Guaranteed smooth switch scheduling with low complexity

Satya R. Mohanty and Laxmi N. Bhuyan
Department of Computer Science and Engineering, University of California, Riverside, CA 92521
satya@cs.ucr.edu, bhuyan@cs.ucr.edu

***Abstract*— A smooth scheduling with guaranteed rate service and bounded packet delay is a desired objective of any switch scheduling algorithm. We present a scheme that generates low jitter schedules with low computational complexity. The scheduler uses an integer decomposition of the rate-matrix, similar to the Birkhoff-von Neumann decomposition. It improves the delay and jitter performance of the smooth scheduler as described in Keslassy et. al. with an increase in the number of permutation matrices that the switch fabric has to cycle through. This increase is shown to be a constant for all practical purposes. Two algorithms are presented that have time complexity $O(n^2 + \log n)$ and space complexity $O(n^2)$ and $O(n)$ respectively. An existing scheduling algorithm for single links, *Smoothed Round Robin*, is employed for scheduling the permutation matrices. This algorithm has a computational complexity overhead of $O(1)$ and ensures smooth scheduling.**

I. Introduction

Amongst all high speed switch fabrics of core routers that are prevalent today, the crossbar is the most common owing to its simplicity and ability to support a variety of scheduling mechanisms. Typically a crossbar switch consists of N input ports and N output ports and each input is connected to every output. Most switches also employ input queuing with virtual output queues at each input port. This enables the switch fabric to operate at only a relatively moderate speedup compared to the input/output ports that operate at the line rate. Input switches with a moderate speedup core require some buffering at the output, however, when the crossbar runs at the same speed as the line-rate, only input buffers are required. In such systems, data transfer from inputs to outputs takes place synchronously during discrete time intervals via fixed number of bytes called cells. Therefore, in such switches, time can be thought of as being divided into slots and data transfer takes place during time slots. The objective is to design scheduling algorithms that guarantee conflict-free transfer, i.e an input port is not required to transmit cells to more than one output port in each time slot and, similarly, at most one cell can be transferred to any given output in one time slot.

II. Background and Related Work

In the frame based approach[1] by Keslassy et. al., at the start of a frame period, a matrix of rate requirements (from various input ports to corresponding output ports), called a guaranteed rate table (GRT), is assigned to the scheduler. The scheduler constructs a conflict-free matching called the *schedule table* from this GRT, and sends packets through the crossbar according to this matching.

This research was supported in part by NSF grant CCF 0311437

Chang et. al. first showed that achieving this desired conflict-free matching property of the scheduling objective is equivalent to the problem of decomposing the GRT matrix into constituent permutation matrices and enabling the crossbar connections corresponding to these permutations. The time that the switch remains in each configuration is proportional to the coefficient of the corresponding permutation matrix in the decomposition. They showed that an existing algorithm, called the Birkhoff-Von Neumann [2] decomposition, for decomposing any bimarkov matrix into a convex combination of permutation matrices could be used for this problem. Any switch that uses this algorithm is called the Birkhoff-Von Neumann switch, hereby abbreviated as BVN. The scheduling algorithm employed in this switch first transforms the GRT matrix into a doubly stochastic matrix and then decomposes the latter into a convex combination of individual permutation matrices through a series of iterations. Each iterative phase successively isolates a permutation until the decomposition is complete. The BVN decomposition is based on either a maximum matching or a simplified Ford-Fulkerson at each stage of the algorithm. Arekapudi et. al [3] showed how to speed up this process and also reduce memory access by using the Slepian-Duguid theorem that achieves the decomposition in one iteration.
Associated with this requirement are two potential problems, low throughput and packet delay. Our aim is to design scalable switch architectures using scheduling algorithms that have demonstrably low computational complexity, provide isolation, and guarantee good throughput, low packet delay and fairness properties to individual flows.

III. Low Jitter Algorithms

If the BVN decomposition of a rate matrix R is to generate a set of permutation matrices Y^k for $k = 1, 2, \ldots, K$ such that

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

the main objective of the *Integer Low Jitter Decomposition* (ILJD) is to approximate the rate matrix R such that

$$R \approx \sum_{k} m_k X^k, \ \ D = \min \sum_{m=1}^{K} m_k$$

where $X^k s$ are full or partial permutation matrices with the property that no two permutations have a non-zero entry in the same position. The motivation for this kind of a scheduling formulation is the authors' contention that the BVN decomposition causes any given entry to be striped across several permutation matrices, and therefore irrespective of the type of algorithm used to schedule the permutation matrices, there is no control on when the individual entries will be scheduled. It is shown that this integer programming problem (ILJD) is NP hard, and an approximation heuristic is provided as a solution. Similar algorithms have been proposed recently for slightly different problems, e.g. [4], [5].

This heuristic algorithm called the *Greedy Low-Jitter Decomposition* (GLJD) attempts to minimize the number of permutation matrices needed for approximating the GRT R, subject to the condition of non-overlapping positions of non-zero entries in each permutation. It also attempts to minimize the stretch factor D (the so called *bandwidth requirement* of the schedule tables. At each stage of the iteration, the GLJD greedily chooses non-conflicting entries (two entries of a matrix are said to be non-conflicting if they do not share the same row or column) of the GRT and groups them into a permutation matrix. The GRT can now be approximated by a linear combination of these permutation matrices instead of a convex combination as would be the case in the BVN decomposition. Further, in this linear combination, the coefficient of each of these permutation matrices is taken to be the largest entry of all the non-conflicting elements that constitute that permutation. The underlying assumption is that these entries are approximately equal. When this assumption does not hold, then a flow can be impacted adversely. The following example illustrates this fact.

Consider a GRT in the form of a doubly stochastic matrix R with entries r_{ij}, where r_{ij} represents the desired rate from the input port i to the output port j.

$$R = \begin{bmatrix} 0.48 & 0.04 & 0.48 \\ 0.04 & 0.48 & 0.48 \\ 0.48 & 0.48 & 0.04 \end{bmatrix}$$

The GLJD generates the following decomposition for this GRT.

$$\begin{aligned} R \leq \quad & 0.48 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + \quad 0.48 \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \\ & +0.48 \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \end{aligned}$$

The schedule time obtained by this algorithm is $0.48 \times 3 = 1.44$. Therefore it cannot accommodate a low-jitter traffic load of more than $\frac{1}{1.44} = 0.694$ of the offered traffic. The reason for the discrepancy is the compromise made in generating greedily fewer number of permutation matrices than the BVN decomposition. For instance, the entry $(3,3)$ desiring only a rate of 0.04 is associated with the permutation having a weight coefficient of 0.48.

It is instructive that one can obtain a different decomposition with a larger number of permutation matrices but with a better stretch factor D. The idea is to represent each entry in the GRT according to its binary representation, separate out entries that have the similar powers of 2 into individual matrices, and then decompose these matrices into permutation matrices. For example the GRT of the previous example would be decomposed as:

$$R = \begin{bmatrix} 0.32+0.16 & 0.04 & 0.32+0.16 \\ 0.04 & 0.32+0.16 & 0.32+0.16 \\ 0.32+0.16 & 0.32+0.16 & 0.04 \end{bmatrix}$$

$$\begin{aligned} R \quad = \quad & 0.32 \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix} + 0.16 \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix} \\ & +0.04 \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

which can finally be decomposed as a convex combination of permutation matrices as shown below.

$$\begin{aligned} R \quad = \quad & 0.32 \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} + 0.32 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \\ & +0.16 \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} + 0.16 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \\ & +0.04 \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

This decomposition is exact but comes at a higher cost, the number of matrices is now increased. The key question therefore is to have an upper bound for the number of these matrices. We know from [1], using the results of the greedy on-line edge coloring problem [6] (Bar-Noy et. al), that the greedy decomposition requires at most $2N-1$ matrices. If we assume that the granularity of bandwidth is 1 bit (in practice a valid assumption), then a bandwidth of 4 gigabits will necessitate at most 32 matrices. Thus we will need a total of $32(2N-1)$ matrices. The matrices (some of them may be identical) can be can be economically stored by using efficient encoding schemes to represent the unique ones. In the above example, finally, there are only 3 unique permutation matrices.

$$\begin{aligned} R \quad = \quad & 0.48 \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} + 0.48 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \\ & +0.04 \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

In general if the maximum allocated bandwidth is R_{max}bits, and the minimum bandwidth is 1bit, for an $N \times N$ input crossbar switch the upper bound on the number of permutations is $(2N-1)\log R_{max}$. If the minimum bandwidth is

```
1) $\forall i, j$ encode each entry $r_{ij}$ in binary i.e.
   $r_{ij} = a_k a_{k-1} \ldots a_0$; $\forall i, j = 1, \ldots N$
2) for $l = 0 \ldots k$,
     if $a_l == 1$
         $X^l_{ij} = 1$
     else
         $X^l_{ij} = 0$
```

Fig. 1. Matrix Decomposition into binary weights

some floating point number some appropriate scaling needs to be done at first so that the GRT has only integer entries. If the highest possible entry is $R_{\mathcal{C}}$ after the scaling conversion, then the number of permutations is $(2N-1)logR_{\mathcal{C}} \approx O(N)$. In practice, it is likely that this upper bound is not reached, as is evident from the above example. We see therefore that the premium on space for storing these permutations is only increased by a constant for all practical purposes.

A. *Binary Matrix Decomposition* BMD

Given any matrix R (and therefore any GRT), BMD proceeds in two steps, first invoking Algorithm 1 and then 2. These algorithms are presented formally in pseudo-code in Figures 1 and 2. The first algorithm groups together similar powers of 2 from each entry of R into a matrix (the associated coefficient is the power of 2 involved in the grouping). The second algorithm takes each such matrix, greedily decomposes it into unique permutation or sub-permutation (partial) matrices. At the end of the final decomposition we have K (maybe partial) permutation matrices, every permutation matrix associated with the same coefficient is non-conflicting and we will refer to them as switching matrices, matrix X^i for instance will refer to the ith switching matrix.

The rationale behind this argument is that the granularity of allocating bandwidth is discrete and the fact that the logarithm of the maximum port capacity is of the order of a few tenths or at most a hundred, and likely to remain so in the forseeable future. Once this decomposition is achieved, it is necessary to schedule these permutations or switching matrices.

Computational Complexity: Algorithm 1 has a computational complexity of $O(N^2 logR_{\mathcal{C}}) \approx O(N^2)$. Greedy decomposition of an individual matrix is $O(N^2 log\, N)$. Since at the end of Algorithm 1, we have $log\ R_{\mathcal{C}}$ number of matrices, where $R_{\mathcal{C}}$ can be taken to be a constant, the total computational complexity is $O(N^2 log\, N)$.

We make an observation here that provides some insight into the following approach that we adopt to schedule the switching matrices. In line with the idea of Keslassy [1], we observe that apart from the number of permutations across which a given port-pair entry is striped, the jitter also depends on the intervening periods in which a particular switching matrix is employed in the configuration. In various switching papers this aspect has been fulfilled by using a generalized processor

```
1) $k := 0$, and $\forall R \in \{X^i\}$ in Algorithm 1 do:
2) Sort all $(i, j)$ elements of the matrix $R$ in descending order into list $L$
3) do until $L \neq \emptyset$
     • Let $c_{i'j'} = Pop(L), c_{i'j'}$ is head of $L$
     • Let $NC(i'j')$ be the set of all non-conflicting non-zero entries of $L$, i.e. $NC(i'j') = \{x_{ij}\} \mid x_{ij} \in L, i \neq i', j \neq j'$.
     • $\forall x_{ij} \in NC(i'j'$, set corresponding $(i, j)$ entries of an empty matrix $P_l$ to 1. The matrix $P_l$ will be associated with a weight coefficient $c_{i'j'}$.
     • If $\exists j \mid j < l \wedge P_j = P_l$
         discard $P_l$; $m_j = m_j + c(i'j')$
       else
         Set $P_k = P_l, m_k = c(i'j'), k = k + 1$;
     • Delete the $NC(i'j')$ entries from list $L$ i.e. $L = L - NC(i'j')$.
```

Fig. 2. Greedy Non Conflicting decomposition

type [7] approach, either determining a start [1] or a finish time [1], [2] for the jth schedule of matrix X_k.

This contributes another $O(N)$ complexity to the total scheduling. We propose to solve this problem through an O(1) complexity scheduling method, the Smoothed Round Robin [8]. This algorithm was proposed for differentiating flows based on reservations for a single link. We will apply the basic idea of this method to switch scheduling.

IV. Smoothed Round Robin SRR scheduling for switching matrices

The SRR algorithm is described in detail in [8]. Nevertheless, we present an overview of the SRR algorithm for reasons of clarity. The key idea in SRR is to use a weight matrix WM and an associated weighted spread sequence WSS, the algorithm scans the terms of the WSS to determine which flows to serve. Each row of the weight matrix represents the reservation of a flow encoded in binary. For any given weight matrix and a corresponding WSS, the SRR algorithm scans the WSS term by term; when the term is i, it will serve the column $k - i$, where k is the order of the WSS. All flows that have a non-zero entry for that column will be serviced. A small example should suffice to illustrate. Suppose there are 3 flows f_1, f_2, f_3 with rates $r_1 = 64Kbps$, $r_2 = 128Kbps, r_3 = 256Kbps$, competing for a single link of capacity $512Kbps$. The normalized weights of the flows are $w_1 = 1, w_2 = 2, w_3 = 4$. Then the WM is given by

$$WM = \begin{bmatrix} WV_1 \\ WV_2 \\ WV_3 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

and corresponding WSS is $1, 2, 1, 3, 1, 2, 1$. The flow service sequence will be given by

$$f_3, f_2, f_3, f_1, f_3, f_2, f_3.$$

TABLE I
NOTATION USED IN ALGORITHMS AND PROOFS

M	Weight Matrix
$N \times N$	Crossbar switch, N input ports, N output ports
Y^i, X^i	i^{th} Permutation matrices (possibly partial)
K	Number of Permutation matrices
m_i	weight of i^{th} permutation matrix
L_{max}	Length of a cell (in bytes)
S^i	The i^{th} spread sequence
D_i	The i^{th} doubly linked list
P_{dl}	Iterator for the doubly linked list
P_C	Iterator for scanning the WSS
$k_s, (k_{max})$	Current (Maximum) Order of the spread sequence
col	Column of the WM currently being scanned
C	Bandwidth of each port
J_{max}	Maximum jitter that can be encountered

Instead of considering individual flows, we take into account each switching matrix X^k itself. The weighted coefficient of X^k in the decomposition represents the fraction of the normalized desired input-output port rate at which data is to be transferred when the switching configuration is set to X^k. The GRT is generalized, if necessary, to a doubly sub-stochastic matrix) to form the WM. This may be done according to some definition of fairness measure[9], [10]. Also the coefficients m_k, encoded in binary, constitute the k^{th} row of WM. In what follows we will denote as a round the total length of a frame in slots. We call the scheduling algorithm that we propose SRRSW (Smooth Round Robin for switches).

A. *Algorithm* SRRSW

The Pseudo-code for SRRSW consists of three main routines, Add matrix(), Schedule Matrix() and Delete Matrix() as presented in Figure 3. Add() and Delete() subroutines are called at the start of a fresh round. The lowest granularity at which bandwidth is allocated to a port pair is the slot transfer time, i.e. the time taken by a cell to traverse the switch fabric. SRRSW scans the WSS (generated for a given GRT), to get a column of the WM, and sets up crossbar connections according to switch configurations that have non-zero entry in that column of the WM.

B. *Elementary Properties OF* SRRSW

Similar to SRR, SRRSW always forwards packets when there are active flows in the system and is therefore work-conserving (considering the switch as the scheduling entity, not for the port-pair). Some elementary properties of SRRSW are introduced here for sake of completeness. The reader can refer to [8] for details. In a switch scenario, it helps to identify the outport j^{th} bound traffic from input port i, (hereby referred to as port-pair (ij)) as analogous to a flow in the single-link scenario.

Fact 4.1: SRRSW finishes a round when it starts from the first term of the kth WSS, and after visiting all the $1, 2 \ldots k$ terms, back to the beginning of the sequence again.
Analysis of frame based schedulers (including BVN,GLJD) all assume that all port-pair (ij) traffic is continuously backlogged during a frame round. Assuming this is true the following holds

```
1) Add Matrix($X^i$)
   At the start of a frame round:
   • $id = getXid(X^i)$.
   • Form a Weight Vector $WV_{id}$ for $X^i$ from $m_i$.
   • Add $WV_{id}$ to last row of Matrix $M$.
   • Insert nodes corresponding to $WV_{id}$ into
     $DL_0, DL_1, \ldots, DL_{k_{max}-1}$
   • if new columns are added into $M$
       update $k_s$;
2) Delete Matrix($X^i$)
   At the start of a frame round:
   • $id = getXid(X^i)$
   • Remove the $WV_{id}$ row from $M$.
   • Remove corresponding nodes from
     $DL_0, DL_1, \ldots, DL_{k_{max}-1}$ according to
     the coefficients of $X^i, m_i$.
   • if(empty columns are deleted from M)
       update $k_s$;
       $Pc = Pcmod(2k_s)$;
3) Schedule Matrix()
   During a frame round:
   $col$ is current column of WSS
   • $loop$: do while $(P_{dl} \rightarrow X_{id} \neq tail_{col})$
       Schedule $(P_{dl} \rightarrow X_{id})$
       $P_{dl} \rightarrow X_{id} = P_{dl} \rightarrow next$
   • $P_c = P_c + 1$. If $P_c == 2^{k_s}, P_c = 1$
   • Find next $col$ to scan, set $P_{dl}$ accordingly.
     Go to $loop$.
```

Fig. 3. Add, Delete and Schedule Matrix Algorithms

Fact 4.2: SRRSW visits port pair (ij) r_{ij} times in a round, where r_{ij} is the weight of port-pair (ij).

Proof: To see that fact 4.2 holds, we make the observation that each permutation matrix X^k with coefficient m_k is visited by SRRSW m_k times in a round, and since $r_{ij} = \Sigma_{k=0, X^k(ij) \neq 0}^{K} m_{ij}$, each input-output port pair (ij) is indeed visited r_{ij} times in a round. ■

Fact 4.3: Suppose port-pair (ij) is backlogged, and has been visited by SRRSW x times from time 0 to t, and $S_{ij}(0,t)$ denotes the bytes served by SRRSW of port-pair (ij), then,

$$S_{ij}(0,t) = xL_{max}$$

Fact 4.4: Suppose port-pairs $(ij), (i'j')$ with desired rates $(r_{ij}, r_{i'j'})$ are continuously backlogged in a round, and the number of visits to each by SRRSW is V_{ij} and $V_{i'j'}$ respectively. Then at the end of a round

$$\frac{r_{ij}}{V_{ij}} - \frac{r_{i'j'}}{V_{i'j'}} = 0$$

C. *Analysis of* SRRSW *Jitter Bounds*

We now provide some important performance results concerning the jitter, which was one of the primary motivations behind this work. The following corollary is obvious from Fact 4.2

Corollary 4.5: The mean jitter for the traffic from input port i to output port j with a desired rate (weight) r_{ij} is $\frac{1}{r_j}$.

Proof: The total number of visits to all switching matrices X^k, in which entry ij exists is r_{ij}. Therefore the mean jitter is $\frac{1}{r_{ij}}$. ■

The above corollary guarantees that the desired mean jitter is achieved. However we would like to explore whether a bound exists on the worst case jitter. A trivial upper bound is the scheduling interval for a complete frame. We would like to determine whether a tighter bound exists. The following theorem establishes that such a bound does in fact exist.

Theorem 4.6: Suppose there are K permutation matrices at the end of the BMD and the Greedy Non Conflicting algorithms in Figures 1,2. Let $m_k = \Sigma_{n=0}^{i} a_{k,i} = 1$, and $i \leq k_s - 1$, where k_s is the order of the WSS. Then the maximum jitter,

$$J_{max} \leq \frac{2L_{max}}{m_k} + \frac{L_{max}(K-1)}{C}$$

Proof: An input-output port-pair (ij) is allowed to transmit cells only when the scheduler chooses a switching matrix X^k such that the entry X_{ij}^k is non-zero. We have to find the maximum value of intervals, in terms of time slots, between two consecutive visits to switching matrices that have a non-zero (ij)-th entry.

Notice that even though the problem statements are similar, this is not equivalent to the *Schedule Delay Bound Theorem* (Th. 3) of SRR. In SRRSW since a port-pair (ij) may be striped across different permutation matrices, it is not enough to consider the maximum number of intervals between two contiguous visits of the algorithm to the same column of the WM. Notice, however, that the worst case will happen when an $(ij)^{th}$ entry is present only in one switching matrix, call this X^f. The weight of this switching matrix X^f is denoted by $m_f = \Sigma_{n=0}^{k} a_{f,n} 2^n$. There are two cases:

1) $2^i \leq m_f < 2^{i+1} - 1$.

if $m_f = 2^i$, then clearly there exists an integer i_1 s.t $a_{f,i_1} = 0$. Otherwise, assume for the sake of contradiction that there does not exist $i_1, i_1 < i,\ s.t\ a_{f,i} = 0$. Then $m_f = 2^{i+1} - 1$, which violates the assumption. So i_1 exists. Between adjacent visits to $(ij)^f$, the algorithm will visit the switching matrices that have entries in the columns according to the spread spectrum $S^{k_s-i_1-1}, (k_s - i_1), S^{k_s-i_1-1}$, and the column i. Thus

no. of visits $= 2\Sigma_{j=1}^{K} a_{j,i+1} + 2^2\Sigma_{j=1}^{K} + \ldots$
$+2^{k_s-i-1}\Sigma_{j=1}^{K} a_{j,k_s-1} + \Sigma_{j=1}^{K} a_{j,i_1} + \Sigma_{j=1}^{K} a_{j,i}$
$= \Sigma_{n=i}^{k_s-1}(2^{n-i}\Sigma_{j=1}^{K} a_{j,n}) + \Sigma_{j=1}^{K} a_{j,i_1}$
$= \frac{1}{2^i}\Sigma_{n=i}^{k_s-1}(2^n\Sigma_{j=1}^{K} a_{j,n}) + \Sigma_{j=1}^{K} a_{j,i_1}$
$= \frac{1}{2^i}(\Sigma_{n=i}^{k_s-1}(2^n\Sigma_{j=1}^{K} a_{j,n}) - \Sigma_{n=0}^{i-1} 2^n \Sigma_{j=1}^{K} a_{j,i_1}) + \Sigma_{j=1}^{K} a_{j,i_1}$
$= \frac{1}{2^i}(C - \Sigma_{n=0}^{i-1} 2^n \Sigma_{j=1}^{K} a_{j,i_1}) + \Sigma_{j=1}^{K} a_{j,i_1}$

Therefore,

$J_{max} = \frac{L_{max}}{C}[\frac{1}{2^i}(C - \Sigma_{n=0}^{i-1} 2^n \Sigma_{j=1}^{K} a_{j,i_1}) + \Sigma_{j=1}^{K} a_{j,i_1}]$
$< \frac{2L_{max}}{2^i} - \frac{L_{max}}{C}(\frac{1}{2^i}\Sigma_{j=1}^{K} a_{j,n} - \Sigma_{j=1}^{N} a_{j,y})$

2) $m_f = 2^{i+1} - 1$.

Here the chain is at most $S^{k_s-i-1}S^{k_s-i-1}$, Therefore the no. of visits $= 2\Sigma_{j=1}^{K} a_{j,i+1} + 2^2\Sigma_{j=1}^{K} a_{j,i}$

and similarly, like in the previous case, $< \frac{2L_{max}}{2^i} - \frac{L_{max}}{C}(\frac{1}{2^i}\Sigma_{j=1}^{K} a_{j,n})$

From both of the cases, the maximum jitter is $J_{max} < \frac{2L_{max}}{m_f} + \frac{L_{max}(K-1)}{C}$ ■

Since K is $O(N)$, then $J_{max} = O(N)$, this means that as the number of ports increase, the jitter deteriorates linearly. It also means that although each flow gets its share in a round according to its weight, the worst case jitter is bounded by the number of permutation matrices.

V. Conclusions and Future Work

We propose a method to reduce the number of permutation matrices and achieve low jitter. The SRRSW algorithm is readily scalable because the SRR algorithm is inherently scalable. By adjusting the rate granularity, different bandwidths can be accomodated. SRRSW is quite suitable for high-speed networks where the configuration overhead in scheduling permutation matrices is not high. As is common to all round-robin schedulers SRRSW cannot provide short-term fairness. An interesting problem to explore would be to determine the minimum speedup required to provide hard guarantees, and whether such guarantees are possible at all. Future work can also explore similar round-robin based low complexity scheduling algorithms deployed in switches and the performance evaluation of a network consisting of such switches through analytical or simulation techniques.

References

[1] I. Keslassy, Murali Kodialam, T.V.Lakshman, and Dimitrios Stilladis. On guaranteed smooth scheduling for input-queued switches. *IEEE INFOCOM*, March 2003.

[2] C.S Chang, J.W. Chen, and H.Y. Huang. On service garantees for input-buffered crossbar switches: a capacity decompostion approach by birkhoff and von neumann. *IEEE IWQoS, London*, 1999.

[3] Srikanth Arekapudi, Shang-Tse Chuang, Isaac Keslassy, and N. McKeown. Confi guring a load balanced switch in hardware. *Hot Interconnects 12, Stanford*, August 2004.

[4] Alexander Kesselman and Kirill Kogan. Non-preemptive scheduling of optical switches. *Proceedings of the IEEE Global Telecommunications Conference*, October 2004.

[5] Marcelo Prais and Celso C. Ribeiro. Reactive GRASP: An application to a matrix decomposition problem in TDMA traffi c assignment. *INFORMS Journal on Computing*, 12:164–176, 2000.

[6] A. Bar-Noy, R. Motwani, and J. Naor. The greedy algorithm is optimal for on-line edge coloring. *Information Processing Letters*, 44(5):251–253, 1992.

[7] Abhay K. Parekh and Robert G. Gallager. A generalized processor sharing approach to fbw control in integrated services networks: The single-node case. *IEEE/ACM Transactions on Networking*, June 1993.

[8] Guo Chuanxiong. SRR: an O(1) time complexity packet scheduler for fbws in multi-service packet networks. *Proceedings of the ACM SIGCOMM,*, August 2001.

[9] R. Yim, N. Devroye, V. Tarokh, and H.T Kung. Achieving fairness in two-dimensional generalized processor sharing. In *Proceedings of the 22nd Biennial Symposium on Communications*, pages 185–187, April 2004.

[10] Xiao Zhang and Laxmi Bhuyan. Achieving fairness and throughput for best-effort traffi c in input-queued crossbar switches. 2005.