Approximating Fluid Schedules in Crossbar Packet-Switches and Banyan Networks

Michael Rosenblum, *Member, IEEE*, Constantine Caramanis, *Member, IEEE*, Michel X. Goemans, and Vahid Tarokh, *Member, IEEE*

***Abstract*—We consider a problem motivated by the desire to provide flexible, rate-based, quality of service guarantees for packets sent over input queued switches and switch networks. Our focus is solving a type of online traffic scheduling problem, whose input at each time step is a set of desired traffic rates through the switch network. 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 transmissions in which only whole packets are sent. We prove worst-case bounds on the additional buffer use, which we call backlog, that results from using such an approximation.**

We first consider the $N \times N$, input queued, crossbar switch. Our main result is an online packet-scheduling algorithm using no speedup that guarantees backlog at most $(N+1)^2/4$ packets at each input port and each output port. Upper bounds on worst-case backlog have been proved for the case of constant fluid schedules, such as the $N^2 - 2N + 2$ bound of Chang, Chen, and Huang (INFOCOM, 2000). Our main result for the crossbar switch is the first, to our knowledge, to bound backlog in terms of switch size N for arbitrary, time-varying fluid schedules, without using speedup.

Our main result for Banyan networks is an exact characterization of the speedup required to maintain bounded backlog, in terms of polytopes derived from the network topology.

***Index Terms*—Combinatorics, graph theory, network calculus, packet-switching, scheduling.**

Manuscript received March 21, 2005; revised October 6, 2005; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor F. Neri. This material is based upon research supported in part by the National Science Foundation under the Alan T. Waterman Award, Grant No. CCR-0139398, under Contracts ITR-0121495 and CCR-0098018, and under a National Science Foundation Graduate Fellowship. 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. Earlier versions of parts of this work were published in the Proceedings of IEEE INFOCOM 2004, Hong Kong, and in the Proceedings of the 38th Annual Conference on Information Sciences and Systems (CISS), Princeton, NJ, 2004.

M. Rosenblum was with the Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139 USA. He is now with the Electrical Engineering and Computer Science Department, University of California, Berkeley, CA 94720 USA (e-mail: mrosen@cs.berkeley.edu).

C. Caramanis is with the Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX 78712 USA (e-mail: cmcaram@ece.utexas.edu).

M. X. Goemans is with the Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139 USA (e-mail: goemans@math.mit.edu).

V. Tarokh is with the Division of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138 USA (e-mail: vahid@deas.harvard.edu).

Digital Object Identifier 10.1109/TNET.2006.886320

I. Introduction

In our approach to packet-scheduling, 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 (as in [1]–[15]). 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.

We define a metric, called *backlog*, that measures the gap in cumulative service between the fluid policy and the packetized policy. This metric is similar to those used in [5], [10], [11]. Our bounds on backlog depend on the *speedup* used by a packet-scheduling algorithm, that is, the ratio of the rate at which the packetized policy sends packets to the rate at which the fluid policy sends fractional packets. Our goal is to find online, packet-scheduling algorithms using the minimum possible speedup that guarantee bounded backlog for any fluid policy.

Other approaches to scheduling are also possible. Much analysis has been done in models where the input traffic is assumed to have certain statistical properties (e.g., [16]–[19]). In such models, it is often shown that the queue lengths, considered as a stochastic process, converge to a limiting distribution with finite expectation. The bounds we obtain, however, are more robust in that the arrival process is not assumed to have any statistical properties; we treat the fluid policy as adversarial, and derive worst-case guarantees on backlog. Other works analyzing switch scheduling from an adversarial standpoint include [1]–[7], [9]–[14], [20]–[24].

Our bounds on backlog are not asymptotic; they apply to all switch sizes and to time increments of any finite duration. Furthermore, our bounds not only apply to constant fluid policies (that is fluid policies that schedule the same set of fractional packets at each time step), but apply to arbitrary, time-varying fluid policies. Our upper bounds on backlog hold when packet-scheduling algorithms must decide which packets to transmit at each time step with no knowledge of the future fluid schedule.

Kam and Siu [10], using a traffic model equivalent to ours, give a packet-scheduling algorithm for the input queued, crossbar switch using speedup 2 that guarantees bounded backlog for any fluid policy. Their proof technique does not extend to the case of no speedup; they underscore "the unavailability of combinatorial proof techniques for our no-speedup scenario." [10]. Our main result for the input queued, crossbar switch is a combinatorial proof that worst-case backlog can be kept bounded using no speedup. To our knowledge, this is the first packet-scheduling algorithm using no speedup that has been shown to maintain bounded backlog for arbitrary, time-varying fluid policies on the input queued, crossbar switch.

After analyzing the single, input queued, crossbar switch, we turn to a class of multistage switch networks called Banyan networks. Banyan networks have been studied extensively in the literature due to their parallel capacity, modularity, expandability, and because they lend themselves to efficient implementation (see for instance, [25], [26], and references therein). We first prove that when no speedup is used, bounded backlog results such as those we give for the crossbar switch do not exist for arbitrary switch networks or even for 4×4 Banyan networks. However, if the packet-scheduling algorithm is allowed to use enough speedup, it can maintain bounded backlog. We prove that if speedup s is sufficient to maintain bounded backlog for any constant fluid policy, then speedup s is also sufficient to maintain bounded backlog for any time-varying fluid policy. For the $N \times N$ Banyan network, we give an exact characterization of the necessary and sufficient speedup to maintain bounded backlog for any fluid policy, in terms of polytopes derived from the topology of Banyan networks. Using this characterization, we calculate this necessary and sufficient speedup for 4×4 and 8×8 Banyan networks. We then use it to compute upper bounds on this necessary and sufficient speedup for the $N \times N$ Banyan network, and give a polynomial-time packet-scheduling algorithm that guarantees these bounds.

The layout of this work is as follows. In Section II, we present results from related work. We specify the traffic model in Section III. In Section IV, worst-case backlog for the $N \times N$, input queued, crossbar switch is analyzed. We turn our attention to Banyan networks starting with Section V, which gives a summary of our results for these networks. In Section VI, we define and discuss the structure of Banyan networks. In Section VII, we prove that even for 4×4 Banyan networks *for the simple case of a constant fluid policy*, it is not possible to maintain bounded backlog using no speedup. This motivates our analyzing the necessary and sufficient speedup for maintaining bounded backlog in Sections VIII–X. Section XI summarizes our results and gives directions for future research.

II. Related Work

A number of authors have worked on the problem of approximating fluid schedules for the $N \times N$, input queued, crossbar switch with virtual output queueing (which is defined in the next section). Chang, Chen, and Huang [1] present a packet-scheduling algorithm that guarantees backlog at most $N^2 - 2N + 2$ for any constant fluid policy. This algorithm is based on a Birkhoff–von Neumann decomposition of the rate matrix of the constant fluid policy; the decomposition is a weighted sum of permutation matrices,[1] each representing a set of packets that can be simultaneously transmitted in one time step. The algorithm schedules each such permutation matrix with frequency according to its weight in the decomposition. The algorithm requires initial run-time $O(N^{4.5})$ to compute the decomposition, and online run-time $O(\log N)$ to determine which element of the decomposition to schedule at each time step, for the $N \times N$, input queued, crossbar switch. In contrast, our upper bounds on worst-case backlog given in Section IV-B are tighter for $N \geq 3$, and apply to the more general case of time-varying fluid policies for which the packet scheduling algorithm only knows the fluid policy up to the current time step; that is, we assume the algorithm has no knowledge of the future desired traffic rates in deciding which packets to schedule at each time step. In this case, it is not possible to compute a schedule for all time steps in advance, and so our packet-scheduling algorithm in Section IV-B does most of its work online; the online run-time of our algorithm is $O(N^{2.5})$ to compute which packets are sent at each time step. In [27], we give a modified version of this algorithm with online run-time $O(N^2)$. This is, up to a constant factor, the same as the time required to read all components of a (time-varying) fluid matrix.

If a packet-scheduling algorithm, given any fluid policy as input, outputs a packetized policy with backlog less than 1 at all time steps, we say the algorithm *tracks*, as in [15]. Charny [5] gives a simple packet-scheduling algorithm using speedup 6 that tracks any constant fluid policy.[2] In Section IV, we discuss how worst-case backlog for time-varying fluid policies can be significantly greater than that for constant fluid policies.

Tabatabaee, Georgiadis, and Tassiulas [15] consider the problem of tracking arbitrary fluid policies on the $N \times N$, input queued, crossbar switch. They attempt to characterize for which N there exist packet-scheduling algorithms that track. They prove that any fluid policy for the 2×2, input queued, crossbar switch can be tracked, and propose several heuristics for approximating fluid policies by packetized policies on larger switches. Bonuccelli and Clo [4] construct a constant fluid policy for the 4×4, input queued, crossbar switch that cannot be tracked. This untrackable fluid policy can be extended to larger switch sizes.

Kam and Siu [10] provide bounds on worst-case backlog for time-varying fluid policies on the $N \times N$, input queued, crossbar switch, when speedup at least 2 is used. They formulate a credit-based system, which is equivalent to the model used here, and in which each input port, output port pair receives a fractional credit (which corresponds to fluid in our model) at each time step based on a (possibly time-varying) service contract. They present an algorithm for determining which packets to send based on outstanding credits (which correspond to backlog in our model); their algorithm is based on finding a stable marriage matching. They show that outstanding credit can be kept bounded in the worst-case when speedup at least 2 is used; as noted above, their proof technique does not extend to the case of no speedup.

Using a credit-based model similar to that used by Kam and Siu [10], Koksal [11] bounds backlog on the $N \times N$, input queued, crossbar switch (called "service lag" in his work) when speedup is strictly greater than 1; these upper bounds tend to infinity as speedup approaches 1.

III. Traffic Model and Definitions

A. Transmission Constraints

We define the transmission constraints for the input queued, crossbar switch and for Banyan networks. All packets are assumed to have the same size. Time is considered discrete, and

[1]A *permutation matrix* is an $N \times N$, {0,1}-valued matrix with a single 1 in each row and in each column.

[2]Due to a small difference in the model used by Charny and that used here, her result holds in our model using constant speedup slightly larger than 6.

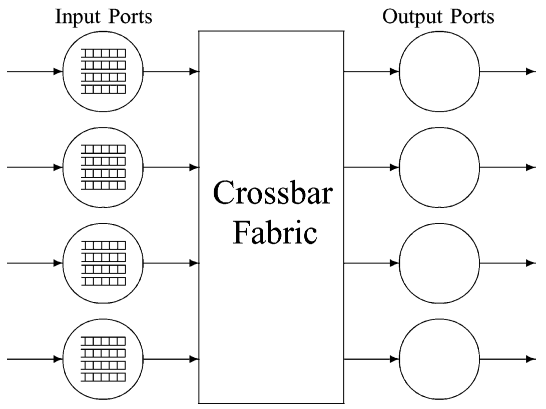

Fig. 1. A 4 × 4, input queued, crossbar switch. Each input port has 4 virtual output queues, one corresponding to each output port.

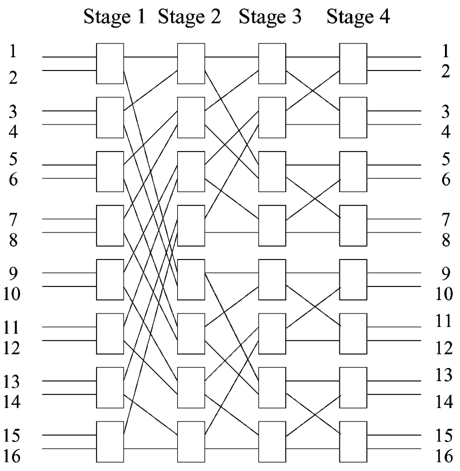

Fig. 2. A 16 × 16 Banyan network.

is normalized so that the data rate (capacity) of any input or output port is one packet per time step. On the $N \times M$, input queued, crossbar switch, one can 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. Virtual output queueing is used to avoid head-of-line blocking; that is, a packet arriving at any input port is placed in one of the M separate queues at that input port, depending on the packet's destination output port (see [15] for more details). Fig. 1 is a diagram of a 4 × 4, input queued, crossbar switch with virtual output queueing. We refer to input queued, crossbar switches simply as *crossbar switches*. In this paper, the only type of crossbar switch we analyze is $N \times N$; however, we use a result for $N \times 1$ switches in Section IV in proving a lower bound on backlog for the $N \times N$ crossbar switch.

A Banyan network is a set of *switch elements*, that is, 2×2 crossbar switches, interconnected by links, with a structure defined in Section VI. Banyan networks are *layered networks*, that is, the set of switch elements in a Banyan network can be partitioned into *stages* $S_1, S_2, \ldots, S_m$ such that for $h < m$, any outgoing link from a switch element in stage S_h connects to a switch element in stage S_{h+1}. Incoming links to S_1 are called *input ports* and outgoing links from S_m are called *output ports*. Fig. 2 depicts a 16×16 Banyan network, which has four stages.

One property of Banyan networks is that each input port, output port pair (which we simply refer to as an *input*, *output pair*), is connected by a unique path through the network [25]. We refer to this as the *unique-path property*.[3] We consider Banyan networks with virtual output queueing at each input port, but with no queueing between stages S_h and S_{h+1} for $h \geq 1$. Each link has unit capacity. Since we do not allow packets to be dropped, if input port i is transmitting a packet to output port j, then any input, output pair (k, l) whose (unique) path shares at least one link with the path from i to j is blocked from transmitting a packet at the same time.

B. Fluid Policies, Packetized Policies, Backlog, and Speedup

We now define fluid policies, packetized policies, backlog, and speedup for the $N \times M$ crossbar switch and for Banyan networks. A fluid policy represents the ideal, packet-scheduling behavior.

A *fluid policy* for the $N \times M$ crossbar switch or for the $N \times N$ Banyan network is a sequence of fractional packet transmissions in which the sum traversing each link is at most one at each time step. It is represented by a sequence of non-negative-valued, $N \times M$, *fluid matrices* $\{F^{(t)}\}_{t>0}$, where $F_{ij}^{(t)}$ represents the fraction of a packet sent from input port i at time step t with output port j as its destination. For the crossbar switch, each fluid matrix must satisfy the constraint that each row sum (corresponding to the total fluid using each input port) and each column sum (corresponding to the total fluid using each output port) is at most 1;[4] an example is given in Fig. 3. This constraint is equivalent to the no overbooking constraint in [1] and to the constraint defining feasible rates in [5]. In contrast to a single crossbar switch, for a Banyan network each fluid matrix must satisfy a stricter set of constraints due to the potential for internal packet collisions, as we discuss in Section VI-B below. In general, we call a non-negative-valued, $N \times M$ matrix a *valid fluid matrix* if for the corresponding, fractional packet transmissions, the sum traversing each link is at most one. Note that by definition a fluid policy is represented by a sequence of valid fluid matrices. We next define a packetized policy, which should approximate a given fluid policy.

A *packetized policy* for the $N \times M$ crossbar switch or for the $N \times N$ Banyan network is a sequence of whole packet transmissions in which at most one packet traverses each link at each time step. It is represented by a sequence of {0,1}-valued, $N \times M$, *packetized matrices* $\{P^{(t)}\}_{t>0}$, where $P_{ij}^{(t)}$ is 1 if a packet is transmitted from input port i at time step t with output port j as its destination. For the crossbar switch, each packetized matrix must satisfy the constraint that there is at most a single entry with value 1 in each row and in each column; a {0,1}-valued matrix satisfying this constraint is called a *sub-permutation matrix*.

[3]The unique-path property can be proven by induction on the number of stages in the Banyan network, using the recursive structure given in Section VI.

[4]A non-negative-valued matrix is called *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 called *doubly stochastic*.

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

Fig. 3. Three time steps of a fluid policy (first column), packetized policy (second column), and their cumulative differences (third column). 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. Note that any packet-scheduling algorithm, given the fluid steps above and having already set $P^{(1)}$ and $P(2)$ as above, cannot set $P^{(3)}$ to be a (full) permutation matrix. This follows from the requirement that a packet-scheduling algorithm can only send a packet from input port i to output port j at time t if $C_{ij}^{(t-1)} + F_{ij}^{(t)} > 0$.

In general, we call a {0,1}-valued, $N \times M$ matrix a *valid packetized matrix* if for the corresponding packet transmissions, at most one packet traverses each link.

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 t, and the cumulative number of whole packets sent by the packetized policy up to and including time t. This information is stored in the $N \times M$, *cumulative difference matrix* $C^{(t)}$, for $t \geq 0$. In particular, $C^{(0)} := \mathbf{0}$, the all zero matrix, and $C^{(t)} := \sum_{h=1}^{t}(F^{(h)} - P^{(h)})$ for $t \geq 1$. For time step t, and for a set of input, output pairs, we define their *backlog* to be the sum of corresponding entries in $(C^{(t)})^{+}$.[5]

We define a *packet-scheduling algorithm* to be a deterministic, online algorithm that at each time step t, given fluid matrices $\{F^{(1)}, \ldots, F^{(t)}\}$, outputs a packetized matrix $P^{(t)}$. We require that for a packet-scheduling algorithm to send a packet from input port i at time t with output port j as its destination (that is, for it to set $P_{ij}^{(t)} = 1$), we must have $C_{ij}^{(t-1)} + F_{ij}^{(t)} > 0$. This ensures that all entries of the cumulative difference matrix $C^{(t)}$ are greater than -1, so that for each input, output pair the packetized policy never gets more than one packet ahead of the fluid policy.[6]

We say a packet-scheduling algorithm maintains backlog at most b per input port if in each row of $(C^{(t)})^{+}$, the sum of entries is at most b at each time step t. Similarly, we say a packet-scheduling algorithm maintains backlog at most b per output port if in each column of $(C^{(t)})^{+}$, the sum of entries is at most b at each time step t.

In general, a scheduler having speedup s means that packets can be sent across the switch fabric s times as fast as the line rates at the input ports, as in [5], [10], [11], [16]. In our model, we let speedup represent the ratio of the rate at which the packetized policy is allowed to send packets to the rate at which the fluid policy is allowed to send fractional packets. We model speedup $s \geq 1$ by requiring that in a single time step, the sum of total fluid traversing each link can be at most $1/s$; the constraint on the packetized policy that at most one packet can traverse each link per time step remains unchanged. We say that an algorithm uses *no speedup* if $s = 1$. Intuitively, speedup s means that the desired rates reflected in the fluid schedule, which the packetized policy should emulate, use at most $1/s$ of any link's capacity; thus, with speedup, it is easier for the packet-scheduler to keep backlog bounded. We show in Section VII that even for some simple network topologies, no packet-scheduling algorithm can maintain bounded backlog for all fluid policies without using speedup.

We say speedup s is sufficient for maintaining bounded backlog if there exists a packet-scheduling algorithm using speedup s that maintains bounded backlog for all fluid policies. Similarly, we say speedup s is necessary for maintaining bounded backlog if every packet-scheduling algorithm that maintains bounded backlog for every fluid policy uses speedup at least s.

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

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

We prove below that no packet-scheduling algorithm can maintain backlog at most $(N+1)/e - 2$ per input port (or per output port) for every fluid policy, without using speedup; similarly, no packet-scheduling algorithm can maintain backlog at most $\ln(N+1) - 1$ for each input, output pair, for every fluid policy, without using speedup. We use a construction given independently by Adler *et al.* [2] and Rosenblum *et al.* [28] for the $N \times 1$ switch to prove these results.

Let H_m denote the mth harmonic number; that is, $H_0 := 0$, and for $m \geq 1$, we have $H_m := \sum_{j=1}^{m} 1/j$. In [2] and [28], for

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

[6]See Bennett and Zhang [3] for a discussion of the importance of this constraint in the context of Generalized Processor Sharing.

any packet-scheduling algorithm on the $N \times 1$ crossbar switch using no speedup, a construction is given of a time-varying fluid policy such that for any $m : 1 \leq m \leq N$, there exists a set of m input ports that by time step N has backlog at least $m(H_N - H_m)$. This latter term can be approximated using the following fact, proved in Appendix A of [27]: For all $N \geq 1$,

1) $(N+1)/e - 2 < \max_{m:1 \leq m \leq N} m(H_N - H_m) < N/e$.
2) $\ln(N+1) - 1 < H_N - 1 \leq \ln N$.

Thus, in the construction from [2], [28], by time step N the backlog of the set of all input ports is more than $(N+1)/e - 2$; also, at time step N there is an input port with backlog more than $\ln(N+1) - 1$.

We can adapt this construction to the $N \times N$ crossbar switch, by treating a particular column of each $N \times N$ fluid matrix as a fluid policy on the $N \times 1$ crossbar switch, and by similarly treating this column of each packetized matrix. This is possible since an $N \times N$ matrix is a valid fluid matrix for the $N \times N$ crossbar switch if and only if each column is a valid fluid matrix for the $N \times 1$ crossbar switch and each row is a valid fluid matrix for the $1 \times N$ crossbar switch; the analogous statement holds for valid packetized matrices. Translating the bounds on backlog from this adapted construction gives the theorem below.

Theorem 1: For the $N \times N$ crossbar switch, for every packet-scheduling algorithm using no speedup, for any output port j, one can construct a time-varying fluid policy such that by time step N, the backlog of output port j is more than $(N+1)/e - 2$; also, at time step N, for some input port i, the pair (i, j) has backlog more than $\ln(N+1) - 1$.

Worst-case backlog for time-varying fluid policies can be significantly greater than that for constant fluid policies. An extension of the theorem above (which follows from Theorem 2 in [27]) is that for any packet-scheduling algorithm using speedup $s \geq 1$ on the $N \times N$ crossbar switch, there exists a time-varying fluid policy causing some input, output pair to have backlog more than $(1/s)(\ln(N+1) - 1)$. This is in stark contrast with Charny's result [5], in which a simple packet-scheduling algorithm using speedup 6 is shown to track (that is, maintain backlog less than 1 for each input, output pair), given any constant fluid policy on the $N \times N$ crossbar switch.

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

Our main result for crossbar switches is the following theorem.

Theorem 2: Packet-scheduling Algorithm 1 below, given any time-varying, fluid policy, builds a packetized policy that maintains backlog at most $[(N+1)^2/4 - 1]$ per input port and per output port for the $N \times N$ crossbar switch. The algorithm uses no speedup.

Algorithm 1: The algorithm builds a packetized policy $\{P^{(t)}\}_{t>0}$ from a given fluid policy $\{F^{(t)}\}_{t>0}$. At each time step t the algorithm has access to fluid matrices $F^{(1)}, \ldots, F^{(t)}$ and must output packetized matrix $P^{(t)}$. Below we describe iteration $t+1$, for $t > 0$, in which the algorithm computes packetized matrix $P^{(t+1)}$ based on $C^{(t)}$ and $F^{(t+1)}$. We set $P := P^{(t+1)}$, $C := C^{(t)}$, and $F := F^{(t+1)}$ for clarity of exposition. The algorithm maintains the following invariant for all time steps t:

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

There are three main steps in the packet-scheduling algorithm. First, the algorithm dominates[7] $C + F$ 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 matrix P is set to be the sub-permutation matrix defined as

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

The lemma below shows how the first step of the algorithm is computed.

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

Proof: The above lemma follows since for any given doubly sub-stochastic matrix that is not doubly stochastic, there must be a row and a column with sums strictly less than 1. One can then augment the entry in such a row and column until either the row sum or the column sum equals one. The process can be repeated (at most $2N$ times) until one has a doubly stochastic matrix. ■

We defer the proof that Algorithm 1 is well-defined and satisfies Invariant 1, which implies Theorem 2, to Appendix I. The proof relies on a lemma, which we prove and discuss here, since it is the main combinatorial result underpinning Theorem 2.

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

The above statement is tight in that for N odd, for any non-negative $w < (N+1)^2/4$ (and for N even, for any non-negative $w < N(N+2)/4$), there exists a non-negative-valued, $N \times N$ matrix with row sums and column sums equal to w that does not dominate any permutation matrix.

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. Define the bipartite graph $G := (V_1, V_2, E)$ in which V_1 is the set of rows of D, V_2 is the set of columns of D, and the set of edges is defined as $E := \{(i,j) \in V_1 \times V_2 : D_{ij} \geq 1\}$. Our assumption means that there are no perfect matchings[8] in G. Thus, by Hall's Matching Theorem[9] [29], 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 $j \in C$, $D_{ij} < 1$.

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

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

[9]Consider a bipartite graph $G = (V_1, V_2, E)$ with $|V_1| = |V_2|$. For a subset of nodes $V \subseteq V_1$, let $N(V)$ denote the neighborhood of V, that is, the subset of nodes in V_2 that are adjacent to at least one node in V. Hall's Matching Theorem states that a perfect matching exists in G if and only if for each subset $V \subseteq V_1$, we have $|V| \leq |N(V)|$.

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 $m \times (N - m + 1)$.[10] 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 $m[w - (N - m + 1)]/(m - 1)$. But since for any value of m, $m(N - m + 1) \leq (N + 1)^2/4 \leq w$, the sum of entries in such a column is strictly greater than

$$\frac{m}{m-1}\left[w - (N - m + 1)\right] = \frac{mw - m(N - m + 1)}{m - 1} \geq w$$

a contradiction, proving the lemma. Note that 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 $w \geq N(N + 2)/4$.

We now prove the first statement of the lemma is tight in the sense described above. For N odd, let $m = (N + 1)/2$, and define the matrix D with block structure as above and with each entry in $D^{(1)}$ having value $4w/(N + 1)^2 < 1$, each entry in $D^{(2)}$ and $D^{(3)}$ having value $2w/(N+1)$, and each entry in $D^{(4)}$ having value 0. Since any permutation matrix must have value 1 at some entry in the block $D^{(1)}$, the matrix D does not dominate any permutation matrix. For N even, instead let $m = (N+2)/2$, and define the matrix D with block structure as above and with each entry in $D^{(1)}$ having value $4w/(N(N + 2)) < 1$, each entry in $D^{(2)}$ having value $2w/(N + 2)$, each entry in $D^{(3)}$ having value $2w/N$, and each entry in $D^{(4)}$ having value 0. Again, since any permutation matrix must have value 1 at some entry in the block $D^{(1)}$, the matrix D does not dominate any permutation matrix. ∎

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 $\pi_1, \ldots, \pi_{d+1}$ such that M dominates $\sum_{i=1}^{d+1} \pi_i$.

Note that we can use the Birkhoff–von Neumann theorem (see e.g., [30]) to immediately obtain a similar, but weaker version of Lemma 2. This follows since by the Birkhoff–von Neumann theorem, every $N \times N$ matrix D with non-negative entries and row and column sums equal to $N^2 - 2N + 2$ can be decomposed into a weighted sum of $N^2 - 2N + 2$ permutation matrices, where all weights are non-negative, and sum to $N^2 - 2N + 2$. Since at least one of the weights must be ≥ 1, there exists a permutation matrix π that is dominated by D.

We now bound the running time of an iteration of Algorithm 1, in which a packetized matrix is computed. 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 in an $N \times N$ bipartite graph, which is $O(N^{2.5})$ [31].

In [27], we show how Algorithm 1 can be modified, for any $\epsilon > 0$, to take $O((1/\epsilon)N \log N)$ time to compute each packetized matrix, using $N/\log N$ parallel processors, and giving a bound on worst-case backlog of $(1 + \epsilon)(N + 1)^2/4$. This modified algorithm does not compute a single, perfect matching at each time step, but instead uses pipelined, batch scheduling and a fast algorithm for edge-coloring bipartite multigraphs from [32], [33] to compute long sequences of packet transmissions.[11] Serializing this modified algorithm gives a packet-scheduling algorithm that guarantees bounded backlog, and that takes time $O(N^2)$ to compute each packetized matrix. This is, up to a constant factor, the same as the time required to read all components of a (time-varying) fluid matrix.

To our knowledge, it is an open question whether scheduling maximum-weight, bipartite matchings using backlog as weights, similar to the technique of McKeown, Anantharam, and Walrand in [18], would guarantee bounded backlog for all time-varying fluid policies. A key difference in the scheduling problem in [18] and our problem is that the former assumes i.i.d. probabilistic arrivals, while our scenario involves analysis of worst-case, or adversarial, desired traffic rates. Since the most efficient known algorithm for computing a maximum-weight matching in a bipartite graph has complexity $O(N^{2.5} \log N)$ in the case of polynomially bounded weights [34], any scheduling algorithm relying on such an approach would need a running time at least as large.

V. Banyan Networks

For the rest of this work we look at the necessary and sufficient speedup to maintain bounded backlog for Banyan multistage switch networks. The design of packet-scheduling algorithms for such networks is significantly more difficult than for a single crossbar switch, because of the potential for overloading internal links. Packets originating from different input ports and sent to different output ports may follow routes that use the same link of an internal switch element; these packets cannot be simultaneously transmitted, since this would result in an overloaded link and thus a dropped packet.

After presenting the structure and some properties of Banyan networks in Section VI, we show in Section VII that already for small Banyan networks, speedup is necessary for maintaining bounded backlog. For the 4×4 Banyan network, we show speedup at least 4/3 is required for maintaining bounded backlog.

Section VIII contains the core of our methodology. We characterize the required speedup to maintain bounded backlog for all fluid policies in terms of two polytopes derived from the link graph (defined below) of a Banyan network. We first state a result, which follows directly from a theorem of Koksal [11], characterizing the necessary and sufficient speedup for maintaining bounded backlog for *constant* fluid policies. Our first theorem strengthens this result, and proves that if speedup s is sufficient for maintaining bounded backlog for all *constant* fluid policies, then in fact it is sufficient for maintaining bounded backlog for *arbitrary* fluid policies.

In Section IX, we revisit the 4×4 Banyan network, and show, using the machinery developed in Section VIII, that speedup 4/3 is in fact necessary and sufficient for maintaining bounded backlog for arbitrary fluid policies. We also briefly discuss our

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

[11]Edge-colorings of bipartite multigraphs were used by Lee and Lam [23] to compute efficient schedules for switch networks.

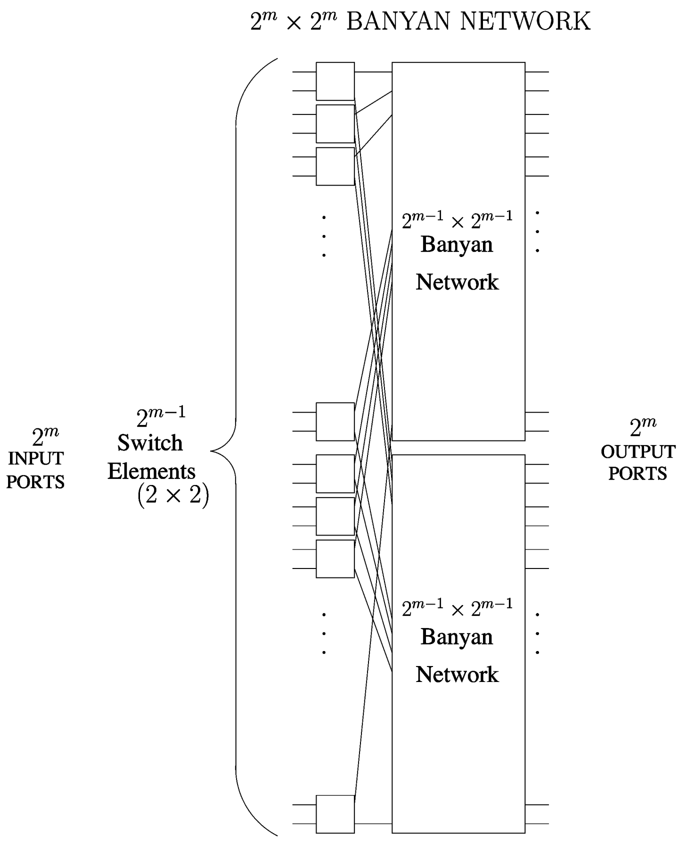
$2^m \times 2^m$ BANYAN NETWORK
2^m INPUT PORTS
2^{m-1} Switch Elements (2×2)
$2^{m-1} \times 2^{m-1}$ Banyan Network
$2^{m-1} \times 2^{m-1}$ Banyan Network
2^m OUTPUT PORTS

Fig. 4. Recursive construction of a $2^m \times 2^m$ Banyan network, for $m \geq 2$.

computer-aided analysis of polytopes, for which the details are given in [27], that indicates the necessary and sufficient speedup to keep backlog bounded on 8 × 8 Banyan networks is also 4/3.

In Section X, we show that for a Banyan network with N input ports, speedup $s = \log_2 N + 1$ is sufficient for maintaining bounded backlog for an arbitrary fluid policy. In this case, we show how to implement the packet-scheduling algorithm of Section VIII, using speedup $s = \log_2 N + 1$, to compute each packetized matrix in time polynomial in N.

VI. Structure of Banyan Networks

A. Recursive Construction and Properties

$N \times N$ Banyan networks have N input ports and N output ports for N a power of 2, and can be constructed recursively by appropriately connecting smaller Banyan networks. The following construction, depicted in Fig. 4, is from [25]. The 2 × 2 Banyan network is simply the 2 × 2 crossbar switch. For $m \geq 2$ and $N = 2^m$, the $N \times N$ Banyan network can be constructed by connecting 2^{m-1}, 2 × 2 crossbar switches to two $2^{m-1} \times 2^{m-1}$ Banyan networks as shown in Fig. 4. The first (topmost) 2 × 2 crossbar switch has its first outgoing link connected to the first input of the top $2^{m-1} \times 2^{m-1}$ Banyan network, and has its second outgoing link connected to the first input of the bottom $2^{m-1} \times 2^{m-1}$ Banyan network. The second 2 × 2 crossbar switch has its first outgoing link connected to the second input of the top $2^{m-1} \times 2^{m-1}$ Banyan network, and has its second outgoing link connected to the second input of

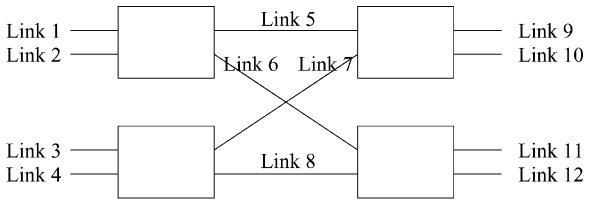

Fig. 5. A 4 × 4 Banyan network.

the bottom $2^{m-1} \times 2^{m-1}$ Banyan network. This process is continued until all 2^{m-1} of the 2 × 2 crossbar switches are connected, at which point the $2^m \times 2^m$ Banyan network is fully constructed. It has m stages.

Another property of Banyan networks is expressed in the following lemma, which deals with sets of input, output pairs and the paths connecting them. A *path* through a $2^m \times 2^m$ Banyan network is a sequence of links $l_0, l_1, \ldots, l_m$, where link l_h is between stages h and $h+1$, and for $h : 0 < h \leq m$, link l_h is an outgoing link from the switch element with incoming link l_{h-1}.

Lemma 3: For any set S of input, output paths through a Banyan network such that each pair of paths in S shares some link, there is some link l contained in all paths in S.[12]

The lemma is proved in Appendix II.

B. The Link Graph of a Banyan Network

We define the *link graph* $G = (V, E)$ of a Banyan network as follows: the link graph has a node for every input, output pair (i, j). Two nodes (i, j), (k, l) are connected by an edge in the graph if the unique path from input i to output j shares a link with the path from k to l. In Figs. 5 and 6 we show the 4 × 4 Banyan network, and the associated link graph G. Note that a {0,1}-valued, $N \times N$ matrix is a valid packetized matrix if and only if the set of entries with value 1 corresponds to a *stable set*, that is, a set of nodes with no edges between them, in the link graph of the switch network.

Consider link 2 in Fig. 5. Link 2 is required for any packet transmission from input 2 to outputs 1, 2, 3 or 4. Therefore, in a packetized model, at most one of these four transmissions can occur per time step. In the link graph, this constraint is represented by a *clique*, that is a set of nodes with an edge between each pair, {(2,1),(2,2),(2,3),(2,4)}; this corresponds to clique A in Fig. 6. Similarly, link 8 is required for transmission from 3 to 3, 3 to 4, 4 to 3, and 4 to 4, so among these input, output pairs, at most one transmission can take place. In the link graph, we have a clique among nodes {(3,3),(3,4),(4,3),(4,4)}; this corresponds to clique B in the figure.

By Lemma 3 above, each clique in the link graph corresponds to a set of input, output pairs, all of whose paths contain some link l.

We now show an important connection between the cliques in the link graph and the set of valid fluid matrices for the Banyan

[12]In general, a family of sets is said to have the *Helly property* if for any subfamily of pairwise nondisjoint sets, the intersection of the sets in the subfamily is nonempty [35]. For Banyan networks, this lemma shows that the set of paths through the network has the Helly property, where each path represents the set of links it contains, two paths are considered disjoint if they have no links in common, and the intersection of a set of paths is the set of links common to all of them.

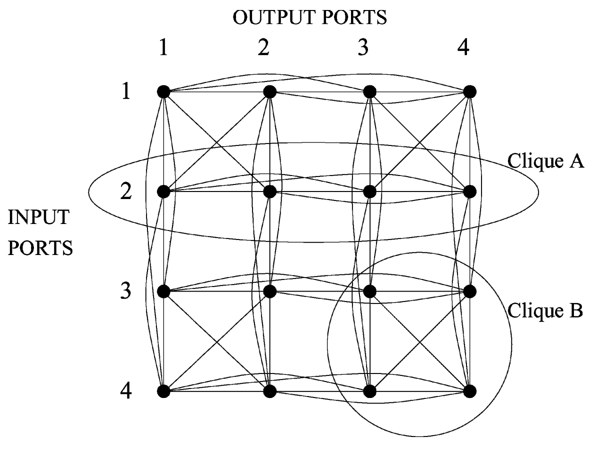

Fig. 6. The link graph corresponding to the above 4 × 4 Banyan network.

network. By Lemma 3, a non-negative-valued, $N \times N$ matrix F is a valid fluid matrix if and only if for each clique Q in the link graph G, the following *clique constraint* is satisfied:

$$\sum_{(i,j)\in Q} F_{ij} \le 1. \tag{1}$$

We will see that the structure of the link graph (which is derived from the topology of the switch network) has an intimate connection with how much speedup is necessary and sufficient to maintain bounded backlog for all fluid policies on a Banyan network.

VII. Speedup is Required

In this section, we exhibit a behavior of the Banyan network that is fundamentally different from the crossbar switch. We exhibit a constant fluid policy for which, using no speedup, it is impossible to maintain bounded backlog. Recall the 4 × 4 Banyan network in Fig. 5. Consider the constant fluid policy, with each fluid matrix $F^{(t)}$ equal to the matrix

$$F = \begin{bmatrix} \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \end{bmatrix}.$$

Note that for every clique Q in the link graph G (see Fig. 6 above) we have

$$\sum_{(i,j)\in Q} F_{ij} = 1$$

and the above is indeed a valid fluid matrix. Suppose that at each time step, this fluid matrix is requested, and knowing this stationary policy in advance, we wish to choose a valid packetized policy so that the total backlog after M time steps is minimized. While we cannot transmit fractional values with packetized policies, if we could transmit unit value along four of the eight pairs of positive entries in the fluid matrix at one time step, and then transmit the remaining four at the next time step, then the backlog would remain bounded. However, one can verify that a packetized policy cannot transmit any more than three of the eight pairs of positive entries of the fluid policy, at any given step. For instance, if (1,1) is transmitted, this rules out {(1,4),(3,1),(2,2)}. Then if, say, (2,3) is transmitted, (3,3) is ruled out, and of the two that remain, {(4,2),(4,4)}, only one can be transmitted. The same can be seen to be true for any possible set of choices. Therefore, any packetized policy can only transmit 3 units per time step, while the fluid policy transmits 4 units each time step. Thus, regardless of which packetized policy we choose, the backlog becomes unbounded. In fact, we have proved that the minimum speedup required on a 4 × 4 Banyan network for maintaining bounded backlog for any constant fluid policy is at least 4/3. In Section IX we show that this result is tight.

VIII. Characterization of Required Speedup

In this section, we give a characterization of the required speedup for maintaining bounded backlog for all fluid policies in Banyan networks. In addition, we develop the essential elements of our polyhedral and combinatorial methodology that we use in Sections IX and X. We define the polytope $\mathcal{P}$ to be the convex hull[13] of the set of valid packetized matrices, and the polytope $\mathcal{F}$ to be the set of valid fluid matrices when no speedup is used. Using terminology from polyhedral combinatorics, we note that the polytopes $\mathcal{P}$ and $\mathcal{F}$ are, respectively, the *stable set polytope* and the *fractional stable set polytope* of the link graph of the Banyan network; the stable set polytope and the fractional stable set polytope for general graphs have been studied extensively in the combinatorics literature (see [34], [36] for details).

We have $\mathcal{P} \subseteq \mathcal{F}$, since any convex combination of a set of valid packetized matrices is a valid fluid matrix. The example of the 4 × 4 Banyan network in Section VII above shows that this inclusion can be strict. For a switch network with N input ports and N output ports, the dimension of $\mathcal{P}$ is N^2, since any stable set polytope is always full-dimensional.

Recall that we model a scheduling algorithm using *speedup* $s \ge 1$ by requiring for each fluid matrix in any fluid policy, that its link usage totals at most $1/s$ for each link. This is equivalent to requiring for all fluid matrices $F^{(t)}$ in any fluid policy, that $F^{(t)} \in (1/s)\mathcal{F}$.

If $\mathcal{P} = \mathcal{F}$, then every valid fluid matrix can be written as a convex combination of valid packetized matrices and so for any constant fluid policy, bounded backlog can be maintained using no speedup (by simply scheduling the valid packetized matrices in the decomposition at the right frequencies). In graph theoretic terms, $\mathcal{P} = \mathcal{F}$ is equivalent to the link graph being *perfect*. For this, as well as combinatorial results cited elsewhere in this work, we refer the interested reader to [34], [36] for further details. Many classes of perfect graphs are known, and in particular, the link graph of a crossbar switch is perfect, as it can be seen to be the line graph of a complete bipartite graph. This is

[13]A *convex combination* of matrices in a set S is a finite sum of the form $\sum \lambda_i M_i$, where for each i, $\lambda_i \ge 0$, $M_i \in S$, and we have $\sum \lambda_i = 1$. The *convex hull* of a set S of matrices is the set of all convex combinations of matrices in S.

$$F = \begin{bmatrix} F_{11} & F_{12} & F_{13} & F_{14} \\ F_{21} & F_{22} & F_{23} & F_{24} \\ F_{31} & F_{32} & F_{33} & F_{34} \\ F_{41} & F_{42} & F_{43} & F_{44} \end{bmatrix} = \frac{1}{3}\begin{bmatrix} 0 & 0 & F_{13} & F_{14} \\ 0 & 0 & F_{23} & F_{24} \\ F_{31} & F_{32} & F_{33} & F_{34} \\ F_{41} & F_{42} & F_{43} & F_{44} \end{bmatrix} + \frac{1}{3}\begin{bmatrix} F_{11} & F_{12} & 0 & 0 \\ F_{21} & F_{22} & 0 & 0 \\ F_{31} & F_{32} & F_{33} & F_{34} \\ F_{41} & F_{42} & F_{43} & F_{44} \end{bmatrix}$$
$$+\frac{1}{3}\begin{bmatrix} F_{11} & F_{12} & F_{13} & F_{14} \\ F_{21} & F_{22} & F_{23} & F_{24} \\ 0 & 0 & F_{33} & F_{34} \\ 0 & 0 & F_{43} & F_{44} \end{bmatrix} + \frac{1}{3}\begin{bmatrix} F_{11} & F_{12} & F_{13} & F_{14} \\ F_{21} & F_{22} & F_{23} & F_{24} \\ F_{31} & F_{32} & 0 & 0 \\ F_{41} & F_{42} & 0 & 0 \end{bmatrix}$$

Fig. 7. We obtain the upper bound on speedup for the 4 × 4 Banyan network by decomposing any 4 × 4, valid fluid matrix into four parts as shown above.

a graph-theoretic explanation of the fact that no speedup is required for maintaining bounded backlog on the single crossbar switch.

Koksal shows for layered, multistage switch networks, that it is possible to maintain bounded backlog for a constant, fluid policy scheduling fluid matrix F at each time step if and only if $F \in \mathcal{P}$ [11]. This implies that using speedup s, bounded backlog can be maintained for all *constant* fluid policies if and only if $(1/s)\mathcal{F} \subseteq \mathcal{P}$. We show below that the necessary and sufficient speedup for maintaining bounded backlog for all constant fluid policies is the same as that for maintaining bounded backlog for arbitrary fluid policies. This implies, for any switch network operating strictly slower than the minimum required speedup, *even for constant fluid policies, bounded backlog cannot be maintained*; as soon as the switch network runs at least as fast as the minimum required speedup, then *bounded backlog can be maintained for any fluid policy*.

Theorem 3: Using speedup s, bounded backlog can be maintained for all *arbitrary* fluid policies if and only if $(1/s)\mathcal{F} \subseteq \mathcal{P}$.

Proof: Koksal, in his result from [11] mentioned above, showed that if $(1/s)\mathcal{F} \nsubseteq \mathcal{P}$, then for any matrix $F \in (1/s)\mathcal{F}$ for which $F \notin \mathcal{P}$, backlog cannot be kept bounded for the constant fluid policy scheduling F at each time step.

To show the opposite direction it suffices to exhibit, in the case where $(1/s)\mathcal{F} \subseteq \mathcal{P}$, a packet-scheduling algorithm using speedup s that maintains bounded backlog for any fluid policy.

Assume $(1/s)\mathcal{F} \subseteq \mathcal{P}$ holds, so that each fluid matrix $F^{(t)} \in \mathcal{P}$. We present a packet-scheduling algorithm using speedup s that, for any fluid policy, maintains backlog at most N^2 per input port and per output port. The algorithm maintains the following invariant:

Invariant 2: $C^{(t)} \in N^2\mathcal{P}$: This invariant implies that no input port can have backlog more than N^2. We present the algorithm below and prove inductively that it maintains the invariant above. Note that the invariant holds at time step $t = 0$, since $C^{(0)} = \mathbf{0}$, which is in $N^2\mathcal{P}$.

Algorithm 2: Given a fluid policy, this packet-scheduling algorithm computes the packetized policy as follows:

For $t \geq 0$, by our assumption above that $F^{(t+1)} \in \mathcal{P}$, and assuming the invariant holds at time step t, we have $C^{(t)} + F^{(t+1)} \in (N^2+1)\mathcal{P}$. Since $\mathcal{P}$ is an N^2-dimensional polytope, Caratheodory's theorem[14] says that any point in $\mathcal{P}$ can be written as a convex combination of at most N^2+1 vertices, which in this case are packetized matrices. Thus, we can decompose $C^{(t)} + F^{(t+1)}$ into a convex combination of at most N^2+1 vertices of $(N^2+1)\mathcal{P}$. At least one matrix in the decomposition must now have weight at least 1. Set packetized matrix $P^{(t+1)}$ to be one such matrix.

It follows that $C^{(t+1)} = C^{(t)} + F^{(t+1)} - P^{(t+1)} \in N^2\mathcal{P}$, and so the invariant holds at time step $t+1$. Thus, for any fluid policy, Algorithm 2 maintains backlog at most N^2 per input port and per output port. ■

In the proof of the above result, we use Caratheodory's Theorem to decompose $C^{(t)} + F^{(t+1)}$ into a convex combination of packetized matrices. Caratheodory's Theorem, however, is not constructive in general (unless, for example, one has a description of $\mathcal{P}$ in terms of linear inequalities). In Appendix III, for $(1/s)\mathcal{F} \subseteq \mathcal{P}$, we give a modified, packet-scheduling algorithm using speedup s whose only nonconstructive step is decomposing each fluid matrix $F^{(t)}$ into a convex combination of packetized matrices; we show this algorithm maintains bounded backlog for any fluid policy. We give an algorithm in Section X that for the case of a Banyan network with N input ports and for speedup $s = \log_2 N + 1$, computes such a decomposition in time polynomial in N; in this case, combining the two algorithms, we have a packet-scheduling algorithm that runs in time polynomial in N and maintains bounded backlog for any fluid policy.

IX. Speedup Required for 4 × 4 Banyan Networks

In Section VII, we exhibited a constant fluid policy on a 4 × 4 Banyan network that requires speedup at least 4/3 for maintaining bounded backlog. Using the results of Section VIII above, we show that in fact speedup $s = 4/3$ is necessary and sufficient for maintaining bounded backlog for arbitrary fluid policies on the 4 × 4 Banyan network.

From the above discussion, it is sufficient to show $(3/4)\mathcal{F} \subseteq \mathcal{P}$ for the 4 × 4 Banyan network. To show this, decompose any valid fluid matrix F into a linear combination of four matrices, each with the four entries in one corner set to 0, as shown in Fig. 7. The weight of each matrix is 1/3. We then use the fact that the subgraph corresponding to one of these matrices with a corner deleted, is a perfect graph. Recall from our discussion above, that we have $\mathcal{P} = \mathcal{F}$ if and only if the link graph is perfect. Therefore, one can further decompose any of these four matrices into a convex combination of valid packetized matrices. That the subgraph corresponding to one of these matrices with a corner deleted is perfect, follows from the fact (see [36]) that

[14]Caratheodory's theorem states that if $X \subseteq \boldsymbol{R}^d$, then any point in the convex hull of X, $\mathrm{conv}(X)$, may be written as a convex combination of at most $(d+1)$ points of X (see, e.g., [37]).

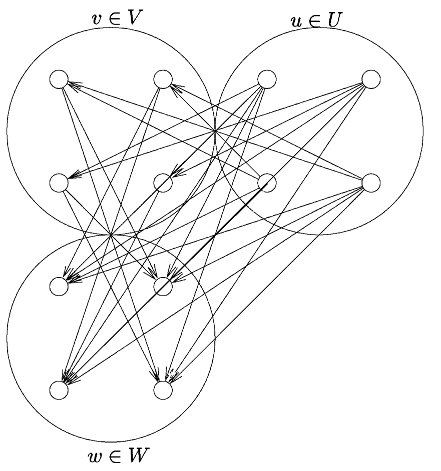

Fig. 8. This is the complement of a 4 × 4 Banyan network link graph with the corner removed. The edges are oriented so that edges between nodes in U and nodes in V are directed towards V, edges between U and W are directed towards W, and edges between V and W are directed towards W. Therefore, this is a directed, acyclic, transitive graph.

the resulting subgraph is the complement of a so-called *comparability graph.* A comparability graph is such that its edges can be oriented so they form a directed, acyclic, transitive graph $D = (V, A)$. Here, *transitive* means that for any nodes u, v, w, if $(u, v) \in A$ and $(v, w) \in A$, then also $(u, w) \in A$. In Fig. 8 we exhibit such an orientation of the complement of the subgraph obtained when the bottom right corner of the link graph of the 4 × 4 Banyan network is removed. It is well known (see e.g., [36]) that complements of comparability graphs are perfect. Thus, each of the four matrices in the linear decomposition of F given in Fig. 7 can be written as a convex combination of valid packetized matrices. Replacing each of the four matrices in Fig. 7 by such a convex decomposition results in a non-negative, linear combination of valid packetized matrices, with the sum of weights 4/3. This shows that $(3/4)\mathcal{F} \subseteq \mathcal{P}$. We can then use packet-scheduling Algorithm 2 above with 4/3 speedup to build, for any given fluid policy, a packetized policy that maintains backlog at most 16 packets per input port and per output port.

Using the software package **cdd+**[15] to enumerate and analyze over 500,000 vertices of certain polytopes, we argue in [27] that the necessary and sufficient speedup to keep backlog bounded for any fluid policy on 8 × 8 Banyan networks is also 4/3. The details of this computation are given in [27].

X. Bounds on Speedup Required for $N \times N$ Banyan Networks

The main result of this section is a greedy algorithm for decomposing any valid fluid matrix on a Banyan network. This algorithm is an extension of the maximal matching algorithm used by Smiljanić on the $N \times N$ crossbar switch [39].

Theorem 4: For a Banyan network with N input ports, we exhibit an algorithm (Algorithm 3 below) that, for any $F \in (1/(\log_2 N + 1))\mathcal{F}$, decomposes F into a convex combination of $N^2 + 1$ vertices of $\mathcal{P}$; the algorithm runs in time polynomial in N.

[15]**cdd+** is an implementation by Komei Fukuda of the Double Description Method [38] for generating all vertices and extreme rays of a general convex polyhedron given by a system of linear inequalities. See http://www.cs.mcgill.ca/~fukuda/soft/cddman/node2.html for details.

An immediate corollary is $(1/(\log_2 N + 1))\mathcal{F} \subseteq \mathcal{P}$. Using Algorithm 3 below as a subroutine in Algorithm 4, we have our main theorem for Banyan networks, which gives an upper bound on the speedup required for maintaining bounded backlog for any fluid policy.

Theorem 5: For a Banyan network with N input ports, we have a packet-scheduling algorithm using speedup $\log_2 N + 1$ that maintains bounded backlog for any fluid policy, and that runs in time polynomial in N.

To prove Theorem 4, it suffices to show for any $F \in (1/(\log_2 N + 1))\mathcal{F}$, that one can compute in time polynomial in N, a decomposition of F into a linear combination

$$F = \sum_{k=1}^{k'} \gamma_k R^{(k)} \qquad (2)$$

for some $k' \leq N^2$, for non-negative γ_k summing to at most 1, and with $R^{(k)}$ a valid packetized matrix for each $k \leq k'$. The greedy algorithm below produces the decomposition (2). We give the algorithm, and then the proof of correctness.

We use the notation that for a stable set S of the link graph G of a Banyan network, χ^S denotes the $N \times N$, valid packetized matrix with value 1 at entries corresponding to elements of S and with value 0 otherwise.

Algorithm 3: Let F be a given fluid matrix in $(1/(\log_2 N + 1))\mathcal{F}$.

```
1) Set $\hat{F} \leftarrow F$ and set $k \leftarrow 1$.
2) Repeat while $\hat{F} \neq \mathbf{0}$:
    • Find a maximal stable set in the link graph $G$ restricted
      to the set of nodes with positive value in $\hat{F}$. Call it $S_k$.
    • Set $\gamma_k$ to be the largest value such that $\hat{F} - \gamma_k \chi^{S_k}$ is
      non-negative.
    • Set $R^{(k)} \leftarrow \chi^{S_k}$ and $\hat{F} \leftarrow \hat{F} - \gamma_k R^{(k)}$ and then incre-
      ment $k$ by 1.
```

Since at each iteration of step 2, at least one entry of $\hat{F}$ is set to 0, the algorithm terminates after $k' \leq N^2$ iterations. Upon termination, we have $F = \sum_{k=1}^{k'} \gamma_k R^{(k)}$. It remains to show that the sum of γ_k's is at most 1. Just before the last iteration k', there exists an input, output pair (i, j) such that $\hat{F}_{ij} > 0$. Recall that in an $N \times N$ Banyan network there are $\log_2 N$ stages, and hence every input, output path consists of $\log_2 N + 1$ links. Let $(l_0, l_1, \ldots, l_{\log_2 N})$ be the unique path from input i to output j.

For each iteration $k \leq k'$, let L_k denote the links among $\{l_0, l_1, \ldots, l_{\log_2 N}\}$ that, for some pair (i', j') with $R^{(k)}_{i'j'} = 1$, are contained in the path from input port i' to output port j'. Each L_k is nonempty, since otherwise adding (i, j) to stable set S_k would result in a larger stable set, which is not possible since a *maximal* stable set is selected at each iteration of the algorithm.

Since we assumed $F \in (1/(\log_2 N + 1))\mathcal{F}$, for each $d : 0 \leq d \leq \log_2 N$ we have

$$\sum_{k: l_d \in L_k} \gamma_k \leq \frac{1}{\log_2 N + 1}.$$

Since as argued above each L_k contains some link in $\{l_0, l_1, \ldots, l_{\log_2 N}\}$, we have

$$\sum_{k=1}^{k'} \gamma_k \leq \sum_{d=0}^{\log_2 N} \sum_{k:l_d \in L_k} \gamma_k \leq (\log_2 N + 1)\frac{1}{\log_2 N + 1} = 1.$$

Thus we have bounded $\sum_{k=1}^{k'} \gamma_k$ by 1, as desired. ■

To show the algorithm above runs in time polynomial in N, it suffices to show that each stable set S_k can be found in time polynomial in N. In [27], we show that all the stable sets S_k can be found in total time $O(N^3 \log^2 N)$.

By a similar argument, it can be shown for any layered, unit-capacity, unique-path, multistage switch network, that speedup equal to the longest path length in the network is sufficient for maintaining bounded backlog.

XI. Conclusion and Further Extensions

In this paper, we have considered under what conditions there exist packet-scheduling algorithms that maintain bounded backlog for arbitrary time-varying fluid policies for the crossbar switch, and the Banyan network. For the crossbar switch, it has long been known that maintaining bounded backlog is possible in two restricted settings: if the fluid policy is not allowed to vary over time, or, if the fluid policy varies, and we have speedup. It was not known, until now, whether maintaining bounded backlog is possible in general. We give a combinatorial construction of a packet-scheduling algorithm that without any speedup, maintains bounded backlog in the worst case, for an arbitrary (possibly adversarially constructed) time-varying fluid policy.

Next, we showed that in contrast to the crossbar switch, Banyan networks require speedup in order to maintain bounded backlog for arbitrary fluid policies. With this motivation, we turned to analyzing the necessary and sufficient speedup to maintain bounded backlog for Banyan networks. Translating the problem into essentially one of containment of polytopes, we characterized the necessary and sufficient speedup required to maintain bounded backlog for an $N \times N$ Banyan network. Furthermore, we computed the exact speedup required to maintain bounded backlog for the 4×4, and 8×8 Banyan network, and obtained logarithmic bounds on the speedup required for a general $N \times N$ Banyan network. In [27], some of these results are extended to the much more general setting of arbitrary switch fabrics.

Computing the exact speedup required for general Banyan networks, and other networks of interest, remains an interesting and stimulating open problem. The area of general networks poses further computational and theoretical challenges. Constructing efficient scheduling algorithms, and computing the fundamental boundaries of the tradeoffs between speedup, backlog, and delay, seems to be a research area well worth further attention and study.

Appendix I
Proof that Algorithm 1 is Well-Defined and Satisfies Invariant 1

The proof is by induction on the time step t. 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 t. Recall that we set $P := P^{(t+1)}$, $C := C^{(t)}$, and $F := F^{(t+1)}$ for clarity of exposition. We first show that the algorithm is well-defined at time step $t+1$:

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

From the first two steps of the algorithm, we have

$$C + F \leq B. \tag{3}$$

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

$$(C + F - \pi)^+ \leq (B - \pi)^+ = B - \pi \tag{4}$$

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

$$(C + F - \pi)^+ = (C + F - P)^+. \tag{5}$$

Therefore, from (4), 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 $C^{(t+1)} := C + F - P$, this proves Invariant 1 for time step $t+1$. The induction is complete. ■

Appendix II
Proof of the Helly Property

Next we prove Lemma 3 from Section VI-A, which says that for any set S of input, output paths through a Banyan network such that each pair of paths in S shares some link, there is some link l contained in all paths in S.

Proof: The proof is by induction on the size of the Banyan network. For 2×2 Banyan networks, one can verify that any set of paths S such that each pair of paths shares some link must either contain a single path, or be a set of two paths. The lemma trivially holds in this case.

Assume the lemma holds for $2^{m-1} \times 2^{m-1}$ Banyan networks, for some $m \geq 2$. We show it holds for the $2^m \times 2^m$ Banyan network $\mathcal{N}$, using the recursive structure shown in Fig. 4. Take any set S of input, output paths through the Banyan network $\mathcal{N}$ such that each pair of paths in S shares some link. If all paths in S have the same first link, the lemma holds. If not, then either the last link in each path in S is one of the first 2^{m-1} output ports, or the last link in each path in S is one of the last 2^{m-1} output ports. This follows since if the last link in $p_1 \in S$ were one of the first 2^{m-1} output ports and the last link in $p_2 \in S$ were one of the last 2^{m-1} output ports, then their only shared link could be their first links; all the other paths in S, which were assumed to share a link with p_1 and a link with p_2, by the structure of the Banyan network must share their common first link, which we assumed was not the case. Thus, either the last link in each path in S is one of the first 2^{m-1} output ports, or the last link in each path in S is one of the last 2^{m-1} output ports. In other words, for S' the set of paths that result when each path in S has its first link removed, one of the two $2^{m-1} \times 2^{m-1}$ Banyan networks in the recursive construction of $\mathcal{N}$ contains all paths in S'; let $\mathcal{N}'$ denote this $2^{m-1} \times 2^{m-1}$ Banyan network. The previous sentence implies that if paths $p_1, p_2 \in S$ have the same first link, they must also have the same second link. This, and our assumption that each pair of paths in S shares some link imply that each pair of paths in S' shares some link. Now, the lemma holds by the inductive hypothesis applied to $\mathcal{N}'$, which contains all paths in S'. ∎

APPENDIX III
A CONSTRUCTIVE VERSION OF CARATHEODORY

In Theorem 3 of Section VIII, we characterized exactly the necessary and sufficient speedup required to maintain bounded backlog. In Algorithm 2 given there, we appealed to Caratheodory's theorem, which as we noted, is not constructive. Here, for $(1/s)\mathcal{F} \subseteq \mathcal{P}$, we give a modified, packet-scheduling algorithm using speedup s whose only nonconstructive step is decomposing each fluid matrix $F^{(t)}$ into a convex combination of packetized matrices; we show this algorithm maintains bounded backlog for any fluid policy. Recall that we gave an algorithm in Section X that for the case of a Banyan network with N input ports and for speedup $s = \log_2 N + 1$, computes such a decomposition in time polynomial in N; in this case, combining the two algorithms, we have a packet-scheduling algorithm that runs in time polynomial in N and maintains bounded backlog for any fluid policy.

Assume $(1/s)\mathcal{F} \subseteq \mathcal{P}$, so that for each fluid matrix $F^{(t)}$, (which by definition is in $(1/s)\mathcal{F}$ when speedup s is used) we have $F^{(t)} \in \mathcal{P}$. Also, assume that given any fluid matrix $F^{(t)}$, we can compute a decomposition of it into a convex combination of at most N^2 packetized matrices.

The packet-scheduling algorithm below maintains the following invariant:

Invariant 3: For all time steps t, we have matrices $Q^{(1)}, \dots, Q^{(N^2+1)}$, which are vertices of $\mathcal{P}$, and non-negative coefficients $\lambda_1, \dots, \lambda_{N^2+1}$ (all of which may be different at different time steps) such that

$$C^{(t)} = \sum_{i=1}^{N^2+1} \lambda_i Q^{(i)}$$

and

$$\sum_{i=1}^{N^2+1} \lambda_i = N^2.$$

This invariant implies that for all t, $C^{(t)} \in N^2\mathcal{P}$, which means that the packet-scheduling algorithm, for any fluid policy, maintains backlog at most N^2 per input port and per output port at all time steps. We present the algorithm below and prove inductively that it maintains the invariant at all time steps. Note that the invariant holds at time $t = 0$ since here $C^{(0)} = \mathbf{0}$ and so we could initially set for all i, $\lambda_i := N^2/(N^2+1)$, and $Q^{(i)} := \mathbf{0}$.

Algorithm 4: For $t \geq 0$, given $C^{(t)}$ and fluid matrix $F^{(t+1)}$, construct packetized matrix $P^{(t+1)}$ as follows:

By our assumption above, we can compute a decomposition of $F^{(t+1)}$ as

$$F^{(t+1)} = \sum_{j=1}^{N^2+1} \gamma_j R^{(j)}$$

where for all j, $\gamma_j \geq 0$ and $R^{(j)}$ are vertices of $\mathcal{P}$, and the sum of γ_j's is 1. (It is this decomposition that we compute in polynomial time for speedup $s = \log_2 N + 1$ for Banyan networks in Section X.) Now, if we assume Invariant 3 holds for time step t, we have

$$C^{(t)} + F^{(t+1)} = \sum_{i=1}^{N^2+1} \lambda_i Q^{(i)} + \sum_{j=1}^{N^2+1} \gamma_j R^{(j)}.$$

The sum of λ_i's and γ_j's is $N^2 + 1$. Let

$$T := \sum_{i=1}^{N^2+1} \lambda_i Q^{(i)} + \sum_{j=1}^{N^2+1} \gamma_j R^{(j)}.$$

Caratheodory's theorem now tells us that T can also be expressed as a weighted sum of just $N^2 + 1$ matrices from the set $\{Q^{(i)}\} \cup \{R^{(j)}\}$, with the weights summing to $N^2 + 1$. We are now in a position to compute this expression explicitly, and efficiently. Consider the problem of finding non-negative variables $\{\hat{\lambda}_i\} \cup \{\hat{\gamma}_i\}$, so that the resulting weighted sum of $Q^{(i)}$ and $R^{(i)}$ (weighted by $\{\hat{\lambda}_i\}$ and $\{\hat{\gamma}_i\}$, respectively) equals T, and so that the sum of the $\{\hat{\lambda}_i\}$ and $\{\hat{\gamma}_i\}$ is $N^2 + 1$. This is a linear system with $2N^2 + 2$ variables (the $\{\hat{\lambda}_i\}$ and the $\{\hat{\gamma}_i\}$), and $N^2 + 1$ constraints: N^2 constraints for the N^2 entries of T, and then an additional constraint on the sum of the variables. Using $\boldsymbol{x} = (\boldsymbol{\lambda}, \boldsymbol{\gamma})$ to represent the variables, and $\boldsymbol{A}$ the $N^2 + 1$ equality constraints, we can write the linear system as $\boldsymbol{Ax} = \boldsymbol{b}$, where $\boldsymbol{b} = (\mathrm{vec}(T), N^2 + 1)$. We note that we need the decomposition of $F^{(t+1)}$ in order to define the linear system and the constraint matrix $\boldsymbol{A}$, since its definition depends on the matrices $R^{(j)}$ of the decomposition. Now, the values $\{\lambda_i, \gamma_i\}$ we have computed satisfy these equality constraints and furthermore are non-negative. Therefore, the polyhedral set: $\{\boldsymbol{x} : \boldsymbol{Ax} = \boldsymbol{b}, \boldsymbol{x} \geq 0\}$ is nonempty. Moreover, since it is contained in the positive orthant, it must also have at least one extreme point. At any extreme point of this feasible set of solutions, there must be at least $2N^2 + 2$ tight constraints. In other words, at least $N^2 + 1$ of the variables must be equal to zero. Finding extreme points is a standard exercise in linear optimization, and in particular, it can be done in polynomial time [30]. Then, let $\hat{\boldsymbol{x}} = \{\hat{\lambda}_i, \hat{\gamma}_i\}$ be such an extreme point. Now out of the original $Q^{(i)}$ and $R^{(i)}$ matrices, consider only the N^2+1 matrices corresponding to nonzero weights. Rename these matrices to be $\{Q^{(1)},\ldots,Q^{(k)}\}$ and the corresponding weights to be $\{\lambda_1,\ldots,\lambda_k\}$. If $k<N^2+1$, then for $i: k+1\le i\le N^2+1$, rename $Q^{(i)}:=\mathbf{0}$ and $\lambda_i:=0$.

Since the sum of λ_i's is N^2+1, one of them must be at least 1. Let i' be the least i such that $\lambda_i\ge 1$. Set $P^{(t+1)}$ to be $Q^{(i')}$. Subtract one from $\lambda_{i'}$.

Thus,

$$C^{(t+1)}=C^{(t)}+F^{(t+1)}-P^{(t+1)}=\sum_{i=1}^{N^2+1}\lambda_i Q^{(i)}$$

with $\sum_{i=1}^{N^2+1}\lambda_i=N^2$, proving the invariant holds at time step $t+1$. We have shown that the packet-scheduling algorithm maintains, for any fluid policy, backlog at most N^2 per input port and per output port. ■

Acknowledgment

The authors would like to thank T. Erlebach for pointing out the recent paper of Adler *et al.* [2], whose result was quoted in Section IV-A. They would also like to thank S. Fiorini and D. Spielman for their helpful feedback.

References

[1] C.-S. Chang, W.-J. Chen, and H.-Y. Huang, "Birkhoff-von Neumann input buffered crossbar switches," in *Proc. IEEE INFOCOM*, 2000, pp. 1614–1623.

[2] 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," presented at the ACM Symp. Parallel Algorithms and Architectures, San Diego, CA, Jun. 2003.

[3] J. C. R. Bennett and H. Zhang, "WF^{2}Q: worst-case fair weighted fair queueing," in *Proc. IEEE INFOCOM*, 1996, pp. 120–128.

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

[5] A. Charny, "Providing QoS guarantees in input-buffered crossbar switches with speedup," Ph.D. dissertation, Mass. Inst. Technol., Cambridge, MA, Aug. 1998.

[6] R. Cruz, "A calculus for network delay. I. Network elements in isolation," *IEEE Trans. Inf. Theory*, vol. 37, no. 1, pp. 114–131, Jan. 1991.

[7] ——, "A calculus for network delay. II. Network analysis," *IEEE Trans. Inf. Theory*, vol. 37, no. 1, pp. 132–141, Jan. 1991.

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

[9] M. J. Girone, "Tracking switch fluid policies: bounding lookahead," Master's project, Dept. Elect. Eng. Comput. Sci., Mass. Inst. Technol., Feb. 2002.

[10] A. C. Kam and K.-Y. Siu, "Linear complexity algorithms for QOS support in input-queued switches with no speedup," *IEEE J. Sel. Areas Commun.*, vol. 17, no. 6, pp. 1040–56, Jun. 1999.

[11] C. E. Koksal, "Providing quality of service over high speed electronic and optical switches," Ph.D. dissertation, Mass. Inst. Technol., Cambridge, MA, Sep. 2002.

[12] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: the single-node case," *IEEE/ACM Trans. Netw.*, vol. 1, no. 3, pp. 344–357, Jun. 1993.

[13] ——, "A generalized processor sharing approach to flow control in integrated services networks: the multiple node case," *IEEE/ACM Trans. Netw.*, vol. 2, no. 2, pp. 137–150, Apr. 1994.

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

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

[16] J. G. Dai and B. Prabhakar, "The throughput of data switches with and without speedup," in *Proc. IEEE INFOCOM*, 2000, pp. 556–564.

[17] E. Leonardi, M. Mellia, F. Neri, and M. Marsan, "On the stability of input-queued switches with speed-up," *IEEE/ACM Trans. Netw.*, vol. 9, no. 1, pp. 104–118, Feb. 2001.

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

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

[20] M. Andrews, B. Awerbuch, A. Fernandez, T. Leighton, Z. Liu, and J. Kleinberg, "Universal-stability results and performance bounds for greedy contention-resolution protocols," *J. ACM*, vol. 48, no. 1, pp. 39–69, 2001.

[21] M. Andrews and L. Zhang, "The effects of temporary sessions on network performance," in *Proc. 11th Annu. ACM-SIAM Symp. Discrete Algorithms (SODA)*, San Francisco, CA, 2000, pp. 448–457.

[22] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D. P. Williamson, "Adversarial queuing theory," *J. ACM*, vol. 48, no. 1, pp. 13–38, 2001.

[23] T. Lee and C. Lam, "Path switching—a quasi-static routing scheme for large-scale ATM packet switches," *IEEE J. Sel. Areas Commun.*, vol. 15, no. 5, pp. 914–924, Jun. 1997.

[24] L. Tassiulas and L. Georgiadis, "Any work-conserving policy stabilizes the ring with spatial re-use," *IEEE/ACM Trans. Netw.*, vol. 4, no. 2, pp. 205–208, Apr. 1996.

[25] A. Pattavina, *Switching Theory, Architectures and Performance in Broadband ATM Networks*. New York: Wiley, 1998.

[26] S. Keshav, *An Engineering Approach to Computer Networking*. Boston, MA: Addison-Wesley, 1997.

[27] M. A. Rosenblum, "Approximating fluid schedules in packet-switched networks," Ph.D. dissertation, Mass. Inst. Technol., Cambridge, MA, Sep. 2004.

[28] M. Rosenblum, M. X. Goemans, and V. Tarokh, "Universal bounds on buffer size for packetizing fluid policies in input queued, crossbar switches," in *Proc. IEEE INFOCOM*, Hong Kong, Mar. 2004, pp. 1126–1134.

[29] P. Hall, "On representatives of subsets," *J. London Math. Soc.*, no. 10, pp. 26–30, 1935.

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

[31] J. Hopcroft and R. Karp, "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs," *SIAM J. Computing*, no. 2, pp. 225–231, 1973.

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

[33] H. Gabow, "Using euler partitions to edge color bipartite multigraphs," *Int. J. Comput. Inf. Sci.*, no. 5, pp. 345–355, 1976.

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

[35] R. Wenger, "Helly-type theorems and geometric transversals," in *Handbook of Discrete and Computational Geometry*, J. E. Goodman and J. O'Rourke, Eds. Cleveland, OH: CRC Press, 1997.

[36] M. Grotschel, L. Lovász, and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*. Berlin: Springer-Verlag, 1993.

[37] J. Matoušek, *Lectures on Discrete Geometry*. New York: Springer, 2002.

[38] T. Motzkin, H. Raiffa, G. Thompson, and R. Thrall, *Contributions to Theory of Games*, H. W. Kuhn and A. W. Tucker, Eds. Princeton, NJ: Princeton Univ. Press, 1953, vol. 2.

[39] A. Smiljanić, "Flexible bandwidth allocation in high-capacity packet-switches," *IEEE/ACM Trans. Netw.*, vol. 10, no. 2, pp. 287–293, Apr. 2002.

Michael Rosenblum (S'03–M'04) received the B.S. degree in symbolic systems and the M.S. degree in mathematics from Stanford University, Stanford, CA, in 1998. He received the Ph.D. degree in applied mathematics from the Massachusetts Institute of Technology (MIT), Cambridge, MA, in 2004.

He is currently a postdoctoral researcher in biostatistics at University of California, Berkeley. His research interests include applications of statistics to optimization under uncertainty, with public health applications; for example, efficiently estimating optimal dynamic treatment regimens from randomized trial and observational study data.

Constantine Caramanis (S'05–M'06) received the A.B. degree in mathematics from Harvard University, Cambridge, MA, in 1999, and the M.S. and Ph.D. degrees in electrical engineering and computer science from the Massachusetts Institute of Technology (MIT), Cambridge, in 2001 and 2006, respectively.

He is currently an Assistant Professor of electrical and computer engineering at The University of Texas at Austin. His research interests include optimization and control under uncertainty, learning theory, and applications to communications, networks, and scheduling.

Michel X. Goemans received the Ph.D. degree in operations research from the Massachusetts Institute of Technology (MIT), Cambridge, MA, in 1990.

He is currently a Professor of applied mathematics at MIT, where he is a member of the Theory of Computation Group of the MIT Computer Science and Artificial Intelligence Laboratory and a member of the MIT Operations Research Center. His research interests include areas of combinatorics, theoretical computer science, optimization, and telecommunication.

Dr. Goemans received the Sloan Foundation Research Fellowship in 1995, the National Science Foundation Career Award in 1996, the IBM University Partnership Faculty Award in 1999, the American Mathematical Society Delbert Ray Fulkerson Prize in 2000, and the MIT School of Science Dean's Educational and Student Advising Award in 2004. He has been a member of the ACM since 2004.

Vahid Tarokh (M'06) received the Ph.D. degree in electrical engineering from the University of Waterloo, Ontario, Canada, in 1995.

After heading the Department of Wireless Communications and Signal Processing at AT&T Labs-Research, he was Associate Professor in the Department of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology (MIT), Cambridge, MA, from 2000 to 2002. Since then, he has been a Gordon-Mckay Professor of Electrical Engineering in the Division of Engineering and Applied Sciences at Harvard University, Cambridge, MA. He is a co-founder of Center for Communications and Networking (CNC) (jointly with Professor H. T. Kung) and a member of Harvard Broadband Communications Laboratory. His current research focuses in the area of networking, in particular the design of efficient network protocols, wireless networks, and algorithms for scheduling and switching.

Dr. Tarokh has received the Gold Medal of the Governor General of Canada in 1995, the IEEE Information Theory Society Prize Paper Award in 1999, and the National Science Foundation Alan T. Waterman Award in 2001. He was selected one of the Top 100 Inventors of the Year for 1999–2002 by the Technology Review Magazine. In 2003, he received an honorary D.Sc. from the University of Windsor.