May 1991

LIDS-TH-2035

National Science Foundation
Grant NSF-ECS-8552419

Bellcore, Inc.

Dupont

Army Research Office
Grant ARO-DAAL03-86-K-0171

ROUTING AND PERFORMANCE EVALUATION IN INTERCONNECTION NETWORKS

George D. Stamoulis

May 1991 LIDS-TH-2035

ROUTING AND PERFORMANCE EVALUATION IN INTERCONNECTION NETWORKS

by

George D. Stamoulis

This report is based on the unaltered thesis of George D. Stamoulis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Electrical Engineering and Computer Science at the Massachusetts Institute of Technology in May 1991. This research was carried out at the M.I.T. Laboratory for Information and Decision Systems, and was supported by the National Science Foundation under Grant NSF-ECS-8552419 with matching funds from Bellcore Inc. and Dupont, and by the Army Research Office under Grant ARO-DAAL03-86-K-0171.

Massachusetts Institute of Technology
Laboratory for Information and Decision Systems
Cambridge, MA 02139

ROUTING AND PERFORMANCE EVALUATION IN INTERCONNECTION NETWORKS

by

GEORGE D. STAMOULIS

Diploma, Electrical Engineering, National Technical University of Athens, 1987.
S.M., Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1988.
Electrical Engineer, Massachusetts Institute of Technology, 1991.

SUBMITTED TO THE DEPARTMENT OF
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June 1991

Signature of Author _____
Department of Electrical Engineering and Computer Science
May 8, 1991

Certified by _____
John N. Tsitsiklis
Associate Professor of Electrical Engineering
Thesis Supervisor

Accepted by _____
Arthur C. Smith
Chairman, Department Committee on Graduate Studies

ROUTING AND PERFORMANCE EVALUATION IN INTERCONNECTION NETWORKS

by

GEORGE D. STAMOULIS

Submitted to
the Department of Electrical Engineering and Computer Science
on May 8, 1991 in partial fulfillment
of the requirements for the Degree of

DOCTOR OF PHILOSOPHY

ABSTRACT

We analyze routing problems for message-passing parallel computers. The underlying network topology is the binary hypercube. We assume that each message is transmitted in a packet of unit length. First, we study static problems, where all packets involved are available at the same time. We derive an algorithm for performing the total exchange task (matrix transposition) in the shortest possible time. We also derive an efficient algorithm for the task where each member of a subset of nodes wishes simultaneously to broadcast a packet to all other nodes. The completion time of this algorithm is within a small multiplicative factor from the corresponding lower bound, for any possible subset of broadcasting nodes. Both of our static algorithms can be easily implemented in a distributed fashion.

We then consider dynamic routing problems, where packets are generated at random times over an infinite time-horizon. The analysis of such problems constitutes the main focus of our research. Our motivation is the understanding of the communication issues arising in general purpose asynchronous computation. First, we analyze the problem of multiple node-to-node communications in the hypercube: each node generates packets according to a Poisson process; each packet has a single destination, which is selected randomly. We consider a simple greedy routing scheme, where every packet takes a particular shortest path leading from its origin to its destination. We analyze this scheme by treating the hypercube as a queueing network with deterministic servers, and by using a stochastic comparison with a product-form queueing network. We thus prove that the greedy scheme can sustain any throughput value less than 1, while inducing an average delay of $\Theta(d)$ (that is, of the order of magnitude of the diameter d of the hypercube) for any fixed throughput value. We also prove that, in heavy traffic, the average delay has optimal asymptotic behavior. We then extend these results to the butterfly network. Finally, we consider the dynamic routing problem of multiple broadcasts in the hypercube, where packets to be broadcast are generated according to Poisson processes. We devise and analyze two routing schemes that can sustain high throughput values regardless of the size of the hypercube; at the same time, in light traffic, the average time required to complete a broadcast is $\Theta(d)$, which is the optimal order of magnitude. The first routing scheme is analyzed exactly and closed form results are obtained. The performance of the second scheme is investigated using an approximate model; a simulation study shows that the resulting formulas are very accurate. All of the routing schemes considered in both dynamic problems are distributed and of the on-line type; that is, all routing decisions are made on the basis of past information only.

Thesis Supervisor: Dr. John N. Tsitsiklis
Title: Associate Professor of Electrical Engineering

Acknowledgements

I would like to express my deepest gratitude to Professor John Tsitsiklis, my thesis advisor, for his wise and generous guidance, as well as for his unfailing encouragement and support throughout my studies at M.I.T. Besides an advisor, he has been an invaluable friend.

I am thankful to my thesis readers, Professors Dimitri Bertsekas and Tom Leighton, for the stimulating discussions we have had and for their suggestions on my thesis. I am also indebted to Professor Bertsekas for his friendship and support during the past years.

Professor Jeffrey Shapiro, my academic counselor, has always been very encouraging and helpful.

I wish to thank Dr. Teunis Ott of Bellcore for his helpful suggestion regarding Chapter 5.

My thanks to all members of the Laboratory for Information and Decision Systems — faculty, students and staff — for an excellent working environment.

For their love and support, I am grateful to my family. Without them, nothing would have been possible.

Table of Contents

List of Figures

Chapter 2

Chapter 3

Chapter 4

Chapter 5

Chapter 6

Chapter 7

1. Introduction

1.1 PARALLEL COMPUTERS

The impressive progress in hardware technology during the past decades has enabled the use of multiple processors within the same computer. In fact, there exist computers with as many as 65,000 processors. Such systems are primarily used for the solution of large numerical problems, such as the ones arising in weather prediction, computational fluid dynamics, image processing, optimization etc. In these applications, multiple calculations (or other operations) related to the same problem are *concurrently* performed by different processors. Thus, by parallelizing a significant portion of the computations, the solution is derived much faster than in a serial computer. Typically, processors within a parallel computer have to exchange information, in order to proceed with their respective calculations. There are basically two approaches for establishing this kind of communication. Under the first one, prevalent in *message-passing* systems, processors communicate by sending messages to each other through an *interconnection network*. Under the other approach, adopted in the so-called *shared-memory* systems, processors communicate by reading and writing into a shared part of the memory. In reality, however, such systems include either a bus or a

crossbar switch, connecting all processors to all memory locations. Thus, a significant amount of communications take place even in a shared-memory system.

With the increasing size of the parallel computers in use and the increasing need for fast and real-time computation, performing the interprocessor communications efficiently has become an extremely important issue. Thus, related *routing* problems have received extensive attention of researchers, especially during the past decade. In this Ph.D. Thesis, we analyze several such problems, primarily pertaining to message-passing systems. Most of our analysis is closely related to the popular hypercube architecture. Nevertheless, we also investigate more general issues, as well as develop analytical tools that are more broadly applicable.

1.2 COMMUNICATIONS IN MESSAGE-PASSING SYSTEMS

The discussion to follow refers to a *message-passing* system. Associated with each such system is a network, where each node corresponds to a different processor. Ideally, it would be desirable for this network to be *complete*, that is, each node to be connected to all other nodes. Since this is not practically feasible unless the number of processors is very small, a multitude of special network *topologies* have been developed, such as the *linear array*, the *2-dimensional array*, the *binary hypercube*, the *pyramid* etc; see [BeT89]. Of course, the performance of a parallel computer depends significantly on the underlying topology. Before addressing this topic in §1.3, we discuss some general issues without making any reference to a particular network architecture. We shall only assume that the network graph is connected, i.e. there is a *directed path* connecting each pair of nodes; note that each (directed) *arc* of this graph corresponds to a *directed communication link* between the start and the end nodes.

1.2.1 Prototype Communication Tasks

There are certain prototype communication tasks (scenarios) that arise very often during the execution of parallel algorithms; see [BeT89]. In this subsection, we briefly present those of the prototype tasks that are related to our research.

The simplest conceivable communication task is *node-to-node communication*, where some node wishes to send a message to some other node. This task may be accomplished by transmitting the message along one of the paths connecting the two end-

nodes. Another simple task is the *single node broadcast*, where some node wishes to send the *same* message to all other nodes; this task arises, for example, when one of the processors has computed the final result of a series of calculations and wishes to report it to all other processors. A single node broadcast may be accomplished by transmitting the message along a spanning tree (of the network graph) emanating from the broadcasting node. (That is, a spanning tree with all of its arcs pointing away from this node.)

A generalization of the single node broadcast task is the *multinode broadcast*, where all nodes wish to perform a broadcast at the same time. This situation arises in parallel iterative algorithms of the form $x := f(x)$, where $f : \Re^n \rightarrow \Re^n$ and n is the number of processors; typically, the ith processor knows the function f_i and updates x_i. Assume that the problem is *dense*, i.e. each entry of the function $f(x)$ depends explicitly on almost all entries of x; then, once x_i is updated, its new value must be broadcast to all other processors, in order to be used in their subsequent calculations. If all processors are perfectly *synchronized*, then all entries of the vector x become available to be broadcast at the same time.

In the prototype tasks involving broadcasts, each transmitting node sends the same message to all other nodes. Other interesting tasks arise when this is not the case. In particular, when some node wishes to send a *different* message to each of the other nodes, then a *single node scatter* arises. This situation may occur when a certain processor has some special responsibility in coordinating the operation of the network. An even more general task is the *total exchange*, where *every* node wishes to transmit a different message to each of the other nodes. This task arises in computing the transpose of a matrix A, in the case where each processor stores initially a different row. Indeed, assume that processor i knows initially all entries $a_{i1}, \ldots, a_{in}$; after transposition of the matrix, node k should store all entries $a_{1k}, \ldots, a_{nk}$. Therefore, for each pair (i, k) where $k \neq i$, processor i has to send the value of entry a_{ik} to processor k; clearly, this situation corresponds to a total exchange. Finally, of interest is also the *permutation* task, where each node sends exactly one message and receives exactly one message; this task is often considered as a "benchmark" for comparing the performance of different networks.

1.2.2 Static and Dynamic Routing Problems

As discussed in the previous subsection, the set of origin-destination pairs associated with a prototype task usually has a special structure. In the context of the network architectures used in practice, this fact usually enables the development of efficient routing algorithms for such tasks. When designing such an algorithm, it is more convenient to assume that all messages to be transmitted are available at the same time and that the communication task is to be performed *once* and in the *absence* of other transmissions in the network. These favorable assumptions apply to cases where all (or almost all) of the processors are actually engaged in the same problem and they are synchronized with each other to a significant extent. A routing problem arising under these conditions is called *static*, because it is known *a priori* exactly what messages it involves. When designing a routing algorithm for such a problem, the objective is usually to minimize the *completion* time of the algorithm, that is to complete all transmissions as fast as possible. Most often, it is also desirable for a fast algorithm to be efficient with respect to the number of transmissions involved and the amount of memory required, and not to be complicated to implement.

Unlike the case of static problems, the situation is less predictable in the so-called *dynamic* routing problems, where it is assumed that there are multiple tasks to be performed; these tasks are taken to be generated at *random* time instants over an *infinite* time-horizon, and they may *interfere* with each other. For such a problem, it is only meaningful to design a routing *scheme*, namely a set of rules to be applied *regardless* of what messages will be generated in the future; such a scheme should be of the *on-line* type, that is each message should be routed only on the basis of *past* information. When dealing with a dynamic routing problem, the objective is to design a scheme that makes efficient use of the available resources, thus attaining satisfactory *throughput*, while introducing a small amount of *delay* per task.

In order to clarify the distinction between the two types of problems, let us consider the iterative algorithm $x := f(x)$ mentioned in §1.2.1. If all processors are synchronized, then a multinode broadcast task will be performed at the end of each iteration. If no other transmissions take place in the underlying interconnection network, then designing the corresponding algorithm is a static problem; the same algorithm will be executed periodically, namely once after each iteration. On the other hand, the situa-

tion is different in an *asynchronous* environment, where processors operate at *variable* speeds; thus, each processor completes its kth iteration at a different time, and proceeds with the next one *without* waiting for processors that are running late. In such a case, there arise multiple *single node* broadcasts, generated at various time instants and interfering with each other. Of course, one could claim that the multinode broadcast also consists of a multitude of interfering single node broadcasts. Even though this is true, the problem is still characterized as static, because all of the "constituent" single node broadcasts are *known* to be generated at the same time.

1.2.3 The Model for Communications

In the course of our analysis, we adopt a simple model for communications; this is as follows:

(a) The time axis is divided in *slots* of unit length; all nodes follow the same clock.

(b) Each message is transmitted in a *single packet*. Transmitting a packet from some node to one of its neighbors takes *unit* time; this includes the "actual" transmission time, the propagation delay and time for processing.

(c) Packets propagate within the network in a *store-and-forward* fashion.

(d) Only one packet may traverse a (directed) arc per slot; all transmissions are error-free.

(e) Each node can transmit packets through *all* of its output ports and at the same time receive packets through *all* of its input ports.

(f) Each node has *infinite* buffer capacity.

A few comments on our model are in order. First, the assumption that packets have fixed length is in agreement with several standards for communications, such as the Asynchronous Transfer Mode for lightwave networks [Min89]. The assumption that each message consists of only one packet is somewhat simplistic, since there may be messages that do not "fit" in a single packet. In other models appearing in the literature (e.g. the ones of [SaS85] and [JoH89]), messages are taken to consist of several packets of unit size, which may be split and recombined later. Such an extension can be easily accommodated by the routing algorithms designed under our model; the converse is *not* always true, as will be seen in §1.4.1.

The assumption that messages are taken to propagate in a store-and-forward fashion is again standard in the context of parallel computers. Another alternative, which has recently attracted increased attention, is the so-called *wormhole* routing, where each bit of a packet is forwarded immediately towards the destination, without "waiting" for the tail of the packet; see [DaS87]. This may prove advantageous in cases where messages are rather long and cannot be split in shorter packets, while the traffic is *light*. On the other hand, under wormhole routing, additional provision has to be taken for deadlock prevention, which complicates the routing algorithms. We prefer to adopt the "traditional" store-and-forward routing, because it is still applied widely; moreover, due to its simplicity, it provides us with more insight of the main issues affecting the performance of the various routing algorithms. In fact, for all routing problems to be analyzed, the performance of *any* algorithm (or routing scheme) is *limited* by the usage of some critical communication resource. All routing algorithms to be presented attain *high* and *efficient* utilization of the respective critical resource; thus, introducing wormhole routing would not provide considerable opportunity for designing algorithms with improved performance.

Finally, the assumption of infinite buffer capacity is not realistic; however, it is adopted mainly for analytical convenience. In the course of our analysis, we shall also derive estimates of the buffer capacity required in practice.

Though simple, we believe that our model captures the essence of communications in the context of message-passing systems, while it allows the design of routing algorithms that are both powerful and easily adaptable to other models.

1.2.4 Distributed Routing

When solving a problem in a parallel computer, there is always the need for some *centralized* coordination of the various processors; for example, when a prototype communication task arises, some special mechanism has to trigger the initialization of the task. It is often required that there is only limited use of centralized control. This requirement is motivated by the fact that excessive application of centralized control would occupy a considerable amount of the system's computational and communication resources. An algorithm that makes *minimal* use of centralized control is called *distributed*. This definition is not precise, because it not entirely clear what the "minimum" possible extent of centralized coordination is for each particular problem; below,

we clarify our notions of distributed *routing* algorithms and schemes.

For a static routing problem, we shall assume that all nodes are notified of the communication scenario and when transmissions start. In a distributed algorithm for such a problem, we shall assume that each node knows when to transmit its *own* packets and which paths to choose for each of them; regarding the packets it has to *forward*, each node can figure when to do so, by looking at the paths of the packets received and by performing some *local* computations. It will be assumed that each packet has a special field of information, where its path is stored. (In fact, it is often the case that the path to be followed by a packet can be determined by the corresponding origin-destination pair and some small amount of additional information, without having to provide the entire sequence of intermediate nodes.) Similar assumptions apply to the distributed schemes pertaining to dynamic routing problems.

As already mentioned, the advantage of a distributed algorithm lies on the fact that the central controller imposes limited additional load to the system; on the other hand, such an algorithm may be more complicated. However, an algorithm designed for a distributed environment can be easily adapted to run in the presence of centralized control; the converse is usually not true. Therefore, for all routing problems to be analyzed, we shall be primarily interested in designing distributed algorithms (or schemes).

1.3 SELECTING A NETWORK ARCHITECTURE

So far, we have discussed several general issues on communications in message-passing systems, without referring to any particular architecture. Among the multitude of architectures used in practice, we have selected the d-dimensional *binary hypercube* as the underlying network of most of our analysis. The nodes of this network constitute a lattice of points of the d-dimensional space, with each point having *binary*-valued coordinates. Thus, the d-dimensional binary hypercube (to be also referred to as d-cube) has 2^d nodes, with each of them having d neighbors, namely the d "nearest" nodes in the d-dimensional space. (A formal definition of the hypercube network is presented in §2.1, together with the most important of its topological properties.) The primary reasons for selecting the hypercube as our main topology of reference are as follows:

(a) Existence of several actual multiprocessing systems using the hypercube topology, such as the Connection Machine (with 64K processors), the NCUBE/10 (with 1K processors) etc; see [Hwa87].

(b) Close relation of the hypercube with several of the other network topologies used in practice. Such topologies can be either imbedded in the hypercube (e.g., the 2-dimensional array) or obtained therefrom by introducing certain modifications (e.g., the butterfly).

(c) Potential for designing very efficient routing algorithms and schemes, due to powerful topological properties.

(d) Potential for neat analysis of the problems of interest.

In order to clarify the advantages of the hypercube, we shall briefly compare it with the 2-dimensional array, the nodes of which form a square lattice in the 2-dimensional space; this network is particularly suitable for the numerical solution of partial differential equations (with discretization of the continuous variables). For a generic problem however, it is not that appropriate, due to its large *diameter*. Indeed, for a 2-dimensional array with N nodes, the diameter is $\Theta(\sqrt{N})$; since the degree per node of the 2-dimensional array is 4, the ideal would have been to have a diameter of $\Theta(\ln N)$. [The notation $\Theta(\cdot)$ denotes order of magnitude.] On the other hand, a hypercube with N nodes has diameter $\log_2 N$, at the expense of increasing the degree per node to $\log_2 N$; thus, communication among "remote" nodes can be performed much faster in this network.

The only disadvantage of the hypercube is that it has logarithmic in the number of nodes degree; this property hinders the construction of actual hypercubes with *arbitrarily* many processors. Nevertheless, there is very little difficulty in practice, due to the impressive progress in hardware technology. For theoretical reasons primarily, there have been constructed several constant-degree networks that can *simulate* the hypercube with small slowdown; e.g., the Cube-Connected Cycles (see [BeT89]). However, most of the routing algorithms for these networks are first designed in the context of the hypercube, and then they are converted according to standard processes; see [LeL90].

To summarize, the hypercube constitutes a network architecture that is both exciting to analyze and important for practical applications.

1.4 A BRIEF SURVEY OF THE ROUTING LITERATURE

In the present section, we briefly summarize the routing literature; it should be noted that publications closely related to our research are discussed in §1.5.

The literature on *static* routing problems in the various network topologies is rather extensive. Regarding static routing in the hypercube, Bertsekas et al. [BOSTT91] have devised optimal algorithms for a variety of prototype tasks under the model for communications discussed in §1.2.3. Previously, Saad and Schultz [SaS85], as well as Johnsson and Ho [JoH89], had constructed optimal or nearly optimal algorithms for the same tasks, under a somewhat different model for communications. In particular, they assume that messages consist of a fixed number of unit-size packets, and they allow both splitting the messages and recombining the constituent packets at no overhead. In [Var90], Varvarigos developed a methodology for designing optimal algorithms for a class of symmetric communication tasks (called isotropic tasks); Varvarigos also analyzed the multinode broadcast task, under the assumption that packets have random lengths with exponential distribution. Other references related to static routing in the hypercube may be found in [BeT89], [JoH89] and [BOSTT91].

The communication tasks considered in the aforementioned articles as well as the respective algorithms do not employ any *randomization*. In his famous article [Val82], Valiant has demonstrated how to use randomization in order to perform a deterministic task. In particular, in the context of the d-cube, he considered the permutation task and showed that it may be accomplished in time $\Theta(d)$ with high probability, by using a randomized two-phase algorithm. [Recall that the diameter of the d-cube equals d; hence, most permutations require at least $\Theta(d)$ time units.] In the first phase of the algorithm in [Val82], each packet chooses a random intermediate destination (with all nodes being equiprobable) and is sent there; in the second phase, each packet travels from its intermediate destination to its actual one. In a later work, Valiant and Brebner [VaB81] modified this algorithm, thus simplifying considerably the analysis.

The permutation task is also important for emulating *shared-memory* systems. As already mentioned in §1.1, such systems usually include a crossbar switch network, connecting the various processors with the memory locations. Thus, an important problem is to route permutations where each node of one front of the switch (namely, each processor) has a packet to send to a different node of the opposite front (that is, to

a different memory location). Aleliunas [Ale82] and Upfal [Upf84] derived randomized permutation algorithms for constant-degree networks; by using a two-phase approach (such as in [Val82] and [VaB81]), these algorithms complete in time $\Theta(\ln N)$ with high probability, where N is the number of nodes. [Notice that $\Theta(\ln N)$ is the optimal order of magnitude for the completion time, because the diameter is also $\Theta(\ln N)$.] Further improvement was attained by Pippenger [Pip84], who derived an algorithm with similar performance but requiring only constant-size buffer at each node. However, under this algorithm, a deadlock may arise with small (yet positive) probability. The first deadlock-free permutation algorithm with constant-size buffers was derived by Ranade [Ran87]; this algorithm pertains to the butterfly network, and runs in time $\Theta(\ln N)$ with high probability.

Each of the aforementioned references is dealing with static routing in the hypercube or in some other standard network. Static routing problems have also been analyzed for *general* network graphs. Leighton and Rao [LeR88] have analyzed the uniform multicommodity flow problem, which is essentially an FDMA version of the total exchange task; these authors establish an upper and a lower bound for the maximum possible rate (per node pair) of exchanging information. Leighton et al. [LMR88] have analyzed the scheduling problem for routing n node-to-node communications over a set of *prespecified* paths (in a general graph); for each such set of paths, they prove the existence of an efficient off-line scheduling (that is, to be derived prior to starting transmissions), which involves only constant-size queues.

All of the references mentioned so far are dealing with static routing problems. The literature on dynamic problems is less extensive; most of the related works are discussed in §1.5.2.

1.5 OVERVIEW OF THE RESULTS — MOTIVATION — COMPARISON WITH PREVIOUS WORK

In this section, we present a summary of the specific problems analyzed and the results derived in this Ph.D. Thesis; we also discuss our motivation for considering these problems and we compare our work to the related literature.

Prior to presenting our research in Chapters 3-7, we give some background material in Chapter 2. In particular, we present the definition of the hypercube network and

the most important of its topological properties, together with some imbeddings of spanning trees to be used in the analysis. Later in that chapter, we describe some queueing systems to be encountered in the subsequent analysis; these systems are considered in *discrete time* and involve deterministic servers.

1.5.1 Static Routing Problems

In Chapters 3 and 4, we consider two static routing problems in the context of the d-cube.

In Chapter 3, we analyze the *total exchange* task, where each node has a different packet to send to each of the other nodes; see also §1.2.1. We derive a simple lower bound for the time required for *any* algorithm to perform this task, namely that the completion time is always at least 2^{d-1} time units. Then, we prove the existence of an algorithm that attains *exactly* this lower bound. Apart from being unimprovable with respect to its completion time, this algorithm also attains 100% utilization of the network arcs. Initially, the algorithm is constructed *recursively* with respect to the dimensionality d of the network. Following the analysis, we present a relatively simple *distributed* implementation for this optimal algorithm. As already mentioned in §1.2.1, the total exchange is among the prototype tasks arising in matrix computations, and thus, it is of interest to devise an optimal algorithm for performing it.

Our optimal algorithm of Chapter 3 for the total exchange in the d-cube was the first ever published to accomplish the task in the minimum possible completion time, namely in 2^{d-1} time units; note that this result was first presented in [BOSTT91]. Other total exchange algorithms were previously given in [SaS85] and in [JoH89]; however, in both these articles it is assumed that M different packets are transmitted per origin-destination pair. The corresponding lower bound for the completion time is $M2^{d-1}$ and it is met *exactly* by our algorithm, by treating all M packets per origin-destination pair as one and rescaling the slot duration by M. On the other hand, the algorithm of [SaS85] does not attain this lower bound, unless M is a *multiple* of d; for $M = 1$, which corresponds to our model for communications, the algorithm of [SaS85] takes time $d2^{d-1}$. Regarding the total exchange algorithms of [JoH89], most of them complete in time $M2^{d-1}$ *only* when M is a multiple of d, or when d is a *prime* number; for $M = 1$, all of the algorithms in [JoH89] complete in more than 2^{d-1} time units, except for one of the algorithms, which attains this lower bound only

when d is prime. Following the derivation of our total exchange algorithm, Varvarigos [Var90] and Edelman [Ede91], each taking a different approach from ours, devised other optimal such algorithms.

In Chapter 4, we consider the communication task of K *simultaneous broadcasts*, where K of the nodes of the d-cube wish to broadcast a packet at the same time. Note that the single node broadcast task (resp. the multinode broadcast) is a special case of this task with $K = 1$ (resp. $K = 2^d$). Regarding this problem, we first prove that the completion time of *any* algorithm for this task is at least $\max\{d, (\frac{2^d-1}{2^d})\frac{K}{d}\}$ time units. Then, we develop an algorithm that attains this lower bound within a *small* multiplicative constant factor. This algorithm applies to *any* K and to *any* K-tuple of broadcasting nodes; in fact, neither the subset of broadcasting nodes nor K has to be known prior to performing the task. The algorithm is simple to implement in a *distributed* fashion. During its initial phase, the broadcasting nodes achieve some form of *cooperation*. We also argue that some kind of cooperation is actually *necessary* for an algorithm to complete the task in time $\Theta(\max\{d, \frac{K}{d}\})$ for *all* sets of broadcasting nodes, unless if randomization is used. Finally, we present some algorithms pertaining to cases where K has one of special values.

Motivation for analyzing the aforementioned problem is as follows: Consider again the distributed version of the iterative algorithm $x := f(x)$. If all nodes are perfectly synchronized and the iterations are implemented in a *Jacobi* fashion, then all entries of the vector x are updated at the same time; thus, a multinode broadcast arises at the end of each iteration. However, even in synchronous environments, there are cases where *not* all of the x_i's are updated at the same time; e.g., in *Gauss-Seidel* algorithms, or when the iterations are implemented in a *multigrid* fashion, where some variables are updated more frequently than others. It is in such cases that a simultaneous broadcast by a *subset* of nodes arises.

The efficient algorithm of Chapter 4 for the task of K simultaneous broadcasts in the d-cube is new. To the best of our knowledge, the problem had not been analyzed in the previous literature, except of course for the special cases $K = 1$ and $K = 2^d$, which correspond to a single node broadcast and a multinode broadcast respectively. When specialized to the multinode broadcast task (namely, for $K = 2^d$), our algorithm completes in at most twice the optimal time, but it is much simpler to implement

than the optimal multinode broadcast algorithm of [BOSTT91] or the corresponding algorithms of [SaS85] and [JoH89]. The problem of K simultaneous broadcasts in the d-cube was considered recently by Varvarigos and Bertsekas [VaB91], who used a different approach and derived an algorithm that (in general) is faster than ours.

1.5.2 Dynamic Routing Problems

In Chapters 5, 6 and 7, we turn our attention to dynamic routing problems, analyzed in the context of the hypercube. The analysis of these problems constitutes the main focus of our research.

We begin with the problem of *multiple node-to-node communications* over an infinite time-horizon, which is analyzed in Chapter 5. In particular, we assume that each hypercube node generates packets according to a *Poisson* process, independently of the other nodes. Each packet has a *single destination*, which is selected *randomly* according to a certain rule; for a special case of this rule, the destination distribution is *uniform*. We analyze the problem in *steady-state*; first, we derive the *necessary condition for stability*, valid for any legitimate routing scheme. We also derive two *lower* bounds for the average *delay* per packet; one of them applies to all legitimate routing schemes, while the other applies to all schemes of a certain class. We then analyze the performance of a *greedy* routing scheme, where each packet chooses a particular path leading to its destination and attempts to traverse each arc of this path as fast as possible. This scheme is *distributed* and extremely simple to implement; regarding its performance, we prove that the scheme is very *efficient*. In particular, its stability properties are optimal; moreover, for any fixed traffic level (that maintains stability), the average delay per packet is $\Theta(d)$, namely of the same order of magnitude as in the case of *zero* traffic; finally, for any fixed dimensionality d, the rate of increase of the average delay in *heavy* traffic is also optimal. We also extend the results to the context of the *butterfly* network, where *greedy* routing is the most reasonable scheme to apply. Finally, we discuss some open problems related to the analysis of adaptive routing schemes, where a packet's path is influenced by contention. One such scheme has very satisfactory performance in practice, but seems to be analytically intractable.

The problem of multiple node-to-node communications in the hypercube (over an infinite time-horizon) was considered in several articles, which we discuss below; all of them deal with the case of uniform destination distribution. Abraham and Padmanab-

han [AbP86] have constructed an approximate model for this problem, under various assumptions on the buffer capacity of the nodes. In particular, they assume that packets advance in their respective paths independently of each other; the model involves some parameters, which are determined by solving a system of nonlinear equations. Greenberg and Hajek [GrH89] have analyzed this problem under *deflection* routing, where packets encountering contention are temporarily misrouted, rather than stored or dropped. The analysis in [GrH89] is approximate too. Varvarigos [Var90] has formulated a Markov chain model for evaluating the performance of deflection routing, and has investigated its steady-state statistics numerically.

The problem of multiple node-to-node communications has been analyzed in the context of the 2-dimensional array by Greenberg and Goodman [GrG86], with their analysis being again approximate. Recently, Leighton [Lei90] analyzed the problem for the same network and proved that a simple greedy routing scheme has very satisfactory average performance. Similar problems were also analyzed by Mitra and Cieslak [MiC87], as well as by Hajek and Cruz [HaC87], in the context of the extended Omega network, which is a crossbar switch. In contrast with all the other articles, where packets were taken to have fixed length, it was assumed in both [HaC87] and [MiC87] that, for each individual packet, transmission times over the various arcs are independent and exponentially distributed random variables. This assumption is called "Kleinrock's independence approximation" and simplifies the analysis considerably; even though it constitutes a reasonable approximation for several problems on data networks (see [BeG87]), it is rather unrealistic for problems arising in parallel computers. Finally, Bouras et al. [BGST87] considered the problem of multiple node-to-node communications in the context of Banyan networks, which are also crossbar switches; these authors investigate the performance of greedy routing, under the assumption of fixed packet lengths; however, the analysis in [BGST87] appears to be incorrect.

To the best of our knowledge, all of our results established in Chapter 5 are new, except for the necessary condition for stability, which is straightforward. Moreover, our analysis provides the first proof that some routing scheme for performing multiple node-to-node communications over an infinite time-horizon (on either the d-cube or the butterfly) has *both* optimal stability properties and an average delay of $\Theta(d)$ per packet. In fact, proving that *greedy* routing has these properties has been an important open question in the routing literature. The routing scheme considered in Chapter 5

is the first one in the hypercube to be analyzed rigorously. Excluding the results in [HaC87] and [MiC87], which are based on Kleinrock's independence approximation, the only rigorous performance analysis of a routing scheme is that in [Lei90], in the context of the 2-dimensional array.

Comparing the results of Chapter 5 to those of [Lei90], it is worth noting that ours are more *explicit*. To the best of our knowledge, our greedy routing scheme is the first for which the bounds on the delay are expressed by simple formulae involving only the rate of generating packets and the dimensionality d of the network (without adopting the independence approximation). This is due to the *new approach* followed in the derivation of our results: First, it is established that the hypercube behaves as a queueing network with deterministic servers (each corresponding to an arc) and with Markovian routing among the various servers; then, by using sample-path arguments, it is proved that the total number of packets present in this queueing network is dominated by that corresponding to a product-form network. The approach followed in [Lei90] involves combinatorial arguments, which are based on the following idea: A packet that suffers large delay should have collided with an untypically large number of other packets; by careful enumeration of such "bad" scenarios, it is then proved that the probability for a packet to be delayed excessively is small. It should be noted that our approach relies on the assumption of Poisson packet-generating processes. (Recall, however, that there are very few tractable queueing systems that do not involve Poisson arrivals.) Nevertheless, we hope that our analysis will be suggestive of the efficient performance of greedy routing under more general packet-generating processes.

In the course of the analysis of Chapter 5, we establish an interim proposition that is rather generally applicable. This enables the extension of our results on the hypercube to all crossbar switch networks with arcs arranged in *levels*, such as the Omega and the Banyan networks, and the butterfly (which is actually considered in Chapter 5); in fact, such an extension is possible even if each arc involves a different (yet fixed) transmission time per packet.

In Chapters 6 and 7, we analyze a problem where *multiple broadcasts* are performed over an infinite time-horizon, again in the hypercube network. Packets are generated according to the model adopted in Chapter 5; however, each packet is now to be *broadcast*, rather than sent to a single node. Again, we are interested in devising

routing schemes that make efficient use of the available resources, while introducing a small amount of *delay*. We first derive a *necessary condition for stability* as well as two *lower* bounds for the average *delay* per packet; one of them applies to all legitimate routing schemes, while the other applies to all schemes of a certain class. We then introduce two *distributed* routing schemes that prove to be rather efficient with respect to their stability properties. Though both schemes are simple in principle, their delay properties seem to be analytically intractable; thus, at this point, we resort to simulation and/or approximate analysis. The corresponding numerical results suggest very satisfactory performance of both schemes. We then present another *distributed* scheme, which is somewhat more complicated but still simple to implement. For both the stability and the delay properties of the scheme, we prove that they are rather satisfactory. The aforementioned problem, as well as the results derived in Chapter 6 and 7 are new. In the analysis of Chapter 7, we derive an interim result on the delay induced by a tree of paths; this result seems to applicable to other routing problems too.

Motivation for analyzing the two dynamic routing problems of Chapters 5 and 6-7 is as follows: As already mentioned, dynamic routing problems pertain to *asynchronous* environments, where the underlying assumptions of static routing may be far from reality. Such an example was already given in §1.2.3, regarding the iteration $x := f(x)$. The problem of Chapters 6 and 7, where packets to be broadcast are generated at random time instants, is closely related to this situation, provided that the function $f(x)$ is *dense*. On the other hand, the problem of multiple node-to-node communications seems to match better a situation where each node actually corresponds to several processors and computations on several *sparse* problems are performed at the same time. These two problems are simple (and extreme) cases of the routing problems arising in *general purpose* computation. In this general context, it may occur that packets received by a node influence the generation of subsequent packets as well as their length; moreover, it may even occur that each packet is destined for a different *subset* of nodes, which is possibly determined by the packets previously received. In fact, in such chaotic situations, the statistics of the packet generating processes may be unknown or ill-defined. The analysis of the two dynamic problems of Chapters 5 and 6-7 may be viewed as a first step towards treating such general problems.

As already mentioned, for both dynamic problems analyzed we derive *universal*

lower bounds for the average delay per packet; namely, lower bounds applying to the delay induced by *any* legitimate routing scheme. This concept is new and helps in better understanding the limitations imposed on the performance of all routing schemes. By taking these limitations into account, more realistic performance objectives may be set; also, the performance of the schemes analyzed may be better evaluated.

1.6 SUMMARY OF THE CONTRIBUTION OF THIS RESEARCH

The contribution of our research was discussed in detail in §1.5. In summary, the main new results contained in this Ph.D. Thesis are as follows:

(a) The first exactly optimal algorithm for the total exchange task in the hypercube.

(b) An efficient algorithm for the communication task of K simultaneous broadcasts in the hypercube, applying to all possible subsets of broadcasting nodes.

(c) The first rigorous proof of the efficiency of a greedy scheme for routing multiple node-to-node communications (over an infinite time-horizon) in the hypercube or the butterfly.

(d) The first analysis of a dynamic routing problem involving broadcasts in the hypercube arising over an infinite time-horizon.

2. Background Material

In this chapter, we present some background material on the hypercube network and on discrete-time queueing systems.

2.1 THE HYPERCUBE NETWORK

2.1.1 Definition

Most of our research is related to the d-dimensional *binary hypercube* (to be referred to as d-dimensional hypercube or simply d-cube). This network consists of 2^d nodes, numbered from 0 to $2^d - 1$; e.g., see [BeT89]. Associated with each node z is a binary identity $(z_d, \ldots, z_1)$, which coincides with the binary representation of the number z. Each *arc* of the hypercube connects two nodes whose binary identities differ in a *single* bit. That is, arc (z, y) exists if and only if $z_i = y_i$ for $i \neq m$ and $z_m \neq y_m$ (or equivalently $|z - y| = 2^{m-1}$) for some $m \in \{1, \ldots, d\}$. Note that (z, y) stands for a *unidirectional* arc pointing from z to y; of course, if arc (z, y) exists, so does arc (y, z). It is easily seen that the d-cube has $d2^d$ arcs. In Figure 2.1, we present the 3-dimensional hypercube.

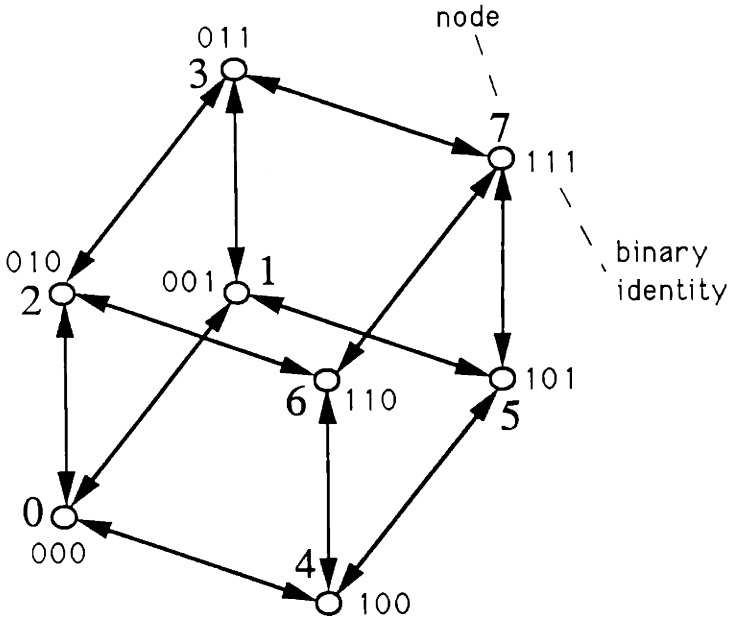

Figure 2.1: The 3-dimensional hypercube.

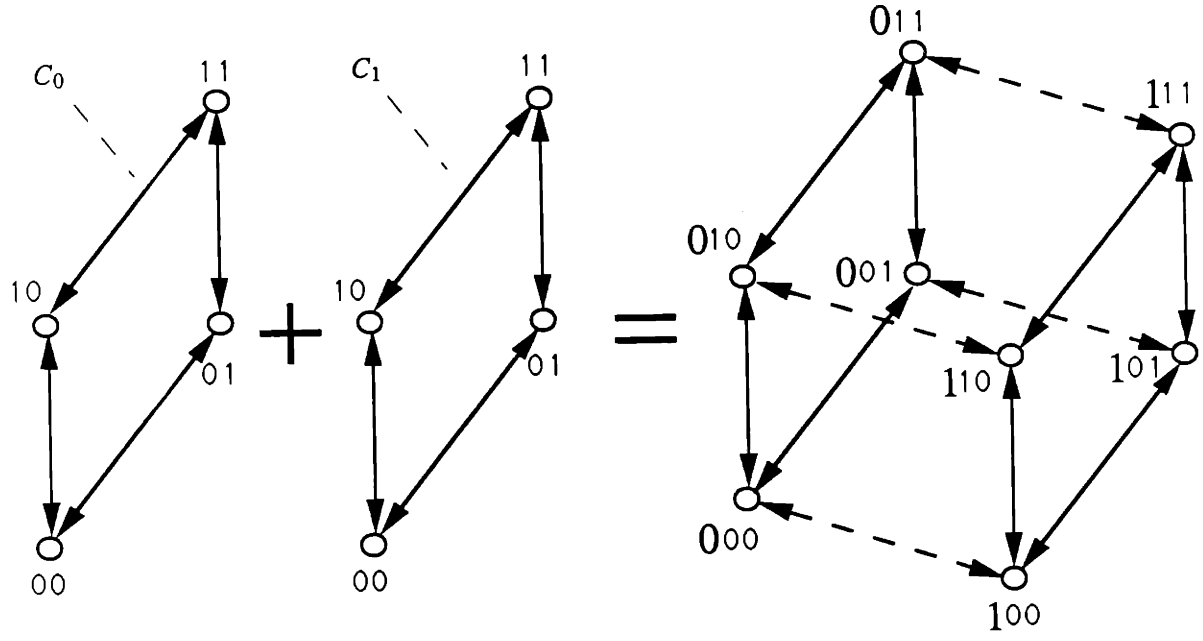

Figure 2.2: Recursive construction of the 3-dimensional hypercube.

A d-cube can be constructed *recursively*, by connecting two $(d-1)$-cubes. Indeed, it suffices to start with two such hypercubes C_0 and C_1, and then introduce two unidirectional arcs between each pair of nodes with the same binary identity. The resulting network is a d-cube; the binary identity of each node is obtained by adding a leading 0 or 1 to its previous identity, depending on whether the node belongs to C_0 or to C_1. The recursive construction of the 3-cube is depicted in Figure 2.2.

2.1.2 Hypercube Dimensions — Hamming Distance — Paths

Let there be two nodes z and y of the d-cube. We denote by $z \oplus y$ the vector $(z_d \oplus y_d, \ldots, z_1 \oplus y_1)$, where $\oplus$ is the symbol for the XOR operation. The ith (from the right) entry of $z \oplus y$ equals 1 if and only if $z_i \neq y_i$. Moreover, for $j \in \{1, \ldots, d\}$, we denote by e_j the node numbered 2^{j-1}; that is, all entries of the binary identity of e_j equal 0 except for the jth one (from the right), which equals 1. Nodes $e_1, \ldots, e_d$ are the only neighbors of node $(0, \ldots, 0)$. In general, each node z has exactly d neighbors, namely nodes $z \oplus e_1, \ldots, z \oplus e_d$. Clearly, arc (z, y) exists if and only if $z \oplus y = e_m$ for some $m \in \{1, \ldots, d\}$; such an arc is said to be of the mth *type*. The set of arcs of the mth type is called the mth *dimension*.

The *Hamming distance* between two nodes z and y is defined as the number of bits in which their binary identities *differ*; this equals the number of unity entries of $z \oplus y$, and is to be denoted as $H(z, y)$. Clearly, if the ith entry of $z \oplus y$ is equal to 1, then any *path* from z to y contains at least one arc of the ith *type*; for otherwise, the end node x of the path is such that $x_i = z_i$, and thus $x \neq y$. Hence, each path from z to y contains *at least* as many arcs as the Hamming distance between the two nodes. Moreover, there exist paths containing *exactly* this many arcs. Each such path is called a *shortest* path from z to y, for obvious reasons. It is straightforward that the *diameter* of the d-cube equals d.

For two nodes z and y of the d-cube, let $H(z, y) = k$ and let $i_1, \cdots, i_k$ be the entries of $z \oplus y$ that equal 1. Any *shortest* path from z to y contains k arcs, one arc of each of the types $i_1, \ldots, i_k$. There exist $k!$ different such paths from z to y, each corresponding to a different *order* of crossing the hypercube dimensions $i_1, \ldots, i_k$. The path in which the dimensions are crossed in *increasing index-order* is called the *canonical* path from z to y; e.g., for $z = (0, 0, 0, 1)$ and $y = (1, 0, 1, 0)$ in the 4-cube, the corresponding

canonical path is as follows:

$$(0,0,0,1) \rightarrow (0,0,0,0) \rightarrow (0,0,1,0) \rightarrow (1,0,1,0)\,.$$

2.1.3 The Completely Unbalanced Spanning Tree

The completely unbalanced spanning tree rooted at some node z is defined as the spanning tree $\mathcal{T}$ with the following property: Every node y is reached from the root z through the *canonical* path from z to y. Note that all spanning trees considered throughout our analysis are *directed*, since all hypercube arcs are taken unidirectional.

One can easily see that the collection of all canonical paths originating at node z constitute a tree $\mathcal{T}$ rooted at z. Indeed, it suffices to check that the following is true: If node y' belongs to the canonical path from z to y, then the part of this path that ends at y' is the canonical path from z to y'. A completely unbalanced spanning tree of the 3-cube is presented in Figure 2.3; the root of that tree is node $(0,0,0)$.

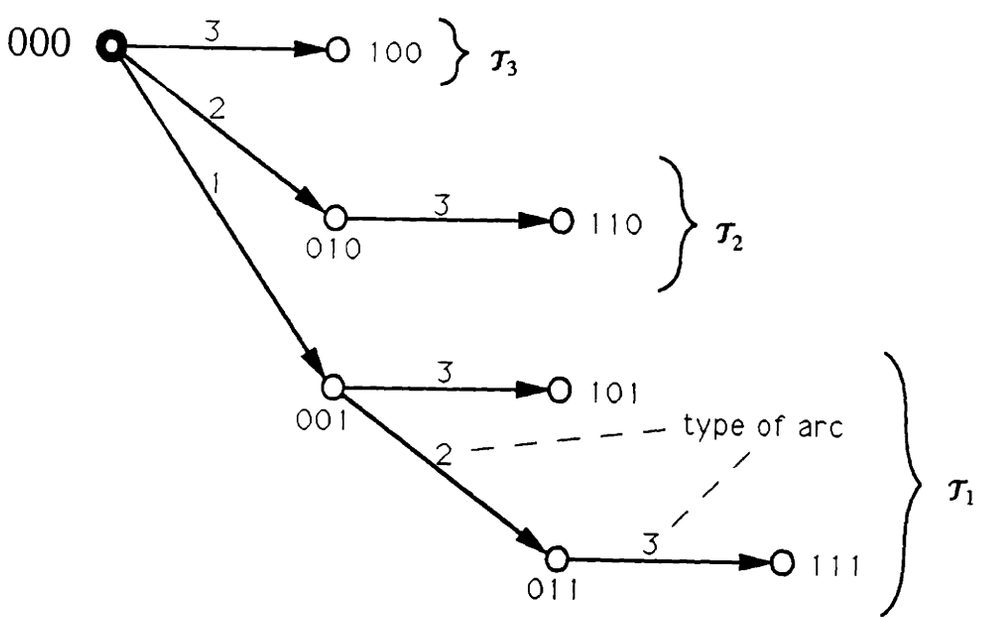

Figure 2.3: A completely unbalanced spanning tree of the 3-cube.

The completely unbalanced spanning tree $\mathcal{T}$ rooted at node z has d subtrees, which are denoted as $\mathcal{T}_1, \ldots, \mathcal{T}_d$. In particular, subtree $\mathcal{T}_i$ is hanging from node $z \oplus e_i$ and it comprises all nodes y with the following property: $y_1 = z_1, \ldots, y_{i-1} = z_{i-1}$ and $y_i \neq z_i$ (see Figure 2.3). Therefore, subtree $\mathcal{T}_1$ contains 2^{d-1} nodes, subtree $\mathcal{T}_2$ contains 2^{d-2} nodes etc, subtree $\mathcal{T}_d$ contains 1 node; hence the terminology "completely unbalanced". Another interesting property of a completely unbalanced spanning tree is that it has 2^{d-1} leaves.

Notice now that for any binary d-tuples v, w and y we have $(v \oplus y) \oplus (v \oplus w) = y \oplus w$. This implies that if (y, w) is an arc of the d-cube, then so does $(v \oplus y, v \oplus w)$ and vice versa; moreover, the two arcs are of the *same type*. Next, consider a spanning tree $\mathcal{S}$ rooted at some node z. The aforementioned property implies that the set of arcs $\{((x \oplus z) \oplus y, (x \oplus z) \oplus w) : (y, w) \in \mathcal{S}\}$ constitute a spanning tree $\mathcal{S}'$ rooted at node x. The tree $\mathcal{S}'$ is said to be a *translation* of the tree $\mathcal{S}$. Clearly, the two trees are *isomorphic*; that is, there exists a $1-1$ mapping $g : \{0, \ldots, 2^d - 1\} \rightarrow \{0, \ldots, 2^d - 1\}$ such that if (y, w) is an arc of $\mathcal{S}$ then $(g(y), g(w))$ is an arc of $\mathcal{S}'$; notice that $g(y) \stackrel{\text{def}}{=} y \oplus v$. It is straightforward to check that the completely unbalanced spanning tree rooted at a node z can be obtained by translating the one rooted at node $(0, \ldots, 0)$.

Consider now a spanning tree with the following property: Each node y is reached from the root z through that *shortest* path in which the hypercube dimensions are crossed in the order $(2, 3, \ldots, d, 1)$. It is easily seen that this tree is *isomorphic* to the completely unbalanced spanning tree rooted at z. By considering different orders for crossing the hypercube dimensions, we obtain other mutually isomorphic trees. Henceforth, we call *all* of these trees completely unbalanced, as well.

Completely unbalanced spanning trees have been used extensively in algorithms for hypercube communications; see [SaS85], [JoH89] and [BOSTT91]. (Johnsson and Ho [JoH89] use the terminology "spanning binomial tree" instead.) Apart from being convenient to deal with, these trees can be described in a very *concise* way, which reduces significantly the *memory requirements* of the algorithms using them; in particular, for each completely unbalanced spanning tree, it suffices to specify the root and the order of crossing the hypercube dimensions.

2.1.4 The d Disjoint Spanning Trees

Johnsson and Ho [JoH89] have constructed an imbedding of d *arc-disjoint* spanning trees in the d-cube; they call them "d Edge-Disjoint Spanning Binomial Trees" (dESBT). This imbedding consists of d completely unbalanced spanning trees, to be denoted as $\mathcal{T}^{(1)}, \ldots, \mathcal{T}^{(d)}$. Tree $\mathcal{T}^{(j)}$ is rooted at node e_j; the *order* of crossing the hypercube dimensions in the paths of $\mathcal{T}^{(j)}$ is as follows:

$$(j \bmod d) + 1, [(j+1) \bmod d] + 1, \ldots, [(j+d-1) \bmod d] + 1 \, .$$

In Figure 2.4, we present this construction for $d = 3$; this figure is taken from [JoH89].

The d disjoint spanning trees comprise a total of $d(2^d - 1)$ hypercube arcs, since each of them consists of $2^d - 1$ different arcs. There are d arcs that do not belong in any of these trees; these are the d arcs outgoing from node $(0, \ldots, 0)$, namely $(0, e_1), \ldots, (0, e_d)$.

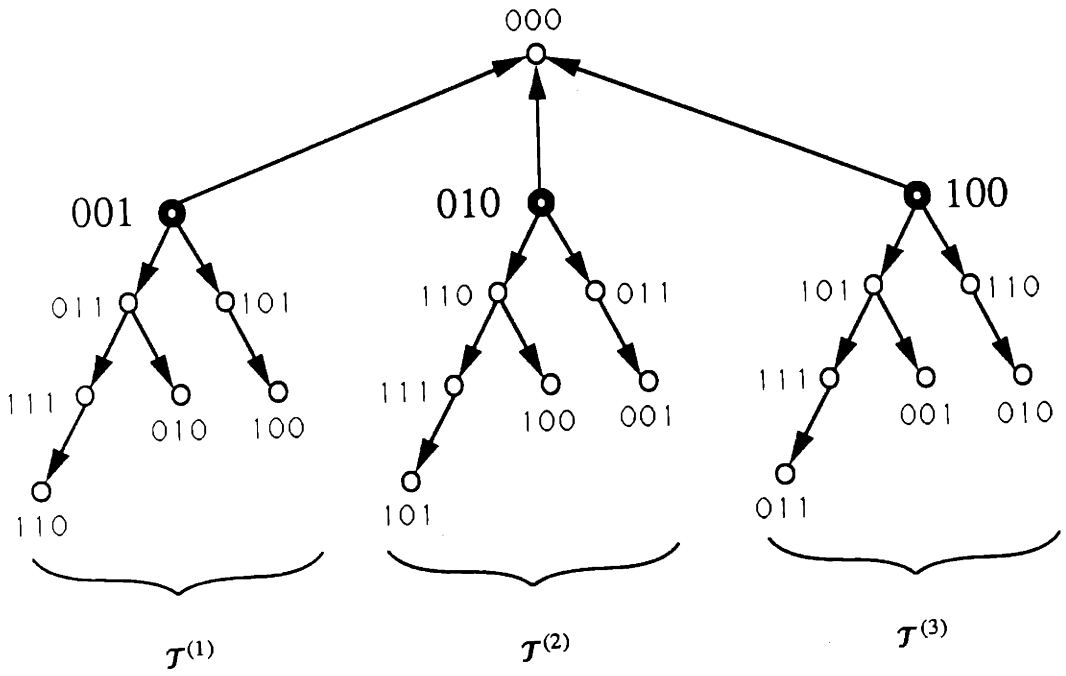

Figure 2.4: The d disjoint spanning trees, for $d = 3$.

2.2 DISCRETE-TIME QUEUES

2.2.1 The Discrete-Time $G/D/1$ Queue With Unit Service Time

In this subsection, we briefly present some properties of the discrete-time $G/D/1$ queue with unit service time; this is a single server queueing system operating as follows:

(a) The time axis is divided into slots of unit duration.

(b) The number of arrivals A_k during slot k is distributed as the random variable A, for $k = 0, \ldots$; thus, we have $E[z^{A_k}] = E[z^A] \stackrel{\text{def}}{=} G_A(z)$. Arrivals within different slots are mutually independent. Customers arrive in the system just before the end of each slot.

(c) Each customer's service time equals one slot.

We denote by Y_k the number of customers present in the system at the beginning of the kth slot, including the customer in service (if any). It follows from the above discussion that

$$Y_{k+1} = [Y_k - 1]^+ + A_k \,, \quad \text{for } k = 0, \ldots \,, \tag{2.1}$$

where $[\alpha]^+ \stackrel{\text{def}}{=} \max\{0, \alpha\}$. The queueing system considered is *stable* if and only if either $E[A] < 1$ or $\Pr[A = 1] = 1$. By the term "stability" it is meant that, as $n \to \infty$, the distribution of the total time spent by the nth customer in the system converges to a *proper* distribution, which is independent of the initial condition (namely, of the value of Y_0); note that a proper distribution corresponds to a random variable that is finite with probability 1. Henceforth, we assume $E[A] < 1$; in this case, the steady-state distribution of Y_k also is well-defined, and can be derived by using standard tools of queueing theory. In particular, we have [KoK77]

$$G_Y(z) \stackrel{\text{def}}{=} \lim_{k \to \infty} E[z^{Y_k}] = (1 - E[A]) \frac{z - 1}{z - G_A(z)} G_A(z) \,, \quad \text{for } |z| \leq 1 \,. \tag{2.2}$$

Using the fact $\Pr[Y = 0] = \lim_{z \to 0} G_Y(z)$, it follows that

$$\lim_{k \to \infty} \Pr[Y_k \neq 0] = E[A] \,.$$

The expected number N of customers in the system (in steady-state) can be calculated by evaluating $\frac{dG_Y(z)}{dz}|_{z=1}$; thus, it follows from (2.2) that

$$N \stackrel{\text{def}}{=} \lim_{k \to \infty} E[Y_k] = E[A] + \frac{E[A(A-1)]}{2(1 - E[A])} \,.$$

Using Little's formula, we have

$$T = \frac{N}{E[A]} = 1 + \frac{E[A(A-1)]}{2E[A](1-E[A])}\,, \tag{2.3}$$

where T is the steady-state average *delay* per customer, the service time *included.*

2.2.2 The Discrete-Time $M/D/1$ Queue With Synchronization

We consider a discrete-time $G/D/1$ queue with unit service time and with the number of arrivals per slot assuming the Poisson distribution. In this case, we have $E[A(A-1)] = E^2[A]$; thus, (2.3) gives

$$T = 1 + \frac{E[A]}{2(1-E[A])}\,. \tag{2.4}$$

So far, we have assumed that arrivals may occur only at the end of each slot. If we assume instead that the arrival instants are points of a *continuous-time* Poisson process with rate $\lambda < 1$, then the analysis leading to (2.2) still holds. (Again, service may only start at the beginning of a slot.) However, a correction in the expression for the average delay T is necessary. In particular, we have to add to the right-hand quantity in (2.4) the expected duration of the time elapsing between the arrival instant of some customer and the beginning of the next slot. This quantity is called the average *synchronization time* and it equals $\frac{1}{2}$. Thus, we obtain

$$T = \frac{3}{2} + \frac{E[A]}{2(1-E[A])} = \frac{3}{2} + \frac{\lambda}{2(1-\lambda)}\,. \tag{2.5}$$

Finally, consider the queueing system described above, but modified as follows: Service may only start every Δ slots, say in the beginning of the slots numbered $0, \Delta, \ldots$; moreover, the service time duration is now set to Δ. We will be referring to this system as the discrete-time $M/D/1$ queue with *synchronization.* The number of arrivals during Δ successive slots is Poisson, with rate $\lambda\Delta$. Moreover, the arrivals in different intervals of duration Δ are mutually independent. Thus, in order to derive the expression for the steady-state average delay T, we simply have to apply (2.5) with $\lambda\Delta$ instead of λ and rescale time by Δ; thus, we obtain

$$T = \Delta\left[\frac{3}{2} + \frac{\lambda\Delta}{2(1-\lambda\Delta)}\right]. \tag{2.6}$$

Clearly, the discrete-time $M/D/1$ queue with synchronization is *stable* if and only if $\lambda\Delta < 1$.

3. Total Exchange in the Hypercube

3.1 INTRODUCTION

The total exchange task is among the prototype communication scenarios arising in parallel computation. In the course of this task, each node sends a *different* packet to each other node. For example, this situation arises in the transposition of a matrix stored by row; see §1.2.1. In this chapter, we first prove a *lower* bound on the time required for *any* algorithm to perform a total exchange in the d-dimensional hypercube. We then prove the *existence* of an algorithm that attains this lower bound *exactly*; finally, we present a rather simple *implementation* of the optimal algorithm.

The algorithm derived in this chapter was the first ever published to attain the minimum possible time for the total exchange task in the d-cube; the result appears in [BOSTT91]. Among the previously existing algorithms, the fastest was one derived by Johnsson and Ho [JoH89], which however is exactly optimal only when d is a *prime* number. In fact, the algorithms by Saad and Schultz [SaS85] and the ones of [JoH89] were derived under the assumption that M packets are exchanged per origin-destination pair; these algorithms attain exactly the corresponding optimal time only for M being a *multiple* of d or for prime d (see also §1.4.1). Even though we are

here considering the case $M = 1$, the algorithm to be presented is optimal for any value of M; indeed, it suffices to *rescale* the slot duration by M, and treat all M packets corresponding to an origin-destination pair as one. Following the derivation of our total exchange algorithm, Varvarigos [Var90] and Edelman [Ede91] devised other optimal such algorithms, by using a different approach.

3.2 THE LOWER BOUND

For any node x of the d-cube, there exist $\binom{d}{k}$ nodes y at Hamming distance k from x, for all $k \in \{1, \ldots, d\}$. Each packet transmitted by x and destined for such a node y has to undertake at least k transmissions. Furthermore, for an *arbitrary* total exchange algorithm for the d-cube, let $R(x)$ denote the total number of transmissions of packets *originating* at node x, where both the initial and intermediate transmissions (if any) are included. It follows from the above discussion that

$$R(x) \geq \sum_{k=1}^{d} k \binom{d}{k} = d2^{d-1}\,, \quad \text{for } x = 0, \ldots, 2^d - 1\,. \tag{3.1}$$

Since the d-cube contains $d2^d$ unidirectional arcs, at most $d2^d$ transmissions may take place during each time unit. Therefore, the completion time T_d of the arbitrary algorithm considered satisfies

$$T_d \geq \frac{1}{d2^d} \sum_{x=0}^{2^d-1} R(x)\,. \tag{3.2}$$

This together with (3.1) implies that

$$T_d \geq 2^{d-1}\,. \tag{3.3}$$

Hence, any total exchange algorithm in the d-cube takes at least 2^{d-1} time units. Moreover, for an algorithm to attain this lower bound, it is necessary that both inequalities (3.1) and (3.2) hold with equality; thus, packets should follow *shortest* paths and all arcs should be *busy* (in both directions) during all of the 2^{d-1} time units. In what follows, we present an algorithm for which $T_d = 2^{d-1}$. In light of the above, this algorithm is *optimal* (i.e., unimprovable) with respect to both the time and the number of transmissions criteria and achieves 100% arc utilization.

3.3 DERIVATION OF THE OPTIMAL ALGORITHM

Following a methodology similar to that in [BeT89], we construct the optimal algorithm *recursively*. In particular, we shall assume that we have an optimal algorithm for total exchange in the d-cube with certain properties to be stated later, and we shall use this algorithm in order to perform an optimal total exchange in the $(d+1)$-cube.

First, let us assume that we have an algorithm for total exchange in the d-cube, not necessarily of the optimal duration; then, we can construct a total exchange algorithm for the $(d+1)$-cube as follows: We decompose the $(d+1)$-cube into two d-cubes, denoted as C_0 and C_1. Without loss of generality we assume that C_0 contains nodes $0, \ldots, 2^d - 1$ and that their counterparts in C_1 are nodes $2^d, \ldots, 2^{d+1} - 1$, respectively; see also §2.1.1. The algorithm for the $(d+1)$-cube consists of three phases. In phase 1, there is a total exchange within each of the sub-cubes C_0 and C_1, using the available algorithm for the d-cube; each node in C_0 exchanges its packets with the other nodes in C_0 and similarly for sub-cube C_1. In phase 2, each node transmits to its counterpart node in the opposite d-cube *all* of the 2^d packets that are destined for the nodes of the opposite d-cube. In phase 3, again there is a total exchange in each of the two d-cubes; however, the packets exchanged in this phase are the ones *received in phase 2*. The recursive construction of the algorithm for the the total exchange in the $(d+1)$-cube is illustrated in Figure 3.1, which is taken from [BeT89].

The simplest way to accomplish the total exchange in the $(d+1)$-cube is to perform the three phases sequentially; letting $\tilde{T}_d$ denote the completion time for the d-cube algorithm, we thus obtain the following recursion:

$$\tilde{T}_{d+1} = \tilde{T}_d + 2^d + \tilde{T}_d \,,$$

where we have used the fact that phase 2 lasts for 2^d time units. (Notice that each node has to send 2^d packets in phase 2, one packet per time unit.) Assuming now that the algorithms for the d-cube, the $(d-1)$-cube etc, the 2-cube were *all* derived by the *same* recursive construction (with $\tilde{T}_1 = 1$), we obtain $\tilde{T}_d = d2^{d-1}$; this exceeds the lower bound of (3.3) by a factor of d. Notice, however, that phases 1 and 2 can be performed *simultaneously*; indeed, these two phases involve *disjoint* sets of packets and use *disjoint* sets of arcs. By performing phase 3 *after* the end of phases 1 and 2, we obtain

$$\bar{T}_{d+1} = \max\{\bar{T}_d, 2^d\} + \bar{T}_d \,;$$

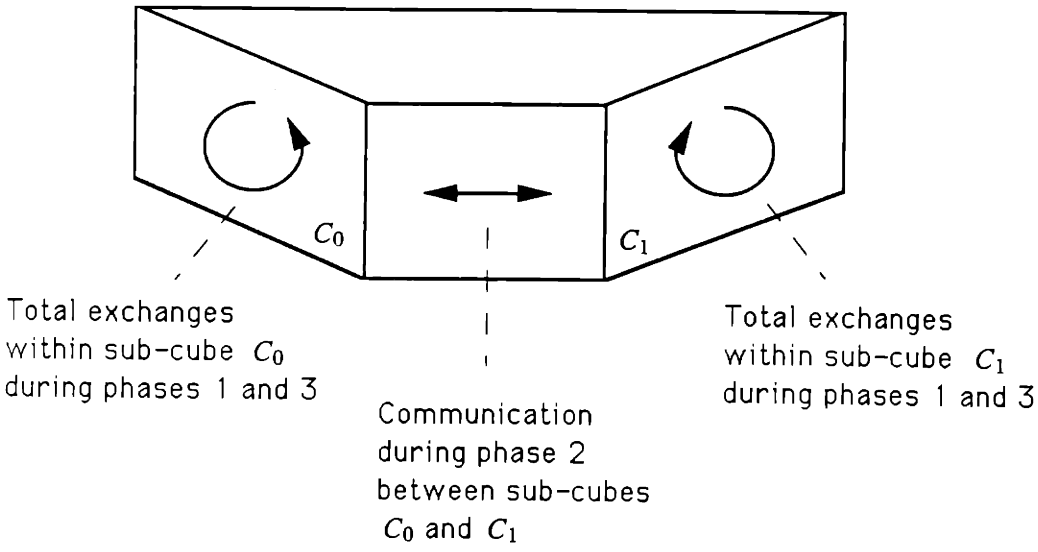

Figure 3.1: Recursive construction of the algorithm
for total exchange in the $(d+1)$-cube.

solving this recursion, we have $\bar{T}_d = 2^d - 1$. This algorithm is derived in [BeT89]; its completion time exceeds the lower bound of (3.3) by almost a factor of 2. Here, we improve on this time by allowing phase 3 to start *before* phase 2 ends. To illustrate how this is possible, consider the packet originating at some node $x \in C_0$ and destined for its counterpart node in C_1, namely $x + 2^d$, and the packet originating at $x + 2^d$ and destined for x. These packets are not transmitted at all during phase 3. Therefore, if they are transmitted last in phase 2, then phase 3 can start one time unit before the end of phase 2. This idea can be generalized as follows: Clearly, if it were guaranteed for all nodes $x \in C_0$ (and resp. for all nodes $z \in C_1$) that all packets originating at x and going to C_1 (and resp. originating at z and going to C_0) arrive *in time* for phase 3 at node $x+2^d$ (and resp. at node $z-2^d$), then phase 3 can be performed *just after* phase 1, *without* completing phase 2. In such a case, phase 3 would proceed *uninterrupted*; thus, the first half of phase 2 would be carried out simultaneously with phase 1, while the second half would be carried out simultaneously with phase 3 and we would have $T_{d+1} = 2T_d$. Assuming that the algorithms for the d-cube, the $(d-1)$-cube etc, the 2-cube were *all* obtained by the *same* recursive construction (with $T_1 = 1$), we now

have $T_d = 2^{d-1}$; hence, this family of recursively constructed algorithms would achieve the lower bound of (3.3). In what follows we prove that such a recursive construction is indeed feasible.

Suppose that the *optimal* total exchange algorithm has *already* been devised for the d-cube. Let $N_d(x,n)$ denote the number of its *own* packets that node x has transmitted up to and including time n, for $n = 1,\ldots,2^{d-1}$. [Note that $N_d(x,n)$ ranges from 1 to $2^d - 1$.] We can use $N_d(x,n)$ to express the requirement that phase 3 packets originating at nodes of C_0 are *available in time* at the appropriate node of C_1, so that phase 3 begins right after phase 1 and continues uninterrupted (with phase 2 still being in progress). In particular, it is necessary that

$$N_d(x,n) \leq 2^{d-1} + n - 1\ , \quad \text{for } n = 1,\ldots,2^{d-1} \text{ and } x = 0,\ldots,2^d - 1\ . \tag{3.4}$$

To see this, note that the left-hand quantity in (3.4) is the number of packets originating at node $x \in C_0$ that must be transmitted by node $x + 2^d$ during the first n time units of phase 3, while the right-hand quantity in (3.4) equals the number of available time units within phase 2 for transferring these packets from node x to node $x + 2^d$. (Recall that each node x sends its phase 2 packets one by one.) There is also a requirement analogous to (3.4) for the nodes of C_1. Since the existence of the optimal algorith for the d-cube was *assumed*, it is *not* known whether or not (3.4) holds. Thus, we shall *also* assume that this is indeed the case, and we shall prove that (3.4) holds for the algorithm to be constructed for the $(d+1)$-cube; the formal argument is presented next.

Proposition 3.1: For every d, there exists a total exchange algorithm for the d-cube, satisfying $T_d = 2^{d-1}$ and

$$N_d(x,n) \leq 2^{d-1} + n - 1\ , \quad \text{for } n = 1,\ldots,2^{d-1} \text{ and } x = 0,\ldots,2^d - 1\ . \tag{3.5}$$

Proof: We have $T_1 = 1$ and $N_1(i,1) = 1$, for $i = 0,1$, which proves the inductive hypothesis for $d = 1$. Furthermore, we assume that for some d we have a total exchange algorithm for the d-cube that satisfies both $T_d = 2^{d-1}$ and (3.5). Let $s(x,y,d)$ denote the time unit in this algorithm during which node x transmits its *own* packet that is destined for node y (with $x \neq y$). We shall construct a three-phase total exchange algorithm for the $(d+1)$-cube that satisfies the inductive hypothesis, namely both

$T_{d+1} = 2^d$ and (3.5). Suppose that packets are transmitted during phase 2 according to the following rules: (In view of the symmetry of the transmissions of nodes of the d-cubes C_0 and C_1, we describe the rules for packet transmissions of phase 2 only for the nodes of C_0.)

(a) Each node $x \in C_0$ transmits its packets to node $x + 2^d$ in the *order* in which the latter node forwards them in phase 3 (ties are broken arbitrarily); thus, the packet destined for node $y \in C_1$ (with $y \neq x+2^d$), is transmitted before the packet destined for $y' \in C_1$ (with $y' \neq x + 2^d$) only if

$$s(x, y - 2^d, d) \leq s(x, y' - 2^d, d) .$$

(b) Each node $x \in C_0$ transmits its packet destined for node $x + 2^d$ *last.*

We claim that under the above rules, phase 3 can proceed *uninterrupted* after phase 1. To show this, consider any node $x \in C_0$ (the case of $x \in C_1$ can be treated analogously). At the end of phase 1, node x has received exactly 2^{d-1} packets from node $x+2^d$, since phase 1 lasts for 2^{d-1} time units (by the inductive hypothesis). Hence, $n-1$ time units after the end of phase 1, node x has received exactly $2^{d-1} + n - 1$ packets from node $x + 2^d$. On the other hand, the total number of packets of node $x+2^d$ that x forwards during the first n time units of phase 3 is exactly $N_d(x, n)$. Since $N_d(x, n) \leq 2^{d-1} + n - 1$ for all $n = 1, \ldots, 2^{d-1}$ [by the assumed relation (3.5)], and node $x+2^d$ transmits its packets to node x according to the above rules, node x *always* has *enough* packets from node $x + 2^d$ for transmission if phase 3 begins immediately after phase 1. Since node $x \in C_0$ was chosen arbitrarily, this holds for all $x \in C_0$.

Consider the total exchange algorithm for the $(d+1)$-cube, whereby phase 3 proceeds uninterrupted immediately following phase 1 as described above. Since, according to the inductive hypothesis, each of phases 1 and 3 takes $T_d = 2^{d-1}$, this algorithm takes time $T_{d+1} = 2T_d = 2^d$, which is the desired duration. There remains to show that the second part of the inductive hypothesis [namely, (3.5)] is satisfied for $d + 1$. For any node x, let $N_{d+1}(x, n)$ denote the number of node x's own packets that x has transmitted up to and including time n of this algorithm. In the first 2^{d-1} time units of the algorithm, phases 1 and 2 execute simultaneously; since packets of phase 2 are transmitted by each node one by one, we have

$$N_{d+1}(x, n) = N_d(x, n) + n , \quad \text{for } n = 1, \ldots, 2^d .$$

Combining this inequality with the fact $N_d(x,n) \leq 2^d - 1$ (which holds for all n), we obtain

$$N_{d+1}(x,n) \leq 2^d + n - 1\ , \quad \text{for } n = 1,\ldots,2^{d-1}\ . \tag{3.6}$$

In the next $2^d - 1$ time units of this algorithm, phases 2 and 3 execute simultaneously; since node x does not transmit any packet of its own during phase 3, we have

$$N_{d+1}(x,n) = 2^d + n - 1\ , \quad \text{for } n = 2^{d-1} + 1,\ldots,2^d\ .$$

Combining this inequality with (3.6), we obtain

$$N_{d+1}(x,n) \leq 2^d + n - 1\ , \quad \text{for } n = 1,\ldots,2^d\ .$$

Since the choice of x was arbitrary, it follows that (3.5) holds for the $(d+1)$-cube as well, which completes the proof. **Q.E.D.**

3.4 IMPLEMENTATION OF THE OPTIMAL ALGORITHM

In this section, we present the rules used by the nodes for transmitting their own packets and forwarding the packets they receive from other nodes, whenever further transmission is required. These rules can be easily derived by "unfolding" the recursive construction of the optimal algorithm.

The presentation of these rules becomes more clear, by using the binary representation of node identities. Recall that e_k denotes the node numbered 2^{k-1}, for $k = 1,\ldots,d$ (see §2.1.2). Also, $\bar{x}_k$ denotes the reverse of bit x_k, namely $\bar{x}_k = (x_k + 1) \bmod 2$.

We first describe the order in which an arbitrary node x transmits its *own* packets. It follows easily from the discussion in §2.1.2 that each packet crosses the hypercube dimensions in *decreasing* index-order. Thus, a packet originating at node x and destined for node y follows the *reverse* of the canonical path from y to x (*). Furthermore, by the structure of the three phases presented in §2.1.2, it is seen that, for each arc $(x, x \oplus e_k)$, node x sends all of its *own* packets to cross this arc *before* forwarding

(*) The fact that the index-order of crossing the various dimensions is the decreasing is a consequence of the choice of C_0 and C_1 in the recursive construction of the algorithm; if C_0 and C_1 were chosen so that $C_0 = \{x : x_1 = 0\}$, then this index-order would be the increasing, and the resulting paths would be the canonical ones.

through $(x, x \oplus e_k)$ any packets received by other nodes. Therefore, at time instants $1, \ldots, 2^{k-1}$, node x transmits those of its own packets that are destined for nodes

$$(x_d, \ldots, x_{k+1}, \bar{x}_k, v_{k-1}, \ldots, v_1)\,, \text{ where } v_m = 0 \text{ or } 1 \text{ for } m = 1, \ldots, k-1\,.$$

All these packets are sent through arc $(x, x \oplus e_k)$; the last packet to be transmitted in this group is the one destined for node $x \oplus e_k$. (The aforementioned rules apply for $k = 1, \ldots, d$.) For $x = 0$, the exact *order* in which x transmits its packets on each of its outgoing arcs may be derived by using a sequence of d tables, which are constructed iteratively. The kth table consists of k columns, the mth of which contains the destinations of packets transmitted through arc $(0, e_m)$. The first table contains only e_1. For $k = 2, \ldots, d$, the first $k-1$ columns of the kth table are (obviously) identical to those of the $(k-1)$st table, whereas its last column contains the entries of the $(k-1)$st table with their kth bit being set to 1. In the last column, entries corresponding to the same row of the $(k-1)$st table appear one after the other, ordered (arbitrarily) from left to right; entries corresponding to different rows of the $(k-1)$st table are ordered from top to bottom. The last element of the kth column of the kth table is e_k. This scheme follows easily from the recursive construction of the optimal algorithm; for example, given the $(d-1)$st table, the way of constructing the dth column of the dth table follows from the assumed order of transmitting the packets of phase 2 (see also the proof of Proposition 3.1). In Figure 3.2, we present the construction of the d tables for the case $d = 4$. For any other node x, the corresponding order of destinations may be obtained by XOR-ing each entry of the dth table for node 0 with the binary identity of x. For practical applications, it seems preferable to construct these tables prior to the starting transmissions, rather than computing each row on-line. This is due to the fact that the entries of mth row of the kth table cannot be derived recursively by operating *only* on the $(m-1)$st row of the $(k-1)$st table.

Furthermore, we focus on the packets arriving in some node x and present the rules under which these packets are *forwarded* by x (whenever necessary). Packets arrive in x, through arc $(x \oplus e_k, x)$ in groups of 2^{k-1}, for $k = 1, \ldots, d$. Each group consists of all packets originating from the same node $(y_d, \ldots, y_{k+1}, \bar{x}_k, x_{k-1}, \ldots, x_1)$ (where $y_m = 0$ or 1, for $m = k+1, \ldots, d$) and destined for all nodes of the form $(x_d, \ldots, x_k, v_{k-1}, \ldots, v_1)$ (where $v_m = 0$ or 1, for $m = 1, \ldots, k-1$). [The origin and destination sets of the packets traversing arc $(x \oplus e_k, x)$ are implied by the fact that

Table 1	
Time	**Arc** $(0, e_1)$
1	0001

Table 2		
Time	**Arc** $(0, e_1)$	**Arc** $(0, e_2)$
1	0001	0011
2		0010

Table 3			
Time	**Arc** $(0, e_1)$	**Arc** $(0, e_2)$	**Arc** $(0, e_3)$
1	0001	0011	0101
2		0010	0111
3			0110
4			0100

Table 4				
Time	**Arc** $(0, e_1)$	**Arc** $(0, e_2)$	**Arc** $(0, e_3)$	**Arc** $(0, e_4)$
1	0001	0011	0101	1001
2		0010	0111	1011
3			0110	1101
4			0100	1010
5				1111
6				1110
7				1100
8				1000

Figure 3.2: Derivation of the order of packet transmissions for $d = 4$.

packets cross the various hypercube dimensions in decreasing index-order.] The order of group arrivals is *lexicographic* in $y \oplus x$ [or equivalently in $(y_d \oplus x_d, \ldots, y_{k+1} \oplus x_{k+1})$]. As an example, let us consider the case $x = 0$ and $k = d-1$. Each packet to be received through arc $(e_{d-1}, 0)$ originates from one of nodes e_{d-1}, e_d, and $e_d \oplus e_{d-1}$. (Note that these three nodes are given in lexicographic order.) In the recursive construction of the algorithm, the packets from node e_{d-1} belong to phase 1, while those from e_d and $e_d \oplus e_{d-1}$ belong to phase 3 and thus they are received later; it can also be seen that the packets from e_d are received by node 0 prior to those from $e_d \oplus e_{d-1}$, by considering the recursive structure of the optimal algorithm for the $(d-1)$-cube.

Routing of the packets to be forwarded by node x is accomplished as follows: A packet destined for node $(x_d, \ldots, x_k, v_{k-1}, \ldots, v_1)$ joins the queue of packets to be transmitted by x through arc $(x, x \oplus e_{k^*})$, where

$$k^* = \max_{1 \le m \le k-1} \{m : v_m = \bar{x}_m\} .$$

Packets originating from the same node and placed in the same queue preserve their order of arrival. Packets originating from different nodes y, z and placed in the same queue are ordered according to the lexicographic order between $y \oplus x$ and $z \oplus x$. (This can be seen by an example similar to that presented in the previous paragraph.) Forwarding packets through arc $(x, x \oplus e_k)$ starts at time $2^{k-1}+1$, for $k = 1, \ldots, d-1$; at each time, the packet at the *top* of the queue for arc $(x, x \oplus e_k)$ is forwarded through that arc. No forwarding takes place through arc $(x, x \oplus e_d)$.

The rules presented above follow from the recursive construction of the optimal algorithm. Our earlier analysis guarantees that packets are always in time at the intermediate nodes (if any) of the paths they have to traverse. Notice that the algorithm is implemented in a fully *distributed* fashion. Indeed, the travelling schedule of each packet is *locally* determined at the intermediate nodes of its path, by examining only the packet's origin and destination. Thus, packets do not have to carry any *timing* information, which makes the implementation rather efficient for practical applications. Another nice property of the optimal algorithm is that it does *not* require any additional *buffer capacity* to be implemented. Indeed, since all hypercube arcs are always busy, each node receives d packets and transmits d packets per slot; thus, the total number of packets stored at each node remains *constant* (namely, it equals $2^d - 1$) for the entire duration of the algorithm.

4. Multiple Simultaneous Broadcasts in the Hypercube

4.1 INTRODUCTION

During the execution of parallel algorithms, it is often necessary that several processors simultaneously broadcast packets to all others. The case where *all* processors wish to perform a broadcast at the same time corresponds to the multinode broadcast task; among others, this task arises in the synchronous distributed execution of Jacobi iterations; see §1.2.1. In this chapter, we consider a similar routing problem; in particular, we assume that each member of a *subset* of K processors wishes to broadcast a packet. This situation arises when a distributed iterative algorithm is implemented in a multigrid or Gauss-Seidel fashion; see also §1.4.1.

We analyze the problem defined above in the context of the d-dimensional hypercube. We first derive a *lower* bound for the time required to perform K simultaneous broadcasts, applying to *any* $K \in \{1, \ldots, 2^d\}$ and to *any* routing algorithm. We then develop (in §4.4) a simple *distributed* algorithm that meets this lower bound *within a small multiplicative constant factor*. This efficient algorithm works even if *no* node of

the hypercube knows the value of K or the identities of any other broadcasting nodes. The main idea of the algorithm is as follows: The K packets to be broadcast are *split evenly* among the d disjoint spanning trees $\mathcal{T}^{(1)}, \ldots, \mathcal{T}^{(d)}$ presented in §2.1.4; each packet is sent to the *root* of one of these d trees, which will eventually broadcast the packet along that same tree. The *even* splitting of the K packets among the d trees is accomplished by performing a *parallel prefix* task prior to starting transmissions; we briefly discuss this task in §4.3. When specialized to the multinode broadcast task (namely, for $K = 2^d$), our algorithm completes in at most twice the optimal time, but it is much *simpler* to implement than the multinode broadcast algorithms of [SaS85], [JoH89] and [BOSTT91]. In addition to the general algorithm of §4.4, we also present other efficient algorithms pertaining to cases where K is *known* to have one of certain values.

The problem formulated above as well as the results to be derived are new. Recently, the problem was considered by Varvarigos and Bertsekas [VaB91], who used a different approach and derived an algorithm that (in general) is faster than the one of §4.4.

Throughout the analysis, it will be assumed that the K broadcasting nodes are *distinct*, unless otherwise specified.

4.2 LOWER BOUNDS

Clearly, under *any* routing algorithm, K broadcasts involve a total of at least $(2^d - 1)K$ packet transmissions. Since at most $d2^d$ transmissions may be performed in each slot, this task requires at least $\frac{2^d-1}{d2^d}K = \Theta(\frac{K}{d})$ time units. Furthermore, the completion time of each broadcast is no less than the *diameter* d of the d-cube. Hence, it follows that under *any* routing algorithm and for *any* K-tuple of broadcasting nodes, the task considered takes time $\Omega(\max\{d, \frac{K}{d}\})$, and in particular at least $\max\{d, (\frac{2^d-1}{2^d})\frac{K}{d}\}$ time units. The notation $T = \Omega(\max\{d, \frac{K}{d}\})$ should be interpreted as "T is of larger order of magnitude than $\max\{d, \frac{K}{d}\}$"; that is, there exists a constant C such that $T \geq C \max\{d, \frac{K}{d}\}$ for all $K \leq 2^d$ and for $d = 1, \ldots$

We are interested in devising an algorithm that attains the optimal order of magnitude $\Theta(\max\{d, \frac{K}{d}\})$ of the completion time, for *any* $K \in \{1, \ldots, 2^d\}$ and for *any* K-tuple of broadcasting nodes. The simplest possible distributed algorithm for our task would be as follows: Each of the 2^d nodes of the hypercube is *confined* to broad-

cast its packet (if it has one) along a *prespecified* spanning tree. Unfortunately, such an algorithm would *not* always attain the optimal order of magnitude for the completion time. Indeed, for any fixed node x and for any of the 2^d prespecified trees except for the one rooted at x, there exists exactly one arc of the form $(x \oplus e_j, x)$ that belongs to the tree. Thus, there exists some arc $(x \oplus e_{j^*}, x)$ that belongs to at least $\frac{2^d-1}{d}$ of the trees. Therefore, as long as $K \leq \frac{2^d-1}{d}$, an adversary can choose the K broadcasting nodes in such a way that *all* of the packets will be received by node x through arc $(x \oplus e_{j^*}, x)$; in this case, the broadcasts last for at least K time units. [Similarly, if K is $\Theta(d^{-\epsilon}2^d)$ with $0 < \epsilon < 1$, then the adversary can force $\Omega(\frac{K}{d^{1-\epsilon}})$ of the packets to be transmitted over the same arc.] The above argument shows that, in the worst case, the completion time of the task will not be of the optimal order of magnitude, unless the paths to be followed by the packets are chosen in an *adaptive* fashion. In the algorithm to be presented, this is attained by performing a parallel prefix task (see §4.3) prior to starting transmissions of the actual packets.

It is worth noting that the above conclusion (and the argument we used) is reminiscent of an important result by Borodin and Hopcroft [BoH82] on the inefficiency of *oblivious* routing when performing a *permutation* task. It is proved therein that, if for each origin-destination pair there is a predetermined path to be followed, then there exists a permutation that will take time $\Omega(d^{-\frac{3}{2}}2^d)$; on the other hand, it is known that each permutation can be accomplished within $4d$ time units, by choosing an appropriate set of paths (see [LeL90]).

4.3 PARALLEL PREFIX

Let $a_0, \ldots, a_{2^d-1}$ be given scalars. A special case of the *prefix* problem [LaF80] is defined as follows: Compute all partial sums of the form $\sum_{y=x}^{2^d-1} a_y$, for $x = 0, \ldots, 2^d-1$. This prefix problem can be solved efficiently in parallel in time $2d$, by using $2^{d+1}-1$ processors connected in a complete binary tree with *bidirectional* arcs [LeL90]. The main idea of the corresponding parallel algorithm is as follows: The partial sums of the form $\sum_{y=x}^{2^d-1} a_y$ with $x \geq 2^{d-1}$ can be computed by solving a prefix problem with input $a_{2^{d-1}}, \ldots, a_{2^d-1}$; the partial sums of the form $\sum_{y=x}^{2^d-1} a_y$ with $x \leq 2^{d-1}-1$ can be computed by solving a prefix problem with input $a_0, \ldots, a_{2^{d-1}-1}$ and then adding $\sum_{y=2^{d-1}}^{2^d-1} a_y$ to the results. Notice that both prefix sub-problems are of *half* the size of the original prefix problem and may be solved *in parallel.* Assuming that the time

required for performing an addition is negligible compared to the duration of a time slot, the prefix problem is solved in $2d$ slots, provided that the various communications among the tree-connected processors are pipelined appropriately.

A parallel prefix task can be performed also in the d-cube in time $2d$, by *imbedding* a complete binary tree with $2^{d+1}-1$ nodes and bidirectional arcs [LeL90]. Such an imbedding is presented in Figure 4.1, which is taken from [LeL90]; notice that after collapsing the virtual arcs of this graph, we obtain a completely unbalanced spanning tree (with bidirectional arcs) rooted at node $(0,0,0)$ (see also Figure 2.3). In the beginning of the prefix task, node x of the d-cube knows the value of a_x; at the end of the task, x knows the value of $\sum_{y=x}^{2^d-1} a_y$.

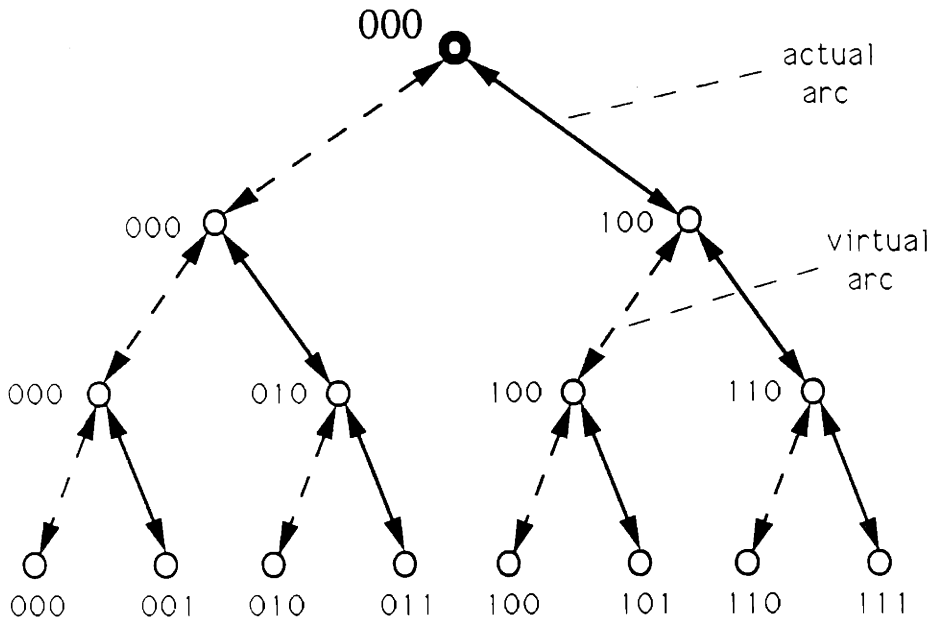

Figure 4.1: Imbedding a complete binary tree in the d-cube.

4.4 AN EFFICIENT ALGORITHM

In this section, we present a distributed algorithm for performing K simultaneous broadcasts in time $\Theta(\max\{d, \frac{K}{d}\})$ for any choice of K and of the broadcasting nodes. The main idea of the algorithm is as follows: The K packets to be broadcast are *split*

evenly among the d disjoint spanning trees $\mathcal{T}^{(1)}, \ldots, \mathcal{T}^{(d)}$; each of the packets is sent to the *root* of one of these d trees, which will eventually broadcast the packet along that tree. In more detail, the algorithm consists of three phases:

Phase 1: A *prefix* task is implemented (see §4.3), with input $a_0, \ldots, a_{2^d-1}$, where $a_x = 1$ if node x wishes to broadcast a packet, and $a_x = 0$ otherwise. This prefix computation lasts for $2d$ time units; after its completion, node x knows the value of $\sum_{y=x}^{2^d-1} a_y \stackrel{\text{def}}{=} r_x$. Notice that if node x is to broadcast a packet, then r_x equals its *rank* under the *decreasing* order within the subset of broadcasting nodes. Also note that $r_0 = K$. As an example, suppose that nodes 0, 2, 3 and 6 of the 3-cube wish to perform a broadcast; then, the prefix computation gives $r_7 = 0$, $r_6 = r_5 = r_4 = 1$, $r_3 = 2$, $r_2 = r_1 = 3$, and $r_0 = 4 = K$.

Phase 2: For each broadcasting node x, its respective packet is sent to the root $e_{j(x)}$ of tree $\mathcal{T}^{(j(x))}$, where the index $j(x)$ is determined according to the following rule:

$$j(x) \stackrel{\text{def}}{=} (r_x - 1) \bmod d + 1 \,.$$

The path to be followed by the packet of node x is the *reverse* of the path from $e_{j(x)}$ to x that is contained in $\mathcal{T}^{(j(x))}$. Let $N^{(j)}$ be the number of packets to be received by root e_j; since the r_x's of the *broadcasting* nodes are *distinct* and *consecutive*, taking all the values $K, \ldots, 1$, it follows easily that $N^{(j)}$ equals either $\lfloor \frac{K}{d} \rfloor$ or $\lceil \frac{K}{d} \rceil$, for all $j \in \{1, \ldots, d\}$. Therefore, the packets to be broadcast are split among the d disjoint spanning trees $\mathcal{T}^{(1)}, \ldots, \mathcal{T}^{(d)}$ as evenly as possible. Since the d disjoint trees remain disjoint after reversing all their constituent arcs, packets sent to *different* roots *do not interfere*. Due to pipelining, all $N^{(j)}$ packets destined for root e_j will have been received after at most $N^{(j)} + d - 1$ slots from the beginning of the present phase. Therefore, all of the transmissions involved in this phase will have been completed after $\max_{j=1,\ldots,d}\{N^{(j)} + d - 1\} = \lceil \frac{K}{d} \rceil + d - 1$ slots (and possibly earlier). The termination of this phase may be *detected* as follows: Due to pipelining, after the dth slot of phase 2 each root e_j receives its packets *continuously*; that is, if a packet is received by e_j at time $t \geq d + 1$, then e_j also received a packet during time $t - 1$ through the same incident arc, for otherwise the packet received at time t would have arrived earlier. (Recall that packets corresponding to different trees do not interfere.) Thus, root e_j can tell that it received all of its packets when there occurs a slot $t \geq d+1$ with no new arrivals; then, e_j sends a termination packet to node 0, which declares the termination

of the entire phase when it receives d such packets (*). The additional overhead for termination detection is only 3 slots. (Note that termination packets do not interfere with the actual ones, because arcs of the type $(e_j, 0)$ do not belong to any of the d "reversed" disjoint trees; see §2.1.4.) All nodes other than $e_1, \ldots, e_d$ do not have to detect termination of the present phase, because they are not supposed to trigger the next one. Hence, the duration of this phase is at most $\lceil \frac{K}{d} \rceil + d + 2$ slots.

Phase 3: Each of the roots $e_1, \ldots, e_d$ broadcasts the packets received during phase 2. Root e_j broadcasts the corresponding $N^{(j)}$ packets along tree $\mathcal{T}^{(j)}$; just after forwarding the $N^{(j)}$th packet, root e_j starts broadcasting [along $\mathcal{T}^{(j)}$] a termination packet. Again, packets broadcast along different trees *do not interfere*. By pipelining successive broadcasts over the same tree (including the termination packets), this phase is completed in $\max_{j=1,\ldots,d}\{N^{(j)} + d\} = \lceil \frac{K}{d} \rceil + d$ slots; termination is detected *individually* by each node.

It follows from the description of the algorithm that its total duration is at most $2\lceil \frac{K}{d} \rceil + 4d + 2$ time slots, which is $\Theta(\max\{d, \frac{K}{d}\})$.

For $K \gg d^2$, the upper bound on the completion time exceeds the lower bound $\max\{d, (\frac{2^d-1}{2^d})\frac{K}{d}\}$ by a factor that is very close to 2. In fact, for the case $K = 2^d$, which corresponds to a multinode broadcast, phase 1 of the algorithm is not necessary, because it is known that $r_x = 2^d - x$ for every node x. We thus obtain a *multinode* broadcast algorithm with completion time at most $2\lceil \frac{2^d}{d} \rceil + 2d + 2$ slots. In fact, the duration of this algorithm can be computed exactly, because it is known which path is to be followed by each packet. (Recall that the duration of the phase 2 was only bounded from above, because its exact value depends on the particular K-tuple of broadcasting nodes; however, for $K = 2^d$, there is only one possible such K-tuple.) We refrain from carrying out this computation, because it appears to be very tedious. Nevertheless, it may be seen that the resulting multinode broadcast algorithm takes at

(*) An alternative way for detecting termination of this phase is for each root e_j to send to node 0 the number of messages received during each slot; termination is declared by node 0 when the *sum* of all these numbers reaches the value of K, which is known to node 0 since phase 1. Though somewhat more subtle, we prefer the method introduced earlier, because it involves fewer messages and does not require knowledge of K (even by node 0); the advantage of this will become more clear in §4.5.

least $\lfloor \frac{2^{d-1}}{d} \rfloor + \lceil \frac{2^d}{d} \rceil + d$ slots, because node e_{d-1} has to receive at least $\lfloor \frac{2^{d-1}}{d} \rfloor$ packets through arc $(e_d \oplus e_{d-1}, e_{d-1})$ in the phase 2. [This follows from the fact that, in the paths of tree $\mathcal{T}^{(d-1)}$, the hypercube dimensions are crossed in the order $d, 1, 2, \ldots, d-1$ (see §2.1.4); thus, all nodes x with $x \geq 2^{d-1}$ constitute a subtree of $\mathcal{T}^{(d-1)}$ hanging from arc $(e_{d-1}, e_d \oplus e_{d-1})$, and at least $\lfloor \frac{2^{d-1}}{d} \rfloor$ of them have to send their respective packet to root e_{d-1}.] Therefore, the completion time of the new multinode broadcast algorithm exceeds the optimal value $\lceil \frac{2^d-1}{d} \rceil$ by a factor between $\frac{3}{2}$ and 2. However, this suboptimal algorithm is much *simpler* to implement than the multinode broadcast algorithms of [SaS85], [JoH89] and [BOSTT91]. Indeed, the former algorithm involves a *total* of $d+1$ spanning trees, whereas the latter involve a total of at least 2^d trees; also the trees used by the algorithm discussed above can be described in a rather *concise* way, which reduces its memory requirements even further (see also §2.1.3).

Furthermore, for $K \ll d^2$, the duration of the algorithm exceeds the lower bound $\max\{d, (\frac{2^d-1}{2^d})\frac{K}{d}\}$ by a factor that is close to 4. Finally, for $K = \Theta(d^2)$, the corresponding factor is between 2 and 6, with the worst case arising for $K = d^2$. (It should also be noted that the quantity $\max\{d, (\frac{2^d-1}{2^d})\frac{K}{d}\}$ is not necessarily a tight lower bound for the completion time of the task.) In fact, $K = \Theta(d^2)$ is the largest order of magnitude for K that can possibly lead to a completion time of $\Theta(d)$, i.e. of the same order of magnitude as the time for a single node broadcast.

The algorithm presented above for the K simultaneous broadcasts in the d-cube is fully *distributed*, that is, it does not require any centralized coordination. Moreover, the algorithm is *non-oblivious* (i.e., adaptive), meaning that the paths followed by different packets are not selected independently; this is due to the first phase of the algorithm, where the broadcasting nodes coordinate in order to split evenly the packets among the d disjoint spanning trees. An oblivious *randomized* version of our algorithm is presented in §4.5.

Throughout the derivation of the efficient algorithm, it was assumed that the K broadcasting nodes were distinct. If this is not the case, then the value of a_x (in the prefix computation) should be set to the *number* of packets to be broadcast by node x; K now stands for the total number of packets to be broadcast. If node x has $a_x \geq 2$ packets, then it should send its mth packet to the root indexed by $(r_x - a_x - 1 + m) \bmod d+1$, for $m = 1, \ldots, a_x$. The duration of the algorithm is again

$2\lceil \frac{K}{d} \rceil + 4d + 2$ time slots; notice also that the lower bound derived in §4.2 still applies.

4.5 A RANDOMIZED VERSION OF THE EFFICIENT ALGORITHM

In this section, we present a randomized version of the algorithm of §4.4; the new algorithm accomplishes the K simultaneous broadcasts in time $\Theta(\max\{d, \frac{K}{d}\})$ with high probability, for any K-tuple of broadcasting nodes. The main idea is to skip phase 1 (where a parallel prefix task is performed), and *randomly* split the packets among the d trees. In particular, we now assume that each broadcasting node x selects randomly the value of $j(x)$, with $\Pr[j(x) = i] = \frac{1}{d}$ for all $i \in \{1, \ldots, d\}$. Again, let $N^{(j)}$ denote the number of nodes having chosen $T^{(j)}$; now, $N^{(j)}$ is a random variable. Clearly, if $\max\{N^{(1)}, \ldots, N^{(d)}\} \leq C\frac{K}{d}$ (with $C \geq 1$), then the task can be accomplished in at most $2C\frac{K}{d} + 2d + 2$ time slots, by performing phases 2 and 3 exactly as in the algorithm of §4.4; on the other hand, it follows from the Chernoff bound that the inequality $\max\{N^{(1)}, \ldots, N^{(d)}\} \leq C\frac{K}{d}$ holds with high probability, for an appropriate value of the constant C. Notice that the termination of phase 2 cannot be detected by the alternative method that counts the total number of packets received by the root nodes, because the value of K has *not* been previously computed.

As an example, let us consider the case $K = d^2$; we have

$$\Pr[N^{(j)} = n] = \binom{d^2}{n}\left(\frac{1}{d}\right)^n\left(1 - \frac{1}{d}\right)^{d^2-n}, \quad \text{for } n = 0, \ldots, d^2 .$$

This together with the Chernoff bound implies that

$$\Pr[N^{(j)} \geq Cd] \leq \left(1 - \frac{1}{d} + \frac{z}{d}\right)^{d^2} z^{-Cd} \leq e^{(z-1)d} z^{-Cd}, \qquad \forall z \geq 1;$$

applying this with $z = C$ and using the fact $1 + \alpha < e^{\alpha}$, it follows that

$$\Pr[N^{(j)} \geq Cd] \leq \left(\frac{1}{C}\right)^{Cd} e^{(C-1)d}, \qquad \forall C \geq 1 .$$

Applying the union bound, we obtain after some algebra

$$\Pr[\max\{N^{(1)}, \ldots, N^{(d)}\} \geq Cd] \leq \left(\frac{e}{C}\right)^{Cd}, \qquad \forall C \geq 1 .$$

Hence, for any $C > e$, the algorithm completes in less than $2(C+1)d + 2$ slots, with probability at least $1 - (\frac{e}{C})^{Cd}$ for any d^2-tuple of broadcasting nodes.

The randomized algorithm presented above is *oblivious*. Hence, for the task of K simultaneous broadcasts, randomized oblivious routing is efficient, while *deterministic* oblivious routing is *inefficient* (see §4.2). Again, this conclusion is reminiscent of the well-known results of [Val82] and [BoH82] for the permutation task.

4.6 EFFICIENT ALGORITHMS FOR SPECIAL CASES

In this section, we present simple efficient algorithms for performing K-simultaneous broadcasts in the d-cube in the cases where K is *known* to satisfy $K = O(d)$, $K = 2$ or $K = d$.

4.6.1 The Case $K = O(d)$

Consider the following *distributed* algorithm: Each of the K nodes broadcasts its packet along a completely unbalanced spanning tree rooted at itself, with all of these trees having the *same* index-order of crossing the hypercube dimensions; e.g. the increasing index-order. Suppose that a copy $\mathcal{P}_{z,j}(x)$ of the packet originating at a node x wishes to traverse some arc $(z, z \oplus e_j)$ at the same time with the copy $\mathcal{P}_{z,j}(y)$ of another packet originating at node y. Then, both $\mathcal{P}_{z,j}(x)$ and $\mathcal{P}_{z,j}(y)$ are destined for the *same* subset of nodes, namely all nodes w satisfying $w_1 = z_1, \ldots, w_{j-1} = z_{j-1}$. Therefore, if $\mathcal{P}_{z,j}(x)$ traverses arc $(z, z, \oplus e_j)$ before $\mathcal{P}_{z,j}(y)$, then $\mathcal{P}_{z,j}(y)$ (or copies thereof to be generated later) will *never* be delayed again due to copies of the packet originating at node x. This argument implies that each copy of a packet suffers at most $K - 1$ units of delay caused by contention; thus, the algorithm terminates after at most $d + K - 1$ time units. Unfortunately, this upper bound for the completion time is of the optimal order of magnitude $\Theta(\max\{d, \frac{K}{d}\})$ only if K is $O(d)$; on the other hand, since each node is *confined* to a prespecified spanning tree, when K is not $O(d)$ there are cases where the algorithm does not complete in $\Theta(\max\{d, \frac{K}{d}\})$ time units, according to the negative result of §4.2.

The algorithm above is faster than the one presented in §4.4 for all $K \leq 3d$, and it should be used only when K is known to be $O(d)$.

4.6.2 The Case $K = 2$

Next, we present a *distributed* algorithm for performing 2 broadcasts simultaneously in d time units, which is clearly the fastest possible.

We denote by x and y the two *distinct* broadcasting nodes; moreover, let $\bar{x}$ and $\bar{y}$ be the nodes at Hamming distance d from x and y, respectively. Using the algorithm presented in §4.6.1, we can perform the 2 simultaneous broadcasts in at most $d+1$ time units. Since each copy of a packet suffers at most one unit of delay caused by contention, the algorithm will last for exactly d slots if the following property holds: Node $\bar{x}$ receives the packet of node x after d slots and at the same time node $\bar{y}$ receives the packet of node y after d slots. This is guaranteed by introducing the following simple priority discipline: Assume that two copies $\mathcal{P}_{z,j}(x)$ and $\mathcal{P}_{z,j}(y)$ of the packets under broadcast collide at arc $(z, z \oplus e_j)$; if node $\bar{x}$ (and resp. node $\bar{y}$) will eventually receive a copy of $\mathcal{P}_{z,j}(x)$ [and resp. of $\mathcal{P}_{z,j}(y)$], then $\mathcal{P}_{z,j}(x)$ [and resp. $\mathcal{P}_{z,j}(y)$] should be transmitted first. To see that this priority discipline works, it suffices to show the following property: If $\mathcal{P}_{z,j}(x)$ is destined for node $\bar{x}$, then $\mathcal{P}_{z,j}(y)$ *cannot* be destined for node $\bar{y}$, and vice versa. To prove this, notice that, when the hypercube dimensions are crossed in increasing index-order (or in any other prespecified index-order), the paths from x to $\bar{x}$ and from y to $\bar{y}$ are *disjoint.* Indeed, since both paths have length d, it follows that if an arc $(w, w \oplus e_j)$ were shared by these two paths, then we would have $w_1 \neq x_1, \ldots, w_{j-1} \neq x_{j-1}, w_j = x_j, \ldots, w_d = x_d$ and at the same time $w_1 \neq y_1, \ldots, w_{j-1} \neq y_{j-1}, w_j = y_j, \ldots, w_d = y_d$; these imply that $x = y$, which is a contradiction.

4.6.3 The Case $K = d$

For $K = d$, the algorithm of §4.6.1 lasts for at most $2d-1$ slots. This upper bound can be tightened by introducing priority disciplines such as the one presented in §4.6.2. Nevertheless, there still exist cases where the algorithm would take more than d time units. Below, we present an algorithm that completes in d time units; however, this algorithm assumes that K is *known* to equal d and that each broadcasting node x *knows* its rank r_x within the d-tuple of broadcasting nodes. The algorithm is as follows: Node x will broadcast its packet along the completely unbalanced spanning tree (rooted at x) in which the hypercube dimensions are crossed in the following index-order:

$$r_x \bmod d + 1, (r_x + 1) \bmod d + 1, \ldots, (r_x + d - 1) \bmod d + 1 \, ;$$

moreover, the packet of node x may cross the arcs of dimension $(r_x + m - 2) \bmod d + 1$ *only* at the mth slot. To see that copies of different packets never collide, it suffices

to show that $(r_x + m - 2) \bmod d + 1 \neq (r_y + m - 2) \bmod d + 1$ for $x \neq y$; this follows from the fact $r_x \neq r_y$ while both r_x and r_y belong to $\{1, \ldots, d\}$.

As already established in §4.4, the ranks of the broadcasting nodes can be computed in $2d$ time slots, by running a parallel prefix phase. If this overhead is taken into account, then the total duration of the algorithm would be $3d$ slots; this is better than the time $4d + 3$ taken by the algorithm of §4.4, but it exceeds the completion time $2d - 1$ attained by the simple algorithm of §4.6.1. Of course, if the *same* d-tuple of nodes is to perform a simultaneous broadcast several times, then the computation of the ranks should be carried out only once; in such a case, the present algorithm might be preferable. In the extreme case where *one* node has d packets to broadcast, then the parallel prefix computation is redundant, and the algorithm takes d time units, which is the fastest possible.

5. Multiple Node-to-Node Communications in the Hypercube

5.1 INTRODUCTION

So far, we have only dealt with static routing problems. In the present chapter, we consider the *dynamic* problem of performing multiple node-to-node communications; that is, packets generated randomly over an infinite time-horizon have to be transmitted from one node of the hypercube to another.

5.1.1 Problem Definition

The precise definition of the problem to be analyzed is as follows: Each node of the d-cube generates packets according to a *Poisson* process with rate λ; different nodes generate their packets independently of each other. Each packet has a *single* destination, which is selected *randomly*; in particular, we assume that

$$\Pr[\text{a packet generated by node } x \text{ is destined for node } z] = p^{H(x,z)}(1-p)^{d-H(x,z)}, \tag{5.1}$$

where the parameter $p \in (0,1]$ is a *constant* (independent of d) and $H(x,z)$ denotes the Hamming distance between nodes x and z (see §2.1.2); different packets make their selections independently of each other.

The rule (5.1) for choosing a packet's destination may be implemented as follows: Every packet $\mathcal{P}$ flips each of the bits of the identity of its origin x, and the resulting binary string is taken as the identity of $\mathcal{P}$'s destination; each bit-flip is performed with probability p, independently of the others. Hence, the average Hamming distance between a packet's origin and destination equals dp.

It is easily seen from (5.1) that for $p = \frac{1}{2}$ the destination distribution is *uniform*; that is, each node (including its origin) is equally likely to be chosen as a packet's destination. This is the case usually considered in the literature; see §1.5.2 (*). For $p < \frac{1}{2}$, the destination distribution favors nodes at shorter distance from a packet's origin; in this case, packet transmissions tend to be more localized.

For analytical convenience, we shall assume that time is *continuous*. The results to be derived can be easily extended to the case of slotted time, which is treated in §5.4.

The problem defined above is *invariant under translation*; that is, if each hypercube node is relabelled from x to $x \oplus y^*$ (where y^* is a fixed d-bit string), then the statistics of the various random variables are not affected.

5.1.2 Setting the Performance Objectives

For the routing problem defined in §5.1.1, our objective is essentially to route as many packets per time unit as possible, without each packet suffering excessive delay due to contention. As already explained in §1.2.2, we are interested in devising efficient *on-line* routing *schemes*, which means that packets are to be routed only on the basis of past information; this restriction is very reasonable for our problem, because it is not known at any time what packets will be generated in the future.

As will be proved in §5.2.1, the inequality

$$\rho \stackrel{\text{def}}{=} \lambda p < 1 \tag{5.2}$$

(*) In most of the related works, a packet's origin is not a permissible destination; however, the results to be derived (when rescaled appropriately) also apply to this case.

is a *necessary* condition for *stability*; that is, if this inequality does not hold, then, with probability 1, the number of undelivered packets will grow to infinity as time elapses. The parameter ρ will be called the *load factor* of the system. Therefore, we are interested in devising a routing scheme that is guaranteed to be stable for all $\rho < 1$. Moreover, regarding the average delay T per packet, it is desirable that for each $\rho < 1$ there exists some C_ρ (which does not depend on d) such that $T \leq C_\rho d$. Note that T is defined as the steady-state average time elapsing until a packet reaches its destination.

The aforementioned requirement on the delay is motivated as follows: In light of (5.2), a routing scheme preserving stability for all $\rho < 1$ can sustain the same rate *per node* of generating packets for a hypercube network of *arbitrary* dimensionality d. On the other hand, since each packet has to travel at an average Hamming distance of dp (see §5.1.1), in the *abscence* of contention it would take an average of dp time units to send a packet to its random destination, by applying shortest-path routing. Thus, the average delay per packet under zero traffic is $\Theta(d)$, namely of the order of magnitude of the diameter of the d-cube. (Recall that p is constant.) Since ρ does not depend on d, it is also desirable that the average delay per packet in the presence of contention is still $\Theta(d)$, with the multiplicative factor involved depending only on the load of the network.

5.1.3 The Greedy Routing Scheme — Summary of the Results

The simplest approach to our routing problem is for each packet to choose a shortest path leading to its destination and attempt to traverse this path as fast as possible. Although it is intuitively clear that such *greedy* schemes may possibly be efficient, their performance had not been analyzed rigorously in the previous literature; see §1.5.2. We shall prove that a certain greedy scheme is very efficient, and, in particular, that it meets the performance objectives set in §5.1.2.

The routing scheme to be analyzed is as follows: Consider a packet originating at node x and destined for node z; this packet will be routed through the *canonical* path from x to z (see §2.1.2); e.g., a packet travelling from node $(0,0,0,0)$ to node $(1,0,1,1)$ in the 4-cube would follow the path

$$(0,0,0,0) \rightarrow (0,0,0,1) \rightarrow (0,0,1,1) \rightarrow (1,0,1,1)\,.$$

Packets advance at their respective paths as *fast* as possible. No *idling* occurs, hence

the characterization "greedy"; that is, it never happens that an arc $(y, y \oplus e_k)$ is idle while one (or more) of the packets stored at node y have to cross this arc. Finally, contention is resolved on a FIFO basis; thus, whenever several packets present at a node y wish to traverse the same arc, priority is given to the one that arrived at y the first.

It will be proved that the routing scheme above is stable for all $\rho < 1$, which, in light of (5.2), is the broadest possible stability region. By the term "stability" it is meant that, as $n \to \infty$, the distribution of the total time spent by the nth packet in the network (i.e., of the nth sojourn time) converges to a *proper* distribution, which is independent of the initial state; recall that a proper distribution corresponds to a random variable that is finite with probability 1. It will also be established that, for $\rho < 1$, the delay T induced by the greedy scheme satisfies

$$dp + p\frac{\rho}{2(1-\rho)} \leq T \leq \frac{dp}{1-\rho} .$$

Of particular interest is the upper bound on the delay, which coincides with the average delay in an $M/M/1$ queue with utilization ρ and average service time dp. (Notice that, for our routing scheme, dp equals the total average propagation time per packet, which may also be viewed as average "service time".) The above result guarantees that, for any fixed ρ, each packet reaches its destination in an average time $\Theta(d)$, as prescribed in §5.1.2. Notice also that under *heavy* traffic (i.e., for $\rho \to 1$) the delay T increases as $\frac{1}{1-\rho}$. It will be established that this behavior is *optimal* for any fixed d. Indeed, it will be proved that $\liminf_{\rho \to 1}[(1-\rho)T] > 0$ under *any* legitimate routing scheme; this result is a consequence of a *universal* lower bound for the delay, that is, a bound applying under *all* routing schemes. We shall also discuss some open questions related to the routing problem of interest, and we shall analyze the case of slotted time as well.

The results above may be easily extended to the d-dimensional *butterfly*. This cross-bar network is an "unfolded" version of the d-cube; see §5.6.1 and [BeT89]. In this context, it is assumed that packets are generated at one of the fronts of the butterfly and destined for a randomly chosen node at the opposite front; the destination distribution is identical to that presented in (5.1), except for the fact that x and z belong to opposite fronts of the butterfly. Crossing the dimensions in increasing index-order (as in the canonical paths of the hypercube) is the only legitimate choice of paths for

the butterfly. Thus, in this context, our scheme simply reduces to greedy routing; this will be seen to be stable for all $\rho < 1$, where ρ is now defined as $\rho \stackrel{\text{def}}{=} \lambda \max\{p, (1-p)\}$; moreover, for $\rho < 1$, the average delay T satisfies

$$d + p\frac{\lambda p}{2(1-\lambda p)} + (1-p)\frac{\lambda(1-p)}{2[1-\lambda(1-p)]} \leq T \leq \frac{dp}{1-\lambda p} + \frac{d(1-p)}{1-\lambda(1-p)} .$$

Again, the delay T is $\Theta(d)$ for any fixed $\rho < 1$, which is the optimal order of magnitude; also, the behavior of T under heavy traffic will be seen to be optimal, for any fixed d.

5.1.4 Summary of the Related Literature — Contribution

The dynamic routing problem of this chapter has been dealt with in several articles, which were discussed in detail in §1.5.2. In particular, Abraham and Padmanabhan [AbP86] approximately analyzed the problem for the case of uniform destination. Greenberg and Hajek [GrH89] approximately analyzed the performance of *deflection* routing, where packets may be temporarily misrouted, rather than stored or dropped. Varvarigos [Var90] has formulated a Markov chain model for evaluating the performance of deflection routing, and has investigated its steady-state statistics numerically. All three [AbP86], [GrH89] and [Var90] are dealing with the hypercube network. The same problem has been analyzed in the context of the 2-dimensional array by Greenberg and Goodman [GrG86], and in the context of the extended Omega network by Mitra and Cieslak [MiC87], as well as by Hajek and Cruz [HaC87]; all these works are approximate (see §1.5.2.) Recently, Leighton [Lei90] proved that greedy routing in the 2-dimensional array has very satisfactory average performance. Finally, Bouras et al. [BGST87] investigated the performance of greedy routing in the context of Banyan networks; however, the analysis therein appears to be incorrect.

To the best of our knowledge, the results derived in this chapter are new. Moreover, our analysis provides the first proof that some routing scheme (on either the d-cube or the butterfly) is stable for all $\rho < 1$ while satisfying the requirement for $\Theta(d)$ average delay; proving that greedy routing has these properties has been an important open question in the routing literature. Also, this is the first routing scheme for which the bounds on the delay are expressed in simple formulae involving the system's parameters ρ and d. Finally, the approach for deriving the aforementioned results is new as well: It is established that the hypercube (resp., the butterfly) behaves as a queueing network with deterministic servers (each corresponding to an arc) and with

Markovian routing among the various servers; then, by using sample-path arguments, it is shown that the total number of packets present in this queueing network is dominated by that corresponding to a product-form network. This approach contrasts with the combinatorial approach used in [Lei90], which is based on the following idea: A packet suffering large delay should have collided with an untypically large number of packets; by careful enumeration of such "bad" scenarios, it is then proved that the probability for a packet to be delayed excessively is small. Our approach relies on the assumption of Poisson arrivals; nevertheless, we hope that our analysis will be suggestive of the efficient perfomance of greedy routing under more general packet-generating processes.

Finally, it should be noted that similar results may be proved for other crossbar switches with arcs arranged in levels (e.g, the Omega and Banyan networks), even if different arcs have different (yet fixed) transmission times. This is due to the generality of an interim result established in our analysis, namely Proposition 5.9.

5.2 PRELIMINARY RESULTS

First, we formalize an earlier observation, which will be used several times in the analysis. Consider a fixed packet $\mathcal{P}$ generated at node x. Let $\mathcal{B}_i$ denote the event that packet $\mathcal{P}$ will choose a destination z such that $z_i \neq x_i$; notice that if event $\mathcal{B}_i$ occurs, then $\mathcal{P}$ will have to cross an arc of the ith dimension in order to reach its destination. As already explained in §5.1.1 [see the comment below (5.1)], the following is true:

Lemma 5.1: For any fixed packet $\mathcal{P}$, events $\mathcal{B}_1, \ldots, \mathcal{B}_d$ are mutually independent, with $\Pr[\mathcal{B}_i] = p$ for $i = 1, \ldots, d$. Independence prevails both with and without conditioning on the origin of the packet. ■

5.2.1 The Necessary Condition for Stability

In this subsection, we derive the necessary condition for stability. The average total number of packets generated in the network per time unit equals $\lambda 2^d$. Recall now that each packet has to undertake an average of at least dp transmissions under any routing scheme (see §5.1.1); thus, it follows that during each time unit an average total demand for at least $(\lambda 2^d)(dp)$ packet transmissions is generated in the system. Since at most $d2^d$ packet transmissions may take place per time unit, it follows that the system can be stable only if $(\lambda 2^d)(dp) < d2^d$. Thus, we obtain the following necessary

condition for stability under *any* routing scheme:

$$\rho \stackrel{\text{def}}{=} \lambda p < 1 \,, \tag{5.2}$$

where ρ is the *load factor* of the system. This terminology is appropriate, because when $\rho \approx 1$ all hypercube arcs are almost always busy, even if no redundant packet transmissions take place. Notice that this necessary condition for stability applies under more general arrival processes as well.

5.2.2 Lower Bounds on the Delay

First, we establish a *universal* lower bound on the steady-state average delay T per packet; that is, a bound that applies to *any* routing scheme. Recall that T is defined as the steady-state average of the time elapsing between the moment a packet is generated until it reaches its destination.

Proposition 5.2: The average delay T per packet induced by any routing scheme satisfies

$$T \geq \max\{dp, p\mathcal{D}(2^d;\rho)\} = \Omega\left(dp + p\frac{\rho}{2^d(1-\rho)}\right) \,, \qquad \forall \rho < 1 \,,$$

where $\mathcal{D}(2^d;\rho)$ is the average delay for the $M/D/2^d$ queue with unit service time and arrival rate $2^d\rho$. ■

Proof: Consider a fixed packet $\mathcal{P}$ generated at node x; if its random destination satisfies $z_1 \neq x_1$ (that is, if event $\mathcal{B}_1$ occurs for $\mathcal{P}$), then $\mathcal{P}$ will not reach its destination until it traverses at least one arc of the 1st dimension. Let W be the average time until a packet crosses the 1st dimension, with the convention that packets that do not have to cross the 1st dimension contribute zero to this average; clearly, there holds $T \geq W$. It is straightforward to see that the value of W can only decrease if we introduce the following conditions:

(a) Each packet for which event $\mathcal{B}_1$ has not occurred never crosses the 1st dimension.

(b) Each packet for which event $\mathcal{B}_1$ has occurred is *available* upon its generation at all nodes. Moreover, such a packet will only cross the *first* available arc of type 1; this actually leads to the minimum possible value of W, as proved in [StT91].

Under these assumptions, the 2^d arcs of the 1st dimension operate as an $M/D/2^d$ queue. The input stream of this queue consists of all packets for which event $\mathcal{B}_1$

occurs; by Lemma 5.1, this stream is Poisson with rate $\lambda 2^d p = 2^d \rho$. The average delay induced by this queue equals $\mathcal{D}(2^d; \rho)$; since only a fraction p of the packets "joins" this $M/D/2^d$ queue, we have

$$W \geq p\mathcal{D}(2^d; \rho) \,. \tag{5.3}$$

Recall now that $T \geq dp$ (under any routing scheme) and $T \geq W$; these facts together with (5.3) imply that

$$T \geq \max\{dp, p\mathcal{D}(2^d; \rho)\} \,. \tag{5.4}$$

Furthermore, it is known [Bru71] that

$$\mathcal{D}(2^d; \rho) \geq 1 + \frac{\rho}{2^{d+1}(1-\rho)} \,;$$

combining this with (5.4), it follows that

$$T = \Omega\left(\max\left\{dp, p + p\frac{\rho}{2^{d+1}(1-\rho)}\right\}\right) = \Omega\left(dp + p\frac{\rho}{2^d(1-\rho)}\right) \,,$$

where we have also used the inequality $\max\{\alpha_1, \alpha_2\} \geq \frac{1}{2}(\alpha_1 + \alpha_2)$. The proof of the result is now complete. **Q.E.D.**

The universal lower bound of Proposition 5.2 shows that $\liminf_{\rho \to 1}[(1-\rho)T] > 0$, for any fixed d, under any routing scheme; thus, under heavy traffic, the delay T grows at least as fast as $\frac{1}{1-\rho}$. (Since p was taken fixed, $\rho \to 1$ means $\lambda \to \frac{1}{p}$.) As far as asymptotics with respect to d are concerned, the bound appears to be loose, due to the presence of the factor $\frac{1}{2^d}$. Below, we establish a sharper lower bound applying to a restricted but fairly broad class of routing schemes.

As suggested by the proof of Proposition 5.2, a scheme that comes close to attaining the universal lower bound for the delay T (if there exists such a scheme) would schedule transmissions *adaptively*. This claim is further supported by Proposition 5.3, which establishes a lower bound on the average delay per packet induced by *oblivious* schemes. Under such a scheme, each packet selects its path independently of the existing traffic and insists on traversing the selected path, regardless of the contention encountered en route (see [BoH82]). We also assume that all packets generated at the same node follow the same rules, which are time-independent. We now present the lower bound on the delay induced by oblivious schemes.

Proposition 5.3: The average delay T per packet induced by any *oblivious* routing scheme satisfies

$$T \geq \max\left\{dp, p\left[1+\frac{\rho}{2(1-\rho)}\right]\right\} = \Omega\left(dp + p\frac{\rho}{1-\rho}\right) . \qquad \blacksquare$$

Proof: This proof is similar to that of Proposition 5.2. We consider a node x and an arc $(y, y \oplus e_1)$; under any oblivious scheme, the following is true: For each packet generated at x, arc $(y, y \oplus e_1)$ is the *first* arc of type 1 to be crossed by such a packet with probability $q_{x,y}$, *independently* of all other events occurring in the network; moreover, there holds

$$\sum_{y=0}^{2^d-1} q_{x,y} \geq p\,, \qquad \forall x \in \{0,\ldots,2^d-1\}\,, \tag{5.5}$$

because it is with probability p that some packet generated by node x will *necessarily* cross an arc of the 1st dimension (due to the occurrence of event $\mathcal{B}_1$). Let W be the average time until a packet crosses the 1st dimension for the first time, with the convention that packets that do not have to cross the 1st dimension contribute zero to this average; obviously, we have $T \geq W$. For any oblivious routing scheme, the value of W can only decrease if we introduce the following conditions:

(a) Each packet to cross the 1st dimension is only delayed at the first time it does so.

(b) Each packet to cross arc $(y, y \oplus e_1)$ is available at node y upon its generation.

Under these conditions, each arc $(y, y \oplus e_1)$ is fed by a group of 2^d Poisson streams. We denote by r_y the total arrival rate of the compound Poisson stream; this rate is constant, because the routing rules were taken time-independent. Obviously, we have

$$r_y = \lambda \sum_{x=0}^{2^d-1} q_{x,y}\,, \qquad \forall y \in \{0,\ldots,2^d-1\}\,. \tag{5.6}$$

Clearly, arc $(y, y \oplus e_1)$ behaves as an $M/D/1$ queue with unit service time. Therefore (see [Kle75]), the average delay W_y per packet joining this queue is given as follows:

$$W_y = 1 + \frac{r_y}{2(1-r_y)}\,.$$

Using this, we obtain

$$W \geq \frac{1}{\lambda 2^d}\sum_{y=0}^{2^d-1} r_y W_y = \frac{1}{\lambda 2^d}\sum_{y=0}^{2^d-1} r_y\left[1+\frac{r_y}{2(1-r_y)}\right]\,. \tag{5.7}$$

Combining (5.5) and (5.6), we have

$$\sum_{y=0}^{2^d-1} r_y \geq \lambda 2^d p \,. \tag{5.8}$$

Notice now that $r[1 + \frac{r}{2(1-r)}]$ is a *convex* and *increasing* function of r; therefore, in light of (5.8), the right-hand quantity in (5.7) is minimized when $r_y = \lambda p$ for all $y \in \{0, \ldots, 2^d - 1\}$. Thus, it follows that

$$W \geq \frac{1}{\lambda 2^d} \sum_{y=0}^{2^d-1} \lambda p \left[1 + \frac{\lambda p}{2(1-\lambda p)}\right] = p\left[1 + \frac{\rho}{2(1-\rho)}\right] .$$

This together with the facts $T \geq W$ and $T \geq dp$ implies that

$$T \geq \max\left\{dp, p\left[1 + \frac{\rho}{2(1-\rho)}\right]\right\} ;$$

the proof is completed by using the inequality $\max\{\alpha_1, \alpha_2\} \geq \frac{1}{2}(\alpha_1 + \alpha_2)$. **Q.E.D.**

Proposition 5.3 implies that, for a fairly broad class of schemes, the universal lower bound on the delay T (see Proposition 5.2) is rather loose. Suppose now that, under oblivious routing, we allow packets generated at each node x to take into account the routing decisions made by packets previously generated at the *same* node x (instead of selecting the path for each packet on individual basis). This assumption is appropriate for *distributed* systems, where packets do not have any specific information on the travelling schedule of other packets generated elsewhere in the network. It is an interesting open question to investigate whether Proposition 5.3 still holds. Related to this question are the server allocation problems discussed by Stamoulis and Tsitsiklis in [StT91]; it is assumed therein that customers arrive in a multiserver system through several streams, and each stream is scheduled on the basis of individual information. We conjecture that Proposition 5.3 is still valid, because each packet has a very limited knowledge of the routing decisions made within the entire network. If this is indeed the case, then a scheme violating the lower bound given by Proposition 5.3 should involve *centralized coordination* and/or *adaptive* routing.

It is worth noting that Propositions 5.2 and 5.3 hold for any destination distribution that is invariant under translation; that is, when the probability that a packet

originating at node x is destined for node z equals $f(x \oplus z)$, which depends only on $x \oplus z$. In such a case, the load factor ρ is defined as

$$\rho \stackrel{\text{def}}{=} \max\{\rho_1, \ldots, \rho_d\} ;$$

ρ_j is the load factor for the jth dimension and equals

$$\rho_j \stackrel{\text{def}}{=} \lambda \sum_{\{y : y_j = 1\}} f(y) .$$

Moreover, the condition $\rho < 1$ is still necessary for stability under any routing scheme.

5.2.3 Simple Non-Greedy Schemes

In this subsection, we discuss routing schemes based on *pipelining* successive instances of an algorithm used in *static* routing. For simplicity, we assume that the destination distribution is uniform (i.e., $p = \frac{1}{2}$).

As already mentioned in §1.4, in the first phase of the *permutation* algorithm of [VaB81], each packet selects a destination at random (with all nodes being equiprobable) and travels there. This algorithm has the following property: there exists a constant $R > 1$ such that the first phase takes time less than Rd (and close to this value) with high probability.

Consider now the following routing scheme for our problem: At time $t = 0$, each node selects one of its packets; all selected packets are routed as in the first phase of the permutation algorithm of [VaB81]. These packets arrive at their respective destinations at time t_1, where $t_1 \leq Rd$ with high probability. At time t_1, each node selects another one of its packets, and the selected packets are again routed as in the first phase of the algorithm of [VaB81] etc.

Under this scheme, each node x routes one of its packets every Rd time units approximately. (For simplicity, we ignore the overhead required for detecting termination of each run of the static algorithm.) Since each node x generates packets at a rate λ, stability may prevail only if $\lambda Rd < 1$, or equivalently $\rho < \frac{1}{2Rd}$. Therefore, for any fixed ρ, the simple scheme described becomes unstable for large d. This undesirable performance is due to the fact that the algorithm of [VaB81] makes *inefficient* use of the communication resources of the hypercube network; indeed, the average traffic per arc is $O(\frac{1}{d})$ packets per time unit, for each of the two phases of that algorithm.

The unsatisfactory stability region of the aforementioned non-greedy routing scheme can be improved by pipelining successive instances of an efficient static algorithm for d permutations, such as the one by Chang and Simon [ChS86] or that by Valiant [Val89]. Each of these articles presents an algorithm for routing d permutations on the d-cube in $\Theta(d)$ time with high probability. Both algorithms result in an average traffic of $\Theta(1)$ packets per arc and time unit; thus, by pipelining successive instances of either of them we would obtain a routing scheme maintaining stability for $\rho < \rho^*$ with ρ^* being some small constant; e.g., using the algorithm of [ChS86] would lead to $\rho^* \approx 0.005$, which is very small compared to the upper bound given by (5.2).

All of the schemes described above are non-greedy, i.e. they involve *idling*; in particular, it often occurs that packets wait at their respective origins, while some of the arcs to be traversed are idle. As will be seen in §5.3, avoidance of this idling phenomenon improves performance dramatically.

5.3 PERFORMANCE ANALYSIS OF THE GREEDY SCHEME

In this section, we analyze the efficient greedy routing scheme for the hypercube. As already mentioned in §5.1.3, the scheme is as follows: Each packet travels from its origin to its destination through the corresponding canonical path (that is, by crossing the dimensions required in increasing index-order). Packets advance at their respective paths as fast as possible; no idling occurs. Also, whenever several packets present at a node y wish to traverse the same arc, then priority is given to the one that arrived at y the first.

This routing scheme is the *non-idling* version of one of the schemes described in §5.2.3 (namely, of that based on the permutation algorithm of [VaB81]). It will be seen in §5.3.1 that, under this scheme, the hypercube is equivalent to a queueing network with certain useful properties. The analysis in §§5.3.2–5.3.5 deals with the performance of this equivalent queueing network.

Throughout this section, the time axis is taken *continuous*.

5.3.1 The Equivalent Queueing Network

It is straightforward that, under our routing scheme, the d-cube may be viewed as a queueing network, with $d2^d$ deterministic FIFO "servers"; each "server" has unit

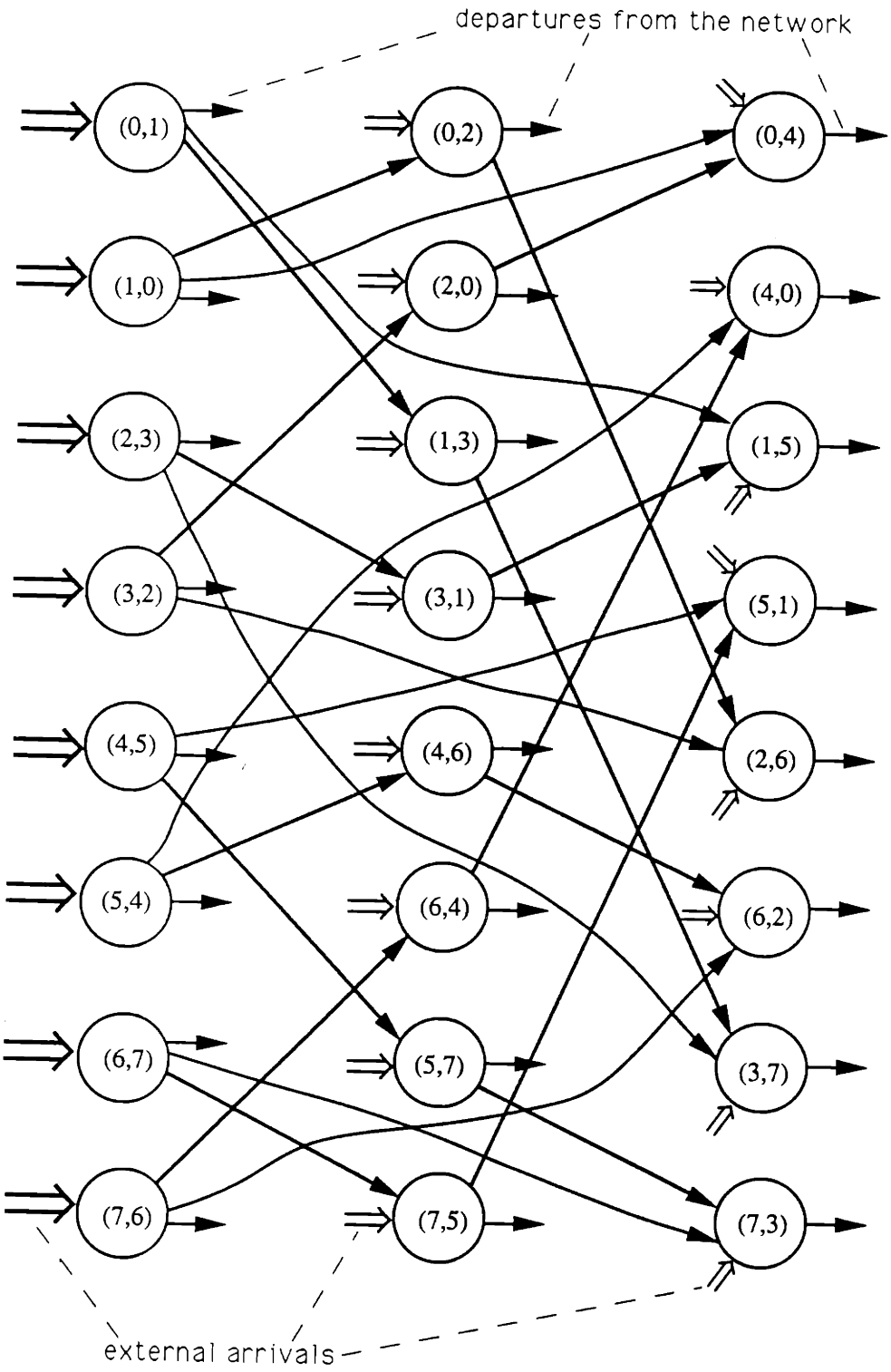

Figure 5.1: The equivalent network Q for the 3-dimensional hypercube.

service duration and corresponds to a hypercube *arc*. This equivalent queueing network (to be referred to as $\mathcal{Q}$) has the following properties:

Property A: The external arrival stream at any arc $(x, x \oplus e_i)$ is Poisson with rate $\lambda p(1-p)^{i-1}$; streams corresponding to different arcs are mutually independent.

To see this property, consider a packet $\mathcal{P}$ generated at node x of the d-cube. With probability $p(1-p)^{i-1}$ the destination of $\mathcal{P}$ satisfies $z_1 = x_1, \ldots, z_{i-1} = x_{i-1}$ and $z_i \neq x_i$ (see Lemma 5.1). Since packets cross the hypercube dimensions in increasing index-order, it follows that each of the packets generated by node x will join the queue for arc $(x, x \oplus e_i)$ with probability $p(1-p)^{i-1}$.

Property B: After crossing arc $(y, y \oplus e_i)$, a packet will *never* traverse again an arc $(z, z \oplus e_j)$ with $j \in \{1, \ldots, i\}$. Thus, the equivalent network $\mathcal{Q}$ is a *layered* network; that is, its servers are organized in d levels, with the ith level comprising all arcs $(y, y \oplus e_i)$ for $y \in \{0, \ldots, 2^d - 1\}$, i.e. all arcs of the ith dimension. Upon "service completion" at a certain level, a packet either joins a queue at a higher level (not necessarily at the next one) or it departs from the network.

In Figure 5.1, we present the equivalent network $\mathcal{Q}$ for the 3-cube (which is depicted in Figure 2.1).

Property C: Routing in the equivalent network $\mathcal{Q}$ is *Markovian*. In particular, upon crossing arc $(y, y \oplus e_i)$, a packet takes one of the following actions: either it joins the queue at arc $(y \oplus e_i, y \oplus e_i \oplus e_j)$ with probability $p(1-p)^{j-i-1}$ for $j = i+1, \ldots, d$; or it departs from the network with probability $(1-p)^{d-i}$. After crossing arc $(y, y \oplus e_d)$, a packet departs from the network with probability 1. Different packets make their routing decisions independently of each other.

The validity of Property C requires some clarification; in particular, in light of Property B, we need to show the following result:

Lemma 5.4: Consider a fixed packet $\mathcal{P}$, which has just crossed arc $(y, y \oplus e_i)$; there holds

$$\Pr[\mathcal{P} \text{ will cross } (y \oplus e_i, y \oplus e_i \oplus e_j) \mid \mathcal{P} \text{ has crossed } (y, y \oplus e_i)] = p(1-p)^{j-i-1}, \quad \text{for } i < j \leq d. \qquad \blacksquare$$

Proof: Let x denote the origin of packet $\mathcal{P}$. Clearly, in order to prove the lemma, it

suffices to establish the following result:

$$\Pr[\mathcal{P} \text{ will cross } (y \oplus e_i, y \oplus e_i \oplus e_j) \mid \mathcal{P} \text{ has crossed } (y, y \oplus e_i) \text{ and } \mathcal{P} \text{ originated at } x] = p(1-p)^{j-i-1}, \quad (5.9)$$

for any permissible origin x of $\mathcal{P}$. Notice, however, that $\mathcal{P}$ will skip the dimensions $i+1, \ldots, j-1$ and cross the jth dimension next if and only if events $\mathcal{B}_{i+1}, \ldots, \mathcal{B}_{j-1}$ did not occur while $\mathcal{B}_j$ occurred. Hence, proving (5.9) is equivalent to proving the following:

$$\Pr[\overline{\mathcal{B}}_{i+1}, \ldots, \overline{\mathcal{B}}_{j-1}, \mathcal{B}_j \mid \mathcal{P} \text{ has crossed } (y, y \oplus e_i) \text{ and } \mathcal{P} \text{ originated at } x] = p(1-p)^{j-i-1}, \quad (5.10)$$

where $\overline{\mathcal{B}}_\ell$ is the complement of event $\mathcal{B}_\ell$. Since hypercube dimensions are crossed in increasing index-order, knowledge of the origin of $\mathcal{P}$ and of the fact that $\mathcal{P}$ has just crossed arc $(y, y \oplus e_i)$ provides information only on the first i bits of the destination of $\mathcal{P}$; thus, (5.10) follows from the independence result of Lemma 5.1. **Q.E.D.**

According to the proof of Lemma 5.4, propagation of a packet $\mathcal{P}$ on the hypercube may also be visualized as follows: Upon generation, $\mathcal{P}$ decides whether or not to cross dimension 1; the probability that it decides positively equals p. If it does so, then it takes its step on this dimension and *then* it decides whether or not to cross dimension 2; on the other hand, if it does not decide to cross dimension 1, then it considers crossing dimension 2 etc.

5.3.2 Changing Service Discipline in the Equivalent Network

In the previous subsection, we established that, under the routing scheme analyzed, the hypercube is equivalent to a queueing network $\mathcal{Q}$ with deterministic FIFO servers and Markovian routing. In this subsection, we prove the following result (to be stated formally in Proposition 5.9):

If the service discipline at the servers of $\mathcal{Q}$ is *changed* from FIFO to *Processor Sharing* (PS), then the number of packets contained in the network (at any fixed time) *stochastically increases.*

The network $\tilde{\mathcal{Q}}$ operating under the PS service discipline will be seen to be of the *product form*; hence, its performance can be analyzed easily. Thus, by using the above

result, we shall derive a sufficient condition for stability (in §5.3.3) and an upper bound on the average delay for the original network $\mathcal{Q}$ (in §5.3.4).

Recall that under the PS discipline all customers present at a server receive an equal proportion of service *simultaneously*; see [Wal88], p. 354. For example, consider a deterministic PS server, with unit service rate; assume that it has two customers to serve, with the first customer arriving at time 0 and the second at time $\frac{1}{4}$. Upon arrival of the second customer, the first one has $\frac{3}{4}$ units of service remaining; however, due to the presence of the second customer, she will be served at rate $\frac{1}{2}$; thus, she will depart at time $\frac{1}{4} + 2\frac{3}{4} = \frac{7}{4}$. Similarly, it can be seen that the second customer will depart at time 2. Notice that we are using the term "service rate" for a PS server (rather than the term "service duration"), because the time duration for which a customer receives service depends on previous and future arrivals.

The proof of the stochastic comparison between networks $\mathcal{Q}$ and $\tilde{\mathcal{Q}}$ is based on several lemmas that establish sample-path results; these we present next. For simplicity, all queueing systems to be considered in these lemmas are taken initially *empty*.

Lemma 5.5: Let there be a deterministic FIFO server with unit service duration. For a fixed sequence $t_1, t_2, \ldots$ of arrival times, let $D_1, D_2, \ldots$ denote the corresponding sequence of departure times. Similarly, let $\tilde{D}_1, \tilde{D}_2, \ldots$ be the departure times for a deterministic PS server, with unit service rate, fed by the same input stream. There holds

$$D_i \leq \tilde{D}_i \,, \quad \text{for } i = 1, \ldots \qquad \blacksquare$$

Proof: Clearly, we have $D_1 = t_1 + 1$. In the context of the PS server, the 1st customer will depart at time $t_1 + 1$ only if no other customers arrive until that time; otherwise, service of the 1st customer will be slowed down, and she will depart later than $t_1 + 1$. It follows that

$$\tilde{D}_1 \geq t_1 + 1 = D_1 \,. \tag{5.11}$$

It is well-known that the PS discipline is *work-conserving*; see [Wal88], pp. 353-354. That is, the unfinished work $w(t)$ at time t is the *same* for both the FIFO and the PS servers considered. By definition of $w(t)$, we have

$$D_i = t_i + w(t_i-) + 1 \,, \quad \text{for } i = 1, \ldots \tag{5.12}$$

We now consider the ith arrival at the PS server, where $i \geq 2$. If $w(t_i-) = 0$, then reasoning as in proving (5.11), it follows that $\tilde{D}_i \geq t_i + 1 = D_i$. Assume now that $w(t_i-) \neq 0$; it is straightforward that customers depart from a deterministic PS server in the order they arrive; hence, the ith customer may depart only after an amount $w(t_i-) + 1$ of work has been finished by the server. Therefore, we have

$$\tilde{D}_i \geq t_i + w(t_i-) + 1 = D_i \,,$$

where we have also used (5.12). The proof of the lemma is now complete. **Q.E.D.**

Let there be two streams of events, one occurring at times $\tau_1, \tau_2, \ldots$ and the other at times $\tau_1', \tau_2', \ldots$ If $\tau_i \leq \tau_i'$ for $i = 1, \ldots$, then the latter stream of events will be said to be a *delayed version* of the former. For example, as implied by Lemma 5.5, for any fixed arrival stream, the departing stream of a deterministic PS server is a delayed version of the one of the corresponding FIFO server.

Lemma 5.6: Let there be a deterministic FIFO server with unit service duration. Let $D_1, D_2, \ldots$ (resp. $D_1', D_2', \ldots$) be the sequence of departure times corresponding to a fixed sequence $t_1, t_2, \ldots$ (resp. $t_1', t_2', \ldots$) of arrival times. If $t_i \leq t_i'$ for $i = 1, \ldots$, then

$$D_i \leq D_i' \,, \quad \text{for } i = 1, \ldots \qquad \blacksquare$$

Proof: There holds

$$D_1 = t_1 + 1 \qquad \text{and} \qquad D_i = \max\{D_{i-1}, t_i\} + 1 \,, \quad \text{for } i = 2, \ldots \,;$$

similarly,

$$D_1' = t_1' + 1 \qquad \text{and} \qquad D_i' = \max\{D_{i-1}', t_i'\} + 1 \,, \quad \text{for } i = 2, \ldots$$

Using these facts and the assumption $t_i \leq t_i'$ for $i = 1, \ldots$, the result follows by a straightforward inductive argument. **Q.E.D.**

The result to be established next is based on Lemmas 5.5 and 5.6; generalizing this will lead to the main result of this subsection, namely Proposition 5.9. We consider the queueing network $\mathcal{G}$ depicted in Figure 5.2a. This consists of three deterministic FIFO servers with unit service duration, denoted by S_1, S_2 and S_3. Customers completing

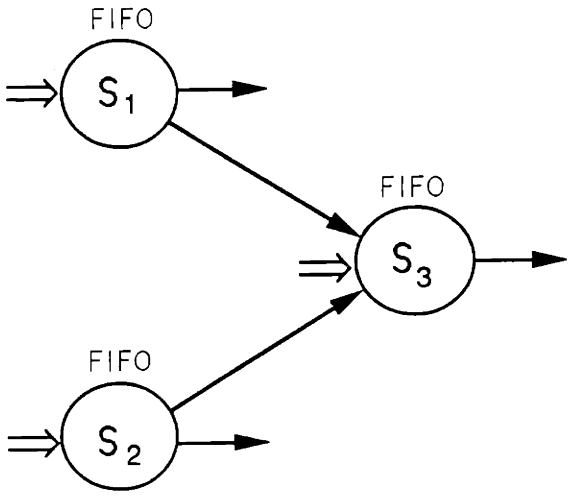

Figure 5.2a: Network $\mathcal{G}$.

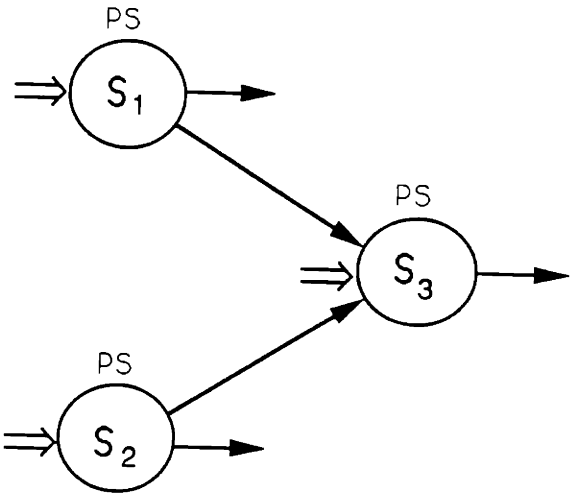

Figure 5.2b: Network $\tilde{\mathcal{G}}$.

service at S_1 or S_2 either depart from the network or they join the queue at S_3; the routing decisions are Markovian. Obviously, $\mathcal{G}$ is a *layered* network (see §5.3.1). We define a *sample path* ω of $\mathcal{G}$ as the following collection of information:

(a) The *external* arrival times at servers S_1, S_2 and S_3.

(b) The routing decision made by the ith customer upon service completion at S_1 (resp. S_2) for $i = 1, \ldots$

Clearly, given a sample path ω, network $\mathcal{G}$ evolves in a *deterministic* fashion. We also consider network $\tilde{\mathcal{G}}$, which is a network identical to $\mathcal{G}$ except for the fact that PS service discipline applies to the servers of $\tilde{\mathcal{G}}$ (instead of FIFO); see Figure 5.2b. The result to be proved is as follows:

Lemma 5.7: For a particular sample path ω, let $B(t)$ [resp. $\tilde{B}(t)$] denote the number of customers departing from $\mathcal{G}$ (resp. $\tilde{\mathcal{G}}$) during the interval $[0,t]$; for any ω, there holds

$$B(t) \geq \tilde{B}(t)\,, \qquad \forall t \geq 0\,. \qquad \blacksquare$$

Proof: First, we consider a network $\mathcal{G}'$ obtained from $\mathcal{G}$ by changing the service discipline *only* at S_1 and S_2 (from FIFO to PS); this network is depicted in Figure 2c.

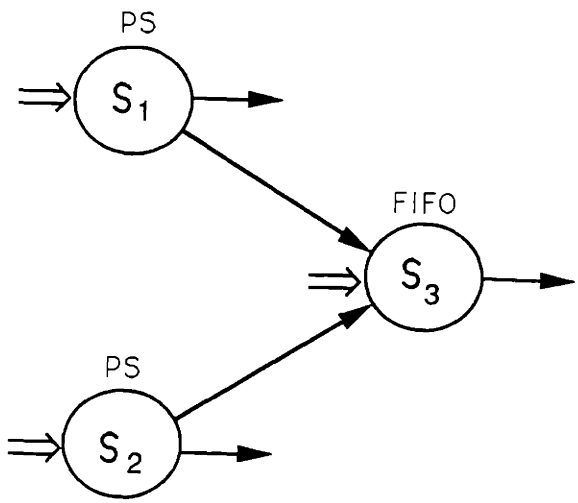

Figure 5.2c: Network $\mathcal{G}'$.

We define as the *output stream* of a server the stream of customers completing service therein, including those that do not depart immediately from the network.

Notice that server S_1 is not affected at all by the presence of the other two servers; the same statement applies to server S_2. Therefore, using Lemma 5.5, it is seen that the output stream of server S_1 in $\mathcal{G}'$ is a delayed version of that corresponding to S_1 of $\mathcal{G}$. Recalling also that the routing decisions of customers completing service are the same for networks $\mathcal{G}$ and $\mathcal{G}'$, it follows that the substream of customers departing from $\mathcal{G}'$ at S_1 is a delayed version of the corresponding substream in $\mathcal{G}$. Similar statements apply to the streams stemming from S_2.

Next, we consider the stream feeding S_3 in $\mathcal{G}'$; this stream is a delayed version of that feeding S_3 in $\mathcal{G}$, because each arrival at S_3 of $\mathcal{G}'$ corresponds to an arrival at S_3 of $\mathcal{G}$ that occurs *no later*. [Recall the aforementioned "comparison" of the output streams of S_1 (resp. S_2) in the two networks and the coupling of the routing decisions.] Therefore, applying Lemma 5.6, the output stream from S_3 of $\mathcal{G}'$ is a delayed version of that corresponding to S_3 of $\mathcal{G}$. According to Lemma 5.5, the former output stream is delayed *further* when the service discipline at S_3 of $\mathcal{G}'$ is changed from FIFO to PS. This modification (which yields network $\tilde{\mathcal{G}}$) does not affect the streams of customers departing from the 1st level. Therefore, for each of the servers of $\tilde{\mathcal{G}}$, its departing stream is a delayed version of that of the corresponding server of $\mathcal{G}$; this proves the result in question.

Its should be noted that customers joining S_3 may get out of order when changing the service discipline; thus, a *particular* customer may depart earlier from $\tilde{\mathcal{G}}$ than from $\mathcal{G}$. Nevertheless, this does not affect the validity of the Lemma 5.7. **Q.E.D.**

Next, we generalize Lemma 5.7. In the context of network $\mathcal{Q}$, a sample path ω is defined as the collection of information comprising all external arrival times and all routing decisions. Notice that routing decisions at each "server" are identified by the *order* they are made, not by the identity of the packets deciding; e.g. "the 1st packet to cross arc $(e_1 \oplus e_2, e_1)$ will advance to $(e_1, e_1 \oplus e_3)$, the 2nd such packet will depart from the network etc". Such an identification of the routing decisions is legitimate, because routing in $\mathcal{Q}$ is *Markovian*. As in Lemma 5.7, we denote as $\tilde{\mathcal{Q}}$ the network obtained from $\mathcal{Q}$ after changing the service discipline of all servers from FIFO to PS.

Lemma 5.8: For a particular sample path ω, let $B(t)$ [resp. $\tilde{B}(t)$] denote the number of packets that have departed from $\mathcal{Q}$ (resp. $\tilde{\mathcal{Q}}$) during the interval $[0,t]$; for any ω, there holds

$$B(t) \geq \tilde{B}(t)\,, \qquad \forall t \geq 0\,. \qquad \blacksquare$$

Proof: This proof is done by applying repeatedly the argument used in proving Lemma 5.7. In particular, we replace the FIFO servers by PS ones, on a level-by-level basis, starting from the 1st level and proceeding one level at a time. As in the proof of Lemma 5.7, it follows that at the jth step of this process all streams stemming from levels $1, \ldots, j-1$ remain *unaffected*, while all streams stemming from levels $j, \ldots, d$ are *delayed*. The network resulting at the dth step coincides with $\tilde{\mathcal{Q}}$, because it contains only PS servers. Thus, it follows that, for the same sample path ω, all streams departing from network $\tilde{\mathcal{Q}}$ constitute delayed versions of the corresponding streams departing from $\mathcal{Q}$. The proof of the result is now complete.

Notice that packets may get out of order at certain steps (see also the proof of Lemma 5.7). Nevertheless, this creates no difficulty, due to the particular coupling of routing decisions. If one insists on tracing the path followed by a *particular* packet [say the first to arrive at "server" $(0, e_1)$] it may occur that this *changes* at some steps of the process described above; this is of no importance, because the "comparison" of the various streams still applies, even though the streams may consist of different packets at each step. **Q.E.D.**

All Lemmas 5.5, 5.6, 5.7 and 5.8 were established for systems that are initially empty; it is straightforward that each of these results also applies when the queueing systems compared have the *same* initial condition.

Now that we have established Lemma 5.8, we can easily prove the following result:

Proposition 5.9: Let $N(t)$ [resp. $\tilde{N}(t)$] denote the (random) total number of packets present in network $\mathcal{Q}$ (resp. $\tilde{\mathcal{Q}}$). If both networks start at the same initial state, then

$$N(t) \leq_{\text{st}} \tilde{N}(t)\,, \qquad \forall t \geq 0\,. \qquad \blacksquare$$

Proof: On a sample-path basis, there holds $N(t) = B(t) - A(t)$, where $A(t)$ [resp. $B(t)$] is the number of arrivals at (resp. departures from) network $\mathcal{Q}$ during $[0,t]$; a similar relation holds for network $\tilde{\mathcal{Q}}$. Using Lemma 5.8, we have $N(t) \leq \tilde{N}(t)$ on a sample-path basis. Relaxing the coupling of the arrival processes and the routing decisions in the two networks, we obtain the stochastic inequality in question (*).

Q.E.D.

(*) $\leq_{\text{st}}$ is the symbol for stochastic domination; for two real-valued random variables V_1 and V_2, there holds $V_1 \leq_{\text{st}} V_2$ if $\Pr[V_1 > v] \leq \Pr[V_2 > v]$ for all v.

Notice that both Lemma 5.8 and Proposition 5.9 apply to *all* layered networks with Markovian routing and deterministic FIFO servers (possibly with different service times); thus, if the FIFO discipline is changed to PS, then the total number of packets in such a network increases in the stochastic sense.

5.3.3 The Stability Region of the Greedy Routing Scheme

In this subsection, we derive the stability region of the greedy routing scheme for the hypercube. First, we prove that the greedy routing scheme *balances* the traffic over the various hypercube arcs, despite the fact that arcs of different dimensions are treated differently.

Proposition 5.10: The total arrival rate at any arc of the d-cube equals $\lambda p = \rho$. ■

Proof: By symmetry among the hypercube nodes, all arcs belonging to the same dimension j have the same total arrival rate θ_j. Furthermore, the total arrival rate for the jth dimension equals $2^d \lambda p$, because each of the packets generated within the d-cube crosses the jth dimension for an expected number of p times. Hence, we have $2^d \theta_j = 2^d \lambda p$, which gives $\theta_j = \lambda p = \rho$ for all $j \in \{1, \ldots, d\}$. **Q.E.D.**

Of course, Proposition 5.10 implies that the total arrival rate at any server of the equivalent network $\mathcal{Q}$ is also ρ. The same statement applies to the network $\tilde{\mathcal{Q}}$ with the PS servers. Notice now that $\mathcal{Q}$ is an *acyclic* network with FIFO servers; thus, according to [Wal88], p. 246, network $\mathcal{Q}$ is stable if the total arrival rate for each server is less than the corresponding service rate. Recalling that all servers of $\mathcal{Q}$ have unit service rate, we reach the following conclusion:

Proposition 5.11: The greedy scheme under analysis is stable for all $\rho < 1$. ■

In light of the necessary condition for stability $\rho < 1$ (see §5.2.1), our greedy routing scheme has *optimal* stability properties. Note that the proof of Proposition 5.11 does not make use of Proposition 5.9. Nevertheless, we presented the latter result first, because it is used in the discussion to follow.

Regarding the network $\tilde{\mathcal{Q}}$ with the PS servers, our analysis so far has proved the following properties:

(a) Each server of $\tilde{\mathcal{Q}}$ is fed externally by a Poisson process. Arrival processes corresponding to different servers are independent; see Property A of §5.3.1.

(b) $\tilde{\mathcal{Q}}$ consists of PS servers, with the total arrival rate per server being ρ; service times corresponding to different packets and/or different servers are (trivially) independent.

(c) Routing among the various servers of $\tilde{\mathcal{Q}}$ is Markovian.

It follows from these properties that network $\tilde{\mathcal{Q}}$ is also stable for all $\rho < 1$; see [Wal88], pp. 93-94. Moreover, this network is of the *product-form*, meaning that the number of packets present in the various servers are independent random variables; the steady-state probability that a particular server of $\tilde{\mathcal{Q}}$ hosts n packets equals $(1-\rho)\rho^n$.

In the next subsection, we bound the average delay T per packet induced under this scheme in steady-state; this coincides with the steady-state average time spent per packet in network $\mathcal{Q}$. The *existence* of this steady-state performance measure is guaranteed (for $\rho < 1$) by Proposition 5.11. [Recall our definition of stability, namely that the sojourn time of the nth packet converges (for $n \to \infty$) to a proper random variable.] Before proceeding with the analysis for T, we clarify a fine mathematical point, regarding the steady-state properties of $\mathcal{Q}$.

Due to the assumption of Poisson arrivals, the state of network $\mathcal{Q}$ is described by the number of packets present at each server and the residual service time for the packet in "service" (that is, under transmission) at each busy server. This state-vector evolves as a homogeneous continuous-time multidimensional Markov process; certain entries of the state-vector are discrete (i.e., integer-valued), while the rest are continuous. This process is irreducible and aperiodic, and it will also be seen to be recurrent; thus, it is intuitively expected that, in the stable case (i.e., for $\rho < 1$), its steady-state distribution exists and is proper. However, to the best of our knowledge, such a technical result does not appear in the related literature, except for a stability condition for non-product form queueing networks, which is presented by Borovkov in [Bor87]. We refrain from using the result of [Bor87], because there is some doubt as to its correctness. Below, we prove the *existence* of $\lim_{t\to\infty} E[N(t)]$, by using arguments from renewal theory.

First, notice that for $\rho < 1$, network $\mathcal{Q}$ *regenerates* infinitely often. Indeed, for $\rho < 1$, the network $\tilde{\mathcal{Q}}$ with the PS servers is stable, and it *empties* infinitely often under any sample path ω. (Recall the product-form property of $\tilde{\mathcal{Q}}$.) Using Proposition 5.9 (which also holds on a sample-path basis), it is seen that network $\mathcal{Q}$ also empties

infinitely often. Whenever $\mathcal{Q}$ empties, it regenerates, because the past history of the network does not affect its future evolution (due to the memoryless property of the Poisson process); the regeneration points form a *renewal* process. By evaluating $\int N(u)du$ over the various regeneration periods (i.e, over intervals between successive regeneration points), we obtain a sequence of independent and identically distributed random variables. By the Law of Large Numbers, it follows that

$$\lim_{t\to\infty} \frac{1}{t} \int_0^t N(u)du = N \,, \tag{5.13}$$

on a sample-path basis, where N is a *constant*; using Proposition 5.9, it follows easily that N is *finite* [see also (5.16) below]. Furthermore, it is straightforward that the duration of a regeneration period of network $\mathcal{Q}$ is a *continuous* random variable, with well-defined probability density function; that is, the probability for this random variable to assume any particular value equals 0. Applying a result from renewal reward theory (see [Ros83] and [Gal90]), it follows from (5.13) that

$$\lim_{t\to\infty} E[N(t)] = \lim_{t\to\infty} \frac{1}{t} \int_0^t N(u)du = N \,;$$

thus, the *sample-path* average total number of packets present in $\mathcal{Q}$ coincides with the corresponding steady-state *ensemble* average. Reasoning similarly, it can also be showed that the limit (as $n \to \infty$) of the sample mean of the sojourn times of packets $1, \ldots, n$ equals the ensemble average T, for each sample path. Thus, Little's law applies to network $\mathcal{Q}$, and gives

$$N = (\lambda 2^d)T \,; \tag{5.14}$$

this will be used in the delay analysis to follow.

5.3.4 The Bounds for the Delay Induced by the Scheme

The main result of this subsection is the upper bound on the steady-state average delay T per packet under our greedy routing scheme; this result is presented next.

Proposition 5.12: The delay T of the greedy routing scheme under analysis satisfies

$$T \leq \frac{dp}{1-\rho} \,, \qquad \forall \rho < 1 \,. \qquad \blacksquare$$

Proof: As already mentioned in §5.3.3, the network $\tilde{\mathcal{Q}}$ with the PS servers is of the *product form*, provided that it is stable. In particular, for any $\rho < 1$, the steady-state

probability that a particular server of $\tilde{\mathcal{Q}}$ hosts n packets equals $(1-\rho)\rho^n$. Therefore, the steady-state average total number $\tilde{N}$ of packets present in $\tilde{\mathcal{Q}}$ is given by

$$\tilde{N} = d2^d \frac{\rho}{1-\rho} . \tag{5.15}$$

It is well-known that if $V_1 \leq_{st} V_2$ then $E[V_1] \leq E[V_2]$ (see [Ros83]); combining this with Proposition 5.9, we have

$$\lim_{t\to\infty} E[N(t)] \leq \lim_{t\to\infty} E[\tilde{N}(t)] ,$$

or equivalently $N \leq \tilde{N}$; this together with (5.15) implies that

$$N \leq d2^d \frac{\rho}{1-\rho} . \tag{5.16}$$

Using also Little's law [see (5.14)], we obtain

$$T \leq \frac{dp}{1-\rho} .$$

Since network $\mathcal{Q}$ is equivalent with the d-cube under our greedy routing scheme, the proof of the result is now complete. **Q.E.D.**

Next, we comment on the number of packets stored per hypercube *node*. The steady-state average such number equals $\frac{N}{2^d}$; this quantity does not exceed $d\frac{\rho}{1-\rho}$, as implied by (5.16). Thus, it is seen that, for any fixed ρ, the average size of the queue built at each node is $O(d)$. In fact, one can show that the total number of packets within the d-cube is $O(d2^d)$ with high probability. Indeed, by Proposition 5.9 and the product-form property of $\tilde{\mathcal{Q}}$, for $t \to \infty$, the random variable $N(t)$ is stochastically dominated by the sum of $d2^d$ independent geometrically distributed random variables with expected value $\frac{\rho}{1-\rho}$. Using the Chernoff bound, it follows that, for $t \to \infty$, $N(t) \leq d2^d \frac{\rho}{1-\rho}(1+\epsilon)$ with high probability, for any $\epsilon > 0$.

Next, we present a lower bound on the delay T.

Proposition 5.13: The delay T of the greedy routing scheme under analysis satisfies

$$T \geq dp + p\frac{\rho}{2(1-\rho)} , \qquad \forall \rho < 1 . \qquad \blacksquare$$

Proof: Each arc of the 1st hypercube dimension is only fed by a Poisson stream with rate $\rho < 1$; using the expression for the average size of an $M/D/1$ queue (see

[Kle75]), it follows that a packet to cross this dimension is delayed by an average of $1 + \frac{\rho}{2(1-\rho)}$. Recall now that each packet crosses the jth dimension with probability p, for $j = 1, \ldots, d$. Taking into account the average delay induced by the 1st dimension and *only* the average *propagation* time for each of dimensions $2, \ldots, d$ (which equals p), it follows easily that

$$T \geq (d-1)p + p\left[1 + \frac{\rho}{2(1-\rho)}\right] ;$$

this immediately proves the result. **Q.E.D.**

It should be noted that another lower bound for T follows from Proposition 5.3, which is applicable because our routing scheme is *oblivious*; the lower bound of Proposition 5.13 is sharper by a factor of at most 2.

As a final comment, notice that the network $\tilde{\mathcal{Q}}$ with the PS servers has the *same* steady-state distribution as the *Jackson* network $\hat{\mathcal{Q}}$ obtained from $\mathcal{Q}$ by replacing the deterministic servers with *exponential* ones (with unit service rate). However, a result analogous to Proposition 5.9 would not hold if we had considered network $\hat{\mathcal{Q}}$ (instead of $\tilde{\mathcal{Q}}$). Even though the steady-state average delay of Jackson network $\hat{\mathcal{Q}}$ also equals $\frac{dp}{1-\rho}$, we believe that Proposition 5.12 cannot be established easily by comparing these two networks; our arguments related to this point are presented in Appendix 5.A.

5.3.5 Asymptotic Behavior of the Average Delay

Proposition 5.12 implies that $T = \Theta(d)$ for any fixed ρ, which is the optimal order of magnitude. Also, combining Propositions 5.12 and 5.13, we obtain

$$\frac{p}{2} \leq \lim_{\rho \to 1}[(1-\rho)T] \leq dp ,$$

which implies that under heavy traffic (i.e., for $\rho \to 1$) the delay induced by the greedy routing scheme behaves as $\frac{1}{1-\rho}$, for any fixed d. According to the discussion in §5.2.2 (following the proof of Proposition 5.2), this asymptotic behavior of T is also optimal.

As already mentioned in §5.1.1, the average *propagation delay* per packet equals dp, which is independent of ρ. Hence, the steady-state average *queueing* delay per packet equals $T - dp$; using Propositions 5.12 and 5.13, we have

$$p\frac{\rho}{2(1-\rho)} \leq T - dp \leq \rho\frac{dp}{1-\rho} . \tag{5.17}$$

Notice that, for fixed ρ, the upper bound for $T - dp$ is $\Theta(d)$ while the lower bound is $\Theta(1)$. We can actually close this gap, by proving that

$$T - dp \geq [(d-1)\rho]p^2(1-p)\,, \qquad \forall p \in (0,1)\,. \tag{5.18}$$

The underlying idea for proving this result is as follows: For $p \in (0,1)$, a packet $\mathcal{P}$ faces *additional* contention for each arc of dimensions $2, \ldots, d$ it crosses; that is, when $\mathcal{P}$ crosses an arc of a dimension $j \geq 2$, it contends with packets that had not entered the path of $\mathcal{P}$ up to this point. It can be proved that the probability for *at least one* collision per dimension $j \geq 2$ is bounded from below by a fraction of ρ (depending only p); using this, (5.18) follows after some algebra. We refrain from presenting the technical details, because the rigorous argument should first be carried out in slotted time, and then extended to the case of continuous time [by means of (5.26), which is proved in §5.4]. A similar (yet simpler) argument is given in §5.6.4, applying to the butterfly network; see Proposition 5.18.

The lower bound of (5.18) is not tight for heavy traffic, because it does not grow to infinity for $\rho \to 1$. This can be remedied by including the queueing delay for the 1st dimension in the left-hand quantity of (5.18); thus, we obtain

$$T - dp \geq [(d-1)\rho]p^2(1-p) + p\frac{\rho}{2(1-\rho)}\,, \qquad \forall p \in (0,1)\,.$$

Nevertheless, this bound in *not* within a constant factor from the upper bound of (5.17). We *conjecture* that, for all $p \in (0,1)$, the latter bound is tight; namely, we conjecture that $T \geq \alpha\frac{dp}{1-\rho}$, where α depends only on p (and $\alpha < 1$). On the other hand, for $p = 1$, it is easily seen that the *lower* bound in (5.17) is tight. Indeed, in this case, each packet generated at node x is destined for node $\bar{x}$, where each entry of the binary identity of $\bar{x}$ is the complement of the corresponding entry of x. As was proved in §4.6.2, the canonical paths from x to $\bar{x}$ and from y to $\bar{y}$ are *disjoint* (for $y \neq x$). Thus, for $p = 1$, packets generated at different nodes follow disjoint paths; this easily gives that $T = d + \frac{\rho}{2(1-\rho)}$.

5.4 THE CASE OF SLOTTED TIME

In the analysis so far, it was assumed that the time axis is *continuous*. In the present section, we extend our results to the case of *slotted* time. In particular, we assume that the time axis is divided in slots of duration τ; all nodes are synchronized to the same clock. Since packets are taken to have unit length, we may assume, without loss of generality, that $\tau \leq 1$ and, in particular, that $\frac{1}{\tau}$ is integer; otherwise, there will be some waste due to the fact that packets do not "fit" exactly to time slots. (Of course, the simplest case is $\tau = 1$, but the more general one is not any harder.) Furthermore, it is assumed that each node of the hypercube generates a new *batch* of packets at the beginning of each slot, namely at each time $k\tau$ with $k \in \{0, 1, \ldots\}$. The batch size has Poisson distribution with expected value $\lambda\tau$; thus, the rate per node of generating packets is the same as in the case of continuous time. Notice that batches generated at different times and/or different nodes have independent sizes. Again, each packet has a single destination, which is chosen according to the rule used in the case of continuous time [see (5.1)]. The system's state is recorded at times $k\tau+$, while transmissions starting at time $k\tau$ are assumed to terminate at time $(k + \frac{1}{\tau})\tau$. (Recall that $\frac{1}{\tau}$ is integer.) Thus, a packet starting transmission immediately upon arrival, will be recorded as present in the arc for exactly $\frac{1}{\tau}$ time slots, as required. Finally, packets are to be routed to their respective destinations under the greedy scheme analyzed in the previous section. Again, when several packets contend for the same arc $(y, y \oplus e_j)$, priority is allotted to the one that arrived at node y the first. Since time is slotted, ties may often occur; it is assumed that a tie (among the highest priority packets) is resolved in a fair randomized way.

In order to analyze the routing problem under the new assumptions, we consider the slotted-time version of the network $\mathcal{Q}$ defined in §5.3.1. The new network will be denoted by $\overline{\mathcal{Q}}$; it has the same properties as $\mathcal{Q}$, except for the fact that packets are generated in the way presented above and that all events occur in discrete time. It can be easily seen that the total arrival rate per server of $\overline{\mathcal{Q}}$ equals ρ. Thus, reasoning as in the case of continuous time (see §5.3.3), it follows that stability applies for all $\rho < 1$, which is the broadest possible region. (The inequality $\rho < 1$ is still a necessary condition for stability, because the argument used in §5.2.1 also applies to slotted time.)

Furthermore, we consider a sample path ω of network $\mathcal{Q}$ (see §5.3.2); it is apparent that a sample path $\overline{\omega}$ of $\overline{\mathcal{Q}}$ can be obtained from ω as follows: For each server $(x, x \oplus e_i)$, we take the external arrivals occurring under ω during the interval $[k\tau, (k+1)\tau)$ and we consider them as the external arrivals occurring at $(x, x \oplus e_i)$ at time $k\tau$ under the sample path $\overline{\omega}$. Thus, all external arrival streams are *advanced* in time.

We now assume that the two networks $\mathcal{Q}$ and $\overline{\mathcal{Q}}$ are coupled in the way presented above. Let $\overline{B}(k\tau)$ be the number of departures from network $\overline{\mathcal{Q}}$ up to and *including* time $k\tau$. Also, let $\overline{A}(k\tau)$ be the number of external arrivals at $\overline{\mathcal{Q}}$ up to and *including* time $k\tau$. Finally, let $\overline{N}(k\tau+)$ be the total number of packets present in $\overline{\mathcal{Q}}$ at time $k\tau+$; notice that

$$\overline{N}(k\tau+) = \overline{A}(k\tau) - \overline{B}(k\tau)\,; \tag{5.19}$$

also, by the definitions introduced in the proof of Proposition 5.10, we have

$$N(k\tau) = A(k\tau) - B(k\tau)\,. \tag{5.20}$$

Furthermore, applying Lemma 5.5 on a level-by-level basis, starting from the 1st level of servers, it is seen that $\overline{B}(k\tau) \geq B(k\tau)$, because all streams arising in the context of $\overline{\mathcal{Q}}$ are advanced versions of the corresponding streams in $\mathcal{Q}$. Notice that this result relies on the fact that $\frac{1}{\tau}$ is *integer*; otherwise, there would be idling in $\overline{\mathcal{Q}}$ for some fraction of the time, and thus the "comparison" would not have been possible. Therefore, it follows from (5.19) and (5.20) that

$$\overline{N}(k\tau+) \leq N(k\tau) + \overline{A}(k\tau) - A(k\tau)\,. \tag{5.21}$$

By construction of sample path $\overline{\omega}$ in slotted time, it is seen that $\overline{A}(k\tau) - A(k\tau)$ equals the total number X_k of external arrivals occurring in continuous time during the interval $[k\tau, (k+1)\tau)$; this together with (5.21) implies that

$$\overline{N}(k\tau+) \leq N(k\tau) + X_k\,; \tag{5.22}$$

note that the random variable X_k is *independent* of $N(k\tau)$ and assumes the Poisson distribution with expected value $(\lambda 2^d)\tau$.

An alternative sample path $\overline{\omega}'$ in slotted time may be constructed as follows: For each server $(x, x \oplus e_i)$, we take the external arrivals occurring under the continuous-time sample path ω during the interval $[(k-1)\tau, k\tau)$; these arrivals are considered

as the ones occurring at $(x, x \oplus e_i)$ at time $k\tau$ under the sample path $\overline{\omega}'$. Thus, all external arrival streams are now *delayed* in time. Notice that $\overline{\omega}'$ is the *same* as sample path $\overline{\omega}$ considered previously, except for the fact that all events of $\overline{\omega}$ are *shifted* by one slot to the right. Using this and reasoning exactly as in proving (5.22), it follows that

$$N\big((k+1)\tau\big) \leq \overline{N}(k\tau+)\,. \tag{5.23}$$

As in the case of continuous time, the state of network $\overline{\mathcal{Q}}$ is described by the number of packets present at each server and the residual service time for the packet in "service" at each busy server. However, the state-vector now evolves as a homogeneous Markov chain with *countable* state-space. (Since time is slotted, the "residual" service times may only take finitely many values.) Clearly, this Markov chain is irreducible and aperiodic; moreover, (5.22) implies that if $\rho < 1$, then $\overline{N}(k\tau+)$ is finite with probability 1, because the same property applies to $N(k\tau)$. Therefore, using a well-known result on Markov chains with countable state-space (e.g., see [Ros83], p. 109), it follows that the steady-state distribution of the state-vector of $\overline{\mathcal{Q}}$ *exists* and is *proper* for all $\rho < 1$.

Henceforth, we assume that stability applies. Let $\overline{N}$ (resp. $\overline{T}$) denote the steady-state average total number of packets (resp. average delay per packet) for network $\overline{\mathcal{Q}}$. By Little's law, we have

$$\overline{N} = (\lambda 2^d)\overline{T}\,. \tag{5.24}$$

On the other hand, using (5.22) and (5.23), it follows that

$$\lim_{k\to\infty} E[N\big((k+1)\tau+\big)] \leq \lim_{k\to\infty} E[\overline{N}(k\tau+)] \leq \lim_{k\to\infty} E[N(k\tau+)] + \lim_{k\to\infty} E[X_k]\,,$$

or equivalently

$$N \leq \overline{N} \leq N + (\lambda 2^d)\tau\,, \tag{5.25}$$

where we have also used the fact $E[X_k] = (\lambda 2^d)\tau$. Combining (5.24) with (5.14) and (5.25), we finally obtain

$$T \leq \overline{T} \leq T + \tau\,. \tag{5.26}$$

Therefore, the delay induced by the greedy routing scheme in slotted time has similar asymptotic properties as that in continuous time (both for $\rho \to 1$ and for $d \to \infty$). Also, the number of packets stored per node has similar properties as in the case of continuous time.

Finally, similar results apply when it is assumed that new packets are generated in continuous time while transmissions occur in discrete time. The new value of the steady-state average delay per packet is $\overline{T} + \frac{\tau}{2}$; the term $\frac{\tau}{2}$ equals the average synchronization time (see §2.2.2).

5.5 OPEN PROBLEMS

In §5.3.5, we discussed an open question related to the lower bound for the delay induced by the greedy routing scheme. In this section, we present open questions related to other schemes that can also be used in our routing problem.

In the greedy scheme analyzed in §5.3, there is only one permissible path for each origin-destination pair (namely, the corresponding canonical path). A natural extension of this scheme is to allow each packet to choose from *multiple shortest paths* leading to its destination. This selection would be randomized; again, packets would traverse their respective paths as fast as possible, subject to contention. For such a scheme to be efficient, the rule for path selection should *balance* the traffic among the various hypercube arcs. (e.g., the traffic is balanced when all shortest paths for each origin-destination pair are permissible and equiprobable.) It is reasonably expected that all greedy *shortest-path* schemes with balanced traffic are efficient, in the sense that they meet the performance objectives set in §5.1.2. Such a result appears to be rather hard to prove. In particular, with multiple permissible paths, the corresponding equivalent queueing network is *not* layered; moreover, packets do *not* propagate in a Markovian fashion, which complicates the analysis even further.

So far, we have only considered oblivious schemes. We now turn our attention to *adaptive* schemes; for simplicity, we assume that time is slotted. The "ultimate" adaptive scheme is *deflection* routing, where, after departing from its origin, each packet $\mathcal{P}$ travels *continuously* until it reaches its respective destination z. At each intermediate node y, packet $\mathcal{P}$ attempts to traverse one of the arcs belonging to a *shortest* path from y to z, in order to *reduce* its Hamming distance from the destination z; such arcs are referred to as *preferred* ones. If no preferred arc is available, $\mathcal{P}$ traverses one of the remaining arcs, and its Hamming distance from z *increases*. As already mentioned in §1.5.2, deflection routing has only been analyzed numerically or approximately in the previous literature. Notice that, under this routing scheme,

the number of packets *in transit* cannot exceed the number of hypercube arcs. This property allows for modelling the system as a *finite-state* Markov chain, and derive either numerical or approximate estimates of the various performance measures. On the other hand, few results have been derived in closed-form expressions, because the analysis is very complicated; thus, derivation of explicit results on deflection routing is a very hard open problem in the routing literature.

Consider now the following variation of deflection routing: packets that cannot traverse a preferred arc are *stored*, rather than misrouted. Thus, the Hamming distance from a packet's location to its destination *never* increases (as might occur under deflection routing). In order to define the scheme completely, we also have to specify the rule of assigning packets to their preferred arcs; similar to [Var90], we assume that packets at *smaller* distance from their respective destinations have *priority* over the ones at larger distance. The rationale for this is that the former packets have fewer preferred arcs, and thus when they contend for one of them they should be favored, because they have less alternatives than packets to travel at larger distance. It is intuitively clear that the adaptive scheme described above is very efficient; indeed, given the state of the system at each particular time, the number of packets advancing towards their respective destinations is very close (if not equal) to the maximum possible. In Figure 5.3, we compare simulation outcomes for the adaptive scheme to the ones for the greedy scheme analyzed in §5.3, for the case of uniform destination distribution ($p = \frac{1}{2}$). It is seen that the average queueing delay $T - dp$ per packet is very small under the adaptive scheme, even though the simulations correspond to very heavy traffic (namely, $\rho = 0.90$). In fact, under the adaptive scheme, $T - dp$ does *not* seem to increase with d, which would be a very interesting property of the scheme. Unfortunately, the adaptive scheme is very hard to analyze; thus, evaluation of its performance appears to be a very challenging open question.

Unfortunately, most of the open problems presented above are rather hard. A seemingly more tractable one is to analyze our routing problem under an arbitrary *destination distribution.* For this case, it may be profitable to "mix" the packets by first sending each of them to a random intermediate node, as is done for the permutation task in [VaB81] and [Val82]. Such a "mixing" may result in improved delay properties under medium traffic, at the expense of reducing the maximum traffic that may be sustained by the system.

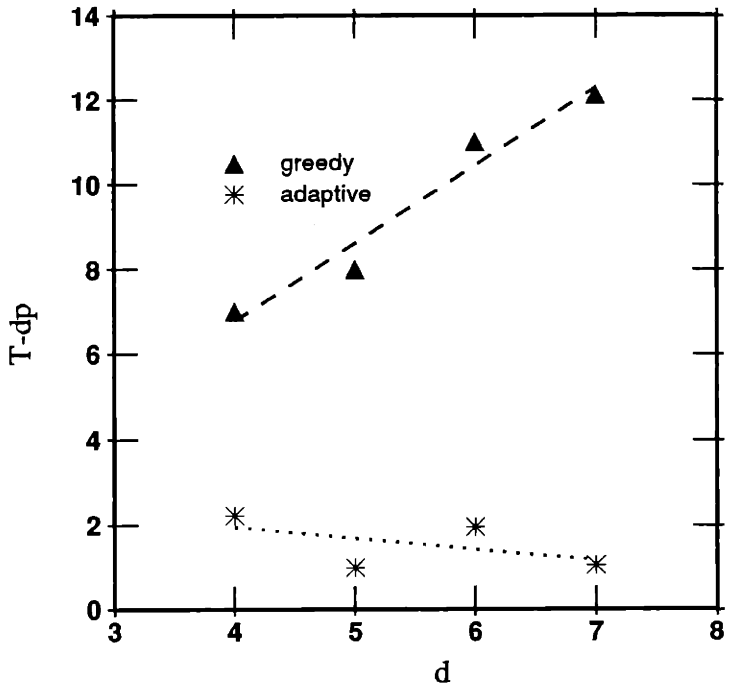

Figure 5.3: Comparing the average queueing delay per packet under the greedy and the adaptive schemes, for $\rho = 0.90$.

5.6 GREEDY ROUTING ON THE BUTTERFLY NETWORK

In this section, we extend the results derived for the hypercube to the butterfly network. First, we briefly describe the basic properties of this network.

5.6.1 The Butterfly Network

The d-dimensional butterfly is an "unfolded" version of the d-cube. It consists of $(d+1)2^d$ nodes, organized in $d+1$ levels, with each level having 2^d nodes. In particular, for $j \in \{1, \ldots, d+1\}$, the nodes of the jth level are denoted by $[x; j]$ where $x \in \{0, \ldots, 2^d - 1\}$. For $j \neq d+1$, each node $[x; j]$ is connected to two nodes, namely $[x; j+1]$ and $[x \oplus e_j; j+1]$; see Figure 5.4, where the 2-butterfly is depicted. Therefore, there exist two types of arcs:

(a) Arcs of the form $[x; j] \rightarrow [x; j+1]$, which are referred to as *straight* arcs; for notational convenience, arc $[x; j] \rightarrow [x; j+1]$ will be denoted by $(x; j; \mathrm{s})$.

(b) Arcs of the form $[x; j] \rightarrow [x \oplus e_j; j+1]$, which are referred to as *vertical* arcs; for notational convenience, arc $[x; j] \rightarrow [x \oplus e_j; j+1]$ will be denoted by $(x; j; \mathrm{v})$.

The butterfly network is a crossbar switch; packets are assumed to be generated at the 1st level and destined for the $(d+1)$st level. It is easily seen that for each origin-destination pair $[x;1]$ and $[z;d+1]$ there corresponds a *unique* path, which consists of d arcs. In particular, let $i_1, \ldots, i_k$ be the entries in which the binary identities of x and z differ, with $i_1 < i_2 < \cdots < i_k$; then, the path from $[x;1]$ to $[z;d+1]$ contains exactly k vertical arcs, namely

$$(x; i_1; \mathrm{v}), (x \oplus e_1; i_2; \mathrm{v}), \ldots, (x \oplus e_1 \cdots \oplus e_{i_{k-1}}; i_k; \mathrm{v});$$

the remaining $d-k$ arcs of the path are straight arcs. Notice that these k vertical arcs correspond to the arcs of the canonical path from node x to node z in the d-cube.

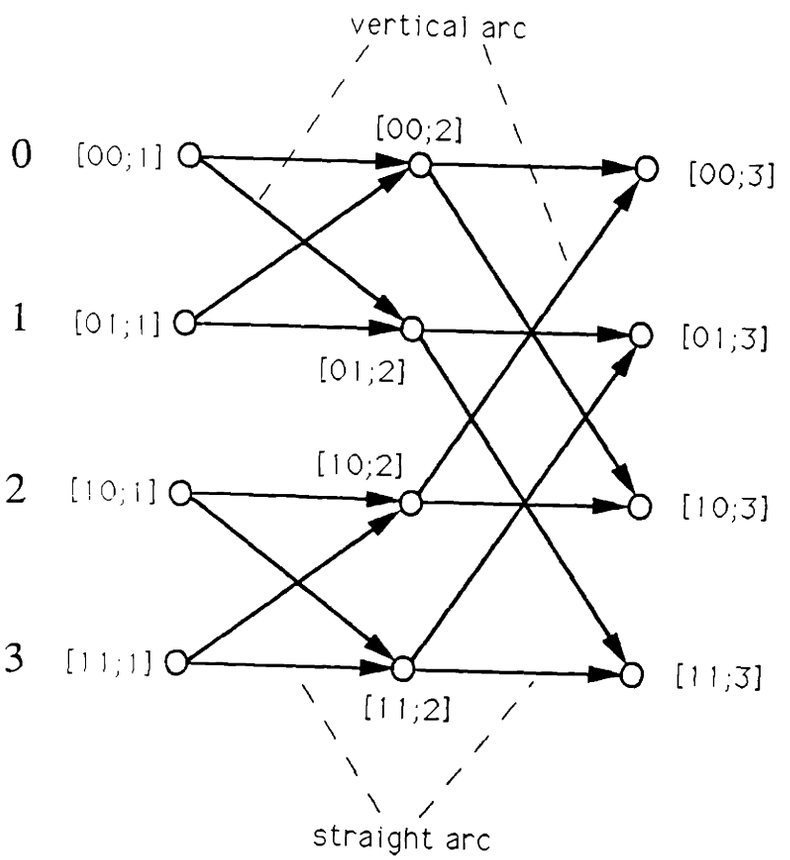

Figure 5.4: The 2-dimensional butterfly.

5.6.2 The Routing Problem

The dynamic routing problem to be analyzed is essentially the same as that in the context of the d-cube. That is, each node of the 1st level independently generates packets according to a Poisson process with rate λ; all packets have unit transmission time. Each packet has a single destination in the $(d+1)$st level; this destination is selected randomly, according to the following rule:

$$\Pr\left[\text{a packet generated by node } [x;1] \text{ is destined for node } [z;d+1]\right] = p^{H(x,z)}(1-p)^{d-H(x,z)} ,$$

where $p \in [0,1]$ is a constant; recall that $H(x,z)$ denotes the Hamming distance of the binary representations of x and z. Again, different packets make their selections independently of each other. Notice that, for $p = \frac{1}{2}$, the destination distribution is uniform over the nodes of the $(d+1)$st level; that is, each such node is equally likely to be chosen as a packet's destination.

5.6.3 Preliminary Results

First, we note that a result analogous to Lemma 5.1 applies to the present context; however, $\mathcal{B}_j$ now corresponds to the event that a packet has to traverse a *vertical* arc stemming from the jth level. Furthermore, notice that arcs $(x;1;\mathrm{s})$ and $(x;1;\mathrm{v})$ may only be traversed by packets generated by node x. Therefore, packets to traverse arc $(x;1;\mathrm{v})$ form a Poisson stream with rate λp; similarly, packets to traverse arc $(x;1;\mathrm{s})$ form a Poisson stream with rate $\lambda(1-p)$. Recalling that all packets have unit transmission time, it follows that the inequalities $\lambda p < 1$ and $\lambda(1-p) < 1$ are both *necessary* conditions for stability of *any* routing scheme. Combining these conditions, we obtain the following result: Stability may prevail only if

$$\rho \stackrel{\text{def}}{=} \lambda \max\{p, 1-p\} < 1 . \tag{5.27}$$

Notice that, for given λ, the maximum value of ρ occurs for $p = \frac{1}{2}$. For $p > \frac{1}{2}$, the vertical arcs become the bottleneck of the system; for $p < \frac{1}{2}$, the straight arcs become the bottleneck of the system; this will become more clear in Proposition 5.15.

Next, we present a *universal* lower bound on the average delay T per packet.

Proposition 5.14: Under *any* routing scheme, there holds

$$T \geq d + p\frac{\lambda p}{2(1-\lambda p)} + (1-p)\frac{\lambda(1-p)}{2[1-\lambda(1-p)]} . \qquad \blacksquare$$

Proof: When no idling occurs, the value W_{v} (resp. W_{s}) of the average delay induced by arc $(x;1;\mathrm{v})$ [resp. $(x;1;\mathrm{s})$] equals that of an $M/D/1$ queue with arrival rate λp [resp. $\lambda(1-p)$] and unit service duration; when idling occurs, these delay values are larger. Thus, we have [Kle75]

$$W_{\mathrm{v}} \geq 1 + \frac{\lambda p}{2(1-\lambda p)} \qquad \text{and} \qquad W_{\mathrm{s}} \geq 1 + \frac{\lambda(1-p)}{2[1-\lambda(1-p)]} \,. \tag{5.28}$$

Note that after a packet arrives at the 2nd level, it requires at least $d-1$ more time units until it reaches its destination; thus, it is seen that, under any routing scheme, the average delay T per packet satisfies

$$T \geq d - 1 + pW_{\mathrm{v}} + (1-p)W_{\mathrm{s}} \,.$$

This together with (5.28) proves the reult. **Q.E.D.**

Equation (5.27) as well as Proposition 5.14 demonstrate the limitations applying to the performance of any routing scheme. The scheme to be analyzed below is the simplest possible:

Packets are routed in a *greedy* fashion; that is, each packet advances at its respective path as fast as possible. When several packets contend for the same arc, then priority is allotted on a FIFO basis.

In fact, given that there is only one path per origin-destination pair, greedy routing is the most natural scheme arising in the context of the butterfly. It will be shown in §5.6.4 that this simple scheme is very efficient.

5.6.4 Performance Analysis of Greedy Routing

Similar to the hypercube (see §5.3.1), under greedy routing, the butterfly may be viewed as a queueing network $\mathcal{R}$ with $d2^{d+1}$ deterministic FIFO "servers"; each of them has unit service duration and corresponds to an *arc*. In Figure 5.5, we present the network $\mathcal{R}$ corresponding to the 2-dimensional butterfly. The main properties of the equivalent network $\mathcal{R}$ are as follows:

Property A: $\mathcal{R}$ is a *layered* network; it consists of d levels, with the jth level comprising all arcs $(x;j;\mathrm{s})$ and $(x;j;\mathrm{v})$. In fact, each packet receives one time unit of "service" at each level, contrary to the network described in §5.3.1, where a packet might skip some of the levels.

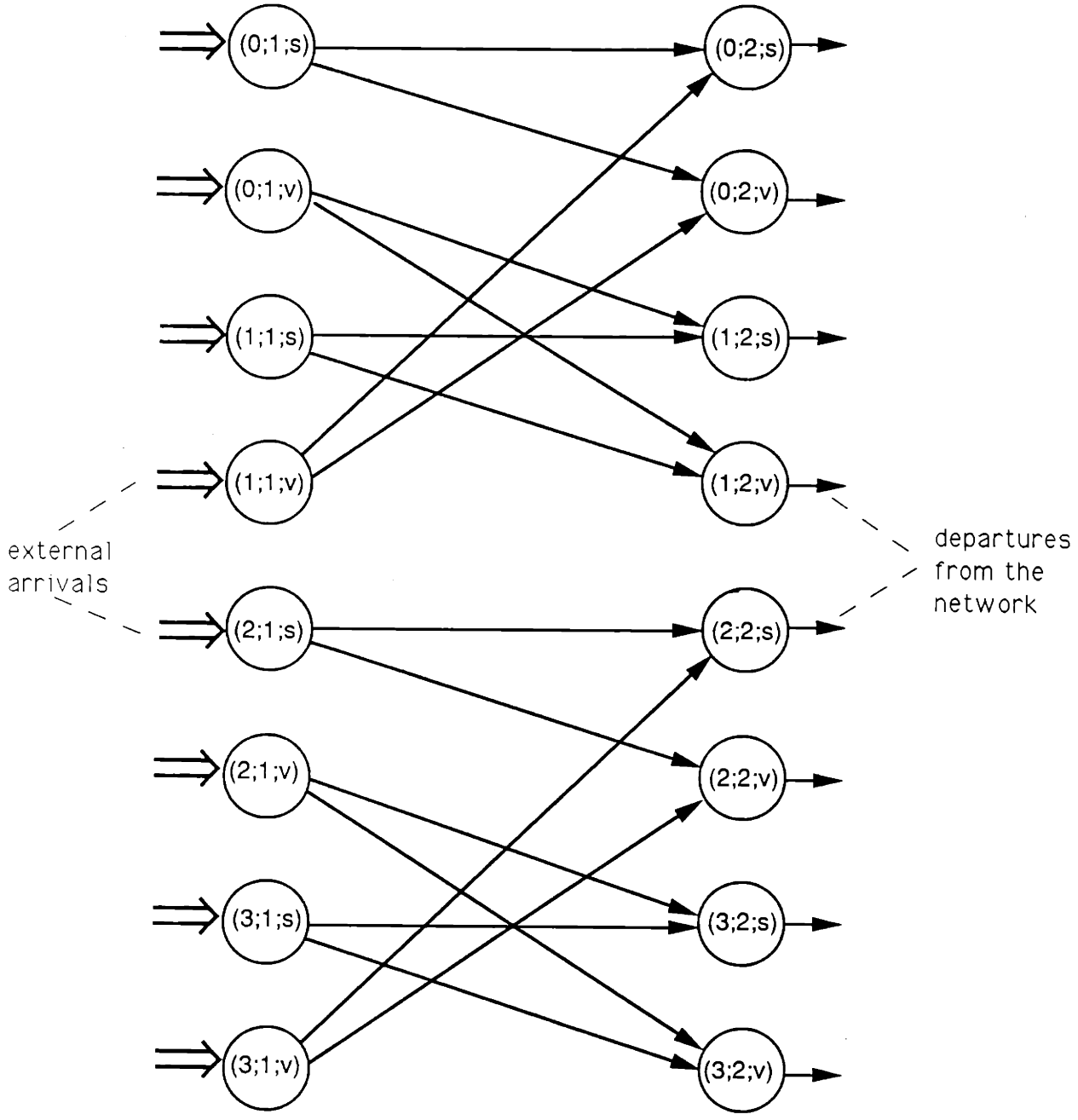

Figure 5.5: The equivalent queueing network $\mathcal{R}$ for the 2-dimensional butterfly.

Property B: Routing is *Markovian.* In particular, after traversing arc $(y;j;\mathrm{s})$ [resp. $(y;j;\mathrm{v})$], where $j \neq d$, a packet takes one of the following two actions: either it joins the queue for arc $(y;j+1;\mathrm{s})$ [resp. $(y \oplus e_j;j+1;\mathrm{s})$] with probability $1-p$; or it joins the queue for arc $(y;j+1;\mathrm{v})$ [resp. $(y \oplus e_j;j+1;\mathrm{v})$] with probability p. After crossing arc $(y;d;\mathrm{s})$ [resp. $(y;d;\mathrm{v})$], a packet departs from the network with probability 1. Different packets make their routing decisions independently of each other.

This property may be established by reasoning as in Lemma 5.4; recall that an independence result, analogous to Lemma 5.1, applies to the butterfly as well (see §5.6.3). Another result to be used in the analysis is as follows:

Proposition 5.15: The total arrival rate at each arc $(x;j;\mathrm{s})$ equals $\theta_{\mathrm{s}} = \lambda(1-p)$. Also, the total arrival rate at each arc $(x;j;\mathrm{v})$ equals $\theta_{\mathrm{v}} = \lambda p$. ■

Proof: We fix some $j \in \{1,\ldots,d\}$. By symmetry, all straight (resp. vertical) arcs of the jth level have the same total arrival rate $\theta_{\mathrm{s}}^{(j)}$ [resp. $\theta_{\mathrm{v}}^{(j)}$]. As already mentioned, each packet crosses some straight (resp. vertical) arc of the jth level with probability $1-p$ (resp. p); also, the total arrival rate over all arcs of the jth level equals $\lambda 2^d$, because each packet crosses exactly one arc of this level. Thus, we obtain $\lambda 2^d(1-p) = 2^d\theta_{\mathrm{s}}^{(j)}$ and $\lambda 2^d p = 2^d\theta_{\mathrm{v}}^{(j)}$, which proves the result. **Q.E.D.**

It is seen from Proposition 5.15 that, for $p < \frac{1}{2}$, straight arcs are more congested than the vertical ones; the converse holds for $p > \frac{1}{2}$. For $p = \frac{1}{2}$, all arcs are equally congested.

Furthermore, since $\mathcal{R}$ is an *acyclic* network with Markovian routing and FIFO servers, we can apply the stability result of [Wal88], p. 246. Thus, we reach the following conclusion:

Proposition 5.16: Greedy routing on the butterfly is stable if

$$\lambda p < 1 \qquad \text{and} \qquad \lambda(1-p) < 1\,,$$

or equivalently if $\rho \stackrel{\text{def}}{=} \lambda \max\{p, 1-p\} < 1$. ■

In light of the necessary condition for stability in (5.27), it is seen that greedy routing in the butterfly has *optimal* stability properties.

As in §5.3, we evaluate the performance measures of network $\mathcal{R}$ by comparing it to a product-form network. In particular, we also consider network $\tilde{\mathcal{R}}$, which is identical

to $\mathcal{R}$ except for the fact that all of its servers operate under the *Processor Sharing* (PS) discipline. The total arrival rate for each server of $\tilde{\mathcal{R}}$ is the same as in $\mathcal{R}$, and is given by Proposition 5.15. Therefore, network $\tilde{\mathcal{R}}$ is stable if $\lambda p < 1$ and $\lambda(1-p) < 1$, or equivalently if $\rho < 1$ [see (5.27)]; in the stable case, $\tilde{\mathcal{R}}$ is of the *product form* [Wal88], pp. 93-94. Moreover, Proposition 5.9 applies to the present context, because $\mathcal{R}$ is a layered network with Markovian routing and deterministic FIFO servers; see also the comment on the generality of that result, following its proof. Below, we establish the upper bound for the average delay T per packet induced by greedy routing.

Proposition 5.17: There holds

$$T \leq \frac{dp}{1-\lambda p} + \frac{d(1-p)}{1-\lambda(1-p)}\,, \qquad \forall \rho < 1\,. \qquad \blacksquare$$

Proof: Reasoning as in the end of §5.3.3, it follows that

$$\lim_{t\to\infty} E[N(t)] = N = \lim_{t\to\infty} \frac{1}{t}\int_0^t N(u)du\,,$$

where $N(t)$ is the total number of packets present in the equivalent network $\mathcal{R}$ at time t, and N is the steady-state average of this. Again, applying Little's law, we have

$$T = \frac{N}{\lambda 2^d}\,. \tag{5.29}$$

On the other hand, using Proposition 5.9, we obtain

$$N \leq \tilde{N}\,, \tag{5.30}$$

where $\tilde{N}$ is the steady-state average total number of packets for $\tilde{\mathcal{R}}$. As already mentioned, for $\rho < 1$, network $\tilde{\mathcal{R}}$ is of the product form; this together with Proposition 5.15 implies that the steady-state probability that a particular "server" $(x; j; \mathrm{v})$ [resp. $(x; j; \mathrm{s})$] of $\tilde{\mathcal{R}}$ hosts n packets equals $(1-\lambda p)(\lambda p)^n$ (resp. $[1-\lambda(1-p)][\lambda(1-p)]^n$). Since there exist $d2^d$ "servers" of each of the two types, it follows that

$$\tilde{N} = d2^d \frac{\lambda p}{1-\lambda p} + d2^d \frac{\lambda(1-p)}{1-\lambda(1-p)}\,. \tag{5.31}$$

This together with (5.29) and (5.30) proves the result. **Q.E.D.**

Using Propositions 5.14 and 5.17 and the definition of ρ it follows that

$$\frac{1}{2}\max\{p, 1-p\} \leq \lim_{\rho\to 1}[(1-\rho)T] \leq d\max\{p, 1-p\}\,.$$

Thus, for $\rho \to 1$, the average delay T behaves as $\frac{1}{1-\rho}$ (for fixed d); in light of the universal lower bound of Proposition 5.14, this asymptotic behavior is optimal. Also, for any fixed ρ, we have $T = \Theta(d)$, which is the optimal order of magnitude. Furthermore, notice that the average *queueing* delay per packet equals $T - d$. Proposition 5.14 implies that, for fixed ρ, $T - d$ is $\Omega(1)$, while Proposition 5.17 implies that $T - d$ is $O(d)$; the result to be present next closes this gap for $p \neq 0$ and $p \neq 1$.

Proposition 5.18: There holds

$$T - d \geq \frac{1}{2}(d-1)\rho \min\{p, (1-p)\}\,, \qquad \forall p \in (0,1)\,. \qquad \blacksquare$$

Proof: First, notice that the analysis of §5.4 for the case of *slotted* time applies exactly as presented therein. In particular, using (5.26), we have

$$T \geq \overline{T}\,. \tag{5.32}$$

Henceforth, we focus on the case of slotted time, and we assume that the system is in steady-state; we also take $p \neq 0$ and $p \neq 1$.

We consider a straight arc $(x; j; \mathrm{s})$ with $j \geq 2$; the case of a vertical arc may be treated similarly. Notice that, at each slot k, there are at most two packets joining the queue for this arc; namely, at most one packet just trasmitted over arc $(x; j-1; \mathrm{s})$, and at most one just trasmitted over $(x \oplus e_j; j-1; \mathrm{v})$. Notice that the two streams of packets feeding arc $(x; j; \mathrm{s})$ are mutually *independent*, due to the following fact: No packet ever collides with packets from both of these streams. Indeed, if this were the case, then there would be *two* different directed paths connecting a pair of nodes of the butterfly, which does not apply. Consider now a packet $\mathcal{P}$ joining the queue for $(x; j; \mathrm{s})$ at slot k through arc $(x; j-1; \mathrm{s})$; it follows from the discussion above that $\mathcal{P}$ sees the contending stream emerging from arc $(x \oplus e_j; j-1; \mathrm{v})$ at a *typical* time. This implies that $\mathcal{P}$ will experience a *simultaneous* arrival from the contending stream with probability equal to the rate of this stream, namely $\lambda p(1-p)$. (Recall Proposition 5.15 and Property B in the beginning of this subsection.) In such a case, $\mathcal{P}$ will suffer a unit of *queueing* delay with probability $\frac{1}{2}$, because contention between simultaneous arrivals is resolved on a fair basis.

Generalizing the above conclusion, it is seen that every packet suffers an average *queueing* delay of at least $\frac{1}{2}\lambda p(1-p)$ at *each* arc of levels $2, \ldots, d$. Thus, it follows

that

$$\overline{T} - d \geq \frac{1}{2}(d-1)\lambda p(1-p)\,. \tag{5.33}$$

This together with (5.32) and the definition of ρ proves the result. **Q.E.D.**

The lower bound of Proposition 5.18 is not tight in heavy traffic, because it does not grow to infinity for $\rho \to 1$. This can be remedied by including the queueing delay for the 1st level in the left-hand quantity of (5.33); thus, we obtain $T - d \geq \alpha_1(d\rho) + \alpha_2 \frac{\rho}{1-\rho}$, where α_1 and α_2 only depend on p. On the other hand, Proposition 5.17 shows that $T - d \leq \beta \frac{d\rho}{1-\rho}$, with β only depending on p. Similar to the case of the hypercube (see the end of §5.3.5), it is conjectured that the *upper* bound is tight for all $p \in (0,1)$; for $p = 0$ and for $p = 1$, the lower bound of Proposition 5.14 is tight, because packets originating at different nodes follow *disjoint* paths.

Finally, we comment on the number of packets stored per node of the butterfly; first, notice that only the nodes of levels $1, \ldots, d$ have to store packets. For $t \to \infty$, an overall estimate of the average number of packets per node at time t is provided by the quantity $\frac{N}{d2^d}$; using (5.30) and (5.31), it follows that

$$\frac{N}{d2^d} \leq \frac{\lambda p}{1-\lambda p} + \frac{\lambda(1-p)}{1-\lambda(1-p)} \stackrel{\text{def}}{=} q_\rho\,.$$

This estimate is quite favorable because it suggests that the "overall" average queue-size per node is $O(1)$ for any fixed ρ. However, it is not guaranteed that this bound holds for the average number of packets stored by the nodes of each individual level. It is conjectured that this is actually the case; the following result provides strong evidence for this claim: As $t \to \infty$, the total number of packets stored by the nodes of levels $1, \ldots, j$ at time t does not exceed $j2^d q_\rho(1+\epsilon)$ with high probability, for any $\epsilon > 0$ (and for any $j \in \{1, \ldots, d\}$). This result may be proved by applying stochastic domination between the first j levels of networks $\mathcal{R}$ and $\tilde{\mathcal{R}}$, and using the product-form property of $\tilde{\mathcal{R}}$.

Finally, related to the above discussion is the following open problem: Analyze the performance of greedy routing under the assumption that nodes in levels $2, 3, \ldots, d$ of the butterfly have *constant* [that is, $\Theta(1)$] buffer capacity. Even though it is reasonable to conjecture that the average delay would still be $\Theta(d)$ for any $\rho < 1$, proving such a result appears to be a rather challenging task.

APPENDIX 5.A

In this appendix, we elaborate on a comment presented at the end of §5.3.4, namely that Proposition 5.12 does not seem to follow easily from a comparison between network $\mathcal{Q}$ and the Jackson network $\hat{\mathcal{Q}}$. (Recall that $\hat{\mathcal{Q}}$ is obtained from $\mathcal{Q}$ by replacing all the deterministic servers with exponential ones having unit service rate.) Here we present some "unsuccessful attempts" for proving Proposition 5.12 in this way.

Assume that both networks start at time 0 with one packet just generated at node 0 and destined for node e_1. Then, we have $N(t) \geq 1$ for all $t \in [0,1)$ with probability 1; on the other hand, the total number $\hat{N}(t)$ of packets of the Jackson network may drop to 0 (with positive probability) at any time $t \in (0,1)$. Hence, a stochastic inequality such as $N(t) \leq_{\mathrm{st}} \hat{N}(t)$ does *not* apply for $t \in [0,1)$.

Furthermore, notice that a result such as $E[N(t)] \leq E[\hat{N}(t)]$ $\forall t \geq 0$ would have been sufficient for proving Proposition 5.12. However, this result does *not* apply either. Indeed, let us consider again the aforementioned initial state for the two networks. We fix a $t^* \in (0,1)$; obviously, $E[N(t^*)] \geq 1$. On the other hand, with probability $1-e^{-t^*}$, the first packet departs from the network by time t^*. Thus, we have

$$E[\hat{N}(t^*)] \leq e^{-t^*} + E[A(t^*)] = e^{-t^*} + t^*(\lambda 2^d)\,,$$

which is smaller than unity for sufficiently small λ. [Recall that $A(t^*)$ denotes the total number of packets generated during the interval $[0,t^*)$ excluding the one initially present in the network.] It follows that $E[\hat{N}(t^*)] < 1 \leq E[N(t^*)]$ for sufficiently small λ.

Finally, since both networks $\mathcal{Q}$ and $\hat{\mathcal{Q}}$ are ergodic (for $\rho < 1$), Proposition 5.12 would also be implied by the following result:

$$\lim_{k\to\infty}\left[\frac{1}{k}\sum_{i=1}^{k}\Big(E[D_i]-t_i\Big)\right] \leq \lim_{k\to\infty}\left[\frac{1}{k}\sum_{i=1}^{k}\Big(E[\hat{D}_i]-t_i\Big)\right]\,, \qquad (5.A.1)$$

where t_i denotes the generation time of the ith packet under a particular sample path ω, and D_i (resp. $\hat{D}_i$) denotes the departure time of this packet from network $\mathcal{Q}$ (resp. $\hat{\mathcal{Q}}$). [Recall that each of the sample-path averages in $(5.A.1)$ converges to the corresponding steady-state average delay per packet.] The most straightforward way

to prove (5.A.1) is to show that

$$\sum_{i=1}^{k} E[D_i] \leq \sum_{i=1}^{k} E[\hat{D}_i] \,, \quad \text{for } k = 1, \ldots \tag{5.A.2}$$

However, this result does *not* apply to all sample paths. Below, we present a simple counterexample; for simplicity we take $d = 2$, so that it is equivalent (for certain sample paths) to consider the network $\mathcal{G}$ of Figure 5.2a instead of $\mathcal{Q}$. The corresponding Jackson network will be referred to as $\hat{\mathcal{G}}$.

Assume that at time 0 there are two packets $\mathcal{P}_1$ and $\mathcal{P}_2$ present in each network; $\mathcal{P}_1$ is located at server S_1, while $\mathcal{P}_2$ is at located server S_2; both packets are to join server S_3. The other packets of the sample path are generated much later, so that they do not influence $\mathcal{P}_1$ and $\mathcal{P}_2$ at all. Below, we prove that

$$E[D_1 + D_2] > E[\hat{D}_1 + \hat{D}_2] \,, \tag{5.A.3}$$

which disproves (5.A.2).

In the context of $\mathcal{G}$, we have either $D_1 = D_2 + 1 = 3$ or $D_2 = D_1 + 1 = 3$, depending on how the tie in server S_3 is broken. In both cases, we obtain

$$D_1 + D_2 = 5 \,. \tag{5.A.4}$$

In the context of $\hat{\mathcal{G}}$, let $X_i^{(j)}$ denote the service time of $\mathcal{P}_i$ at S_j; all these random variables are independent and assume the exponential distribution with mean 1. Clearly, two equivalent cases should be considered in the analysis, namely $X_1^{(1)} \leq X_2^{(2)}$ and $X_1^{(1)} > X_2^{(2)}$; the first (resp. second) case corresponds to $\mathcal{P}_1$ (resp. $\mathcal{P}_2$) departing first from the network. Due to symmetry, we only consider the case $X_1^{(1)} \leq X_2^{(2)}$. We have

$$E[\hat{D}_1 + \hat{D}_2] = E[\hat{D}_1 + \hat{D}_2 \mid X_1^{(1)} \leq X_2^{(2)}] \,. \tag{5.A.5}$$

Furthermore,

$$\begin{aligned} E[\hat{D}_1 \mid X_1^{(1)} \leq X_2^{(2)}] &= E[X_1^{(1)} + X_1^{(3)} \mid X_1^{(1)} \leq X_2^{(2)}] \\ &= E[X_1^{(1)} \mid X_1^{(1)} \leq X_2^{(2)}] + E[X_1^{(3)}] \\ &= \frac{3}{2} \,, \end{aligned} \tag{5.A.6}$$

where we have used the fact $E[\min\{R_1, R_2\}] = \frac{1}{2}$ for exponential random variables with mean 1. Furthermore, *given* that $X_1^{(1)} \leq X_2^{(2)}$, we have $X_2^{(2)} = X_1^{(1)} + Z$ where Z is exponentially distributed with mean 1 and is independent of $X_1^{(1)}$. Notice also that service of $\mathcal{P}_2$ at S_3 starts at time $X_1^{(1)} + \max\{X_1^{(3)}, Z\}$. It follows from the above discussion that

$$\begin{aligned} E[\hat{D}_2 \mid X_1^{(1)} \leq X_2^{(2)}] &= E[X_1^{(1)} + \max\{X_1^{(3)}, Z\} + X_2^{(3)} \mid X_1^{(1)} \leq X_2^{(2)}] \\ &= E[X_1^{(1)} \mid X_1^{(1)} \leq X_2^{(2)}] + E[\max\{X_1^{(3)}, Z\}] + E[X_2^{(3)}] \\ &= \frac{1}{2} + \frac{3}{2} + 1 \\ &= 3\,, \end{aligned}$$

where we have used the fact $E[\max\{R_1, R_2\}] = \frac{3}{2}$ for exponential random variables with mean 1. Using also (5.*A*.6) and (5.*A*.5), we obtain

$$E[\hat{D}_1 + \hat{D}_2] = \frac{9}{2}\,.$$

This together with (5.*A*.4) proves (5.*A*.3). Of course, the fact that (5.*A*.2) does not hold does not necessarily imply that (5.*A*.1) does not apply either; however, this fact suggests that if (5.*A*.1) applies, then its proof should be rather subtle.

6. Multiple Broadcasts in the Hypercube — Part I: First Results and Direct Schemes

In the dynamic routing problem of Chapter 5, it was assumed that each packet has a single destination. In this chapter and in the next one, we consider another dynamic problem, where each packet is *broadcast* to all nodes.

6.1 INTRODUCTION

6.1.1 Problem Definition — Motivation

The precise definition of the problem to be analyzed is as follows: Each node of the d-cube generates packets according to a Poisson process with rate λ; different nodes generate their packets independently of each other. Each packet is to be broadcast to all nodes. We assume that no other packet transmissions are taking place in the network.

We propose several routing schemes for broadcasting packets in the aforementioned context, and we analyze their throughput and delay properties in steady-state. As in

Chapter 5, we are interested in *distributed* routing schemes of the *on-line* type; that is, the routing decisions for the various packets are made locally, without knowledge of future events (see also §1.2.2) The underlying model for communications is the one presented in §1.2.3. However, it is taken that packets are generated by *continuous-time* processes, even though transmissions occur in *slotted time*. This assumption simplifies the analysis; with minor modifications, the results to be derived also apply to a system such as the one considered in §5.4, where batches of packets are generated at the end of each slot.

Motivation for studying the present problem arises from the context of *asynchronous computation*. Let us consider once more the distributed implementation of the iterative algorithm $x := f(x)$ (see §1.2.1). At the end of every iteration, each processor i has to broadcast the new value of x_i, in order that the other processors use it in their subsequent computations. If all processors are perfectly synchronized, then all entries of the vector x are to be broadcast at the same time, which gives rise to a multinode broadcast. However, there are cases where the new values of the x_i's are not all computed at the same time, while faster processors do *not* wait for slower ones; thus, packets to be broadcast are generated at unpredictable times. Such a model for asynchronous computation is appropriate when the speed of each processor is *variable*, while all processors are equally fast "on the average". Related asynchronous algorithms have received considerable attention in the literature, but have primarily been analyzed from the point of view of convergence (see [BeT89]). This kind of analysis often involves some a priori assumptions on the time required to broadcast a message. Here we make a first attempt to investigate this communication delay in the context of the hypercube network. For analytical tractability, we have assumed that packets are generated by the various nodes according to independent Poisson processes; we hope that our analysis will be suggestive of the results holding under more general packet-generating processes. To best of our knowledge, the problem formulated above as well as the results to be presented are new.

6.1.2 Summary of the Results

Whenever some node of a network wishes to broadcast a packet, it just has to transmit it along a spanning tree emanating from this node. (That is, a spanning tree with all of its arcs pointing away from this node.) If no other packet transmissions

take place at this time, then all nodes will have received the packet under broadcast after some time equal to the depth of the selected tree. This simple communication task is the *single node broadcast*; in the d-cube, it can be performed in d time units, by transmitting the packet along any tree with depth d. (There are several such trees from which to choose.)

Unfortunately, matters are not that simple in the presence of *contention*. Indeed, consider the following simple scheme for our dynamic routing problem: each node chooses a spanning tree rooted at itself and broadcasts all of its packets along that tree. It will be seen in §6.3.1 that the performance of such a routing scheme may possibly be rather poor, in the sense that the traffic accommodated can be extremely low. Of course, there is an upper bound for the traffic that can be sustained by *any* routing scheme. This bound is given by the following *necessary* condition for *stability*:

$$\rho \stackrel{\text{def}}{=} \lambda \frac{2^d - 1}{d} < 1 \,; \tag{6.1}$$

that is, if this inequality does not hold, then the number of packets whose broadcast has not been completed grows to infinity as time elapses. The parameter ρ will be called the *load factor* of the system. Thus, we are interested in routing schemes that are stable for all $\rho < \rho^*$, where ρ^* is a constant (preferably close to 1). This requirement is satisfied by routing schemes performing multinode broadcasts *periodically*. Unfortunately, these schemes will be seen to introduce unacceptably high *delay*, even in light traffic (namely, for $\rho \approx 0$). Thus, we shall be mainly concerned with devising schemes that also satisfy a requirement on the average *delay* per packet in light traffic; namely, that $T = \Theta(d) + O(d)\rho$ for small values of ρ. (Note that T is defined as the steady-state average time spent by a packet in the system until its broadcast is completed.) This requirement for the delay is motivated by the fact that it takes d time units to perform a single node broadcast in the d-cube in the absence of other transmissions; thus, it is desirable that contention does not increase this delay by more than a factor depending on the load of the network. More discussion on this point is given in §6.2.3, following the derivation (in §6.2.2) of a *universal lower* bound for T; that is, a bound applying to *any* routing scheme.

In §6.3 and in Chapter 7, we present several routing schemes that meet the aforementioned performance objectives. The schemes discussed in §6.3 are characterized as *direct* ones, because each packet is broadcast along a spanning tree rooted at the node

where it was generated. These schemes are stable even for $\rho \approx 1$, while they seem to satisfy the desirable delay properties. Unfortunately, the analysis is intractable at that point; nevertheless, we provide some strong evidence for the aforementioned claims, based on simulation (see §6.4) and on an approximate model developed in §6.5. In Chapter 7, we present an *indirect* routing scheme; that is, all packets are sent to one of nodes $e_1, \ldots, e_d$, which are in charge of performing the broadcasts along the d disjoint spanning trees $\mathcal{T}^{(1)}, \ldots, \mathcal{T}^{(d)}$ introduced in [JoH89] (see also §2.1.4). We prove *rigorously* that this scheme is stable for all $\rho < \rho^* \approx \frac{2}{3}$, while it satisfies $T \approx 3d + 1 + \frac{9}{4}\rho$ for small ρ. The analysis of the indirect scheme is based on an interim result (namely, Lemma 7.2), concerning the delay induced in a tree of paths; this result appears to be applicable to other routing problems involving trees.

In evaluating the performance of the various schemes, we also consider the steady-state average *queue-size* Q per node. Our schemes appear to be efficient also with respect to queue-sizes. Study of the behavior of the measure Q aims at estimating the buffer capacity required for applying the schemes in practice. In fact, the indirect scheme of Chapter 7 is *deadlock-free* when implemented with finite buffers; for the rest of the schemes, deadlock prevention can be achieved by using standard techniques (see §7.4).

The problem of multiple broadcasts in the hypercube is considerably more intractable than that of multiple node-to-node communications (in Chapter 5). Thus, for simplicity, it will be taken for granted that if a routing scheme is stable, then all steady-state statistics are well-defined. A more formal analysis such as that of §5.3.3 would complicate our discussion even further, while obscuring the main issues related to routing.

6.2 PRELIMINARY RESULTS

6.2.1 The Necessary Condition for Stability

The average total number of packets generated in the network during one slot equals $\lambda 2^d$. Broadcasting a packet (using any routing scheme) requires at least $2^d - 1$ transmissions. Therefore, during each slot, an average total demand for at least $\lambda 2^d(2^d - 1)$ packet transmissions is generated in the system. Since at most $d2^d$ packet transmissions may take place during each slot, it follows that the system can be stable only if

$\lambda 2^d(2^d-1) < d2^d$. Thus, we have the following necessary condition for stability:

$$\rho \stackrel{\text{def}}{=} \lambda \frac{2^d-1}{d} < 1 \,, \tag{6.1}$$

where ρ is the *load factor* of the system. This terminology is appropriate, because when $\rho \approx 1$ all hypercube arcs are almost always busy, even if no redundant packet transmissions take place.

In light of (6.1), asymptotics with respect to ρ will be taken with *fixed* d; thus, $\rho \to 1$ should be interpreted as $\lambda \to \frac{d}{2^d-1}$, while $\rho \to 0$ should be interpreted as $\lambda \to 0$.

6.2.2 Lower Bounds on the Delay

First, we establish a *universal* lower bound on the delay T; that is, a bound that applies to *any* routing scheme. Recall that T is defined as the steady-state average of the time elapsing between the moment a packet is generated until the completion of its broadcast.

Proposition 6.1: The average delay T per packet induced by any routing scheme satisfies

$$T \geq \max\left\{ d \,, \frac{2^d-1}{2^d}\left[\mathcal{D}(d;\rho) + \frac{1}{2}\right] \right\} = \Omega\left(d + \frac{\rho}{d(1-\rho)} \right) \,,$$

where $\mathcal{D}(d;\rho)$ is the average delay for the discrete-time $M/D/d$ queue with unit service time and arrival rate $d\rho$. ■

Proof: We denote by $\mathcal{L}(x)$ the set of arcs *incoming* at node x; that is,

$$\mathcal{L}(x) \stackrel{\text{def}}{=} \left\{ (x \oplus e_j, x) \mid j = 1, \ldots, d \right\} .$$

Any legitimate routing scheme for performing broadcasts must conform to the following two constraints:

(a) For every node $y \neq x$, each packet generated at y must traverse at least one arc in the set $\mathcal{L}(x)$.

(b) If a packet generated at node y traverses arc $(x \oplus e_j, x)$, then either $x \oplus e_j = y$ or the packet has previously traversed an arc in the set $\mathcal{L}(x \oplus e_j)$.

The first of the above constraints guarantees that each packet is received by all nodes, while the second constraint guarantees that a packet traverses an arc only after being received by the arc's starting node.

We now fix a node x and we focus on the transmissions to be performed over the arcs of the set $\mathcal{L}(x)$. At each slot there are a number of such transmissions pending, including the ones currently in progress. Suppose that we *relax* constraint (b); that is, we assume that a packet may traverse any arc of $\mathcal{L}(x)$, provided that it has already been generated. This is equivalent to assuming that all *neighbors* of x receive instantaneously any packet generated elsewhere in the network. Transmissions over the arcs of $\mathcal{L}(x)$ may now be done *sooner*, because the permissible schedules are *less constrained* than previously. Therefore, by relaxing constraint (b), node x receives packets no later than previously; thus, the average time W_x required for the "typical" packet to reach node x may only *decrease*. The smallest possible value for W_x would be attained if the following were true: None of the packets generated at x traverses any arc of $\mathcal{L}(x)$, while any packet generated elsewhere *only* traverses the *first available* arc of $\mathcal{L}(x)$. Clearly, in this case, the d arcs of set $\mathcal{L}(x)$ operate as a discrete-time $M/D/d$ queue with synchronization, having unit service time and arrival rate $\lambda(2^d - 1) = d\rho$. [The definition of this queueing system is analogous to that of the discrete-time $M/D/1$ queue with synchronization, given in §2.1.2; recall also the definition of ρ in (6.1).] Taking also the average synchronization time into account (which equals $\frac{1}{2}$, as argued in §2.1.2), we obtain

$$W_x \geq \frac{2^d - 1}{2^d}\left[\mathcal{D}(d;\rho) + \frac{1}{2}\right], \tag{6.2}$$

where $\mathcal{D}(d;\rho)$ is the average delay for a discrete-time $M/D/d$ queue with unit service time and arrival rate $d\rho$; the factor $\frac{2^d-1}{2^d}$ accounts for the fact that packets generated by node x do not join this queue (while they are instantaneously received by x). Combining (6.2) with the obvious facts $T \geq W_x$ and $T \geq d$, we obtain

$$T \geq \max\left\{d\,,\, \frac{2^d - 1}{2^d}\left[\mathcal{D}(d;\rho) + \frac{1}{2}\right]\right\}. \tag{6.3}$$

Furthermore, it is known [Bru71] that $\mathcal{D}(d;\rho)$ satisfies

$$\mathcal{D}(d;\rho) \geq \frac{1}{2} + \frac{\rho}{2d(1-\rho)}.$$

This together with (6.3) and the fact $\frac{2^d-1}{2^d} \geq \frac{1}{2}$ implies that

$$T \geq \max\left\{d\,,\, \frac{\rho}{4d(1-\rho)}\right\};$$

the proof is completed by using the inequality $\max\{\alpha_1, \alpha_2\} \geq \frac{1}{2}(\alpha_1 + \alpha_2)$. **Q.E.D.**

As suggested by the proof of Proposition 6.1, a scheme that comes close to attaining the universal lower bound for the delay T (if there exists such a scheme) would schedule transmissions *adaptively* and/or by making use of *global* information. This claim is further supported by Proposition 6.2, which establishes a sharper lower bound on the delay T induced by *oblivious* schemes. Under such a scheme, each packet decides which paths to follow *independently* of all other packets in the network and insists on traversing the selected paths, regardless of the contention encountered en route; all packets generated by the same node follow the *same* rules (which are time-independent). Clearly, the class of oblivious schemes comprises all schemes where each packet independently selects which tree to be broadcast along by using a randomized rule that depends only on the identity of its origin node. The routing schemes discussed in §6.3 are of this type. We now present the lower bound on the delay induced by oblivious schemes.

Proposition 6.2: The average delay T per packet induced by any *oblivious* routing scheme satisfies

$$T \geq \max\left\{d\,,\,\frac{2^d-1}{2^d}\left[\frac{3}{2}+\frac{\rho}{2(1-\rho)}\right]\right\} = \Omega\left(d+\frac{\rho}{1-\rho}\right)\,. \qquad \blacksquare$$

Proof: This proof is similar to that of Proposition 6.2. Again, we fix a node x and we consider the set $\mathcal{L}(x)$ of arcs incoming at x. Each packet generated at some node z will attempt to cross some of the arcs of $\mathcal{L}(x)$; which arcs will be crossed is determined by a randomized rule depending only on node z. Of course, all legitimate oblivious schemes are subject to constraints (a) and (b) presented in the proof of Proposition 6.1. Again, we *relax* constraint (b); this may only result in node x receiving packets *earlier* than previously. Recall now that, under an oblivious scheme, packets select their respective paths independently. Therefore, after relaxing constraint (b), each arc $(x \oplus e_j, x)$ is fed by a Poisson stream with rate r_j; by constraint (a), we have $\sum_{j=1}^{d} r_j \geq \lambda(2^d - 1)$. [Notice that the arrival stream pertaining to each arc $(x \oplus e_j, x)$ has constant rate, because the routing rules were taken time-independent.] Furthermore, the average time W_x for a packet to reach node x is minimized if packets generated by x do not cross any arcs of $\mathcal{L}(x)$ while packets generated elsewhere cross exactly *one* such arc;

in this case, we have

$$\sum_{j=1}^{d} r_j = \lambda(2^d - 1) = d\rho \,, \tag{6.4}$$

where we have also used (6.1). As proved in §2.1.2, the the average delay induced by a discrete-time $M/D/1$ queue with synchronization (having arrival rate r and unit service time) equals $\frac{3}{2} + \frac{r}{2(1-r)}$; see (2.5). Thus, it follows that

$$W_x \geq \sum_{j=1}^{d} \frac{r_j}{\lambda 2^d} \left[\frac{3}{2} + \frac{r_j}{2(1-r_j)} \right] \,, \tag{6.5}$$

where we have also taken into account the fact that packets generated by node x do not join the queue for any of the arcs of $\mathcal{L}(x)$. Notice now that $r[\frac{3}{2} + \frac{r}{2(1-r)}]$ is a convex function of r; therefore, in light of (6.4), the right-hand quantity in (6.5) is minimized for $r_1 = \cdots = r_d = \lambda \frac{2^d - 1}{d} = \rho$. Thus, we obtain

$$W_x \geq \sum_{j=1}^{d} \frac{\rho}{\lambda 2^d} \left[\frac{3}{2} + \frac{\rho}{2(1-\rho)} \right] = \frac{2^d - 1}{2^d} \left[\frac{3}{2} + \frac{\rho}{2(1-\rho)} \right] .$$

Combining this with the obvious facts $T \geq W_x$ and $T \geq d$, we have

$$T \geq \max \left\{ d \,, \frac{2^d - 1}{2^d} \left[\frac{3}{2} + \frac{\rho}{2(1-\rho)} \right] \right\} . \tag{6.6}$$

Since $\frac{2^d - 1}{2^d} \geq \frac{1}{2}$, and $\max\{\alpha_1, \alpha_2\} \geq \frac{1}{2}(\alpha_1 + \alpha_2)$, it follows that the right-hand quantity of (6.6) is $\Omega(d + \frac{\rho}{1-\rho})$; this completes the proof of the result. **Q.E.D.**

Each of the lower bounds derived consists of two additive terms; one of them accounts for the minimum *propagation delay* incurred per broadcast (namely, d), while the other term corresponds to a minimum overhead due to *contention*. The universal lower bound implies that $\liminf_{\rho \to 1}[(1-\rho)T]$; that is, the average delay T under heavy traffic is $\Omega(\frac{1}{1-\rho})$. (Recall that asymptotics with respect to ρ are taken with fixed d.)

Proposition 6.2 implies that for a rather broad class of schemes the universal lower bound on the delay T is loose, due to the presence of the factor $\frac{1}{d}$. Suppose now that, under oblivious routing, we allow packets generated at each node x to take into account the routing decisions made by packets previously generated by the *same* node x. It is an interesting open question to investigate whether Proposition 6.2 still holds.

We conjecture that this is true, because each packet has very limited knowledge of the routing decisions taken within the entire network. Related to this open question are the server allocation problems discussed by Stamoulis and Tsitsiklis in [StT91]. If our conjecture is true, then a scheme coming close to attaining the universal lower bound on T should either involve *centralized coordination* or some form of *adaptive* routing.

Finally, it is worth noting the similarities between the results of this subsection and those of §5.2.2, as well as between the respective proofs.

6.2.3 Setting the Performance Objectives

The bounds provided by (6.1) and Proposition 6.1 demonstrate the *limitations* on the performance of *any* scheme that may be applied in our routing problem. Given these bounds, we are interested in devising schemes that come fairly close to meeting *both* of them.

Starting with the asymptotic properties of the delay T, it follows from Proposition 2.1 that $T = \Omega(d)$ for any fixed ρ, and $T = \Omega(\frac{1}{1-\rho})$ for any fixed d. Thus, *ideally*, we would desire to devise a scheme maintaining stability for all $\rho < 1$ and satisfying $T \leq K\frac{d}{1-\rho}$ (with K being some constant). Such a scheme would have the broadest possible stability region and optimal asymptotic delay properties. Again, there is remarkable similarity between this discussion and the one of §5.1.3, where we presented the performance objectives for the problem of multiple node-to-node communications. Unfortunately, the present problem is considerably more intractable than that of Chapter 5. Thus, we have to "lower our expectations" and settle for somewhat weaker performance objectives. In particular, we are interested in devising schemes with the following properties:

(a) Stability should be maintained (for all d) even when the load factor ρ is $\Theta(1)$.

(b) Under very light traffic, i.e. for $\rho \approx 0$, the average delay T should equal Kd, with K being some constant; that is, $\lim_{\rho \to 0} T = Kd$. This requirement is suggested by the fact that it takes d slots to perform a broadcast (in the d-cube) in the absence of contention. Also, the *first-order* approximation of T (as a function of ρ) should be of the form $T \approx Kd + O(d)\rho$. This requirement basically implies that, for moderately small values of ρ (yet independent of d), the additional delay due to contention should not be larger than a multiple of the delay attained in the absence of contention.

Motivation for considering the first-order approximation of T is as follows: As implied by Propositions 6.1 and 6.2, the delay T will definitely be large when the system is loaded close to capacity (i.e., for $\rho \approx 1$). Thus, it is expected that in practical applications (and particularly in situations modelling asynchronous algorithms), the load factor ρ has some small or moderate value. (A negligible value of ρ would make very inefficient use of the network, while the routing problem would be trivial.)

Based on these performance objectives, we have devised and analyzed several efficient schemes, which we present in the sections to follow as well as in Chapter 7.

6.3 DIRECT ROUTING SCHEMES

6.3.1 A Simple Approach to the Problem

The most straightforward approach to our problem is as follows: Each of the nodes broadcasts its packets along a certain spanning tree emanating from itself. Such a scheme can possibly have rather *poor* performance. In fact, its performance depends heavily on the selection of the trees. For example, consider the case where each node routes its packets along the corresponding unbalanced spanning tree in which the hypercube dimensions are crossed in increasing index-order (see §2.1.3). Every node z receives through its adjacent arc of the jth type all packets originating at nodes x satisfying $x_m = z_m$ for $m > j$ and $x_j \neq z_j$. Thus, during each slot, there are generated an average of $\lambda 2^{j-1}$ packets that will eventually have to traverse arc $(z \oplus e_j, z)$. It follows that the simple scheme under analysis may be stable only if $\lambda 2^{j-1} < 1$ for $j = 1, \ldots, d$, or equivalently

$$\rho < \frac{2}{d}\left(1 - \frac{1}{2^d}\right).$$

Hence, the load factor that can be sustained by the above simple scheme *deteriorates* to 0 as the dimensionality d of the hypercube increases; moreover, the first of the performance objectives set in §6.2.3 is not met, because there exists no constant ρ^* such that stability be guaranteed (for all d) if $\rho < \rho^*$. The reason for this undesirable behavior is that some of the arcs are shared by far more trees than the others.

A potential remedy to the above problem is to select 2^d trees (one rooted at each node) such that all arcs are shared by approximately the *same* number of trees. There does exist such a set of trees, namely the ones used in the optimal multinode broadcast

algorithm of [BOSTT91]. Since this algorithm lasts for $\lceil \frac{2^d-1}{d} \rceil$ slots, it follows that each arc is shared by at most $\lceil \frac{2^d-1}{d} \rceil$ of the trees. Broadcasting the packets along these trees will create no additional bottlenecks in any of the arcs. As is discussed in §§6.3.3-6.4, this scheme performs rather satisfactorily.

An alternative way of balancing traffic over the hypercube arcs is to allow for *multiple* trees per node and distribute among them the packets to be broadcast; an efficient routing scheme of this spirit is presented in §6.3.3. Both this scheme and the one mentioned above are closely related to the periodic schemes, which we discuss next.

6.3.2 Performing Multinode Broadcasts Periodically

In this subsection, we discuss schemes where an efficient *static* algorithm for performing multinode broadcasts is run *periodically*. Though very efficient with respect to their stability regions, these schemes induce large delays even in light traffic; for this reason, they will be modified in §6.3.3.

The optimal completion time for the multinode broadcast task is $\lceil \frac{2^d-1}{d} \rceil$ slots, and it is attained by an algorithm constructed by Bertsekas et al. [BOSTT91]. (To see that this quantity is a lower bound on the time required for the task, just notice that each node has to receive $2^d - 1$ packets, while it may only receive at most d packets per slot.) By pipelining successive instances of this optimal algorithm, each node may broadcast one packet every $\lceil \frac{2^d-1}{d} \rceil$ slots. In fact, the packets generated at each node z form a discrete-time $M/D/1$ queue with synchronization; the arrival rate is λ and the "service time" duration equals $\lceil \frac{2^d-1}{d} \rceil$. Therefore, the scheme is stable if and only if $\lambda \lceil \frac{2^d-1}{d} \rceil < 1$, or equivalently

$$\rho < \frac{(2^d-1)/d}{\lceil (2^d-1)/d \rceil} . \tag{6.7}$$

It is known that $\frac{2^d-1}{d}$ is never an integer for $d \geq 2$; see [BOSTT91] for a simple proof of this fact. This implies that the scheme under analysis becomes unstable if ρ is sufficiently close to 1. Of course, if d is not very small, then the right-hand quantity in (6.7) is very close to 1; e.g., for $d = 10$ (which is a realistic value), this quantity equals 0.9932. Thus, for all practical purposes, the load factor that can be accommodated under this scheme is very high. On the other hand, since each period of the scheme

lasts for $\lceil \frac{2^d-1}{d} \rceil$ time units, we have $T = \Omega(\frac{2^d}{d})$ even for $\rho \approx 0$; thus, the delay induced by this periodic scheme is unacceptably *high*, even in very light traffic.

Another periodic routing scheme is obtained by pipelining successive instances of the optimal static algorithm for the d simultaneous broadcasts, which was constructed by Saad and Schultz [SaS85]. For this communication task, every node z has d packets to broadcast; each of these packets is routed along a completely unbalanced spanning tree rooted at node z. In particular, one of the packets is routed along that tree where the hypercube dimensions are crossed in the order $1, 2, \ldots, d$; one of the packets is routed along that tree where the hypercube dimensions are crossed in the order $2, 3, \ldots, d, 1$ etc; one of the packets is routed along that tree where the hypercube dimensions are crossed in the order $d, 1, 2, \ldots, d-1$. Each packet that traverses the hypercube dimensions under the order $j, (j \bmod d) + 1, \ldots, [(j + d - 2) \bmod d] + 1$ will be called a j-packet; for each packet, the corresponding value of j is the called the *tag* of the packet. The algorithm of [SaS85] takes time $2^d - 1$, which is the optimal duration.

Returning to the periodic scheme for our dynamic routing problem, let us assume (for simplicity) that each packet selects its tag randomly, with all values $1, \ldots, d$ being equiprobable. Then, prior to being broadcast, all j-packets generated at the same node join a discrete-time $M/D/1$ queue with synchronization, having arrival rate $\frac{\lambda}{d}$ and "service time" duration $2^d - 1$. Thus, the present periodic scheme is stable if and only if $\frac{\lambda}{d}(2^d - 1) < 1$, or equivalently $\rho < 1$, which is the broadest possible stability region. However, the average delay T per packet is extremely high, even in very light traffic; indeed, since each period of the scheme lasts for $2^d - 1$ slots, we have $T = \Omega(2^d)$ even for $\rho \approx 0$. Notice that the same properties would apply even if packets were assigned their tags in a round-robin mode, which is the best possible rule. [That is, if the kth packet generated at each node z were assigned a tag-value $(k-1) \bmod d + 1$; this would give rise to an $M/D/d$ queue at each node z, instead of d separate $M/D/1$ queues.]

Other periodic schemes may be obtained by pipelining instances of other nearly optimal multinode broadcast algorithms, such as the ones in [JoH89]. It is apparent that these periodic schemes would share the main properties of the two schemes described above; namely, that stability is maintained even for high values of the load

factor ρ, whereas the delay performance in light traffic is highly unsatisfactory. This is primarily due to the extensive occurrence of *idling*, caused by the periodic feature of these schemes; that is, it often occurs that arcs are idle while packets have to wait for the next period in order to cross them. As will be seen in the next subsection, avoidance of this idling phenomenon improves performance dramatically.

6.3.3 Non-Idling Versions of the Periodic Schemes

Under the *non-idling* versions of each of the periodic schemes of §6.3.2, routing is simply performed as follows: Each packet is broadcast along the same tree as in the periodic scheme, and proceeds *as fast as possible*, subject to contention. Both non-idling schemes to be presented appear to perform rather satisfactorily. Unfortunately, their delay properties seem to be analytically intractable; thus, they are studied by means of approximations and simulations.

First, we establish that the stability region of each non-idling scheme is the *same* as that of the corresponding periodic scheme. Let us consider the one based on the optimal multinode broadcast algorithm of [BOSTT91]; here, each packet generated by node x is broadcast along that tree used by x in the algorithm of [BOSTT91]. This scheme is basically of the simple form considered in §6.3.1, because it involves one spanning tree per node; moreover, it belongs to the class of oblivious schemes defined in §6.2.2. As already mentioned in §6.3.1, each arc of the d-cube is shared by at most $\lceil \frac{2^d-1}{d} \rceil$ of the trees used; thus, during each slot, there are generated an average of at most $\lambda \lceil \frac{2^d-1}{d} \rceil$ packets that will eventually have to traverse any particular arc $(z, z \oplus e_j)$. Therefore, the standard argument of calculating the average demand for transmissions over each arc does not impose any *necessary* condition for stability other than (6.7), which is a rather favorable one. Below, we show that (6.7) is also a *sufficient* condition for stability of the non-idling scheme, provided that a simple *distributed* priority scheme is used.

We fix a node x; let τ_n be the time instant when the nth packet was generated by x, for $n = 1, \ldots$ We assume that the nth such packet is attached with a *label*, denoted by ℓ_n; we have $\ell_n = \ell$ if the following is true: In the *periodic* version of the algorithm, the nth packet would have been broadcast by node x in the ℓth period, that is during the multinode broadcast starting at time $(\ell - 1)\lceil \frac{2^d-1}{d} \rceil + 1$. It is easily seen that these

labels may be updated in a recursive way, according to the following simple rule:

$$\ell_{n+1} = \max\left\{\ell_n + 1, \left\lceil \frac{\tau_n}{\lceil (2^d-1)/d \rceil} \right\rceil + 1\right\} ,$$

and

$$\ell_1 = \left\lceil \frac{\tau_1}{\lceil (2^d-1)/d \rceil} \right\rceil + 1 .$$

Having defined these labels, we consider the following rule for *contention resolution*: If two or more packets wish to traverse the same arc at the same time, then priority is given to that with the *smallest* label; if there are more than one such packet, then priority is given to the one (with minimum label) that would had crossed this arc the *first* during the multinode broadcast algorithm of [BOSTT91]. Using this priority discipline, the *order* of the various packet transmissions is *preserved* when the periodic routing scheme is converted to a non-idling one; thus, each packet now arrives at its destination *no later* than under the periodic version of the scheme; therefore, inequality (6.7) is a *sufficient* condition for the stability of the non-idling scheme as well.

Next, we consider the non-idling version of the second periodic scheme of §6.3.2, which is based on the optimal algorithm for the d simultaneous broadcasts of [SaS85]. Under this non-idling scheme, packets are simply routed as follows: Upon generation, each packet randomly selects its tag-value j, and is to be broadcast by crossing the hypercube dimensions in the order $j, (j \bmod d) + 1, \ldots, [(j+d-2) \bmod d] + 1$; each packet proceeds as fast as possible, subject to contention. Using a priority discipline such as the one presented in the previous paragraph, we can guarantee stability of the non-idling scheme for the same region as for the periodic scheme, namely for all $\rho < 1$. In our discussion of §§6.4 and 6.5, we assume that contention is resolved on a FIFO basis, which is a more natural priority discipline.

Regarding the average delay T, it is straightforward that $T \approx d + \frac{1}{2}$ for $\rho \approx 0$, under both non-idling schemes, because the trees used for the various broadcasts have *depth* d. Indeed, notice first that $T \geq d + \frac{1}{2}$, for obvious reasons. (The term $\frac{1}{2}$ accounts for the average synchronization time.) Furthermore, for each non-idling scheme, consider an *inferior* routing scheme performing *single node* broadcasts one after the other along the same trees; since only one broadcast may be completed every d slots, this scheme is stable if $\lambda 2^d d < 1$, or equivalently if $\rho < \rho_d^* = \frac{2^d-1}{d^2 2^d}$; moreover, the inferior scheme induces an average delay $\tilde{T} = d + \frac{1}{2} + \frac{d\rho}{2(\rho_d^* - \rho)}$ per packet. Since $T \leq \tilde{T}$, it follows

that $\lim_{\rho \to 0} T \leq d + \frac{1}{2}$, which in turn implies that $T \approx d + \frac{1}{2}$ for $\rho \approx 0$. (Recall that asymptotics with respect to ρ are taken with fixed d.)

Regarding the *first-order* term in T, it is *conjectured* that it is $O(d)\rho$ under both non-idling schemes; in particular, we believe that it equals $\Theta(d)\rho$ [rather that being $o(d)\rho$], because each packet faces *additional contention* for each arc it attempts to cross. Given the complex structure of the paths followed by the various packets (under both routing schemes), it appears that good bounds for T are very hard (if at all possible) to derive formally. For this reason, we resorted to simulations; the experimental results obtained are presented in the next section. In fact, the aforementioned conjectures for the delay will be seen to agree with the simulation outcomes to a satisfactory extent.

6.4 EXPERIMENTAL RESULTS FOR THE DIRECT SCHEMES

In this section, we discuss the delay performance of the two routing schemes of §6.3.3, as investigated by means of simulation; each of the outcomes to be reported was obtained over a period of 1,000 slots.

As already mentioned in §6.3.3, it is expected (for both schemes) that the first-order term in T is $\Theta(d)\rho$. In order to examine the validity of this conjecture, we have plotted the simulation outcomes for measure $T - d - \frac{1}{2}$ as a function of d, for various values of ρ. (Recall that $d + \frac{1}{2}$ is the value for the zero-order term in T, under both schemes.) In particular, in Figure 6.1 we present these plots for the scheme using the trees of the multinode broadcast algorithm of [BOSTT91]. Apparently, for all three values of ρ considered, $T - d - \frac{1}{2}$ is an *increasing* function of d; in fact, it seems that $T - d - \frac{1}{2}$ grows linearly in d (for fixed ρ), with the corresponding *slope* being increasing in ρ. (Note however, that the points obtained for the smaller values of d do not match very well to a linear pattern.) Similar conclusions are drawn by observing Figure 6.2, which corresponds to the scheme based on the algorithm of [SaS85], using d completely unbalanced spanning trees per node. (Note that the vertical axis of this figure is in a different scale than that of Figure 6.1.) In fact, the conjecture that $T - d - \frac{1}{2}$ grows *linearly* in d (for fixed ρ) is supported rather strongly by the plots of Figure 6.2. This conjecture will be further supported by an approximate model for delay analysis, which is developed in §6.5.

The simulation outcomes reported so far provide us with strong evidence that

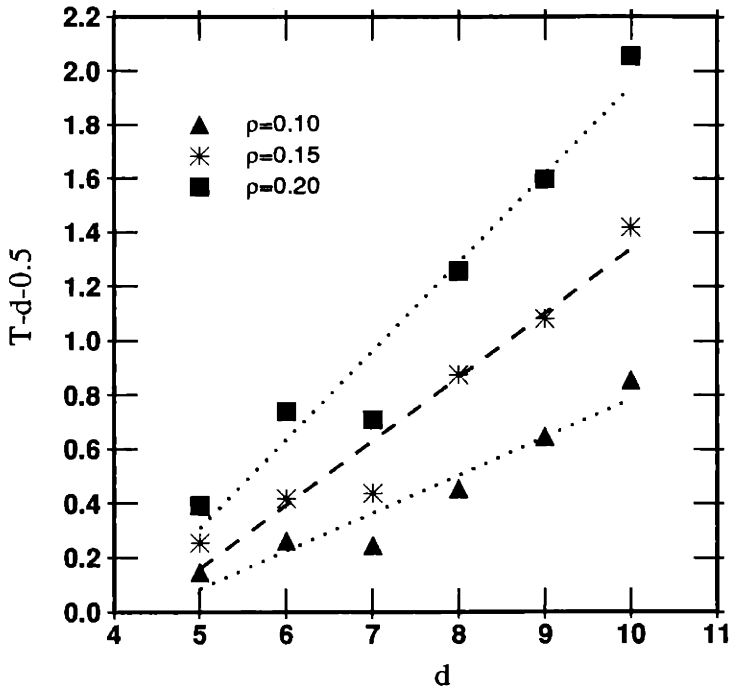

Figure 6.1: Experimental results for $T - d - \frac{1}{2}$, for the routing scheme using the trees of the multinode broadcast algorithm of [BOSTT91].

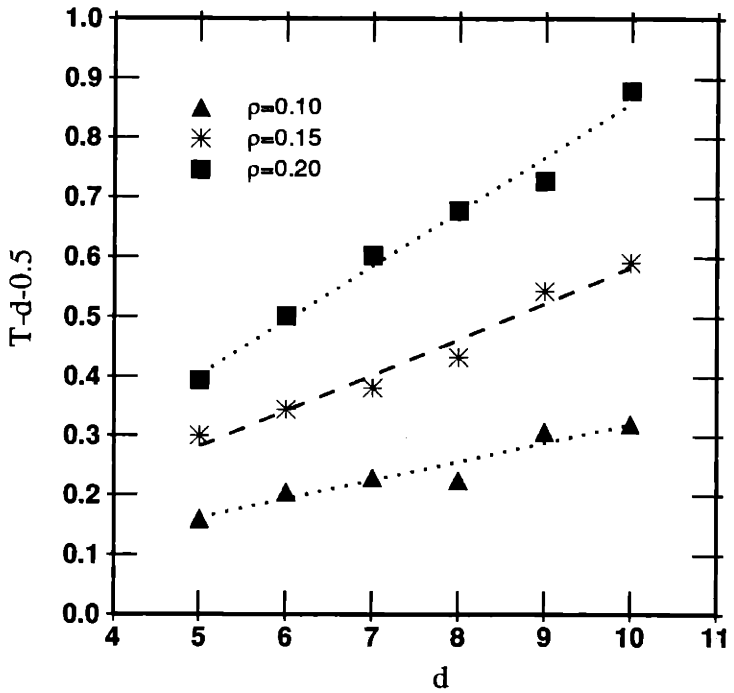

Figure 6.2: Experimental results for $T - d - \frac{1}{2}$, for the routing scheme using the trees of the algorithm for the d simultaneous multinode broadcasts of [SaS85].

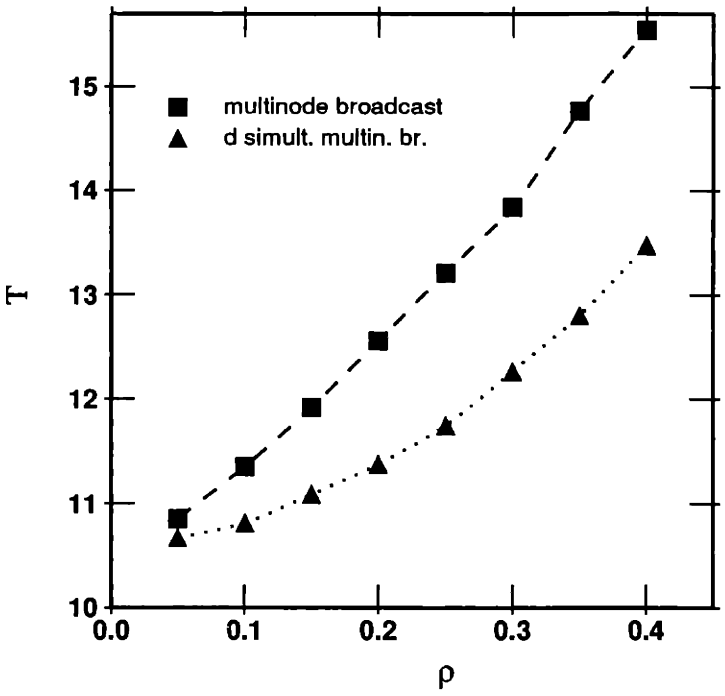

Figure 6.3: Comparing the delay induced by the two non-idling schemes, for $d = 10$.

$T \approx d + \frac{1}{2} + \Theta(d)\rho$ for small ρ, under both routing schemes considered; thus, it appears that our schemes meet the second performance objective set in §6.3.3.

Next, we compare the delay performance of the two schemes. Observing Figures 6.1 and 6.2, it is seen that the scheme using the d completely unbalanced trees per node is superior to that using the trees of the algorithm of [BOSTT91]. This fact is observed more clearly in Figure 6.3 (where $d = 10$ and ρ varies from 0.05 to 0.4), and in Figure 6.4 (where d varies from 5 to 10 and $\rho = 0.15$); it is worth noting that for fixed ρ, the first-order term $T - d - \frac{1}{2}$ in the delay grows more *steeply* in d under the scheme using the trees of [BOSTT91].

The superiority of the scheme using the d completely unbalanced spanning trees per node becomes even more apparent by studying the behavior of the steady-state average *queue-size* Q per node. In evaluating this performance measure, we consider a packet as stored at some node x only if x has yet to *forward* it towards one of its neighbors; packets to start transmission(s) immediately are also included. Note that a packet to be forwarded to several neighbors of x is assumed to occupy *one* unit of buffer capacity, because it would be wasteful for x to create the necessary copies of the packet prior to its transmission. In Figure 6.5, we have plotted the values of Q (for

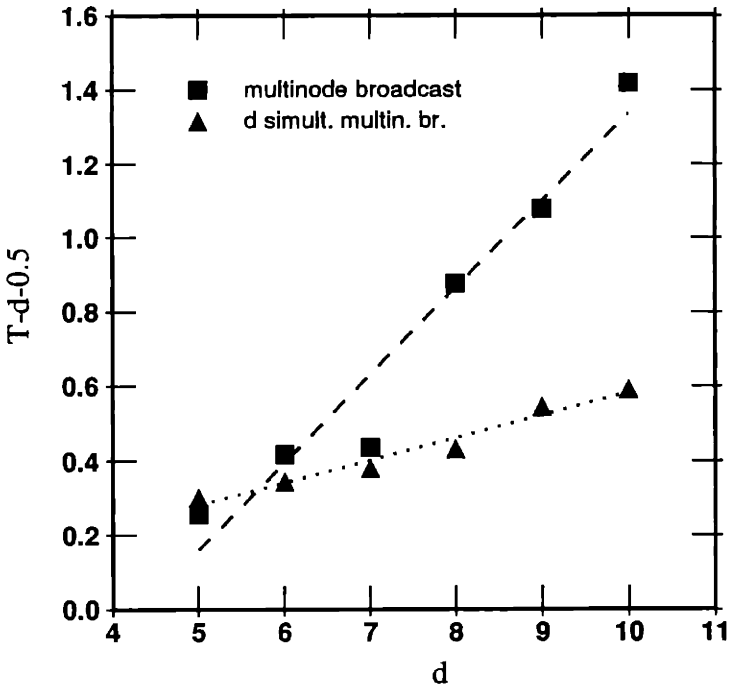

Figure 6.4: Comparing the delay induced by the two non-idling schemes, for $\rho = 0.15$.

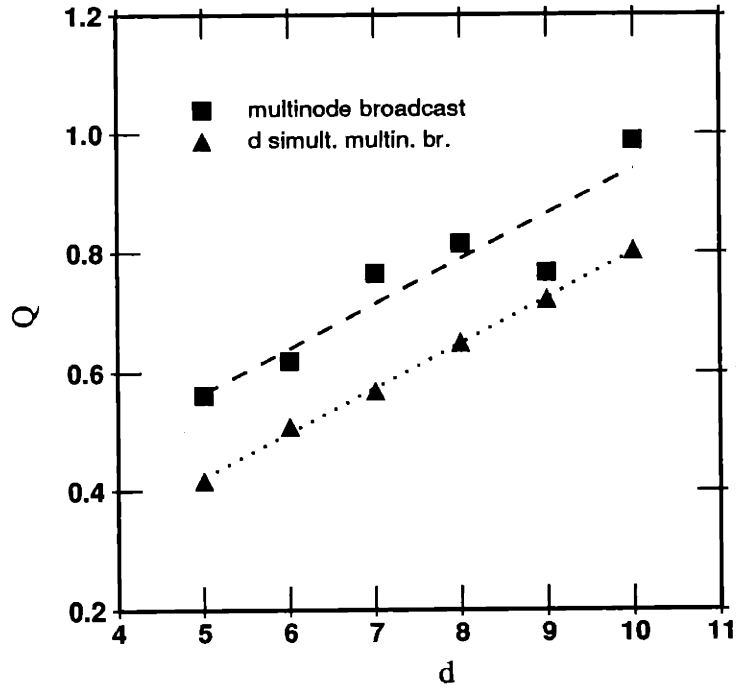

Figure 6.5: Comparing the queue-sizes for the two non-idling scheme, for $\rho = 0.15$.

both schemes) with d varying from 5 to 10 and $\rho = 0.15$; these results were obtained from the same simulation runs as the ones of Figure 6.4. It is seen that larger queues are built under the scheme using the trees of the algorithm of [BOSTT91].

The above discussion implies that the scheme using the d completely unbalanced trees per node is superior to that using the trees of the multinode broadcast algorithm of [BOSTT91]. The latter scheme is primarily of theoretical interest, due to its complexity in implementation; indeed, the underlying trees of [BOSTT91] are rather complicated and cannot be described in a concise way, unlike the completely unbalanced spanning trees (see below). Thus, this scheme will not be considered any further. It is an interesting open problem to generate a set of 2^d *simple* spanning trees (one rooted at each node) so that no arc is contained in more than $\Theta(\frac{2^d}{d})$ arcs; as argued in §6.3.1, such a set of trees may possibly qualify for efficient direct routing of multiple broadcasts.

The routing scheme using d completely unbalanced spanning trees per node is very simple to implement; each packet can be forwarded correctly by the various intermediate nodes, provided that it only carries the identity of its origin and its tag-value (indicating the adopted order of crossing the hypercube dimensions). In the next section, we investigate the performance of this non-idling scheme further. Henceforth, we shall be referring to it as the *efficient direct* scheme or simply the direct scheme.

6.5 APPROXIMATE ANALYSIS OF THE DIRECT SCHEME

In this section, we develop an *approximate* model for analyzing the delay properties of the efficient direct scheme. Recall that under this scheme each packet is broadcast along one of d particular completely unbalanced spanning trees rooted at the packet's origin; selection among the permissible trees is randomized, with all of them being a priori equiprobable (see also §6.3.2). The scheme involves no idling, that is, packets traverse the appropriate arcs as fast as possible, subject to contention.

As established in §6.3.3, the scheme is stable for all $\rho < 1$, which is the broadest possible stability region. Regarding the steady-state average delay T per packet, we only proved that $\lim_{\rho \to 0} T = d + \frac{1}{2}$. The behavior of T was further investigated by means of simulations, and it was observed that $T \approx d + \frac{1}{2} + \Theta(d)\rho$ for small ρ; below, we derive an approximate expression for T, which will be seen to be in excellent agreement

with the simulation outcomes when ρ is not large.

In the analysis to follow, it is assumed that the various random processes involved are in steady-state. We fix a node x. Let $Y_k^{(i)}$ be the number of packets waiting to cross arc $(x, x \oplus e_i)$ at the beginning of the kth slot, including the packet to be transmitted. Moreover, let B_k be the number of packets generated by node x during the kth slot; since arc $(x, x \oplus e_i)$ belongs to all of the d trees that may possibly be selected by a packet generated by x, all B_k newly generated packets will join the queue for this arc. Also, let $P_k^{(m,i)}$ be the number of packets received by node x (during the kth slot) through arc $(x \oplus e_m, x)$ and wishing to traverse arc $(x, x \oplus e_i)$. Notice that $P_k^{(m,i)} \in \{0, 1\}$; that is, all of these random variables are of the Bernoulli type. Clearly, we have

$$Y_{k+1}^{(i)} = [Y_k^{(i)} - 1]^+ + B_k + \sum_{m=1}^{d} P_k^{(m,i)} \ , \text{ for } k = 1, \ldots \ ; \tag{6.8}$$

recall that $[\alpha]^+ = \max\{\alpha, 0\}$.

By complete symmetry, the traffic is split *evenly* (on the average) among the various arcs. Thus, each of them is busy at a particular slot with probability ρ. Using also the definition of $P_k^{(m,i)}$, we obtain

$$E[P_k^{(m,i)}] = \rho g_{m,i} \ , \tag{6.9}$$

where $g_{m,i}$ is the probability that a packet has to cross arc $(x, x \oplus e_i)$ given that it has crossed arc $(x \oplus e_m, x)$. In Appendix 6.A, we prove that

$$g_{m,i} = \frac{2^{[(m-i) \bmod d]} - 1}{2^d - 1} \ , \qquad \forall (m,i) \in \{1, \ldots, d\}^2 \ ; \tag{6.10}$$

notice that, by symmetry among the various nodes, the parameters $(g_{m,i})_{(m,i) \in \{1,\ldots,d\}^2}$ are *independent* of x. Also, there holds $g_{i,i} = 0$, which is due to the fact that no packet having crossed arc $(x, x \oplus e_i)$ attempts to cross arc $(x \oplus e_i, x)$.

Next, we introduce the following *approximating assumptions:*

Assumption A: For any pair (m,i), the random variables $(P_k^{(m,i)})_{k=1,\ldots}$ are taken *independent* and identically distributed.

Assumption B: The processes $(P_k^{(1,i)})_{k=1,\ldots}, \ldots, (P_k^{(d,i)})_{k=1,\ldots}$ are taken to be mutually *independent.*

These two assumptions are of similar spirit as those in the approximate models of [AbP86] and [GrH89]; note, however, that those models pertain to the problem of multiple node-to-node communications of Chapter 5, which is considerably different than the present routing problem.

In the analysis to follow, all the equalities to be derived are approximate (unless otherwise specified), since they are based on the two assumptions above.

We define the random process $(A_k^{(i)})_{k=1,\ldots}$ as follows:

$$A_k^{(i)} \stackrel{\text{def}}{=} B_k + \sum_{m=1}^{d} P_k^{(m,i)} \,. \tag{6.11}$$

$(B_k)_{k=1,\ldots}$ is (actually) a renewal process, and is independent of $(P_k^{(1,i)})_{k=1,\ldots}, \ldots,$ $(P_k^{(d,i)})_{k=1,\ldots}$. Thus, under our approximation, $(A_k^{(i)})_{k=1,\ldots}$ is taken as a renewal process that assumes the distribution of a random variable $A^{(i)}$. Since B_k is a Poisson random variable with mean λ, it follows from (6.9) and (6.11) that

$$E[A^{(i)}] = \lambda + \sum_{m=1}^{d} \rho g_{m,i} \,, \tag{6.12}$$

and

$$\begin{aligned} \text{var}[A^{(i)}] &= \lambda + \sum_{m=1}^{d} \rho g_{m,i}(1 - \rho g_{m,i}) \\ &= \lambda + \rho \sum_{m=1}^{d} g_{m,i} - \rho^2 \sum_{m=1}^{d} g_{m,i}^2 \,, \end{aligned} \tag{6.13}$$

where $\text{var}[A^{(i)}]$ denotes the variance of $A^{(i)}$. (We have also used the fact that $P_k^{(m,i)}$ is a Bernoulli process, for $m = 1, \ldots, d$.) Using (6.10), we have

$$\begin{aligned} \sum_{m=1}^{d} g_{m,i} &= \sum_{m=1}^{d} \frac{2^{[(m-i) \bmod d]} - 1}{2^d - 1} \\ &= \frac{1}{2^d - 1} \sum_{m=1}^{d} 2^{[(m-i) \bmod d]} - \frac{d}{2^d - 1} \\ &= \frac{1}{2^d - 1} \sum_{m=1}^{d} 2^{m-1} - \frac{d}{2^d - 1} \\ &= 1 - \frac{d}{2^d - 1} \,. \end{aligned} \tag{6.14}$$

This together with (6.12), implies that

$$E[A^{(i)}] = \lambda + \rho\left(1 - \frac{d}{2^d - 1}\right) = \rho\,, \tag{6.15}$$

where we have also used the fact $\rho = \lambda\frac{2^d-1}{d}$. This result is *actually* true and could have been taken for granted, because $E[A^{(i)}]$ is the average traffic rate for arc $(x, x \oplus e_i)$, which equals ρ. Furthermore, using (6.10) and reasoning as in proving (6.14), it follows (after some algebra) that

$$\sum_{m=1}^{d} g_{m,i}^2 = \frac{1}{(2^d-1)^2}\left[d + \frac{1}{3}(4^d - 1) - 2(2^d - 1)\right];$$

this together with (6.13) and (6.14) implies that

$$\mathrm{var}[A^{(i)}] = \rho - \rho^2\frac{1}{(2^d-1)^2}\left[d + \frac{1}{3}(4^d - 1) - 2(2^d - 1)\right]. \tag{6.16}$$

Furthermore, combining (6.11) with (6.8), we obtain

$$Y_{k+1}^{(i)} = [Y_k^{(i)} - 1]^+ + A_k^{(i)}\,, \text{ for } k = 1, \ldots \tag{6.17}$$

Since the arrival process $(A_k^{(i)})_{k=1,\ldots}$ was taken as a renewal process, it follows from (6.17) that $(Y_k^{(i)})_{k=1,\ldots}$ may be approximated by the process of the number of customers in a discrete-time $G/D/1$ queue with unit service time. [Recall the definition of this queueing system in §2.2.1, and compare (2.1) with (6.17).] Let $D^{(i)}$ be the average delay associated with this queue; as usual, $D^{(i)}$ includes the "service" time [i.e., the transmission time over arc $(x, x \oplus e_i)$]. Using (2.3) in §2.1.2, we have

$$\begin{aligned} D^{(i)} &= 1 + \frac{E[A^{(i)}(A^{(i)} - 1)]}{2E[A^{(i)}](1 - E[A^{(i)}])} \\ &= 1 + \frac{\mathrm{var}[A^{(i)}] + (E[A^{(i)}])^2 - E[A^{(i)}]}{2E[A^{(i)}](1 - E[A^{(i)}])} \\ &= \frac{1}{2} + \frac{\mathrm{var}[A^{(i)}]}{2E[A^{(i)}](1 - E[A^{(i)}])}\,. \end{aligned}$$

Combining this with (6.15) and (6.16), we obtain

$$D^{(i)} = \frac{1}{2} + \frac{1}{2(1-\rho)}\left(1 - \frac{\rho}{(2^d-1)^2}\left[d + \frac{1}{3}(4^d - 1) - 2(2^d - 1)\right]\right), \text{ for } i = 1, \ldots, d. \tag{6.18}$$

Note that the above expression is independent of i, which is due to complete symmetry among the d dimensions of the hypercube.

So far, we have derived an approximate expression for the average delay suffered by a packet while waiting to cross an arc of the ith type. The overall delay of a packet will be approximated with the delay suffered in the *longest* path; this path consists of d arcs, one from each of the d hypercube dimensions. Thus, using (6.18) and taking also the average synchronization time into account, we obtain the following approximate formula for the average delay per packet:

$$T \approx \frac{d}{2} + \frac{d}{2(1-\rho)}\left(1 - \frac{\rho}{(2^d-1)^2}\left[d + \frac{1}{3}(4^d-1) - 2(2^d-1)\right]\right) + \frac{1}{2}\,. \qquad (6.19)$$

For d not being very small (say $d \geq 10$), the above formula may be simplified to the following:

$$T \approx \frac{d}{2} + \frac{d}{2(1-\rho)}(1 - \frac{\rho}{3}) + \frac{1}{2} = d + \frac{1}{2} + d\frac{\rho}{3(1-\rho)}\,.$$

Furthermore, for small ρ, we have

$$T \approx d + \frac{1}{2} + \frac{d}{3}\rho\,. \qquad (6.20)$$

As will be seen below, the approximate formula (6.19) is in excellent agreement with the simulation outcomes, for $\rho \leq 0.3$. This together with (6.20) supports the conjecture that, for small ρ, there holds $T \approx d + \frac{1}{2} + \Theta(d)\rho$. Also notice that the right-hand quantity in (6.19) is bounded from above by $d + \frac{1}{2} + \frac{d\rho}{2(1-\rho)}$, and thus satisfies the "ideal" performance objective presented in §3.2.3; unfortunately, (6.19) is only an approximation.

Next, we investigate the accuracy of (6.19). In Table 6.1, we compare the simulation outcomes for the 8-dimensional hypercube with the estimate given by (6.19) for $d = 8$; each of the simulation outcomes was obtained over a period of 5,000 slots. Clearly, there is *excellent agreement* between the experimental results and the corresponding approximate estimate for T, for values of ρ ranging from 0.05 to 0.3; for all such values of ρ, the magnitude of the relative error is less than 2%. In fact, the agreement for values of $\rho \leq 0.25$ is striking; in all simulations performed, the relative error did not exceed 1% for $\rho \leq 0.25$. Unfortunately, the accuracy of the approximate formula (6.19)

$d = 8$			
ρ	**Simulation**	**Approximation**	**Relative Error**
0.025	8.553	8.569	0.19%
0.050	8.620	8.641	0.24%
0.075	8.671	8.718	0.54%
0.100	8.763	8.799	0.41%
0.125	8.831	8.884	0.60%
0.150	8.950	8.974	0.27%
0.175	9.064	9.070	0.07%
0.200	9.165	9.172	0.08%
0.225	9.307	9.280	-0.29%
0.250	9.480	9.396	-0.89%
0.275	9.620	9.519	-1.05%
0.300	9.808	9.652	-1.59%
0.325	10.032	9.794	-2.37%
0.350	10.246	9.947	-2.92%
0.375	10.524	10.112	-3.91%
0.400	10.733	10.291	-4.12%
0.425	11.002	10.486	-4.69%
0.450	11.295	10.699	-5.28%
0.475	11.712	10.931	-6.67%
0.500	12.201	11.187	-8.31%

Table 6.1

deteriorates gradually for $\rho > 0.3$. In Table 6.2, we investigate the accuracy of the approximate formula (6.19) for different values of the hypercube dimension d; again, excellent agreement is observed for all $d = 5, \ldots, 10$ under moderately light traffic (namely, for $\rho = 0.1$, $\rho = 0.15$ and $\rho = 0.2$). The experimental results of Table 6.2 were obtained over periods of 1,000 slots.

APPENDIX 6.A

In this appendix, we prove equation (6.10) in §6.4; this equation is as follows:

$$g_{m,i} = \frac{2^{[(m-i) \bmod d]} - 1}{2^d - 1} . \qquad (6.A.1)$$

Recall that $g_{m,i}$ denotes the probability that a packet has to cross arc $(x, x \oplus e_i)$

$\rho = 0.10$			
d	**Simulation**	**Approximation**	**Relative Error**
5	5.659	5.696	0.65%
6	6.705	6.729	0.31%
7	7.729	7.763	0.44%
8	8.725	8.799	0.85%
9	9.806	9.835	0.30%
10	10.819	10.871	0.48%
$\rho = 0.15$			
d	**Simulation**	**Approximation**	**Relative Error**
5	5.800	5.811	0.19%
6	6.844	6.863	0.28%
7	7.881	7.918	0.47%
8	8.933	8.974	0.46%
9	10.043	10.031	-0.12%
10	11.091	11.089	-0.02%
$\rho = 0.20$			
d	**Simulation**	**Approximation**	**Relative Error**
5	5.894	5.940	0.78%
6	7.001	7.002	0.01%
7	8.102	8.092	-0.12%
8	9.177	9.172	-0.05%
9	10.227	10.253	0.25%
10	11.379	11.335	-0.39%

Table 6.2

given that it has to cross arc $(x \oplus e_m, x)$ (*); note that, as implied by $(6.A.1)$, $g_{m,i}$ is independent of x.

Clearly, we have $g_{i,i} = 0$, which agrees with $(6.A.1)$ for $m = i$. Henceforth, we assume that $m \neq i$.

We consider a fixed packet, denoted by $\mathcal{P}$. Let Γ_j be the event that $\mathcal{P}$ is a j-packet, namely that $\mathcal{P}$ crosses the hypercube dimensions under the order $j, (j \bmod d) + 1, \ldots, [(j + d - 2) \bmod d] + 1$; see also §6.3.2. Also, let Δ_l^z be the event that $\mathcal{P}$ has to cross

(*) The expression "the packet has to cross arc $(z, z \oplus e_j)$" should be interpreted as follows: "arc $(z, z \oplus e_j)$ belongs to the tree selected by the packet".

arc $(z, z \oplus e_l)$. Using this notation, we have

$$g_{m,i} = \Pr[\Delta_i^x | \Delta_m^{x \oplus e_m}] = \sum_{j=1}^{d} \Pr[\Delta_i^x | \Gamma_j \text{ and } \Delta_m^{x \oplus e_m}] \cdot \Pr[\Gamma_j | \Delta_m^{x \oplus e_m}] . \qquad (6.A.2)$$

Furthermore, we have

$$\Pr[\Gamma_j | \Delta_m^{x \oplus e_m}] = \frac{\Pr[\Delta_m^{x \oplus e_m} | \Gamma_j] \cdot \Pr[\Gamma_j]}{\sum_{l=1}^{d} \Pr[\Delta_m^{x \oplus e_m} | \Gamma_l] \cdot \Pr[\Gamma_l]} . \qquad (6.A.3)$$

There holds $\Pr[\Gamma_l] = \frac{1}{d}$ for $l = 1, \ldots, d$, because each of the d permissible trees has an a priori probability of $\frac{1}{d}$ to be selected by the fixed packet $\mathcal{P}$. Using this, it follows from (6.A.3) that

$$\Pr[\Gamma_j | \Delta_m^{x \oplus e_m}] = \frac{\Pr[\Delta_m^{x \oplus e_m} | \Gamma_j]}{\sum_{l=1}^{d} \Pr[\Delta_m^{x \oplus e_m} | \Gamma_l]} . \qquad (6.A.4)$$

Notice now that an l-packet generated at node y will have to traverse arc $(z, z \oplus e_m)$ if and only if it will reach node z *prior* to crossing the mth dimension (possibly allowing for $z = y$); recalling the order under which l-packets cross the hypercube dimensions, it is seen that this occurs if and only if one of the following holds:

(a) $m < l$ and $z_m = y_m, \cdots, z_{l-1} = y_{l-1}$;

(b) $m \geq l$ and $z_m = y_m, \cdots, z_d = y_d, z_1 = y_1, \cdots, z_{l-1} = y_{l-1}$, with an obvious interpretation for $l = 1$.

For fixed values of z, l and m, the above condition holds for $2^{[(m-l) \bmod d]}$ different nodes y. Since each node is a priori equiprobable to have generated the fixed packet $\mathcal{P}$, it follows that

$$\Pr[\Delta_m^z | \Gamma_l] = \frac{2^{[(m-l) \bmod d]}}{2^d} .$$

Applying this with $z = x \oplus e_m$ and using (6.A.4), we obtain

$$\Pr[\Gamma_j | \Delta_m^{x \oplus e_m}] = \frac{2^{[(m-j) \bmod d]}/2^d}{\sum_{l=1}^{d} 2^{[(m-l) \bmod d]}/2^d} = \frac{2^{[(m-j) \bmod d]}}{2^d - 1} , \qquad (6.A.5)$$

where we have used the fact $\sum_{l=1}^{d} 2^{[(m-l) \bmod d]} = \sum_{l=1}^{d} 2^{l-1} = 2^d - 1$.

Combining (6.A.2) with (6.A.5), we have

$$g_{m,i} = \sum_{j=1}^{d} \Pr[\Delta_i^x | \Gamma_j \text{ and } \Delta_m^{x \oplus e_m}] \frac{2^{[(m-j) \bmod d]}}{2^d - 1} . \qquad (6.A.6)$$

Conditioning on the union of the events Γ_j and $\Delta_m^{x\oplus e_m}$ is equivalent to conditioning on the fact "the j-packet $\mathcal{P}$ has to cross arc $(x \oplus e_m, x)$". Given this fact, $\mathcal{P}$ has to cross arc $(x, x \oplus e_i)$, with probability 1, if and only if it does *not* cross the ith dimension prior to crossing the mth dimension; otherwise, the probability that $\mathcal{P}$ has to cross $(x, x \oplus e_i)$ equals 0. Again, recalling the order under which j-packets cross the hypercube dimensions, it is seen that $\mathcal{P}$ will cross arc $(x, x \oplus e_i)$ if and only if one of the following holds: $j \leq m < i$, or $i < j \leq m$, or $m < i < j$. Based on the above discussion, we consider different three cases:

Case $m < i < d$**:** Using (6.*A*.6), we have

$$\begin{aligned} g_{m,i} &= \sum_{j=1}^{m} \frac{2^{[(m-j) \bmod d]}}{2^d - 1} + \sum_{j=i+1}^{d} \frac{2^{[(m-j) \bmod d]}}{2^d - 1} \\ &= \frac{1}{2^d - 1} \sum_{j=1}^{m} 2^{m-j} + \frac{1}{2^d - 1} \sum_{j=i+1}^{d} 2^{d+m-j} \\ &= \frac{1}{2^d - 1} \left(2^m - 1 + 2^{d+m-i} - 2^m\right) \\ &= \frac{1}{2^d - 1} \left(2^{d+m-i} - 1\right), \end{aligned}$$

which proves (6.*A*.1) for the present case, since $[(m - i) \bmod d] = d + m - i$.

Case $m < i = d$**:** Using (6.*A*.6), we have

$$g_{m,i} = \sum_{j=1}^{m} \frac{2^{[(m-j) \bmod d]}}{2^d - 1} = \frac{1}{2^d - 1} \sum_{j=1}^{m} 2^{m-j} = \frac{1}{2^d - 1} (2^m - 1),$$

which proves (6.*A*.1) for the present case, since $[(m - i) \bmod d] = [(m - d) \bmod d] = m$.

Case $m > i$**:** Using (6.*A*.6), we have

$$g_{m,i} = \sum_{j=i+1}^{m} \frac{2^{[(m-j) \bmod d]}}{2^d - 1} = \frac{1}{2^d - 1} \sum_{j=i+1}^{m} 2^{m-j} = \frac{1}{2^d - 1} (2^{m-i} - 1),$$

which obviously proves (6.*A*.1) for the present case.

The above three cases establish (6.*A*.1) for $m \neq i$; since (6.*A*.1) is trivially true for $m = i$, the proof is completed. **Q.E.D.**

7. Multiple Broadcasts in the Hypercube — Part II: Indirect Schemes

In this chapter, we continue with our analysis of the routing problem introduced in Chapter 6; namely, the dynamic problem of multiple broadcasts in the hypercube. In our previous analysis, we have established the performance objectives for the schemes of interest and we have investigated the performance of several *direct* schemes. In this chapter, we analyze the performance of an *indirect* scheme and discuss some variations thereof; our indirect scheme will prove to be rather efficient with respect to the performance objectives set in §6.2.3.

7.1 AN INDIRECT ROUTING SCHEME

7.1.1 Introduction

Consider the following simple routing scheme: *All* packets are sent to a specific node, which broadcasts them along a spanning tree emanating from itself. By pipelining

successive broadcasts, it is seen that this scheme can route only one broadcast per slot; thus, stability can be maintained only if $\lambda 2^d < 1$, or equivalently $\rho < \frac{1}{d}(1 - \frac{1}{2^d})$. Therefore, the boundary-value of the stability region is $O(\frac{1}{d})$. The reason for this poor performance is that only a fraction $\frac{1}{d}$ of the available hypercube arcs are used for broadcasting the packets.

The above discussion leads to the following idea: We consider the d disjoint spanning trees $\mathcal{T}^{(1)}, \ldots, \mathcal{T}^{(d)}$ introduced in [JoII89] (see §2.1.4). We assume that each packet is sent to one of nodes $e_1, \ldots, e_d$; each root node e_j broadcasts the packets it receives along spanning tree $\mathcal{T}^{(j)}$. If each of the d trees receives the *same* amount of traffic, then, stability may possibly be maintained even for $\rho = \Theta(1)$. A routing scheme of this spirit is presented in the next subsection and analyzed thereafter. Since packets are not broadcast directly by their respective origins, the scheme to be presented is characterized as *indirect*, contrary to the schemes introduced in §6.3.

It will be established *rigorously* that the routing scheme to be analyzed is stable for all $\rho < \frac{2}{3}(1 - \frac{1}{2^d}) \approx \frac{2}{3}$, while it satisfies $T \approx 3d + 1 + \frac{9}{4}\rho$ for small ρ. Thus, this scheme meets the performance objectives set in §6.2.3. It will also be seen to be *deadlock-free* when implemented with $\Theta(d)$ buffer capacity per node. In addition to the analysis, we discuss potential methods for devising even better indirect routing schemes, and we present some related open problems. Finally, in §7.4, we compare the efficient direct and indirect schemes.

7.1.2 The Rules of the Routing Scheme

In what follows, we present the set of rules for routing the packets. Rules A, B and C are the main ones, and require that a packet is sent to the root of one of the trees $\mathcal{T}^{(1)}, \ldots, \mathcal{T}^{(d)}$, from where it is actually broadcast; transmissions towards the roots may only be performed every three slots, while the rest of the time is dedicated to transmissions heading away from the roots. Rules D and E are only introduced for analytical tractability.

Rule A: Each packet generated at some node selects the tree along which it will be broadcast. Selection is *randomized*, with the only permissible trees being $\mathcal{T}^{(1)}, \ldots, \mathcal{T}^{(d)}$; each of them is assigned an a priori probability $\frac{1}{d}$. Different packets make their selections independently.

Rule B: Consider a packet, originating at some node y, that has chosen tree $\mathcal{T}^{(j)}$. This packet must be *sent to the root* e_j of this tree, which will actually perform the broadcast; the path to be followed is the reverse of the path from e_j to y that is contained in $\mathcal{T}^{(j)}$. That is, this packet will traverse the reverse of those arcs of $\mathcal{T}^{(j)}$ that lead from e_j to y. Note that packets generated by the root nodes $e_1, \ldots, e_d$ also follow Rules A and B, as well as the rules presented below. Thus, it may occur that a packet generated by node e_m is sent to some other root e_j, in order to be broadcast along $\mathcal{T}^{(j)}$.

Rule C: Consider an arc $(z, z \oplus e_i)$ belonging to $\mathcal{T}^{(j)}$, while its reverse arc $(z \oplus e_i, z)$ belongs to some other of the d disjoint trees, say to $\mathcal{T}^{(m)}$. Because of Rule B, it is possible that some packet to be broadcast along $\mathcal{T}^{(m)}$ has to traverse arc $(z, z \oplus e_i)$ while heading towards the root node e_m; we impose the restriction that such an arc traversal is permissible *only every three slots*. In order to make this rule more specific, we define $\mathcal{C}_0 \stackrel{\text{def}}{=} \{t \geq 0 : t \bmod 3 = 0\}$, $\mathcal{C}_1 \stackrel{\text{def}}{=} \{t \geq 0 : t \bmod 3 = 1\}$ and $\mathcal{C}_2 \stackrel{\text{def}}{=} \{t \geq 0 : t \bmod 3 = 2\}$. Arc $(z, z \oplus e_i)$ may be traversed by packets that have selected tree $\mathcal{T}^{(j)}$ (where the arc belongs) only during time slots in $\mathcal{C}_1 \cup \mathcal{C}_2$. Moreover, this arc may be traversed by packets that have selected tree $\mathcal{T}^{(m)}$ [where its reverse arc $(z \oplus e_i, z)$ belongs] only during time slots in $\mathcal{C}_0$. Thus, every three slots, each of the d trees is *reversed*, and all of its arcs point towards its root. Slots in the set $\mathcal{C}_0$ are actually used for sending packets to the respective root nodes, while slots in the set $\mathcal{C}_1 \cup \mathcal{C}_2$ are used for the broadcasts. As mentioned in §2.1.4, arcs of the form $(0, e_i)$ do not belong to any of the d disjoint trees; these d arcs are only used during slots in $\mathcal{C}_0$.

Rule D: Consider a packet generated at some node y that has selected tree $\mathcal{T}^{(j)}$. If y is *not a leaf* of $\mathcal{T}^{(j)}$, then before the packet considers to cross the first arc of its path to e_j, it has to traverse one *virtual* arc located at node y (see Figure 7.1, for $d = 3$). Such arcs may be traversed only during slots in $\mathcal{C}_0$. It is assumed that packets to be routed along different trees have to cross different virtual arcs, even if they have been generated at the same node. It is straightforward that virtual arcs can be realized by appropriately delaying packets in their respective origins.

Rule E: Every root node e_j has a pair of buffers B_1 and B_2. Consider a packet that has selected tree $\mathcal{T}^{(j)}$; let node y be the origin of this packet. If y belongs to the largest subtree $\mathcal{T}_1^{(j)}$ of $\mathcal{T}^{(j)}$, then the packet will be placed in buffer B_1; if y belongs to any

subtree other than $\mathcal{T}_1^{(j)}$ (or if $y = e_j$), then it will be placed in buffer B_2; see Figure 7.1. (The notation for the d subtrees of $\mathcal{T}^{(j)}$ is similar to that introduced in §2.1.3.) During slots $t+1, t+2$, with $t \in \mathcal{C}_0$, root node e_j broadcasts one packet from each of buffers B_1 and B_2. Which of the two packets will be broadcast first is determined by tossing a fair coin. In the case where one of the two buffers is empty, *only one* packet is broadcast; again, the slot when the broadcast will start is determined by tossing a fair coin. Of course, if both B_1 and B_2 are empty, then no further action is taken.

As already mentioned, Rules D and E are introduced for analytical tractability; instead of commenting on them at this point, we proceed with the analysis of the routing scheme, which will reveal the raison d' être for these rules. More discussion on the proposed scheme is presented in §7.3.

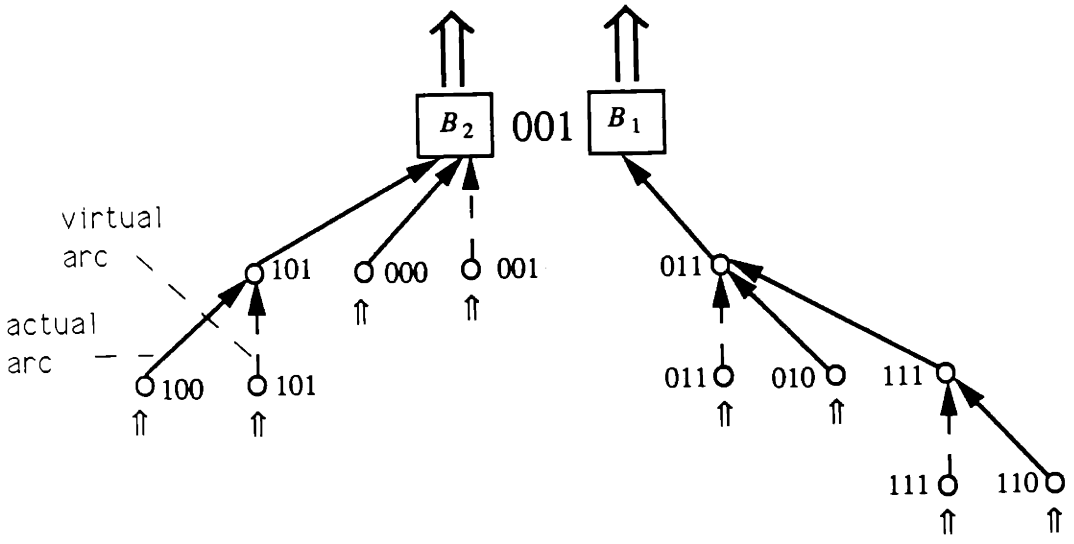

Figure 7.1: Introducing the virtual arcs and buffers B_1 and B_2 in $\mathcal{T}^{(1)}$, in the 3-cube.

7.2 PERFORMANCE ANALYSIS OF THE INDIRECT SCHEME

In this section, we analyze the performance of the indirect routing scheme introduced in §7.1. First, we establish some technical results to be used in the analysis to follow.

7.2.1 Auxiliary Results

First, we consider a tree $\mathcal{T}$ of n paths of the *same length*, with all paths having their *final* arc in common. Packets arrive at the starting nodes $s_1, \ldots, s_n$ of the paths and exit only at the common end f (see Figure 7.2a); packets are stored in the intermediate nodes of the tree (if necessary) and are forwarded as soon as possible. All packet transmissions start at the beginning of slots and each of them lasts for one slot. We claim that if we *collapse* all paths into a *single* path P (with the same length as before) and we *combine* the arrival processes, then the departure process at node f will remain the same (see Figure 7.2b). This result is proved in Lemma 7.1, and it is basically a consequence of synchronization and pipelining.

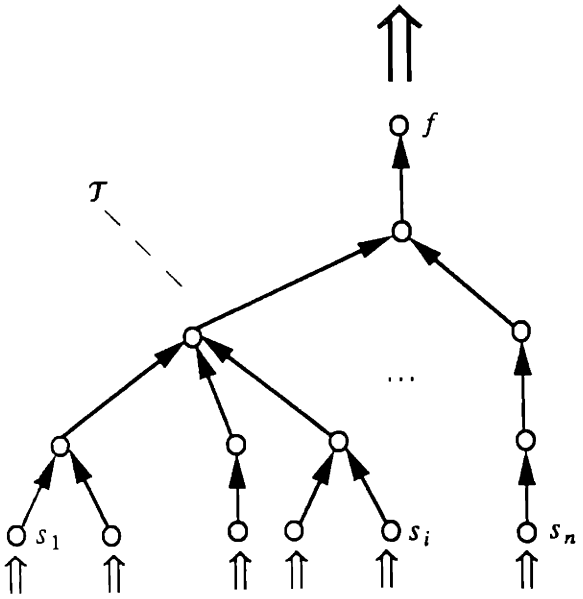

Figure 7.2a: The tree $\mathcal{T}$ of paths.

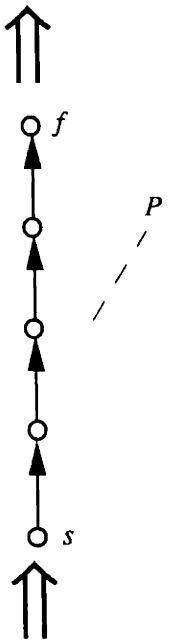

Figure 7.2b: The single path P.

In the context of the tree $\mathcal{T}$ of paths, we denote by $A_i(t)$ the number of packets that arrive at node s_i just before the end of slot t. Moreover, we denote by $F(t)$ the number of packets that depart from node f at slot t; clearly, $F(t)$ equals either 0 or 1.

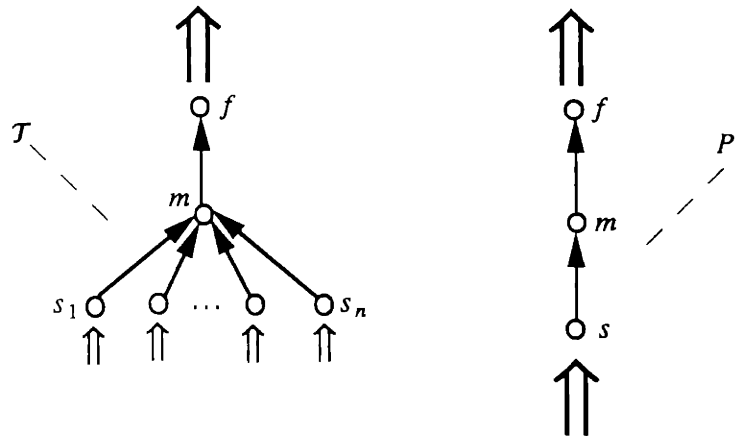

Figure 7.3 The simple case for Lemma 7.1.

In the context of the single path P, we define $\tilde{A}(t)$ and $\tilde{F}(t)$ in a similar way. All the above processes are defined for $t = 0, \ldots$; both systems start operating at time $t = 0$. The result to be proved is as follows:

Lemma 7.1: If $\tilde{A}(t) = \sum_{i=1}^{n} A_i(t)$ for $t = 0, \ldots$, then $\tilde{F}(t) = F(t)$ for $t = 0, \ldots$ ■

Proof: First, we consider the case where all paths have length 2 (see Figure 7.3). We denote by $M(t)$ a *binary* variable that equals 1 if and only if there is some packet buffered at node m of the tree $\mathcal{T}$ just before the end of slot t. The variable $\tilde{M}(t)$ refers to the single path P, and is defined in a similar way. Below, we prove that

$$\tilde{F}(t) = F(t) \quad \text{and} \quad \tilde{M}(t) = M(t)\,, \quad \text{for } t = 0, \ldots \tag{7.1}$$

The proof will be done by induction on t.

Clearly, we have $\tilde{M}(0) = M(0) = 0$ and $\tilde{F}(0) = F(0) = 0$, which proves (7.1) for $t = 0$. Next, we assume that the induction hypothesis holds for all $t \in \{0, \ldots, t^*\}$; based on this, we shall prove that it holds for $t = t^* + 1$, as well. Indeed, we have $F(t^* + 1) = M(t^*)$, because a packet may depart from the tree $\mathcal{T}$ at the end of slot $t^* + 1$ only if it were present at node m at the end of slot t^*. Similarly, we have $\tilde{F}(t^* + 1) = \tilde{M}(t^*)$. By the induction hypothesis, we have $\tilde{M}(t^*) = M(t^*)$; thus, it follows that $\tilde{F}(t^* + 1) = F(t^* + 1)$, which establishes the first part of (7.1) for $t = t^* + 1$. There remains to show that $\tilde{M}(t^* + 1) = M(t^* + 1)$. We have $M(t^* + 1) = 1$ if and only if some packets that arrived at the tree by the end of slot

t^* have not departed by the end of slot $t^* + 1$. That is, we have $M(t^* + 1) = 1$ if and only if $\sum_{t=0}^{t^*} \sum_{i=1}^{n} A_i(t) > \sum_{t=0}^{t^*+1} F(t)$. Similarly, we have $\tilde{M}(t^* + 1) = 1$ if and only if $\sum_{t=0}^{t^*} \tilde{A}(t) > \sum_{t=0}^{t^*+1} \tilde{F}(t)$. Recall now that $\tilde{A}(t) = \sum_{i=1}^{n} A_i(t)$ for $t = 0, \ldots,$ by assumption. Moreover, we have $\tilde{F}(t) = F(t)$ for $t = 0, \ldots, t^*$ (by the induction hypothesis) and we have established that $\tilde{F}(t^* + 1) = F(t^* + 1)$. Thus, it follows that if $M(t^* + 1) = 1$ holds then $\tilde{M}(t^* + 1) = 1$ also holds, and vice versa. Clearly, this implies that $\tilde{M}(t^* + 1) = M(t^* + 1)$; the proof of (7.1) for $t = t^* + 1$ is complete.

Now that the lemma has been established for the simple case in Figure 7.3, the result is easily extended for the general case in Figure 7.2. It suffices to progressively collapse the paths of the tree $\mathcal{T}$, starting from the lowest level. **Q.E.D.**

Lemma 7.1 holds even if packets arrive according to some continuous time process, provided that all packet transmissions start at the beginning of slots. Also, the lemma still holds if packet transmissions can only start at the beginning of slots numbered $0, \Delta, \ldots$ and each transmission lasts for Δ slots. On the other hand, Lemma 7.1 does not hold if some of the paths have different length than the others; nevertheless, we are still able to prove an interesting result applying to such a case; this result is discussed next.

We now consider a tree $\tilde{\mathcal{T}}$ consisting of n paths with possibly *different* lengths $l_1, \ldots, l_n$; see Figure 7.4, where $l_1 = 4$ and $l_n = 2$. Again, new packets enter the tree at the leaves and exit at the common end f. For each of the starting nodes $s_1, \ldots, s_n$, new packets are now assumed to arrive according to a *Poisson* processes with rate λ; arrivals at different starting nodes are mutually independent. Transmissions may only start at the beginning of slots numbered $0, \Delta, \ldots,$ and each of them lasts for Δ slots. Let $\tilde{D}$ denote the steady-state average delay per packet induced in the tree $\tilde{\mathcal{T}}$. Below, we present the stability condition for $\tilde{\mathcal{T}}$ and the expression for $\tilde{D}$; these results are proved in Appendix 7.A.

Lemma 7.2: The tree $\tilde{\mathcal{T}}$ of paths is a stable queueing system if and only if $\lambda n \Delta < 1$. Moreover, in the stable case, the steady-state average delay $\tilde{D}$ per packet is given as follows:

$$\tilde{D} = \Delta\left[\frac{3}{2} + \frac{\lambda n \Delta}{2(1 - \lambda n \Delta)}\right] + \frac{\Delta}{n} \sum_{i=1}^{n} l_i - \Delta \,. \qquad \blacksquare$$

Lemma 7.2 will be seen to be very useful in the subsequent analysis.

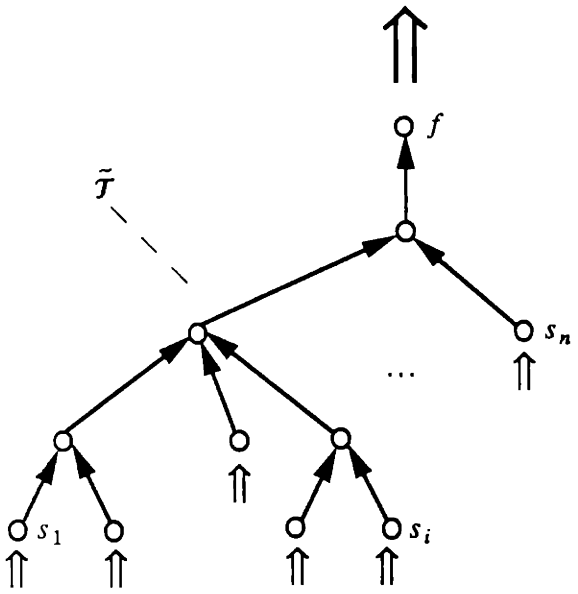

Figure 7.4 The tree $\tilde{T}$ of paths with different lengths.

7.2.2 The Condition for Stability

We begin with an observation to be used throughout the analysis. In particular, we notice that packets routed along different trees *do not interfere* at all, in the following sense: Two such packets will never attempt to traverse the same arc at the same time. Indeed, if both packets are under broadcast, then they cannot collide, because they have to traverse disjoint sets of arcs. (Recall that the d trees used for routing are disjoint.) If both packets are heading towards the respective root nodes, again their paths are disjoint. (Notice that the d disjoint trees remain disjoint after reversing their arcs.) Even if these packets have been generated at the same node and are still traversing the corresponding virtual arcs, they cannot collide, as guaranteed by Rule D. Finally, if one of the packets is under broadcast and the other is heading towards the corresponding root, then they may not both take a step at the same time, because of Rule C.

Since packets routed along different trees do not interfere, we may analyze the performance of the scheme *separately* for each tree. In fact, we have to consider only one of them, because the d trees are isomorphic and are treated symmetrically by the routing rules. For the rest of the analysis, we analyze the queueing phenomena

involving packets routed along $\mathcal{T}^{(1)}$.

First, we derive the condition for stability of the scheme:

Proposition 7.3: The indirect scheme under analysis is stable if and only if

$$\rho < \frac{2}{3}\left(1 - \frac{1}{2^d}\right). \tag{7.2}$$

Proof: Clearly, after a packet starts being broadcast, it does not encounter any additional contention, because successive broadcasts are pipelined. Thus, the traffic load that can be accommodated by the scheme is determined exclusively by the processes of packets departing from buffers B_1 and B_2. Due to Rule E, these two processes are *independent*; hence, the stability region of the scheme coincides with the *intersection* of the stability regions obtained by considering each of buffers B_1 and B_2.

First, let us consider the set of all paths leading to buffer B_1 of e_1. Since these paths originate from nodes in the largest subtree $\mathcal{T}_1^{(1)}$ of $\mathcal{T}^{(1)}$, they have their final arc in common [namely, arc $(e_2 \oplus e_1, e_1)$]; see Figure 7.1. Moreover, because of Rule C, all transmissions in these paths take place every 3 slots; finally, due to the *virtual* arcs added in the non-leaf nodes of $\mathcal{T}^{(1)}$ (see Rule D), all new packets are now generated at "leaves", each fed by a Poisson process with rate $\frac{\lambda}{d}$. (Notice that, due to Rule D, each non-leaf node of $\mathcal{T}^{(1)}$ is converted to a "leaf" hanging from a virtual arc.) These properties of the set of paths leading to buffer B_1 allow us to apply Lemma 7.2 with $\Delta = 3$, $n = 2^{d-1}$ and with $\frac{\lambda}{d}$ instead of λ. (Recall that B_1 is in charge of all packets originating in the largest subtree $\mathcal{T}_1^{(1)}$, which contains 2^{d-1} nodes; see §2.1.3.) Thus, the set of these paths is stable if and only if $3\frac{\lambda}{d}2^{d-1} < 1$; by the definition of ρ in (7.1), this condition is equivalent to (7.2).

Next, we consider the process of packets departing from buffer B_2 of e_1. The paths leading to this buffer do *not* have their final arc in common. Nevertheless, the process departing from B_2 is the *same* as if these paths were sharing their final arcs; this is due to the fact that packets depart from B_2 *one-by-one* and every three slots (because of Rule E), which would also hold if the paths leading to B_2 had their final arc in common. Again, we may apply Lemma 7.2 with $\Delta = 3$, $n = 2^{d-1}$ and with $\frac{\lambda}{d}$ instead of λ. (Recall that all nodes that do not belong to subtree $\mathcal{T}_1^{(1)}$ send their packets to buffer B_2; since $\mathcal{T}_1^{(1)}$ contains 2^{d-1} nodes, there are 2^{d-1} nodes remaining, including e_1.) Thus, it follows that the set of paths leading to B_2 is stable if and only if $3\frac{\lambda}{d}2^{d-1} < 1$, which is equivalent to (7.2).

So far, we have established that both sets of paths leading to buffers B_1 and B_2 are stable if and only if (7.2) holds. Thus, as argued in the beginning of the proof, condition (7.2) is necessary and sufficient for our routing scheme to be stable. **Q.E.D.**

7.2.3 Derivation of the Average Delay and the Average Queue-Size

Next, we derive the expression for the steady-state average delay T per packet under our indirect scheme.

Proposition 7.4: The average delay T for the indirect scheme under analysis is given as follows:

$$T = 3d + 1 + \frac{3\rho}{2[\frac{2}{3}(1 - \frac{1}{2^d}) - \rho]} \,. \qquad \blacksquare$$

Proof: The delay T may be expressed as the sum of two terms R and V, where R is the average time for a packet to reach e_1 and exit from the corresponding buffer, and V is the average additional time until the packet's broadcast is completed. As already argued in the proof of Proposition 7.3, the set of paths leading to buffers B_1 (resp. B_2) of e_1 satisfies the conditions of Lemma 7.2 with $\Delta = 3$, $n = 2^{d-1}$ and with $\frac{\lambda}{d}$ instead of λ; thus, the average time R_1 (resp. R_2) elapsing until a packet exits from B_1 (resp. B_2) is given as follows:

$$R_1 = 3\Big[\frac{3}{2} + \frac{3\frac{\lambda}{d}2^{d-1}}{2(1 - 3\frac{\lambda}{d}2^{d-1})}\Big] + \frac{3}{2^{d-1}} \sum_{i=1}^{2^d-1} l_i^{(1)} - 3 \,, \tag{7.3}$$

and

$$R_2 = 3\Big[\frac{3}{2} + \frac{3\frac{\lambda}{d}2^{d-1}}{2(1 - 3\frac{\lambda}{d}2^{d-1})}\Big] + \frac{3}{2^{d-1}} \sum_{i=1}^{2^d-1} l_i^{(2)} - 3 \,, \tag{7.4}$$

where $l_i^{(j)}$ is the length of the ith path leading to buffer B_j (for $j = 1, 2$), with paths numbered arbitrarily. Since each of buffers B_1 and B_2 is in charge of broadcasting the packets originating from 2^{d-1} hypercube nodes, an arbitrary packet routed along $\mathcal{T}^{(1)}$ is equally likely to be traveling in either of the two sets of paths. Thus, using (7.3) and (7.4), we have

$$R = \frac{1}{2}(R_1 + R_2) = 3\Big[\frac{3}{2} + \frac{3\frac{\lambda}{d}2^{d-1}}{2(1 - 3\frac{\lambda}{d}2^{d-1})}\Big] + \frac{3}{2^d}\Big[\sum_{i=1}^{2^d-1} l_i^{(1)} + \sum_{i=1}^{2^d-1} l_i^{(2)}\Big] - 3 \,. \tag{7.5}$$

Clearly, $\sum_{i=1}^{2^d-1} l_i^{(1)} + \sum_{i=1}^{2^d-1} l_i^{(2)}$ equals the sum of the Hamming distances $H(y, e_1)$ of all nodes y from root e_1 *plus* a unit contribution per *non-leaf* node in $\mathcal{T}^{(1)}$; the latter contribution accounts for the virtual arcs added by Rule D. We have

$$\sum_{y=0}^{2^d-1} H(y, e_1) = \sum_{k=0}^{d} k \binom{d}{k} = d2^{d-1} ,$$

because there are $\binom{d}{k}$ nodes at Hamming distance k from each fixed node z; also note that $\mathcal{T}^{(1)}$ has 2^{d-1} non-leaf nodes, because it is a completely unbalanced spanning tree (see §2.1.3). Therefore, it follows from (7.5) that

$$\begin{aligned} R &= 3\Big[\frac{3}{2} + \frac{3\frac{\lambda}{d}2^{d-1}}{2(1-3\frac{\lambda}{d}2^{d-1})}\Big] + \frac{3}{2^d}(d2^{d-1} + 2^{d-1}) - 3 \\ &= \frac{3}{2}d + 3 + \frac{3\rho}{2[\frac{2}{3}(1-\frac{1}{2^d}) - \rho]} , \end{aligned} \tag{7.6}$$

where we have also used the definition of ρ, namely $\rho \stackrel{\text{def}}{=} \lambda\frac{2^d-1}{d}$.

Once a packet exits from the corresponding buffer, the time required for its broadcast to be completed depends on the slot when the broadcast starts. Thus, if the broadcast starts at a slot in $\mathcal{C}_1$, then it is completed in time $\lceil\frac{d}{2}\rceil + d - 1$. If the broadcast starts at a slot in $\mathcal{C}_2$, then it is completed in time $\lfloor\frac{d}{2}\rfloor + d$. Since both these events occur with probability $\frac{1}{2}$ (because of Rule E), it follows that

$$V = \frac{1}{2}\Big(\Big\lceil\frac{d}{2}\Big\rceil + d - 1 + \Big\lfloor\frac{d}{2}\Big\rfloor + d\Big) = \frac{3}{2}d - \frac{1}{2} . \tag{7.7}$$

Recalling the definitions of R and V, it is seen that

$$T = R + V - \frac{3}{2} . \tag{7.8}$$

The correction term $-\frac{3}{2}$ is due to the following fact: In estimating R, each packet is considered as departing from B_1 (or from B_2) at some time $(t+3) \in \mathcal{C}_0$, while the packet *actually* starts being broadcast either at time $t+1$ or at $t+2$; thus, an average of $\frac{3}{2}$ slots per packet is counted in *both* R and V. The proof is completed by combining (7.6), (7.7) and (7.8). **Q.E.D.**

It follows from Proposition 7.4 that, under light traffic, we have $T \approx 3d + 1 + \frac{9}{4}\rho$. Therefore, the delay induced by the scheme under analysis meets the objective set in

§6.2.3. It is also worth noting that the first-order term in T is $\Theta(1)\rho$, contrary to the efficient direct scheme, for which the corresponding term was observed to be $\Theta(d)\rho$ (see §6.5).

Next, we study the steady-state average queue-size Q per node. Under our indirect scheme, the statistics of the queue-size at a node x *depend* on the position of x in each of the d disjoint trees. (Note that there was no such asymmetry under the direct schemes of §6.3.3.) For example, node 0 is a leaf in all d trees; hence, node 0 only stores packets of its own, which implies that the queues built therein are relatively short. In order to obtain an "overall" estimate of the sizes of the queues built in the various nodes, we now define Q as $\frac{1}{2^d}$ of the average *total* number of packets stored in the entire network. (The number of packets stored at each node is determined in the same way as in the definition of Q in §6.4.) Below, we derive the expression for Q.

Proposition 7.5: There holds

$$Q = \frac{3d}{4}\rho\frac{2^d-2}{2^d-1} + 3\rho\frac{d}{2^d-1}\left(\frac{3}{2}d + \frac{17}{6} + \frac{3\rho}{2[\frac{2}{3}(1-\frac{1}{2^d})-\rho]}\right) . \qquad \blacksquare$$

Proof: This proof is similar to that of Proposition 7.4. First, we consider only packets that have chosen tree $\mathcal{T}^{(1)}$. Let N be the average total number of such packets stored in the network. By symmetry, we have

$$Q = \frac{dN}{2^d} . \tag{7.9}$$

Clearly, there holds

$$N = N_r + N_b , \tag{7.10}$$

where N_r is the average total number of packets heading towards root e_1 (including the ones already in buffers B_1 and B_2) and N_b is the average total number of stored packets undergoing broadcast. Since successive broadcasts are pipelined, each *non-leaf* node of $\mathcal{T}^{(1)}$ stores (at each slot) at most one of the packets undergoing broadcast along $\mathcal{T}^{(1)}$. Each of the two buffers B_1 and B_2 is fed at a rate of $3\frac{\lambda}{d}2^{d-1}$ (see the proof of Proposition 7.3); since these two buffers are served alternately (because of Rule E), the steady-state probability that a non-leaf node (excluding e_1) stores a packet undergoing broadcast equals $3\frac{\lambda}{d}2^{d-1}$; root e_1 is excluded from the calculation of N_b,

because the packets stored there will be taken into account in the computation of N_r. Since $\mathcal{T}^{(1)}$ has 2^{d-1} non-leaf nodes, it follows from the above discussion that

$$N_b = (2^{d-1}-1)\left(3\frac{\lambda}{d}2^{d-1}\right) = 3\rho\frac{2^{d-1}(2^{d-1}-1)}{2^d-1}\,. \tag{7.11}$$

On the other hand, applying Little's formula, we have

$$N_r = \left(3\frac{\lambda}{d}2^{d-1}\right)R_1 + \left(3\frac{\lambda}{d}2^{d-1}\right)R_2\,, \tag{7.12}$$

where each of the two terms accounts for the contribution of the sets of paths leading to buffers B_1 and B_2 respectively. Recall that R_1 (resp. R_2) is the average time until a packet exits from B_1 (resp. B_2); see also the proof of Proposition 7.4. Using the fact $R_1 + R_2 = 2R$ [see (7.5)] and (7.6), it follows from (7.12) that

$$N_r = 2\left(3\frac{\lambda}{d}2^{d-1}\right)R = 3\rho\frac{2^d}{2^d-1}\left(\frac{3}{2}d + 3 + \frac{3\rho}{2[\frac{2}{3}(1-\frac{1}{2^d})-\rho]}\right)\,. \tag{7.13}$$

Notice, however, that in the above expression for N_r, a packet whose broadcast starts at time $(t+1) \in \mathcal{C}_1$ is *also* considered as stored in the corresponding buffer at time $t+2$. Therefore, with probability $\frac{1}{2}$, the *topmost* packet in buffer B_1 (resp. B_2) contributes to N_r an average of $\frac{2}{3}$ instead of 1. (Notice that a packet stays at the top of the buffer B_1 or B_2 for only one "frame" of 3 slots.) Since B_1 (resp. B_2) is non-empty at time $t \in \mathcal{C}_0$ with probability equal to its average total input rate (namely, $3\frac{\lambda}{d}2^{d-1}$), it is seen that the contribution of B_1 (resp. B_2) to N_r was *overestimated* by $\frac{1}{2}(3\frac{\lambda}{d}2^{d-1})(1-\frac{2}{3})$. Therefore, the correct expression for N_r is that of (7.13) *reduced* by $\frac{\lambda}{d}2^{d-1}$. (This correction is similar to that introduced at the end of the proof of Proposition 7.4.) Thus, it follows from (7.13) that

$$N_r = 3\rho\frac{2^d}{2^d-1}\left(\frac{3}{2}d + \frac{17}{6} + \frac{3\rho}{2[\frac{2}{3}(1-\frac{1}{2^d})-\rho]}\right)\,.$$

Combining this with (7.10) and (7.11), we obtain

$$N = 3\rho\frac{2^{d-1}(2^{d-1}-1)}{2^d-1} + 3\rho\frac{2^d}{2^d-1}\left(\frac{3}{2}d + \frac{17}{6} + \frac{3\rho}{2[\frac{2}{3}(1-\frac{1}{2^d})-\rho]}\right)\,;$$

this together with (7.9) proves the result. **Q.E.D.**

Proposition 7.5 implies that, for small ρ, we have $Q \approx \frac{3d}{4}\rho\frac{2^d-2}{2^d-1} + 3\rho\frac{d}{2^d-1}(\frac{3}{2}d + \frac{17}{6})$; when d is not very small, this simplifies to $Q \approx \frac{3d}{4}\rho$.

7.3 DISCUSSION ON THE SCHEME — FURTHER RESULTS

Relatively to the complexity of our routing problem, the indirect scheme introduced in §7.1.2 has proved to be rather tractable analytically. Since the d trees used for routing are disjoint, Rules A, B and C resulted in the *decoupling* of packets that are routed along different trees. Thus, queueing analysis within each of the d trees was done separately, which was very convenient. Furthermore, due to Rule E, packets routed along the same tree $\mathcal{T}^{(j)}$ were split into two *non-interfering* classes; namely, the class of packets originating in the largest subtree $\mathcal{T}_1^{(j)}$, and the class of packets originating elsewhere. Finally, by introducing the virtual arcs (Rule D), the performance measures of each of the two classes were derived by means of Lemma 7.2, which proved to be a rather powerful result.

As already argued, our indirect scheme meets the performance objectives set in §6.2.3. Though rather satisfactory, the stability properties of the scheme are (unfortunately) non-optimal, since the scheme becomes unstable for $\rho \approx \frac{2}{3}$. It is rather unlikely that the load factor ρ would ever approach this stability limit in a practical application (see §6.2.3); nevertheless, some further improvement of this upper limit may be desirable, since it would also improve the delay properties for moderate loads. In this section, we investigate potential methods for improving the stability properties of the scheme. We also discuss the issue of deadlock prevention, when implementing our indirect scheme with finite buffers.

7.3.1 Limitations on the Stability Properties

Under our indirect scheme, one third of the time (namely, all slots in $\mathcal{C}_0$) is dedicated to performing transmissions towards the roots $e_1, \ldots, e_d$; see Rule C. However, these transmissions constitute a very small portion of those performed overall. Indeed, consider a packet that is generated at some node y, which is at Hamming distance k from root e_j. If this packet selects to be routed along $\mathcal{T}^{(j)}$, then it will undertake $k + 2^d - 1$ transmissions; k of them are required for the packet to reach e_j (due to Rule B) and the rest $2^d - 1$ are undertaken during the broadcast performed by the root e_j. In fact, k of the transmissions heading away from the root are *redundant*, because they

bring the packet to nodes that have already received it.

The above discussion seems to suggest that Rule B and especially Rule C result in a considerable decrease of the maximum load factor that can be accommodated by the routing scheme. However, this claim is *not* correct, as proved below.

Proposition 7.6: Consider a routing scheme that splits evenly the packets of *each* individual node among the d trees $\mathcal{T}^{(1)},\ldots,\mathcal{T}^{(d)}$; each packet assigned to tree $\mathcal{T}^{(j)}$ is only permitted to use the arcs of $\mathcal{T}^{(j)}$ together with their reverse arcs. Then, such a scheme may be stable *only if*

$$\rho < \frac{2}{3}\Big(1 - \frac{1/3}{2^d - 2/3}\Big)\,. \tag{7.14}$$

Proof: We fix some $j \in \{1,\ldots,d-2\}$. We consider node $e_j \oplus e_{j+1}$, which is the neighbor of e_j through the $(j+1)$st dimension; similarly, $e_j \oplus e_{j+1}$ is the neighbor of e_{j+1} through the jth dimension.

In the context of tree $\mathcal{T}^{(j)}$, the hypercube dimensions are crossed in the order $j+1,\ldots,d,1,\ldots,j$; see §2.1.4. Therefore, the largest subtree $\mathcal{T}_1^{(j)}$ is hanging from node $e_j \oplus e_{j+1}$. This implies that all packets originating at any node of $\mathcal{T}_1^{(j)}$ and routed along $\mathcal{T}^{(j)}$ have to traverse arc $(e_j \oplus e_{j+1}, e_j)$, in order to be received by root e_j. Since subtree $\mathcal{T}_1^{(j)}$ contains 2^{d-1} nodes, these packets represent an average total demand for $\frac{\lambda}{d}2^{d-1}$ transmissions over arc $(e_j \oplus e_{j+1}, e_j)$ per time unit. (Recall that the packets of each individual node are split evenly among the d trees.)

In the context of tree $\mathcal{T}^{(j+1)}$, the hypercube dimensions are crossed in the order $j+2,\ldots,d,1,\ldots,j+1$. Therefore, nodes $e_j \oplus e_{j+1}$ and e_j comprise the *second smallest* subtree $\mathcal{T}_{d-1}^{(j+1)}$, which is hanging node from $e_j \oplus e_{j+1}$; node e_j is a leaf. Clearly, all packets generated at any node other than e_j and routed along $\mathcal{T}^{(j+1)}$ have to traverse arc $(e_j \oplus e_{j+1}, e_j)$, in order to be received by e_j. These packets represent an average total demand for $\frac{\lambda}{d}(2^d - 1)$ transmissions over arc $(e_j \oplus e_{j+1}, e_j)$ per time unit.

Any routing scheme may be stable only if the average total demand per time unit for transmissions over any fixed arc is smaller than unity. Considering arc $(e_j \oplus e_{j+1}, e_j)$, it follows from the previous discussion that any routing scheme of the class specified in the proposition may be stable only if

$$\frac{\lambda}{d}2^{d-1} + \frac{\lambda}{d}(2^d - 1) < 1\,;$$

recalling the definition of ρ (namely $\rho \stackrel{\text{def}}{=} \lambda \frac{2^d - 1}{d}$), it is seen that the inequality above is equivalent to (7.14). **Q.E.D.**

Notice that the right-hand quantity in (7.2) is very close to that in (7.14), even for moderately small values of d; e.g., for $d = 8$ they differ by 0.26%, while for $d = 10$ they differ by 0.065%. Thus, the stability region under our indirect scheme is nearly optimal over the class of schemes specified in Proposition 7.6; in fact, it is not known whether the stability region given by (7.14) is attainable by any such scheme.

It should also be noted that, *given* Rules A and B, the choice of Rule C was not "accidental". Indeed, consider Rule C in its most general version, by assuming that transmissions towards the roots are performed over K successive slots, while the broadcasts are performed over L successive slots; it will be shown that $K = 1$ and $L = 2$ (which coincides with Rule C) constitutes the *best* choice. We focus on the packets routed along tree $\mathcal{T}^{(1)}$. Since only L such packets may be broadcast every $K + L$ slots, it follows that stability may apply only if

$$\frac{\lambda}{d} 2^d (K + L) < L \,; \tag{7.15}$$

moreover, every $K + L$ slots, only K packets generated in the largest subtree $\mathcal{T}_1^{(1)}$ may be forwarded to root e_1 through arc $(e_2 \oplus e_1, e_1)$; thus, the inequality

$$\frac{\lambda}{d} 2^{d-1} (K + L) < K$$

is also necessary for stability. This together with (7.15) implies that there should hold

$$\frac{\lambda}{d} 2^d < \min \left\{ \frac{L}{K + L}, \frac{2K}{K + L} \right\} . \tag{7.16}$$

It is easily seen that the right-hand quantity in (7.16) is *maximized* for $L = 2K$. Thus, choosing $K = 1$ and $L = 2$ gives the most favorable *potential* stability region. This stability region is actually *attained* by our scheme, because, for $K = 2L$, inequality (7.18) coincides with the sufficient condition for stability (7.2).

7.3.2 Potential Methods for Improving the Stability Properties

In this subsection, we discuss some modifications in the rules of §7.1.2 that could possibly result in improvement of the stability properties of the scheme; unfortunately, most of the results to be presented are *negative*.

We shall be using the terminology *input load* of a subtree $\mathcal{T}_i^{(j)}$ to denote the average number of packets per slot that are generated at nodes of $\mathcal{T}_i^{(j)}$ and select to be routed along $\mathcal{T}^{(j)}$. Under Rule A, the input load of $\mathcal{T}_i^{(j)}$ equals $\frac{\lambda}{d}2^{d-i}$, for $i = 1, \ldots, d$, because $\mathcal{T}_i^{(j)}$ contains 2^{d-i} nodes; see §2.1.3. Thus, there are large *discrepancies* in the input load received by the various subtrees. This also became apparent in the proof of Proposition 7.6; we showed there that the stability properties of our scheme are *limited* by the *bottlenecks* created in the arcs from which the largest subtrees are hanging.

The above discussion suggests the following potential improvement on the scheme: If it were possible for all subtrees $\mathcal{T}_1^{(j)}, \ldots, \mathcal{T}_d^{(j)}$ to receive the *same* (or approximately the same) input load (for $j = 1, \ldots, d$), then the throughput properties of the scheme would improve significantly (*). In fact, if *perfect balancing* of the input load were possible for all d trees, then the scheme would be stable for values of the load factor close to $\frac{d+1}{d+2}$; this would be attained by dedicating (for each tree $\mathcal{T}^{(j)}$) one slot out of every $d+2$ to transmissions towards root e_j, and the remaining $d+1$ slots to broadcasting one packet originating at each of the subtrees of $\mathcal{T}^{(j)}$ and at e_j. In the proposition to follow, we show that this is *not* possible.

Proposition 7.7: It is not possible to split the input traffic in such a way that each subtree receives $O(\lambda\frac{2^d}{d^2})$ input load. ■

Proof: The underlying idea of this proof is that there exist too many *small* subtrees.

Let $i^* = \lceil 2\log_2 d \rceil$. Subtrees $\mathcal{T}_{i^*}^{(j)}, \ldots, \mathcal{T}_d^{(j)}$ contain a total of $2^{d-i^*+1} - 1$ nodes, which is $\Theta(\frac{2^d}{d^2})$. (Recall that $\mathcal{T}_i^{(j)}$ contains 2^{d-i} nodes.) Therefore, the total input load of the subtrees $\mathcal{T}_{i^*}^{(j)}, \ldots, \mathcal{T}_d^{(j)}$ together with the root e_j is $O(\lambda\frac{2^d}{d^2})$. This conclusion holds for $j = 1, \ldots, d$. Thus, the $d(d - \lceil 2\log_2 d \rceil + 1)$ *smallest* subtrees of all d trees receive a total input load of $O(\lambda\frac{2^d}{d})$. During each slot, a total of $\lambda 2^d$ packets (on the average) are generated within the entire hypercube network. Thus, the total input load of the remaining $d(i^* - 1)$ largest subtrees $\left(\mathcal{T}_1^{(j)}, \ldots, \mathcal{T}_{i^*-1}^{(j)}\right)_{j=1,\ldots,d}$ is $\lambda 2^d - O(\lambda\frac{2^d}{d})$. Since $i^* = \lceil 2\log_2 d \rceil$, it follows that some of these subtrees will receive an input load of $\Omega(\lambda\frac{2^d}{d\log_2 d})$. **Q.E.D.**

(*) In light of Proposition 7.6, this could occur only after modifying Rule A in such a way that each node y is *not* confined to evenly split the packets it generates among the d trees $\mathcal{T}^{(1)}, \ldots, \mathcal{T}^{(d)}$.

The above result is rather discouraging. Note, however, that Proposition 7.7 implies that *nearly perfect* load balancing is not possible. On the other hand, it is conceivable that some improvement on the throughput properties of the scheme could be attained by a different kind of balancing. In particular, suppose that it is possible to partition the various subtrees of each tree (including the root) in L *classes* and balance the total input load per class. Then, by "serving" these classes in a round-robin mode, the scheme would be stable for values of the load factor close to $\frac{L}{L+1}$. Note that perfect balancing corresponds to $L = d+1$, but this has already been proved to be impossible. Our indirect scheme balances the load between *two* classes. One class consists of the largest subtree $\mathcal{T}_1^{(j)}$ and the other consists of the remaining subtrees together with the root e_j. By Rule E, the two classes are served in a round-robin mode. It is an *open* problem whether or not load balancing with more than two classes is feasible. A negative result to be proved below suggests that this open problem is rather hard.

As already argued, load balancing with more than two classes can only be attained if Rule A is modified; the most general form of a rule for splitting packets among the d trees is as follows:

Rule A′: Each node y splits its packets as dictated by Rule A, except for the fact that tree $\mathcal{T}^{(j)}$ is assigned an a priori probability $q_j(y)$, with $\sum_{j=1}^{d} q_j(y) = 1$.

Let us now consider a class of splitting rules to be referred to as *localized.* Under such a rule, all packets generated at nodes within the same subtree $\mathcal{T}_i^{(j)}$ assign the *same* a priori probability to tree $\mathcal{T}^{(j)}$; hence the terminology localized. To clarify this definition, let $i_j(y)$ denote the index of the subtree of $\mathcal{T}^{(j)}$ where node y belongs; that is, there holds $i_j(y) = i$ if $y \in \mathcal{T}_i^{(j)}$; a splitting rule such as Rule A′ is called localized if $q_j(y)$ depends *only* on $i_j(y)$, for $j = 1, \ldots, d$ and for all nodes y. Such a rule would be simpler to implement than one where some of the $q_j(y)$'s depend on the entire vector $\big(i_1(y), \ldots, i_d(y)\big)$. Unfortunately, it appears that Rule A is the *only* localized rule treating the underlying d trees *symmetrically.*

Proposition 7.8: Rule A is the only legitimate localized splitting rule that treats the d trees $\mathcal{T}^{(1)}, \ldots, \mathcal{T}^{(d)}$ symmetrically. ∎

Proof: Clearly, all nodes y belonging to the same subtree $\mathcal{T}_i^{(j)}$ satisfy $q_j(y) = r_i$, where r_i depends neither on y nor on j (because of symmetry among the d trees). Below, we show that $r_i = \frac{1}{d}$ for $i = 1, \ldots, d$, which implies that Rule A is the only

legitimate rule of the specified type.

Since the jth dimension is the last to be crossed in a path of $\mathcal{T}^{(j)}$ (see §2.1.4), node 0 is a leaf of this tree and constitutes the smallest subtree $\mathcal{T}_d^{(j)}$; this property applies for $j = 1, \ldots, d$. Thus, using the fact $\sum_{j=1}^{d} q_j(0) = 1$, we obtain $r_d = \frac{1}{d}$. A similar argument shows that node $(1, \ldots, 1)$ belongs to $\mathcal{T}_1^{(j)}$ for $j = 1, \ldots, d$; this immediately proves that $r_1 = \frac{1}{d}$.

There remains to show that $r_i = \frac{1}{d}$ for $i = 2, \ldots, d-1$; the proof will be done by induction on i. We assume that $r_1 = \cdots = r_{i-1} = \frac{1}{d}$ for some $i \in \{2, \ldots, d-1\}$; based on this, we show that $r_i = \frac{1}{d}$. (The fact $r_1 = \frac{1}{d}$ has already been proved.) We denote by $\hat{z}$ the node with the following binary identity: $\hat{z}_m = 0$ for $m = 1, \ldots, i-1$, and $\hat{z}_m = 1$ for $m = i, \ldots, d$. Recall now that the order of crossing the hypercube dimensions in the paths of tree $\mathcal{T}^{(j)}$ is as follows:

$$(j \bmod d) + 1, [(j+1) \bmod d] + 1, \ldots, [(j+d-1) \bmod d] + 1 .$$

This implies that $\hat{z} \in \mathcal{T}_{i-j}^{(j)}$ for $j = 1, \ldots, i-2$, while $\hat{z} \in \mathcal{T}_1^{(j)}$ for $j = i-1, \ldots, d-1$, and $\hat{z} \in \mathcal{T}_i^{(d)}$. Using also the fact $\sum_{j=1}^{d} q_j(\hat{z}) = 1$, we obtain

$$\sum_{j=1}^{i-2} r_{i-j} + \sum_{j=i-1}^{d-1} r_1 + r_i = 1 ,$$

or equivalently

$$\sum_{j=2}^{i-1} r_j + (d-i+1) r_1 + r_i = 1 ;$$

this together with the induction hypothesis $r_1 = \cdots = r_{i-1} = \frac{1}{d}$ implies that $r_i = \frac{1}{d}$. The proof of the result is now complete. **Q.E.D.**

Proposition 7.8 implies that Rule A is the only symmetric way of splitting the input traffic so that all packets generated at nodes within the same subtree $\mathcal{T}_i^{(j)}$ assign the *same* a priori probability to tree $\mathcal{T}^{(j)}$. Therefore, load balancing with more than two classes would definitely require that each of the parameters $q_j(y)$ depends on the entire vector $\big(i_1(y), \ldots, i_d(y)\big)$. Based on this, we believe that the problem of further load balancing is rather hard, if at all solvable. Moreover, because of the inherent imbalance of the d disjoint spanning trees, we only expect a minimal improvement (if any) on the stability region.

The above discussion motivates us to consider the problem of imbedding d *balanced* disjoint spanning trees in the d-cube; recall that a spanning tree is characterized as balanced if it has d subtrees of approximately the same size. If such an imbedding is possible, then Rule A would remain as in §7.1.2; the input load received by the various subtrees would *automatically* be balanced. Thus, stability would be maintained for all $\rho < 1 - o(1)$. The aforementioned imbedding problem is *open*.

7.3.3 Deadlock Prevention

So far in our analysis, we have assumed that each node has infinite buffer capacity. Of course, in practical applications all nodes have *finite* buffer capacity, and buffer *overflow* may often occur.

There are basically two approaches for dealing with buffer overflow at a node x:

(a) Packets waiting to cross an arc incoming to x are *blocked*, until there is empty space available in the buffer of x.

(b) Packets continue to arrive at x, and they are *dropped* if there is no empty space in the buffer of x.

Under the second approach, it is possible that packets are *partly* broadcast; that is, it may occur that a packet is not received by all nodes. On the other hand, the first approach runs the risk of a *deadlock*. Thus, it is rather important that our indirect scheme is *deadlock-free*. This is due to the decoupling of packets routed along different trees and to the fact that all simultaneous transmissions along the same tree are pointing at the *same* "direction". (That is, all packets are heading either towards or away from the corresponding root.) Hence, there never arises a group of packets blocking one another in a "cyclic" pattern. Therefore, even a buffer capacity of two units per node and per *tree* $\mathcal{T}^{(j)}$ is sufficient to guarantee that no deadlock will ever occur; one unit of buffer capacity is dedicated to packets heading towards root e_j, while the other unit is dedicated to packets already undergoing broadcast along $\mathcal{T}^{(j)}$.

It follows from the above discussion that when implementing the indirect scheme with buffer capacity $\Theta(d)$ per node, all packets *admitted* in the network are guaranteed to be broadcast in finite time. In fact, the same statement applies even in the presence of other packet transmissions (not necessarily broadcasts), provided that each of these packets also is routed along one of the d disjoint trees and conforms to Rules B and C.

7.4 COMPARISON OF THE VARIOUS ROUTING SCHEMES

In this section, we briefly compare the indirect scheme analyzed in §§7.1-7.3 to the efficient direct scheme of §§6.3-6.5 (namely, the non-idling scheme using d completely unbalanced spanning trees per node). Both of them exhibit satisfactory stability properties; of course, the direct scheme is preferable with this respect, because it attains the optimal stability region. Henceforth, we focus on the delay properties of the schemes and on the sizes of the queues involved.

As argued in §6.5, the direct scheme appears to satisfy $T \approx d + \frac{1}{2} + \frac{d}{3}\rho$ for small ρ. On the other hand, the indirect scheme satisfies $T \approx 3d + 1 + \frac{9}{4}\rho$ for small ρ; thus, it is outperformed by the direct scheme with respect to delay under light traffic. This is due to the fact that the *zero-order* term in T is larger under the indirect scheme. Notice however that the indirect scheme analyzed in this chapter exhibits *idling*, due to the periodic *alternation* of the directions of the arcs; see Rule C of §7.1.2. Avoidance of this idling phenomenon would improve the delay, especially in light traffic. In particular, after eliminating Rule C from our indirect scheme, there holds $\lim_{\rho \to 0} T = \frac{3d}{2}$; in order not to break continuity at this point, we prove this result in Appendix 7.B.

We have performed simulation runs for the non-idling version of our indirect scheme; in fact, the simulated scheme did not involve any *redundant* transmissions (see the beginning of §7.3.1). Not surprisingly, it appeared that, for fixed ρ, the *first-order* term in the delay T for this scheme *increases* with d; see Figure 7.5. This is due to the fact that packets routed along different trees interfere in the non-idling version of the scheme; in fact, each packet may now interfere with packets routed along *all* of the other trees. Despite the decrease in the delay T attained by eliminating idling, the direct scheme is still preferable; see Figure 7.6.

Notice that both direct and indirect schemes (including the non-idling one) belong to the *oblivious* class defined in §6.2.2. Thus, their delay properties are limited by the lower bound of Proposition 6.2, namely $T = \Omega(d + \frac{\rho}{1-\rho})$. It is an interesting open question to devise an oblivious scheme for which $T = \Theta(d + \frac{\rho}{1-\rho})$. Looking at Proposition 7.4, it is seen that the indirect scheme would satisfy this property if it were stable for all $\rho < 1$, instead of $\rho < \frac{2}{3}(1 - \frac{1}{2^d})$. Thus, if the imbedding of d balanced disjoint trees is feasible (see the end of §7.3.2), then the corresponding delay may come close to meeting the lower bound for oblivious schemes.

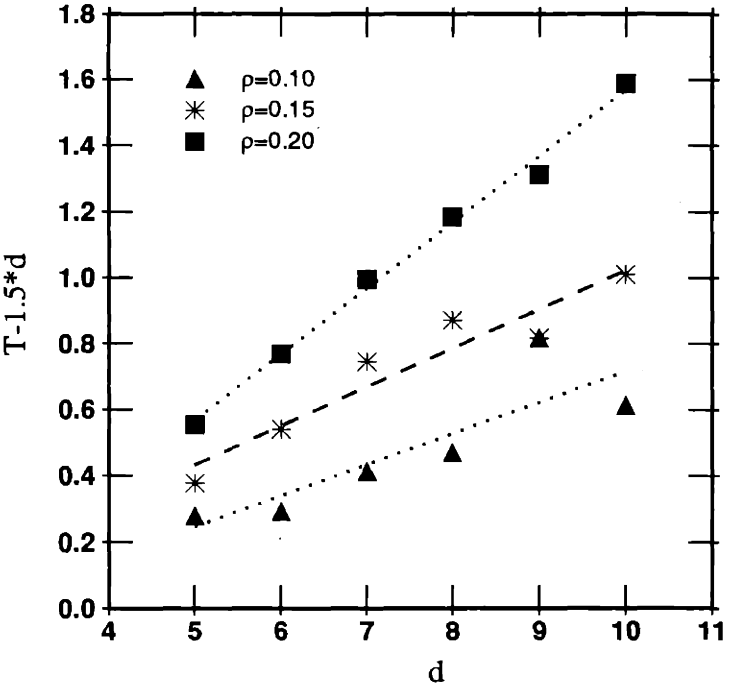

Figure 7.5: The first-order term in the delay induced by the non-idling version of the indirect routing scheme.

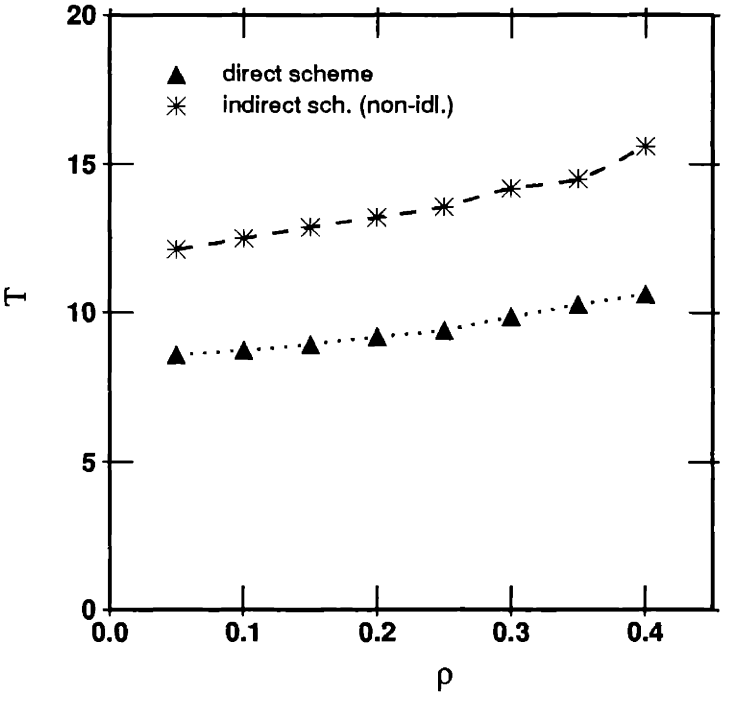

Figure 7.6: Comparing the delay induced by the two schemes, for $d = 8$.

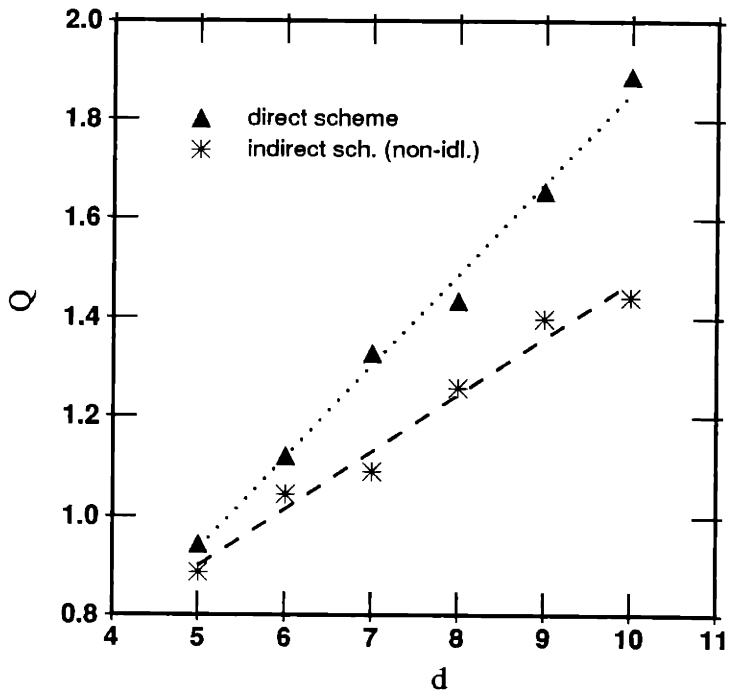

Figure 7.7: Comparing the average queue-size Q per node under the two schemes, for $\rho = 0.30$.

Next, we compare the values of the average queue-size Q for the various schemes; the points for the indirect scheme of §7.1.2 were computed by using Proposition 7.5, while the ones for the other two schemes were obtained experimentally. (Notice the peculiar behavior of Q for the indirect scheme of §7.1.2; this is due to the non-dominant term in the expression of Proposition 7.5, which is non-negligible for small values of d.) As revealed by Figure 7.7, the non-idling version of the indirect scheme is the most efficient one with respect to queue-sizes. It is worth noting that, for fixed ρ, the queue-size Q grows more slowly with d under the non-idling indirect scheme. In order to make the comparison even more clear, we have also plotted the *maximum queue-sizes* M observed in the various simulation runs corresponding to Figure 7.7; each simulation lasted for 1,000 slots. Again, the non-idling indirect scheme is superior to the direct one; see Figure 7.8.

Finally, we discuss the issue of deadlock prevention, when implementing the routing schemes with finite buffers. As proved in §7.3.3, the indirect scheme (in its original version) is deadlock-free when implemented with $\Theta(d)$ buffer capacity per node. Regarding the non-idling indirect scheme, again deadlocks can be prevented by dedicating

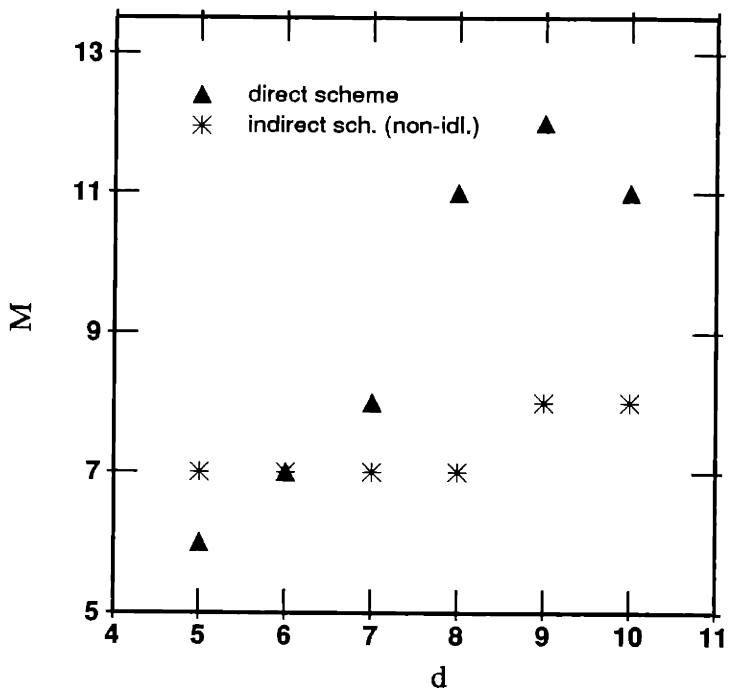

Figure 7.8: Comparing the maximum queue-size M (over all nodes) under the two schemes, for $\rho = 0.30$.

(at each individual node) constant buffer capacity to packets routed along each tree $\mathcal{T}^{(j)}$; packets heading towards root e_j should have access to *different* buffers from those used by packets traveling away from e_j. Despite the contention among packets routed along different trees, there never arises a group of packets blocking the *buffer* of one another in a "cyclic" pattern. As for the direct scheme, deadlocks can be prevented by using one of the standard techniques, such as that of "structured buffer pool" introduced by Raubold and Haenle [RaH76]. According to this well-known technique, the buffer of each node of the d-cube should be partitioned in several segments, with the ith segment being accessible *only* to packets already having traversed $i-1$ arcs. Therefore, a buffer capacity of $\Theta(d)$ per node is required to prevent deadlocks, both for the direct and the non-idling indirect schemes. Thus, in principle, *all* of our schemes can be made deadlock-free, even in the presence of other packet transmissions. Since deadlock prevention techniques often result in degradation of the throughput, it is not clear which technique is the most appropriate for each routing scheme. However, it is expected that, under deadlock prevention, both the original and the non-idling indirect schemes perform considerably better than the direct one, because they are "inherently" deadlock-free. Further investigation of these issues exceeds the scope of

our research.

The conclusion drawn from the previous discussion is that the direct scheme is preferable under light traffic (because it induces smaller delays) and under very heavy traffic (because it maintains stability, unlike the indirect schemes). On the other hand, the non-idling indirect scheme may be preferable under moderate traffic, because it involves smaller queues. Finally, under moderate traffic, deadlock prevention is more straightforward for the two indirect schemes.

APPENDIX 7.A

In this appendix, we prove Lemma 7.2 of §7.2.1, which refers to the tree $\tilde{\mathcal{T}}$ depicted in Figure 7.4; recall that each of the leaves of $\tilde{\mathcal{T}}$ is fed by a Poisson process with rate λ; also, transmissions may only start at the beginning of slots numbered $0, \Delta, \ldots$ and each of them lasts for Δ slots. The result to be established is as follows:

Lemma 7.2: The tree $\tilde{\mathcal{T}}$ of paths is a stable queueing system if and only if $\lambda n \Delta < 1$. Moreover, in the stable case, the steady-state average delay $\tilde{D}$ per packet is given as follows:

$$\tilde{D} = \Delta\left[\frac{3}{2} + \frac{\lambda n \Delta}{2(1-\lambda n \Delta)}\right] + \frac{\Delta}{n}\sum_{i=1}^{n} l_i - \Delta\,. \qquad \blacksquare$$

Proof: First, notice that the average total number of packets generated per interval of Δ slots equals $\lambda n \Delta$; since only one packet may depart from the tree during such an interval, it follows that the condition $\lambda n \Delta < 1$ is *necessary* for stability.

We consider the tree $\mathcal{T}$ obtained from $\tilde{\mathcal{T}}$ by adding a tandem of $l^* - l_i$ incoming arcs to each leaf s_i, where $l^* \stackrel{\text{def}}{=} \max\{l_1, \ldots, l_n\}$; thus, all paths of the new tree $\mathcal{T}$ have the *same* length l^*. For the tree $\tilde{\mathcal{T}}$ of Figure 7.4, we have $l^* = 4$, and the corresponding tree $\mathcal{T}$ coincides with the one of Figure 7.2a. The new tree $\mathcal{T}$ is assumed to operate as $\tilde{\mathcal{T}}$; that is, again transmissions may only start at the beginning of slots numbered $0, \Delta, \ldots$ and each of them lasts for Δ slots.

Furthermore, let us *couple* the arrivals in the two trees $\mathcal{T}$ and $\tilde{\mathcal{T}}$. A straightforward inductive argument shows that the departure process from $\mathcal{T}$ is a *delayed* version of that from $\tilde{\mathcal{T}}$; that is, the jth departure time in $\mathcal{T}$ is greater than (or equal to) the jth departure time in $\tilde{\mathcal{T}}$, for $j = 1, \ldots$ (Notice that, for a single path such as that of Figure 7.2b, the departure process is delayed when the path is augmented.) Therefore, on

a sample-path basis, tree $\mathcal{T}$ contains *at least* as many packets as tree $\tilde{\mathcal{T}}$; this implies that if $\mathcal{T}$ is stable, then $\tilde{\mathcal{T}}$ is stable as well.

Notice now that we may apply Lemma 7.1 and collapse the paths of tree $\mathcal{T}$, because all of them have the same length. (Recall the comments on the validity of Lemma 7.1 in more general cases, following its proof in §7.2.1.) Thus, regarding its departure process, tree $\mathcal{T}$ is equivalent to a tandem P of l^* identical deterministic servers (each with service time Δ) fed by a Poisson process with rate λn; in all servers of this tandem, service may only start at the beginning of slots numbered $0, \Delta, \ldots$ Clearly, *no queueing* takes place in the servers of tandem P *except* for the first one. Thus, P constitutes a stable queueing system if and only if its first server does so, namely if and only if $\lambda n \Delta < 1$. Furthermore, it is an immediate consequence of Lemma 7.1 that the *total number* of packets contained in tree $\mathcal{T}$ at any time t is the same as that in the context of tandem P. Hence, the stability condition for $\mathcal{T}$ coincides with that for P; thus, $\mathcal{T}$ is stable if and only if $\lambda n \Delta < 1$. Using the conclusion of the previous paragraph, it follows that the original tree $\tilde{\mathcal{T}}$ is also stable if $\lambda n \Delta < 1$; this is the same as the necessary stability condition of $\tilde{\mathcal{T}}$, which was derived in the beginning of the proof. Henceforth, we assume that stability applies.

Next, consider a single path, such as the one presented in Figure 7.2b; assuming that the arrival process feeding this path is Poisson, it is obvious that the *steady-state* statistics of the corresponding *departure* process *do not depend on the length* of the path (provided that the path has non-zero length). Using this property, and a straightforward inductive argument, the following result may be proved: At each *non-leaf* node where streams of packets *merge*, the processes feeding the node have the *same* steady-state statistics in both trees $\tilde{\mathcal{T}}$ and $\mathcal{T}$. Consider now an arc *shared* by packets originating from multiple leaves; our previous conclusion implies that this arc induces the *same average delay* per packet in both trees. Recall now how $\mathcal{T}$ was constructed from the original tree $\tilde{\mathcal{T}}$. Observing also Figures 7.4 and 7.2, it is apparent that $\tilde{\mathcal{T}}$ can be obtained from $\mathcal{T}$ by eliminating a tandem of $l^* - l_i$ *contention-free* arcs for each leaf s_i; the arcs eliminated should be among the ones traversed by the packets originating at s_i *prior to meeting* with packets generated elsewhere. We have already proved that each arc shared by packets originating from multiple leaves induces the same average delay in both trees. Hence, the average delay $\tilde{D}$ per packet in the context of $\tilde{\mathcal{T}}$ equals that induced in $\mathcal{T}$ (namely, D) *reduced* by the average time spent per packet in the

contention-free arcs that must eliminated from $\mathcal{T}$ (to yield $\tilde{\mathcal{T}}$). A packet originating at leaf s_i would spend $\Delta(l^* - l_i)$ time units in these arcs; since a typical packet is equally likely to originate at any of the leaves $s_1, \ldots, s_n$ of $\mathcal{T}$, it follows that

$$\tilde{D} = D - \frac{\Delta}{n} \sum_{i=1}^{n} (l^* - l_n) \,. \tag{7.A.1}$$

Notice now that, by the equivalence of tree $\mathcal{T}$ with tandem P, the two systems induce the *same* steady-state delay D per packet. The first server of P operates as a discrete-time $M/D/1$ queue with synchronization (see §2.2.2); thus, the average delay D_1 per packet induced by this server may be derived by applying (2.6) with λn instead of λ. It follows that

$$D_1 = \Delta \left[\frac{3}{2} + \frac{\lambda n \Delta}{2(1 - \lambda n \Delta)} \right] . \tag{7.A.2}$$

Since no queueing takes place in the rest of the servers of tandem P, we have

$$D = D_1 + \Delta(l^* - 1) \,;$$

this together with (7.A.2) implies that

$$D = \Delta \left[\frac{3}{2} + \frac{\lambda n \Delta}{2(1 - \lambda n \Delta)} \right] + \Delta(l^* - 1) \,.$$

Combining this with (7.A.1), we obtain the expression for $\tilde{D}$. **Q.E.D.**

APPENDIX 7.B

In this appendix, we prove the following result (mentioned in §7.4): After eliminating Rule C from our indirect scheme, there holds $\lim_{\rho \to 0} T = \frac{3d}{2}$.

Indeed, focusing on a particular tree $\mathcal{T}^{(j)}$, let $m(x)$ denote the maximum distance between node x and any other node, when only arcs of $\mathcal{T}^{(j)}$ and its reverse are considered. Under the non-idling version of the indirect scheme, packets routed along different trees *interfere*; nevertheless, it can be seen that

$$\lim_{\rho \to 0} T = \frac{1}{2^d} \sum_{x=0}^{2^d - 1} m(x) + \frac{1}{2} \,; \tag{7.B.1}$$

the right-hand quantity equals the average of the maximum *propagation* time per packet, with the term $\frac{1}{2}$ accounting for the average synchronization delay. For $x \in \mathcal{T}_1^{(j)}$, there holds $m(x) = H(x, e_1)+d-1$, because there is a node in the second largest subtree $\mathcal{T}_2^{(j)}$ at distance $d-1$ from root e_j; this follows from the construction of a completely unbalanced spanning tree. Similarly, for $x \notin \mathcal{T}_1^{(j)}$, there holds $m(x) = H(x, e_1) + d$, because the largest subtree $\mathcal{T}_1^{(j)}$ has depth d. Since $\mathcal{T}_1^{(j)}$ has 2^{d-1} nodes, we have

$$\sum_{x=0}^{2^d-1} m(x) = (d-1)2^{d-1} + d2^{d-1} + \sum_{x=0}^{2^d-1} H(x, e_1) = (3d-1)2^{d-1}\,, \qquad (7.B.2)$$

where we have used the fact $\sum_{x=0}^{2^d-1} H(x, e_1) = \sum_{k=0}^{d} k\binom{d}{k} = d2^{d-1}$. Combining (7.B.2) with (7.B.1), it follows immediately that $\lim_{\rho \to 0} T = \frac{3d}{2}$. **Q.E.D.**

8. Conclusion

In this section, we present a summary of this Ph.D. Thesis, and we discuss some directions for further research.

8.1 SUMMARY OF THIS RESEARCH

Our research is focused on routing problems for message-passing parallel computers. We analyzed several problems of communications among the various processors. Such communications arise during the distributed solution of large numerical problems, and they are performed through an underlying interconnection network. Most of our analysis is related to the popular binary hypercube network; we also investigated more general issues, and developed analytical tools that are applicable to other networks too.

The routing problems of interest may be classified in two categories:

(a) Static problems, where all packets to be transmitted are available at the same time, and the communication task is to be performed only once, and in the absence of other packet transmissions.

(b) Dynamic problems, where packets are generated at random times, over an infinite time-horizon; thus, several communication tasks may interfere with each other.

Static problems pertain to synchronous computation; in devising an algorithm for such a problem, the primary objective is to attain fast completion of all transmissions involved. On the contrary, dynamic problems pertain to certain models of asynchronous computation where it is assumed that processors are not synchronized periodically. Regarding dynamic problems, the objective is to devise routing schemes that make efficient use of the communication resources available (thus attaining high throughput), while not introducing large queueing delays; these two performance criteria are often in opposition with each other, and an appropriate compromise has to be made.

In our analysis, we considered routing problems from both categories. First, we analyzed two static problems: The total exchange task in the hypercube, for which we derived an unimprovable algorithm (with respect to its completion time); and the task of simultaneous broadcasts by a subset of hypercube nodes, for which we derived an efficient algorithm applying to all possible subsets of broadcasting nodes. However, the main focus of our research was on dynamic routing problems. In particular, we analyzed the problem of multiple node-to-node communications in the hypercube; we proved that a simple greedy routing scheme has very good performance, thus resolving an important open question of the routing literature. This scheme was analyzed by means of a new approach, treating the hypercube as a queueing network with deterministic servers; the various performance measures are expressed by simple formulae, involving only the basic parameters of the problem. An interim proposition of the analysis enabled the extension of the results to the butterfly and to other crossbar networks. Of similar spirit was the approach used in the analysis of our second dynamic routing problem, namely that of multiple broadcasts in the hypercube; for this we developed several efficient routing schemes. Even though this problem is rather intractable analytically, a considerable part of the analysis is exact; at certain points, however, we resorted to approximations and simulations. All of the aforementioned routing algorithms and schemes can be easily implemented in a distributed fashion. Also, all schemes considered for the dynamic problems are of the on-line type, meaning that all routing decisions are made on the basis of past information only.

The field of routing in interconnection networks is rather extensive. Even though

considerable work has already been done, there still remain several interesting problems to be investigated; in the next subsection, we discuss some directions for further research.

8.2 DIRECTIONS FOR FURTHER RESEARCH

Static routing problems have received extensive attention in the routing literature. Numerous algorithms have been devised for a variety of communication tasks, under several different models. Somewhat surprisingly, slightly different problems may have entirely different solutions (even in the context of the same network), especially if unimprovable algorithms are sought. Thus, there arises a need for a unified approach to devising fast algorithms for static problems. Such approaches have already been presented in the literature; however, none of them deals with the most general class of tasks where each message is broadcast to a subset of processors. We believe that analysis of such tasks (under a unified framework) constitutes an interesting direction for further research.

Regarding the dynamic routing problems analyzed, certain related questions are still open. Thus, for the problem of multiple node-to-node communications in the hypercube, an important question is to investigate the performance of greedy schemes other than the one we analyzed. Also, our greedy routing scheme has yet to be analyzed under the assumption of finite buffer capacity. Finally, the analysis of adaptive schemes such as deflection routing or adaptive shortest-path routing seems to constitute a rather interesting as well as challenging class of open problems. As for the problem of dynamic broadcasts in the hypercube, our approach fell short of investigating the performance of a certain routing scheme that was observed to be rather efficient; further exact analysis of this scheme is an interesting (yet hard) open question.

The literature on dynamic routing problems is somewhat limited, relatively to their importance. The two dynamic problems analyzed in this Ph.D. Thesis constitute special cases of communication problems arising in general purpose computation. In this general context, it is often the case that packets received by a node influence the generation of subsequent packets, as well as their lengths or their destinations. In our opinion, analyzing such general problems seems to be a very interesting and challenging direction for further research.

References

[AbP86] S. Abraham and K. Padmanabhan, "Performance of the Direct Binary n-Cube Network for Multiprocessors", *Proceedings of the 1986 International Conference on Parallel Processing.*

[Ale82] R. Aleliunas, "Randomized Parallel Communication", *Proceedings of the 1st ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pp. 60-72.

[BeG87] D. Bertsekas and R. Gallager, *Data Networks*, Prentice-Hall.

[BeT89] D.P. Bertsekas and J.N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*, Prentice-Hall.

[BGST87] C. Bouras, J. Garofalkis, P. Spirakis, and V. Triantafillou, "Queueing Delays in Buffered Multistage Interconnection Networks", Dept. of Computer Science, Technical Report 289, New York University.

[BoH82] A. Borodin and J.E. Hopcroft, "Routing, Merging and Sorting on Parallel Models of Computation", *Proceedings of the 14th Annual ACM Symposium on Theory of Computing*, pp. 338-344.

[Bor87] A.A. Borovkov, "Limit Theorems for Queueing Networks, Part I", *Theory Probab. Appl.*, vol. 31, pp. 413-427.

[BOSTT91] D.P. Bertsekas, C. Ozveren, G.D. Stamoulis, P. Tseng, and J.N. Tsitsiklis, "Optimal Communication Algorithms for Hypercubes", *J. Parallel Distrib. Comput.*, vol. 11, pp. 263-275.

[Bru71] S.L. Brumelle, "Some Inequalities for Parallel-Server Queues", *Operations Research*, vol. 19, pp. 402-413.

[ChS86] Y. Chang and J. Simon, "Continuous Routing and Batch Routing on the Hypercube", *Proceedings of the 5th ACM Symposium on Principles of Distributed Computing*, pp. 272-281.

[DaS87] W.J. Dally and C.L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks", *IEEE Trans. Comput.*, vol. C-36, pp. 547-553.

[Ede91] A. Edelman, "Optimal Matrix Transposition and Bit Reversal on Hypercubes: All-to-All Personalized Communication", *J. Parallel Distrib. Comput.*, vol. 11, pp. 328-331.

[Gal90] R.G. Gallager, "Discrete Stochastic Processes", Course Notes, Dept. of Electrical Engineering and Computer Science, M.I.T.

[GrG86] A.G. Greenberg and J. Goodman, "Sharp Approximate Models of Adaptive Routing in Mesh Networks", *preliminary report.*

[GrH89] A.G. Greenberg and B. Hajek, "Deflection Routing in Hypercube Networks", *preprint.*

[HaC87] B. Hajek and R.L. Cruz, "Delay and Routing in Interconnection Networks", In A.R. Odoni, L. Bianco, and G. Szago (Eds.), *Flow Control of Congested Networks*, Springer-Verlag.

[Hwa87] K. Hwang, "Advanced Parallel Processing with Supercomputer Architectures", *Proc. IEEE*, vol. 75, pp. 1348-1378.

[JoH89] S.L. Johnsson and C.-T. Ho, "Optimum Broadcasting and Personalized Communication in Hypercubes", *IEEE Trans. Comput.*, vol. C-38, pp. 1249-1267.

[Kle75] L. Kleinrock, *Queueing Systems, Vol. I: Theory*, John Wiley.

[KoK77] H. Kobayashi and A.G. Konheim, "Queueing Models for Computer Communications Systems Analysis", *IEEE Trans. Commun.*, vol. COM-25, pp. 2-29.

[LaF80] R.E. Ladner and M.I. Fischer, "Parallel Prefix Computation", *J. ACM*, vol. 27, pp. 832-838.

[Lei90] F.T. Leighton, "Average Case of Greedy Routing Algorithms on Arrays", *preprint.*

[LeL90] T. Leighton and C.E. Leiserson, "Theory of Parallel and VLSI Computation", Laboratory for Computer Science, Report LCS/RSS 6, M.I.T.

[LeR88] T. Leighton and S. Rao, "An Approximate Max-Flow Min-Cut Theorem for Uniform Multicommodity Flow Problems with Applications to Approximation Algorithms", *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science*, pp. 422-431.

[LMR88] T. Leighton, B. Maggs, and S. Rao, "Universal Packet Routing Algorithms", *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science*, pp. 256-269.

[MiC87] D. Mitra and R.A. Cieslak, "Randomized Parallel Communications on an Extension of the Omega Network", *J. ACM*, vol. 34, pp. 802-824.

[Min89] S.E. Minzer, "Broadband ISDN and Asynchronous Transfer Mode", *IEEE Communications Magazine*, vol. 27, pp. 17-57.

[Pip84] N. Pippenger, "Parallel Communication with Limited Buffers", *Proceedings of the 25th Annual IEEE Symposium on Foundations of Computer Science*, pp. 127-136.

[RaH76] E. Raubold and J. Haenle, "A Method of Deadlock-Free Resource Allocation and Flow Control in Packet Networks", *Proceedings of the 3rd International Conference on Computer Communications*, pp. 483-487.

[Ran87] A. Ranade, "How to Emulate Shared Memory", *Proceedings of the 28th Annual IEEE Symposium on Foundations of Computer Science*, pp. 185-194.

[Ros83] S.M. Ross, *Stochastic Processes*, John Wiley.

[SaS85] Y. Saad and M.H. Schultz, "Data Communication in Hypercubes", Dept. of Computer Sciences, Research Report YALEU/DCS/RR-428, Yale University.

[StT91] G.D. Stamoulis and J.N. Tsitsiklis, "Optimal Distributed Policies for Choosing Among Multiple Servers", Report LIDS-P-2021, Laboratory for Information and Decision Systems, M.I.T.

[Upf84] E. Upfal, "Efficient Schemes for Parallel Communication", *J. ACM*, vol. 31, pp. 507-517.

[VaB81] L.G. Valiant and G.J. Brebner, "Universal Schemes for Parallel Communication", *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, pp. 263-277.

[VaB91] E.A. Varvarigos and D.P. Bertsekas, "New Optimal Broadcasting Algorithms for Hypercube Networks", *preprint.*

[Val82] L.G. Valiant, "A Scheme for Fast Parallel Communication", *SIAM J. Comput.*, vol. 11, pp. 350-361.

[Val89] L.G. Valiant, "General Purpose Parallel Architectures", Report TR-07-1989, Aiken Computation Laboratory, Harvard University.

[Var90] E.A. Varvarigos, "Optimal Communication Algorithms for Multiprocessor Computers", Report CICS-TH-192, Center for Intelligent Control Systems, M.I.T.

[Wal88] J. Walrand, *An Introduction to Queueing Networks*, Prentice-Hall.