Universal Bounds on Buffer Size for Packetizing Fluid Policies in Input Queued, Crossbar Switches

Michael Rosenblum[1]
Department of Mathematics
Massachusetts Institute of Technology
Email: mrosen@math.mit.edu

Michel X. Goemans[†]
Department of Mathematics
Massachusetts Institute of Technology
Email: goemans@math.mit.edu

Vahid Tarokh[‡]
Division of Engineering
and Applied Sciences
Harvard University
Email: vahid@deas.harvard.edu

Abstract— **In this paper, we consider a type of on-line, traffic scheduling problem in input queued, crossbar switches. The input to a problem, at each time step, is a set of desired traffic rates. These traffic rates in general cannot be exactly achieved since they assume arbitrarily small fractions of packets can be transmitted at each time step. The goal of the traffic scheduling problem is to closely approximate the given sequence of traffic rates by a sequence of switch uses in which only whole packets are sent. The focus of this paper is bounding the costs incurred in using such an approximation, in terms of the additional buffer size required.**

We establish universal bounds on the additional buffer size due to sending only whole packets; these bounds do not depend on the particular distribution of the input traffic, require no speedup, and guarantee 100% throughput. Specifically, for an $N \times N$ input queued, crossbar switch, an on-line, packetizing algorithm is presented that guarantees 100% throughput with a buffer requirement of $(N+1)^2/4$ packets per input port with no speedup. The algorithm can be improved to run in $O(N \log N)$ time, using a fast algorithm for edge-coloring bipartite multigraphs. In the reverse direction, it is shown for an $N \times N$ input queued, crossbar switch, that any on-line, packetizing algorithm with no speedup requires a buffer size of $N/e - 2$ packets per input port. We also extend the main packetizing algorithm in this paper to a general class of switch architectures.

Keywords: Deterministic network calculus, Graph theory, Combinatorics.

I. INTRODUCTION AND RELATED WORK

Scheduling algorithms for input queued, crossbar switches have received growing attention due to the massive growth of communication networks. In one approach, the designer first ignores the packet nature of traffic and constructs a schedule under the assumption that packets can be broken into arbitrarily small pieces and sent at different time slots (e.g. [1]–[4]). This schedule is referred to as a *fluid policy*. Next, the designer constructs a *packetized policy*, which approximates the behavior of the fluid policy in order to send packet data.

Other approaches to scheduling are also possible. Much analysis has been done in the case where the input traffic is assumed to have certain statistical properties (e.g. [5]–[7]). In this input traffic model, it is often shown that the queue lengths, considered as a stochastic process, converge to a limiting distribution with finite expectation. The analysis of packetizing fluid policies in this paper, however, as in most of the related work discussed below, focuses on worst-case bounds rather than probabilistic bounds. We give universal bounds on buffer requirements that do not depend on any statistical properties of the input traffic. These bounds on required, additional buffer size are not asymptotic; they apply to all switch sizes and to time increments of any finite duration. In particular, one of the main results in this paper is an on-line, packetizing algorithm that guarantees 100% throughput with a buffer requirement of $(N+1)^2/4$ packets per input port for an $N \times N$ input queued, crossbar switch with no speedup.

Packetizing fluid policies can significantly simplify schedule design. A number of authors have taken this approach in constructing scheduling algorithms that provide certain performance guarantees. Parekh and Gallager proposed Generalized Processor Sharing (GPS),

[1],[‡] This material is based upon research supported by the National Science Foundation under the Alan T. Waterman Award, Grant No. CCR-0139398. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

[1],[†] This material is based upon research partly supported by the National Science Foundation under contracts ITR-0121495 and CCR-0098018. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

which creates a fluid policy that implements weighted fair queueing on a $1 \times N$ crossbar switch [1], [2]. The fluid policy is then approximated by a packetized policy called Packet-by-Packet Generalized Processor Sharing (PGPS). PGPS determines packet service order based on GPS's packet-completion times.

Bennett and Zhang [8] present another GPS-approximating, packetized policy called Worst-case Weighted Fair Queueing. This scheme services the packet with earliest GPS finishing time (assuming no new packets arrive) among those that have already begun GPS service. This scheme, however, suffers from abrupt increases or decreases in delay (known as jitter), as pointed out by Stamoulis and Liebeherr [9]. The jitter problem is addressed in [9] with a modified version of GPS, Slow-Start Generalized Processor Sharing, which creates a smoothed version of the GPS fluid policy. This smoothed fluid policy is then approximated by a packetized policy that determines packet service order based on the fluid policy's packet-completion times.

In the related work described so far, the foundation for each algorithm is a fluid policy created according to GPS. Duffield, Lakshman, and Stiliadis question "the notion that queueing systems should closely emulate a GPS system" [10]. They point out that GPS-emulating systems allocate unused resources to backlogged sessions in proportion to weights that are fixed to reflect long-term bandwidth requirements of traffic flows. These systems thus ignore the instantaneous quality of service needs of applications in redistributing excess bandwidth. Duffield, Lakshman, and Stiliadis present modified fair-queuing schemes that attempt to remedy this problem, by redistributing excess bandwidth not in proportion to weights, but based on maximum flow delay and maximum queue backlog [10].

Stamoulis and Giannakis [11] also point out limitations of emulating GPS systems. In GPS, delay and bandwidth guarantees are coupled in the fixed weights used to schedule packets. To decouple delay and bandwidth allocations, Stamoulis and Giannakis propose using deterministically time-varying weights [11].

The use of time-varying weights in GPS and the redistribution of excess bandwidth based on factors other than weights result in more general fluid policies than in pure GPS. This motivates the investigation of how well one can packetize an arbitrary fluid policy, which may schedule different amounts of fluid at each time step, in terms of minimizing additional delay, buffer size, and jitter. Up to this point, we have described related work focusing on the $1 \times N$ switch; packetizing fluid policies has also been studied in the context of the $N \times N$ crossbar switch.

Chang, Chen, and Huang [12] present a packetizing algorithm for fluid policies on an $N \times N$ crossbar switch, in which the same amount of fluid is scheduled at each time step, and so can be represented by a single, rate matrix. The entries of the rate matrix represent the fractions of bandwidth allocated to users of the switch. Ideally, service at each time slot should exactly mimic the behavior of the rate matrix, but in practice only whole packets can be scheduled; thus, the fluid behavior can only be approximated by a packetized policy. Their packetizing algorithm, based on the Birkhoff-von Neumann decomposition, requires initial run-time $O(N^{4.5})$ and on-line run-time $O(\log N)$ for an $N \times N$ crossbar switch. In this paper, we consider more general fluid policies, in which the amount of fluid scheduled can vary at each time step.

Tabatabaee, Georgiadis, and Tassiulas [13] consider the open problem of packetizing arbitrary fluid policies in an $N \times N$ crossbar switch using first-come first-served, virtual output queues. They attempt to characterize fluid policies that can be packetized so that for each pair of input and output ports, at each time step, the cumulative number of packets sent between these ports differs from the cumulative fluid scheduled between these ports by less than 1; such fluid policies are said to be *trackable.* They prove that any fluid policy for the 2×2 crossbar switch can be tracked, and propose several heuristics for packetizing fluid policies on larger switches. Bonuccelli and Clo [14] construct a simple fluid policy for the 4×4 crossbar switch that cannot be tracked. This untrackable fluid policy can be extended to larger switch sizes.

The result of Bonuccelli and Clo [14] that for many switch sizes there exist untrackable fluid policies, has motivated us to consider a relaxed version of tracking, focusing on buffer size. The present work tackles the problem of bounding additional buffer size when packetizing fluid policies for input queued, crossbar switches. The contributions of this paper are described next.

Section II formulates the problem to be considered in this paper and establishes the notation. In particular, we define backlog, which quantifies how much additional buffer space is used by a packetized policy than is used by the fluid policy it is emulating. The term "buffer requirement" refers to this additional buffer space, or backlog. In Section III, we prove tight bounds on the buffer requirements for packetizing arbitrary fluid policies on $1 \times N$ input queued, crossbar switches. The bounds are universal in that they do not depend on the distribution of the fluid input and assume no speedup. Specifically, for a $1 \times N$ input queued, crossbar switch, a greedy, on-line, packetizing algorithm is presented that guarantees 100% throughput with a buffer requirement of N/e packets per input port with no speedup. We also prove that this packetizing algorithm is optimal for minimizing the buffer requirement in the worst-case. These results for the $1 \times N$ input queued, crossbar switch were proved independently in a recent paper by Adler et al. [15]. In Sections IV and V, we treat $N \times N$ input queued, crossbar switches. We present an on-line, packetizing algorithm that guarantees 100% throughput with a buffer requirement of $(N+1)^2/4$ packets per input port with no speedup. Moreover, it is shown that any on-line, packetizing algorithm with no speedup has a buffer requirement of at least $N/e-2$ packets per input port. In Section V we show for any $\epsilon > 0$ how to modify this algorithm to have $O(\frac{1}{\epsilon} N \log N)$ run-time per fluid arrival, with a buffer requirement of $(1+\epsilon)(N+1)^2/4$ packets per input port. Section VI extends the results of Section IV to more general switch architectures than the crossbar switch, using a resource allocation model of Tassiulas [6]. Finally, Section VII presents our conclusions and directions for future research.

II. Notation and Definitions

We consider the $N \times M$ input queued, crossbar switch operating in discrete time with no speedup. All packets are assumed to have the same size. The line rate (capacity) at all input and output ports is one packet per time step. This switch allows one to send, in one time step, packets from any of the N input ports to any of the M output ports. The only constraints are that in one time step, at most one packet can leave a single input port, and at most one packet can arrive at a single output port. The switch uses virtual output queueing to avoid head-of-line blocking. We first define the notion of a fluid policy, which represents the ideal, packet-scheduling behavior desired.

A *Fluid Policy* for an $N \times M$ crossbar switch is a sequence of (fictitious) switch uses specifying the number of fractional packets that ideally would be sent and received at every time step. This is represented by a sequence of $N \times M$ doubly sub-stochastic matrices[1] $\{F^{(k)}\}_{k>0}$. Then, $F_{ij}^{(k)}$ represents the fraction of a packet that should be sent from input i to output j at time step k. We next define a packetized policy, which should approximate the fluid policy.

A *Packetized Policy* for an $N \times M$ crossbar switch is a sequence of switch uses where only whole packets are sent and received at each time step. It is represented by a sequence of $N \times M$ sub-permutation[2] matrices, $\{P^{(k)}\}_{k>0}$. The value of $P_{ij}^{(k)}$ is 1 if and only if a packet is sent from input i to output j at time step k; otherwise it has value 0.

It is convenient to record, for each input port i and output port j, the difference between the cumulative number of fractional packets scheduled by the fluid policy up to and including time k, and the cumulative number of whole packets sent by the packetized policy up to and including time k. This information is stored in the $N \times M$ matrix $C^{(k)}$, for $k \geq 0$. In particular, $C^{(0)} := \mathbf{0}$, and $C^{(k)} := C^{(k-1)} + F^{(k)} - P^{(k)}$ for $k \geq 1$.

A *packetizing algorithm* takes a fluid policy as input, and outputs a packetized policy with the property that it never sends a packet before the fluid policy has sent some part of that packet. In other words, for all k, the packetized step $P^{(k)}$ is only allowed to have value 1 in entries (i,j) for which $C_{ij}^{(k-1)} + F_{ij}^{(k)} > 0$; this ensures that all entries of the matrix $C^{(k)}$ are greater than -1. We consider only on-line, packetizing algorithms, that is algorithms which have access only to $F^{(1)}, ..., F^{(k)}$ in computing $P^{(k)}$. See Figures 1 and 2 for examples of fluid policies, packetizied policies, and their cumulative differences.

We say that a packetizing algorithm *tracks* if for any given fluid policy, it outputs a packetized policy with the following property: The packetized policy sends each packet at or before the time step in which the fluid policy has scheduled 1 unit of fluid to it [13]. In other words, for all k, the maximum entry of the matrix $C^{(k)}$ is less than 1.

Bonuccelli and Clo [14] construct a fluid policy for the 4×4 crossbar switch that cannot be tracked. Their construction implies that for any $N \times M$ input queued, crossbar switch with $N, M \geq 4$, there exists a fluid policy that cannot be tracked. This negative result motivates the study of packetizing algorithms that satisfy a relaxation of the tracking property defined above.

Backlog, defined next, quantifies how much additional buffer space is used by a packetized policy than is used by the fluid policy it is emulating. For time step k, and

[1] A non-negative valued matrix is doubly sub-stochastic if all its row sums and column sums are ≤ 1. If the row sums and column sums all equal one, the matrix is doubly stochastic.

[2] A $\{0,1\}$-valued matrix is a sub-permutation matrix if all its row sums and column sums are ≤ 1. A permutation matrix is a sub-permutation matrix with a 1 in every row and every column.

$$
\begin{array}{ccccc}
\text{Fluid Policy } F^{(k)} & & \text{Packetized Policy } P^{(k)} & & \text{Cumulative Differences } C^{(k)} \\
\begin{bmatrix} \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \end{bmatrix} & , &
\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} & , &
\begin{bmatrix} \frac{1}{2} & 0 & 0 & -\frac{1}{2} \\ 0 & \frac{1}{2} & -\frac{1}{2} & 0 \\ 0 & -\frac{1}{2} & \frac{1}{2} & 0 \\ -\frac{1}{2} & 0 & 0 & \frac{1}{2} \end{bmatrix} \\
\begin{bmatrix} 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \end{bmatrix} & , &
\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} & , &
\begin{bmatrix} \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ 0 & 0 & 0 & 0 \\ -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 0 \end{bmatrix} \\
\begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{bmatrix} & , &
\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} & , &
\begin{bmatrix} 0 & 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & 0 & -\frac{1}{2} & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & -\frac{1}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{bmatrix}
\end{array}
$$

Fig. 1. Three steps of a fluid policy (first column), packetized policy (second column), and their cumulative difference (third column). Note that since a packetized step can only schedule packets for which some fluid has arrived, $P^{(3)}$ cannot be a permutation matrix; it can only be a sub-permutation matrix. At time step 3, the backlog at input ports $1, 2, 3, 4$ (corresponding to the rows of $C^{(3)}$) is $1, 1/2, 1/2, 1$ respectively. The total backlog at time step 3 is 3.

for a set of pairs of input and output ports, we define their *backlog* to be the positive part of the difference between the cumulative amount of fluid scheduled between these pairs by the fluid policy, and the cumulative number of packets sent between these pairs by the packetized policy. This can be found by summing the entries of the positive part[3] of the cumulative difference matrix, $(C^{(k)})^+$, corresponding to these pairs of input/output ports. In particular, the backlog at input port i is the sum of entries in row i of $(C^{(k)})^+$. The total backlog of all input/output ports is the sum of all the entries of $(C^{(k)})^+$. See Figure 1 for an example.

For a set of pairs of input and output ports, we say that a packetizing algorithm gets at most *b-backlogged* if for all time steps k, their backlog is no more than b. In other words, for all k, the sum of corresponding entries in $(C^{(k)})^+$ is at most b. For example, we say that a packetizing algorithm gets at most b-backlogged per input port if in each row of $(C^{(k)})^+$, the sum of entries is at most b at all time steps.

III. Tight Bounds on Backlog for the $1 \times N$ Crossbar Switch.

Consider the $1 \times N$ crossbar switch, that is, a switch with a single input port and N output ports. The definition of backlog for the $1 \times N$ crossbar switch reduces to the following: For a set of output ports, at time k, their corresponding backlog is the positive part of the difference between the cumulative fluid scheduled to leave these ports by the fluid policy up to and including time k, and the cumulative number of packets sent through these output ports by the packetized policy up to and including time k.

We show that for the $1 \times N$ crossbar switch, the greedy algorithm, which is defined in part B of this section, for building a packetized policy from any given fluid policy is optimal in terms of minimizing the worst-case backlog over all possible sets of output ports. First, we prove that for every on-line, packetizing algorithm, there exists an adversarial fluid policy that gives a lower bound linear in N on the total backlog required to packetize any fluid policy and a lower bound logarithmic in N on the backlog needed at any output port. Next, we show that the greedy algorithm achieves both lower bounds. The results in this section were proved independently in a recent paper by Adler et al. [15].

A. A Lower Bound on Backlog for the $1 \times N$ Crossbar Switch

We design an adversarial fluid policy for a given on-line, packetizing algorithm by, at each time step, scheduling fluid equally to a subset of the output ports to which the algorithm's packetized policy has never sent.

In addition to keeping track at each time step k of the fluid step $F^{(k)}$, packetized step $P^{(k)}$, and cumulative difference $C^{(k)}$, we keep track of a subset of output ports $S_k \subseteq \{1, 2, ..., N\}$. S_k will include all the ports sent to by the packetized policy up to and including time step k; however, S_k may contain additional output ports. Also,

[3]The positive part of a matrix M is denoted M^+, where $M^+_{ij} := \max\{M_{ij}, 0\}$.

Fluid Policy $F^{(k)}$		Packetized Policy $P^{(k)}$		Cumulative Differences $C^{(k)}$
$\begin{bmatrix} \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \end{bmatrix}$	,	$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \end{bmatrix}$	,	$\begin{bmatrix} -\frac{4}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \end{bmatrix}$
$\begin{bmatrix} 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \end{bmatrix}$	,	$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \end{bmatrix}$	,	$\begin{bmatrix} -\frac{4}{5} & -\frac{11}{20} & \frac{9}{20} & \frac{9}{20} & \frac{9}{20} \end{bmatrix}$
$\begin{bmatrix} 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix}$	,	$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 \end{bmatrix}$	,	$\begin{bmatrix} -\frac{4}{5} & -\frac{11}{20} & \frac{47}{60} & \frac{47}{60} & -\frac{13}{60} \end{bmatrix}$
$\begin{bmatrix} 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \end{bmatrix}$	,	$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix}$	,	$\begin{bmatrix} -\frac{4}{5} & -\frac{11}{20} & \frac{77}{60} & \frac{17}{60} & -\frac{13}{60} \end{bmatrix}$

Fig. 2. An example of an adversarial fluid policy for the 5×1 crossbar switch.

we will maintain for all $k \geq 0$ that

$$S_k \subseteq S_{k+1} \tag{1}$$

and $|S_k| = k$. The $1 \times N$ incidence vector χ^{S_k} has value 1 for all components in S_k and has value 0 elsewhere.

Given an on-line, packetizing algorithm, we iteratively construct the adversarial fluid policy $\{F^{(k)}\}$, for $k \in \{1, 2, ..., N-1\}$.

First, we set $S_0 := \emptyset$.

At each iteration k, we define

$$F^{(k)} := \frac{1}{N-(k-1)}([1, 1, ..., 1] - \chi^{S_{k-1}}).$$

Then we run the packetizing algorithm to determine which output port j it sends a whole packet to at time step k. If $j \notin S_{k-1}$, we set $S_k := S_{k-1} \cup \{j\}$; otherwise, we choose the least $i \geq 1$ not in S_{k-1}, and set $S_k := S_{k-1} \cup \{i\}$.

An example of such an adversarial fluid policy for the 5×1 crossbar switch is given in Figure 2.

Let H_m denote the mth harmonic number; that is, $H_0 := 0$, and for $m \geq 1$, we have $H_m := H_{m-1} + 1/m$. Having constructed $\{F^{(k)}\}_{1 \leq k \leq N-1}$, it is not difficult to prove that for any $m : 1 \leq m \leq N$, just after time step $N-1$, the backlog of the set of m most-backlogged output ports is at least $m(H_N - H_m)$. This is true immediately after iteration $N - m$ of the above construction, since the m output ports not in S_{N-m} each have cumulative difference equal to $H_N - H_m$. The sum of cumulative difference values of this set of m output ports remains lower bounded by $m(H_N - H_m)$ at each subsequent time step, since a total of exactly 1 unit of fluid is scheduled among these m output ports (by (1)) and at most one packet is sent by the packetized policy per time step.

Since we will refer to this summation often in the next section, we define for fixed N, and $m : N \geq m > 0$,

$$g(m) := m(H_N - H_m).$$

It is easy to verify that for $m : N \geq m > 0$,

1) $g(m) \geq 0$,
2) $m(g(m+1) + 1) = (m+1)g(m)$,
3) $\ln N - 1 \leq g(1) \leq \ln N$,
4) $N/e - 2 \leq \max_{m:N \geq m > 0} g(m) \leq N/e$.

We have proved the following theorem:

Theorem 1: For the $1 \times N$ crossbar switch, for every on-line, packetizing algorithm, there exists an adversarial fluid policy with the following property: For any $m : N \geq m > 0$, there exists a set S of entries of $C^{(N-1)}$ with $|S| = m$ such that

$$\sum_{i \in S} C_i^{(N-1)} \geq g(m). \tag{2}$$

In particular, there is a set of output ports with backlog at least $N/e - 2$. Also, there is a single output port with backlog at least $\ln N - 1$. We now turn to a converse, which in light of the above theorem shows optimality of the greedy algorithm for minimizing the worst case backlog over all possible sets of pairs of input/output ports.

B. An Upper Bound on Backlog for the $1 \times N$ Crossbar Switch

We prove the following theorem for the greedy algorithm, which constructs a packetized policy that at time $k+1$ sends a packet to output port $l := \arg\max_i \left(C_i^{(k)} + F_i^{(k+1)}\right)$.[4]

Theorem 2: For the $1 \times N$ crossbar switch and for any fluid policy, the greedy packetizing algorithm produces a packetized policy satisfying the following set of inequalities: For all k and for any non-empty set S of entries of $C^{(k)}$, we have

$$\sum_{i \in S} C_i^{(k)} \leq g(|S|). \tag{3}$$

[4]If for all i, $C_i^{(k)} + F_i^{(k+1)} \leq 0$, then the greedy algorithm doesn't send to any output port at time $k+1$.

This implies the greedy packetizing algorithm is at most $\ln N$-backlogged per output port and at most N/e-backlogged over all output ports.

Proof: We show by induction that the greedy algorithm for determining a packetized policy from a fluid policy maintains (3) at each time step. Let S be any non-empty subset of $\{1, 2, ..., N\}$.

The set of inequalities (3) are true initially since $C^{(0)} = \mathbf{0}$ and for all non-empty S, $g(|S|) \geq 0$.

For the inductive step, assume that (3) holds at time step k. We will show it holds for time step $k+1$ regardless of the fluid policy's choice of $F^{(k+1)}$. We set $P := P^{(k+1)}, C := C^{(k)}$, and $F := F^{(k+1)}$ for clarity of exposition.

By definition, the greedy algorithm chooses the $1 \times N$ matrix P to be all 0's except at entry $l := \arg\max_i (C_i + F_i)$, where has value 1. We consider two cases, depending on whether $l \in S$ or not.

Case 1: $l \in S$
Since $\sum_{i=1}^{N} F_i \leq 1$, we have:

$$\begin{aligned} \left[\sum_{i\in S}(C_i + F_i)\right] - 1 &\leq \sum_{i\in S} C_i \\ &\leq g(|S|). \end{aligned}$$

Since the left hand side equals the sum over output ports $i \in S$ of the cumulative difference at time step $k+1$, this proves (3) in this case.

Case 2: $l \notin S$
We have the following inequalities:

$$\begin{aligned} \frac{|S|+1}{|S|}\left[\sum_{i\in S}(C_i + F_i)\right] &\leq \sum_{i\in\{l\}\cup S}(C_i + F_i) \\ &\leq g(|S|+1) + 1. \end{aligned}$$

The first inequality holds since by the greediness of the algorithm, we have for all $i, C_i + F_i \leq C_l + F_l$. The second inequality holds by the inductive hypothesis and since $\sum_{i=1}^{N} F_i \leq 1$.

Since for any $m : N \geq m > 0$, we have $m(g(m+1)+1) = (m+1)g(m)$, we get from the above inequality that

$$\sum_{i\in S}(C_i + F_i) \leq g(|S|).$$

Since the left hand side equals the sum over output ports $i \in S$ of the cumulative difference at time step $k+1$, this proves (3) in this case.

The induction is complete. ■

IV. Bounds on Backlog for the $N \times N$ Crossbar Switch

A. A Lower Bound on Backlog for the $N \times N$ Crossbar Switch

A corollary of the first theorem in the previous section gives the following bound for the $N \times N$ crossbar switch:

Corollary: For the $N \times N$ crossbar switch, for every on-line, packetizing algorithm, and any input port i, there exists an adversarial fluid policy with the following property: For any $m : N \geq m > 0$, there exists a set S of entries in row i of $C^{(N-1)}$ with $|S| = m$ such that

$$\sum_{j\in S} C_{ij}^{(N-1)} \geq g(m). \tag{4}$$

In particular, there exists a set of pairs of input/output ports in row i with backlog at least $N/e - 2$.

The proof is the natural generalization of the proof for the $1 \times N$ crossbar switch from Section III. We construct an adversarial fluid policy by, at each time step, using only output ports to which the algorithm's packetized policy has never sent from input i.

It is not clear how to generalize the proof of the second theorem in Section III, in which the greedy packetizing algorithm is shown to achieve the lower bound resulting from an adversarial fluid policy, to the $N \times N$ crossbar switch. First, the $1 \times N$ and $N \times N$ crossbar switches differ in that there is no clear choice for what the greedy packetizing algorithm should be for the $N \times N$ crossbar switch. A second obstacle is that in the $N \times N$ crossbar switch, it may be impossible to send a permutation matrix at every packetized step since only packets for which some fluid has arrived can be sent in a packetized step; for example, see Figure 1. In light of this fact, it is not initially clear whether backlog can be kept finite at all for the $N \times N$ crossbar switch. The next section proves that it can.

B. An Upper Bound on Backlog for the $N \times N$ Crossbar Switch

We present an on-line, packetizing algorithm that from any fluid policy, builds a packetized policy that is at most $[(N+1)^2/4-1]$-backlogged per input port for the $N \times N$ crossbar switch.

Algorithm 1: The algorithm builds a packetized policy $\{P^{(k)}\}_{k>0}$ from a given fluid policy $\{F^{(k)}\}_{k>0}$. At each iteration, the algorithm computes $P^{(k+1)}$ based on $C^{(k)}$ and $F^{(k+1)}$. This computation is described below, where we set $P := P^{(k+1)}, C := C^{(k)}$, and $F := F^{(k+1)}$ for clarity of exposition. The algorithm maintains the following invariant for all time steps k:

Invariant 1: For all k, the sum of positive entries in any row or column of $C^{(k)}$ is at most $(N+1)^2/4-1$.

There are three main steps in the packetizing algorithm. First, the algorithm dominates[5] $C+F$ by a matrix B with non-negative entries and with all row sums and column sums equal to exactly $(N+1)^2/4$. Next, it finds a permutation matrix π dominated by the matrix B' which is defined as:

$$B'_{ij} := \begin{cases} 1, & \text{if } B_{ij} \geq 1 \\ 0, & \text{otherwise} \end{cases}$$

Lastly, the packetized step P is set to be the sub-permutation matrix defined as:

$$P_{ij} := \begin{cases} \pi_{ij}, & \text{if } C_{ij}+F_{ij} > 0 \\ 0, & \text{otherwise} \end{cases}$$

We will need the lemmas below in order to prove that Algorithm 1 is well-defined and satisfies Invariant 1. The first lemma, proved by König [16], describes how the first step of Algorithm 1 is computed. For this, as well as combinatorial results cited elsewhere in this paper, we refer the interested reader to Schrijver [17] for further details.

Lemma 1: One can dominate any $N \times N$ doubly sub-stochastic matrix by a doubly stochastic matrix in time $O(N^2)$.

Proof: We define a greedy algorithm below to dominate doubly sub-stochastic matrix D by a doubly stochastic matrix D'. First, compute the row sums and column sums of D. Then, loop once through all N^2 entries of D, increasing each of them until either the corresponding row sum or column sum is 1. We claim that this results in a matrix in which all row sums and column sums are 1. If this were not the case for some row, then every column would have a column sum of 1, implying that every row sum is also 1, a contradiction. A similar contradiction follows if any column sum were not 1. ■

The key, technical lemma of this paper is presented below; it will be used to show that the second step of Algorithm 1 is well-defined:

Lemma 2: For any $w \geq (N+1)^2/4$, for any non-negative valued $N \times N$ matrix D with row sums and column sums equal to w, there exists a permutation π dominated by D.

Proof: Assume the claim were false, that is, that there were some non-negative valued $N \times N$ matrix D with row sums and column sums equal to w such that for any permutation, there is at least one corresponding entry in D with value less than 1. By Hall's Theorem [18], this implies that for some $m : 1 \leq m \leq N$ there is a set R of m rows and a set C of $N-m+1$ columns, such that for any entry (i,j) with $i \in R$ and $j \in C$, $D_{ij} < 1$.

We can thus reorder the rows and columns so that the matrix consists of four blocks: $\begin{bmatrix} D^{(1)} & D^{(2)} \\ D^{(3)} & D^{(4)} \end{bmatrix}$, where each entry of $D^{(1)}$ has value less than 1, and $D^{(1)}$ is of dimension $m \times (N-m+1)$.[6] Now, since each row sum equals w, the sum of entries in block $D^{(2)}$ is strictly greater than $m[w-(N-m+1)]$. Thus, there must be some column among the last $m-1$ with sum strictly greater than $\frac{m}{m-1}[w-(N-m+1)]$. But since for any value of m, $m(N-m+1) \leq (N+1)^2/4 \leq w$, we have:

$$\begin{aligned} \frac{m}{m-1}[w-(N-m+1)] &= \frac{mw-m(N-m+1)}{m-1} \\ &\geq w, \end{aligned}$$

a contradiction, proving the lemma.[7] ■

Lemma 2 has the following corollary:

Corollary: For any $d \in \mathbf{Z}^+$, and any non-negative valued matrix M with row sums and column sums equal to $d+(N+1)^2/4$, there exist permutation matrices $\pi_1, ..., \pi_{d+1}$ such that M dominates $\sum_{i=1}^{d+1} \pi_i$.

Note that we can use the Birkhoff-von Neumann theorem (see e.g. [19]) to immediately obtain a similar, but weaker version of Lemma 2. This follows since by the Birkhoff-von Neumann theorem, every matrix B with non-negative entries and row and column sums equal to $(N-1)^2+1$ can be decomposed into a weighted sum of $(N-1)^2+1$ permutation matrices, where all weights are non-negative, and sum to $(N-1)^2+1$. Since at least one of the weights must be ≥ 1, there exists a permutation matrix π that is dominated by B.

We now prove Algorithm 1 is well-defined and that it satisfies Invariant 1 by induction on the time step k.

Proof: The base case, in which $C^{(0)} = \mathbf{0}$, is clear.

For the inductive step, assume the algorithm is well-defined and satisfies Invariant 1 at all time steps up to and including time step k. Recall that we set $P := P^{(k+1)}, C := C^{(k)}$, and $F := F^{(k+1)}$ for clarity of exposition. We first show that the algorithm is well-defined at time step $k+1$:

By Invariant 1 at time step k (using the inductive hypothesis) and the fact that F is doubly sub-stochastic, we have that all row sums and column sums of $(C+F)^+$

[5]Matrix D' dominates matrix D if for all $i, j, D'_{ij} \geq D_{ij}$.

[6]The dimensions of $D^{(2)}, D^{(3)}, D^{(4)}$ can be deduced from the dimensions of $D^{(1)}$.

[7]In fact, for N even, we can get the slightly better bound that $m(N-m+1) \leq N(N+2)/4$. This implies that for N even, the lemma holds for any $w \geq N(N+2)/4$.

are at most $(N+1)^2/4$. By Lemma 1 we can dominate $C+F$ by a non-negative valued matrix with row sums and column sums equal to exactly $(N+1)^2/4$. Thus, the first step in the algorithm is well-defined.

For the second step in the algorithm, we need to show that there exists a permutation matrix dominated by the matrix B'. By Lemma 2, there exists a permutation matrix π dominated by B. Then π must also be dominated by B'.

It remains to show that the algorithm satisfies Invariant 1 at time step $k+1$:

From the first two steps of the algorithm, we have:

$$C + F \leq B. \tag{5}$$

Subtracting π from both sides, and taking the positive parts of both sides gives:

$$(C + F - \pi)^+ \leq (B - \pi)^+ = B - \pi, \tag{6}$$

where the equality on the right follows because B dominates π.

By the construction of P, the matrix $C+F-P$ differs from $C+F-\pi$ only at entries in which both expressions have non-positive values. This implies

$$(C + F - \pi)^+ = (C + F - P)^+. \tag{7}$$

Therefore, from (6), we have $(C + F - P)^+$ is dominated by the non-negative valued matrix $B-\pi$, with row sums and column sums equal to $(N + 1)^2/4 - 1$. Since $C^{(k+1)} := C + F - P$, this proves Invariant 1 for time step $k + 1$. The induction is complete. ■

We now bound the running time of Algorithm 1. The algorithm requires $O(N^2)$ time to compute B and B'. The time required to find a permutation matrix π dominated by B' is of the same order as the time required to find a perfect matching[8] in an N by N bipartite graph, which is $O(N^{2.5})$ [20]. We show in Section V how to use a fast algorithm for edge-coloring bipartite multigraphs to improve this algorithm, reducing the running time per fluid step to $O(N \log N)$.

V. A Packetizing Algorithm with Running Time $O(N \log N)$ per Fluid Step

We present an on-line, packetizing algorithm for an $N \times N$ input queued, crossbar switch with the following performance guarantees, for any $\epsilon > 0$:

- It runs in time $O(\frac{1}{\epsilon} N \log N)$ per fluid step.
- It uses no speedup.
- It guarantees 100% throughput with a buffer requirement of $(1 + \epsilon)(N + 1)^2/4$ packets per input port.

The main idea behind this algorithm is to process batches of accumulated fluid, producing a list of permutation matrices all at once, instead of just one at a time as in Algorithm 1. This list of permutation matrices is guaranteed to exist by the Corollary to Lemma 2, and can be computed quickly using a fast algorithm for edge-coloring bipartite multigraphs.

The algorithm uses pipelining; after an entire batch of fluid steps has arrived, it computes a corresponding list of permutation matrices (which may take as much time as for yet another batch of fluid steps to arrive), and then schedules this batch of permutation matrices as packetized steps. The batch size is set to be the positive-integer valued function $\alpha(N)$, which will be explicitly defined later in order to obtain the bounds claimed above. Each batch of packetized steps, then, will be scheduled $2\alpha(N)$ time slots after the corresponding batch of fluid steps. It will be convenient to keep track of $C^{(b\alpha(N))}$ minus the sum of packetized steps for time steps $b\alpha(N)+1$ to $(b+2)\alpha(N)$. We refer to this quantity as the pipelined, cumulative difference at the end of batch b.

The algorithm maintains the following invariant on the pipelined, cumulative difference at the end of each batch b:

Invariant 2: For any $b \geq 0$, the matrix $\left(C^{(b\alpha(N))} - \sum_{t=b\alpha(N)+1}^{(b+2)\alpha(N)} P^{(t)}\right)^+$ has row sums and column sums at most $(N+1)^2/4$.

This invariant implies that for any time step t, the row and column sums of $(C^{(t)})^+$ are at most $(N + 1)^2/4 + 3\alpha(N)$.

We first give an outline of the packetizing algorithm, and then its complete description with technical details.

Algorithm 2 Outline: The overall structure of the packetizing algorithm is similar to that of Algorithm 1. In the first step, the algorithm dominates the new batch of fluid plus the pipelined, cumulative difference from the end of the previous batch by a non-negative valued matrix D with row sums and column sums equal to $\alpha(N) + (N + 1)^2/4$.

The goal of the second and third steps is to efficiently find $\alpha(N)$ permutation matrices $\pi_1, \ldots, \pi_{\alpha(N)}$ such that D dominates their sum. In step 2, an integer-valued matrix D' is found that has row sums and column sums equal to $\alpha(N)$ and is dominated by D. By König's Theorem [21], it is possible to decompose D' into a

[8] A perfect matching is a set of vertex-disjoint edges that covers all the vertices.

sum of $\alpha(N)$ permutation matrices. This decomposition is efficiently found in step 3. The complete algorithm including technical details is given below.

Algorithm 2: The algorithm builds a new batch of $\alpha(N)$ packetized steps at each iteration. Initially, the batch count b is set equal to 0.
Also, for $1 \le t \le 2\alpha(N)$, the algorithm sets $P^{(t)}$ to be the all zero matrix.

Given $C^{(b\alpha(N))}$, and new batch of consecutive fluid steps $F^{(b\alpha(N)+1)}, \ldots, F^{((b+1)\alpha(N))}$, the algorithm constructs $P^{((b+2)\alpha(N)+1)}, \ldots, P^{((b+3)\alpha(N))}$ as follows:

1) The algorithm dominates $C^{(b\alpha(N))} - \sum_{k=b\alpha(N)+1}^{(b+2)\alpha(N)} P^{(k)} + \sum_{t=b\alpha(N)+1}^{(b+1)\alpha(N)} F^{(t)}$ by a matrix D with non-negative entries, and row sums and column sums equal to $\alpha(N) + (N+1)^2/4$. The algorithm then rounds the value of each entry of D down to the nearest integer.
2) In this step, the algorithm will construct an integer-valued matrix D' that has row sums and column sums equal to $\alpha(N)$ and that is dominated by D. Let the edge-weighted graph G be defined as follows: First a complete, bipartite graph with $2N$ nodes (N on the left corresponding to the input ports, and N on the right corresponding to the output ports of the switch) is formed. The weight of the edge from node i on the left to node j on the right is set to be D_{ij}. We connect a single, source node s to all the nodes on the left side, with edges of weight $\alpha(N)$. We connect all nodes on the right side to a single sink node t, also with edges of weight $\alpha(N)$. Now, the algorithm finds an integral, maximum flow f from the source to the sink through this graph (see e.g. [19]). Let the $N \times N$ matrix D' be defined such that D'_{ij} equals the flow value from node i on the left to node j on the right. By the Corollary to Lemma 2, the flow out of each node on the left, and the flow into each node on the right equals exactly $\alpha(N)$. Also, note that D dominates D'.
3) D' can be viewed as an $\alpha(N)$-regular bipartite graph (possibly with parallel edges) on $2N$ nodes, where the value of D'_{ij} represents the number of edges with endpoints i and j. We find a minimum edge-coloring of this bipartite graph, which by König's Theorem [21] uses $\alpha(N)$ distinct colors. For each color used, the edges with this color form a perfect matching. Since a perfect matching, here a set of N vertex-disjoint edges, corresponds to a permutation matrix in D', the minimum edge-coloring we found represents a list of $\alpha(N)$ permutation matrices that sum to exactly D'.
4) For $m : 1 \le m \le \alpha(N)$, we set $\pi^{((b+2)\alpha(N)+m)}$ to be the mth permutation matrix from the previous step. Lastly, for $k = (b+2)\alpha(N)+1$ to $(b+3)\alpha(N)$, and $k' := k - 2\alpha(N)$, the packetized step $P^{(k)}$ is set to be the sub-permutation matrix defined as:
$$P_{ij}^{(k)} := \begin{cases} \pi_{ij}^{(k)}, & \text{if } C_{ij}^{(k')} - \sum_{t=k'+1}^{k-1} P_{ij}^{(t)} > 0 \\ 0, & \text{otherwise} \end{cases}$$

The proof that Invariant 2 holds is similar to the proof that Invariant 1 holds from Section IV.

The run-time of one iteration of Algorithm 2 is dominated by steps 2 and 3, in which an integral, maximum flow is found, and a minimum edge-coloring is found, respectively. An integral, maximum flow in a graph with N nodes can be computed in $O(N^3)$ time[9] [23]. For step 3 we can use a simple, efficient, and easy to implement minimum edge-coloring algorithm with running time $O(m \log \Delta)$ on a Δ-regular bipartite graph with m edges [24], [25] for the special case of Δ a power of 2. The run time of the entire algorithm is then $O(N^3 + N\alpha(N)\log\alpha(N))$ to compute a batch of $\alpha(N)$ packetized steps.

We now show how the batch computations in Algorithm 2 can be timed so that the run time per fluid step is $O(N^3/\alpha(N) + N\log\alpha(N))$. The algorithm computes packetized steps $P^{(k)}$ for $k = (b+2)\alpha(N)+1$ to $(b+3)\alpha(N)$ during time steps $(b+1)\alpha(N)+1$ to $(b+2)\alpha(N)$. It can do this since by time step $(b+1)\alpha(N)+1$, it has access to the fluid steps that have arrived before this time step, and it will have already computed the previous batch of packetized steps $P^{(k)}$ for $k = (b+1)\alpha(N)+1$ to $(b+2)\alpha(N)$. Since the batch computation is then spread over $\alpha(N)$ time slots, the run time per fluid step is the run time of one batch divided by $\alpha(N)$.

If we set the batch size $\alpha(N)$ to be $\epsilon N^2/24$ rounded up to the nearest power of 2, an iteration of Algorithm 2 will run in time $O(\frac{1}{\epsilon} N \log N)$. Also, Invariant 2 implies for any time step t, the row sums and column sums of $(C^{(t)})^+$ are at most $(N+1)^2/4 + 3\alpha(N) \le (1+\epsilon)(N+1)^2/4$. Thus, the algorithm will maintain the running time and the bound on backlog per input port claimed at the beginning of this section.

In the next section, we extend Algorithm 1 to give upper bounds on backlog for a more general class of switches than the crossbar switches studied above.

[9] or in time $O(N^{8/3} \log N)$ using an algorithm of Goldberg and Rao [22]

VI. PACKETIZING GENERALIZED FLUID POLICIES

Using the resource allocation model of Tassiulas [6], we can generalize the crossbar switch architecture. The crossbar switch's constraints are that in one time step, at most one packet can leave a single input port, and at most one packet can arrive at a single output port. In general, a switch architecture could differently restrict the set of packets that may be simultaneously transmitted without conflict. Following Tassiulas [6], we call an $N \times M$ matrix with elements $\in \{0, 1\}$ an *activation matrix* if the corresponding set of input-output pairs can simultaneously service packets without conflict on a given, generalized switch. We let S denote the set of activation matrices for a generalized switch. In the special case of the input queued, crossbar switch, S is the set of sub-permutation matrices. In this section, we consider switch architectures constrained by an arbitrary set S of $N \times M$ matrices with elements $\in \{0, 1\}$, such that S is down-monotone[10]. Examples of switch architectures fitting our model include output queued switches and Banyan Switch Networks.

Let $conv\{S\}$ denote the convex hull of S. In this resource allocation model, a *generalized, fluid policy* is a sequence of $N \times M$ matrices from $conv\{S\}$. A *generalized, packetized policy* is a sequence of $N \times M$ matrices from S corresponding to a sequence of switch uses. Let $m(S)$ denote the maximum row sum of any matrix in S. In the case of an input queued, crossbar switch, we have $m(S) = 1$.

We show that Algorithm 1 can be modified to give an on-line, generalized, packetizing algorithm for building, from any generalized, fluid policy, a generalized, packetized policy that gets at most $(MN)m(S)$-backlogged per input port for the $N \times M$ generalized switch.

We rely on Caratheodory's theorem, which gives for any $F \in conv\{S\}$, the existence of a decomposition into a convex combination of at most $MN + 1$ elements of S [19]. The main difference between the algorithm in this section and the previous algorithms is that we don't know of an efficient algorithm for finding an element of such a decomposition with large weight. The packetizing algorithm below maintains the following invariant for all time steps k:

Invariant 3: For all k, $(C^{(k)})^+ \in (MN)conv\{S\}$

Algorithm 3: Given a generalized, fluid policy, this packetizing algorithm computes the generalized, packetized policy as follows:

First, it calculates $C^{(k-1)} := \sum_{i=1}^{k-1}(F^{(i)} - P^{(i)})$, and then decomposes $(F^{(k)} + C^{(k-1)})^+$ into a non-negative linear combination of at most $MN + 1$ matrices from S, in which the weights sum to $MN + 1$. At least one matrix in the decomposition must now have weight at least 1. Set π to be one such matrix. Define

$$P_{ij}^{(k)} := \begin{cases} \pi_{ij}, & \text{if } C_{ij}^{(k-1)} + F_{ij}^{k} > 0 \\ 0, & \text{otherwise} \end{cases}$$

By a similar argument as that which followed Algorithm 1, we have that Algorithm 3 maintains the invariant $(C^{(k)})^+ \in (MN)conv\{S\}$ for all time steps k. In other words, Algorithm 3 gets at most $MNm(S)$-backlogged per input port.

[10]We call S down-monotone if for any F, F', both $N \times M$ matrices with elements in $\in \{0, 1\}$ s.t. F dominates F' and $F \in S$, then we have $F' \in S$.

TABLE I
SUMMARY OF BOUNDS ON BACKLOG FOR THE $N \times N$ CROSSBAR SWITCH

	Lower Bound	Upper Bound
Backlog per Input Port	$N/e - 2$	$(N+1)^2/4$
Backlog per Virtual Output Queue	$\ln N - 1$	$(N+1)^2/4$

VII. CONCLUSION

The bounds proved in this paper for the $N \times N$ crossbar switch are summarized in Table I above. A lower bound of $f(N)$ means that for any on-line (deterministic), packetizing algorithm, there exists a fluid policy that when used as input to the packetizing algorithm will cause backlog at least $f(N)$. An upper bound of $g(N)$ means that there exists an on-line, packetizing algorithm that is at most $g(N)$-backlogged when run on any fluid policy.

As can be seen in Table I, gaps remain between the lower and upper bounds proved for backlog. In particular, the gap for the buffer requirement of a virtual output queue, which corresponds to the maximum entry in the cumulative difference matrix $C^{(k)}$, is quite large. Narrowing these gaps is a direction for future research.

Other directions for future work include the following:

- Finding tradeoffs between buffer requirements and run-time of the packetizing algorithms,
- Bounding backlog with speedup in the crossbar switch,
- Bounding backlog for specific switch architectures other than the crossbar switch.

VIII. ACKNOWLEDGMENTS

We would like to thank Thomas Erlebach for pointing out the recent paper of Adler et al. [15] in which the

results of Section III were independently proved.

REFERENCES

[1] Abhay K. Parekh and R.G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single-node case," *ACM/IEEE Transactions on Networking*, vol. 1, 3, June 1993.

[2] Abhay K. Parekh and R.G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The multiple node case," *ACM/IEEE Transactions on Networking*, vol. 2, 2, April 1994.

[3] R. Cruz, "A calculus for network delay.I. network elements in isolation.," *IEEE Transactions on Information Theory*, vol. 37, 1, pp. 114–131, January 1991.

[4] R. Cruz, "A calculus for network delay. II. network analysis.," *IEEE Transactions on Information Theory*, vol. 37, 1, pp. 132–141, January 1991.

[5] N. McKeown, V. Anantharam, and J.C. Walrand, "Achieving 100% throughput in an input-queued switch," in *INFOCOM (1)*, 1996, pp. 296–302.

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

[7] J. G. Dai and B. Prabhakar, "The throughput of data switches with and without speedup," in *INFOCOM (2)*, 2000, pp. 556–564.

[8] J. C. R. Bennett and H. Zhang, "WF^2Q: worst-case fair weighted fair queueing," in *INFOCOM (1)*, 1996, pp. 120–128.

[9] A. Stamoulis and J. Liebherr, "S^2GPS: slow-start generalized processor sharing," in *Proc. of Workshop on Resource Allocation Problems in Multimedia Systems (held in conjunction with IEEE Real-Time Systems Symposium.)*, University of North Carolina at Chapel Hill, December 1996.

[10] N.G. Duffield, T. V. Lakshman, and D. Stiliadis, "On adaptive bandwidth sharing with rate guarantees," in *INFOCOM (3)*, 1998, pp. 1122–1130.

[11] A. Stamoulis and G. B. Giannakis, "Deterministic time-varying packet fair queueing for integrated services networks," in *IEEE Global Telecommunications Conference*, 2000, vol. 1, pp. 621–625.

[12] C. Chang, W. Chen, and H. Huang, "Birkhoff-von Neumann input buffered crossbar switches," in *INFOCOM (3)*, 2000, pp. 1614–1623.

[13] V. Tabatabaee, L. Georgiadis, and L. Tassiulas, "QoS provisioning and tracking fluid policies in input queueing switches," in *INFOCOM*, 2000, pp. 1624–1633.

[14] M.A. Bonuccelli and M.C. Clo, "EDD algorithm performance guarantee for periodic hard-real-time scheduling in distributed systems," *IEEE IPPS/SPDP*, pp. 668–677, April 1999.

[15] M. Adler, P. Berenbrink, T. Friedetzky, L. A. Goldberg, P.W. Goldberg, and M. Paterson, "A proportionate fair scheduling rule with good worst-case performance," in *ACM Symposium on Parallel Algorithms and Architectures*, San Diego, CA, USA, June 2003.

[16] D. König, "Über trennende knotenpunkte in graphen (nebst anwendungen auf determinanten und matrizen)," *Acta Litterarum ac Scientiarum Regiae Universitatis Hungaricae Francisco-Josephinae, Sectio Scientiarum Mathematicarum [Szeged]*, 6, pp. 155–179, 1932-34.

[17] A. Schrijver, *Combinatorial Optimization*, Springer, New York, 2003.

[18] P. Hall, "On representatives of subsets," *The Journal of the London Mathematical Society*, 10, pp. 26–30, 1935.

[19] V. Chvatal, *Linear Programming*, W.H. Freeman, San Francisco, CA, 1983.

[20] R.M. Karp J. Hopcroft, "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs," *SIAM Journal on Computing*, 2, pp. 225–231, 1973.

[21] D. König, "Graphok és alkalmazásuk a determinánsok és a halmazok elméletére [hungarian]," *Mathematikai és Természettudományi Értesitö*, 34, pp. 104–119, 1916.

[22] S. Rao A.V. Goldberg, "Beyond the flow decomposition barrier," *Journal of the ACM*, 45, pp. 783–797, 1998.

[23] R. Rivest T. Cormen, C. Leiserson, *Introduction to Algorithms*, The MIT Press, Cambridge, MA, 1994.

[24] A. Schrijver, "Bipartite edge-colouring in $O(\Delta m)$ time," *SIAM Journal on Computing*, vol. 28, pp. 841–846, 1999.

[25] H.N. Gabow, "Using euler partitions to edge color bipartite multigraphs," *International Journal of Computer and Information Sciences*, 5, pp. 345–355, 1976.

[26] T. Inukai, "An efficient SS/TDMA time slot assignment algorithm," *Communications, IEEE Transactions*, vol. 27, 10, pp. 1449–1455, 1979.

[27] J. von Neumann, *Contributions to the Theory of Games*, vol. 2, Princeton University Press, Princeton, New Jersey, 1953.

[28] A. Charny, "Providing QoS guarantees in input-buffered crossbar switches with speedup, Ph.D. dissertation, MIT," August 1998.

[29] C. E. Koksal, "Providing quality of service over high speed electronic and optical switches. Ph.D. dissertation, MIT," September 2002.

[30] P. Gupta, "Scheduling in input queued switches: A survey," unpublished manuscript.

[31] E. L. Hahne, A. Kesselman, and Y. Mansour, "Competitive buffer management for shared-memory switches," in *ACM Press Proceedings of the thirteenth annual ACM symposium on Parallel algorithms and architectures*, 2001, pp. 53–58.

[32] M.A. Bonuccelli and S. Pelagatti, "Optimal on demand packet scheduling in single-hop multichannel communication systems," *IEEE IPDPS*, May 2000.

[33] B. Prabhakar and N. McKeown, "On the speedup required for combined input and output queued switching," Tech. Rep. CSL-TR-97-738, Stanford University, CA, 1997.

[34] A. Borodin, J. Kleinberg, P. Raghavan, M.Sudan, and D. P. Williamson, "Adversarial queuing theory," *Journal of the ACM*, vol. 48, 1, pp. 13–38, 2001.

[35] M. J. Girone, "Tracking switch fluid policies: Bounding lookahead," Master's project, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, February 2002.