Approximating Fluid Schedules in Packet-Switched Networks

by

Michael Aaron Rosenblum
B.S., Symbolic Systems; M.S., Mathematics
Stanford University, 1998

Submitted to the Department of Mathematics
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
September 2004

Author .....
Department of Mathematics
August 6, 2004

Certified by .....
Michel X. Goemans
Professor of Applied Mathematics
Thesis Co-Supervisor

Certified by .....
Vahid Tarokh
Professor of Electrical Engineering, Harvard University
Thesis Co-Supervisor

Accepted by .....
Rodolfo Ruben Rosales
Chairman, Committee on Applied Mathematics

Accepted by .....
Pavel Etingof
Chairman, Department Committee on Graduate Students

Approximating Fluid Schedules in Packet-Switched Networks

by

Michael Aaron Rosenblum

Submitted to the Department of Mathematics
on August 6, 2004, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

We consider a problem motivated by the desire to provide flexible, rate-based, quality of service guarantees for packets sent over switches and switch networks. Our focus is solving a type of on-line, 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 treat the incoming data as fluid, that is, 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 throughout the network in which only whole packets are sent. We prove worst-case bounds on the additional delay and buffer use that result from using such an approximation. These bounds depend on the network topology, the resources available to the scheduler, and the types of fluid policy allowed.

Thesis Co-Supervisor: Michel X. Goemans
Title: Professor of Applied Mathematics

Thesis Co-Supervisor: Vahid Tarokh
Title: Professor of Electrical Engineering, Harvard University

Acknowledgments

I would like to thank my co-advisors Professor Michel X. Goemans and Professor Vahid Tarokh for their extremely valuable support, encouragement, advice, and joint work on this research. It has been an honor to work with them. I also thank the other members of my thesis committee Professor Daniel Spielman and Professor Daniel Kleitman. I am lucky to have worked jointly with Constantine Caramanis on the results in Chapters 4 and 5; I appreciate not only his hard work and key insights, but also his camaraderie, which made my graduate school experience a more enjoyable one.

Thank you to my parents David and Nancy, and my brothers Daniel and Jonathan for their love and constant support. Also, thank you to Bob, Ann, Hanna, Maia, Alan, and Geoffrey Steinberg for their love and encouragement.

I wholeheartedly thank Brian Dean, Kathleen Dickey, Samuel Fiorini, Thomas Erlebach, Can Emre Koksal, and Daniel Spielman for their helpful feedback on my work. I would like to thank Greg Shomo for his computer help, and CSAIL Reading Room Librarians Paula Mickevich and Maria T. Sensale for their help throughout my research.

Parts of this work are joint with other researchers, and have been previously published, or have been adapted from work previously published. Sections 2.4 and 2.7 were in M. Rosenblum, R. Yim, M.X. Goemans, and V. Tarokh, Worst-case delay bounds for packetizing time-varying fluid schedules for a single server in a packet-switched network. In *Proceedings of the 38th Annual Conference on Information Sciences and Systems (CISS)*, pages 1553–1559, Princeton, NJ, 2004.

Section 2.6 and Chapter 3 were in M. Rosenblum, M. X. Goemans, and V. Tarokh, Universal bounds on buffer size for packetizing fluid policies in input queued, crossbar switches. In *Proceedings of IEEE INFOCOM*, Hong Kong, China, 2004. ©2004 IEEE. Reprinted, with permission.

Chapter 4, with the exception of Section 4.6, was in C. Caramanis, M. Rosenblum,

M.X. Goemans, and V. Tarokh, Scheduling algorithms for providing flexible, rate-based, quality of service guarantees for packet-switching in Banyan networks. In *Proceedings of the 38th Annual Conference on Information Sciences and Systems (CISS)*, pages 160–166, Princeton, NJ, 2004.

The material in this thesis is based upon research supported by the National Science Foundation under the Alan T. Waterman Award, Grant No. CCR-0139398, and under contracts ITR-0121495 and CCR-0098018. Also, I was supported during my first two years at MIT by a Department of Defense NDSEG Fellowship (contract number F46920-99-C-0054). Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the National Science Foundation or of the Department of Defense.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The problems we analyze in this thesis fit into the recently unified framework of adversarial queueing theory. We do not present a comprehensive overview of this field here (we refer the interested reader to [4, 9]); instead, we briefly introduce the main principles of adversarial queueing theory and show how they apply to our model. We compare and constrast the model used here with the models used in related work; however, we postpone detailed discussions of related results for different switch architectures until the chapters dealing with these architectures. Next, in Section 1.2 of this chapter, we describe the main feature of our model, the fluid adversary. In the last section, we present an overview of our results.

1.1 Our Model and Adversarial Queueing Theory

Adversarial queueing theory[1] is the analysis of stability in dynamic, packet-switched networks in which the packet arrival process is adversarial. That is, the evolving state of the queues in a network is considered a game between a packet-scheduling algorithm and an adversary with limited control over where packets arrive in the network, when they arrive, where they must be sent, and when they should arrive there. The packet-scheduling algorithm attempts to route packets to their destinations as quickly and

[1]The description of adversarial queueing theory in the following four paragraphs is paraphrased from [4, 9].

with as little buffering as possible.

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. [18, 41, 57]). 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 worst-case bounds obtained in adversarial queueing theory and in particular this work, however, are more robust in that the arrival process is not assumed to have any statistical properties.

The results in adversarial queueing theory, as in this work, are bounds on the performance of packet-scheduling algorithms as a function of their resources relative to those of the adversarial arrival process, and as a function of the network topology. The performance metrics used in adversarial queueing theory include how large queue sizes (buffers) get and how much time each packet spends in queues, in the worst-case. The performance metrics in this work are variants of these general metrics; they focus on the additional buffer use and delay that result from approximating a schedule that treats data as fluid, by a schedule that sends only whole packets; these performance metrics, which we call *backlog* and *delay*, are further explained in the next section. In this work, the resource available to the packet-scheduling algorithm is how much information it has on the the future decisions of the (fluid) arrival process (which we call lookahead); the adversary is constrained in terms of the rate at which it can insert deadlines for packet deliveries (which we call speedup).

In this work, we analyze adversaries that are free to allocate different rates at different time steps. This is referred to as the "temporary session model" in [5], as opposed to what they call the "permanent session model" used in [6, 7, 11, 12, 16, 17, 43, 44, 58]. The temporary session model, which according to Borodin et al. [9] "intuitively seems to provide a better model for ATM networks having heterogeneous and frequently changing rates of traffic" is used in [1, 4, 5, 9, 24, 34, 37, 53].

The model we use in this thesis, as in the models in [1, 7, 11, 12, 24, 34, 37, 55], allows the adversary to choose arrival and departure deadlines for each packet that should be met as closely as possible by the packet-scheduling algorithm; the adversary

is limited in that the deadlines selected must meet certain rate constraints. The adversary can make the delivery of a packet more urgent by selecting deadlines in which this packet's departure time from the network is soon after its arrival time to the network. These models differ from the models in [4, 5, 6, 9, 16, 17, 43, 44, 53], which allow the rate-constrained adversary to directly inject packets that should be transmitted to their destinations as quickly as possible by the packet-scheduling algorithm.

When we analyze packet-switched networks in this work, we assume that intermediate switches do not have buffers; that is, once a packet leaves its source, it travels without waiting in any queues until it reaches its destination. We do not allow packets to be dropped, which implies that the number of packets traversing any link at any time step can be at most that link's capacity. In such a setting, a centralized router and scheduler seem necessary to coordinate packet transmissions throughout the network. We assume that our packet-scheduling algorithm knows the states of all queues for every switch in the network; we also assume it can communicate instantaneously with each switch to request packet transmissions. The packet-scheduling algorithm can be thought of as a network manager.

1.2 Packet-Scheduling using Fluid Policies

In our approach to packet-scheduling, which is motivated by the goal of providing flexible, rate-based, quality of service guarantees, 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, 6, 7, 11, 12, 16, 17, 21, 24, 34, 37, 43, 44, 53, 55]). 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 use two metrics for how well a packetized policy approximates a fluid policy. The first, *backlog*, measures the gap in cumulative service between the fluid policy and the packetized policy. The second, the additional delay experienced by a packet due

to approximating a fluid policy by a packetized one (which we refer to simply as *delay*) is the difference between when the packetized policy sends a packet and when the fluid policy finishes transmitting that packet. If a packet-scheduling algorithm, given any fluid policy as input, outputs a packetized policy in which each packet experiences no delay, we say the algorithm *tracks*. These metrics are similar to metrics used in [12, 34, 37]. Our bounds on backlog and delay are not asymptotic; they apply to all switch sizes and to time increments of any finite duration.

The present work tackles the problem of bounding backlog and delay when time-varying fluid policies are approximated by packetized policies in individual switch elements and in switch networks.

1.3 Overview of Results

We briefly describe our results for each chapter. Switch architectures of increasing complexity are considered, starting with a single $N \times 1$ switch in Chapter 2 and building to general switch networks in Chapter 5.

Chapter 2 is devoted to analyzing backlog and delay for a single $N \times 1$ switch. This switch models processor sharing, in which N data streams (input ports) compete for processing time of a single server. First, we characterize exactly how much lookahead is required to track with no speedup. Next, we prove tight bounds on worst-case backlog when speedup, but no lookahead, is used. We then turn to analyzing the trade-offs between speedup and lookahead required to bound delay. We prove for packet-scheduling algorithms using bounded lookahead and no speedup that worst-case packet delay grows linearly with the number of input ports N. Also, we prove for algorithms using no lookahead that worst-case delay grows linearly with N and decreases exponentially with speedup. We look at algorithms using both lookahead and speedup; for fixed speedup and maximum tolerable delay, we lower bound the necessary lookahead (by a function linear in N) and upper bound sufficient lookahead (by a function linear in N). We end Chapter 2 with an example of how the above bounds can be translated into bounds on queue length and packet waiting time for

leaky-bucket arrival processes under the temporary session model of Andrews and Zhang [5].

In Chapter 3, we establish worst-case bounds on backlog for the $N \times N$ crossbar switch. Specifically, a packet-scheduling algorithm using no speedup and no lookahead is presented that guarantees backlog at most $(N+1)^2/4$ packets at each input port and at each output port, for any fluid policy. 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 $N \times N$ crossbar switch. The algorithm can be made to run in $O(N \log N)$ time per packetized step using $N/\log N$ parallel processors, by employing 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 for any packet-scheduling algorithm using no speedup and no lookahead, there is a fluid policy resulting in backlog more than $N/e - 2$ at an input port. We also extend the main packet-scheduling algorithm of this chapter to a generic scheme for bounding backlog for a general class of switch architectures; in Chapter 4, we give a polynomial-time algorithm implementing this scheme for Banyan networks.

Our results in Chapter 4 focus on a class of multistage switch networks called Banyan networks. We first prove that when no speedup is used, bounded backlog results like those proven in Chapter 3 for the $N \times N$ crossbar switch do not exist for arbitrary switch networks or even for 4×4 or larger Banyan networks. However, if the packet-scheduling algorithm is allowed to use 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. We prove that speedup 4/3 is necessary and sufficient to keep backlog bounded for any fluid policy on 4×4 Banyan networks. We also determine, through detailed analysis of polytopes and non-trivial computations using the package **cdd+**[2], that the necessary and sufficient speedup to keep backlog

[2] **cdd+** is, according to its author Komei Fukuda, an "implementation of the Double Description Method [42] for generating all vertices (i.e. extreme points) and extreme rays of a general convex polyhedron given by a system of linear inequalities." See http://www.cs.mcgill.ca/~

bounded for any fluid policy on 8×8 Banyan networks is also $4/3$. For general $N \times N$ Banyan networks, we give a polynomial time algorithm using speedup $\log_2 N + 1$ that keeps backlog bounded for any fluid policy.

We extend some of the results proved for Banyan networks to general, input-queued switch networks, which we simply refer to as general networks, in Chapter 5. A general network consists of a set of interconnected, crossbar switches, along with a set of data sources and data sinks. General networks, unlike Banyan networks, may be non-layered, may have multiple paths connecting each source and destination, and may have different integer capacities on each link. The main result in this chapter is an upper bound on the required speedup to maintain bounded backlog for any fluid policy; this bound is derived by analyzing integral and fractional static flows through the general network. We apply this bound to show that for d the maximum number of links in any simple, source-sink path in a general network, we have for any $\epsilon > 0$, a packet-scheduling algorithm using speedup $d + \epsilon$ that, for any given fluid policy, maintains bounded backlog.

fukuda/soft/cddman/node2.html for details.

Chapter 2

The $N \times 1$ Switch

2.1 Introduction

In this chapter, we consider only the $N \times 1$ switch, which models processor sharing. This type of switch has been extensively studied in the literature, for example in [1, 6, 16, 17, 21, 24, 43, 44, 47, 48, 52, 53]. The analysis of the $N \times 1$ switch gives intuition that will be useful in understanding the algorithms and lower bounds for $M \times N$ switches and general switch networks.

We define the traffic model in terms of fluid and packetized policies. Based on these, we define the performance metrics of delay and backlog. We also consider resources available to a packet-scheduling algorithm, focusing on speedup and lookahead. Next, we present worst-case bounds on these performance metrics in terms of the available resources.

2.2 Traffic Model

We consider a single server sending packets from N input ports (or, equivalently, from N data streams) to a shared output port within a packet-switched network operating in discrete time. All packets are assumed to have the same size, and time is normalized so that the maximum service rate (capacity) of any input port is one packet per time step. At each time step, the server can select one input port and

send one packet from it. At any time step, the server may decide to send no packets.

Our traffic model is similar to that used by Cruz [16, 17], Parekh and Gallager [43, 44], and Tabatabaee, Georgiadis, and Tassiulas [55], in that input traffic is first considered fluid and satisfies some burstiness and long-term rate constraints; for example, in [43, 44] and sometimes in [16, 17], the input traffic is a linear bounded arrival process, which is sometimes called a "leaky bucket" controlled process. A significant difference between the fluid constraints in [16, 17, 43, 44], and the fluid constraints used here and in [24, 34, 37, 55] is that in the former cases these constraints generally apply to the fluid at each input port as in a permanent session model, while in our model the constraints apply to the sum of fluid over all input ports as in a temporary session model [5]. Thus, our traffic model allows for variation in the rates of each input port, as long as the sum of rates is no more than the server capacity. This is made concrete in our definition of a fluid policy, which represents the ideal, rate-based, packet-scheduling behavior.

A *Fluid Policy* for a single server with N input ports is a sequence of rate allocations specifying the number of fractional packets that ideally would be sent to the output port from each input port at each time step. This is represented by a sequence of vectors $\{F^{(t)}\}_{t>0}$, each of length N with non-negative entries summing to at most 1, in which $F_i^{(t)}$ represents the fraction of a packet that should be sent from input port i at time step t. The *fluid start time* of the kth packet from input port i is the first time step such that a total of more than $k-1$ units of fluid have been sent from input port i by the fluid policy. The *fluid completion time* for the kth packet from input port i is the first time step at which at least a total of k units of fluid have been sent from input port i by the fluid policy. We next define a packetized policy, which should approximate the fluid policy.

A *Packetized Policy* for a single server with N input ports is a sequence of packet transmissions, one per time slot. This is represented by a sequence of $\{0,1\}$-valued vectors $\{P^{(t)}\}_{t>0}$ each of length N and containing at most a single 1. $P_i^{(t)}$ is 1 if and only if a packet is sent from input port i at time step t. We say that the packetized policy sends the kth packet from input port i at time t, if a total of exactly k packets

have been sent by the packetized policy from input port i by time t.

Ideally, given a fluid policy, a packetized policy would be constructed that sends each packet at or after that packet's fluid start time, and at or before that packet's fluid completion time. When this occurs, we say, as in [55], that the packetized policy *tracks* the given fluid policy. Since in many cases tracking is not possible, we study the problem of closely approximating a fluid policy by a packetized policy that sends each packet at or after that packet's fluid start time, but possibly after that packet's fluid completion time. The motivation for keeping one constraint and relaxing the other is based on our interpretation that a packet has arrived in its input queue by its fluid start time, and should ideally be sent by its fluid completion time; we do not relax the constraint that a packetized policy can only send a packet at or after that packet's fluid start time, since in our interpretation, a packet may not have arrived in its input queue before its fluid start time, and we want to ensure each packet is scheduled at or after its arrival time in the queue.[1] We define a *packet-scheduling algorithm* to be a deterministic algorithm that takes a fluid policy as input (possibly online), and outputs (one step at a time) a packetized policy that sends each packet at or after that packet's fluid start time. At a given time step, when a packet-scheduling algorithm outputs a packetized vector that sends packet p, we simply say that the packet-scheduling algorithm sends packet p.

We study two metrics of how well a packetized policy approximates a given fluid policy. First, we define the additional delay experienced by a packet due to approximating a fluid policy by a packetized one (which we refer to simply as *delay*) to be the positive part[2] of the difference between the time at which the packetized policy sends the packet and the packet's fluid completion time; if a packet experiences delay 0, we say it *experiences no delay*. Second, we define the *backlog* at a single input port to be the positive part of the difference between the cumulative fluid that has been sent from it by the fluid policy and the cumulative number of packets that have

[1] See Bennett and Zhang [6] for a discussion of the importance of the constraint that a packetized policy can only send a packet at or after that packet's fluid start time, in the context of Generalized Processor Sharing.

[2] The positive part of a vector V is denoted V^+, where $V_i^+ := \max\{V_i, 0\}$.

been sent from it by the packetized policy. The backlog for a set of input ports is the sum of backlog for each input port in the set. Backlog can be computed, at each time step t, by taking the positive part of the *cumulative difference vector* $C^{(t)} := \sum_{h=1}^{t}(F^{(h)} - P^{(h)})$. For notational convenience, we define $C^{(0)} := \mathbf{0}$, the vector with all zeros. See Figures 2-1 and 2-2 for examples of a fluid policy, a corresponding packetized policy, and their cumulative difference. For a set of input ports, we say that a packet-scheduling algorithm gets at most *b-backlogged* if for all time steps t, their backlog is at most b. In other words, for all t, the sum of corresponding entries in $(C^{(t)})^+$ is at most b. For example, we say that a packet-scheduling algorithm gets at most b-backlogged per input port if for each input port i, $C_i^{(t)} \leq b$ at all time steps t.

There are several immediate consequences of the above definitions. First, since by definition a packet-scheduling algorithm can only send a packet at or after that packet's fluid start time, a packet-scheduling algorithm can only send a packet from input port i at time t if $C_i^{(t-1)} + F_i^{(t)} > 0$. This implies that for any packet-scheduling algorithm, we have

$$\forall i, t, \quad C_i^{(t)} > -1. \tag{2.1}$$

Second, a packetized policy tracks a fluid policy if and only if for all t, and for all $i \leq N$, we have $-1 < C_i^{(t)} < 1$.

For $L : 0 \leq L < \infty$, we say that a packet-scheduling algorithm uses *lookahead* L if the set of fluid vectors it has access to in computing packetized vector $P^{(t)}$ is $\{F^{(1)}, F^{(2)}, \ldots, F^{(t+L)}\}$. We say that a packet-scheduling algorithm uses *lookahead* ∞ if it has access to all fluid vectors in computing packetized vector $P^{(t)}$. If the lookahead for a packet-scheduling algorithm is not specified, it is assumed to use lookahead 0.

In general, a server is said to use speedup $s \geq 1$ if it processes packets s times as fast as the maximum input line rate at a single input port. In this work, it will be more convenient to use a related definition; we say that a packet-scheduling algorithm uses *speedup* $s \geq 1$ when we restrict all fluid policies to consist of fluid vectors $F^{(t)}$

such that $\sum_{i=1}^{N} F_i^{(t)} \leq 1/s$. We say that an algorithm uses *no speedup* if $s = 1$. If the speedup for a packet-scheduling algorithm is not specified, it is assumed to use no speedup.

We define the *worst-case backlog* for a given switch size, lookahead, and speedup to be the minimum over all deterministic packet-scheduling algorithms of the maximum over all fluid policies of the maximum resulting backlog over all time steps. *Worst-case delay* is defined as in the previous sentence, with "backlog" replaced by "delay."

2.3 Results Overview

In the following sections, we give bounds on worst-case backlog and delay as a function of the lookahead and speedup used, and the switch size (that is, the number of input ports, N). In Section 2.4, we present results from related work. In Section 2.5, we analyze exactly how much lookahead is required to track all fluid policies when no speedup is used. Next, in Section 2.6, we give tight bounds on worst-case backlog when speedup, but no lookahead, is used. In Section 2.7, we bound worst-case packet delay as a function of the lookahead and speedup used, and the switch size; we get bounds on how much lookahead is required to track when speedup is used as a corollary to one of these bounds. In Section 2.8, we show how to translate bounds from the previous sections into bounds on queue length and packet waiting time for leaky-bucket arrival processes under the temporary session model of Andrews and Zhang [5].

2.4 Related Work

A number of authors have used fluid policies as a first step in constructing packet-switched schedules for the $N \times 1$ switch. Parekh and Gallager analyzed Generalized Processor Sharing (GPS), which creates a fluid policy that schedules fractional packets in proportion to fixed weights $w_1, \ldots, w_N$ on an $N \times 1$ switch [43, 44]. In GPS, at each time step, for S the subset of input ports whose queues are non-empty, the

fluid policy schedules $\frac{w_i}{\sum_{i' \in S} w_{i'}}$ fractional packets from input port i. The fluid policy is then approximated by a packetized policy; the packet-scheduling algorithm that constructs this packetized policy, and which uses no lookahead, is called Packet-by-Packet Generalized Processor Sharing (PGPS). PGPS, which was first proposed by Demers, Shenker, and Keshav [19] who called it Weighted Fair Queueing, determines which packet to send at each time step based on the fluid completion times resulting from GPS. PGPS first tries to determine which packet has the earliest fluid completion time among the packets that have not already been sent by the packetized policy; if it can determine this, it sends this packet. Since PGPS uses no lookahead, however, it may not be able to determine which packet has the earliest fluid completion time. In this case PGPS, at each time step t, computes an approximate fluid completion time for each packet; it sets this approximate fluid completion time to be the fluid completion time that GPS would output if no new packets arrived for time steps $> t$. It then sends the packet with the earliest approximate fluid completion time among the packets that have not already been sent by the packetized policy. Parekh and Gallager show that PGPS tracks the fluid policy created by GPS.

Bennett and Zhang [6] present another GPS-approximating, packet-scheduling algorithm called Worst-case Weighted Fair Queueing. This packet-scheduling algorithm is identical to PGPS, except that at each time step, it sends the packet with earliest approximate fluid completion time among the packets *with fluid start time at most* t that have not already been sent by the packetized policy. This algorithm ensures that no packet is sent by the packetized policy before its fluid start time, thus preventing the packetized policy from getting more than one unit ahead of the fluid policy. Bennett and Zhang prove that this packet-scheduling algorithm tracks the fluid policy created by GPS. The resulting packetized policy, however, suffers from abrupt increases or decreases in delay (known as jitter), as pointed out by Stamoulis and Liebeherr [52]. The jitter problem is addressed in [52] with a modified version of GPS, Slow-Start Generalized Processor Sharing, which creates a smoothed version of the GPS fluid policy by using time-varying weights.

In the related work described in the previous two paragraphs, 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" [21]. They point out that since GPS-emulating systems allocate resources to sessions in proportion to weights that are fixed to reflect long-term requirements of traffic flows, these systems do not meet the instantaneous quality of service needs of some applications. Duffield, Lakshman, and Stiliadis present modified algorithms for constructing fluid and packetized policies, with the aim of meeting some of these needs. First, each input port is assigned a weight w_i, where $\sum_{i=1}^{N} w_i \leq 1$. At each time step, for S the subset of input ports whose queues are non-empty, the fluid policy schedules w_i fractional packets from input port i. Note that the sum of fractional packets scheduled in a single time step may be less than one; the *excess bandwidth* at a single time step is defined as one minus this sum. Duffield, Lakshman, and Stiliadis consider algorithms that allocate excess bandwidth to input ports based on factors such as how long each packet has been waiting in its queue, and how many packets are waiting in each queue. These factors could be difficult to predict, and could change dramatically in a short period of time (e.g. due to a burst of packets). The packetized policy is constructed by scheduling packets as in PGPS, except that whenever the integer part of the cumulative excess bandwidth (over time and over all input ports) increases, a packet may instead be sent according to the factors discussed above [21].

Stamoulis and Giannakis [53] also point out the undesirability of using fixed weights to create schedules, as is done in GPS. To allow for more flexible schedules, they propose using deterministically time-varying weights at each input port. They give an efficient algorithm for building a packetized policy from a fluid policy that is known completely in advance. This packet-scheduling algorithm provides guaranteed delay bounds that, under certain assumptions, are close to those provided by the Earliest Completion Time packet-scheduling algorithm, which we define in Section 2.5.1; in that Section, we show the Earliest Completion Time algorithm gives optimal, worst-case delay bounds when infinite lookahead is used.

The use of time-varying weights in GPS and the allocation of excess bandwidth based on factors other than fixed weights result in more general, and less predictable,

fluid policies than those resulting from GPS. The desire for this added flexibility in designing fluid policies motivates the investigation of how well one can approximate, on-line, an arbitrary fluid policy by a packetized policy, in terms of minimizing delay and backlog.

Adler et al. [1] consider the problem of minimizing worst-case backlog for the $N \times 1$ switch. They show that the Largest Cumulative Difference algorithm, which we define in Section 2.6 below, is optimal in terms of minimizing the worst-case backlog over all possible sets of input ports when no lookahead and no speedup are used.

2.5 Exactly How Much Lookahead is Required to Track with No Speedup

If a packet-scheduling algorithm, when given a fluid policy as input, outputs a packetized policy that tracks it, we simply say that the packet-scheduling algorithm tracks that fluid policy. Also, if a packet-scheduling algorithm tracks all fluid policies, we simply say that it tracks. We say that an $N \times 1$ switch requires lookahead exactly L to track if the following two conditions hold:

1. There exists a packet-scheduling algorithm using lookahead L that tracks.

2. For every packet-scheduling algorithm using lookahead strictly less than L, there exists a fluid policy that it does not track.

In this section, we show for every N exactly how much lookahead an $N \times 1$ switch needs to track when no speedup is used. In Section 2.7 below, we give bounds on the necessary and sufficient lookahead to track when speedup is used (as a special case of Theorem 6).

We now prove the following theorem, which asserts the correctness of Table 2.1. In this table, an entry of ∞ indicates that for the given switch size, no packet-scheduling algorithm with finite lookahead tracks, but there is a packet-scheduling algorithm using lookahead ∞ that tracks.

Table 2.1: Exact Lookahead Required to Track on $N \times 1$ Switches with No Speedup

N:	1	2	3	4	$N \geq 5$
Lookahead:	0	0	0	1	∞

Theorem 1 *The entries in Table 2.1 indicate exactly how much lookahead is required to track for $N \times 1$ switches when no speedup is used.*

Proof: First, we show that the lookahead needed to track on the $N \times 1$ switch is a non-decreasing function of N. This follows since a packet-scheduling algorithm using lookahead L that tracks on an $N \times 1$ switch, can be converted into an algorithm using lookahead L that tracks on an $(N-1) \times 1$ switch. The algorithm for the $(N-1) \times 1$ switch first converts each fluid vector $F^{(t)} = (f_1, \ldots, f_{N-1})$ into the fluid vector $(f_1, \ldots, f_{N-1}, 0)$, which it uses as input to the $N \times 1$ packet-scheduling algorithm; the algorithm for the $(N-1) \times 1$ switch then converts each packetized vector $P^{(t)} = (p_1, \ldots, p_N)$ output from the $N \times 1$ packet-scheduling algorithm into the packetized vector $(p_1, \ldots, p_{N-1})$.

We now prove that each entry in Table 2.1 is correct. First, we exhibit a packet-scheduling algorithm that tracks on $N \times 1$ switches, using lookahead as in the table. Second, we prove corresponding lower bounds on exactly how much lookahead is required to track.

2.5.1 Upper Bounds on Lookahead Required to Track

We describe a packet-scheduling algorithm that tracks on the 3×1 switch using lookahead 0, that tracks on the 4×1 switch using lookahead 1 and that tracks on the $N \times 1$ switch for $N \geq 5$ using lookahead ∞. The packet-scheduling algorithm *Earliest Completion Time* at each time step t sends the packet with earliest, fluid completion time among the packets with fluid start time at most t that have not been sent by the packetized policy before time t. Since in general a packet-scheduling algorithm has bounded lookahead L, it may not be able to determine which packet has the earliest, fluid completion time; if for some i, $C_i^{(t-1)} + F_i^{(t)} > 0$ and for all i' such that

$C_{i'}^{(t-1)} + F_{i'}^{(t)} > 0$, we have $C_{i'}^{(t-1)} + \sum_{t'=t}^{t+L} F_{i'}^{(t')} < 1$, then the Earliest Completion Time algorithm sends a packet from input port

$$\arg\max_i \{C_i^{(t-1)} + \sum_{t'=t}^{t+L} F_i^{(t')} : C_i^{(t-1)} + F_i^{(t)} > 0\}. \tag{2.2}$$

If there are any ties between packets for earliest fluid completion time or for achieving the maximum in (2.2), then among the tied packets, the one from the input port with smallest index is sent. If for all i, $C_i^{(t-1)} + F_i^{(t)} \leq 0$, then no packet is sent by the packetized policy at time t. The Earliest Completion Time algorithm is similar to the Earliest Due Date algorithm studied in [7] and the Earliest Deadline First algorithm discussed in [53]. For an overview of results on the Earliest Deadline First scheduling algorithm, see Stankovic [54]. The optimality of the Earliest Deadline First algorithm in certain scenarios was given in 1955 by Jackson [33].

Note that under the Earliest Completion Time algorithm, for all time steps t,

$$\sum_{i=1}^{N} C_i^{(t)} \leq 0. \tag{2.3}$$

This can be shown by induction on the time step t, using the fact that a packet is sent at time step t if and only if some entry of $C^{(t-1)} + F^{(t)}$ has positive value.

The 3×1 Switch

We prove that the Earliest Completion Time algorithm, using lookahead 0 and no speedup, tracks on the 3×1 switch. The proof is by induction on the time step t. For $t = 0$, we have $C^{(0)} = \mathbf{0}$, so trivially for all $i \leq N$, we have $-1 < C_i^{(0)} < 1$. For $t \geq 0$, assume the algorithm tracks at time step t; that is, for all $i \leq N$, we have $-1 < C_i^{(t)} < 1$. By (2.1) and (2.3), there is at most one entry of $C^{(t)} + F^{(t+1)}$ with value ≥ 1, and such an entry must have value < 2 by the inductive hypothesis. If there is such an entry, Earliest Completion Time would send one packet from the corresponding input port. Thus, every entry of $C^{(t+1)}$ has value less than 1. By induction, we have that the Earliest Completion Time algorithm tracks. □

The 4×1 Switch

We prove that the Earliest Completion Time algorithm, using lookahead 1 and no speedup, tracks on the 4×1 switch. First, we restate the Earliest Completion Time algorithm for the special case of lookahead $L = 1$. Given the most recent fluid step $F^{(t)}$, it sets the corresponding packetized step $P^{(t)}$ as follows:
If there is an input port i such that $C_i^{(t-1)} + F_i^{(t)} \geq 1$, it sets to 1 the entry of $P^{(t)}$ corresponding to the least numbered such input port. Else, it finds the least numbered input port among those with positive value in $C^{(t-1)} + F^{(t)}$ (if there is one) and that has maximum value in $C^{(t-1)} + F^{(t)} + F^{(t+1)}$, and sets the corresponding entry of $P^{(t)}$ to 1. It sets the remaining entries of $P^{(t)}$ to 0.

To show that this packet-scheduling algorithm tracks, we prove by induction on the time step t that the following invariants hold for all time steps t:

1. For each input port i, $C_i^{(t)} < 1$.

2. $\sum_{1 \leq i \leq N} (C_i^{(t)})^+ < 2$. In other words, the sum of backlog over all input ports is < 2.

3. If a packet is sent from input port j in the packetized step at time t, and $C_j^{(t)} \geq 0$, then $\sum_{1 \leq i \leq N} (C_i^{(t)})^+ < 1$. In other words, if a packet is sent in a packetized step at or after its fluid completion time, the sum of backlog over all input ports is less than 1.

The base case is trivial, since $C^{(0)} = \mathbf{0}$. For the inductive step, for $t \geq 0$, assume the invariants hold for $C^{(t)}$. We will show that the choice of packetized step $P^{(t+1)}$ made by the algorithm maintains the invariants. We consider two cases:

Case 1: $\sum_{1 \leq i \leq N} (C_i^{(t)})^+ < 1$.
This case is easy, since $\sum_{1 \leq i \leq N} (C_i^{(t)} + F_i^{(t+1)})^+ < 2$. If there is an input port i' such that $C_{i'}^{(t)} + F_{i'}^{(t+1)} \geq 1$, then the Earliest Completion Time algorithm sets $P_{i'}^{(t+1)}$ to be 1; in this case, $\sum_{1 \leq i \leq N} (C_i^{(t+1)})^+$ is less than 1, so all the invariants are maintained. Otherwise, every entry of $C^{(t)} + F^{(t+1)}$ is less than 1, and so if the algorithm sends a

packet from input port j at time $t+1$, we will have $C_j^{(t+1)} < 0$; thus, regardless of the choice of $P^{(t+1)}$, all the invariants will be maintained at step $t+1$.

Case 2: $\sum_{1 \le i \le N} (C_i^{(t)})^+ \ge 1$.

By Invariant 1, and Inequalities 2.1 and 2.3, there must be exactly two entries (call them i_1, i_2) with negative values in $C^{(t)}$ and two entries (call them i_3, i_4) with positive values in $C^{(t)}$. Some packet must have been sent at step t by the packetized policy since the Earliest Completion Time algorithm always sends a packet at time t if there is a positive-valued entry in $C^{(t-1)} + F^{(t)}$. Since we are assuming $\sum_{1 \le i \le N} (C_i^{(t)})^+ \ge 1$, by Invariant 3, the packet sent at step t came from an input port whose value in $C^{(t)}$ is negative; without loss of generality, assume this input port is i_1.

Consider the case in which $C_{i_1}^{(t)} + F_{i_1}^{(t+1)} \ge 0$. By Inequalities 2.1 and 2.3, the sum of positive values in $C^{(t)} + F^{(t+1)}$ is less than 2, and so at most one of the entries in $C^{(t)} + F^{(t+1)}$ can have value ≥ 1; if there is such an entry, the packet-scheduling algorithm sets the corresponding entry in $P^{(t+1)}$ to 1, and all the invariants are maintained. Otherwise every entry in $C^{(t)} + F^{(t+1)}$ is less than 1, and so as argued at the end of Case 1 above, this guarantees all invariants are maintained at step $t+1$.

Otherwise, consider the case in which $C_{i_1}^{(t)} + F_{i_1}^{(t+1)}$ is negative. At time step t, the packet-scheduling algorithm sent a packet from input port i_1 because input port i_1 had maximum value in $C^{(t-1)} + F^{(t)} + F^{(t+1)}$ among the input ports with positive value in $C^{(t-1)} + F^{(t)}$. Thus, we have that $C_{i_3}^{(t)} + F_{i_3}^{(t+1)}$ and $C_{i_4}^{(t)} + F_{i_4}^{(t+1)}$ are at most 1 more than $C_{i_1}^{(t)} + F_{i_1}^{(t+1)}$. Then every entry in $C^{(t)} + F^{(t+1)}$ is less than 1. Thus, since we are considering the case in which $C_{i_1}^{(t)} + F_{i_1}^{(t+1)} < 0$, we have that $C^{(t+1)}$ has at most two positive-valued entries, each with value less than 1; this implies invariants 1 and 2 hold at time step $t+1$. Also, this implies that if the algorithm sends a packet from input port j at time $t+1$, we will have $C_j^{(t+1)} < 0$, and so invariant 3 holds at time step $t+1$. □

The $N \times 1$ Switch for $N \ge 5$; Infinite Lookahead

Table 2.1 indicates that infinite lookahead is both necessary and sufficient to track on the $N \times 1$ switch, for $N \ge 5$. We show in Section 2.5.2 that for any packet-scheduling

algorithm on the 5×1 switch using bounded lookahead, there exists a fluid policy it does not track. However, if the Earliest Completion Time packet-scheduling algorithm uses lookahead $L = \infty$, that is it has access to the entire fluid policy in computing each packetized step, it tracks on the $N \times 1$ switch for any N. This follows taking $L = \infty$ in the following lemma. Note that the Earliest Completion Time packet-scheduling algorithm using lookahead $L = \infty$ at each time t knows which packet has the earliest completion time among the packets with fluid start time at most t.

Lemma 1 *The Earliest Completion Time algorithm with lookahead $L \leq \infty$ and using speedup $s \geq 1$ constructs a packetized policy such that every packet with fluid completion time at most L experiences no delay.*

Proof: We show the following claim, which implies the lemma, by induction on t: For all (positive-integer valued) time steps $t \leq L$, all packets with fluid completion time at most t are sent by the packetized policy by time step t. The base case $t = 1$ follows since at most one packet is completed by the fluid policy at time $t = 1$, and such a packet would be sent at that time by the packetized policy. For the inductive step, assume for all time steps $< t$ that the claim is true.

If the packetized policy does not send a packet at time t with fluid completion time t, then no packet is completed by the fluid policy at time step t that hadn't already been sent at an earlier time by the packetized policy (since otherwise the Earliest Completion Time algorithm would have scheduled such a packet at time t). In this case, by the inductive hypothesis, the claim holds at time step t.

Otherwise, we have that the packetized policy sends a packet at time step t that has fluid completion time t. Let t' be the greatest time step $< t$ at which the packetized policy did not send a packet with fluid completion time $\leq t$; if no such time step exists, set $t' := 0$. Then by the inductive hypothesis, at each time step in $[t' + 1, t]$, the packetized policy sends a packet with fluid completion time in $[t' + 1, t]$. Let P denote the set of packets completed by the fluid policy during time steps $[t' + 1, t]$, but not sent by the packetized policy before time t'. We now show that all packets in P are sent by the packetized policy by time t, which implies the claim at time step

t. Each packet in P has fluid start time greater than t', since otherwise the Earliest Completion Time algorithm using lookahead $L \geq t$ would have sent a packet at time t' with fluid completion time $\leq t$, contradicting our choice of t'. Thus, the number of packets in P is at most $\lfloor (t - t')/s \rfloor$, the total available fluid that could be scheduled during $[t'+1, t]$. But since one packet from P is sent by the packetized policy at every time step in $[t'+1, t]$, we have that all packets in P are sent by the packetized policy by time t. This implies that all packets completed by the fluid policy at or before time step t are sent by the packetized policy at or before t, completing the induction and proving the lemma. □

Note that the same argument as above can be used to prove that if for some time step $t \geq 0$, each entry of the cumulative difference vector $C^{(t)}$ has non-positive value, then the Earliest Completion Time algorithm using lookahead $L < \infty$ constructs a packetized policy such that every packet with fluid completion time in $[t+1, t+L]$ experiences no delay.

An immediate consequence of the lemma above is that the Earliest Completion Time algorithm tracks any constant fluid policy (as long as the algorithm knows in advance that the fluid policy will be constant).

2.5.2 Lower Bounds on Exactly How Much Lookahead is Required to Track

Recall the second part of the definition of requiring lookahead exactly L from the beginning of this section :
For every packet-scheduling algorithm using lookahead strictly less than L, there exists a fluid policy that it does not track.

To prove lower bounds on exactly how much lookahead is needed to track that correspond to Table 2.1, we first construct, for any packet-scheduling algorithm using lookahead 0 and no speedup, a fluid policy that it does not track on the 4×1 switch. Next, we exhibit for any $L : 0 \leq L < \infty$ and any packet-scheduling algorithm using lookahead L and no speedup, a fluid policy it does not track on the 5×1 switch.

Fluid Policy $F^{(t)}$		Packetized Policy $P^{(t)}$		Cumulative Difference $C^{(t)}$
$\begin{bmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \end{bmatrix}$	,	$\begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$	,	$\begin{bmatrix} -\frac{3}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \end{bmatrix}$
$\begin{bmatrix} 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix}$	,	$\begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}$	,	$\begin{bmatrix} -\frac{3}{4} & \frac{7}{12} & \frac{7}{12} & -\frac{5}{12} \end{bmatrix}$
$\begin{bmatrix} 0 & \frac{1}{2} & \frac{1}{2} & 0 \end{bmatrix}$	,	$\begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}$	,	$\begin{bmatrix} -\frac{3}{4} & \frac{13}{12} & \frac{1}{12} & -\frac{5}{12} \end{bmatrix}$

Figure 2-1: An example of an adversarial fluid policy for the 4×1 switch.

The 4×1 Switch

We construct, for any packet-scheduling algorithm on the 4×1 switch using no lookahead and no speedup, a fluid policy that it does not track. The fluid policy depends on the packet-scheduling algorithm under consideration. The fluid policy first sends 1/4 (of a packet) from each input port. At time step 2, the fluid policy sends 1/3 from every input port except the one from which the packetized policy sent a packet at time 1.[3] At time step 3, the fluid policy sends 1/2 from every input port that received 1/3 unit of fluid at time step 2 except the one from which the packetized policy sent a packet at time 2.[4] Now, there is some input port that has been scheduled a total of $1/2+1/3+1/4 > 1$ fluid, but from which no packet has been sent by the packetized policy, showing that this packet-scheduling algorithm does not track. Thus, no packet-scheduling algorithm using no lookahead and no speedup can track every fluid policy. □

See Figure 2-1 for an example of such an adversarial fluid policy. In Section 2.6.1, we extend the argument given here to $N \times 1$ switches when speedup but no lookahead is used, to give tight lower bounds on backlog.

The 5×1 Switch

We want to build, for any $L < \infty$ and any packet-scheduling algorithm using lookahead L and no speedup, a fluid policy that it does not track on the 5×1 switch. This will follow from the proof of the first half of the more general Theorem 4 in

[3]If no packet was sent by the packetized policy at time 1, then 1/3 is sent from each of the first three input ports.

[4]If no packet was sent by the packetized policy at time 2, then 1/2 is sent from two of the three input ports from which 1/3 was sent at time step 2.

Section 2.7.1; see footnote 7 in Section 2.7.1.

The proof that the entries in Table 2.1 are correct is complete. □

2.6 Tight Bounds on Backlog with No Lookahead

The results in this section on backlog with no lookahead were proved independently in a recent paper by Adler et al. [1], for the case when no speedup is used. Recall the definition of backlog: for a set of input ports, at time t, their corresponding backlog is the positive part of the difference between the cumulative fluid sent from these ports by the fluid policy up to and including time t, and the cumulative number of packets sent from these ports by the packetized policy up to and including time t.

We show that the Largest Cumulative Difference packet-scheduling algorithm, defined next, is optimal in terms of minimizing the worst-case backlog over all possible sets of input ports when no lookahead is used. The *Largest Cumulative Difference* algorithm constructs a packetized policy that at time t sends a packet from input port $\arg\max_i \left(C_i^{(t-1)} + F_i^{(t)}\right)$. If for all i, $C_i^{(t-1)} + F_i^{(t)} \leq 0$, then the Largest Cumulative Difference algorithm doesn't send from any input port at time t. Else, if several input ports achieve the maximum $\max_i \left(C_i^{(t-1)} + F_i^{(t)}\right)$, then among these input ports, a packet from the smallest numbered one is sent.

First, we prove that for every packet-scheduling algorithm using speedup $s \geq 1$ and no lookahead, there exists an adversarial fluid policy that gives a lower bound linear in N/s on the sum of backlog over all input ports and a lower bound of $\frac{1}{s}(\ln(N+1)-1)$ on the backlog of some single input port. Next, we show that the Largest Cumulative Difference algorithm using speedup s and no lookahead achieves both lower bounds.

2.6.1 Lower Bounds on Backlog

We design an adversarial fluid policy for a given packet-scheduling algorithm using speedup s and no lookahead[5] by, at each time step, scheduling fluid equally among a subset of the input ports from which the algorithm's packetized policy has never sent.

In addition to keeping track at each time step t of the fluid vector $F^{(t)}$, packetized vector $P^{(t)}$, and cumulative difference vector $C^{(t)}$, we keep track of a subset of input ports $I_t \subseteq \{1, 2, ..., N\}$. I_t will include all the input ports sent from by the packetized policy up to and including time step t; however, I_t may contain additional input ports. Also, we will maintain for all $t \geq 0$ that

$$I_t \subseteq I_{t+1} \tag{2.4}$$

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

Given a packet-scheduling algorithm using speedup s and no lookahead, we iteratively construct an adversarial fluid policy $\{F^{(t)}\}$, for $t \in \{1, 2, ..., N-1\}$ as follows:

First, we set $I_0 := \emptyset$.

At each iteration $t \geq 1$, we define

$$F^{(t)} := \frac{1}{s(N-t+1))}([1, 1, ..., 1] - \chi^{I_{t-1}}).$$

Then we run the packet-scheduling algorithm to determine which input port j, if any, the packetized policy sends a whole packet from at time step t. If no packet is sent or if $j \in I_{t-1}$, we choose the least $i \geq 1$ not in I_{t-1}, and set $I_t := I_{t-1} \cup \{i\}$. Otherwise, we set $I_t := I_{t-1} \cup \{j\}$. An example of such an adversarial fluid policy for the 4×1 switch is given in Figure 2-1.

Let H_m denote the mth harmonic number; that is, $H_0 := 0$, and for $m \geq 1$, we have $H_m := \sum_{j=1}^{m} 1/j$. Having constructed $\{F^{(t)}\}_{1 \leq t \leq N-1}$, we see that for any

[5]We generalize this adversarial fluid policy to packet-scheduling algorithms using lookahead in Section 2.7.2.

$m : 1 \leq m \leq N$, immediately after time step $N - m$, the m input ports not in I_{N-m} each have cumulative difference equal to $\frac{1}{s}(H_N - H_m)$. Thus, the backlog of the set of m most-backlogged input ports is $\frac{m}{s}(H_N - H_m)$ immediately after time step $N - m$.

Since we will refer to this function often in the next section, we define for fixed N, fixed $s \geq 1$, and $m : 1 \leq m \leq N$,

$$g(m) := \frac{m}{s}(H_N - H_m).$$

In Appendix A, we prove that for $m : 1 \leq m \leq N$,

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

We have proved the following theorem:

Theorem 2 *For the $N\times 1$ switch, for every packet-scheduling algorithm using speedup $s \geq 1$ and no lookahead, there exists an adversarial fluid policy with the following property: For any $m : 1 \leq m \leq N$, there exists a time step $t \leq N - 1$ and a set I of m input ports such that*

$$\sum_{i\in I} C_i^{(t)} = g(|I|). \tag{2.5}$$

In particular, at some time step the sum of backlog over all input ports is more than $\frac{1}{s}((N+1)/e - 2)$. Also, at some time step there is a single input port with backlog more than $\frac{1}{s}(\ln(N+1) - 1)$.

We now turn to a converse, which in light of the above theorem shows optimality of the Largest Cumulative Difference algorithm for minimizing worst case backlog when no lookahead is used.

2.6.2 Upper Bounds on Backlog

We prove the following theorem for the Largest Cumulative Difference algorithm:

Theorem 3 *For the $N \times 1$ switch and for any fluid policy, the Largest Cumulative Difference packet-scheduling algorithm using speedup $s \geq 1$ produces a packetized policy satisfying the following set of inequalities: For all t and for any non-empty set I of input ports, we have*

$$\sum_{i \in I} C_i^{(t)} \leq g(|I|). \tag{2.6}$$

This implies for the Largest Cumulative Difference packet-scheduling algorithm, that the sum of backlog over all input ports is at most $N/(es)$, and the backlog for each input port is at most $\frac{1}{s} \ln N$.

Proof: We show by induction that the Largest Cumulative Difference algorithm for constructing a packetized policy from a fluid policy maintains (2.6) at each time step. Let I be any non-empty subset of $\{1, 2, ..., N\}$.

The set of inequalities (2.6) are true initially since $C^{(0)} = \mathbf{0}$ and for all non-empty I, $g(|I|) \geq 0$.

For the inductive step, assume that (2.6) holds at time step t. We show it holds for time step $t+1$ regardless of the fluid policy's choice of $F^{(t+1)}$. We set $P := P^{(t+1)}, C := C^{(t)}$, and $F := F^{(t+1)}$ for clarity of exposition.

If for all $i, C_i + F_i \leq 0$ then we are done. Otherwise, the Largest Cumulative Difference algorithm chooses the $N \times 1$ matrix P to be all 0's except at entry $l := \arg\max_i (C_i + F_i)$, at which P has value 1. We consider two cases, depending on whether $l \in I$ or not.

Case 1: $l \in I$

Since $\sum_{i=1}^{N} F_i \leq 1/s \leq 1$, we have:

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

Since the left hand side equals the sum over input ports $i \in I$ of the cumulative difference at time step $t+1$, this proves (2.6) in this case.

Case 2: $l \notin I$

We have the following inequalities:

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

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 $\sum_{i=1}^{N} F_i \leq 1/s$. Since for any $m : 1 \leq m \leq N$, we have
$m(g(m+1)+1/s) = (m+1)g(m)$, we get from the above inequality that

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

Since the left hand side equals the sum over input ports $i \in I$ of the cumulative difference at time step $t+1$, this proves (2.6) in this case.

The induction is complete. □

2.7 Bounds on Delay

We analyze the trade-offs between speedup and lookahead required to bound worst-case delay for time-varying fluid policies. Before delving into the proofs, we present our main results, which are summarized in Table 2.2, and discuss them briefly. The first result bounds packet delay for packet-scheduling algorithms using bounded lookahead but no speedup.

Theorem 4 *For any packet-scheduling algorithm that uses bounded lookahead and no speedup on an $N \times 1$ switch, there exists a fluid policy causing a packet to be delayed at least $\lfloor N/e \rfloor - 1$ steps. Conversely, the Earliest Completion Time algorithm using no lookahead and no speedup, guarantees that each packet is delayed at most $\lfloor N/e \rfloor$*

steps.

A surprising consequence of the above theorem is that the delay bounds do not depend on the amount of lookahead used, as long as it is bounded. In particular, the Earliest Completion Time algorithm using no lookahead achieves a bound on delay within one time step of the lower bound. Another consequence of Theorem 4 is that for 6 or more input ports, even with arbitrary, bounded lookahead, there exist fluid policies that cannot be tracked with no speedup.[6] The next result bounds packet delay when speedup is allowed, but with no lookahead.

Theorem 5 *The Earliest Completion Time algorithm, on an $N \times 1$ switch using speedup $s \geq 1$ but no lookahead, guarantees that each packet is delayed less than N/e^s steps. Furthermore, for any packet-scheduling algorithm using speedup s and no lookahead, there exists an adversarial fluid policy causing a packet to have delay at least $\lfloor (N+1)/e^s \rfloor - 1$.*

A consequence of Theorem 5 is that with no lookahead, speedup $\max\{\ln N, 1\}$ is sufficient to track, while speedup greater than $\ln(N+1) - \ln 2$ is necessary to track. Lastly, we analyze algorithms using both lookahead and speedup.

Theorem 6 *For any packet-scheduling algorithm using speedup $s > 1$ and lookahead $L \geq 0$ there is a fluid policy causing some packet to be delayed at least $(\lfloor (N+1)/e^s - 1 \rfloor - L)^+$ time steps. Also, the Earliest Completion Time algorithm using speedup $s > 1$ and lookahead $L \geq 0$ causes each packet to be delayed at most $(\lceil (N+1)s/(s-1) \rceil - L)^+$ time steps.*

Theorem 6 implies that for speedup $s > 1$, the necessary and sufficient lookahead to track is at least $\lfloor (N+1)/e^s - 1 \rfloor$ and at most $\lceil (N+1)s/(s-1) \rceil$.

For fixed speedup and delay bound, Theorem 6 can be rephrased to give lower and upper bounds linear in N on the necessary and sufficient lookahead to meet the delay bound:

[6] In the proof of Theorem 4, we show the same holds for 5 input ports.

Table 2.2: Delay Bounds for $N \times 1$ Switches as a Function of Speedup and Lookahead Used. A lower bound of $f(N, s, L)$ means that for any packet-scheduling algorithm using speedup s and lookahead L on an $N \times 1$ switch, there exists a fluid policy that causes a packet to be delayed at least $f(N, s, L)$ steps. An upper bound of $g(N, s, L)$ means that the Earliest Completion Time packet-scheduling algorithm using speedup s and lookahead L on an $N \times 1$ switch guarantees delay at most $g(N, s, L)$ for any fluid policy.

Speedup s	Lookahead L	Lower Bound on Delay	Upper Bound on Delay
None $(s = 1)$	$L < \infty$	$\lfloor N/e \rfloor - 1$	$\lfloor N/e \rfloor$
$s \geq 1$	None $(L = 0)$	$\lfloor (N+1)/e^s \rfloor - 1$	$\lfloor (N+1)/e^s \rfloor$
$s > 1$	$L < \infty$	$\lfloor (N+1)/e^s - 1 \rfloor - L$	$(\lceil (N+1)s/(s-1) \rceil - L)^+$

Corollary: For speedup $s > 1$ and delay bound $d \geq 0$, lookahead at least $\lfloor (N+1)/e^s - 1 \rfloor - d$ is required to meet this delay bound. Also, for speedup $s > 1$, and delay bound $d \geq 0$, the Earliest Completion Time algorithm using lookahead $\lceil (N+1)s/(s-1) \rceil - d$ causes each packet to be delayed at most d time steps.

However, the gap between these linear bounds can grow large as a function of speedup s.

We now prove Theorems 4, 5, and 6.

2.7.1 Proof of Theorem 4: Bounds on Delay using Finite Lookahead, No Speedup

We start by proving the first half of Theorem 4, which states that for any packet-scheduling algorithm that uses bounded lookahead and no speedup on an $N \times 1$ switch, there exists a fluid policy causing a packet to be delayed at least $\lfloor N/e \rfloor - 1$ steps.

Proof: For any given packet-scheduling algorithm that uses bounded lookahead L and no speedup, we design an adversarial fluid policy that forces some packet to be delayed by at least $\lfloor N/e \rfloor - 1$ steps. Since for $N \leq 5$ designing such an adversarial fluid policy is trivial, we assume $N \geq 6$. In each step of this fluid policy, either fluid is scheduled equally across a subset of the first $N - 1$ input ports or one unit of fluid is sent from input port N. For each $j \geq 0$, we keep track of a subset I_j of input ports

from $\{1, ..., N-1\}$; the subset I_{j-1} will be used to determine the input ports that will not have fluid scheduled from them at time step $t_j := j + (j-1)L$, for $j \geq 1$. For notational convenience, we set $t_0 := 0$.

We maintain for all $j \geq 0$, that $|I_j| = j$ and $I_j \subseteq I_{j+1}$. The $N \times 1$ incidence vector χ^{I_j} has value 1 for all components in I_j and has value 0 elsewhere.

Given a packet-scheduling algorithm, we iteratively construct the adversarial fluid policy $\{F^{(t)}\}$, for $t \in \{1, 2, ..., t_{\lceil N-N/e \rceil}\}$.

For each $j \geq 1$, we define for time step t_j,

$$F^{(t_j)} := \frac{1}{N-j}([1, 1, ..., 1, 0] - \chi^{I_{j-1}}).$$

For all other time steps t, we set $F^{(t)} := [0, 0, ..., 0, 1]$. It remains to describe how I_j for $j \geq 0$ are determined.

We set $I_0 := \emptyset$. For each $j \geq 1$, we run the packet-scheduling algorithm to determine from which input ports the packetized policy sends packets during time steps $t_{j-1}+1$ through t_j. If there exists an input port in $\{1, ..., N-1\} \setminus I_{j-1}$ from which the packetized policy has sent at least one packet at some time step in $[1, t_j]$, we set $I_j := I_{j-1} \cup \{d_j\}$, where d_j is some such input port. Otherwise, we set $I_j := I_{j-1} \cup \{d_j\}$, where d_j is some input port in $\{1, ..., N-1\} \setminus I_{j-1}$. Note that the packetized policy can be generated through time step t_j before d_j has been specified. This follows since the choice of d_j is not revealed in the fluid policy until time step $t_{j+1} = t_j + L + 1$, and the algorithm has lookahead bounded by L. An example of such an adversarial fluid policy for $N = 6$ input ports with lookahead $L = 1$ is given in Figure 2-2. Having specified how the fluid policy is constructed, we show below that it forces some packet to be delayed at least $\lfloor N/e \rfloor - 1$ time steps.

Denote by R_j the subset of input ports $\{1, ..., N-1\}$ from which at least one packet has been sent by the packetized policy at or before time step t_j. Then we have the following lemma:

Lemma 2 *For all $j \geq 0$, $|R_j \setminus I_j| \leq C_N^{(t_j)}$.*

Fluid Policy $F^{(t)}$							Packetized Policy $P^{(t)}$							Cumulative Difference $C^{(t)}$						
$[\mathbf{\frac{1}{5}}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$0]$	,	$[1$	0	0	0	0	$0]$	,	$[-\frac{4}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$0]$
$[0$	0	0	0	0	$1]$	,	$[0$	0	0	0	0	$1]$	,	$[-\frac{4}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$0]$
$[\mathbf{0}$	$\mathbf{\frac{1}{4}}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$0]$	,	$[0$	1	0	0	0	$0]$	,	$[-\frac{4}{5}$	$-\frac{11}{20}$	$\frac{9}{20}$	$\frac{9}{20}$	$\frac{9}{20}$	$0]$
$[0$	0	0	0	0	$1]$	,	$[0$	0	1	0	0	$0]$	,	$[-\frac{4}{5}$	$-\frac{11}{20}$	$-\frac{11}{20}$	$\frac{9}{20}$	$\frac{9}{20}$	$1]$
$[\mathbf{0}$	$\mathbf{0}$	$\frac{1}{3}$	$\frac{1}{3}$	$\mathbf{\frac{1}{3}}$	$0]$	,	$[0$	0	0	0	1	$0]$	,	$[-\frac{4}{5}$	$-\frac{11}{20}$	$-\frac{13}{60}$	$\frac{47}{60}$	$-\frac{13}{60}$	$1]$
$[0$	0	0	0	0	$1]$	,	$[0$	0	0	0	0	$1]$	,	$[-\frac{4}{5}$	$-\frac{11}{20}$	$-\frac{13}{60}$	$\frac{47}{60}$	$-\frac{13}{60}$	$1]$
$[\mathbf{0}$	$\mathbf{0}$	$\mathbf{\frac{1}{2}}$	$\frac{1}{2}$	$\mathbf{0}$	$0]$	,	$[0$	0	0	0	0	$1]$	,	$[-\frac{4}{5}$	$-\frac{11}{20}$	$\frac{17}{60}$	$\frac{77}{60}$	$-\frac{13}{60}$	$0]$

Figure 2-2: An example of a fluid policy for a single server with $N = 6$ input ports, where the rows correspond to successive time steps. The first packet from the sixth input port is completed at time step 2 by both the fluid and packetized policies, and thus has no delay. The second packet from the sixth input port is completed by the fluid policy at time step 4 but not completed by the packetized policy until time step 6, so its delay is 2. The boldface numbers in the fluid policy represent the elements of I_j, defined in the proof of Theorem 4, at time step t_j.

Proof: We prove the claim by induction on j. The base case $t_0 = 0$ follows trivially. For the inductive step, assume the inequality in the lemma holds for some $j \geq 0$.

Denote by R'_{j+1} the set of input ports in $\{1, ..., N-1\}$ from which at least one packet has been sent by the packetized policy at or before time step $t_{j+1} - 1$. By the inductive hypothesis ($|R_j \setminus I_j| \leq C_N^{(t_j)}$) and the fact that for time steps $t \in [t_j + 1, t_{j+1} - 1]$ the fluid vector $F^{(t)} = [0, 0, ..., 0, 1]$, we have $|R'_{j+1} \setminus I_j| \leq C_N^{(t_{j+1}-1)}$.

If $R_{j+1} \setminus I_{j+1} = \emptyset$, then $|R_{j+1} \setminus I_{j+1}| \leq C_N^{(t_{j+1})}$, since for all $t', C_N^{(t')} \geq 0$. The latter inequality holds since only whole units of fluid are ever added or subtracted from input port N, and packet-scheduling algorithms are only allowed to send a packet at or after its fluid start time. Thus, we need only consider the situation $R_{j+1} \setminus I_{j+1} \neq \emptyset$, which implies that an input port from $R_{j+1} \setminus I_j$ is added to the set I_j to form I_{j+1}. If the packetized policy at step t_{j+1} sends a packet from input port N, we have

$$|R_{j+1} \setminus I_{j+1}| = |R'_{j+1} \setminus I_j| - 1 \leq C_N^{(t_{j+1}-1)} - 1 = C_N^{(t_{j+1})}.$$

Else, we have

$$|R_{j+1} \setminus I_{j+1}| \leq |R'_{j+1} \setminus I_j| \leq C_N^{(t_{j+1}-1)} = C_N^{(t_{j+1})}.$$

For both the above cases, we see that the inequality in the lemma holds for $j+1$, completing the induction and the proof of Lemma 2. □

For any $j \geq 1$, just after time step t_j, the $N-1-j$ input ports in $\{1, ..., N-1\} \setminus I_j$ have each been scheduled a total of exactly $H_{N-1} - H_{N-1-j}$ fluid.[7] For $j' := \lceil N - N/e \rceil$, we have that by just after time step $t_{j'}$, each of the $N-1-j'$ input ports in $\{1, ..., N-1\} \setminus I_{j'}$ have been scheduled a total of exactly $H_{N-1} - H_{N-1-j'} \geq \ln \frac{N}{N-j'} \geq 1$ fluid. The subset of these input ports that the packetized policy has not sent any packets from at or before time step $t_{j'}$ is exactly $\{1, ..., N-1\} \setminus (R_{j'} \cup I_{j'})$. Thus, each input port in $\{1, ..., N-1\} \setminus (R_{j'} \cup I_{j'})$ has a packet completed by the fluid policy but not yet sent by the packetized policy by just after time step $t_{j'}$. Using Lemma 2, we get that the number of packets completed by the fluid policy but not yet sent by the packetized policy at or before time step $t_{j'}$ is at least

$$|\{1, ..., N-1\} \setminus (R_{j'} \cup I_{j'})| + C_N^{(t_{j'})} \geq N - 1 - j' = \lfloor N/e \rfloor - 1.$$

One of these packets will be delayed by at least $\lfloor N/e \rfloor - 1$ time steps, proving Theorem 4.[8] □

In the next section, we prove the first claim in Theorem 5, which yields the converse in Theorem 4 as a special case.

2.7.2 Proof of Theorem 5: Bounds on Delay using Speedup, but No Lookahead

We now prove the first part of Theorem 5, which says that the Earliest Completion Time algorithm, using speedup $s \geq 1$ but no lookahead, guarantees that each packet

[7] For the special case of $N = 5$ and $j = 3$, we have that the single input port in $\{1, 2, 3, 4\} \setminus I_3$ has been scheduled a total of $1/4 + 1/3 + 1/2 > 1$ fluid by just after time t_3. Either $C_N^{(t_3)} \geq 1$, or by Lemma 2 the packetized policy has not sent any packets from the input port in $\{1, 2, 3, 4\} \setminus I_3$ by just after time t_3. In either case, we see that the packet-scheduling algorithm does not track. This proves that for the 5×1 switch, for any packet-scheduling algorithm using bounded lookahead and no speedup, there exists a fluid policy that cannot be tracked, as claimed in Section 2.5.2.

[8] The gap of 1 between the upper and lower bounds on worst-case packet delay from Theorem 4 cannot in general be improved; in particular, in Section 2.5 we showed for the case of $N = 4$ input ports that with lookahead $L = 1$ the worst-case delay is 0, but with no lookahead ($L = 0$) the worst-case delay is 1.

is delayed less than N/e^s steps. It will be convenient to refer, at a given time step, to the packets that have been completed by the fluid policy but not sent by the packetized policy by this time step; we call these the *urgent packets*. The idea of the proof is to upper bound the number of urgent packets at any time step, since the delay experienced by a packet under the Earliest Completion Time algorithm is at most the number of urgent packets at this packet's fluid completion time. We first show that with speedup $s \geq 1$ and no lookahead, the number of urgent packets at any time step resulting from the Earliest Completion Time packet-scheduling algorithm is identical to the number of urgent packets resulting from the Largest Cumulative Difference packet-scheduling algorithm (which was defined in Section 2.6). We then use a result from Section 2.6.2 to help prove that the number of urgent packets is always less than N/e^s for this latter packet-scheduling algorithm.

Lemma 3 *When operating on the same fluid policy and using speedup $s \geq 1$ and no lookahead, at each time step the Earliest Completion Time and Largest Cumulative Difference algorithms result in exactly the same number of urgent packets.*

Proof: We use C and $\hat{C}$ to denote the cumulative differences resulting from the Largest Cumulative Difference and Earliest Completion Time algorithms, respectively. For clarity, we set $C := C^{(t)}$, $\hat{C} := \hat{C}^{(t)}$, and $F := F^{(t+1)}$. Then the number of urgent packets at time step t resulting from the Largest Cumulative Difference algorithm is $\sum_{k:C_k \geq 1} \lfloor C_k \rfloor$, and the number of urgent packets resulting from the the Earliest Completion Time algorithm is $\sum_{k:\hat{C}_k \geq 1} \lfloor \hat{C}_k \rfloor$. Note that if the set of input ports $\{k : C_k + F_k \geq 1\}$ is non-empty, then the Largest Cumulative Difference algorithm sends a packet from an input port in this set; else, if this set is empty and for some k, $C_k + F_k > 0$, it sends a packet from $\arg\max_k (C_k + F_k)$; otherwise if $\forall k, C_k + F_k \leq 0$, it doesn't send any packet. The same is true of the Earliest Completion Time algorithm, with C replaced by $\hat{C}$. We show below that since both algorithms have the above property, we will have for all time steps $\sum_{k:C_k \geq 1} \lfloor C_k \rfloor = \sum_{k:\hat{C}_k \geq 1} \lfloor \hat{C}_k \rfloor$.

First, since at each time step the same fluid vector is fed to both algorithms, and only whole (unit size) packets are sent, we have that for each input port k the

fractional part of C_k and of $\hat{C}_k$ are equal. The following, two-part claim together with the previous fact implies the lemma.

Claim: For all time steps,

$$\forall k, \left(\text{If } C_k < 0 \text{ or } \hat{C}_k < 0, \text{ then } C_k = \hat{C}_k\right). \tag{2.7}$$

$$\sum_{k=1}^{N} C_k = \sum_{k=1}^{N} \hat{C}_k. \tag{2.8}$$

We prove the claim by induction on the time step t. The base case $t = 0$ follows since $C^{(0)} = \hat{C}^{(0)} = \mathbf{0}$. For the inductive step, assume the claim holds at the current time step t. To prove (2.7) holds at the next time step, we first note that (2.7) of the inductive hypothesis implies

$$\{k : C_k + F_k \geq 0\} = \{k : \hat{C}_k + F_k \geq 0\}.$$

By (2.7) and (2.8) of the inductive hypothesis, we then have

$$\sum_{k:C_k+F_k\geq 0} C_k + F_k = \sum_{k:\hat{C}_k+F_k\geq 0} \hat{C}_k + F_k.$$

Since for all k the fractional part of $C_k + F_k$ and of $\hat{C}_k + F_k$ are equal, the previous equation implies

$$\sum_{k:C_k+F_k\geq 0} \lfloor C_k + F_k \rfloor = \sum_{k:\hat{C}_k+F_k\geq 0} \lfloor \hat{C}_k + F_k \rfloor.$$

Thus, either there exist k, k' such that $C_k + F_k \geq 1$ and $\hat{C}_{k'} + F_{k'} \geq 1$, or else for all j, $C_j + F_j = \hat{C}_j + F_j$. In the former case, the Largest Cumulative Difference and Earliest Completion Time algorithms send packets from input ports i, i' respectively, such that $C_i + F_i \geq 1$ and $\hat{C}_{i'} + F_{i'} \geq 1$; this implies (2.7) holds at time step $t+1$. In the latter case, both packet-scheduling algorithms behave identically, and thus (2.7) holds at time step $t+1$. Equation (2.8) holds at the next time step since either both algorithms send a packet or neither does. The induction is complete and the Claim above is proved.

Now Lemma 3 follows easily from the Claim just proved. By (2.8) and the fact that for each input port k the fractional part of C_k and of $\hat{C}_k$ are equal, we have $\sum_{k=1}^{N} \lfloor C_k \rfloor = \sum_{k=1}^{N} \lfloor \hat{C}_k \rfloor$. From this equation and (2.7), we have $\sum_{k:C_k \geq 0} \lfloor C_k \rfloor = \sum_{k:\hat{C}_k \geq 0} \lfloor \hat{C}_k \rfloor$, which implies Lemma 3. □

Next, we prove that the Largest Cumulative Difference algorithm always results in less than N/e^s urgent packets. We use Inequality 2.6 proved in Section 2.6.2, which bounds the cumulative difference under this packet-scheduling algorithm. We repeat this inequality here. For all time steps, and for any non-empty set of input ports $I \subseteq \{1, \ldots, N\}$, we have

$$\sum_{i \in I} C_i \leq |I|(H_N - H_{|I|})/s.$$

We can upper bound the number of urgent packets by analyzing this inequality for I the set of input ports with at least one urgent packet (that is, $I := \{i : C_i \geq 1\}$). If $I = \emptyset$, the number of urgent packets would be 0, so we need only consider the case in which $I \neq \emptyset$. Note that $I \neq \emptyset$ and the inequality above imply $H_N - 1 \geq s$. We have

$$|I| \leq \sum_{i:C_i \geq 1} \lfloor C_i \rfloor \leq |I|(H_N - H_{|I|})/s. \tag{2.9}$$

For $M := \max\{j : H_N - H_j \geq s\}$, we have from the previous inequalities that $|I| \leq M$. In the second section of Appendix A, we give straightforward proofs of the following three claims:

1. $M < N/e^s$.

2. For fixed N and s, the function $g(m) := \frac{m}{s}(H_N - H_m)$
   is increasing for $m : 1 \leq m \leq M$.

3. $g(M) < M + 1$.

Then we have

$$\sum_{i:C_i \geq 1} \lfloor C_i \rfloor \leq g(M) < M + 1.$$

where the first inequality follows from (2.9) and Claim 2 above. Thus, by Claim 1 above, the number of urgent packets resulting from the Largest Cumulative Difference algorithm is less than N/e^s. By Lemma 3, N/e^s is also a strict upper bound on the number of urgent packets resulting from the Earliest Completion Time algorithm. This proves the first part of Theorem 5.

We now analyze a generalization of the adversarial fluid policy constructed in Section 2.6.1 in order to prove the converse in Theorem 5 and the first part of Theorem 6. Assume we are given a packet-scheduling algorithm using speedup $s \geq 1$ and lookahead $L < \infty$. For any $d : 0 \leq d < N/e^s$,[9], we define for each $t \geq 1$, (where the sets of input ports I_t are defined exactly as in Section 2.6.1):

$$F^{(t)} := \begin{cases} \frac{1}{Ns}[1, 1, ..., 1], & \text{if } t \leq L \\ \frac{1}{s(N-(t-L-1))}([1, 1, ..., 1] - \chi^{I_{t-L-1}}), & \text{otherwise.} \end{cases}$$

We have just after time step $N - d$ that the cumulative difference in each of the input ports in $\{1, 2, ..., N\} \setminus I_{N-d}$ is

$$\begin{cases} \frac{N-d}{Ns}, & \text{for } N - d \leq L \\ \frac{L}{Ns} + \frac{H_N - H_{d+L}}{s}, & \text{otherwise} \end{cases} . \tag{2.10}$$

Now define $r(N, s, d) := \lfloor (N + 1)/e^s - d - 1 \rfloor$.

If $r(N, s, d) \geq L \geq 0$, then the cumulative difference given in (2.10) equals or exceeds 1. In this case, at least d packets are completed by the fluid policy but not sent by the packetized policy by just after time step $N - d$. For fixed N and s, we have $r(N, s, d) \geq 0$ for $d \leq (N+1)/e^s - 1$, which proves for any packet-scheduling algorithm using speedup s and no lookahead, there exists an adversarial fluid policy causing a packet to have delay at least $\lfloor (N + 1)/e^s - 1 \rfloor$. This is the converse in Theorem 5.

[9] As shown earlier in this section, the Earliest Completion Time algorithm using speedup $s \geq 1$ and no lookahead guarantees each packet is delayed less than N/e^s steps. Thus, if $d \geq N/e^s$, then we have no hope of constructing an adversarial fluid policy for this particular algorithm that causes a packet to be delayed more than d steps.

For fixed speedup $s \geq 1$ and lookahead $L < \infty$, we have for $d = \lfloor (N+1)/e^s - 1 \rfloor - L$ that $r(N, s, d) = L$. Thus, some packet experiences delay at least $\lfloor (N+1)/e^s - 1 \rfloor - L$. This proves the first part of Theorem 6.

2.7.3 Proof of Theorem 6: Bounds on Lookahead, Speedup to Guarantee Low Delay

Since the first part of Theorem 6 was shown to follow from the analysis in the previous section, we now turn to the second part of Theorem 6, which states that the Earliest Completion Time algorithm using speedup $s > 1$ lookahead $L < \infty$ guarantees that no packet is delayed more than $(\lceil (N+1)s/(s-1) \rceil - L)^+$ time steps. That is, for any given fluid policy, this algorithm constructs a packetized policy that sends every packet at most $(\lceil (N+1)s/(s-1) \rceil - L)^+$ steps after the time that it is completed by the fluid policy.

We first prove the following lemma:

Lemma 4 *For any fluid policy, the Earliest Completion Time algorithm using speedup $s > 1$ and lookahead $L \geq \lceil (N+1)s/(s-1) \rceil$ tracks.*

Proof: Assume at some time step $t \geq 0$, each entry of $C^{(t)}$ is non-positive. Then by Lemma 1[10] of Section 2.5.1, each packet with fluid completion time in $[t+1, t+L]$ experiences no delay. The number of packets with fluid start time and fluid completion time in $[t+1, t+L]$ is at most L/s. This fact, and the fact that the number of packets with fluid start time in $[t+1, t+L]$ but with fluid completion time $> t+L$ is at most N (that is, one per input port), imply that the number of packets with fluid start time in $[t+1, t+L]$ is at most $L/s + N \leq L - 1$. Our assumption that each entry of $C^{(t)}$ is non-positive implies that each packet with fluid start time at most t has been sent by the packetized policy by time step t. Since the packetized policy can only send a packet at or after its fluid start time, and since at each time step $t' \in [t+1, t+L]$, if $C^{(t')}$ has a positive-valued entry, then the Earliest Completion Time algorithm sends

[10] See the comment just after the proof of this lemma.

a packet, we have that there must be some time step $\tau \in [t+1, t+L]$ such that every entry in $C^{(\tau)}$ is non-positive.

Since $C^{(0)} = \mathbf{0}$, the above argument gives that each packet with fluid completion time in $[1, L]$ experiences no delay and for some $t_1 \in [1, L]$, every entry in $C^{(t_1)}$ is non-positive. Applying the above argument at time step t_1 gives that each packet with fluid completion time in $[t_1 + 1, t_1 + L]$ experiences no delay and for some $t_2 \in [t_1 + 1, t_1 + L]$, every entry in $C^{(t_2)}$ is non-positive. Repeating this argument results in a strictly increasing sequence (where $t_0 := 0$) $t_0, t_1, t_2, ...$ such that for each k, each packet with fluid completion time in $[t_k + 1, t_k + L]$ experiences no delay and $t_{k+1} \in [t_k + 1, t_k + L]$. Thus, the Earliest Completion Time algorithm using the aforementioned lookahead tracks. □

We present a modified version of the Earliest Completion Time algorithm using speedup $s > 1$ and lookahead $L < \infty$ that guarantees delay at most $d := (\lceil (N + 1)s/(s-1) \rceil - L)^+$. The modified algorithm sends no packets for the first d time steps. At each subsequent time step t, it simulates the Earliest Completion Time algorithm using lookahead $L + d$ for time steps[11] 1 to $t - d$ on the same fluid policy $\{F^{(1)}, F^{(2)}, \ldots\}$ to find the packetized vector it would output at time step $t - d$. The modified algorithm schedules this packetized vector at time step t. Each packet is sent by the modified algorithm exactly d steps after the time it would have been sent by the Earliest Completion Time algorithm using lookahead $L+d \geq \lceil (N+1)s/(s-1) \rceil$ acting on the same fluid policy $F^{(1)}, F^{(2)}, \ldots$. Since we showed in the lemma above that the Earliest Completion Time algorithm tracks using lookahead $\geq \lceil (N + 1)s/(s - 1) \rceil$, we have that the modified algorithm sends each packet at most d time steps after its fluid completion time. This completes the proof of the second part of Theorem 6. □

[11] If the modified algorithm keeps track of the packetized vectors and cumulative difference vectors produced by the Earliest Completion Time algorithm, the modified algorithm need only simulate one time step of the Earliest Completion Time algorithm to find the next packetized vector it outputs.

2.8 Applying Our Bounds to Time-Varying, Leaky-Bucket Constrained Arrival Processes

We give an example of how our bounds on tracking, backlog, and delay can be applied when some information is provided on the packet arrival process. Here, we consider the temporary session model of Andrews and Zhang [5]:
At each input port, service is provided in consecutive sessions, with one session completely finishing before another begins. Each session j is *active* during a time interval $[t_j, u_j]$, in which packets arrive subject to a leaky-bucket constraint with parameters ρ_j, σ_j; that is, for any t', u' such that $t_j \leq t' < u' \leq u_j$, the number of packets that arrive at input port i during the time interval $[t', u']$ is at most $\sigma_j + \rho_j(u' - t')$. Note that session j packets may still be waiting at the input port after time step u_j; also, there may be times when no session is active. The sessions, with corresponding time intervals and parameters σ_j, ρ_j, may be completely different for each input port.

Given a fixed speedup s and lookahead L, we can do admission control of requests for leaky-bucket constrained, temporary sessions. Our goal is to accept as many sessions as possible and still make certain backlog and/or delay guarantees to these sessions. We accept a request to reserve a temporary session (call it session j) at input port i with time interval $[t_j, u_j]$ and with leaky-bucket constraint parameters ρ_j, σ_j, if setting $F_i^{(t)} := \rho_j$ for $t \in [t_j, u_j + \lceil(\sigma_j + 1)/\rho_j\rceil]$ doesn't violate the fluid constraint that for all time steps t', $\sum_k F_k^{(t')} \leq 1/s$. Additional fluid may be allocated to any input port (for example, when it is desirable to allocate excess bandwidth based on current queue size as argued for in [21]) as long as for all time steps t', $\sum_k F_k^{(t')} \leq 1/s$. Note that if lookahead L is used by a packet-scheduling algorithm, then all session reservations and additional fluid allocations must be made at least L steps in advance. If admission control is done in this way, we have the following guarantees for packets from session j at input port i:

- During session j, the number of packets in the queue at input port i is at most $\sigma_j + 1$ plus the backlog (due to approximating the fluid policy by the packetized

policy).

- The number of time steps a packet waits in the queue is at most $\lceil(\sigma_j + 1)/\rho_j\rceil$ plus the delay due to approximating the fluid policy by the packetized policy (which we simply refer to as "delay" in this work).

These guarantees follow from similar arguments as used by Charny in Section 3.3.1, Theorem 7 of [12].

Thus, we have from Section 2.6 that during session j at input port i, the queue at this input port never has more than $\sigma_j + 1 + \frac{1}{s}\ln N$ packets, when the Largest Cumulative Difference packet-scheduling algorithm using speedup s and no lookahead is used. Note that a burst of σ_j packets at the first time step forces the queue at input port i to hold at least $\sigma_j - 1$ packets.

Also, we have from Section 2.7 that during session j at input port i, no packet waits in the queue at this input port more than $\lceil(\sigma_j+1)/\rho_j\rceil + N/e^s$ time steps, when the Earliest Completion Time packet-scheduling algorithm using speedup s and no lookahead is used; when speedup $s > 1$ and lookahead $\lceil(N+1)s/(s-1)\rceil$ are used by this algorithm, no packet waits more than $\lceil(\sigma_j + 1)/\rho_j\rceil$ time steps. This last bound is at most $\lceil 1/\rho_j\rceil$ more than the wait time the last packet of a burst of σ_j packets would experience if session j were allocated exactly ρ_j fluid per time step in an ideal switch that sent data as fluid.

Chapter 3

The $N \times N$, Input Queued, Crossbar Switch

We establish worst-case bounds on backlog for the input queued, crossbar switch when neither speedup[1] nor lookahead is used. Specifically, for an $N \times N$ input queued, crossbar switch, we present an on-line, packet-scheduling algorithm that guarantees backlog at most $(N+1)^2/4$ packets at each input port and at each output port for any fluid policy. The algorithm can be made to run in $O(N \log N)$ time using $N/\log N$ parallel processors, by employing a fast algorithm for edge-coloring bipartite multigraphs. In the reverse direction, we show on the $N \times N$ input queued, crossbar switch, that for any on-line, packet-scheduling algorithm using no speedup and no lookahead, there is a fluid policy that causes an input port or output port to have backlog at least $(N+1)/e-2$ packets. We also extend the packet-scheduling algorithm for crossbar switches to a general class of switch architectures.

The layout of this chapter is as follows: Section 3.1 extends the traffic model from Chapter 2 to $N \times M$ input queued, crossbar switches (which we refer to simply as crossbar switches). In particular, we define backlog for crossbar switches, which quantifies how much additional buffer space is used by a packetized policy than is used by the fluid policy it is emulating. In Section 3.2, we present results from related work.

[1]The problem of bounding backlog for crossbar switches when speedup strictly greater than 1 is used is analyzed by Koksal in [37].

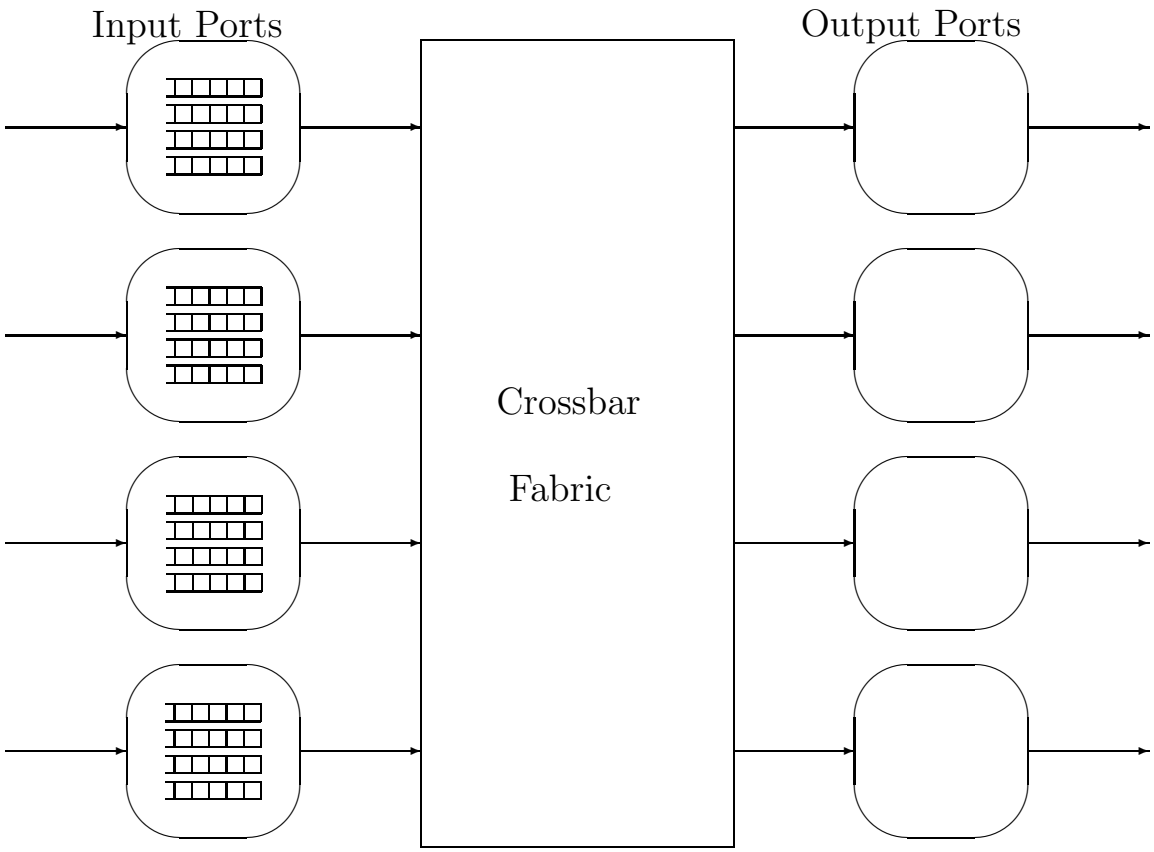

Figure 3-1: A 4×4 input queued, crossbar switch. Each input port has 4 virtual output queues, one corresponding to each output port.

In Sections 3.3 and 3.4, we analyze worst-case backlog for $N \times N$ crossbar switches. Section 3.5 extends the results of Section 3.3 to more general switch architectures than the crossbar switch, using a resource allocation model of Tassiulas [57]. Finally, Section 3.6 summarizes the chapter and gives directions for future research.

3.1 Traffic Model

We extend the traffic model from Section 2.2 of Chapter 2 from $N \times 1$ input queued switches to $N \times M$ input queued, crossbar switches. All packets are assumed to have the same size, and time is normalized so that the maximum data rate (capacity) of any input or output port is one packet per time step. The $N \times M$ crossbar 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; 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 [55] for more details.). Figure 3-1 is a diagram of a 4×4 crossbar switch. A fluid policy, as in the case of the $N \times 1$ switch, represents the ideal, packet-scheduling behavior.

A *Fluid Policy* for an $N \times M$ crossbar switch is a sequence of rate allocations specifying the number of fractional packets that ideally would be sent from each input port to each output port at each time step. This is represented by a sequence of $N \times M$ doubly sub-stochastic matrices[2] $\{F^{(t)}\}_{t>0}$. The fraction of a packet that should be sent from input i to output j at time step t is represented by $F_{ij}^{(t)}$. The *fluid start time* of the kth packet to be transmitted from input port i to output port j is the first time step such that a total of more than $k-1$ units of fluid have been sent from input port i to output port j by the fluid policy. The *fluid completion time* for the kth packet to be transmitted from input port i to output port j is the first time step at which at least a total of k units of fluid have been sent from input port i to output port j by the fluid policy. 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 (whole) packet transmissions, one per time slot. It is represented by a sequence of $N \times M$ sub-permutation[3] matrices, $\{P^{(t)}\}_{t>0}$. The value of $P_{ij}^{(t)}$ is 1 if and only if a packet is sent from input i to output j at time step t; otherwise it has value 0.

We say that a packet-scheduling algorithm *tracks* on a crossbar switch if for any given fluid policy, it sends each packet at or after its fluid start time and at or before its fluid completion time [55]. Bonuccelli and Clo [7] construct a fluid policy for the 4×4 crossbar switch that cannot be tracked. Their construction implies that for

[2]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.

[3]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.

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 packet-scheduling algorithms that satisfy a relaxation of the tracking property.

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. We define the *backlog* at a single input port, output port pair (i, j), to be the positive part of the difference between the cumulative fluid that has been sent from i to j by the fluid policy and the cumulative number of packets that have been sent from i to j by the packetized policy. The backlog for pair (i, j) at time step t is the (i, j) entry of the positive part[4] of the $N \times M$ *cumulative difference matrix* $C^{(t)} := \sum_{h=1}^{t}(F^{(h)} - P^{(h)})$. For notational convenience, we define $C^{(0)} := \mathbf{0}$, the matrix with all zeros. See Figure 3-2 for an example of a fluid policy, a packetized policy, and their cumulative difference. The backlog for a set of input port, output port pairs is the sum of backlog for each input port, output port pair in the set. For a set of input port, output port pairs, we say that a packet-scheduling algorithm gets at most *b-backlogged* if for all time steps t, their backlog is no more than b. In other words, for all t, the sum of corresponding entries in $(C^{(t)})^+$ is at most b. For example, we say that a packet-scheduling algorithm gets at most b-backlogged 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 that a packet-scheduling algorithm gets at most b-backlogged per output port if in each column of $(C^{(t)})^+$, the sum of entries is at most b at each time step t.

We define a *packet-scheduling algorithm* to be a deterministic algorithm that takes a fluid policy as input (possibly online), and outputs (one step at a time) a packetized policy that sends each packet at or after that packet's fluid start time. In other words, for all t, the packetized step $P^{(t)}$ is only allowed to have value 1 in entries (i, j) for which $C_{ij}^{(t-1)} + F_{ij}^{(t)} > 0$; this ensures that all entries of the cumulative difference matrix $C^{(t)}$ are greater than -1. In this chapter, we consider only packet-scheduling algorithms using no speedup and no lookahead.

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

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

Figure 3-2: Three steps of a fluid policy (first column), packetized policy (second column), and their cumulative difference (third column). Since a packet-scheduling algorithm can only send a packet at or after its fluid start time, a packet-scheduling algorithm has to set $P^{(3)}$ to be a sub-permutation matrix that is not a (full) 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.

3.2 Related Work

A number of authors have worked on the problem of approximating fluid schedules for $N \times N$ crossbar switches with virtual output queueing. Chang, Chen, and Huang [11] present a packet-scheduling algorithm for approximating fluid policies on an $N \times N$ crossbar switch, in which the same fluid step F is repeated at each time step; we call this a *constant* fluid policy. Ideally, service at each time slot should exactly mimic the behavior of the fluid matrix F, but in practice only whole packets can be scheduled; thus, the fluid behavior can only be approximated by a packetized policy. Their packet-scheduling algorithm, based on a Birkhoff-von Neumann decomposition of the fluid matrix F, guarantees bounded backlog for any constant fluid policy. The algorithm requires initial run-time $O(N^{4.5})$ and on-line run-time $O(\log N)$ for an $N \times N$ crossbar switch.

Charny [12] gives a simple packet-scheduling algorithm using speedup 6,[5] that

[5]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.

tracks any constant fluid policy. In the work here, we consider more general fluid policies, in which the amount of fluid scheduled can vary at each time step. Worst-case backlog and delay for time-varying fluid policies when lookahead is bounded can be significantly greater than that for constant fluid policies. For example, Theorem 6 in Section 2.7 implies that for any packet-scheduling algorithm on an $N \times N$ crossbar switch using speedup s and lookahead L, there exists a time-varying fluid policy such that for some input port, output port pair, the delay is at least $\lfloor (N+1)/e^s \rfloor - 1 - L$. Charny's packet-scheduling algorithm, however, uses speedup 6 and for any constant fluid policy guarantees (for any N) that no packet experiences any delay.

Tabatabaee, Georgiadis, and Tassiulas [55] consider the problem of tracking arbitrary fluid policies in an $N \times N$ 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 crossbar switch can be tracked, and propose several heuristics for approximating fluid policies by packetized policies on larger switches. Bonuccelli and Clo [7] construct a constant, 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 [7] that for many switch sizes there exist untrackable fluid policies, has motivated us to consider the two relaxations of tracking, backlog and delay.

Kam and Siu [34] provide bounds on worst-case backlog for time-varying fluid policies on $N \times N$ crossbar switches, 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, using speedup 2. Their proof technique does not extend to the case of no speedup; they assert "the unavailability of combinatorial proof techniques for our no-speedup scenario." [34]. The main result of this chapter is

a combinatorial proof that worst-case backlog can be kept bounded using no speedup.

Using a credit-based model similar to that used by Kam and Siu [34], Koksal [37] bounds backlog on the $N \times N$ crossbar switch (called "service lag" in his work) when speedup is strictly greater than 1. The upper bounds given tend to infinity as speedup approaches 1.

3.3 Bounds on Backlog for the $N \times N$ Crossbar Switch

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

A corollary of Theorem 2 in Section 2.6.1 of the previous chapter gives the following bound for the $N \times N$ crossbar switch:

Corollary: For the $N \times N$ crossbar switch, for every packet-scheduling algorithm using no speedup and no lookahead, for any column j, there exists an adversarial fluid policy with the following property: For any $m : 1 \leq m \leq N$, there exists a time step $t \leq N - 1$ and a set I of m input ports such that

$$\sum_{i \in I} C_{ij}^{(t)} = m(H_N - H_m). \tag{3.1}$$

In particular, at some time step the backlog of the set of pairs $\{(i, j)\}_{1 \leq i \leq N}$ is more than $(N + 1)/e - 2$; also, at some time step there is a single pair (i, j) with 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 corollary above (which follows from Theorem 2) is that for any constant speedup s, for any packet-scheduling algorithm on an $N \times N$ crossbar switch, there exists a time-varying fluid policy such that for some input port, output port pair, its backlog is more than $\frac{1}{s}(\ln(N+1)-1)$. This is in stark constrast with Charny's [12] result, in which a simple packet-scheduling

algorithm using speedup 6 is shown to track any constant fluid policy on the $N \times N$ crossbar switch.

It is not clear how to generalize the proof of Theorem 3 in Section 2.6.2, in which the Largest Cumulative Difference packet-scheduling algorithm is shown to achieve the lower bound in Theorem 2, to the $N \times N$ crossbar switch. First, the $N \times 1$ and $N \times N$ crossbar switches differ in that there is no clear choice for what the Largest Cumulative Difference packet-scheduling algorithm should be for the $N \times N$ crossbar switch. A second obstacle is that on the $N \times N$ crossbar switch, it may be impossible for a packet-scheduling algorithm to send a (full) permutation matrix at every time step, since a packet-scheduling algorithm can only send a packet at or after its fluid start time; for example, see Figure 3-2 and the accompanying caption. In light of this fact, it is not initially clear whether backlog can be kept bounded at all for the $N \times N$ crossbar switch. The next section proves that it can.

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

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

Algorithm 1

The algorithm builds a packetized policy $\{P^{(t)}\}_{t>0}$ from a given fluid policy $\{F^{(t)}\}_{t>0}$. At each iteration, the algorithm computes $P^{(t+1)}$ based on $C^{(t)}$ and $F^{(t+1)}$. This computation is described below, where 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[6] $C + F$ 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 [39], 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 [49] for further details.

Lemma 5 *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:

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

Lemma 6 *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 Matching Theorem [29], 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 $m \times (N-m+1)$.[7] 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.[8] □

Lemma 6 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. [13]) to immediately obtain a similar, but weaker version of Lemma 6. This follows since by the

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

[8] 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 $w \geq N(N+2)/4$.

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 t.

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 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 5 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 6, 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:

$$C + F \leq B. \tag{3.2}$$

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

$$(C + F - \pi)^+ \leq (B - \pi)^+ = B - \pi, \tag{3.3}$$

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

$$(C + F - \pi)^+ = (C + F - P)^+. \tag{3.4}$$

Therefore, from (3.3), 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. □

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[9] in an N by N bipartite graph, which is $O(N^{2.5})$ [30]. We show in Section 3.4 how to use a fast algorithm for edge-coloring bipartite multigraphs to reduce the running time per fluid step to $O(N \log N)$, when $N/\log N$ parallel processors are used.

3.4 A Packet-Scheduling Algorithm with Running Time $O(N \log N)$ per Fluid Step, Using $N/\log N$ Parallel Processors

We present a packet-scheduling algorithm for an $N \times N$ input queued, crossbar switch with the following performance guarantees, for any $\epsilon : 0 < \epsilon < 1$:

- It runs in time $O(\frac{1}{\epsilon} N \log N)$ per fluid step, using $N/\log N$ parallel processors.
- It uses no speedup.
- It guarantees backlog at most $(1+\epsilon)(N+1)^2/4$ packets per input port and per output port.

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

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 6, 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 sums and column sums of $(C^{(t)})^+$ are at most $(N+1)^2/4 + 3\alpha(N)$.

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

Algorithm 2 Outline: The overall structure of the packet-scheduling 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 [38], it is possible to decompose D' into a 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 $1 \leq t \leq 2\alpha(N)$, 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

   $C^{(b\alpha(N))} - \sum_{t=b\alpha(N)+1}^{(b+2)\alpha(N)} P^{(t)} + \sum_{t=b\alpha(N)+1}^{(b+1)\alpha(N)} F^{(t)}$

   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. [13]).

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 6, 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 [38] 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 $m : 1 \leq m \leq \alpha(N)$, we set $\pi^{((b+2)\alpha(N)+m)}$ to be the mth permutation matrix from the previous step. Lastly, for $t = (b+2)\alpha(N)+1$ to $(b+3)\alpha(N)$, the packetized step $P^{(t)}$ is set to be the sub-permutation matrix defined as:

$$P_{ij}^{(t)} := \begin{cases} \pi_{ij}^{(t)}, & \text{if } C_{ij}^{((b+1)\alpha(N))} - \sum_{h=(b+1)\alpha(N)+1}^{t-1} P_{ij}^{(h)} > 0 \\ 0, & \text{otherwise} \end{cases}$$

The proof that Invariant 2 holds is similar to the proof that Invariant 1 holds from Section 3.3.

The run-time of one iteration of Algorithm 2 is dominated by steps 1, 2, and 3, in which $\alpha(N)$ fluid steps are summed, an integral, maximum flow is found, and a minimum edge-coloring is found, respectively. If we assume the algorithm uses $\beta(N)$ parallel processors, then the sum of $\alpha(N)$ fluid steps can be computed is time $O(N^2\alpha(N)/\beta(N))$. An integral, maximum flow in a graph with N nodes can be computed in $O(N^3)$ time[10] [15]. 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 [50, 23] for the special case of Δ a power of 2.

[10] or in time $O(N^{8/3} \log N)$ using an algorithm of Goldberg and Rao [25]

The run time of the entire algorithm is then $O(N^2\alpha(N)/\beta(N)+N^3+N\alpha(N)\log\alpha(N))$ 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 $O(N^2/\beta(N) + N^3/\alpha(N) + N\log\alpha(N))$. The algorithm computes packetized steps $P^{(t)}$ for $t = (b+2)\alpha(N)+1$ 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 $P^{(t)}$ for $t = (b+1)\alpha(N)+1$ 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, and the number of parallel processors $\beta(N)$ to be $N/\log N$, 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 and per output port claimed at the beginning of this section.[11]

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.

3.5 Packet-Scheduling Generalized Fluid Policies

Using the resource allocation model of Tassiulas [57], 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 [57], 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

[11] In the case of $\epsilon N^2/24 < 1$, we can run Algorithm 1 instead of Algorithm 2, which results in run time $O(N^{2.5})$ (so also $O(N/\epsilon)$) per fluid step and backog per input port and per output port at most $(N+1)^2/4$.

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[12]. 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 or column 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 a generalized, packet-scheduling algorithm using no speedup and no lookahead for building, from any generalized, fluid policy, a generalized, packetized policy that gets at most $(MN)m(S)$-backlogged per input port and per output 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 [13]. The main difference between the algorithm in this section and the previous algorithms is that we don't know of an efficient way to find an element of such a decomposition with large weight. In Chapters 4 and 5, we will give polynomial-time implementations of the algorithm in this section for Banyan networks and for general networks, respectively. The packet-scheduling algorithm below maintains the following invariant for all time steps t:

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

Algorithm 3

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

[12]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 3.1: Summary of bounds on backlog for the $N \times N$ Crossbar Switch

	Lower Bound	Upper Bound
Backlog per input port and per output port	$(N+1)/e - 2$	$(N+1)^2/4$
Backlog per Virtual Output Queue	$\ln N - 1$	$(N+1)^2/4$

First, it calculates $C^{(t-1)} := \sum_{h=1}^{t-1}(F^{(h)} - P^{(h)})$, and then decomposes $(F^{(t)} + C^{(t-1)})^+$ 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

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

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

3.6 Chapter Summary

The bounds proved in this paper for the $N \times N$ crossbar switch are summarized in Table 3.1 above. A lower bound of $f(N)$ means that for any deterministic, packet-scheduling algorithm using no speedup and no lookahead, there exists a fluid policy that when used as input to the packet-scheduling algorithm will cause backlog at least $f(N)$. An upper bound of $g(N)$ means that there exists a packet-scheduling algorithm using no speedup and no lookahead that is at most $g(N)$-backlogged when run on any fluid policy.

As can be seen in Table 3.1, gaps remain between the lower and upper bounds proved for backlog. In particular, the gap for the backlog of a virtual output queue, which corresponds to the maximum entry in the cumulative difference matrix $C^{(t)}$, is quite large. Narrowing these gaps, and finding tighter tradeoffs between backlog and run-time of packet-scheduling algorithms are directions for future research.

Chapter 4

Banyan Networks

Multistage switching fabrics, based on the interconnection of small switch components, allow for the efficient, modular construction of high-performance routers. We look at the resources, in terms of buffer size and switch speedup, required to provide flexible, rate-based, quality of service guarantees for Banyan multistage switch networks.

We consider Banyan networks in which all links have unit capacity (that is, at most one packet can traverse a link in one time step), and buffering is only allowed just before the first-stage input ports and just after the last-stage output ports. The design of packet-scheduling algorithms for such networks is significantly more difficult than for crossbar switches, 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.

In this chapter, we focus on bounding backlog in Banyan networks; our results for Banyan networks are interesting in themselves, and focusing on them allows a concrete demonstration of the techniques and algorithms we use. In the next chapter, we extend some of the bounds derived here to general, input-queued switch networks, which do not in general have the properties that allow the relatively simpler analysis of Banyan networks in this chapter; these properties of Banyan networks include

being layered, having unit-capacity links, and having for each source and destination a unique path connecting them.

4.1 Notation and Definitions

4.1.1 Structure of 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, [45], [35], and references therein). A Banyan network is a set of *switch elements*, that is 2×2 crossbar switches, interconnected by links, with a structure defined below.

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}; we say that a link connecting a switch element in S_h to a switch element in S_{h+1} is *between stages* h *and* $h+1$, and for convenience we say an incoming link to S_1 is between stages 0 and 1, and an outgoing link from stage S_m is between stages m and $m+1$. Incoming links to S_1 are called *input ports* and outgoing links from S_m are called *output ports*. Figure 4-1 depicts a 16×16 Banyan network, which has 4 stages.

$N \times N$ Banyan networks, which have N input ports and N output ports for N a power of 2, can be constructed recursively by appropriately connecting smaller Banyan networks. The following construction, depicted in Figure 4-2, is from [45]. 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 Figure 4-2. 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

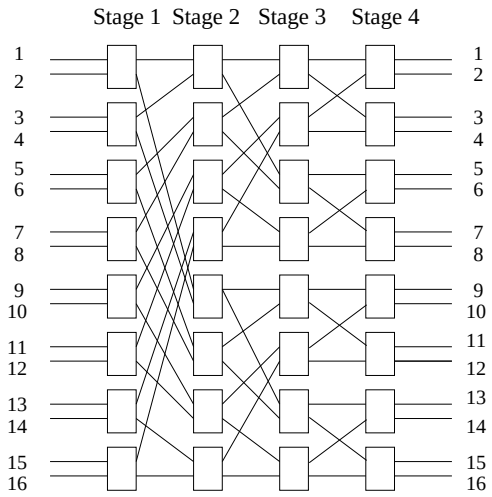

Figure 4-1: A 16 by 16 Banyan Network built from 2×2 crossbar switch elements.

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 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.

As in the previous chapters, we assume time is discrete. One time step is set to be the time required for a 2×2 crossbar switch to transmit a packet across the switch fabric and through an outgoing link. We say a set of packets *is sent* at time step $t > 0$ to mean each packet in the set traverses an incoming link of a first-stage 2×2 switch element during time step t. For each $h : 1 \leq h \leq m$, during time step $t + h$, each packet in this set is transmitted across the switch fabric of a 2×2 switch element in stage h of the Banyan network and through one of its outgoing links, arriving at a stage $h + 1$ switch element (or at an output port if $h = m$). Note that packets sent at different time steps are never in danger of traversing the same link at the same time, since at each time step they traverse links connected to different stages of the Banyan network.

One of the properties of the Banyan network is that each input port, output port pair (which we simply refer to as an *input, output pair* in this chapter), is connected by a unique path through the network [45]. We call this the *unique-path property.*

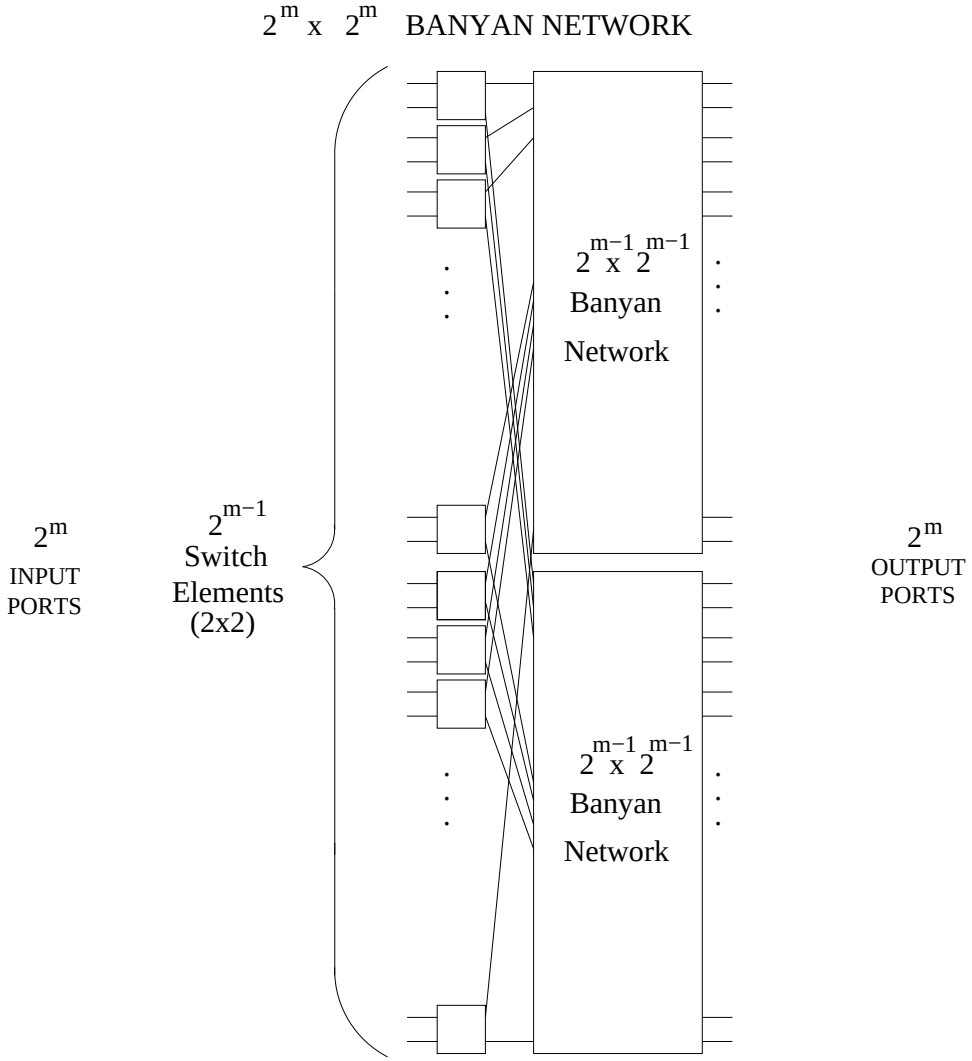

Figure 4-2: Recursive construction of a $2^m \times 2^m$ Banyan network, for $m \geq 2$.

The unique-path property can be proven by induction on the number of stages in the Banyan network, using the recursive structure given above. We consider Banyan networks without buffers in the intermediate stages and with all links having unit capacity. Since we don't allow packets to be dropped, if input i is transmitting a packet to output 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. We refer to the set of input, output pairs (k, l) that are blocked by pair (i, j), as the neighborhood of (i, j).

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 7 *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.*[1]

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 Figure 4-2. 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

[1] In general, a family of sets is said to have the *Helly property* if for any subfamily of pairwise non-disjoint sets, the intersection of the sets in the subfamily is non-empty [60]. 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.

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'. □

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 Figures 4-3 and 4-4 we show the 4×4 Banyan network, and the associated link graph G. Consider link 2 in Figure 4-3. 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 Figure 4-4. 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 7 above, each clique in the link graph corresponds to a set of input,

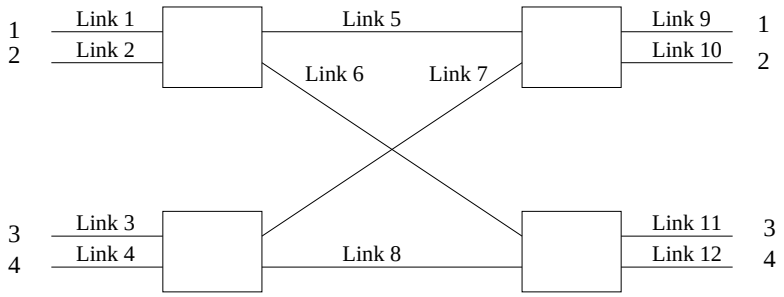

Figure 4-3: A 4×4 Banyan Network.

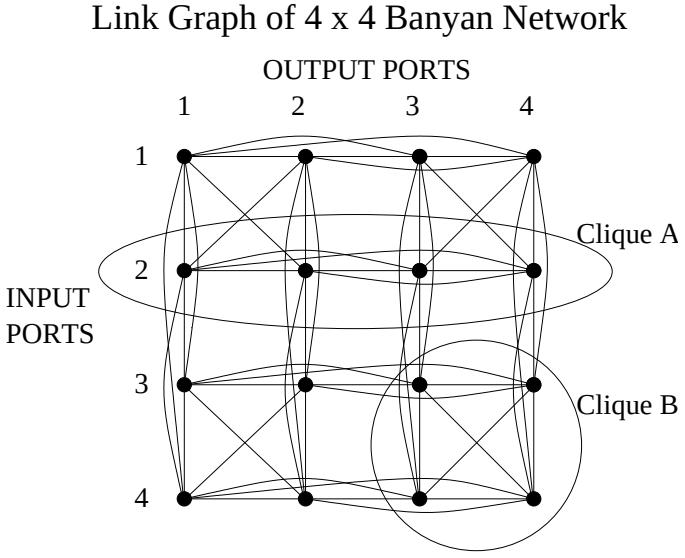

Figure 4-4: The link graph corresponding to the above 4×4 Banyan Network.

output pairs, all of whose paths contain some link l.

Valid Transmissions and Clique Constraints

We now show the connection between cliques in the link graph and sets of packets that can be simultaneously transmitted without causing any packets to be dropped. A *valid transmission* is a set of fractional packets sent at the same time step t, in which for $0 \leq h \leq m$, for each link between stages h and $h+1$, the total traversing that link at time step $t+h$ is at most one. By Lemma 7, a transmission sending x_{ij} from input i to output j is valid if and only if for each clique Q in the link graph G, the following *clique constraint* is satisfied:

$$\sum_{(i,j)\in Q} x_{ij} \leq 1. \tag{4.1}$$

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

4.1.2 Definition of Fluid Policy and Packetized Policy

As mentioned in the previous section, one time step is set to be the time required for a single crossbar switch to transmit a packet waiting at an input port across the switch fabric and through an outgoing link. At each time step, a new fluid step is made available to the packet-scheduling algorithm, which outputs a packetized step for that time step. Buffering is only allowed at the input ports and output ports. The switch uses virtual output queueing at each input port to avoid head-of-line blocking. All packets are assumed to have the same size.

A *Fluid Policy* for an $N \times N$ Banyan network is a sequence of valid transmissions specifying the number of fractional packets that ideally would be sent over the multi-stage switch. This is represented by a sequence of $N \times N$ non-negative-valued, *fluid matrices* $\{F^{(t)}\}_{t>0}$, where $F_{ij}^{(t)}$ represents the fraction of a packet sent from input i of the first stage at time step t, and with output j of the last stage as its destination. Each fluid matrix must satisfy the constraint that the total fluid traversing any link

in the Banyan network is at most 1. A *constant* fluid policy is a fluid policy where for some fluid matrix F, $F^{(t)} = F$ for all t.

A *Packetized Policy* for an $N \times N$ Banyan network is a sequence of valid transmissions where only whole packets are sent and received at each time step. It is represented by a sequence of $N \times N$ *packetized matrices*, $\{P^{(t)}\}_{t>0}$ where $P_{ij}^{(t)}$ has value 1 if a packet is sent from input i at time step t, with output j as its destination; otherwise it has value 0. Each packetized matrix must obey the constraint that for each link in the network, at most one packet traverses it; in other words, no packet collisions are tolerated in the packetized policy. Note that a $\{0,1\}$-valued 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.

4.1.3 Definition of Cumulative Difference, Backlog, Speedup

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 $N \times N$ 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 pairs of input ports and output ports, we define their *backlog* to be the sum of corresponding entries in $(C^{(t)})^+$. In order for a packet-scheduling algorithm to send a packet from input port i to output port j, we require $C_{ij}^{(t-1)} + F_{ij}^{(t)} > 0$, just as for the crossbar switch. This requirement reflects our interpretation of $\lceil C_{ij}^{(t-1)} + F_{ij}^{(t)} \rceil$ representing the number of packets waiting at input port i ready to be sent to output port j; when this quantity is zero, there is no packet to be scheduled. This requirement also ensures that the packetized policy never gets ahead of the fluid policy it is emulating.

In general, a switch network is said to use speedup $s \geq 1$ if all its switch elements and all its internal links send packets s times as fast as the line rate of the input ports. We model a scheduling algorithm using *speedup* $s \geq 1$ by requiring that for each fluid

matrix in any fluid policy, the link usage totals at most $1/s$ for each link; that is, we require each fluid matrix F to satisfy the following clique constraints (which are more stringent than those of (4.1)): For each clique Q in the link graph,

$$\sum_{(i,j)\in Q} F_{ij} \leq 1/s. \tag{4.2}$$

We say that an algorithm uses *no speedup* if $s = 1$.

We focus on solving the on-line, traffic scheduling problem, whose input at each time step is the corresponding fluid matrix from a fluid policy. The goal at each time step is to choose a packetized matrix so that backlog remains bounded for all time steps. 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 uses speedup at least s.

4.2 Results

We begin this chapter with a negative result. Specifically, we prove that bounded backlog results of the type developed in the previous chapter for the crossbar switch do not exist for arbitrary switch networks. In fact, this is true even for Banyan networks *for the simple case of a constant fluid policy.* This motivates our analyzing the necessary and sufficient speedup for maintaining bounded backlog. Specifically, given a Banyan network we seek the minimum amount of speedup that is sufficient for maintaining bounded backlog for all fluid policies. Universal bounds for this required speedup are established.

First, as concrete motivation for the sequel, in Section 4.3, we show 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 4.4 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 of a Banyan network. We first state a result, which follows directly from a theorem of Koksal [37], 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 4.5 we revisit the 4×4 Banyan network, and show, using the machinery developed in Section 4.4, that speedup $4/3$ is in fact necessary and sufficient for maintaining bounded backlog for arbitrary fluid policies. In Section 4.6 we determine, through analysis of polytopes and non-trivial computations that the necessary and sufficient speedup to keep backlog bounded on 8×8 Banyan networks is also $4/3$.

In Section 4.7 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. Furthermore, in this case we show how to implement the packet-scheduling algorithm of Section 4.4 to compute each packetized matrix in time polynomial in N.

4.3 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 Figure 4-3. 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 each link in the network has total demand equal to unity. In other words, for every clique Q in the link graph G (see Figure 4-4 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 4.5 we show that this result is tight.

4.4 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 which we use throughout. We define the polytope $\mathcal{P}$ to be the convex hull 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 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 [27],[49] for details).

For Banyan networks, we have that $\mathcal{P} \subseteq \mathcal{F}$ (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 4.3 above shows that this inclusion can be strict. For the $N \times N$ Banyan network, the dimension of $\mathcal{P}$ is N^2.

Recall that we model a scheduling algorithm using *speedup* $s \geq 1$ 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 $F^{(t)} \in \frac{1}{s}\mathcal{F}$.

If $\mathcal{P} = \mathcal{F}$ then every fluid matrix can be seen as a convex combination of packetized matrices and so for any constant fluid policy, bounded backlog can be maintained using no speedup (by simply scheduling the 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* (see [27]). 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 a graph-theoretic explanation of the fact that no speedup is required for maintaining bounded backlog on crossbar switches.

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 $F \in \mathcal{P}$ [37]. This implies that using speedup s, bounded backlog can be maintained for all *constant* fluid policies if and only if $\frac{1}{s}\mathcal{F} \subseteq \mathcal{P}$. We use ideas similar to those used in the previous chapter to show 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 7 *Using speedup s, bounded backlog can be maintained for all arbitrary fluid policies if and only if $\frac{1}{s}\mathcal{F} \subseteq \mathcal{P}$.*

Proof: Because of Koksal's result mentioned above, it suffices to exhibit, in the case where $\frac{1}{s}\mathcal{F} \subseteq \mathcal{P}$, a packet-scheduling algorithm using speedup s that maintains bounded backlog for any fluid policy.

Assume $\frac{1}{s}\mathcal{F} \subseteq \mathcal{P}$ holds, so that for each fluid matrix $F^{(t)}$, (which is by definition in $\frac{1}{s}\mathcal{F}$ when speedup s is used) we have $F^{(t)} \in \mathcal{P}$. 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 4 $C^{(t)} \in N^2\mathcal{P}$.

This invariant implies that no input port can have backlog more than N^2. We present the algorithm below, which is modeled after Algorithm 3 of the previous chapter, and prove it maintains the invariant above inductively. Note that the invariant holds at time step $t = 0$, since $C^{(0)} = \mathbf{0}$, which is in $N^2\mathcal{P}$.

Algorithm 4

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 gives 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, Algorithm 4, for any fluid policy, 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 (unless one has a description of $\mathcal{P}$ in terms of linear inequalities). We give a modified, packet-scheduling algorithm below that only requires decomposing each fluid matrix $F^{(t)}$ into a convex combination of packetized matrices, and that maintains bounded backlog for any fluid policy. We show in Section 4.7 that for the case of a Banyan Network with N input ports and speedup $s = \log_2 N + 1$, such a decomposition can be computed in time polynomial in N; thus, in this case the packet-scheduling algorithm below runs in time polynomial in N and maintains bounded backlog for any fluid policy.

Again we assume that $\frac{1}{s}\mathcal{F} \subseteq \mathcal{P}$, so that for each fluid matrix $F^{(t)}$, (which is by definition in $\frac{1}{s}\mathcal{F}$ when speedup s is used) we have $F^{(t)} \in \mathcal{P}$. The algorithm maintains the following invariant:

Invariant 5 *For all time steps t, we have matrices $Q^{(1)}, \ldots, Q^{(N^2+1)}$, which are vertices of $\mathcal{P}$, and non-negative coefficients $\lambda_1, \ldots, \lambda_{N^2+1}$ (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 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 it maintains the invariant at all time steps inductively. 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 5

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

Since $F^{(t+1)} \in \frac{1}{s}\mathcal{F} \subseteq \mathcal{P}$, this fluid matrix can be decomposed 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 4.7. Now, if we assume Invariant 5 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 $N^2 + 1$. This follows from the fact that we have N^2 equalities defining T and one equality constraining the sum of the λ_i's and the γ_j's, all these equalities being defined on the variables $\{\lambda_i\} \cup \{\gamma_j\}$. Here, the decomposition can be done in polynomial time: as long as there are more than $N^2 + 1$ matrices with positive weights, we can reduce this number by taking any nonzero vector y in the null space of the matrix defining the $N^2 + 1$ equalities and modifying the variables $\{\lambda_i\} \cup \{\gamma_j\}$ in the direction y until one more variable becomes 0. This variable is then removed from consideration and the process is repeated until we are down to at most $N^2 + 1$ matrices with positive 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 $i : k + 1 \leq i \leq N^2 + 1$, 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 $\lambda_i \geq 1$. 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. □

4.5 Speedup Required for 4×4 Banyan Networks

In Section 4.3 we exhibited a constant fluid policy on a 4×4 Banyan network that required speedup 4/3 for maintaining bounded backlog. Using the results of Section 4.4 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.

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

Figure 4-5: We obtain the upper bound on speedup for the 4×4 Banyan network by decomposing any 4×4 fluid policy into four parts as shown above.

From the above discussion, it is sufficient to show $\frac{3}{4}\mathcal{F} \subseteq \mathcal{P}$ for the 4×4 Banyan network. To show this, decompose any fluid matrix F into a linear combination of four matrices, each with the four entries in one corner set to 0, as shown in Figure 4-5. The weight of each matrix is 1/3. We then use the fact that one can further decompose any of these four matrices into a convex combination of packetized matrices. This follows since the subgraph corresponding to one of these matrices with a corner deleted, 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 figure 4-6 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. [27]) that complements of comparability

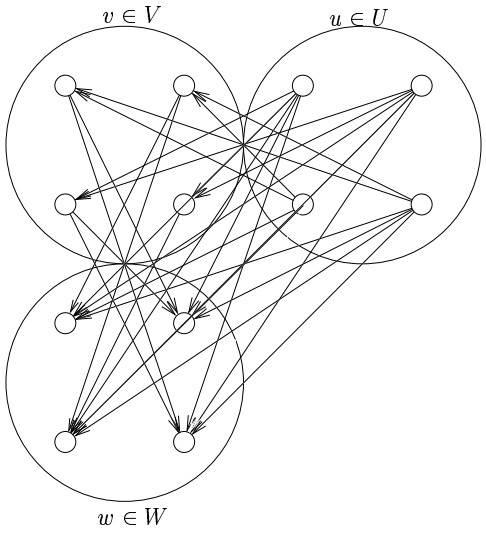

Figure 4-6: This is the complement of a 4×4 Banyan network link graph with the corner removed. The edges are oriented so that edges between nodes in U and nodes in V are directed towards V, edges between U and W are directed towards W, and edges between V and W are directed towards W. Therefore this is a directed, acyclic, transitive graph.

graphs are perfect. Thus any fluid matrix can be written as a convex combination of packetized matrices. Doing this for all four quadrants results in a nonnegative, linear combination of valid packetized matrices, with the sum of weights at most 4/3. This shows that $\frac{3}{4}\mathcal{F} \subseteq \mathcal{P}$. We can then use packet-scheduling Algorithm 4 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.

4.6 Speedup Required for 8×8 Banyan Networks

We determine, through detailed analysis of polytopes and computations using the package **cdd+**[2], that the necessary and sufficient speedup to keep backlog bounded on 8×8 Banyan networks is $4/3$. This section can be skipped without loss of continuity by readers interested in the results of later sections.

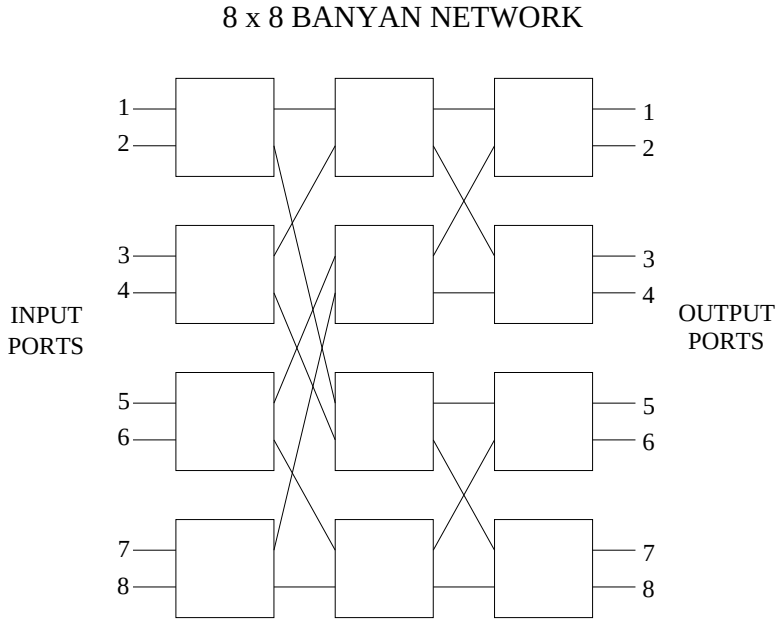

Figure 4-7: An 8×8 Banyan network.

4.6.1 Lower Bound on Speedup for 8×8 Banyan Networks

The lower bound of $4/3$ on the speedup required to maintain bounded backlog follows from a similar argument as for the lower bound on the 4×4 Banyan network in Section 4.3. Consider the constant fluid policy that sends the following fluid matrix at each time step:

[2]**cdd+** is, according to its author Komei Fukuda, an "implementation of the Double Description Method [42] for generating all vertices (i.e. extreme points) 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.

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

One can verify that F satisfies the clique constraints (4.1), and so is a valid fluid matrix. A total of eight units are transmitted by each fluid step. However, at most six packets corresponding to positive-valued entries in F can be sent in any packetized step. Indeed, at most three of the eight 1/2's in F printed in boldface can be sent in one packetized step, and the same holds for the other eight 1/2's in F. Thus, for any packet-scheduling algorithm using speedup less than 4/3, backlog will be unbounded. Next, we determine an upper bound that matches this lower bound on backlog.

4.6.2 Upper Bound on Speedup for 8×8 Banyan Networks

We determine that 4/3 speedup is sufficient[3] to maintain bounded backlog for any fluid policy on the 8×8 Banyan network. We use the characterization in Theorem 7 that if $\frac{1}{s}\mathcal{F} \subseteq \mathcal{P}$, then speedup s is sufficient to maintain bounded backlog for any fluid policy. Our strategy is to show that for every vertex v of $\mathcal{F}$, we have $(3/4)v \in \mathcal{P}$, which implies the 4/3 upper bound on required speedup. The main difficulty is that the number of vertices of $\mathcal{F}$ is too large to efficiently enumerate and check individually. To address this problem, we show, using properties of the Banyan network including certain symmetries of its link graph, that for a relatively small subset T of the vertices

[3]In fact, our technique determines the *necessary and sufficient* speedup to keep backlog bounded for any fluid policy. However, since we rely on computations by **cdd+** to show this, we included the direct proof that 4/3 speedup is necessary to maintain bounded backlog in Section 4.6.1 above.

of $\mathcal{F}$ (defined below), we have

$$\min\{s \geq 1 : \frac{1}{s}\mathcal{F} \subseteq \mathcal{P}\} = \min\{s \geq 1 : \frac{1}{s}T \subseteq \mathcal{P}\}.$$

We next use the software package **cdd+** to enumerate the elements of T. We then consider each element $v \in T$ individually, and calculate $\min\{s \geq 1 : \frac{1}{s}v \in \mathcal{P}\}$. Lastly, we take the maximum of the previous calculation over all $v \in T$, and show it equals 4/3.

Outline

We outline the steps involved in determining $\min\{s \geq 1 : \frac{1}{s}\mathcal{F} \subseteq \mathcal{P}\} = 4/3$ for 8×8 Banyan networks.

1. We first define the polytope $\mathcal{F}_{\text{TIGHT}}$ that consists of all $F \in \mathcal{F}$ in which a certain set of clique constraints (defined below) is tight. We show

$$\min\{s \geq 1 : \frac{1}{s}\mathcal{F} \subseteq \mathcal{P}\} = \min\{s \geq 1 : \frac{1}{s}\mathcal{F}_{\text{TIGHT}} \subseteq \mathcal{P}\}. \tag{4.3}$$

   This allows us to focus on the set of vertices of $\mathcal{F}_{\text{TIGHT}}$, rather than all the vertices of $\mathcal{F}$.

2. We determine the number of linearly independent clique constraints for the 8×8 Banyan network, which is an upper bound on the number of positive entries in any vertex in $\mathcal{F}_{\text{TIGHT}}$. We use this fact to classify the set of vertices of $\mathcal{F}_{\text{TIGHT}}$ into four (overlapping) groups, each defined by a property that is used in step 4.

3. We define the group of automorphisms Γ of the link graph of a Banyan network, which represent the symmetries in the link graph. We show that for any vertex v of $\mathcal{F}_{\text{TIGHT}}$, for $\Gamma(v)$ the orbit (defined below) of this vertex under the group of automorphisms, we have

$$\min\{s \geq 1 : \frac{1}{s}v \in \mathcal{P}\} = \min\{s \geq 1 : \frac{1}{s}\Gamma(v) \subseteq \mathcal{P}\}.$$

Thus, in calculating (4.3), it suffices to find $\min\{s \geq 1 : \frac{1}{s}T \subseteq \mathcal{P}\}$ for T a subset of the vertices of $\mathcal{F}_{\text{TIGHT}}$ whose orbit $\Gamma(T)$ includes the set of all vertices of $\mathcal{F}_{\text{TIGHT}}$.

4. We characterize a subset T of the vertices of $\mathcal{F}_{\text{TIGHT}}$ whose orbit $\Gamma(T)$ includes the set of all vertices of $\mathcal{F}_{\text{TIGHT}}$, using the classification of vertices into groups with certain properties from step 2. We use **cdd+** to enumerate the elements of T.

5. We calculate $\min\{s \geq 1 : \frac{1}{s}v \in \mathcal{P}\}$ for each $v \in T$ by solving a linear program. The maximum of this calculation over all $v \in T$ is shown to equal $4/3$.

1. Clique Constraints and $\mathcal{F}_{\text{TIGHT}}$

We start by considering the correspondance between links in a $2^m \times 2^m$ Banyan network and certain cliques in its link graph. We associate each link in a Banyan network with the clique of nodes in the link graph corresponding to the input, output pairs whose (unique) paths through the Banyan network contain this link. Consider any link l between stages h and $h+1$. By construction of the Banyan network, this link is connected to 2^h consecutive input ports and is also connected to 2^{m-h} consecutive output ports. Furthermore this set of input ports is of the form $[p2^h + 1, (p+1)2^h]$ for some integer p with $0 \leq p \leq 2^{m-h} - 1$, and similarly this set of outputs is of the form $[q2^{m-h} + 1, (q+1)2^{m-h}]$ for some integer q with $0 \leq q \leq 2^h - 1$. We refer to the clique in the link graph corresponding to this set of input, output pairs as $Q_{h,k}^{(m)}$ where $k = p2^h + q$ is an integer between 0 and $2^m - 1$.

We drop the superscript m in $Q_{h,k}^{(m)}$ when it is clear from the context. For a $2^m \times 2^m$ Banyan network, we let

$$\mathcal{Q} := \{Q_{h,k} : 0 \leq h \leq m, 0 \leq k \leq 2^m - 1\}.$$

When we refer to $Q_{h,k} \in \mathcal{Q}$ throughout this section, we implicitly assume h and k satisfy $0 \leq h \leq m, 0 \leq k \leq 2^m - 1$. We sometimes abuse the notation and refer to $Q_{h,k} \in \mathcal{Q}$ as a clique in the link graph, where we mean the set of nodes corresponding

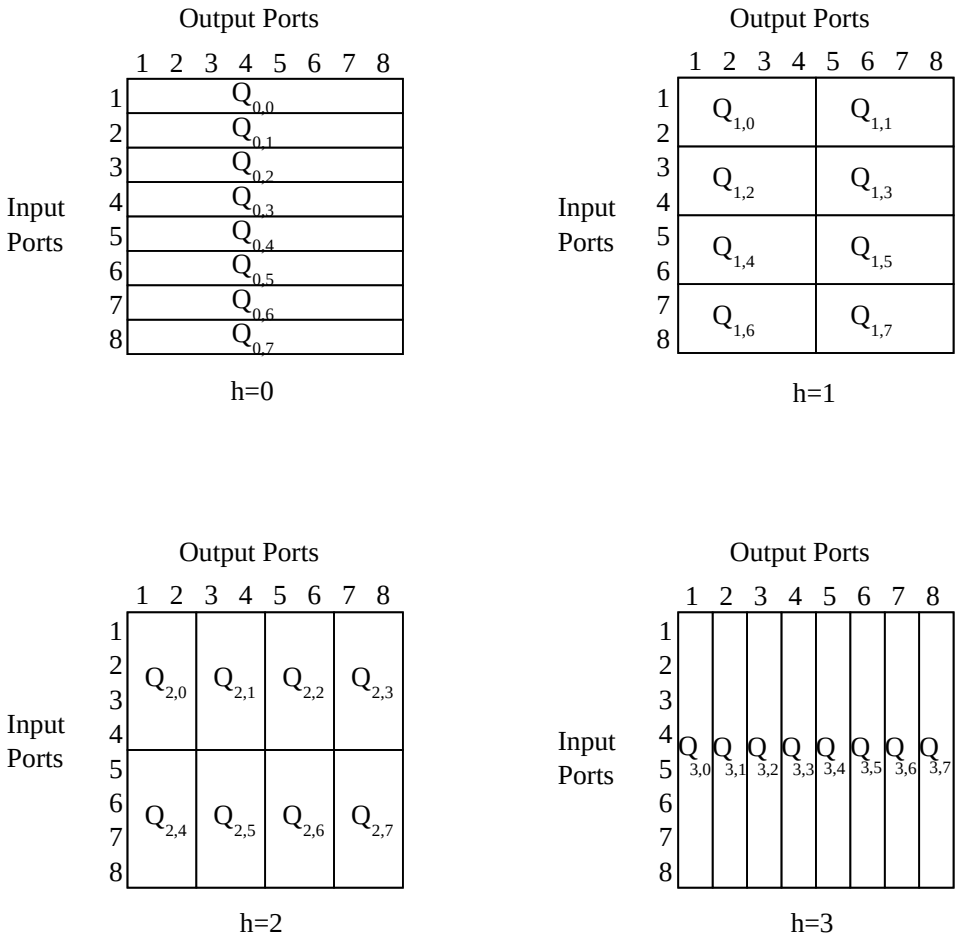

Figure 4-8: The set of cliques $\mathcal{Q} = \{Q_{h,k}\}_{0 \le h \le 3, 0 \le k \le 7}$ for an 8×8 Banyan network. Each clique $Q_{h,k}$ is represented by a rectangle of input, output pairs corresponding to the nodes in the clique. For example, $Q_{0,0}$ is the set of nodes corresponding to $\{(1, j) : 1 \le j \le 8\}$.

to the input, output pairs in $Q_{h,k}$. The set of cliques $\mathcal{Q}$ for the 8×8 Banyan network is shown in Figure 4-8.

The set $\mathcal{Q}$ is especially useful since a consequence of Lemma 7 of Section 4.1.1 above is that any clique in the link graph is contained in some clique in $\mathcal{Q}$. Also, each clique in $\mathcal{Q}$ contains 2^m nodes. We therefore refer to this set of cliques as the *maximum cliques* for the link graph of a Banyan network. The following lemma, which we prove for general, $N \times N$ Banyan networks, deals with constraints connected to the set of maximum cliques $\mathcal{Q}$.

Lemma 8 *For any $N \times N$ Banyan network, for any $F \in \mathcal{F}$, there is an $F' \in \mathcal{F}$ such that $F \le F'$ and for any $Q_{h,k} \in \mathcal{Q}$, we have $\sum_{(i,j) \in Q_{h,k}} F'_{ij} = 1$. Furthermore, if*

F is $\{0,1\}$-valued, then there is a $\{0,1\}$-valued F' with properties as in the previous sentence.

Proof: Take any $F \in \mathcal{F}$ such that for some clique $Q_{h,k} \in \mathcal{Q}$, we have $\sum_{(i,j)\in Q_{h,k}} F_{ij} < 1$. Let l_h be the link between stages h and $h+1$ of the Banyan network for which the set of input, output pairs whose paths contain l_h is $Q_{h,k}$.

If $h < m$, consider the 2×2 switch element u_{h+1} in stage $h+1$ of the Banyan network that has l_h as an incoming link. The sum of entries in F corresponding to the input, output pairs whose paths contain an *incoming* link to switch element u_{h+1} equals the sum of entries in F corresponding to the input, output pairs whose paths contain an *outgoing* link from u_{h+1}. Thus, for at least one of the outgoing links of the switch element u_{h+1}, the corresponding clique $Q_{h+1,k'} \in \mathcal{Q}$ satisfies $\sum_{(i,j)\in Q_{h+1,k'}} F_{ij} < 1$. Let l_{h+1} be such an outgoing link from switch element u_{h+1}. We can repeat the previous argument to produce a sequence of links $l = l_h, l_{h+1}, \ldots, l_m$, each of whose corresponding clique $Q_{h',k'}$ satisfies $\sum_{(i,j)\in Q_{h',k'}} F_{ij} < 1$. We can apply the same argument in the reverse direction (as long as $h > 0$) to produce a sequence of links $l_0, l_1 \ldots, l_h$, each of whose corresponding clique $Q_{h',k'}$ satisfies $\sum_{(i,j)\in Q_{h',k'}} F_{ij} < 1$.

Now $l_0, l_1, \ldots, l_h, \ldots, l_m$ is a path from some input port i to some output port j of the Banyan network. We increase the value of F_{ij} until one of the cliques $Q_{h',k'}$ associated with a link in this path satisfies $\sum_{(i,j)\in Q_{h',k'}} F_{ij} = 1$. This will not violate any other clique constraints, since for each clique in the link graph, there is a clique in $\mathcal{Q}$ that contains it, by Lemma 7 of Section 4.1.1. Since the only cliques in $\mathcal{Q}$ that contain (i,j) are those associated with a link in the (unique) path $l_0, l_1, \ldots, l_h, \ldots, l_m$ from i to j, no clique constraint is violated.

We repeat this entire procedure until $\sum_{(i,j)\in Q_{h,k}} F_{ij} = 1$ for each clique $Q_{h,k} \in \mathcal{Q}$. This requires at most $N(\log_2 N + 1)$ iterations, since at least one link has its corresponding clique-sum increased to equal 1 at each iteration.

If F is $\{0,1\}$-valued, then the above procedure produces F' that is also $\{0,1\}$-valued.[4] □

[4]In fact, the second sentence in the lemma is a direct consequence of Lemma 7 of Section 4.1.1.

Let $\mathcal{F}_{\text{TIGHT}}$ be the set of all $F \in \mathcal{F}$ such that for any $Q_{h,k} \in \mathcal{Q}$, the sum $\sum_{(i,j)\in Q_{h,k}} F_{ij} = 1$. Since $\mathcal{P}$ is $\mathbf{R}_+$ down-monotone, that is $D \in \mathcal{P}$ and $\mathbf{0} \leq D' \leq D$ implies $D' \in \mathcal{P}$ [49], the lemma above implies

$$\min\{s \geq 1 : \frac{1}{s}\mathcal{F} \subseteq \mathcal{P}\} = \min\{s \geq 1 : \frac{1}{s}\mathcal{F}_{\text{TIGHT}} \subseteq \mathcal{P}\}.$$

Thus, to prove a 4/3 upper bound on required speedup, it suffices to show for each vertex v of $\mathcal{F}_{\text{TIGHT}}$, we have $(3/4)v \in \mathcal{P}$.

We prove the following lemma, which gives a useful property of vertices of $\mathcal{F}_{\text{TIGHT}}$.

Lemma 9 *For the* 8×8 *Banyan network, for any vertex* v *of* $\mathcal{F}_{TIGHT}$ *, any* $i \in [1,4]$*, and any* $j, j' \in [1,8]$ *with* $j \neq j'$*, at least one entry in the set* $S := \{(2i-1,j),(2i,j),(2i-1,j'),(2i,j')\}$ *must have value* 0 *in* v*.*

Proof: For every clique in $\mathcal{Q}$ its intersection with S is either the empty set, S, $\{(2i-1,j),(2i,j)\}$, $\{(2i-1,j'),(2i,j')\}$,$\{(2i-1,j),(2i-1,j')\}$, or $\{(2i,j),(2i,j')\}$. If for each entry in S, the value in v were positive, we could increase the value of each entry in $\{(2i-1,j),(2i,j')\}$ by some small $\delta > 0$, and decrease the value of each entry in $\{(2i-1,j'),(2i,j)\}$ by δ, so that all still had values in $(0,1)$. Since the sum over all the other clique constraints for cliques in $\mathcal{Q}$ would be unchanged, the resulting v' would be in $\mathcal{F}_{\text{TIGHT}}$. Similarly, we could decrease the value of each entry in $\{(2i-1,j),(2i,j')\}$ by some small $\delta' > 0$, and increase the value of each entry in $\{(2i-1,j'),(2i,j)\}$ by δ', so that all still had values in $(0,1)$. This proves that v is not a vertex [13], contradicting our assumption, so the lemma holds. □

2. The Number of Linearly Independent Maximum-Clique Constraints

We now consider the set of linear constraints that define the polytope $\mathcal{F}_{\text{TIGHT}}$. We have $F \in \mathcal{F}_{\text{TIGHT}}$ iff all of the following hold:

Maximum-Clique Constraints:[5] For each $Q_{h,k} \in \mathcal{Q}$, we have $\sum_{(i,j)\in Q_{h,k}} F_{ij} = 1$.

(4.4)

[5]Recall that we justified calling $\mathcal{Q}$ the set of maximum cliques, since as discussed just befor the proof of Lemma 8 above, every clique in the link graph is contained in a clique in $\mathcal{Q}$, and each clique in $\mathcal{Q}$ contains 2^m nodes.

Non-negativity Constraints: For each (i, j), we have $F_{ij} \geq 0$.

For the 8×8 Banyan network, there are 32 maximum-clique constraints and 64 non-negativity constraints. However, some of the maximum-clique constraints are redundant. We specify a set $\mathcal{C} \subseteq \mathcal{Q}$ of 20 cliques whose corresponding maximum-clique constraints are linearly independent, and for which the other 12 maximum-clique constraints are linear combinations of these 20 constraints.

$$\begin{aligned}\mathcal{C} \;:=\; & \{Q_{0,0}, Q_{0,1}, Q_{0,2}, Q_{0,3}Q_{0,4}, Q_{0,5}, Q_{0,6}, Q_{0,7},\\ & Q_{1,0}, Q_{1,2}, Q_{1,4}, Q_{1,6},\\ & Q_{2,0}, Q_{2,2}, Q_{2,4}, Q_{2,6},\\ & Q_{3,0}, Q_{3,2}, Q_{3,4}, Q_{3,6}.\}\end{aligned}$$

We verified using **cdd+** that the set of maximum-clique constraints corresponding to the cliques in $\mathcal{C}$ are linearly independent. It may be helpful to consult Figure 4-8 to visualize the structure of the cliques in the next lemma.

Lemma 10 *For the 8×8 Banyan network, the maximum-clique constraint corresponding to each clique in $\mathcal{Q}\backslash\mathcal{C}$ can be expressed as a linear combination of maximum-clique constraints corresponding to cliques in $\mathcal{C}$.*

Proof: For $k \in \{1, 3, 5, 7\}$, the maximum-clique constraint corresponding to clique $Q_{1,k}$ is implied by the maximum-clique constraints for $Q_{1,k-1}, Q_{0,k-1}, Q_{0,k}$. In particular, for such k and for any 8×8 matrix (x_{ij}), we have $\sum_{(i,j)\in Q_{1,k}} x_{ij} = \sum_{(i,j)\in Q_{0,k-1}} x_{ij} + \sum_{(i,j)\in Q_{0,k}} x_{ij} - \sum_{(i,j)\in Q_{1,k-1}} x_{ij}$. Similarly, for $k \in \{1, 5\}$, the maximum-clique constraint corresponding to clique $Q_{2,k}$ is implied by the maximum-clique constraints for $Q_{2,k-1}, Q_{1,k-1}, Q_{1,k+1}$. For $k \in \{3, 7\}$, the maximum-clique constraint corresponding to clique $Q_{2,k}$ is implied by the constraints for cliques $Q_{2,k-1}, Q_{1,k-2}, Q_{1,k}$. Again similarly, for $k \in \{1, 3, 5, 7\}$, the maximum-clique constraint corresponding to clique $Q_{3,k}$ is implied by the maximum-clique constraints for cliques $Q_{3,k-1}, Q_{2,(k-1)/2}, Q_{2,(k+7)/2}$. □

Since there are 20 linearly independent maximum-clique constraints, any vertex

of $\mathcal{F}_{\text{TIGHT}}$ can have at most 20 non-zero entries [13]. We use this fact to prove the following lemma. Again, it may be helpful to consult Figure 4-8 to visualize the structure of the cliques described in the lemma.

Lemma 11 *Any vertex v of $\mathcal{F}_{TIGHT}$ for the 8×8 Banyan network has at least one of the following properties:*

1. There is an entry (i, j) such that $v_{ij} = 1$.

2. For some $k \in \{0, 2, 4, 6\}$, for each clique $Q_{0,k}, Q_{0,k+1}, Q_{1,k}, Q_{1,k+1}$, there are exactly two entries whose corresponding values in v are positive.

3. For some $k \in \{0, 1, 2, 3\}$, for each clique $Q_{2,k}, Q_{2,k+4}, Q_{3,2k}, Q_{3,2k+1}$, there are exactly two entries whose corresponding values in v are positive.

4. For every pair of rows $2i - 1, 2i$, one of the rows has exactly two entries with positive value in v, and the other row has exactly three entries with positive value in v. Also, the same is true for every pair of columns $2j - 1, 2j$. Also, for each $Q_{h,k} \in \mathcal{Q}$, there are at least two entries in $Q_{h,k}$ whose corresponding values in v are positive.

Proof: We show that any vertex v of $\mathcal{F}_{\text{TIGHT}}$ that does not have Properties 1, 2, or 3 above, must have Property 4. Assume v does not have Property 1; that is, for all $i, j \in \{1, \ldots, 8\}, v_{ij} < 1$. Since v satisfies the maximum-clique constraints (4.4), we have that in each clique $Q_{h,k} \in \mathcal{Q}$, there are at least two nodes whose corresponding value in v is positive. If we further assume v does not have Property 2 above, then for each $i \in \{1, 2, 3, 4\}$, the pair of rows $2i - 1, 2i$ must contain at least 5 entries with positive value in v; since v is a vertex, it can have at most 20 entries with positive value, so each pair of rows $2i - 1, 2i$ must contain *exactly* 5 entries with positive value in v. Since each row must contain at least two entries with positive value (otherwise Property 1 holds, which we assumed was not the case), we have for each pair of rows $2i - 1, 2i$ that one must contain exactly two entries with positive values in v and the other must contain three such entries. If we further assume v does not have

Property 3, a similar argument can be applied to every pair of columns $2j-1, 2j$, showing that v must have property 4. □

We use the above lemma, combined with symmetries of the link graph discussed in the next step, to classify the vertices of $\mathcal{F}_{\text{TIGHT}}$ in step 4.

3. The Automorphism Group of the Link Graph

We reduce the number of vertices of $\mathcal{F}_{\text{TIGHT}}$ that need to be considered by taking advantage of certain symmetries of the link graph. This is similar to the technique used by Deza et al. [20] to characterize the vertices of the metric polytope; Deza et al. use the symmetry group of this polytope to enumerate orbits (defined below) of the vertices, as well as to characterize certain relations between these orbits. Here, we explore certain symmetries of the polytope $\mathcal{P}$, which can be expressed in terms of the automorphism group of the link graph.

The *automorphism group* Γ for a graph $G = (V, E)$ with nodes V and edges E is defined to be the set of bijections from V to V that preserve edges; that is, a bijection $\gamma : V \to V$ is in Γ iff for all pairs of nodes $n_1, n_2 \in V$ (with $n_1 \neq n_2$) there is an edge in E between n_1 and n_2 if and only if there is an edge in E between $\gamma(n_1)$ and $\gamma(n_2)$. Note that Γ so defined is indeed a group. For a matrix F with an entry corresponding to each node in V, we let $\gamma(F)$ denote the matrix with (i, j) entry equal to $F_{\gamma((i,j))}$. For a set T of such matrices, we let $\Gamma(T) := \{\gamma(F)\}_{F \in T, \gamma \in \Gamma}$ denote the *orbit* of the set T under Γ. For any $\gamma \in \Gamma$, for any S a stable set (clique) of nodes in V, we have $\gamma(S)$ is also a stable set (clique) of nodes in V. The main property of the automorphism group that we use below is stated in the following lemma.

Lemma 12 *For Γ the automorphism group of the link graph of a $2^m \times 2^m$ Banyan network, for any $\gamma \in \Gamma$, any $F \in \mathcal{F}$, and any $s \geq 1$, we have $(1/s)F \in \mathcal{P}$ iff $(1/s)\gamma(F) \in \mathcal{P}$.*

Proof: If $(1/s)F$ can be expressed as a convex combination $\sum \lambda_j R^{(j)}$ of vertices of $\mathcal{P}$, then $(1/s)\gamma(F)$ can be expressed as the convex combination $\sum \lambda_j \gamma(R^{(j)})$ of vertices of $\mathcal{P}$; here, for each j, $\gamma(R^{(j)})$ is a valid packetized matrix (that is, the entries with value 1 form a stable set in the link graph) since automorphisms of the link graph

map stable sets to stable sets. □

The lemma above implies for $T \subseteq \mathcal{F}_{\text{TIGHT}}$ such that $\Gamma(T)$ is the set of vertices of $\mathcal{F}_{\text{TIGHT}}$, we have

$$\min\{s \geq 1 : \frac{1}{s}\mathcal{F}_{\text{TIGHT}} \subseteq \mathcal{P}\} \quad = \quad \min\{s \geq 1 : \frac{1}{s}T \subseteq \mathcal{P}\}. \tag{4.5}$$

We now consider some specific automorphisms for the link graph $G = (V, E)$ of a Banyan network. For any $w, l : 0 \leq w \leq m-1, 0 \leq l \leq 2^{m-w-1}-1$, define the bijection from V to V that swaps the consecutive sets of rows $R_1 := [(2l)2^w + 1, (2l+1)2^w]$ and $R_2 := [(2l+1)2^w + 1, (2l+2)2^w]$ as

$$\gamma^{\text{rows}}_{R_1 \leftrightarrow R_2}((i,j)) := \begin{cases} (i + 2^w, j), & \text{if } i \in R_1 \\ (i - 2^w, j), & \text{if } i \in R_2 \\ (i,j), & \text{otherwise.} \end{cases}$$

Similarly, for each $w, l : 0 \leq w \leq m-1, 0 \leq l \leq 2^{m-w-1}-1$, define the bijection from V to V that swaps the consecutive sets of columns $C_1 := [(2l)2^w + 1, (2l+1)2^w]$ and $C_2 := [(2l+1)2^w + 1, (2l+2)2^w]$ as

$$\gamma^{\text{columns}}_{C_1 \leftrightarrow C_2}((i,j)) := \begin{cases} (i, j + 2^w), & \text{if } j \in C_1 \\ (i, j - 2^w), & \text{if } j \in C_2 \\ (i,j), & \text{otherwise.} \end{cases}$$

Define the bijection $\gamma^{\text{flip}} : V \to V$ such that $\gamma_{\text{flip}}((i,j)) := (j,i)$. Lastly, let γ^{identity} denote the bijection mapping each (i,j) to itself. All the bijections defined above are automorphisms of the link graph, since for each $h, k : 0 \leq h \leq m, 0 \leq k \leq 2^m - 1$, they map the clique $Q_{h,k}$ onto a clique, and for every edge in the link graph, its endpoints are contained in at least one such clique $Q_{h,k}$. Also, each of the automorphisms defined above is its own inverse.

4. Classifying the Vertices of $\mathcal{F}_{\text{TIGHT}}$ using the Automorphism Group of the Link Graph for 8×8 Banyan Networks

For each of the four classes of vertices of $\mathcal{F}_{\text{TIGHT}}$ in Lemma 11, we find a subset

of such vertices whose orbit under the automorphism group Γ contains the original class. This will allow us, by (4.5) above, to look at each vertex v in the four resulting subsets and calculate $\min\{s \geq 1 : \frac{1}{s}v \in \mathcal{P}\}$; the largest value obtained is then the speedup required to maintain bounded backlog for the 8×8 Banyan network.

We consider each of the four properties in Lemma 11 in turn. We denote the set of vertices of $\mathcal{F}_{\text{TIGHT}}$ that have Property i from this lemma by T_i. For each Property i, we use the automorphism group to show for the subset $T_i' \subseteq T_i$ that satisfies certain additional properties (specified below), we have $T_i \subseteq \Gamma(T_i')$. Then we enumerate the elements of T_i' using **cdd+**.

The program **cdd+** takes a set of linear inequalities as input, and outputs the vertices of the polytope defined by these inequalities; one may specify that only vertices of the polytope that make a specified subset of the inequalities tight should be found.[6] Each time we used **cdd+**, we input the maximum-clique constraints (4.4) corresponding to each clique in $\mathcal{C}$, and the non-negativity constraints. We also specified certain non-negativity constraints to be tight, depending on the subcase being considered. In each subcase, **cdd+** produced the list of corresponding vertices in less than two days. It is necessary to consider the subsets T_i' since the number of elements in each set T_i is so large that **cdd+** could not enumerate them without the computation involved overflowing the 1Gb RAM of the computer (with 2.2 GHz processor) we used.

Vertices of $\mathcal{F}_{\text{TIGHT}}$ with Property 1

First, consider the subset T_1 of vertices of $\mathcal{F}_{\text{TIGHT}}$ for the 8×8 Banyan network such that Property 1 holds from Lemma 11. That is, for each $v \in T_1$, there is an entry (i, j) such that $v_{ij} = 1$. Let T_1' denote the set of vertices v of $\mathcal{F}_{\text{TIGHT}}$ for which $v_{11} = 1$ and $v_{55} = v_{77} = 0$. We will prove the following lemma.

Lemma 13 $T_1 = \Gamma(T_1')$.

Proof: Since automorphisms of the link graph are bijections we have $\Gamma(T_1') \subseteq T_1$. To show the opposite inclusion, take any $v \in T_1$, and let (i, j) be some entry such

[6]The **cdd+** software can also perform more general computations, but we describe only the functions we used.

that $v_{ij} = 1$. It suffices to consider v such that $i \leq 4$ by the following argument: If we show for each v with corresponding $i \leq 4$ that there exists a $v' \in T_1'$ and $\gamma \in \Gamma$ such that $\gamma(v') = v$, then this would imply for any $v \in T_1$ with $i \geq 5$ there exist a $v' \in T_1', \gamma \in \Gamma$ such that $\gamma(v') = \gamma^{\text{rows}}_{\{1,2,3,4\}\leftrightarrow\{5,6,7,8\}}(v)$; thus, $\gamma^{\text{rows}}_{\{1,2,3,4\}\leftrightarrow\{5,6,7,8\}} \circ \gamma(v') = v$, so $v \in \Gamma(T_1')$, which would imply $T_1 \subseteq \Gamma(T_1')$ thus proving the lemma. We use a similar argument many times below, but omit the detailed explanation, indicating only the automorphism(s) involved.

Arguing as above, it suffices to consider v such that $i \leq 2$, since for v with $i \in [3,4]$, we have $\gamma^{\text{rows}}_{\{1,2\}\leftrightarrow\{3,4\}}(v)$ has an element in the first two rows with value 1. Similarly, it suffices to consider v such that $i = 1$, since for v with $i = 2$, we have $\gamma^{\text{rows}}_{\{1\}\leftrightarrow\{2\}}(v)$ has an element in the first row with value 1.

We can apply a similar procedure to the columns, showing it suffices to consider $v \in T_1$ such that $v_{11} = 1$. By Lemma 9, one of the entries $(5,5), (6,5), (5,6), (6,6)$ in v has value 0. Using a subset of the automorphisms $\{\gamma^{\text{rows}}_{\{5\}\leftrightarrow\{6\}}, \gamma^{\text{columns}}_{\{5\}\leftrightarrow\{6\}}\}$, it suffices to consider v with value 0 at $(5,5)$. By a similar argument, using a subset of the automorphisms $\{\gamma^{\text{rows}}_{\{7\}\leftrightarrow\{8\}}, \gamma^{\text{columns}}_{\{7\}\leftrightarrow\{8\}}\}$, it suffices to consider v with value 0 at $(7,7)$. In summary, we have argued that it suffices to consider $v \in T_1$ such that $v_{11} = 0$ and $v_{55} = v_{77} = 0$. Such a v is in T_1', and so is in $\Gamma(T_1')$. We have thus shown $T_1 \subseteq \Gamma(T_1')$, proving the lemma. □

We enumerated each vertex in T_1' using **cdd+**. There are 17148 such vertices. An example of a vertex in T_1' is given in Figure 4-9.

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

Figure 4-9: A vertex of $\mathcal{F}_{\text{TIGHT}}$ in T_1'.

Vertices of $\mathcal{F}_{\text{TIGHT}}$ with Property 2

Next, consider the subset T_2 of vertices of $\mathcal{F}_{\text{TIGHT}}$ for the 8×8 Banyan network such

that Property 2 holds from Lemma 11. That is, for some $k \in \{0, 2, 4, 6\}$, for each clique $Q_{0,k}, Q_{0,k+1}, Q_{1,k}, Q_{1,k+1}$, there are exactly two entries whose corresponding values in v are positive. We define $T_2' \subseteq T_2$ to be the set of vertices v of $\mathcal{F}_{\text{TIGHT}}$ such that for one of the nine subcases $1a, 1b, 1c, 2a, 2b, 2c, 2d, 2e, 2f$ below, the entries of the 2×8 matrix in this subcase marked with a $+$ correspond to the positive-valued entries of the first two rows of v, and the entries marked with a 0 correspond to the 0-valued entries in the first two rows of v.

Case 1a: $\begin{bmatrix} + & + & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & + & + & 0 & 0 \end{bmatrix}$,

Case 1b: $\begin{bmatrix} + & + & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & + & 0 & + & 0 \end{bmatrix}$,

Case 1c: $\begin{bmatrix} + & 0 & + & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & + & 0 & + & 0 \end{bmatrix}$,

Case 2a: $\begin{bmatrix} + & 0 & 0 & 0 & + & 0 & 0 & 0 \\ + & 0 & 0 & 0 & + & 0 & 0 & 0 \end{bmatrix}$,

Case 2b: $\begin{bmatrix} + & 0 & 0 & 0 & + & 0 & 0 & 0 \\ + & 0 & 0 & 0 & 0 & + & 0 & 0 \end{bmatrix}$,

Case 2c: $\begin{bmatrix} + & 0 & 0 & 0 & + & 0 & 0 & 0 \\ + & 0 & 0 & 0 & 0 & 0 & + & 0 \end{bmatrix}$,

Case 2d: $\begin{bmatrix} + & 0 & 0 & 0 & + & 0 & 0 & 0 \\ 0 & + & 0 & 0 & 0 & + & 0 & 0 \end{bmatrix}$,

Case 2e: $\begin{bmatrix} + & 0 & 0 & 0 & + & 0 & 0 & 0 \\ 0 & + & 0 & 0 & 0 & 0 & + & 0 \end{bmatrix}$,

Case 2f: $\begin{bmatrix} + & 0 & 0 & 0 & + & 0 & 0 & 0 \\ 0 & 0 & + & 0 & 0 & 0 & + & 0 \end{bmatrix}$.

We now prove the following lemma.

Lemma 14 $T_2 \subseteq \Gamma(T_2')$.

Proof: It suffices to consider the set of $v \in T_2$ such that for each clique $Q_{0,0}, Q_{0,1}, Q_{1,0}, Q_{1,1}$, there are exactly two entries whose corresponding values in v are positive. This follows since if Property 2 holds from Lemma 11 for some $k \in \{2,4,6\}$, then using a subset of automorphisms $\{\gamma^{\text{rows}}_{\{1,2,3,4\}\leftrightarrow\{5,6,7,8\}}, \gamma^{\text{rows}}_{\{1,2\}\leftrightarrow\{3,4\}}\}$ as in the proof of Lemma 13, we can find a $\gamma \in \Gamma$ such that Property 2 holds for $k = 0$ for $\gamma(v)$. In arguing below that it suffices to consider $v \in T_2$ with certain additional properties, we employ automorphisms in Γ that preserve the properties previously argued for; by the end of the argument, we will have shown that it suffices to consider $v \in T_2$ that have the properties defining T_2', thus proving the lemma.

It suffices to consider the set of $v \in T_2$ such that not only for each clique $Q_{0,0}, Q_{0,1}, Q_{1,0}, Q_{1,1}$ are there exactly two entries whose corresponding values in v are positive, but also $v_{11} > 0$. The latter property can be obtained by using a subset of the automorphisms $\{\gamma^{\text{columns}}_{\{1,2\}\leftrightarrow\{3,4\}}, \gamma^{\text{columns}}_{\{1\}\leftrightarrow\{2\}}, \gamma^{\text{rows}}_{\{1\}\leftrightarrow\{2\}}\}$. We now have that one of the positive-valued entries in v among the entries in $Q_{1,0}$ is $(1,1)$, and from above we have that there are exactly two positive-valued entries in v among $Q_{1,0}$. We consider two cases below (each with subcases), which depend on the row that the positive-valued entry of v among the entries $Q_{1,0} \setminus \{(1,1)\}$ is in.

Case 1: For (i_1, j_1) the entry in $Q_{1,0} \setminus \{(1,1)\} = [1,2] \times [1,4] \setminus \{(1,1)\}$ with positive value in v, we have $i_1 = 1$.

It suffices to consider $(i_1, j_1) \in \{(1,2),(1,3)\}$, since if $(i_1, j_1) = (1,4)$, then $\gamma^{\text{columns}}_{\{3\}\leftrightarrow\{4\}}(v)$ has positive entries at $(1,1)$ and $(1,3)$.

Recall that in each of $Q_{0,0} = \{1\} \times [1,8], Q_{0,1} = \{2\} \times [1,8], Q_{1,0} = [1,2] \times [1,4]$, there are exactly two entries with positive value in v; in this case, the two entries with positive value in row 2 must occur in a subset of columns $[5,8]$. By similar arguments as above, using a subset of the automorphisms $\{\gamma^{\text{columns}}_{\{5,6\}\leftrightarrow\{7,8\}}, \gamma^{\text{columns}}_{\{5\}\leftrightarrow\{6\}}, \gamma^{\text{columns}}_{\{7\}\leftrightarrow\{8\}}\}$, we can assume $v_{2,5} > 0$, and the other entry in row two with positive value (i_2, j_2) is in $\{(2,6),(2,7),(2,8)\}$. It suffices to consider v such that $(i_2, j_2) \in \{(2,6),(2,7)\}$, since if $(i_2, j_2) = (2,8)$, then

$\gamma^{\text{columns}}_{\{7\}\leftrightarrow\{8\}}(v)$ has positive entries at $(2,5)$ and $(2,7)$. Also, it suffices to consider v for which $j_1 \leq j_2 - 4$, since for v such that this is not the case, $\gamma^{\text{rows}}_{\{1\}\leftrightarrow\{2\}} \circ \gamma^{\text{columns}}_{\{1,2,3,4\}\leftrightarrow\{5,6,7,8\}}(v)$ has this property, and preserves the other properties we have discussed.

Thus, we have argued that it suffices to consider $v \in T_2$ such that: $v_{11} > 0; v_{25} > 0$; of the two remaining positive-valued entries in the first two rows of v, one (i_1, j_1) is in $\{(1,2),(1,3)\}$ and the other (i_2, j_2) is in $\{(2,6),(2,7)\}$; and $j_1 \leq j_2 - 4$. In other words, v belongs to Case 1a, 1b, or 1c in the definition of T_2' above.

Case 2: For (i_1, j_1) the entry in $Q_{1,0} \setminus \{(1,1)\} = [1,2] \times [1,4] \setminus \{(1,1)\}$ with positive value in v, we have $i_1 = 2$.

It suffices to consider $(i_1, j_1) \in \{(2,1),(2,2),(2,3)\}$, since if $(i_1, j_1) = (2,4)$, then $\gamma^{\text{columns}}_{\{3\}\leftrightarrow\{4\}}(v)$ has positive entries at $(1,1)$ and $(2,3)$.

Of the two entries with positive value in $Q_{1,1} = [1,2] \times [5,8]$, one must be in the first row and the other in the second in order that there are exactly two entries with positive value in v in each row, which we assumed above was the case. Let $(1, j')$ be one such entry and $(2, j_2)$ be the other. By similar arguments as above, using a subset of $\{\gamma^{\text{columns}}_{\{5,6\}\leftrightarrow\{7,8\}}, \gamma^{\text{columns}}_{\{5\}\leftrightarrow\{6\}}\}$, we can assume $j' = 5$. Also, it suffices to consider v such that $(2, j_2) \in \{(2,5),(2,6),(2,7)\}$ since if $(2, j_2) = (2,8)$, then $\gamma^{\text{columns}}_{\{7\}\leftrightarrow\{8\}}(v)$ has this property. Lastly, it suffices to consider v for which $j_1 \leq j_2 - 4$, since for v such that this is not the case, $\gamma^{\text{columns}}_{\{1,2,3,4\}\leftrightarrow\{5,6,7,8\}}(v)$ has this property, and preserves the other properties we have discussed.

Thus, we have argued that it suffices to consider $v \in T_2$ such that: $v_{11} > 0$; $v_{15} > 0$; of the two remaining positive-valued entries in the first two rows of v, one is $(2, j_1) \in \{(2,1),(2,2),(2,3)\}$ and the other is $(2, j_2) \in \{(2,5),(2,6),(2,7)\}$; and $j_1 \leq j_2 - 4$. In other words, v belongs to Case 2a,2b,2c,2d,2e, or 2f in the definition of T_2' above.

In summary, we argued above that in proving $T_2 \subseteq \Gamma(T_2')$, it suffices to show for each $v \in T_2$ corresponding to one of the nine subcases above, that $v \in \Gamma(T_2')$. Since

every vertex of $\mathcal{F}_{\text{TIGHT}}$ corresponding to one of the nine subcases above is by definition in T_2', we have proved the lemma. □

We enumerated each vertex in T_2' by using **cdd+** to enumerate the vertices corresponding to each subcase $1a, 1b, 1c, 2a, 2b, 2c, 2d, 2e, 2f$ separately.[7] Figure 4-10 gives an example of two vertices in T_2'.

$$\begin{bmatrix} \frac{2}{3} & 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & \frac{2}{3} & 0 \\ 0 & 0 & \frac{2}{3} & 0 & \frac{1}{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & \frac{1}{3} \\ \frac{1}{3} & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & \frac{2}{3} \\ 0 & \frac{1}{3} & 0 & 0 & 0 & \frac{2}{3} & 0 & 0 \\ 0 & 0 & 0 & \frac{2}{3} & 0 & 0 & \frac{1}{3} & 0 \end{bmatrix}, \begin{bmatrix} \frac{2}{7} & 0 & 0 & 0 & \frac{5}{7} & 0 & 0 & 0 \\ 0 & 0 & \frac{5}{7} & 0 & 0 & 0 & \frac{2}{7} & 0 \\ \frac{2}{7} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{5}{7} \\ 0 & \frac{3}{7} & 0 & \frac{2}{7} & 0 & \frac{2}{7} & 0 & 0 \\ 0 & \frac{4}{7} & \frac{2}{7} & 0 & 0 & 0 & 0 & \frac{1}{7} \\ 0 & 0 & 0 & \frac{1}{7} & 0 & \frac{5}{7} & \frac{1}{7} & 0 \\ 0 & 0 & 0 & \frac{4}{7} & \frac{2}{7} & 0 & 0 & \frac{1}{7} \\ \frac{3}{7} & 0 & 0 & 0 & 0 & 0 & \frac{4}{7} & 0 \end{bmatrix}$$

Figure 4-10: Two vertices of $\mathcal{F}_{\text{TIGHT}}$ in T_2'. The vertex v on the left is in subcase 2e of T_2'; this vertex v is such that $\min\{s \geq 1 : (1/s)v \in \mathcal{P}\} = 4/3$. The vertex on the right is in subcase 2f of T_2'.

Vertices of $\mathcal{F}_{\text{TIGHT}}$ with Property 3

Note that for T_3 the set of vertices of $\mathcal{F}_{\text{TIGHT}}$ for the 8×8 Banyan network such that Property 3 holds from Lemma 11, we have $T_3 \subseteq \Gamma(T_2)$, because for any $v \in T_3$, we have $\gamma^{\text{flip}}(v) \in T_2$. This, and Lemma 14, imply the following lemma.

Lemma 15 $T_3 \subseteq \Gamma(T_2')$.

Since we enumerated the elements of T_2' already, no new vertices need to be enumerated in this case.

[7] In order to reduce the run-time of **cdd+** on each of the above nine cases, we specified that certain entries (which depend on the subcase being computed) among the bottom six rows $[3,8] \times [1,8]$ be set to 0. For subcases 2a,2b,2d, we set $v_{53} = v_{77} = 0$; by Lemma 9 any vertex of $\mathcal{F}_{\text{TIGHT}}$ has value 0 in at least one entry in $\{(5,3),(5,4),(6,3),(6,4)\}$, and using automorphisms in $\{\gamma^{\text{columns}}_{\{3\}\leftrightarrow\{4\}}, \gamma^{\text{rows}}_{\{5\}\leftrightarrow\{6\}}\}$ (which do not affect the set of positions in the first two rows with positive entries in v) it suffices to consider vertices v in which $v_{53} = 0$. A similar argument implies that it suffices to consider vertices v such that $v_{77} = 0$. Using similar arguments, we justify setting $v_{53} = 0$ in subcases $2c$,$2e$; setting $v_{55} = v_{77} = 0$ in subcases 1a, 1c; and setting $v_{53} = v_{75} = 0$ in subcase 1b. In each of the nine subcases defining T_2', we specified to **cdd+** which entries must be set to 0, but did not specify that the entries corresponding to a + be strictly positive; thus, the output in each case is the set of vertices of $\mathcal{F}_{\text{TIGHT}}$ with 0's in the first two rows as specified by the case and with some entries set to 0 in the other rows as specified in this footnote. The number of vertices output by **cdd+** in each case is given in the following table.

Case:	1a	1b	1c	2a	2b	2c	2d	2e	2f
Vertices:	10792	33252	131452	1548	2796	11248	14776	70640	401552

Vertices of $\mathcal{F}_{\text{TIGHT}}$ with Property 4

Lastly, we consider the vertices $v \in T_4$ of $\mathcal{F}_{\text{TIGHT}}$ for the 8×8 Banyan network such that Property 4 holds from Lemma 11. That is, for every pair of rows $2i-1, 2i$, one of the rows has exactly two entries with positive value in v, and the other row has exactly three entries with positive value in v. Also, the same is true for every pair of columns $2j-1, 2j$. Also, for each $Q_{h,k} \in \mathcal{Q}$, there are at least two entries in $Q_{h,k}$ whose corresponding values in v are positive.

Using a subset of the automorphisms $\{\gamma^{\text{rows}}_{\{1\}\leftrightarrow\{2\}}, \gamma^{\text{rows}}_{\{3\}\leftrightarrow\{4\}}, \gamma^{\text{rows}}_{\{5\}\leftrightarrow\{6\}}, \gamma^{\text{rows}}_{\{7\}\leftrightarrow\{8\}}\}$, it suffices to consider v such that for every pair of rows $2i-1, 2i$, row $2i-1$ has exactly two entries with positive value in v, and row $2i$ has exactly three entries with positive value in v. Similarly, it suffices to consider v that in addition to the properties already mentioned, have the property in the previous sentence for each pair of columns $2j-1, 2j$.

Let T_4' denote the set of vertices satisfying the above properties and all the properties described in the First Case or all the properties described in the Second Case below:

First Case: The number of entries in $Q_{1,0}$ with positive value in v equals the number of entries in $Q_{1,2}$ with positive value in v.

It suffices to consider v for which there are exactly two entries in $Q_{1,0}$ (and so also in $Q_{1,2}$) with positive value. This follows since the only other possible number of such entries is three, and in this case $\gamma^{\text{rows}}_{\{1,2,3,4\}\leftrightarrow\{5,6,7,8\}}(v)$ has exactly two entries in $Q_{1,0}$ and also in $Q_{1,2}$ with positive value.

Second Case: The number of entries in $Q_{1,0}$ with positive value in v does not equal the number of entries in $Q_{1,2}$ with positive value in v.

One of the cliques $Q_{1,0}, Q_{1,2}$ must have exactly two entries with positive value in v, and the other must have exactly three such entries. It suffices to consider v for which exactly two entries in $Q_{1,0}$ have positive value, since otherwise $\gamma^{\text{rows}}_{\{1,2\}\leftrightarrow\{3,4\}}(v)$ has this property. Similarly, since one of the cliques $Q_{2,0}, Q_{2,1}$ has exactly two entries with positive value in v, and the other has exactly

three such entries, it suffices to consider v for which exactly two entries in $Q_{2,0}$ have positive value (otherwise $\gamma^{\text{columns}}_{\{1,2\}\leftrightarrow\{3,4\}}(v)$ has this property). Similarly, it suffices to consider v for which exactly two entries in $Q_{2,3}$ have positive value, since otherwise $\gamma^{\text{columns}}_{\{5,6\}\leftrightarrow\{7,8\}}(v)$ has this property. Lastly, it suffices to consider v for which exactly two entries in $Q_{1,4}$ have positive value, since otherwise $\gamma^{\text{rows}}_{\{5,6\}\leftrightarrow\{7,8\}}(v)$ has this property.

The above argument implies the following lemma.

Lemma 16 $T_4 \subseteq \Gamma(T'_4)$.

Two examples of vertices in T'_4 are given in Figure 4-11.

$$\begin{bmatrix} \frac{2}{3} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{3} \\ 0 & 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & \frac{1}{3} \\ 0 & 0 & \frac{2}{3} & 0 & 0 & \frac{1}{3} & 0 & 0 \\ 0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 \\ \frac{1}{3} & 0 & 0 & 0 & \frac{2}{3} & 0 & 0 & 0 \\ 0 & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 & 0 & \frac{1}{3} \\ 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & \frac{2}{3} & 0 \\ 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 & \frac{3}{5} & \frac{2}{5} & 0 & 0 & 0 \\ 0 & \frac{2}{5} & 0 & 0 & 0 & \frac{2}{5} & 0 & \frac{1}{5} \\ 0 & 0 & 0 & \frac{1}{5} & 0 & 0 & \frac{4}{5} & 0 \\ \frac{3}{5} & 0 & \frac{1}{5} & 0 & 0 & \frac{1}{5} & 0 & 0 \\ 0 & 0 & \frac{4}{5} & 0 & 0 & 0 & \frac{1}{5} & 0 \\ 0 & \frac{1}{5} & 0 & 0 & \frac{3}{5} & 0 & 0 & \frac{1}{5} \\ 0 & \frac{2}{5} & 0 & 0 & 0 & 0 & 0 & \frac{3}{5} \\ \frac{2}{5} & 0 & 0 & \frac{1}{5} & 0 & \frac{2}{5} & 0 & 0 \end{bmatrix}$$

Figure 4-11: Two vertices of $\mathcal{F}_{\text{TIGHT}}$ in T'_4. The vertex on the left is in the first case of T'_4. The vertex on the right is in the second case of T'_4.

We did not use **cdd+** in this case, but instead enumerated the vertices of T'_4 directly. One way to enumerate each element of T'_4 is to first enumerate each possible subset S of 20 elements of $\{(i,j) : 1 \leq i \leq 8, 1 \leq j \leq 8\}$ such that for each $k \in [1,4]$, there are exactly two elements of S in row $2k-1$, three elements of S in row $2k$, two elements of S in column $2k-1$, three elements of S in column $2k$, and each clique $Q_{h,k} \in \mathcal{Q}$ contains at least two elements of S. For such an S, there is at most one vertex of $\mathcal{F}_{\text{TIGHT}}$ with positive values in each entry of S, and if there is one it can be found by a single matrix inversion; this follows since there are exactly 20 linearly independent maximum-clique constraints corresponding to the set of maximum-cliques $\mathcal{C}$ defined in step 2, that in addition to the non-negativity constraints define $\mathcal{F}_{\text{TIGHT}}$. There are 1344 vertices in the First Case above, and 243 in the Second Case above.

We let $T := T'_1 \cup T'_2 \cup T'_4$. By the lemmas above, we have that the set of vertices of $\mathcal{F}_{\text{TIGHT}}$ is contained in $\Gamma(T)$.

5. Testing each vertex in T

We enumerated each element of the subset T of vertices of $\mathcal{F}_{\text{TIGHT}}$ in step 4 above. We now calculate $\min\{s \geq 1 : \frac{1}{s}v \in \mathcal{P}\}$ for each $v \in T$. This can be done by solving the following linear program, which we call LP1, for each $v \in T$:

$\min \sum \lambda_i$ such that

$v \leq \sum \lambda_i D^{(i)}$,

$\forall i, \lambda_i \geq 0$,

where $\{D^{(i)}\}$ is the set of vertices of $\mathcal{P}$. This linear program has, in addition to the non-negativity constraints, 64 inequality constraints–one for each entry in v. The number of variables λ_i is the number of vertices of $\mathcal{P}$, which is quite large. By the definition of $\mathcal{P}$, its vertices are 8×8, $\{0, 1\}$-valued matrices that satisfy the clique constraints (4.1). By Lemma 8, each vertex $D^{(i)}$ of $\mathcal{P}$ can be dominated by a vertex of $\mathcal{P}$ for which the maximum-clique constraints (4.4) are tight. We then have that the following linear program LP2 has the same optimum value as LP1 above:

$\min \sum \lambda_j$ such that

$v \leq \sum \lambda_j M^{(j)}$,

$\forall j, \lambda_j \geq 0$,

where $\{M^{(j)}\}$ is the set of vertices of $\mathcal{P}$ for which the maximum-clique constraints (4.4) are tight. Here, the number of variables λ_j is the number of such vertices, which we show in the following lemma is 4096.

Lemma 17 *For the 8×8 Banyan network, there are exactly $4096 = 2^{12}$ vertices of $\mathcal{P}$ that satisfy the maximum-clique constraints (4.4).*

Proof: For each vertex $M^{(j)}$ of $\mathcal{P}$ that satisfies the maximum-clique constraints, there is exactly one entry in each maximum-clique $Q_{h,k} \in \mathcal{Q}$ with value 1 (the other 7 entries have value 0). Thus, there are exactly eight entries in $M^{(j)}$ with value 1. We count how many ways there are to choose the eight entries in $M^{(j)}$ with value 1.

There are eight entries in $Q_{1,0} = [1,2] \times [1,4]$, each of which could be set to 1. Having set the values of $M^{(j)}$ for entries in $Q_{1,0}$, there are four possible entries in $Q_{1,2} = [3,4] \times [1,4]$ that could be set to 1 (due to $Q_{2,0}$ and $Q_{2,1}$). Similarly, there are eight possible ways to set the values in entries $Q_{1,5} = [5,6] \times [5,8]$. Having set the values of $M^{(j)}$ for entries in $Q_{1,5}$, there are four possible entries in $Q_{1,7} = [7,8] \times [5,8]$ that could be set to 1 (due to $Q_{2,6}$ and $Q_{2,7}$). We have shown that there are $(8 \cdot 4)^2 = 2^{10}$ ways to set the values of $M^{(j)}$ in entries $([1,4] \times [1,4]) \cup ([5,8] \times [5,8])$.

Having set the values of $M^{(j)}$ in entries $([1,4] \times [1,4]) \cup ([5,8] \times [5,8])$, we now examine how many ways there are to set the values of the remaining entries. The two columns corresponding to the two entries in $[1,4] \times [1,4]$ that have been set to 1 cannot contain another entry with a 1 (due to $Q_{3,0}, Q_{3,1}, Q_{3,2}, Q_{3,3}$). Similarly, the two rows corresponding to the two entries in $[5,8] \times [5,8]$ that have been set to 1 cannot contain another entry with a 1 (due to $Q_{0,4}, Q_{0,5}, Q_{0,6}, Q_{0,7}$). This leaves two columns and two rows in $[5,8] \times [1,4]$ that could have an entry set to 1. Since each row (and each column) can have at most a single entry set to 1, there are 2 ways to select the two entries in $[5,8] \times [1,4]$ to be set to 1. A symmetric argument gives that there are 2 ways to select the two entries in $[1,4] \times [5,8]$ to be set to 1. Thus, there are a total of exactly $2^{10} \cdot 2^2 = 4096$ ways to set the entries in $M^{(j)}$, showing that there are 4096 vertices of $\mathcal{P}$ that satisfy all the maximum-clique constraints. □

A way to reduce the size of the linear program LP2 is to include only the inequality constraints corresponding to entries in v that are positive valued (in addition to the non-negativity constraints), since the inequality constraints corresponding to 0-valued entries of v are implied by the non-negativity constraints. Since as shown in step 2 above, there are at most 20 entries in any vertex v of $\mathcal{F}_{\text{TIGHT}}$ that have positive value, we can reduce the number of inequalities connected with the maximum-clique constraints to 20.

We solved the resulting linear program for each $v \in T$ using Matlab, and found that the maximum over all $v \in T$ of the optimum value is 4/3. We thus determined that 4/3 is an upper bound on the speedup required to maintain bounded backlog for 8×8 Banyan networks.

4.7 A Greedy Algorithm for Decomposing Fluid Matrices

We now give a greedy algorithm for decomposing any fluid matrix on a Banyan network. This algorithm is an extension of the maximal matching algorithm used by Smiljanić on $N \times N$ crossbar switches [51].

Theorem 8 *For a Banyan network with N input ports, we exhibit an algorithm (Algorithm 6 below) that, for any $F \in \frac{1}{\log_2 N+1}\mathcal{F}$, decomposes F into a convex combination of N^2+1 vertices of $\mathcal{P}$; the algorithm runs in time polynomial in N.*

An immediate corollary is $\frac{1}{\log_2 N+1}\mathcal{F} \subseteq \mathcal{P}$. Using Algorithm 6 below as a subroutine in Algorithm 5, 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 9 *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.*

We now prove Theorem 8.

Proof of Theorem 8: It suffices to show for any $F \in \frac{1}{\log_2 N+1}\mathcal{F}$, that one can compute in time polynomial in N, a decomposition of $(\log_2 N + 1)F$ into a non-negative, linear combination

$$(\log_2 N + 1)F = \sum_{j=1}^{N^2} \gamma_j R^{(j)}, \tag{4.6}$$

with $\forall j, R^{(j)}$ a valid packetized matrix, and $\sum_{j=1}^{N^2} \gamma_j \leq \log_2 N + 1$. This follows since if we set $R^{(N^2+1)} := \mathbf{0}$, and $\gamma_{N^2+1} := \log_2 N + 1 - \sum_{j=1}^{N^2} \gamma_j$, we have $F = \sum_{j=1}^{N^2+1} \frac{\gamma_j}{\log_2 N+1} R^{(j)}$, the desired convex combination. The greedy algorithm below produces the decomposition (4.6). We use the notation that for a stable set S of the link graph G of a Banyan network, χ^S denotes the $N \times N$ packetized matrix with value 1 at entries corresponding to elements of S and with value 0 otherwise.

Algorithm 6

Let F be a given fluid matrix in $\frac{1}{\log_2 N+1}\mathcal{F}$.

1. Set $\hat{F} \leftarrow (\log_2 N + 1)F$ and set $j \leftarrow 1$.

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_j.
   - Set $\gamma_j \leftarrow \min_{(k,l)\in S_j} \hat{F}_{kl}$.
   - Set $R^{(j)} \leftarrow \chi^{S_j}$.
   - Set $\hat{F} \leftarrow \hat{F} - \gamma_j R^{(j)}$ and then increment j by 1.

Since at each iteration of step 2, at least one entry of $\hat{F}$ is set to 0, the algorithm terminates after at most N^2 iterations. Upon termination, we have $(\log_2 N + 1)F = \sum_{j=1}^{N^2} \gamma_j R^{(j)}$. It remains to show that the sum of γ_j's is at most $\log_2 N + 1$.

For a fixed node (k, l) in G, we denote the neighborhood of (k, l) as $\mathcal{N}((k, l)) :=$

$$\{(u, v) : (k, l) = (u, v) \text{ or an edge connects } (k, l) \text{ to } (u, v)\}.$$

Denote the last iteration of step 2 of the algorithm by i. Let (k, l) be a node in stable set S_i. For any iteration $j \leq i$ we have that S_j either contains (k, l) or contains a node (u, v) that is connected to (k, l) by an edge in the link graph. The previous sentence follows since otherwise $\{(k, l)\} \cup S_j$ is a stable set whose corresponding elements of $\hat{F}$ are positive at time step j, which would contradict S_j being maximal. Thus we have

$$\sum_{j=1}^{i} \gamma_j \quad \leq \sum_{(u,v)\in\mathcal{N}((k,l))} F_{uv}.$$

The sum on the right hand side is the sum of fluid whose path through the switch network includes at least some link in the (unique) path from input k of the first stage to output l of the last stage. Since for any valid fluid matrix (that is, any element of $\mathcal{F}$), the total fluid traversing any link in the switch network is at most 1, the total

fluid sharing at least one link with the path from k to l is at most the number of links on this path. For a Banyan network with N input ports and output ports (so with $\log_2 N$ stages), all path lengths are one more than the number of stages in the network. Thus, we have bounded $\sum_{j=1}^{i} \gamma_j$ by $\log_2 N + 1$ as desired. □

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.

We now bound the running time of Algorithm 6. First, we show that it is possible to construct all the maximal stable sets S_j from step 2 in total time $O(N^3 \log^2 N)$. As a subroutine to the construction, we check whether adding a new node to a stable set of the link graph G of a Banyan network results in a stable set. In order to do this efficiently, we keep track of which of the $N(\log_2 N + 1)$ constraints in the link graph G are *covered* by the stable set S under consideration, where we say that a set S of nodes *covers* a constraint if at least one node in S is in the corresponding clique. Then we can check, for any node (k, l) in G, whether $\{(k, l)\} \cup S$ is a stable set by checking if none of the $\log_2 N + 1$ constraints whose corresponding cliques contain (k, l) are covered by S.

We construct maximal stable set S_1 by sequentially considering each of the nodes of G with positive corresponding entries in $(\log_2 N + 1)F$, keeping those with no constraints currently covered and then setting all their constraints to "covered." This takes time $O(N^2 \log N)$. For $j \geq 2$, we construct the maximal stable set S_j by starting with the stable set S_{j-1} and removing the set of nodes (call it Z_{j-1}) whose corresponding entries in $\hat{F}$ were set to 0 during iteration $j - 1$. Also, all constraints covered by a node in Z_{j-1} are set to "not covered." We then augment $S_{j-1} \setminus Z_{j-1}$ to form a maximal stable set in G restricted to the nodes with positive value in $\hat{F}$, by considering in turn each node in $\cup_{(u,v) \in Z_{j-1}} \mathcal{N}((u, v))$ with positive value in $\hat{F}$; for each such node, if all its constraints are not covered, we add it to the stable set and cover all its constraints. The resulting stable set S_j is maximal since adding any node with positive value in $\hat{F}$ and not in $\cup_{(u,v) \in Z_{j-1}} \mathcal{N}((u, v))$ to $S_{j-1} \setminus Z_{j-1}$ would result in a set that is not stable (otherwise S_{j-1} is not maximal).

Computing the maximal stable set at iteration $j \geq 2$ takes time $O(|Z_{j-1}|N \log^2 N)$, since the algorithm checks, for each node (u, v) in Z_{j-1}, for each node (k, l) in the neighborhood of (u, v), whether the $\log_2 N + 1$ constraints of (k, l) are covered. Since for each node (u, v), there is at most a single $j \geq 1$ such that $(u, v) \in Z_j$, we have $\sum_{j \geq 1} |Z_j| \leq N^2$. Because the maximum size of a stable set of G is N, the other parts of step 2 can be computed in time $O(N)$ per iteration. Thus, the total run time of Algorithm 6 is $O(N^3 \log^2 N)$.

4.8 Chapter Summary

In this chapter, we have considered under what conditions there exist packet-scheduling algorithms that maintain bounded backlog for arbitrary time varying fluid policies for an $N \times N$ Banyan network. First, we showed that in contrast to the crossbar switch, Banyan networks require speedup in order to maintain bounded backlog for arbitrary fluid policies. Next, we developed the concept of required speedup, and computed the exact speedup required for the 4×4 Banyan network, and obtained logarithmic bounds on the speedup required for a general $N \times N$ Banyan network. Computing the exact speedup required for general Banyan networks, and other networks of interest, remains an interesting and stimulating open problem.

In the next chapter, we bound backlog and speedup for general, input-queued switch networks, which may not be layered, may have links with non-unit capacities, and may have multiple paths connecting a source to a destination. The techniques and algorithms used to derive bounds for general networks are based on those introduced in this and the previous chapter.

Chapter 5

General Input-Queued Switch Networks

We define and analyze backlog for general, input-queued switch networks, which we simply call general networks. In contrast to Banyan networks, we allow general networks to be non-layered, have links with positive-integer capacities, and have multiple paths between each source-sink pair. We prove an upper bound on the speedup required to maintain bounded backlog on a general network for any fluid policy; this bound is derived by analyzing integral and fractional static flows through the general network. In Section 5.3, we apply this upper bound to show that for any $\epsilon > 0$, speedup ϵ plus the network dilation (defined below) suffices to maintain bounded backlog for any fluid policy.

General networks are defined in terms of a directed graph, in which nodes represent crossbar switches and arcs (directed edges) correspond to links connecting crossbar switches. Queueing is only allowed at data sources and sinks. Recall from Chapter 4 that one time step is the time required for a single crossbar switch to transmit a packet across its switch fabric and through an outgoing link. Using graph terminology, a time step is the time required to leave a node and traverse an outgoing arc from that node. We say a set of packets *is sent* at a time step to mean each packet in the set leaves its source node and traverses the outgoing arc from its source node during that time step.

We model fluid and packetized policies for general networks by flows over time, which we define below. We first explain why it is necessary to consider flows over time rather than static flows when analyzing non-layered networks. For layered networks, such as Banyan networks, packets sent at different time steps (that is, corresponding to different packetized steps) never use the same link at the same time. For example, as packets of packetized step $P^{(t+1)}$ traverse links that enter the first stage of a Banyan network, the packets of $P^{(t)}$, sent one time step earlier, traverse links that enter stage 2. In a non-layered network, such as the one depicted in Figure 5-1, packets sent at different time steps could traverse the same arc (representing a link) at the same time step. For example, a packet sent at time t from source a_1 to sink b_1 and a packet sent at time $t-1$ from source a_3 to sink b_3 would both traverse the same arc (the only horizontal arc in the figure) during time step $t+1$. To ensure that the number of packets traversing each arc by a packetized policy in one time step is at most the arc's capacity, each packetized step $P^{(t)}$ for a general network will encode how many packets sent at time t traverse each arc at each time step. Each fluid step $F^{(t)}$ will encode the sum of fractional packets sent at time t that traverse each arc at each time step. This information is conveniently encoded using *flows over time*, described in the next section.

5.1 Model and Definitions

First, we define a general network. Next, we define a packetized policy as a sequence of integral, multicommodity flows over time (also defined below) with certain properties on a general network. We define fluid policies similarly, except we omit the integrality constraint. The cumulative difference and backlog are then defined.

We use notation and definitions for networks and flows from Fleischer and Skutella [22], some of which we reproduce below.

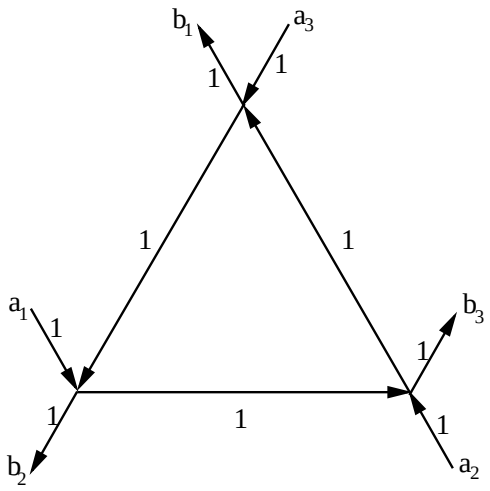

Figure 5-1: A general network with three communicating source-sink pairs $(a_1, b_1), (a_2, b_2), (a_3, b_3)$ where each arc has unit capacity. Here, the dilation d of the general network is 4.

5.1.1 Definition of General Network

A *general network* $\mathcal{N}(V, A)$ has a set of nodes V and arcs (directed edges) A. Each arc $e \in A$ is assigned a positive integer capacity $U(e)$ that represents the maximum number of packets that can traverse that arc in one time step. We also consider a set of terminals $S \subseteq V$ that can be partitioned into sources S^+ and sinks S^-. We assume that each source $v \in S^+$ has a single outgoing arc $e(v)$ and no incoming arcs. Similarly, we assume that each sink $v \in S^-$ has a single incoming arc $e(v)$ and no outgoing arcs. We are given a set of communicating source-sink pairs $\{(a_1, b_1), \ldots, (a_m, b_m)\} \subseteq S^+ \times S^-$, which we call $\mathcal{T}$. We let the *dilation* d of a general network be the maximum number of arcs in any simple path (that is, a path with no cycles) between a communicating source-sink pair. An example of a general network is given in Figure 5-1.

5.1.2 Definitions for Static Flows

We start by defining single-commodity, static flows. A *static flow* x through general network $\mathcal{N}(V, A)$, with single source-sink pair (a, b), assigns every arc $e \in A$ a flow

value x_e such that for each node $v \in V \setminus \{a, b\}$, the *flow conservation constraints*

$$\sum_{e \in \delta^-(v)} x_e = \sum_{e \in \delta^+(v)} x_e,$$

are obeyed, where $\delta^-(v)$ denotes the set of incoming arcs to v and $\delta^+(v)$ the set of outgoing arcs from v. A *maximum flow* from a to b is a nonnegative, feasible flow whose value over arc $e(b)$ (equivalently $e(a)$) is the maximum such value of any nonnegative, feasible flow from a to b. A single-commodity, static flow is *cycle-free* if in each cycle of arcs in the general network, there is some arc with flow value 0.

In the multicommodity setting, we consider a set of commodities $M := \{1, \ldots, m\}$, each of which is defined by a single source-sink pair $(a_i, b_i) \in \mathcal{T}$. A *static multicommodity flow* x through $\mathcal{N}(V, A)$ assigns every arc-commodity pair (e, i) a flow value $x_{e,i}$ such that $x_i := (x_{e,i})_{e \in A}$ is a single-commodity flow as defined above for each $i \in M$. We call x *feasible* if it obeys the *capacity constraints* $\sum_{i \in M} x_{e,i} \leq U(e)$, for all $e \in A$. A *static multicommodity circulation* y is a static multicommodity flow such that for each commodity $i \in M$, no flow leaves any source or arrives at any sink (that is, $\forall i \in M, y_{e(a_i),i} = y_{e(b_i),i} = 0$). A static, multicommodity flow x is *cycle-free* if for each commodity $i \in M$, x_i is a cycle-free, single-commodity flow. We represent static (multicommodity) flows by lower case letters, and (multicommodity) flows over time, defined next, by upper case letters.

5.1.3 Definitions for Multicommodity Flows Over Time

We consider multicommodity flows over time through a general network $\mathcal{N}(V, A)$, with no storage at intermediate nodes. Again, we consider a set of commodities $M := \{1, \ldots, m\}$, each of which is defined by a single source-sink pair $(a_i, b_i) \in \mathcal{T}$. Time is considered discrete, and one time step is set to be the transit time in traversing an arc (which is assumed to be the same for all arcs). A *multicommodity flow over time* F through a general network, for each commodity $i \in M$, arc $e \in A$, and time step $k > 0$, assigns a flow value $F_{e,i}(k)$ such that the following flow conservation

constraints

$$\sum_{e \in \delta^-(v)} F_{e,i}(k) = \sum_{e \in \delta^+(v)} F_{e,i}(k+1),$$

are obeyed for each commodity $i \in M$, each node $v \in V \setminus S$, and each time step $k > 0$. We say a multicommodity flow over time F has *time horizon* $[t_1, t_2]$ if for each $i \in M$, each arc $e \in A$, and each $k \notin [t_1, t_2]$, we have $F_{e,i}(k) = 0$. Sometimes for convenience of notation we refer to $F_{e,i}(k)$ for $k \leq 0$, implicitly assuming $F_{e,i}(k) = 0$ in this case.

A multicommodity flow over time F is said to be *feasible* if for each arc $e \in A$, and each time step $t > 0$, we have

$$\sum_{i \in M} F_{e,i}(t) \leq U(e).$$

The sum of two multicommodity flows over time F and F' is the multicommodity flow over time F'' such that for each commodity $i \in M$, arc $e \in A$, and time step k, we have $F''_{e,i}(k) = F_{e,i}(k) + F'_{e,i}(k)$. The difference between two multicommodity flows is defined similarly. Two multicommodity flows over time F and F' are called *time-shifted versions of each other* if for some integer t, $\forall i \in M, \forall e \in A, \forall k$, we have $F_{e,i}(k) = F'_{e,i}(k-t)$.

A static multicommodity flow or multicommodity flow over time is called nonnegative (integral) if the flow of each commodity over each arc at each time step is nonnegative (an integer).

5.1.4 Definition of Packetized Policy and Fluid Policy

A *packetized policy* for a general network $\mathcal{N}(V, A)$ is a sequence of packet transmissions such that for each arc in the network, the number of packets traversing that arc at any time step is at most that arc's capacity. A packetized policy is represented by a sequence of nonnegative, integral, multicommodity flows over time $\{P^{(t)}\}_{t>0}$ (through the general network) with the following three properties:

- For all $t > 0$, in packetized step $P^{(t)}$, packets can leave source nodes only at

time step t. That is, for all $t > 0$, for each source-sink pair $(a_i, b_i) \in \mathcal{T}$, and $\forall k \neq t$ we have $P^{(t)}_{e(a_i),i}(k) = 0$.

- For all $t > 0$, $P^{(t)}$ has time horizon $[t, t + d - 1]$. This means that the multicommodity flow over time $P^{(t)}$ only uses paths of length at most d, the dilation of the general network.

- For all $t > 0$, $G^{(t)} := \sum_{h=1}^{t} P^{(h)}$ is a feasible, multicommodity flow over time.

A *fluid policy* for a general network $\mathcal{N}(V, A)$ is a sequence of fractional packet transmissions such that for each arc in the network, the sum of fractional packets traversing that arc at each time step is at most that arc's capacity. A fluid policy is represented by a sequence of nonnegative, multicommodity flows over time $\{F^{(t)}\}_{t>0}$ through the network with the three properties above (with P replaced by F).

A packet-scheduling algorithm is said to use *speedup* $s \geq 1$ if we require that any fluid policy satisfy the following additional constraint:
For all $t > 0$, $G^{(t)} := \sum_{h=1}^{t} F^{(h)}$ is a feasible, multicommodity flow over time for the general network with reduced capacities $U'(e) := U(e)/s$.

5.1.5 Definition of Cumulative Difference and Backlog

Consider the feasible, multicommodity flow over time $G^{(t)} := \sum_{h=1}^{t} P^{(h)}$. For a communicating source-sink pair $(a_i, b_i) \in \mathcal{T}$ and arc $e \in A$, the number of commodity i packets that have traversed e, cumulatively through time step $k > 0$ in the flow over time $G^{(t)}$, is $\sum_{j=1}^{k} G^{(t)}_{e,i}(j)$. The *cumulative difference* quantifies how much this number of packets differs from the corresponding sum of fluid. More precisely, the cumulative difference for $i \in M$ over arc $e \in A$ through time step $k > 0$ due to $\{F^{(h)}\}_{h \leq t}$ and $\{P^{(h)}\}_{h \leq t}$ is defined as

$$C^{(t)}_{e,i}(k) := \sum_{j=1}^{k} \sum_{h=1}^{t} (F^{(h)}_{e,i}(j) - P^{(h)}_{e,i}(j)).$$

We require each packet-scheduling algorithm, at each time t, given the first t steps of a fluid policy $\{F^{(h)}\}_{h \leq t}$, to output a packetized step $P^{(t)}$ such that for each commodity

$i \in M$, for each arc $e \in A$, for each time step $k > 0$, we have $C_{e,i}^{(t)}(k) > -1$. This constraint ensures that for each arc, for each time step, for each source-sink pair, the cumulative number of packets scheduled by the packetized policy can only exceed the cumulative fluid scheduled by the fluid policy by less than 1 packet.

The *backlog* for a source-sink pair $(a_i, b_i) \in \mathcal{T}$ at time t is the real number $B(i,t) := (C_{e(b_i),i}^{(t)}(t+(d-1)))^+$; this quantifies how much the packetized policy is lagging behind the fluid policy in terms of the cumulative amount of data delivered from source a_i to sink b_i due to $\{P^{(h)}\}_{h \le t}$ and $\{F^{(h)}\}_{h \le t}$ by the end of their time horizon. The backlog for a source $v \in S^+$ is the sum of backlog over all $(a_i, b_i) \in \mathcal{T}$ with $a_i = v$. Similarly the backlog for a sink $v \in S^-$ is the sum of backlog over all $(a_i, b_i) \in \mathcal{T}$ with $b_i = v$.

5.2 Upper Bound on Required Speedup for General Networks

We can represent each static multicommodity flow x through a general network by the set of values $\{x_{e,i}\}_{e \in A, i \in M}$. Considering this set of values as a vector in $\mathbf{R}^{|A||\mathcal{T}|}$, let $\mathcal{F} \subseteq \mathbf{R}^{|A||\mathcal{T}|}$ denote the set of nonnegative, feasible, static multicommodity flows[1]. Let $\mathcal{F}'$ denote the set of flows in $\mathcal{F}$ that are cycle-free. Let $\mathcal{P} \subseteq \mathbf{R}^{|A||\mathcal{T}|}$ denote the convex hull of *integral*, nonnegative, feasible, static multicommodity flows. Note that both $\mathcal{P}$ and $\mathcal{F}$ are polytopes, and $\mathcal{P} \subseteq \mathcal{F}$. We denote the dimension of $\mathcal{P}$ by $\dim(\mathcal{P})$, which is at most $|A||\mathcal{T}|$.

Theorem 10 *For $s \ge 1$ such that $\frac{1}{s}\mathcal{F}' \subseteq \mathcal{P}$, for any $\epsilon > 0$, there is an algorithm using speedup $s + \epsilon$ that maintains bounded backlog for each source and sink, for any fluid policy, on a general network.*

Before proving the above theorem, we contrast it with Theorem 7 for Banyan networks, which gives tight bounds on the required speedup to maintain bounded backlog in terms of the polytopes $\mathcal{F}$ and $\mathcal{P}$ defined for Banyan networks. The upper bound in Theorem 10 for general networks is not tight, in general. That is, a smaller

[1]We do not specify particular demands for each communicating source-sink pair.

speedup than $\min\{s \geq 1 : \frac{1}{s}\mathcal{F}' \subseteq \mathcal{P}\}$ may suffice to maintain bounded backlog for arbitrary fluid policies. For example, consider the general network in Figure 5-1 with three source-sink pairs $(a_1, b_1), (a_2, b_2), (a_3, b_3)$, and all arcs with unit capacity. We have $\min\{s \geq 1 : \frac{1}{s}\mathcal{F}' \subseteq \mathcal{P}\} \geq 3/2$ for the polytopes corresponding to this general network. This follows since, first, the static flow F that sends $1/2$ unit between each source-sink pair is feasible and cycle-free, so is in $\mathcal{F}'$. The sum of flow values over arcs leaving each source in any $P \in \mathcal{P}$ is at most 1, however, since any *integral*, feasible static flow through the general network sends at most a single unit. Thus, for $s < 3/2$, we have $(1/s)F \notin \mathcal{P}$, and so the upper bound given by Theorem 10 is at least $3/2$. We prove in Appendix B, however, that for any $\epsilon > 0$, speedup $1 + \epsilon$ suffices to maintain bounded backlog for any fluid policy on this general network. The main idea in the proof is that a packetized policy for a general network, which is a sequence of flows over time, could send three packets–one between each of the three source-sink pairs–at each odd time step during some interval of time, and send no packets at each even time step during this interval of time; here, no arc is ever traversed by more than one packet in a single time step, and $3/2$ total packets, on average, are transmitted per time step during the interval.[2]

Proof of Theorem 10: Given $s \geq 1$ such that $\frac{1}{s}\mathcal{F}' \subseteq \mathcal{P}$ and $\epsilon > 0$, we present a packet-scheduling algorithm for a general network that maintains bounded backlog for any fluid policy. The algorithm processes batches of consecutive fluid steps and produces corresponding batches of packetized steps, using pipelining, just as in Algorithm 2 of Chapter 3.

The number of fluid steps input in any batch (and also the number of packetized steps output in any batch) is set to be the positive integer $\alpha := \lceil 2s(d-1)/\epsilon \rceil + d - 1$, for $d > 1$ the dilation of the general network[3]. We use pipelining so that each batch of packetized steps will be scheduled 2α time steps after the corresponding batch of fluid steps. It is also possible to schedule each batch of packetized steps only α time

[2]In the proof in Appendix B, the algorithm sometimes sends other combinations of packets than the pattern described here.

[3]We assume $d > 1$ since backlog can be kept bounded with no speedup for any general network with dilation $d = 1$.

steps after the corresponding batch of fluid steps, so that the first step of a batch of packetized steps is scheduled just after the last step of the corresponding batch of fluid steps; however, waiting 2α time steps after the beginning of a batch of fluid steps, as we do here, allows amortization of the run time of the algorithm over α time steps.

For each batch of α consecutive packetized steps output by the algorithm, the last $d-1$ packetized steps of the batch are each the all zero flow over time. Thus, for any $\beta \geq 1$, by just after time step $\beta\alpha$, all packets sent by the packetized policy at or before this time step have reached their sinks, so cannot interfere with packets scheduled in the next batch. Instead of using speedup s, a slightly greater speedup of $s+\epsilon$ is needed to offset the $d-1$ packetized steps at the end of each batch in which no packets are sent.

Before describing the computation that outputs each batch of packetized steps, we define the *static* multicommodity flows $\{f^{(\beta)}\}_{\beta\geq 0}$, $\{p^{(\beta)}\}_{\beta\geq 0}$ on the same general network $\mathcal{N}(V,A)$ on which the packetized and fluid policies are defined. These static flows are used by the algorithm to keep track of backlog at the end of each batch. For $\beta \geq 1$, $e \in A$, and $i \in M$ let

$$f_{e,i}^{(\beta)} := \sum_{k>0} \sum_{h=(\beta-1)\alpha+1}^{\beta\alpha} F_{e,i}^{(h)}(k), \tag{5.1}$$

$$p_{e,i}^{(\beta)} := \sum_{k>0} \sum_{h=(\beta-1)\alpha+1}^{\beta\alpha} P_{e,i}^{(h)}(k). \tag{5.2}$$

For convenience of notation, let $f^{(0)} = p^{(0)}$ denote the all zero static flow.

Using these static flows, the backlog for a source-sink pair $(a_i, b_i) \in \mathcal{T}$ at the end of batch β can be conveniently represented as

$$B(i,\alpha\beta) \;=\; \left(\sum_{j=0}^{\beta} f_{e(b_i),i}^{(j)} - p_{e(b_i),i}^{(j)}\right)^{+}. \tag{5.3}$$

The following invariant will be shown to hold for each batch of fluid steps processed:

Invariant 6 *For any $\beta \geq 0$, we have for some $k \leq \dim(\mathcal{P}) + 1$, k nonnegative, integral, feasible, static multicommodity flows denoted by $q^{(1)}, \ldots, q^{(k)}$, nonnegative coefficients $\lambda_1, \ldots, \lambda_k$, and a nonnegative, static multicommodity circulation y such that*

$$\left(\sum_{j=0}^{\beta}(f^{(j)} - p^{(j)})\right) - p^{(\beta+1)} - p^{(\beta+2)} \;=\; \sum_{l=1}^{k} \lambda_l q^{(l)} + y,$$

and

$$\sum_{l=1}^{k} \lambda_l < (\alpha - (d-1))(\dim(\mathcal{P}) + 1).$$

The invariant above[4] and (5.3) imply for any $\beta \geq 0$, for time step $\alpha\beta$ we have the following bound on backlog for source-sink pair $(a_i, b_i) \in \mathcal{T}$:

$$B(i, \alpha\beta) < [2\alpha + (\alpha - (d-1))(\dim(\mathcal{P}) + 1)]\, U(e(b_i)).$$

For any source-sink pair $(a_i, b_i) \in \mathcal{T}$ and any time $t > 0$, the total amount of fluid that can be scheduled between the beginning of the batch containing t and time step t is at most $\frac{\alpha+(d-1)}{s+\epsilon} U(e(b_i))$. This implies the backlog for any source-sink pair $(a_i, b_i) \in \mathcal{T}$ at any time $t > 0$ is at most $\frac{\alpha+(d-1)}{s+\epsilon} U(e(b_i))$ more than $\max_{\beta \geq 0} B(i, \alpha\beta)$. Thus, the invariant above implies backlog is bounded for all source-sink pairs for all time steps.

Batch Computation

We now describe the pipelined process of computing a batch of α packetized steps given a batch of α fluid steps, showing inductively that Invariant 6 is maintained for each batch. Initially, for $1 \leq t \leq 2\alpha$, the algorithm sets $P^{(t)}$ to be the all zero flow over time; thus, the invariant above holds for $\beta = 0$ with $k = 1$, $\lambda_1 = 0$, $q^{(1)}$ the all zero static multicommodity flow and y the all zero static multicommodity circulation. The batch count β is initially set to 1. We first give an outline of the four

[4]Note that for any static multicommodity circulation, such as y as described in the invariant above, for any source-sink pair $(a_i, b_i) \in \mathcal{T}$ we have $y_{e(b_i),i} = 0$.

steps involved in producing a batch of packetized steps given a batch of fluid steps.

Outline of Batch Computation

For $\beta \geq 1$, given new batch of consecutive fluid steps $F^{((\beta-1)\alpha+1)}, \ldots, F^{(\beta\alpha)}$, the algorithm constructs the batch of packetized steps $P^{((\beta+1)\alpha+1)}, \ldots, P^{((\beta+2)\alpha)}$ as follows:

1. First, the given batch of fluid steps is used to compute the static multicommodity flow $f^{(\beta)}$, as defined above. The properties of a fluid policy specified in Section 5.1.4 are used to show that $\frac{s+\epsilon}{\alpha+d-1} f^{(\beta)}$ is a nonnegative, feasible, static multicommodity flow. We find a nonnegative, static, multicommodity circulation y' such that $\frac{s+\epsilon}{\alpha+d-1} f^{(\beta)} - y'$ is a cycle-free, nonnegative, feasible, static multicommodity flow, and so is in $\mathcal{F}'$.

2. Using the assumption that $(1/s)\mathcal{F}' \subseteq \mathcal{P}$, we have that $\frac{s+\epsilon}{s(\alpha+d-1)} f^{(\beta)} - y'/s \in \mathcal{P}$. By Caratheodory's theorem, and the definition of $\mathcal{P}$, we can write $\frac{s+\epsilon}{s(\alpha+d-1)} f^{(\beta)} - y'/s$ as a convex combination of nonnegative, integral, feasible, static multicommodity flows.

3. We multiply each coefficient of the convex combination from the previous step by $\frac{s(\alpha+d-1)}{s+\epsilon}$, which by our choice of α is at most $\alpha - (d-1)$. We add the resulting linear combination to the similar linear combination $\sum_{l=1}^{k} \lambda_l q^{(l)}$ from Invariant 6 for the previous batch $\beta - 1$. We show this sum can be expressed, by Caratheodory's theorem[5], as a linear combination with at most $\dim(\mathcal{P}) + 1$ terms and with sum of coefficients less than $(\alpha - (d-1))(\dim(\mathcal{P}) + 2)$.

4. If there is a coefficient in the linear combination from the previous step with value at least $\alpha - (d-1)$, we let q denote the corresponding element (that is, nonnegative, integral, feasible, static multicommodity flow) of the linear combination.[6] We convert the static flow q into a flow over time, and set all but the last $d-1$ packetized steps output to be time-shifted versions of this flow

[5]This is the same technique used in Algorithm 5 of Chapter 4 for Banyan networks.

[6]If each coefficient in the linear combination from the previous step has value $< \alpha - (d-1)$, all packetized steps output in this batch are set to the all zero flow over time.

over time. The remaining packetized steps output are set to the all zero flow over time. This is shown to maintain Invariant 6 for the current batch β.

Full Description of Batch Computation

For $\beta \geq 1$, given new batch of consecutive fluid steps $F^{((\beta-1)\alpha+1)}, \ldots, F^{(\beta\alpha)}$, the algorithm constructs the batch of packetized steps $P^{((\beta+1)\alpha+1)}, \ldots, P^{((\beta+2)\alpha)}$ as follows:

1. Consider the nonnegative, multicommodity flow over time $H^{(\beta)} := \sum_{h=(\beta-1)\alpha+1}^{\beta\alpha} F^{(h)}$, which has time horizon $[(\beta-1)\alpha+1, \beta\alpha+d-1]$. Since the algorithm uses speedup $s+\epsilon$, we have $H^{(\beta)}$ is a feasible multicommodity flow over time for the general network with reduced capacities $U'(e) = U(e)/(s+\epsilon)$. Recall that we defined the *static* multicommodity flow $f^{(\beta)}$ earlier such that for each $i \in M$ and each arc $e \in A$,

$$f_{e,i}^{(\beta)} = \sum_{k>0} H_{e,i}^{(\beta)}(k).$$

Considering the time horizon of $H^{(\beta)}$, we have for each arc $e \in A$,

$$\sum_{i \in M} f_{e,i}^{(\beta)} \leq \frac{\alpha+d-1}{s+\epsilon} U(e).$$

Thus, $\frac{s+\epsilon}{\alpha+d-1} f^{(\beta)}$ is a nonnegative, feasible, static multicommodity flow, and so is by definition in $\mathcal{F}$. We find[7] a nonnegative, static, multicommodity circulation y' such that $\frac{s+\epsilon}{\alpha+d-1} f^{(\beta)} - y'$ is a cycle-free, nonnegative, feasible, static multicommodity flow, and so is in $\mathcal{F}'$.

2. By Caratheodory's theorem, the static multicommodity flow $\frac{s+\epsilon}{s(\alpha+d-1)} f^{(\beta)} - y'/s \in (1/s)\mathcal{F}' \subseteq \mathcal{P}$ can be expressed as a convex combination of $k' \leq \dim(\mathcal{P}) + 1$ nonnegative, integral, feasible, static multicommodity flows

$$\frac{s+\epsilon}{s(\alpha+d-1)} f^{(\beta)} - y'/s = \sum_{j=1}^{k'} \gamma_j r^{(j)}.$$

[7]Such a circulation can be found in polynomial time; see for example [15].

Letting $\gamma'_j := \frac{s(\alpha+d-1)}{s+\epsilon}\gamma_j$ for all j, and letting $y'' := \frac{\alpha+d-1}{s+\epsilon}y'$, we have

$$f^{(\beta)} = \sum_{j=1}^{k'} \gamma'_j r^{(j)} + y'',$$

and by the definition of α above,

$$\sum_{j=1}^{k'} \gamma'_j = \frac{s(\alpha+d-1)}{s+\epsilon} \leq \alpha - (d-1).$$

3. Assuming Invariant 6 holds for $\beta - 1$, we have for $k, \{\lambda_l\}, \{q^{(l)}\}, y$ as specified in the invariant,

$$\begin{aligned}
\sum_{j=0}^{\beta}(f^{(j)} - p^{(j)}) - p^{(\beta+1)} &= \sum_{j=0}^{\beta-1}(f^{(j)} - p^{(j)}) + f^{(\beta)} - p^{(\beta)} - p^{(\beta+1)} \\
&= \sum_{l=1}^{k} \lambda_l q^{(l)} + f^{(\beta)} + y \\
&= \sum_{l=1}^{k} \lambda_l q^{(l)} + \sum_{j=1}^{k'} \gamma'_j r^{(j)} + y'' + y. \qquad (5.4)
\end{aligned}$$

The sum of the λ_l's and γ'_j's is less than $(\alpha - (d-1))(\dim(\mathcal{P}) + 2)$. Let

$$T := \sum_{l=1}^{k} \lambda_l q^{(l)} + \sum_{j=1}^{k'} \gamma'_j r^{(j)}.$$

Caratheodory's theorem now tells us that T can also be expressed as a nonnegative, weighted sum of at most $\dim(\mathcal{P}) + 1$ elements from the set $\{q^{(l)}\} \cup \{r^{(j)}\}$, with the weights summing to less than $(\alpha - (d-1))(\dim(\mathcal{P}) + 2)$. Reset $\{\lambda_l\}$, $\{q^{(l)}\}$, and k so that $\sum_{l=1}^{k} \lambda_l q^{(l)}$ is such a nonnegative, weighted sum. By (5.4), with the reset values of $\{\lambda_l\}$, $\{q^{(l)}\}$, and k, we have

$$\sum_{j=0}^{\beta}(f^{(j)} - p^{(j)}) - p^{(\beta+1)} = \sum_{l=1}^{k} \lambda_l q^{(l)} + y'' + y. \qquad (5.5)$$

4. We now select packetized steps $P^{((\beta+1)\alpha+1)}, \ldots, P^{((\beta+2)\alpha)}$ such that

$P^{((\beta+1)\alpha+1)}, \ldots, P^{((\beta+2)\alpha-(d-1))}$ are time shifted versions of each other, and such that $P^{((\beta+2)\alpha-(d-1)+1)}, \ldots, P^{((\beta+2)\alpha)}$ are the all zero flow over time. If each λ_l is less than $\alpha - (d-1)$, then setting $P^{(t)}$ to the all zero flow over time for $t \in [(b+1)\alpha + 1, (b+2)\alpha]$, the invariant is maintained at the end of batch β and we are done.

Otherwise, for some $l' \leq k$ we have $\lambda_{l'} \geq \alpha - (d-1)$. In this case, we convert the nonnegative, integral, feasible, static flow $q^{(l')}$ into a flow over time X with time horizon $[1, d]$ as follows: First, set X to be the all zero multicommodity flow over time, and set y''' to be the all zero *static* multicommodity flow. Next, for each commodity $i \in M$, the single-commodity static flow $q_i^{(l')} := (q_{e,i}^{(l')})_{e \in A}$ is decomposed into a set of simple, unit-capacity paths $\{\pi_1, \ldots, \pi_n\}$ from a_i to b_i, and a set of nonnegative cycles $c_1, \ldots, c_u$. We incorporate the cycles into y''' by setting $y''' \leftarrow y''' + (\alpha + d - 1) \sum_{u'=1}^{u} c_{u'}$. Each path $\pi_j = (e_1, \ldots, e_r) \in A^r$ is added to the flow over time X in the natural way, where for each $r' \in \{1, \ldots, r\}$ we set $X_{e_{r'},i}(r') \leftarrow X_{e_{r'},i}(r') + 1$. Lastly, for $t \in [(\beta+1)\alpha+1, (\beta+2)\alpha-(d-1)]$, we set $P_{e,i}^{(t)}(k) := X_{e,i}(k-(t-1))$, the flow over time X shifted forward in time by $t-1$ time steps. Set $P^{((\beta+2)\alpha-(d-1)+1)}, \ldots, P^{((\beta+2)\alpha)}$ to be the all zero flow over time. Since for any $k > 0$ and any $e \in A$,

$$\sum_{i \in M} \sum_{h=(\beta+1)\alpha+1}^{(\beta+2)\alpha} P_{e,i}^{(h)}(k) \leq \sum_{i \in M} q_{e,i}^{(l')} \leq U(e),$$

we have that $\sum_{h=(\beta+1)\alpha+1}^{(\beta+2)\alpha} P^{(h)}$ is a feasible multicommodity flow over time. Since each batch of packetized steps ends with $d-1$ steps in which no packets are sent, and assuming (by inductive hypothesis) $\sum_{h=1}^{(\beta+1)\alpha} P^{(h)}$ is a feasible multicommodity flow over time, we have that for $t \leq (\beta+2)\alpha$ the third property specified in Section 5.1.4 holds for the packetized policy so far constructed. The other two properties hold as well, by our construction.

By our selection of packetized steps for this batch, we have $\forall i \in M$, $\forall e \in A$,

$$p_{e,i}^{(\beta+2)} := \sum_{k>0} \sum_{h=(\beta+1)\alpha+1}^{(\beta+2)\alpha} P_{e,i}^{(h)}(k) = (\alpha - (d-1))q_{e,i}^{(l')} - y_{e,i}'''.$$

Thus, by (5.5) we have

$$\left(\sum_{j=0}^{\beta}(f^{(j)} - p^{(j)})\right) - p^{(\beta+1)} - p^{(\beta+2)} = \left(\sum_{l=1}^{k} \lambda_l q^{(l)}\right) + y'' + y - (\alpha-(d-1))q^{(l')} + y'''.$$

Setting $\lambda_{l'} \leftarrow \lambda_{l'} - (\alpha - (d-1))$, the sum of λ_l's is now less than $(\alpha-(d-1))(\dim(\mathcal{P})+1)$; setting $y \leftarrow y + y'' + y'''$, we maintain Invariant 6 for batch β. □

5.3 Upper Bound on Required Speedup in terms of Network Dilation

We prove the following theorem.

Theorem 11 *For d the dilation of a general network, for $\mathcal{F}'$ the set of cycle-free elements of $\mathcal{F}$, we have $(1/d)\mathcal{F}' \subseteq \mathcal{P}$.*

This theorem, in conjunction with Theorem 10 above, implies that for any $\epsilon > 0$, there is an algorithm using speedup $d + \epsilon$ that maintains bounded backlog for each source and each sink, for any fluid policy, on a general network.

Proof: We present an algorithm that given any $f \in \mathcal{F}'$, outputs a decomposition

$$f = \sum_{j=1}^{j'} \gamma_j r^{(j)}, \tag{5.6}$$

where for each $j \leq j'$, $\gamma_j \geq 0$ and $r^{(j)}$ is an integral, feasible, nonnegative, static multicommodity flow, and $\sum_{j=1}^{j'} \gamma_j \leq d$.

This algorithm is an extension of Algorithm 6 from Chapter 4 for similarly decomposing a fluid step for a Banyan network. The main additional difficulties for general

networks are that there may be multiple paths between each source-sink pair, and arcs may have non-unit capacities. The algorithm is given next, followed by a proof that it produces a decomposition as in (5.6).

Algorithm 7

The input is a cycle-free, nonnegative, feasible, static multicommodity flow $f \in \mathcal{F}'$.

1. Set $j \leftarrow 1$. Set $\hat{f} \leftarrow f$.
2. While $\hat{f} \neq \mathbf{0}$ (where $\mathbf{0}$ is the all zero, static multicommodity flow):
3. Set $\hat{U} \leftarrow U$; that is, store the arc capacities in $\hat{U}$.
4. For $i = 1$ to $|M|$:
5. Find an integral, cycle-free, maximum flow g between source-sink pair (a_i, b_i) through the general network with the following capacities: For each arc $e \in A$, its capacity is $\hat{U}(e)$ if $\hat{f}_{e,i} > 0$, and is 0 otherwise. Set $r_i^{(j)}$ to be the flow g.
6. For each arc $e \in A$, set $\hat{U}(e) \leftarrow \hat{U}(e) - r_{e,i}^{(j)}$.
7. End for loop.
8. Set γ_j to be the largest value such that $\hat{f} - \gamma_j r^{(j)}$ is nonnegative.
9. Set $\hat{f} \leftarrow \hat{f} - \gamma_j r^{(j)}$ and then increment j by 1.
10. End while loop.

The algorithm above consists of an inner loop (lines $4-7$) nested inside an outer loop (lines $2-10$). In each iteration j of the outer loop, the flow $\hat{f}$, which stores the difference between f and the partial decomposition of it $\sum_{k=1}^{j-1} \gamma_k r^{(k)}$ computed so far, is passed to the inner loop. The purpose of the inner loop is to compute an integral, feasible, static multicommodity flow $r^{(j)}$ such that for each commodity $i \in M$, we have

- For each $e \in A$, $r_{e,i}^{(j)} > 0$ only if $\hat{f}_{e,i} > 0$, and

- $r_i^{(j)}$ is maximal in the sense that no more flow can be sent from a_i to b_i without violating the capacity constraints or decreasing the flow between a different source-sink pair.

Since f is by definition cycle-free, we have at each iteration of the outer loop that $\hat{f}$ is cycle-free. Thus, the outer loop iterates while there exists an $i \in M$ such that $\hat{f}_{e(b_i),i} > 0$. This implies that at the end of each iteration j of the outer loop, the flow of some commodity over some arc in $\hat{f}$ is reduced to 0 at line 9. The algorithm, therefore, terminates after at most $|A||M|$ iterations of the outer loop. Denote the last iteration of the algorithm by j'. Since $\hat{f} = \mathbf{0}$ just after iteration j', we have

$$f = \sum_{j=1}^{j'} \gamma_j r^{(j)}.$$

It remains to show $\sum_{j=1}^{j'} \gamma_j$ is at most d.

Just before the last iteration j' of the outer loop, there exists a source-sink pair $(a_i, b_i) \in \mathcal{T}$ and a simple path $(e_1, \ldots, e_{k'})$ from a_i to b_i, such that $\hat{f}_{e_k,i} > 0$ for each $k \in \{1, \ldots, k'\}$. We show that $\sum_{j=1}^{j'} \gamma_j \leq k'$.

For any iteration of the algorithm $j \leq j'$, we have for some $k \leq k'$ that

$$\sum_{i' \in M} r_{e_k,i'}^{(j)} = U(e_k). \tag{5.7}$$

The previous statement follows since otherwise, an additional unit of flow would have been scheduled between a_i and b_i along path $(e_1, \ldots, e_{k'})$ at line 5 at iteration j of the outer loop. For each $j \leq j'$, let $q(j)$ denote the smallest index k such that (5.7) holds. Then at line 9 at iteration j of the outer loop, the total flow from all commodities in $\hat{f}$ over arc $e_{q(j)}$ is reduced by $\gamma_j U(e_{q(j)})$. For each $k \leq k'$, since the total flow in $\hat{f}$ along arc e_k remains nonnegative and was at most $U(e_k)$ to start with, we have

$\sum_{j:q(j)=k} \gamma_j \leq 1$. Therefore,

$$\sum_{j=1}^{j'} \gamma_j = \sum_{k=1}^{k'} \sum_{j:q(j)=k} \gamma_j \leq k'.$$

Since k' is at most the network dilation d, and $f = \sum_{j=1}^{j'} \gamma_j r^{(j)}$, we have $(1/d)f \in \mathcal{P}$, which proves Theorem 11. $\square$

Appendix A

A.1 Proofs of Claims from Section 2.6.1

In Section 2.6.1, we define for fixed N, fixed $s \geq 1$, the function

$$g(m) := \frac{m}{s}(H_N - H_m),$$

and we claim for $m : 1 \leq m \leq N$,

$$g(m) \geq 0, \tag{A.1}$$

$$m(g(m+1) + 1/s) = (m+1)g(m), \tag{A.2}$$

$$\frac{1}{s}(\ln(N+1) - 1) < g(1) \leq \frac{1}{s}\ln N, \tag{A.3}$$

$$\frac{1}{s}((N+1)/e - 2) < \max_{m:1\leq m\leq N} g(m) < N/(es). \tag{A.4}$$

Here we prove these four claims. The proof of (A.1) is trivial.

Proof of (A.2): Starting with the left hand side of Equation A.2, we have

$$\begin{aligned} m(g(m+1) + 1/s) &= m[\frac{m+1}{s}(H_N - H_m - \frac{1}{m+1}) + 1/s] \\ &= m[g(m) + \frac{1}{s}(H_N - H_m)] \\ &= mg(m) + \frac{m}{s}(H_N - H_m) \\ &= (m+1)g(m). \end{aligned}$$

□

Proof of (A.3): We have in general for $1 \leq m \leq N$:

$$\ln(N+1) - \ln(m+1) = \int_{m+1}^{N+1} \frac{1}{x} dx \leq H_N - H_m \leq \int_m^N \frac{1}{x} dx = \ln N - \ln m, \quad \text{(A.5)}$$

where the inequalities are strict for $m < N$. Since $g(1) = \frac{1}{s}(H_N - H_1)$, the above set of inequalities for $m = 1$ implies (A.3). □

Proof of (A.4): For $N \leq 4$, the first inequality in (A.4) is trivial. For $N \geq 5$, letting $m := \lfloor (N+1)/e \rfloor - 1$, we have

$$g(m) = \frac{1}{s}(\lfloor (N+1)/e \rfloor - 1)(H_N - H_m) > \frac{1}{s}((N+1)/e - 2)(H_N - H_m).$$

It remains to show $H_N - H_m \geq 1$. By our choice of m, we have $(N+1)/e \geq m+1$. Combining this inequality with (A.5), we have $H_N - H_m \geq \ln(N+1) - \ln(m+1) \geq 1$, which completes the proof of the first inequality in (A.4).

To prove the second inequality in (A.4), we need only consider the case in which $N \geq 3$, since it is straightforward to verify this inequality for $N \leq 2$. By (A.5), we have $g(m) \leq \frac{m}{s} \ln(N/m)$. For real-valued $x \in [1, N]$, the function $f(x) := x \ln(N/x)$ attains its maximum value only for $x = N/e$, at which $f(N/e) = N/e$. Since this maximum value is never attained for any integer m, we have for all $m, g(m) < N/(es)$, proving the second inequality in (A.4). □

A.2 Proofs of Claims from Section 2.7.2

Here we prove the following three claims, for $g(m)$ as defined in the previous section, for $1 \leq s \leq H_N - 1$, and for $M := \max\{j : H_N - H_j \geq s\}$:

1. $M < N/e^s$.

2. For fixed N and s, $g(m)$ is increasing for $m : 1 \leq m \leq M$.

3. $g(M) < M + 1$.

Proof of 1: By our assumption $1 \leq s \leq H_N - 1$, we have $N \geq 4$. Thus, $N/e^s \leq N/e \leq N-1$, so $\lceil N/e^s \rceil < N$. Then by (A.5) we have

$$\begin{aligned} H_N - H_{\lceil N/e^s \rceil} &< \ln N - \ln \lceil N/e^s \rceil \\ &\leq \ln N - \ln(N/e^s) \\ &= s, \end{aligned}$$

in which the first inequality is strict since the inequalities in (A.5) are strict for $m < N$. Thus, $M < N/e^s$. □

Proof of 2: Fix N and s. If $M = 1$, there is nothing to show. Else, for $1 \leq m \leq M-1$ we have $H_N - H_m > s \geq 1$. Thus, $g(m+1) - g(m) = \frac{1}{s}(H_N - H_m - 1) > 0$, which proves $g(m)$ is increasing for $m : 1 \leq m \leq M$. □

Proof of 3: We have

$$\begin{aligned} g(M) &= \frac{M}{s}(H_N - H_M) \\ &= \frac{M}{s}(H_N - H_{M+1} + 1/(M+1)) \\ &< M + \frac{M}{s(M+1)} \\ &< M + 1, \end{aligned} \tag{A.6}$$

where the third line follows since $H_N - H_{M+1} < s$. □

Appendix B

Recall the general network depicted in Figure 5-1 of Section 5.1. We refer to this general network as $\mathcal{N}$. It has three source-sink pairs $(a_1, b_1), (a_2, b_2), (a_3, b_3)$ and each arc has unit capacity. We prove the following theorem, which was claimed in Section 5.2.

Theorem 12 *For the general network depicted in Figure 5-1 of Section 5.1, for any $\epsilon > 0$, speedup $1 + \epsilon$ is sufficient to maintain bounded backlog for any fluid policy.*

Proof: We define a packet-scheduling algorithm that processes batches of consecutive fluid steps and uses pipelining, as in the algorithm in Section 5.2. The number of fluid steps in any batch (and the number of packetized steps in any batch) is set to be the even integer $\alpha := 2\lceil \frac{6/\epsilon+1}{2} \rceil$. We use pipelining so that each batch of packetized steps is scheduled α steps after the corresponding batch of fluid steps. For each batch of packetized steps output, the last three steps are the all zero flow over time. This ensures that packets sent in different batches cannot traverse the same arc simultaneously.

We use the static multicommodity flows $\{f^{(\beta)}\}_{\beta \geq 0}$, $\{p^{(\beta)}\}_{\beta \geq 0}$ defined in (5.1) and (5.2) of Section 5.2 to keep track of backlog at the end of each batch. Recall from Section 5.2 that the backlog for a source-sink pair $(a_i, b_i) \in \mathcal{T}$ at the end of batch β can be conveniently represented as

$$B(i, \alpha\beta) \quad = \quad \left(\sum_{j=0}^{\beta} f^{(j)}_{e(b_i),i} - p^{(j)}_{e(b_i),i} \right)^{+}. \tag{B.1}$$

The algorithm here maintains the following invariant.

Invariant 7 *For any $\beta \geq 0$, for any commodity $i \in \{1, 2, 3\}$, we have*

$$B(i, \alpha\beta) - p^{(\beta+1)}_{e(b_i),i} \leq 0.$$

Using a similar argument as that used in Section 5.2 to prove Invariant 6 implies bounded backlog for all time steps, we have that Invariant 7 implies backlog is bounded (for all source-sink pairs) for all time steps. Before describing the batch computation, we prove the following lemma, which will allow us to show the batch computation is well-defined.

Lemma 18 *For the general network $\mathcal{N}$, for $\beta \geq 1$, if Invariant 7 holds for batch $\beta - 1$, then*

$$\begin{aligned} B(3, \alpha\beta) + B(1, \alpha\beta) &\leq (\alpha + 1)/(1 + \epsilon), \\ B(1, \alpha\beta) + B(2, \alpha\beta) &\leq (\alpha + 1)/(1 + \epsilon), \\ B(2, \alpha\beta) + B(3, \alpha\beta) &\leq (\alpha + 1)/(1 + \epsilon). \end{aligned}$$

Proof: From the structure of the general network $\mathcal{N}$, we have for any $t > 0$ that the fluid sent from a_3 to b_3 at time t and the fluid sent from a_1 to b_1 at time $t + 1$ traverse the same arc (the only horizontal arc in Figure 5-1) at time step $t + 2$. Since each link in $\mathcal{N}$ has unit capacity, and the algorithm uses speedup $1 + \epsilon$, we have that the sum of this fluid is at most $1/(1 + \epsilon)$. That is,

$$F^{(t)}_{e(a_3),3}(t) + F^{(t+1)}_{e(a_1),1}(t + 1) \leq 1/(1 + \epsilon).$$

By the flow conservation constraints and the structure of $\mathcal{N}$, any fluid sent from source a_i traverses arc $e(b_i)$ exactly three time steps later. We thus have for sinks b_3 and b_1 that

$$F^{(t)}_{e(b_3),3}(t + 3) + F^{(t+1)}_{e(b_1),1}(t + 4) \leq 1/(1 + \epsilon). \tag{B.2}$$

The flow conservation constraints and the structure of $\mathcal{N}$ also imply that for each $i \in \{1,2,3\}$, $F^{(t)}_{e(b_i),i}(t') = 0$ unless $t' = t+3$. If we sum the expression (B.2) over the time steps $t \in [(\beta-1)\alpha, \beta\alpha]$, we get from the previous sentence, and using the definitions of $f^{(\beta)}, p^{(\beta)}$ from (5.1), (5.2), that

$$f^{(\beta)}_{e(b_3),3} + f^{(\beta)}_{e(b_1),1} \leq (\alpha+1)/(1+\epsilon). \tag{B.3}$$

By (B.1), we have for the backlog at the end of batch β,

$$B(i,\alpha\beta) \leq \left(B(i,\alpha(\beta-1)) - p^{(\beta)}_{e(b_i),i}\right)^+ + f^{(\beta)}_{e(b_i),i}.$$

Assuming Invariant 7 holds for batch $\beta-1$, we then have by the previous inequality and (B.3) that

$$B(3,\alpha\beta) + B(1,\alpha\beta) \leq (\alpha+1)/(1+\epsilon). \tag{B.4}$$

Similar arguments as above imply the other two inequalities in the lemma. □

Batch Computation

We now describe how the algorithm computes each batch of packetized steps, showing inductively that Invariant 7 is maintained for each batch $\beta \geq 0$. Initially, for $1 \leq t \leq \alpha$, we set $P^{(t)} := \mathbf{0}$, the all zero flow over time. The invariant thus trivially holds for $\beta = 0$. For $\beta \geq 1$, given new batch of consecutive fluid steps $F^{((\beta-1)\alpha+1)}, \ldots, F^{(\beta\alpha)}$, the algorithm constructs the batch of packetized steps $P^{(\beta\alpha+1)}, \ldots, P^{((\beta+1)\alpha)}$ as follows.

The goal of the packet-scheduling algorithm is to schedule packets such that each source a_i sends a total of $\lceil B(i,\alpha\beta)\rceil$ packets during time steps $[\beta\alpha+1, (\beta+1)\alpha]$, and such that no packets are sent during the last three time steps of this interval. This will maintain the invariant for batch β.

We show how the algorithm works in the case in which

$$B(1,\alpha\beta) \geq B(2,\alpha\beta) \geq B(3,\alpha\beta).$$

The other 5 possible orderings of $B(1,\alpha\beta), B(2,\alpha\beta), B(3,\alpha\beta)$ are treated symmetrically by the algorithm.

We partition the interval of time steps during which this batch of packetized steps will be sent, that is $[\beta\alpha+1,(\beta+1)\alpha]$, into the following four consecutive subintervals. (We use the convention that the interval $[x,y]$ is empty if $x>y$.)

$$\begin{aligned}
I_1 &:= [\beta\alpha+1, \beta\alpha+2\lceil B(3,\alpha\beta)\rceil],\\
I_2 &:= [\beta\alpha+1+2\lceil B(3,\alpha\beta)\rceil, \beta\alpha+2\lceil B(2,\alpha\beta)\rceil],\\
I_3 &:= [\beta\alpha+1+2\lceil B(2,\alpha\beta)\rceil, \beta\alpha+\lceil B(1,\alpha\beta)\rceil+\lceil B(2,\alpha\beta)\rceil],\\
I_4 &:= [\beta\alpha+1+\lceil B(1,\alpha\beta)\rceil+\lceil B(2,\alpha\beta)\rceil, (\beta+1)\alpha]
\end{aligned}$$

Assuming that Invariant 7 holds for the previous batch $\beta-1$, Lemma 18 and our choice of α imply $\lceil B(1,\alpha\beta)\rceil+\lceil B(2,\alpha\beta)\rceil \le (\alpha+1)/(1+\epsilon)+2 \le \alpha-3$. Thus, the above partition is well defined, and I_4 contains at least three time steps.

The algorithm behaves differently during each of the above four intervals. During the odd time steps of I_1, each source sends a packet, and during the even time steps of this interval, no source sends any packets. During the odd time steps of I_2, sources a_1 and a_2 each send a packet, and during the even time steps of this interval, no source sends any packets. During I_3, only source a_1 sends a packet, and it does so at each time step in I_3. During I_4, no packets are sent.

One can verify that each source a_i sends a total of $\lceil B(i,\alpha\beta)\rceil$ packets during time steps $[\beta\alpha+1,(\beta+1)\alpha]$, and that at most one packet traverses each arc at each time step. Since no packets are sent during I_4, which contains the last three time steps, each destination b_i receives a total of $\lceil B(i,\alpha\beta)\rceil$ packets during time steps $[\beta\alpha+1,(\beta+1)\alpha]$. Thus, for all $i\in\{1,2,3\}$, we have $B(i,\alpha\beta)-p^{(\beta+1)}_{e(b_i),i}\le 0$, and so Invariant 7 is maintained for batch β. By induction, the invariant holds for all batches $\beta\ge 0$, and the theorem is proved. □

Bibliography

[1] Micah Adler, Petra Berenbrink, Tom Friedetzky, Leslie Ann Goldberg, Paul W. Goldberg, and Mike 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.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1993.

[3] William Aiello, Eyal Kushilevitz, Rafail Ostrovsky, and Adi Rosén. Adaptive packet routing for bursty adversarial traffic. In *Proceedings of the 30th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 359–368, 1998.

[4] Matthew Andrews, Baruch Awerbuch, Antonio Fernandez, Tom Leighton, Zhiyong Liu, and Jon Kleinberg. Universal-stability results and performance bounds for greedy contention-resolution protocols. *Journal of the ACM*, 48(1):39–69, 2001.

[5] Matthew Andrews and Lisa Zhang. The effects of temporary sessions on network performance. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 448–457, San Francisco, California, United States, 2000.

[6] Jon C. R. Bennett and Hui Zhang. WF^2Q: worst-case fair weighted fair queueing. In *INFOCOM (1)*, pages 120–128, 1996.

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

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

[9] Allan Borodin, Jon Kleinberg, Prabhakar Raghavan, Madhu Sudan, and David P. Williamson. Adversarial queuing theory. *Journal of the ACM*, 48(1):13–38, 2001.

[10] C. Caramanis, M. Rosenblum, M.X. Goemans, and V. Tarokh. Scheduling algorithms for providing flexible, rate-based, quality of service guarantees for packet-switching in Banyan networks. In *Proceedings of the 38th Annual Conference on Information Sciences and Systems (CISS)*, pages 160–166, Princeton, NJ, 2004.

[11] Cheng-Shang Chang, Wen-Jyh Chen, and Hsiang-Yi Huang. Birkhoff-von Neumann input buffered crossbar switches. In *INFOCOM (3)*, pages 1614–1623, 2000.

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

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

[14] R. Cole, K. Ost, and S. Schirra. Edge-coloring bipartite multigraphs in $O(E \log D)$ time. *Bolyai Society–Springer-Verlag Combinatorica*, 21(1):5–12, 2001.

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

[16] R. Cruz. A calculus for network delay. I. network elements in isolation. *IEEE Transactions on Information Theory*, 37(1):114–131, January 1991.

[17] R. Cruz. A calculus for network delay. II. network analysis. *IEEE Transactions on Information Theory*, 37(1):132–141, January 1991.

[18] J. G. Dai and Balaji Prabhakar. The throughput of data switches with and without speedup. In *INFOCOM (2)*, pages 556–564, 2000.

[19] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. *Journal of Internetworking: Research and Experience*, 1(1):3–26, 1990.

[20] Antoine Deza, Komei Fukuda, Dmitrii Pasechnik, and Masanori Sato. On the skeleton of the metric polytope. *In J. Akiyama, M. Kano, and M. Urabe, editors, Lecture Notes in Computer Science, Springer-Verlag*, 2098:125–136, 2001.

[21] Nick G. Duffield, T. V. Lakshman, and Dimitrios Stiliadis. On adaptive bandwidth sharing with rate guarantees. In *INFOCOM (3)*, pages 1122–1130, 1998.

[22] Lisa Fleischer and Martin Skutella. *The Quickest Multicommodity Flow Problem*. William J. Cook and Andreas S. Schulz (eds.): Integer Programming and Combinatorial Optimization, Lecture Notes in Computer Science 2337, Springer, Berlin, 2002.

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

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

[25] A.V. Goldberg and S. Rao. Beyond the flow decomposition barrier. *Journal of the ACM*, (45):783–797, 1998.

[26] Mark Goudreau, Stavros G. Kolliopoulos, and Satish Rao. Scheduling algorithms for input-queued switches: Randomized techniques and experimental evaluation. In *INFOCOM*, pages 1624–1643, 2000.

[27] M. Grotschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer–Verlag, Berlin Heidelberg, 1993.

[28] Ellen L. Hahne, Alexander Kesselman, and Yishay Mansour. Competitive buffer management for shared-memory switches. In *ACM Press Proceedings of the thirteenth annual ACM symposium on Parallel algorithms and architectures*, pages 53–58, 2001.

[29] P. Hall. On representatives of subsets. *The Journal of the London Mathematical Society*, (10):26–30, 1935.

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

[31] J. Y. Hui. *Switching and Traffic Theory for Integrated Broadband Networks*. Kluwer Academic Publishers, Boston, 1990.

[32] T. Inukai. An efficient SS/TDMA time slot assignment algorithm. *Communications, IEEE Transactions*, 27(10):1449–1455, 1979.

[33] J.R. Jackson. Scheduling a production line to minimize maximum tardiness. Research Report 43, Management Science Research Project. 1955.

[34] Anthony C. Kam and Kai-Yeung Siu. Linear complexity algorithms for qos support in input-queued switches with no speedup. *IEEE Journal on Selected Areas in Communications*, 17(6):1040–56, June 1999.

[35] S. Keshav. *An Engineering Approach to Computer Networking*. Addison–Wesley Pub Co, Boston, MA, 1997.

[36] J. M. Kleinberg. Approximation algorithms for disjoint paths problems, Ph.D. dissertation, MIT, May 1996.

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

[38] D. Konig. Graphok és alkalmazásuk a determinánsok és a halmazok elméletére [hungarian]. *Mathematikai és Természettudományi Értesito*, (34):104–119, 1916.

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

[40] J. E. Marsden and M.J. Hoffman. *Elementary Classical Analysis, Second Edition*. W.H. Freeman and Company, New York, 1993.

[41] Nick McKeown, Venkat Anantharam, and Jean C. Walrand. Achieving 100% throughput in an input-queued switch. In *INFOCOM (1)*, pages 296–302, 1996.

[42] T.S. Motzkin, H. Raiffa, GL. Thompson, and R.M. Thrall. *Contributions to theory of games; editors, H.W. Kuhn and A.W.Tucker*, volume 2. Princeton University Press, Princeton, RI, 1953.

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

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

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

[46] Balaji Prabhakar and Nick McKeown. On the speedup required for combined input and output queued switching. Technical Report CSL-TR-97-738, Stanford University, CA, 1997.

[47] M. Rosenblum, M. X. Goemans, and V. Tarokh. Universal bounds on buffer size for packetizing fluid policies in input queued, crossbar switches. In *Proceedings of IEEE INFOCOM*, Hong Kong, China, 2004.

[48] M. Rosenblum, R. Yim, M.X. Goemans, and V. Tarokh. Worst-case delay bounds for packetizing time-varying fluid schedules for a single server in a packet-switched network. In *Proceedings of the 38th Annual Conference on Information Sciences and Systems (CISS)*, pages 1553–1559, Princeton, NJ, 2004.

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

[50] Alexander Schrijver. Bipartite edge-colouring in $O(\Delta m)$ time. *SIAM Journal on Computing*, 28:841–846, 1999.

[51] Aleksandra Smiljanić. Flexible bandwidth allocation in high-capacity packet-switches. *IEEE/ACM Transactions on Networking*, 10(2), April 2002.

[52] 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.

[53] Anastasios Stamoulis and Georgios B. Giannakis. Deterministic time-varying packet fair queueing for integrated services networks. In *IEEE Global Telecommunications Conference*, volume 1, pages 621–625, 2000.

[54] John A. Stankovic, Marco Spuri, Krithi Ramamritham, and Giorgio C. Buttazzo. *Deadline Scheduling for Real-Time Systems. EDF and Related Algorithms*. Kluwer Academic Publishers, Boston, 1998.

[55] Vahid Tabatabaee, Leonidas Georgiadis, and Leandros Tassiulas. QoS provisioning and tracking fluid policies in input queueing switches. In *INFOCOM*, pages 1624–1633, 2000.

[56] Éva Tardos. A strongly polynomial algorithm to solve combinatorial linear programs. *Operations Research*, 34:250–256, 1986.

[57] Leandros Tassiulas. Linear complexity algorithms for maximum throughput in radio networks and input queued switches. In *Proc. IEEE Infocom '98*, pages 533–539, 1998.

[58] Leandros Tassiulas and Leonidas Georgiadis. Any work-conserving policy stabilizes the ring with spatial re-use. *IEEE/ACM Transactions on Networking*, 4(2):205–208, 1996.

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

[60] R. Wenger. *Helly-type theorems and geometric transversals.* In: J.E. Goodman and J. O'Rourke, editors, Handbook of Discrete and Computational Geometry, CRC Press, 1997.