{1, ..., n\}$, let $f_k = \max(g_k, h_k)$. First, since $g_1 \geq ... \geq g_n$ and $h_1 \geq ... \geq h_n$, clearly $f_1 \geq ... \geq f_n$ and $n \cdot f_1 \geq \sum_{k=1}^{n} f_k$.

Second, from the comment in the proof of Theorem 6, we know that $T_{GLJD}(R) \leq f_1 + 2\sum_{k=2}^{n} f_k$. Third, from Theorem 9, we also know that $D(R) = T_{ILJD}(R) \geq \sum_{k=1}^{n} f_k$. Hence, combining these three results, we get $\frac{T_{GLJD}(R)}{T_{ILJD}(R)} \leq 2 - f_1/(\sum_{k=1}^{n} f_k) \leq 2 - \frac{1}{n}$.

**Proof for Corollary 11:** Consider $R$ such that $BW(GLJD) = T_{GLJD}(R)$. Then from Theorem 10 $BW(GLJD) \leq \left(2 - \frac{1}{n}\right) \cdot T_{ILJD}(R) \leq \left(2 - \frac{1}{n}\right) \cdot BW(ILJD)$.

**Proof for Theorem 13:** The inequality $\sum_{k=1}^{2d-1} \sigma_k \geq U$ is a property of maximal matchings [13]. By definition, $\epsilon \leq [1]_{ij} = \sum_{k=1}^{n} \pi_k$. Finally, $R = \frac{U}{d} + \epsilon$.