Bounds for Communication Problems in Interconnection Networks under a Linear Cost Model

by

David B. Peters

B.Sc. University of Victoria 1982

M.Math. University of Waterloo 1985

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
in the School
of
Computing Science

© David B. Peters 1995
SIMON FRASER UNIVERSITY
December 1995

All rights reserved. This work may not be
reproduced in whole or in part, by photocopy
or other means, without the permission of the author.

APPROVAL

Name:	David B. Peters
Degree:	Doctor of Philosophy
Title of thesis:	Bounds for Communication Problems in Interconnection Networks under a Linear Cost Model
Examining Committee:	Dr. Slawomir Pilarski Chair
	Dr. Joseph Peters, Senior Supervisor
	Dr. Pavol Hell, Supervisor
	Dr. Thomas Shermer, Supervisor
	Dr. Arthur Liestman, Examiner
	Dr. Ching-Tien Ho, External Examiner
Date Approved:	December 5, 1995

SIMON FRASER UNIVERSITY

PARTIAL COPYRIGHT LICENSE

I hereby grant to Simon Fraser University the right to lend my thesis, project or extended essay (the title of which is shown below) to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users. I further agree that permission for multiple copying of this work for scholarly purposes may be granted by me or the Dean of Graduate Studies. It is understood that copying or publication of this work for financial gain shall not be allowed without my written permission.

Title of Thesis/Project/Extended Essay

Bounds for Communication Problems in Interconnection Networks under a Linear Cost Model.

Author: [illegible]

(signature)

David Bruce Peters

(name)

December 5, 1995

(date)

Abstract

In the analysis of communications under the assumption of linear cost and in the context of a distributed memory interconnection network, many authors have noted that a good method for sending a message of length n along a path of length m is to divide the message into some collection of equal or nearly equal packets, and pipeline the communication. We formalize and prove that this notion is optimal in the context of an interconnection network using store and forward communications under the linear cost model. Armed with this proof technique and bounds, we attack problems of broadcasting and gossiping in the contexts of a ring of processors and a complete interconnection of processors. We acquire a variety of new upper bounds, and several lower bounds to match.

Acknowledgments

An image that comes to mind is of a football receiver, after catching the first touchdown of his career, kneeling in the end zone to pray. While I'm not catching footballs, it's a good analogy for me: I'd like to thank God for enabling me to finish this work. I'm not claiming that God did the work, nor that I had nothing to do with the outcome. But that without Him, for whatever reasons, I wouldn't have got here: yes, exactly that.

I thank my wife Robyn for her encouragement, prayers and support.

My respect and gratitude to my supervisor, Joe Peters (no biological relation). For friendship; honesty; inspiration; patient encouragement well beyond the call of duty. Not to mention the only good coffee on campus.

To Tom Shermer and Art Liestman: for demonstrating again and again that approachability and frivolity can coexist happily with responsibility and competence. My thanks to you both for your support of my academic development.

My thanks too to Binay Bhattacharya, Arvind Gupta, and Pavol Hell, for teaching me all sorts of wonderful things that, unfortunately, don't quite make it into this work. Not consciously, anyway.

And to my friends, the algorithms gang — in alphabetical order — Cheryl, Dave, Patrice, Sheelagh. For camaraderie, discussions, food; more discussions; acres of whiteboard scrawlings; help with TEX, Postscript, system administration, Emacs, calculus, and university bureaucracy. And most importantly, for making life in the lab more than pleasant.

Contents

List of Figures

Chapter 1

Overview

1.1 Introduction

Despite huge advances in the processing power of today's computers over those of the recent past, the tasks these machines are called to address grow even more rapidly. Partly as a consequence, we see a proliferation of parallel machines being constructed: everything from two or four CPU desktop workstations through systolic arrays or hypercubes with thousands of processors. And where once the work of a theorist was clear—one found the fastest algorithm for a given problem, or proved that finding such a good algorithm was unlikely—there is now the added task of defining the machine model under which an analysis is valid.

One of the philosophies of construction for these machines is that of achieving parallelism by constructing a network of many essentially identical processors joined by point-to-point communication links. Typically, the processors run without any central control, and communicate by passing messages along the links. This *interconnection network* is the general setting of our work.

In this context, an algorithm that takes advantage of any parallelism will necessarily involve the transfer of information between processors. In general, this will involve sending messages along a path (or several paths) of communication links. The act of "sending a message along a path" will decompose into a number of discrete information transfers along single links: the transmission of *packets* of information.

We refer of a collection of packet transfers as a *sending scheme*: finding optimal (or near-optimal) sending schemes for a particular model and class of network will be the specific focus of this thesis.

In the same way that an entirely unstructured data space may be improved by the use of data structures, the communications of a specific algorithm could conceivably be entirely chaotic or pattern-less, yet it would seem more desirable for the data transmission to be composed of primitives. We will concentrate on providing good schemes for a couple of specific well-studied communication primitives: *broadcasting* and *gossiping*. Broadcasting is the primitive in which a processor sends copies of a single message to all other processors. Gossiping is the primitive in which each processor broadcasts a message to all others. Many other primitives exist — the reader is directed to [8] for a good survey.

We want good communications schemes. What criterion determines whether a scheme is good or not? We employ a *linear cost model*, in which the time to send a packet over a single communication link is a linear function of the length of the packet. A "good" scheme, then, is one that can be scheduled in minimum total time under the linear cost model. In this thesis, we obtain optimal schemes under that model for several fundamental problems, as well as good time bounds for several more ambitious problems.

In this choice of model, we follow the lead of Saad and Schultz [22], Johnsson and Ho [16], and Fraigniaud [7]. The *unit cost model*, rather than the linear cost model, was the traditional choice. On the other hand, the choice of store-and-forward communications is quite in keeping with most "traditional" (insofar as that word may be applied to distributed computing) treatments. While there is work that deals with other transmission methods (circuit-switched, wormhole, or cut-through routings; broadcast buses; etc.) store-and-forward communications are a lowest common denominator for many distributed machines. Since we want to categorize this particular work more as foundational than exploratory, we have confined ourselves (with the exception of the linear cost model) to the simplest or most ordinary choice in all else.

We begin by establishing an exact bound for sending a message over a single directed path, and then build on that foundation. We apply the result in two dissimilar ways: first, we update existing bounds that contain a sending scheme by employing a provably optimal sending scheme; and second, we use some of the proof techniques to improve bounds.

1.2 The structural model

An *interconnection network*, in this context, is a collection of asynchronous processors, connected one to another by communication links in some connected topology. Of the adjective *asynchronous* we will not make any great fuss. It is important only in the following limited sense: the lower bounds we will discover will not make any restrictions upon the synchronity of the processors; and the cost model will only reflect the time spent by the communication links, not the (potentially incomparable) execution times of the processors.

We will make no specific assumption about the autonomy of the processors. In fact, whether or not the processors of our networks have a centralized control makes little difference: all our algorithms may be implemented in a distributed setting.

What, then, do we mean by "processor", "communication link", or "topology"? We address each in turn.

An example of the sort of processor we're talking about is the Analog Devices 21060 SHARC DSP chip [1]. It's a 40MHz multi-port processor with a local memory of 128K words and 6 direct memory access (DMA) channels for connecting to neighbours. In general, we will denote this sort of entity by the term *processor*: a computation element which has some local memory for storing instructions and data, and a collection of associated DMA channels for communication. Processors will be connected by communication links, which will be managed in some manner by the DMA channels.

As to the communication links themselves, we will only concern ourselves with static point-to-point links: we will not discuss broadcast buses, shared memory, or even reconfigurable switches. We assume that these links are packet switched (or store-and-forward) so that a processor must wait for all bits of a particular packet to arrive before it may forward any portion of the packet. We assume processors are free to repackage information they receive in any way.

We will allow three variants of these links: a *simplex* (or *unidirectional*, or one way) link, and both *half duplex* and *full duplex* two way connections. A simplex link has an associated orientation: a simplex link L connecting P_i to P_j allows the transmission of packets from P_i to P_j but does not allow any transmission in the other direction. A half duplex link, on the other hand, allows transmissions in both directions, but only one direction may be in use at any one time. A full duplex link is essentially identical to a pair of simplex links that share the same endpoints and have opposite direction.

Regardless of whether the individual links are simplex or duplex, the interaction between processors and links also admits a few variants. We will follow [23] and label the variants *link bound*, *processor bound*, and *DMA bound*, as follows. In the link bound (or *all-port*) case, a given processor may use all of its links simultaneously. Alternatively, in the processor bound (or *1-port*) case, a single link might monopolize the attention of a processor to such a degree that no other links may be employed while the first is active. Or there could be some compromise between these two extremes — the DMA bound (or *k-port*) case: the 21060 for example, regardless of how many physical connections it has with neighbours, will not be able to employ more than its 6 DMA channels at once. We will not investigate the DMA bound case: the interested reader is directed to [7] or [8].

The *topologies* we discuss are from graph theory, rather than real analysis. We consider a network as a graph (or digraph in the case of unidirectional links), in which the processors are the nodes, and the communication links the edges. The possible topologies, then, are the categorizations of graphs: complete graphs, cycles, tori, hypercubes, and the like. We will tend to confine our attentions to the cycle and the complete graph, with only occasional forays afield into more exciting topologies.

The component parts of communication schemes will be *messages*, *sub-messages*, *packets*, and occasionally *rounds*. We define these terms as well.

By *message* we will mean a set of indivisible units of data. These *message units* are all of the same size, and form the basis for a measure of the size of the message: a message of size n contains n message units. Arbitrary subsets of a message are termed *sub-messages*, and share the same size measure. The majority of the communications algorithms we will investigate will partition the message into a collection of sub-messages.

By *packet*, we mean something with a bit more solidity: a specific transfer of a sub-message over a link of the network. So while we can divide or amalgamate sub-messages, the same cannot strictly be said of packets: "dividing a packet" can only be a reference to dividing the sub-message that is represented in the packet. We will typically assume that each packet has well defined endpoints: the time it begins transmission, and the time it finishes transmission.

A synchronous sending scheme may be divided into *rounds*. A round is a collection of packets that have two properties. First, no two packets in a particular round may be transmitted on the same link. Second, if two packets belong to different rounds, they cannot overlap in time: either one finishes transmission strictly before the other begins, or vice versa. A round is a convenience for analysis: the time required by an algorithm divided into rounds is simply the sum of the times required by the largest packet in each round.

1.3 The cost model

Traditional sequential analyses are concerned primarily with a measure of the total execution time that is taken by an algorithm, particularly as a function of the size of the input. While this would also be the measure of choice in our distributed setting, the essential asynchronity of the processors makes such a measure difficult: the difficulty in balancing the communication time of an algorithm against the time required for computation within a processing element pales against the simple fact that there is not necessarily a global clock with which to measure total time.

Perhaps as a result, there is no single most approved measure of the performance of a distributed algorithm. Most authors agree that the best measure is some function of the messages employed by an algorithm. Some are content to count either the total number of messages employed by a communication scheme, or the total number of rounds necessary to complete an algorithm. And since neither of these measures exactly reflects the elusive "total time" measure, others employ more complex functions, usually with some practical justification. We will follow the latter camp.

Traditionally, a more sophisticated analysis of a sequential algorithm measures use of both the execution time and the space employed by an algorithm. There is often a natural tradeoff, in practice, between these parameters: an algorithm constrained to occupy less memory may take more time than an unconstrained algorithm; while a minimum time algorithm could conceivably require more memory than is strictly necessary to perform the task.

Moving into a tightly coupled parallel setting, we find that in addition to time and space, one can also find a tradeoff between time and the number of processors allocated to a task. Typically, a parallel analysis will end up with a time bound parameterized by the number of processors.

The distributed setting adds more parameters. In addition to measuring communication time, it could be appropriate to also measure network load, or the proportion of the available DMA channels used. What's more, one could consider variant topologies on n processors merely various restrictions of complete connectivity, and obtain a three-way tradeoff in parameters that are not present in either the sequential or tightly coupled parallel models: time, network load, topology.

The simplest treatment of communication time gives unit cost to each transmission of a message. There are several variants. If the network load constraint is calibrated as "links simultaneously active", and set to 1, the *total messages* measure arises. And with no constraint on network load, we get upper and lower bounds expressed in terms of the number of *rounds* required to send a message. Though this unit cost model represents the traditional majority opinion in distributed analysis, we will not address it at all; rather the reader is directed to the recent general surveys by Hromkovič *et.al.* [15] and Fraigniaud and Lazard [8], or the comprehensive survey of Hedetniemi, Hedetniemi and Liestman [13] which catalogues the early results.

We will treat each transmission as taking time linear in the length of the message. Specifically, any message of length L will take time $\beta + L\tau$ to finish transmission, where β is the start-up cost for activating a link, and τ the propagation cost of sending a single message unit. As a simplifying assumption, we will assert that β and τ are the same for each communication link.

The linear cost model permits a couple of degenerate cases. The first — when τ is zero — is equivalent to the unit cost model. The second — when β is zero — is usually termed the *round model*: for a specific treatment, see [4].

In the larger three-way tradeoff, we will examine several network topologies. But the network load parameter is partly obscured by the choice of τ. Without further constraint, τ is the inverse of the bandwidth of the links, in bits per second. We can investigate the alteration of the proportion of total bandwidth an algorithm uses simply by adjusting τ. We will thus content ourselves with investigating the interrelation of communication mode and network topology with β and τ. problem.

1.4 Motivations and results

Why? Because it hadn't been done, of course. Any thesis says as much.

In this *particular* case, the spark was the "sending question":

> How long does it take to send a message of length n over m communication links under the linear cost model?

The context of this question is the general problem of finding lower bounds for communication problems in interconnection networks under that linear cost constraint. A one-to-one communication in the most restrictive possible network topology appeared to be the ideal starting point. Certainly, it has proved a necessary first step in the investigations of lower bounds for broadcasting and gossiping in this thesis.

Of course, the question "How long does it take ..." can only be completely answered by finding matching upper and lower bounds on the problem. We will refer to this case idiomatically within this work by using (for example) *matching lower bound* to mean implicitly that the lower bound matches an existing upper bound. We will thus answer the sending question in chapter 2 by finding matching lower bounds for both the link bound and processor bound cases. We address the broadcasting problem in chapter 3 for the simplest topologies: the ring, and the complete graph. We treat gossiping in the ring in chapter 4. For both gossiping and broadcasting, we do not always find matching bounds. We will also demonstrate some new algorithms that improve upon the existing upper bounds, and some new lower bounds.

We employ the symbol $b_X(T,n)$ to denote the optimal time for broadcasting a message of size n under communication model X and in topology T. Similarly, we denote the minimum time for each processor to gossip a message of size n in topology T under communication model X by $g_X(T,n)$. We follow the notation of Fraigniaud and Lazard [8] in specifying the subscripts X: the labels F and H respectively denote full duplex and half duplex links, while 1, k, or $*$ respectively denote the processor bound, DMA bound, and link bound cases. Rather than risk any ambiguity with regard to the orientation of simplex links, we will specify the simplex case by restricting the topology to a specific oriented network: the unidirectional cycle. In addition, we will employ $S_X(T,n,m)$ to denote the optimal time to send a message of size n to another processor at a distance of m communication links in topology T. For our "sending question" the topology is omitted for convenience: we employ $S_X(n,m)$.

1.4.1 Lower bounds

The literature contains few examples of tight lower bounds for communication problems under the general linear cost constraint. One reference (Fraigniaud, Miguet and Robert [9]) proves a matching lower bound for the problem of scattering in the unidirectional ring. And recently, Fraigniaud and Peters [10] have demonstrated a matching lower bound for gossiping with unit size messages under the processor bound constraint in the complete graph K_{2m}. Good lower bounds also exist for broadcasting with unit size messages, but these are simply equivalent to lower bounds for the unit cost model.

Most other approaches to lower bounds under the linear cost constraint follow Ho [14]. He notes that a transmission is constrained separately by: its root dominance — the time required for the source to send the whole message; its latency dominance— the propagation delay for the last unit of the message to reach the furthest destination; and its bandwidth dominance — the total bandwidth required divided by the available bandwidth. Yet these three characteristics have their own tradeoff. Decreasing the root dominance could easily increase the bandwidth dominance of a transmission.

For the problems we will consider, the best previously published lower bounds are the work of Johnsson and Ho [16] and Fraigniaud [7].

$$\begin{aligned}
S_*(n,m) &\geq m\beta + (n+m-1)\,\tau & * \\
S_1(n,m) &\geq m\beta + (n+m-1)\,\tau & * \\
b_{F*}(C_m,n) &\geq \left\lfloor \frac{m}{2} \right\rfloor \beta + \left(\left\lceil \frac{n}{2} \right\rceil + \left\lfloor \frac{m}{2} \right\rfloor - 1 \right) \tau & * \\
b_{F1}(C_m,n) &\geq \left\lfloor \frac{m}{2} \right\rfloor \beta + \left(n + \left\lfloor \frac{m}{2} \right\rfloor - 1 \right) \tau \\
b_{F*}(K_m,n) &\geq \beta + \left\lceil \frac{n}{m-1} \right\rceil \tau \\
b_{F1}(K_m,n) &\geq \beta + n\tau \\
g_{F*}(C_m,n) &\geq \max\left\{ \left\lfloor \frac{m}{2} \right\rfloor (\beta+\tau), \frac{1}{2}((m-1)n)\,\tau \right\} & * \\
g_{F1}(C_m,n) &\geq \max\left\{ \left\lfloor \frac{m}{2} \right\rfloor (\beta+\tau), (m-1)n\tau \right\} & *
\end{aligned}$$

The latency dominance constraint of Ho is the starting point for each of these bounds. In a regular graph with m nodes, diameter D and degree Δ, all required information must be received by the destination processor(s): $n\tau$ time for the processor bound broadcast; $n\tau/\Delta$ time for the link bound broadcast; $(m-1)n\tau/\Delta$ time for link bound gossiping; $(m-1)n\tau$ time for processor bound gossiping. In addition, there must be a path of length at least D over which some information must pass — requiring $D(\beta+\tau)$ time. Fraigniaud noted that the latency for broadcasting is additive. In the above enumeration, the bounds that contain a maximum are the original bounds due to Ho, while the remainder have been updated by Fraigniaud.

Fraigniaud and Lazard [8] provide half duplex bounds, using Ho's bandwidth dominance constraint. They note that the total available bandwidth in the link bound half duplex model is $m\Delta/(2\tau)$, and $\lfloor m/2 \rfloor /\tau$ in the processor bound half duplex model. The next collection of bounds follow from the requirements that at least $(m-1)n$ units of data be received for a successful broadcast, and that gossiping exchanges messages with total size $m(m-1)n$.

$$b_{H*}(C_m,n) \geq \max\left\{ \left\lfloor \frac{m}{2} \right\rfloor (\beta+\tau), \frac{(m-1)n}{m}\tau \right\}$$

$$\begin{aligned}
b_{H1}(C_{2m},n) &\geq \max\left\{m(\beta+\tau), \frac{2(m-1)}{m}n\tau\right\} \\
b_{H1}(C_{2m+1},n) &\geq \max\{m(\beta+\tau), 2n\tau\} \\
b_{H*}(K_m,n) &\geq \max\left\{\left\lfloor\frac{m}{2}\right\rfloor(\beta+\tau), \frac{2n}{m}\tau\right\} \\
b_{H1}(K_{2m},n) &\geq \max\left\{m(\beta+\tau), \frac{2(m-1)}{m}n\tau\right\} \\
b_{H1}(K_{2m+1},n) &\geq \max\{m(\beta+\tau), 2n\tau\} \\
g_{H*}(C_m,n) &\geq \max\left\{\left\lfloor\frac{m}{2}\right\rfloor(\beta+\tau), 2(m-1)\lceil n/2\rceil\tau\right\} \\
g_{H1}(C_{2m},n) &\geq \max\{m(\beta+\tau), 2(m-1)n\tau\} \\
g_{H1}(C_{2m+1},n) &\geq \max\{m(\beta+\tau), 2mn\tau\}
\end{aligned}$$

We will see improvements in many of the full duplex bounds: those marked with a $*$. On the other hand, the half-duplex bounds will not be improved upon in this work.

1.4.2 Upper bounds

To the first approximation, the application of two concepts provide the upper bounds for our collection of problems: *pipelining* and *disjoint spanning trees*.

Informally, by *pipelining*, we mean the sort of scheme that occurs in a modern assembly-line automobile plant. Rather than a car being assembled in its entirety before the next is begun, the vehicles are placed in a line composed of (more or less) discrete stages, each of which take (roughly) the same amount of time. While a windshield is being put on one car, the drive train is installed in another, and the wheels bolted to yet another. In our specific sense, a pipeline is the result of making the analogy from sub-messages of information to automobiles, and the process of sending a packet along a link to a stage of the assembly. A pipeline results, for example, when the first processor decides to send sub-messages of some (roughly) fixed size, and all other processors both maintain the original packaging of the sub-messages, and choose to send the information contained in the most recent packet they have received in preference to acquiring new information.

If we pursue the analogy yet further, we note that in an *efficient* pipeline, the stages of the pipeline are as close as possible to the same size. In the automotive plant, if each stage of assembly takes the same time the workers will be idle as little as possible. Similarly, if each packet used in the pipeline is the same size, the communication links will be idle as little as possible.

For our purposes, then, to pipeline a message of size n along a path of m communication links is to break the message up into sub-messages of some size k and send the sub-messages into the path in succession. When k doesn't divide n evenly, we allow the last sub-message to be smaller. For simplicity, in the rest of the thesis, we refer to this sort of subdivision as dividing into "sub-messages of size k".

So how long does a pipeline process take? For any particular cost model (and thus choice of β and τ), there are only the three additional parameters we've seen: the size n of the message; the number of links m over which to send the message, and the size of the largest sub-message, k; we will therefore denote this time by $T_X(n,m,k)$, where X will be the appropriate label to distinguish the link bound and processor bound cases.

For the link bound case, Saad and Schultz [22] note that at least $\lceil n/k \rceil + m - 1$ rounds are required by a pipeline in sub-messages of size k. As a consequence, they employ the value

$$\left(\left\lceil \frac{n}{k} \right\rceil + m - 1\right)(\beta + k\tau)$$

to measure the time required to pipeline a message of size n over m links in packets of size k. Johnsson and Ho [16] note that the final round may be abbreviated when k does not divide n evenly, giving the formula we will employ:

$$S_*(n,m) \leq T_*(n,m,k) = \left(\left\lceil \frac{n}{k} \right\rceil + m - 1\right)\beta + ((m-1)\,k + n)\,\tau.$$

Specifically addressing link bound broadcasting, Johnsson and Ho [16] prove that rather than restricting a pipeline to a single path, one could pipeline into a directed tree: a processor simply sends each packet to each of its descendants in the tree. The pipelined scheme that results can broadcast in packets of size k to each node of a tree with depth m in time $T_*(n, m, k)$. As a result, they tend to model broadcast problems in a network G in terms of finding spanning subtrees for G of small depth.

For cases where a network has multiple arc-disjoint directed spanning trees, they demonstrate a more sophisticated approach. If a network G has a collection of arc-disjoint directed spanning subtrees $\{T_1, T_2, \ldots, T_j\}$, then one can broadcast a message to G by dividing the message into j sub-messages, and pipelining each one into its own spanning tree.

The upper bounds for the processor bound cases we examine have not seen a great deal of study. Saad and Schultz do not consider any processor bound problems. While Johnsson and Ho treat a 1-port case, it is slightly different than ours: they allow a processor to simultaneously listen to one port and transmit through another. And while Fraigniaud and Lazard employ the same processor bound model we will consider, their best upper bound for $T_1(n, m, k)$ is $T_*(2n, m, k)$ — which is at least $k\tau$ larger then the bound we employ:

$$S_1(n, m) \leq T_1(n, m, k) = \left(2\left\lceil\frac{n}{k}\right\rceil + m - 2\right)\beta + ((m-2)\,k + 2n)\,\tau.$$

Fraigniaud and Lazard extend the directed spanning tree and multiple arc-disjoint directed spanning tree schemes of Johnsson and Ho to the processor bound case by adding the notion of a *processor bound labeling.* For any given subtree, the labeling numbers all arcs that leave each vertex with a larger label than is on the arc that enters the vertex. In addition, the labeling ensures that no two edges (in any subtree) adjacent to the same vertex have the same label. (Pairs of arcs in opposite directions that connect two vertices may, however, have the same label.) The largest label ω in the scheme is an upper bound on the number of rounds required to send a single packet through all subtrees. The smallest value ϱ for which every arc adjacent a node has distinct labels modulo ϱ is an upper bound on the number of rounds required to send each subsequent packet. They thus derive an upper bound of

$$\min_{1 \leq k \leq n} \left(\omega + \varrho\left(\left\lceil\frac{n}{k}\right\rceil - 1\right)\right)(\beta + k\tau)$$

for a processor bound broadcast.

For either the link bound or processor bound case, the gossip algorithms presented by any of these authors for the cycle are nowhere near as sophisticated as their broadcast techniques. Nonetheless, the gossip schemes are quite effective: they simply send all information around the ring in one direction, until each processor has seen all messages.

Our contribution to the upper bounds, whether for broadcasting or gossiping, is twofold. In each case, we identify unused (or under-used) communication capacity in the existing schemes, and attempt to exploit it. In addition, we employ sending primitives that we prove are optimal.

Chapter 2

Sending a message over several links

2.1 The problem

Given our linear cost models, the simplest non-trivial communication problem is that of sending a single message of size n along a path of m communication links in some minimum time. As defined in the introduction, we denote this minimum possible sending time by $S_1(n,m)$ for the *processor bound* and $S_*(n,m)$ for the *link bound* constraints. We note that it is irrelevant whether the links are *full-duplex* or *half-duplex*: we will use all links in only one direction. This chapter proves matching lower and upper bounds for this sending problem in both the processor bound and link bound models.

There are two degenerate cases, for which a tight value of $S_*(n,m)$ is known [16]. If $n = 1$ or $\tau = 0$, the message must traverse all m communication links, and $S_*(n,m) = m(\beta + \tau)$. If $\beta = 0$, then since the message must traverse the first link in its entirety, and then has at least a latency of $(m-1)\tau$ before reaching the destination thereafter, $S_*(n,m) = (m + n - 1)\tau$. The general solution we will obtain will reduce, in the degenerate cases, to these same bounds.

This problem was first addressed by Saad and Schultz [22]. They employ the link bound model, and provide (in our notation) an upper bound of

$$S_*(n,m) \le \min_{1\le p\le n}\left\{(p+m-1)\left(\beta+\left\lceil\frac{n}{p}\right\rceil\tau\right)\right\},$$

where p is the number of sub-messages into which the message has been divided. They note that this upper bound is asymptotically equivalent to

$$\left(\sqrt{(m-1)\,\beta}+\sqrt{n\tau}\right)^2.$$

In this thesis, we we will show this value is a **lower** bound on the problem. Based on the formula of Saad and Schultz, a (non-asymptotic) upper bound in closed form would be

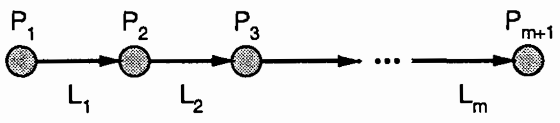
$$S_*(n,m) < \left(\sqrt{(m-1)\,(\beta+\tau)}+\sqrt{(n-1)\,\tau}\right)^2+\beta+\tau.$$

See Appendix A.1 for the derivation of this last inequality.

2.2 Model independent preliminaries

We first establish some notation for the problem:

- There are $m+1$ **processors**, $P_1, P_2, \ldots, P_{m+1}$. Processors with adjacent indices are connected by a communication **link**: P_i and P_{i+1}, for example, are connected by L_i.



- The **message** is an ordered set of n information units, $\{b_1, b_2, \ldots, b_n\}$. (Think of b as a mnemonic for *bit* or *byte.*)

- The message originates at P_1 and must be sent to the destination processor P_{m+1}.

- We let $\{\lambda(i,1), \lambda(i,2), \ldots, \lambda(i,\nu(i))\}$ represent the collection of $\nu(i)$ packets that are sent over L_i. Since the message must be transmitted in its entirety over each link L_i, this collection of packets forms a partition of the message set.

- The ordering of the packets in this partition is the natural order: the order the packets are sent. That is, $\lambda(i,j)$ was transmitted before $\lambda(i,j')$ if $j < j'$.

- Each packet $\lambda(i,j)$ has a **length** (or **size**) $|\lambda(i,j)|$ (simply the number of units in the packet) and a **start time** $st(\lambda(i,j))$. We will usually assume $st(\lambda(1,1)) = 0$.

- For some values of m and n, any broadcast **scheme** that sends a message of length n over m links may be described by a 3-tuple (λ, ν, st).

- We say that two schemes A and B are **equivalent** if there is a 1-1 mapping between the packets of A and the packets of B, that preserves length and start time for each packet.

In a theoretical sense, the problem of sending a message of size n through a network from one processor to another has no further constraints: the order that the message units arrive, for example, is not a factor. In practice, if a sending scheme could only guarantee that a message would arrive, and yet had the difficulty that the message would appear in a random order, the scheme might meet with limited acceptance. Fortunately, as this next lemma shows, such a randomization is not a drawback of *this* particular problem.

Lemma 2.2.1 *For some positive integers n and m, any scheme to send a message of size n over m links (from P_1 to P_{m+1}, as per our notation) can be replaced by an equivalent scheme in which each processor sends the message units in order $b_1, b_2, \ldots, b_n$.*

Proof: We prove the lemma by induction on m.

When $m = 1$, there is only a single link, and any permutation of the units of the message results in an equivalent scheme.

Suppose, then, that the lemma holds for some value $m = k$.

Consider a transmission over $k+1$ links. Much as in the single link case, we can apply a single permutation to the transmission as a whole, and obtain an equivalent scheme. We apply the inverse of the ordering used by the first link to all links simultaneously—the resulting transmission (the "slightly ordered" transmission) will necessarily be equivalent, yet the first link will transmit the message units in order.

From this slightly ordered transmission, we can isolate the last k links. By induction, we know that there is an equivalent transmission in which these last links send the message units in order. If we splice the ordered version of these last k links onto the ordered first link we will have an appropriately ordered equivalent transmission, as long as the second processor is not thereby forced to send a message unit before it has received it. In fact, this last clause cannot be satisfied: it would require that in the slightly ordered transmission, the second processor had sent more units than it has completely received—something prohibited by the model.

Any scheme for sending a message of size n over m links thus has an equivalent scheme in which all links send the message in canonical order. □

As a result of this equivalence, we will only concern ourselves with schemes that send the message in the same order along each link.

2.2.1 A note about figures

Many of the figures in this chapter will be attempting to portray a "transmission" in some visual form. The basic building block will be a packet sent between P_i and P_{i+1}:

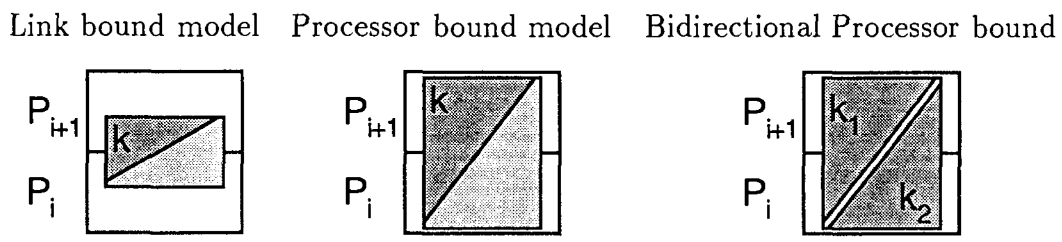


Figure 2.1: Three sample packet transfers.

The general notion behind the diagrams is that time is the x-axis and the activity of each processor is represented in a horizontal band. Rectangles that cross the division between processors represent packet transfers; direction is indicated by dark shading: information is sent toward the processor in which the horizontal edge of a dark triangle appears. (For example, the left two samples of Figure 2.1 show a packet of size k sent from P_i to P_{i+1}; the right sample shows two coincident transfers—size k_1 from P_i to P_{i+1} and size k_2 in the reverse direction.)

Under the link bound model, a processor may exchange information simultaneously with any of its neighbours, while in the processor bound model communication may only occur with a single processor at a time. Since we're only at present concerned with a line of processors, there are only two possible neighbours. The rectangle representing a packet transfer occupies only half of a processor's vertical space under the link bound model, so that communication is possible with the other neighbour. Similarly, under the processor bound model, all of both processors' vertical space is occupied by a rectangle representing a packet transfer.

Clearly, this sort of diagram is not appropriate for any topology more complex than the ring. What's more, it doesn't do a very good job of representing arbitrary two-way transmissions in the link bound model: we will tend to deal with those as two independent unidirectional transmissions. The rightmost sample of Figure 2.1 is sufficient to model full-duplex bidirectional processor bound transfers, however.

2.3 The link bound model

We're interested in the problem of sending a message of size n over m links in the minimum time: we let $S_*(n, m)$ represent this minimum time under the link bound constraint. In the link bound linear model, we assume that a processor may employ any of its adjacent links at the same time: specifically, that a processor may both send a message and receive another message at the same time. The "linear" portion of the model indicates that $\beta + k\tau$ time is required to transfer a single packet of size k over one link.

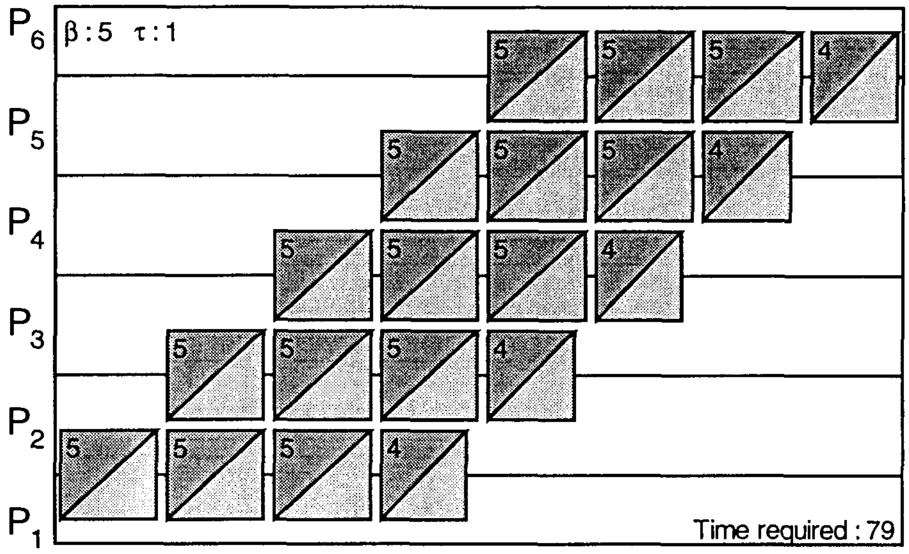


Figure 2.2: A pipelined transmission: $n = 19$, $m = 5$.

Following Johnsson and Ho [16], we obtain an upper bound on $S_*(n,m)$ by examining a pipelined transmission. First we select a packet size k. We then divide the n message units into $\lceil n/k \rceil$ sub-messages: the first $\lceil n/k \rceil - 1$ of size k and the last sub-message of size $n + k - \lceil n/k \rceil k$. Sending these sub-messages along a single link takes time $\lceil n/k \rceil \beta + n\tau$, since there are $\lceil n/k \rceil$ non-overlapping packets sent and a total of n message units sent. As illustrated in Figure 2.2, each additional link adds only $\beta + k\tau$ more time to this total. The time required for sending over m links results from sending the first sub-message (of size k) over the first $m-1$ links in succession, then sending all the packets along the last edge. As a result, a total of

$$(\lceil n/k \rceil + m - 1)\beta + ((m-1)k + n)\tau$$

time is required to send n message units over m links in packets of size k. We denote this quantity $T_*(n,m,k)$, and note that $S_*(n,m) \le T_*(n,m,k)$ for any value of k by construction. (The optimal choice of k to minimize $T_*(n,m,k)$ depends on n, m, β and τ.)

We will demonstrate a matching lower bound of

$$S_*(n,m) \geq \min_{1\leq k\leq n} T_*(n,m,k)$$

for the problem of sending a message of size n over m links with any pattern of packets permitted by the structural model. In contrast to the highly structured pipeline transmission used in the upper bound where the division into sub-messages is done globally and each sub-message is then sent over all links, we will assume no identification between the sets of packets sent over two different links. We proceed by examining the precedence structure of the set of packets in a transmission. Ideally, we'd like to find an ordered set of packets, $\{\lambda_1, \lambda_2, \ldots, \lambda_\nu\}$, in which for each adjacent pair of packets λ_i and λ_{i+1} the second packet must begin its transmission only after the first has finished, and the total time to send all the packets in the set is as large as possible. We proceed by constructing a digraph in which such paths may be relatively easily found.

We define the *packet content digraph* of a particular transmission, as follows:

- The set of nodes of the digraph is the union of the sets of packets in the transmission belonging to a link.

$$\bigcup_{1\leq i\leq m} \{\lambda(i,j) \mid 1 \leq j \leq \nu(i)\}$$

- The edge set is the union of the collections of adjacent packet edges and overlapping packet edges.

  Adjacent packet edges: pairs of adjacent packets on a single edge.

$$\bigcup_{1\leq i\leq m} \{\, (\lambda(i,j), \lambda(i,j+1)) \mid 1 \leq j < \nu(i)\}$$

  And *overlapping packet* edges: pairs of packets on adjacent links that share a unit.

$$\bigcup_{1\leq i<m} \{\, (\lambda(i,j), \lambda(i+1,j')) \mid \lambda(i,j) \cap \lambda(i+1,j') \neq \phi\}$$

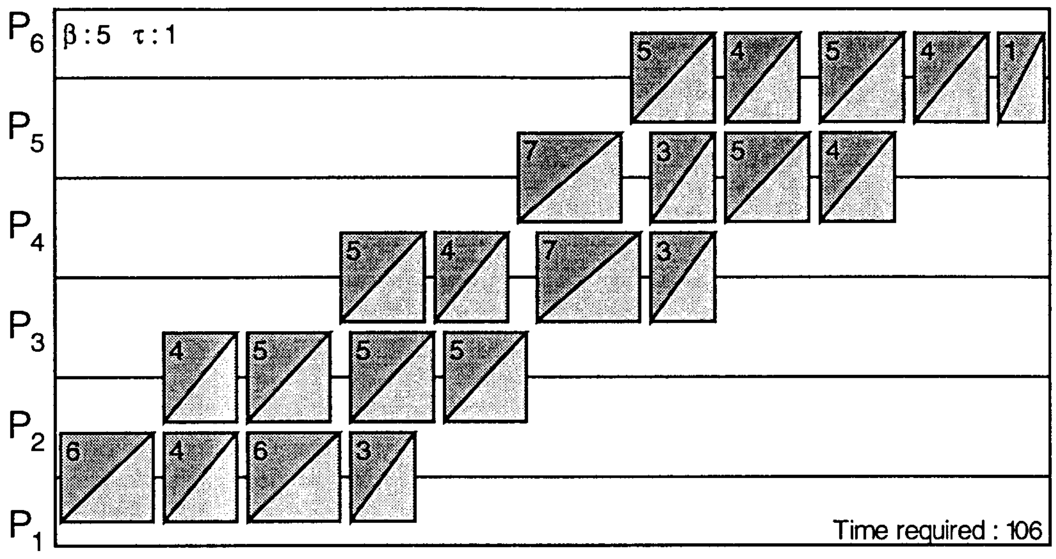


Figure 2.3: A random transmission.

As an illustration of the construction of a packet content digraph, consider the example of Figure 2.3 and Figure 2.4. The node labels in the digraph are the sizes of the respective packets.

The packet content digraph corresponds to the partial order in time of the transmission packets — or at least, that's a good intuition. In particular, if two packets are joined by an edge in the packet content digraph then the first packet must necessarily finish its transmission before the second is sent. (As one would expect, in the transitive closure of the packet content digraph the converse is true.) Accordingly, the total of the times required for the packets in some path in the digraph is a lower bound on the time required by the transmission as a whole. We'll see that every packet content digraph contains a path that requires at least as much time as some pipelined scheme.

In order to more easily manipulate the time required by the various components of a packet content digraph, we'll employ $t()$ as a functor on the digraph itself, as well as its vertices and paths. Any path in a packet content digraph D represents a sequence of packets that may not overlap in time. The time required for a path $\mathcal{A}$ in D is thus just the sum of the times required for each packet in the path. Accordingly,

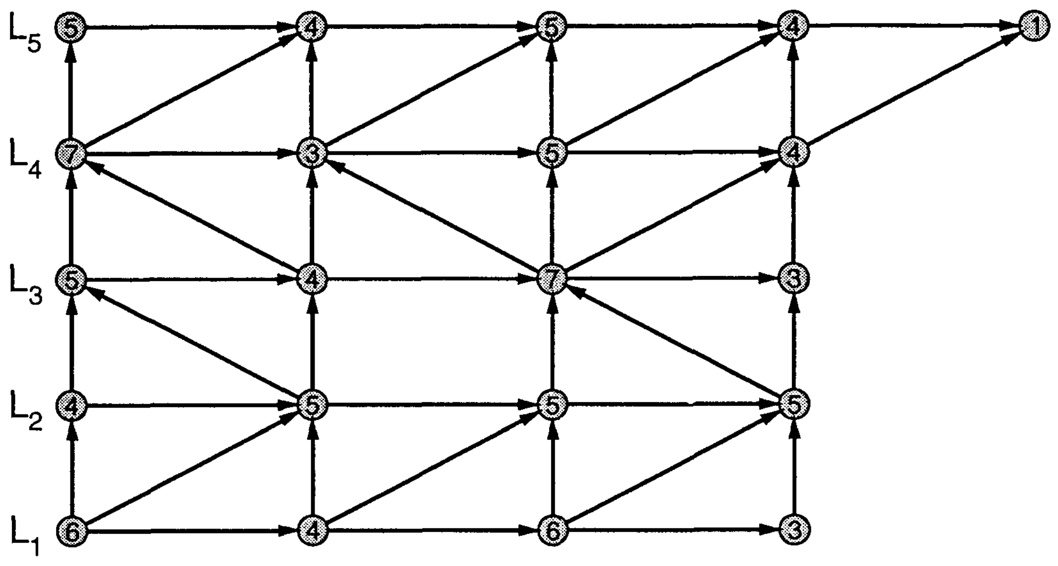


Figure 2.4: The packet content digraph of a random transmission.

we define $t(\lambda(i,j))$, $t(\mathcal{A})$, and $t(D)$:

$$t(\lambda(i,j)) = \beta + |\lambda(i,j)|\,\tau$$

$$t(\mathcal{A}) = \sum_{p\in\mathcal{A}} t(p)$$

$$t(D) = \max_{\mathcal{A}\in D} t(\mathcal{A}).$$

Interestingly, for a transmission scheme (λ, ν, st) the formulation of $t(D)$ is independent of the *start time* function. As a consequence, for the remainder of this section we will speak of transmission schemes as having just the two parameters (λ, ν).

Now, if D is the packet content digraph of some transmission, it would be nice to say that $t(D)$ is the time taken by the transmission. This might not be the case: a transmission could contain arbitrary delays not reflected in the constraints on the digraph. We note only that $t(D)$ is no greater than the time required by the transmission.

As an exercise of using the packet content digraph formalism, the following proposition gives a consistency check for the upper bound at the beginning of this section.

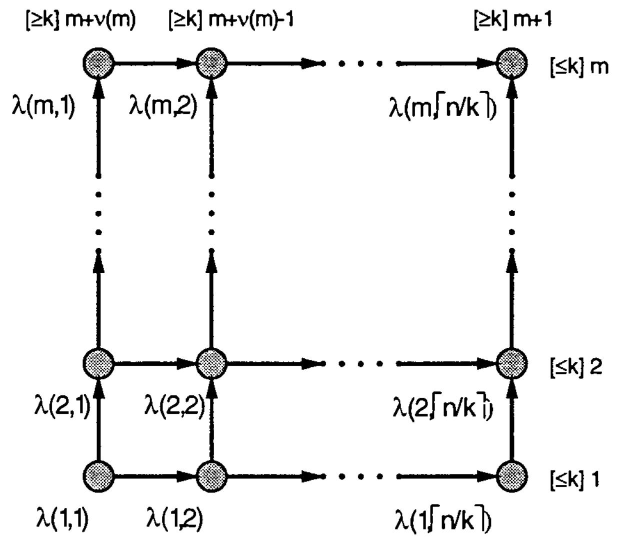


Figure 2.5: The packet content digraph of an equal-packet transmission.

Proposition 2.3.1 *For positive integers n and m, and for each integer k where $1 \leq k \leq n$, the packet content digraph of a transmission that pipelines a message of size n over m links in packets of size k contains a path $\mathcal{A}_u$ such that*

$$t(\mathcal{A}_u) \geq T_*(n, m, k).$$

Proof: Consider a transmission as described at the beginning of this section and illustrated in Figure 2.2: a message of size n divided into $\lceil n/k \rceil$ sub-messages, the first $\lceil n/k \rceil - 1$ of which are of size k. Each link sends the sub-messages in the same order. Moreover, all links send in lock-step so that the start time of any two packets $\lambda(i, j)$ and $\lambda(i', j')$ will be the same when $i + j = i' + j'$.

The packet content digraph (embedded in the Cartesian coordinates using the mapping $\lambda(i, j) \mapsto (i, j)$) forms a $\lceil n/k \rceil$ by m grid graph as in Figure 2.5

The transmission takes at least as much time as any path of packets in its packet content digraph. For example, the path

$$\lambda(1, 1)\lambda(2, 1) \ldots \lambda(m, 1)\lambda(m, 2) \ldots \lambda(m, \lceil n/k \rceil)$$

provides a good bound. Any path beginning at $\lambda(1,1)$ whose only "short" packet is $\lambda(m, \lceil n/k \rceil)$ provides the same bound, namely

$$T_*(n,m,k) = \left(\left\lceil \frac{n}{k} \right\rceil + m - 1\right)\beta + ((m-1)\,k + n)\,\tau.$$

□

In order to prove the lower bound, it will be sufficient to demonstrate the existence of a $\hat{k}$ so that $S_*(n,m) \geq T_*(n,m,\hat{k})$. We will present an algorithm that finds a path of that length in the packet content digraph of a minimal transmission.

To facilitate the description of paths through the digraph, we will first show that the digraph is planar, and provide a standard embedding with which to position the digraph in the plane—thus allowing the ordinary terminology of the Cartesian coordinates to apply to the digraph as well.

We use the standard lexicographic comparison with respect to packets. A packet $\lambda(i,j)$ is lexicographically greater than a packet $\lambda(i',j')$ if either $i > i'$ or $i = i'$ and $j > j'$.

Lemma 2.3.1 *The packet content digraph of a transmission is an acyclic planar digraph. The straight line embedding $\lambda(i,j) \mapsto (i,j)$ into the Cartesian plane is a plane embedding.*

Proof: Let D be the packet content digraph in question.

D must be acyclic: all edges are of the form (p,p') where p' is lexicographically greater than p.

The only edges that could possibly cross (given the embedding in the statement of the lemma) are two *overlapping packet* edges. (An *adjacent packet* edge can only intersect another edge at one of its endpoints.) If two such edges cross, then we must have four packets $\lambda(i,j)$, $\lambda(i,j')$, $\lambda(i+1,\ell)$, and $\lambda(i+1,\ell')$, with $j < j'$, $\ell < \ell'$ and edges $(\lambda(i,j),\lambda(i+1,\ell'))$ and $(\lambda(i,j'),\lambda(i+1,\ell))$. That configuration implies $\lambda(i,j) \cap \lambda(i+1,\ell') \neq \phi$ and $\lambda(i,j') \cap \lambda(i+1,\ell) \neq \phi$. Let b_k be a message unit in the first intersection. Then $\lambda(i,j') \subset \{b_{k+1},\ldots,b_n\}$, since b_k is a member of $\lambda(i,j)$, and all message units in $\lambda(i,j')$ strictly follow units in $\lambda(i,j)$ in message order. Similarly,

$\lambda(i+1,\ell) \subset \{b_1, b_2, \ldots, b_{k-1}\}$ since message units in $\lambda(i+1,\ell)$ are strictly before units of $\lambda(i+1,\ell')$ in message order. This is a contradiction with the fact that the second intersection is nonempty: the packet content digraph must be planar. □

Suppose that we have, for some positive n and m, a transmission (ν, λ) with packet content digraph D that takes time $S_*(n,m)$. We will attempt to find a long path in D that will show (for some positive integer $1 \le k \le n$) the desired bound: $S_*(n,m) \ge t(D) \ge T_*(n,m,k)$. Consider the following algorithm that finds a path $\mathcal{A}$ in D:

```
Algorithm SimpleSearchForward($\nu, \lambda, k$):
  INPUT:   transmission $(\nu, \lambda)$, and positive integer $k$.
  OUTPUT:  positive integer $p$.
  let $i \leftarrow 1$,   $j \leftarrow 1$,   $\mathcal{A} \leftarrow \phi$
  loop
        let $\mathcal{A} \leftarrow \mathcal{A}\lambda(i,j)$
        case $(|\lambda(i,j)|)$
        [$\le k$] do if $j = \nu(i)$ then return $i$
                  let $j \leftarrow j + 1$
              COMMENT:  Move to the *adjacent packet* neighbour of $\lambda(i,j)$.
              end do
        [$> k$] do if $i = m$ then return $m + \nu(m) + 1 - j$
                  let $j \leftarrow j'$ where $j'$ is minimal given $\lambda(i,j) \cap \lambda(i+1,j') \neq \phi$
                  let $i \leftarrow i + 1$
              COMMENT:  Move to the leftmost (in the embedding) *overlapping*
                        *packet* neighbour of $\lambda(i,j)$.
              end do
        end case
  end loop
```

As an example, Figure 2.6 shows the result of running **SimpleSearchForward** on the transmission that provides Figure 2.4 with input $k = 4$ and $k = 5$.

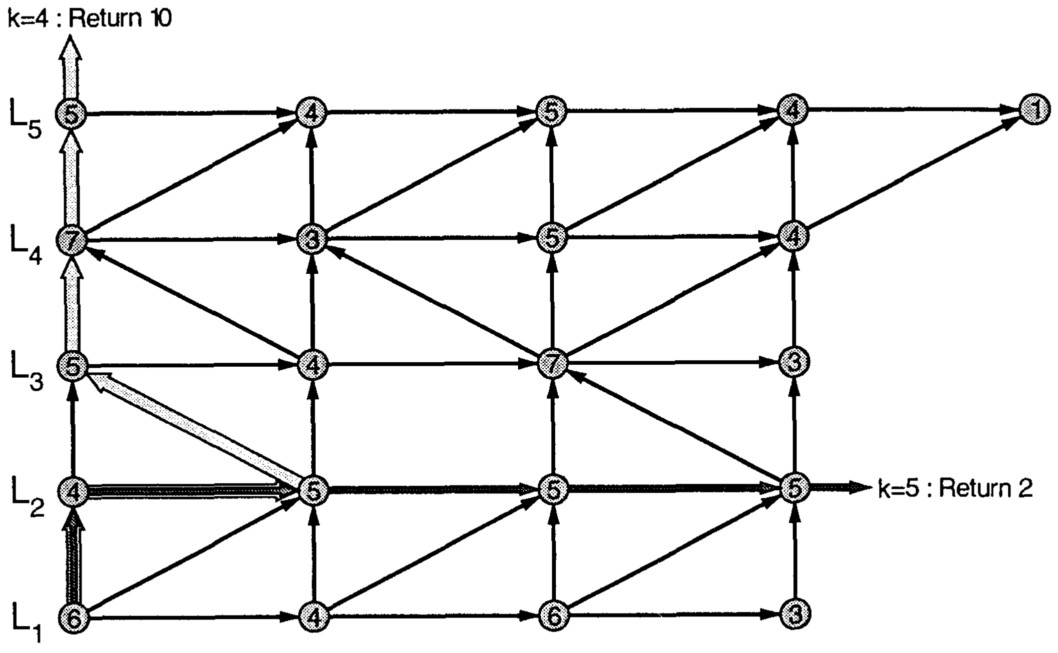


Figure 2.6: Two searches in the digraph of Figure 2.4.

The algorithm wanders through the packet content digraph, looking for a long path. It follows an overlapping packet edge whenever the current packet is larger than the input parameter k, and otherwise just takes the adjacent packet edge. When $k = 0$, **SimpleSearchForward**$(\nu, \lambda, 0)$ follows the left edge of the packet content digraph (in the standard embedding) and returns $m + \nu(m)$ while when $k = n$, **SimpleSearchForward** follows the bottom edge of the digraph and returns 1. As an example, the possible return values are noted about the top and right border of Figure 2.5. There must be some value $\hat{k}$ for which both:

$$\textbf{SimpleSearchForward}(\nu, \lambda, \hat{k} - 1) > m$$

and

$$\textbf{SimpleSearchForward}(\nu, \lambda, \hat{k}) \leq m.$$

In Figure 2.6, for example, $\hat{k} = 5$ satisfies the constraints.

By modifying the algorithm slightly, so that there is a non-deterministic choice of path when faced with packets that are the same size as the input parameter k,

SearchForward(ν, λ, k) will be able to produce the output of **SimpleSearchForward** with both $(\nu, \lambda, k-1)$ and (ν, λ, k) as inputs.

```
Algorithm SearchForward$(\nu, \lambda, k)$:
  INPUT: transmission $(\nu, \lambda)$ and positive integer $k$.
  OUTPUT: positive integer $p$.
  let $i \leftarrow 1$,   $j \leftarrow 1$,   $\mathcal{A} \leftarrow \phi$
  loop
      ASSERT: $t(\mathcal{A}) \geq T_*\left(\sum_{\ell=1}^{j-1} |\lambda(i,\ell)|, i, k\right)$
      let $\mathcal{A} \leftarrow \mathcal{A}\lambda(i,j)$
      NDcase $(|\lambda(i,j)|)$
      $[\leq k]$   do if $j = \nu(i)$ then return $i$
                      let $j \leftarrow j+1$
                   end do
      $[\geq k]$   do if $i = m$ then return $m + \nu(m) + 1 - j$
                      let $j \leftarrow \bar{j}$ where $\bar{j}$ is minimal given $\lambda(i,j) \cap \lambda(i+1,\bar{j}) \neq \phi$
                      let $i \leftarrow i+1$
                   end do
      end NDcase
  end loop
```

We first establish the correctness of the loop invariant. It establishes that at each iteration, whatever path has been traced up to but not including packet $\lambda(i,j)$ takes at least as long as a pipeline in packets of size k that sends the message units that precede $\lambda(i,j)$ over i links.

Lemma 2.3.2 *The assertion of Algorithm* **SearchForward** *is true.*

Proof: We wish to prove that

$$t(\mathcal{A}) \geq T_*\left(\sum_{\ell=1}^{j-1} |\lambda(i,\ell)|, i, k\right).$$

This is initially true. Assume that it remains true through some iterations of the algorithm. Let $\mathcal{A}, i, j$ be the values the variables held at the beginning of the loop where the assertion is true, and let $\bar{\mathcal{A}} = \mathcal{A}\lambda(i,j), \bar{\imath}, \bar{\jmath}$ be their values at the end.

If the $[\leq k]$ portion of the case was executed, then $\bar{\jmath} = j+1$, and $\bar{\imath} = i$.

$$\begin{aligned}
t(\bar{\mathcal{A}}) &= t(\mathcal{A}) + t(\lambda(i,j)) \\
&\geq T_*\left(\sum_{\ell=1}^{j-1} |\lambda(i,\ell)|, i, k\right) + \beta + |\lambda(i,j)|\,\tau \\
&= \left(\left\lceil \frac{1}{k}\sum_{\ell=1}^{j-1} |\lambda(i,\ell)| \right\rceil + i - 1\right)\beta + \left((i-1)\,k + \sum_{\ell=1}^{j-1} |\lambda(i,\ell)|\right)\tau + \beta + |\lambda(i,j)|\,\tau \\
&= \left(\left\lceil \frac{1}{k}\sum_{\ell=1}^{j-1} |\lambda(i,\ell)| \right\rceil + \left\lceil \frac{|\lambda(i,j)|}{k} \right\rceil + i - 1\right)\beta + \left((i-1)\,k + \sum_{\ell=1}^{j} |\lambda(i,\ell)|\right)\tau \\
&\geq \left(\left\lceil \frac{1}{k}\sum_{\ell=1}^{j} |\lambda(i,\ell)| \right\rceil + i - 1\right)\beta + \left((i-1)\,k + \sum_{\ell=1}^{j} |\lambda(i,\ell)|\right)\tau \\
&= \left(\left\lceil \frac{1}{k}\sum_{\ell=1}^{\bar{\jmath}-1} |\lambda(\bar{\imath},\ell)| \right\rceil + \bar{\imath} - 1\right)\beta + \left((\bar{\imath}-1)\,k + \sum_{\ell=1}^{\bar{\jmath}-1} |\lambda(\bar{\imath},\ell)|\right)\tau \\
&= T_*\left(\sum_{\ell=1}^{\bar{\jmath}-1} |\lambda(\bar{\imath},\ell)|, \bar{\imath}, k\right)
\end{aligned}$$

If the $[\geq k]$ portion of the case was executed, then $\bar{\imath} = i+1$, and $\bar{\jmath}$ is the smallest index for which $\lambda(i+1,\bar{\jmath}) \cap \lambda(i,j) \neq \phi$. If we remember that the message units are sent in the same order along each link, the definition of $\bar{\jmath}$ requires that

$$\sum_{\ell=0}^{j-1} |\lambda(i,\ell)| \geq \sum_{\ell=0}^{\bar{\jmath}-1} |\lambda(\bar{\imath},\ell)|$$

with equality only happening when the packets $\lambda(\bar{\imath},\bar{\jmath})$ and $\lambda(i,j)$ have the same first message unit.

$$\begin{aligned}
t(\bar{\mathcal{A}}) &= t(\mathcal{A}) + t(\lambda(i,j)) \\
&\geq T_*\left(\sum_{\ell=1}^{j-1} |\lambda(i,\ell)|, i, k\right) + \beta + |\lambda(i,j)|\,\tau
\end{aligned}$$

$$
\begin{aligned}
&\geq T_*\left(\sum_{\ell=1}^{j-1}|\lambda(i,\ell)|\,,i,k\right)+\beta+k\tau \\
&= \left(\left\lceil\frac{1}{k}\sum_{\ell=1}^{j-1}|\lambda(i,\ell)|\right\rceil+i+1-1\right)\beta+\left((i+1-1)\,k+\sum_{\ell=1}^{j-1}|\lambda(i,\ell)|\right)\tau \\
&\geq \left(\left\lceil\frac{1}{k}\sum_{\ell=1}^{\bar{\jmath}-1}|\lambda(\bar{\imath},\ell)|\right\rceil+\bar{\imath}-1\right)\beta+\left((\bar{\imath}-1)\,k+\sum_{\ell=1}^{\bar{\jmath}-1}|\lambda(\bar{\imath},\ell)|\right)\tau \\
&= T_*\left(\sum_{\ell=1}^{\bar{\jmath}-1}|\lambda(\bar{\imath},\ell)|\,,\bar{\imath},k\right)
\end{aligned}
$$

Thus, in either case, the assertion is still true at the end of the loop, thus, by induction, the assertion must hold for all complete iterations of the algorithm. □

The construction of algorithm **SearchForward** ensures that there is, for each input transmission (ν,λ), some value $\hat{k}$ for which there are at least two possible return values p and p', where $p \leq m < p'$. If $p = m$ then the algorithm has found a packet path that requires at least $T_*(n,m,\hat{k})$ time to complete, establishing the lower bound. Naturally, this will not necessarily be the case.

Figure 2.7 depicts the general situation. Notice that the two paths in the packet content digraph corresponding to the two return values p and p' bound a region that contains the last packet $\lambda(m,\nu(m))$. If we run the "same" algorithm backwards—using algorithm **SearchBackward** with the same input value $\hat{k}$, the resulting backward path must intersect one of the two forward ones. We will show that this composite path provides the required lower bound. For convenience, the possible return values of **SearchBackward** are noted in the figure.

```
Algorithm SearchBackward($\nu, \lambda, k$):
  INPUT:  transmission $(\nu, \lambda)$ and positive integer $k$.
  OUTPUT:  positive integer $p$.
  let $i \leftarrow m$,   $j \leftarrow \nu(m)$,   $\mathcal{B} \leftarrow \phi$
  loop
      ASSERT:  $t(\mathcal{B}) \geq T_*\left(\sum_{\ell=j+1}^{\nu(i)} |\lambda(i,\ell)|\,, m-i+1, k\right)$
```

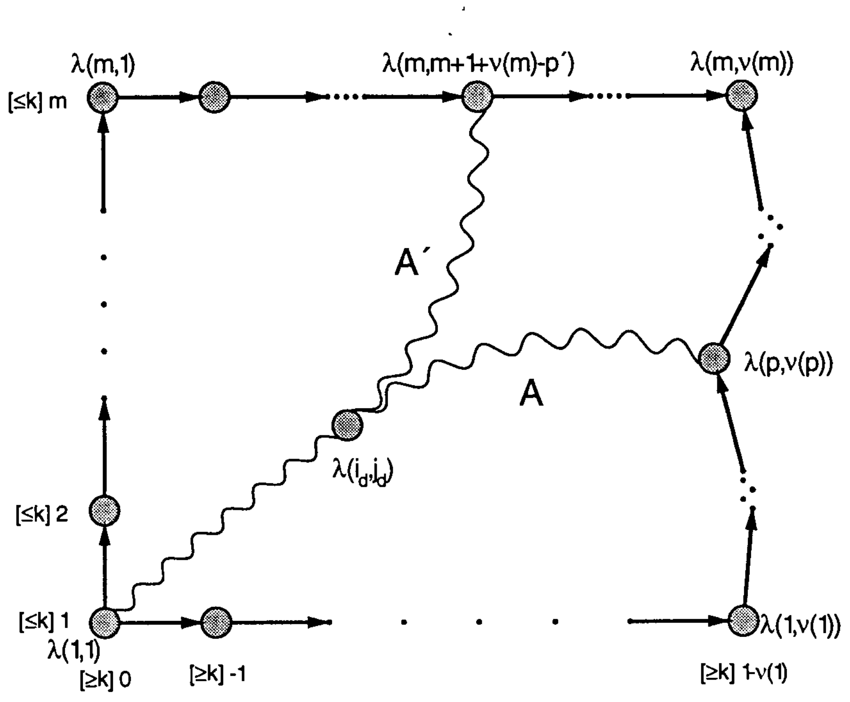


Figure 2.7: Two paths traced by **SearchForward**($\nu, \lambda, \hat{k}$)

```
    let $\mathcal{B} \leftarrow \lambda(i,j)\mathcal{B}$
    NDcase $(|\lambda(i,j)|)$
    $[\leq k]$    do if $j = 1$ then return $i$
                      let $j \leftarrow j - 1$
                  end do
    $[\geq k]$    do if $i = 1$ then return $1 - j$
                      let $j \leftarrow \bar{j}$ where $\bar{j}$ is maximal given $\lambda(i,j) \cap \lambda(i-1,\bar{j}) \neq \phi$
                      let $i \leftarrow i - 1$
                  end do
    end NDcase
end loop
```

We continue with our example: Figure 2.8 shows a possible result of running the backward search on the transmission that gave us Figure 2.4 with $k = 5$.

As the proof of the loop invariant for the backward algorithm is essentially the

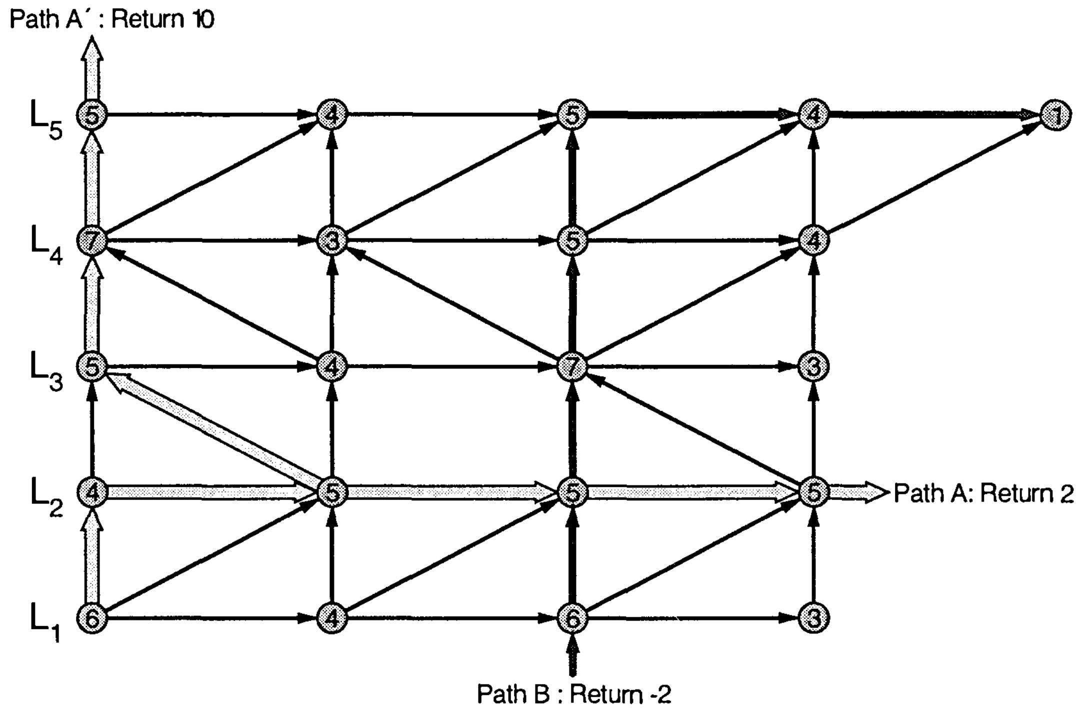


Figure 2.8: A backward search in the digraph of Figure 2.4.

same as for the forward one, we omit the proof.

Lemma 2.3.3 *The loop invariant assertion of Algorithm* **SearchBackward** *is true.*

Again let (ν, λ) be a transmission of a message of size n over m links that takes minimum time $S_*(n, m)$. Let $\hat{k}$ be an integer value for which **SearchForward**$(\nu, \lambda, \hat{k})$ has possible return values both greater than m and less than or equal m. Let $\mathcal{A}$ be the path traced by **SearchForward** in returning the smallest such return value $p \leq m$, and let $\mathcal{A}'$ be the path traced in returning the largest possible return value $p' > m$. And let $\mathcal{B}$ be any path traced by running the algorithm backward beginning from packet $\lambda(m, \nu(m))$. In the next two lemmas, we will first show that $\mathcal{B}$ intersects either $\mathcal{A}$ or $\mathcal{A}'$ in a packet of size no more than $\hat{k}$, then demonstrate that the concatenation of the two relevant sub-paths[1] is long enough to provide the lower bound.

[1]Suppose A is intersected by $\mathcal{B}$. The relevant sub-paths are the minimal sub-path of A that

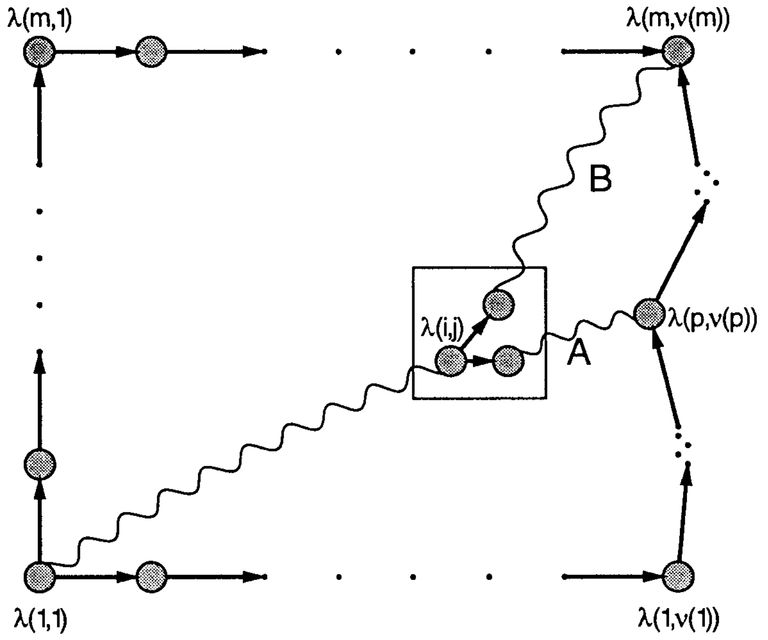


Figure 2.9: Path $\mathcal{A}$ intersects $\mathcal{B}$. Inset is Figure 2.10.

Lemma 2.3.4 *$\mathcal{B}$ intersects either $\mathcal{A}$ or $\mathcal{A}'$ in a packet $\lambda(i,j)$ where $|\lambda(i,j)| \leq \hat{k}$.*

Proof: Suppose first that $\mathcal{A}$, corresponding to the return $p \leq m$, intersects $\mathcal{B}$. (This implies that $|\lambda(p,\nu(p))| \leq \hat{k}$, since all returns less than or equal m are following a small packet — one of size no more than $\hat{k}$.) Figure 2.9 illustrates the scenario.

Let $\lambda(i,j)$ be the node of $\mathcal{A}$ farthest from $\lambda(1,1)$ that is also part of $\mathcal{B}$. If $\lambda(i,j)$ is the last element of $\mathcal{A}$ then $i = p$, and thus $|\lambda(i,j)| \leq \hat{k}$. Otherwise, $\lambda(i,j)$ has two possible successors in $\mathcal{A}$: the adjacent packet neighbour $\lambda(i,j+1)$ and the leftmost overlapping packet neighbour $\lambda(i+1,j')$. If the successor in $\mathcal{A}$ is the adjacent packet neighbour, $|\lambda(i,j)| \leq \hat{k}$ by the definition of the search algorithm.

Accordingly, we assume the worst: suppose the successor of $\lambda(i,j)$ in $\mathcal{A}$ is the overlapping packet neighbour—node o in Figure 2.10. Packet $\lambda(i,j)$ has several possible successors in $\mathcal{B}$: $\lambda(i,j+1)$ and any of the overlapping packet neighbours $\lambda(i+1,j'')$

contains both $\lambda(1,1)$ and the packet of intersection, and the maximal sub-path of $\mathcal{B}$ containing $\lambda(m,\nu(m))$ that does not contain the packet of intersection.

where $j' \leq j''$.

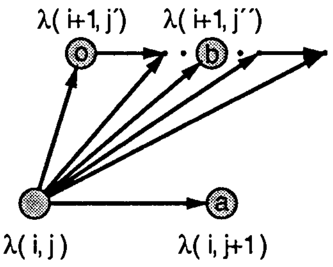


Figure 2.10: First case: the out-edges of $\lambda(i,j)$

The successor of $\lambda(i,j)$ in $\mathcal{B}$ cannot also be the leftmost overlapping packet neighbour of $\lambda(i,j)$. The successor of $\lambda(i,j)$ in $\mathcal{B}$ is thus either the adjacent packet neighbour $\lambda(i,j+1)$ (node a in the figure) or an overlapping packet neighbour $\lambda(i+1,j'')$ where $j' < j''$ (node b in the figure). By the planarity of the packet content digraph, $\mathcal{B}$ separates node o in $\mathcal{A}$ from the end of $\mathcal{A}$—$\lambda(p,\nu(p))$. As a result, there must be another node of intersection between $\mathcal{A}$ and $\mathcal{B}$ after node o: a contradiction with the definition of $\lambda(i,j)$. As a result, node a must be the successor of $\lambda(i,j)$ in $\mathcal{A}$, and so packet $\lambda(i,j)$ satisfies the lemma.

If $\mathcal{B}$ intersects both $\mathcal{A}$ and $\mathcal{A}'$ this first case still applies, so it only remains to consider the case where $\mathcal{B}$ intersects $\mathcal{A}'$ but does not intersect $\mathcal{A}$. Now $\mathcal{A}$ and $\mathcal{A}'$ have non-empty intersection since they both contain $\lambda(1,1)$, but can only diverge after a packet $\lambda(i_d, j_d)$ of size exactly $\hat{k}$. Accordingly, any intersection of $\mathcal{A}'$ with $\mathcal{B}$ must be at a packet $\lambda(i,j)$ with $i > i_d \geq 1$.

Let $\lambda(i,j)$ be the packet of $\mathcal{A}'$ that is also in $\mathcal{B}$ closest to $\lambda(i_d, j_d)$, as illustrated in Figure 2.11

If $\lambda(i,j)$ is the first element of $\mathcal{B}$ then $j = 1$, since $i > 1$. If $\mathcal{B}$ ends on the left edge of the digraph, it must be at a small packet: it must therefore be the case that $|\lambda(i,j)| \leq \hat{k}$.

Otherwise, $\lambda(i,j)$ is not the first element of $\mathcal{B}$. There are thus two possible predecessors for $\lambda(i,j)$ in $\mathcal{B}$, namely $\lambda(i,j-1)$ (node a in Figure 2.12) and the rightmost packet $\lambda(i-1,j'')$ that has $\lambda(i,j)$ as an overlapping packet neighbour (node o in the

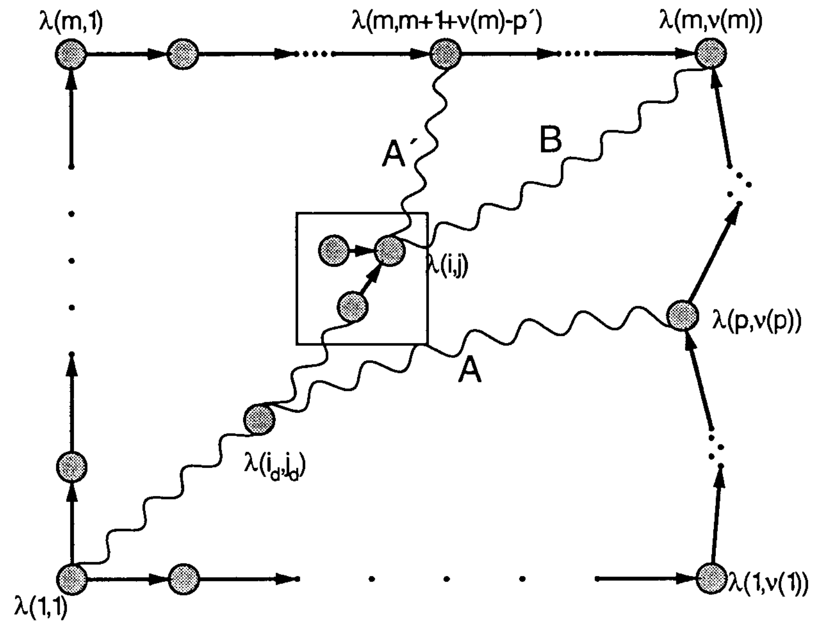


Figure 2.11: Path $\mathcal{A}'$ intersects $\mathcal{B}$. Inset is Figure 2.12.

figure). If the predecessor of $\lambda(i,j)$ in $\mathcal{B}$ is node a, then $\lambda(i,j)$ is of size at most $\hat{k}$ by the definition of the reverse search algorithm.

We again assume the worst: the predecessor of $\lambda(i,j)$ in $\mathcal{B}$ is node o. As before, either the predecessor of $\lambda(i,j)$ in $\mathcal{A}'$ must is coincident with o, or $\mathcal{A}'$ separates o from the end of $\mathcal{B}$: a contradiction in either case. As a result, the predecessor of $\lambda(i,j)$ in $\mathcal{B}$ is $\lambda(i,j-1)$, and $|\lambda(i,j)| \leq \hat{k}$.

Thus, $\mathcal{B}$ intersects either $\mathcal{A}$ or $\mathcal{A}'$ in a packet of size no more than $\hat{k}$. □

We've thus found a path $\mathcal{P}$ in the packet content digraph that extends from $\lambda(1,1)$ to $\lambda(m,\nu(m))$ composed of sub-paths found by the search algorithms. Let $\mathcal{A}_1$ be the path traced by algorithm **SearchForward** (a sub-path of either $\mathcal{A}$ or $\mathcal{A}'$) when the packet being considered is $\lambda(i,j)$, and similarly $\mathcal{B}_1$ be the path traced by **Search-Backward** when $\lambda(i,j)$ is being examined. Path $\mathcal{P} = \mathcal{A}_1\lambda(i,j)\mathcal{B}_1$ provides the lower bound.

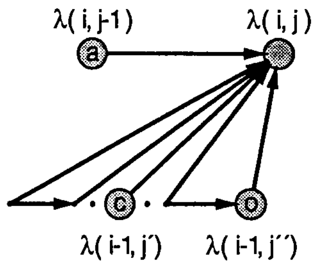


Figure 2.12: Second case: the in-edges of $\lambda(i,j)$

By the loop invariant assertions of **SearchForward** and **SearchBackward**:

$$t(\mathcal{A}_1) \geq T_* \left(\sum_{\ell=1}^{j-1} |\lambda(i,\ell)|, i, \hat{k} \right)$$

and

$$t(\mathcal{B}_1) \geq T_* \left(\sum_{\ell=j+1}^{\nu(i)} |\lambda(i,\ell)|, m+1-i, \hat{k} \right)$$

The time required for the whole path, then, is

$$\begin{aligned}
t(\mathcal{P}) &= t(\mathcal{A}_1) + t(\lambda(i,j)) + t(\mathcal{B}_1) \\
&\geq T_* \left(\sum_{\ell=1}^{j-1} |\lambda(i,\ell)|, i, \hat{k} \right) \\
&\quad + \beta + |\lambda(i,j)|\,\tau \\
&\quad + T_* \left(\sum_{\ell=j+1}^{\nu(i)} |\lambda(i,\ell)|, m+1-i, \hat{k} \right) \\
&= \left(\left\lceil \frac{1}{\hat{k}} \sum_{\ell=1}^{j-1} |\lambda(i,\ell)| \right\rceil + i - 1 \right) \beta \\
&\quad + \left((i-1)\,\hat{k} + \sum_{\ell=1}^{j-1} |\lambda(i,\ell)| \right) \tau \\
&\quad + \left\lceil \frac{|\lambda(i,j)|}{\hat{k}} \right\rceil \beta + |\lambda(i,j)|\,\tau \\
&\quad + \left(\left\lceil \frac{1}{\hat{k}} \sum_{\ell=j+1}^{\nu(i)} |\lambda(i,\ell)| \right\rceil + m - i \right) \beta
\end{aligned}$$

$$
\begin{aligned}
&+ \left((m-i)\,\hat{k} + \sum_{\ell=j+1}^{\nu(i)} |\lambda(i,\ell)| \right) \tau \\
=& \left(\left\lceil \frac{1}{\hat{k}} \sum_{\ell=1}^{j-1} |\lambda(i,\ell)| \right\rceil + \left\lceil \frac{|\lambda(i,j)|}{\hat{k}} \right\rceil + \left\lceil \frac{1}{\hat{k}} \sum_{\ell=j+1}^{\nu(i)} |\lambda(i,\ell)| \right\rceil + m - 1 \right) \beta \\
&+ \left((m-1)\,\hat{k} + \sum_{\ell=1}^{\nu(i)} |\lambda(i,\ell)| \right) \tau \\
\geq& \left(\left\lceil \frac{1}{\hat{k}} \sum_{\ell=1}^{\nu(i)} |\lambda(i,\ell)| \right\rceil + m - 1 \right) \beta + \left((m-1)\,\hat{k} + n \right) \tau \\
\geq& \left(\left\lceil \frac{n}{\hat{k}} \right\rceil + m - 1 \right) \beta + \left((m-1)\,\hat{k} + n \right) \tau \\
=& T_*(n, m, \hat{k}).
\end{aligned}
$$

Though the example is becoming tedious, we revisit Figure 2.4 one more time, to check that the path found is indeed longer than the 79 time units required by the pipelined transmission of Figure 2.2. The highlighted path in Figure 2.13 has nine packets, which contain a total of 42 message units. The pipelined transmission used $\beta = 5$ and $\tau = 1$: with those values, the highlighted path requires 87 time units.

We have thus shown:

Theorem 2.3.1 *The minimum time for sending a message of size n over m links is*

$$S_*(n,m) = \min_{1 \leq k \leq n} \left\{ \left(\left\lceil \frac{n}{k} \right\rceil + m - 1 \right) \beta + ((m-1)\,k + n)\,\tau \right\}.$$

We provide closed forms:

Corollary 2.3.1

$$
\begin{aligned}
\left(\sqrt{(m-1)\,\beta} + \sqrt{n\tau} \right)^2 &\leq S_*(n,m) \\
S_*(n,m) &< \left(\sqrt{(m-1)\,\beta} + \sqrt{(n-1)\,\tau} \right)^2 + \beta + m\tau
\end{aligned}
$$

The appropriate algebra may be found in Appendix A.2.1.

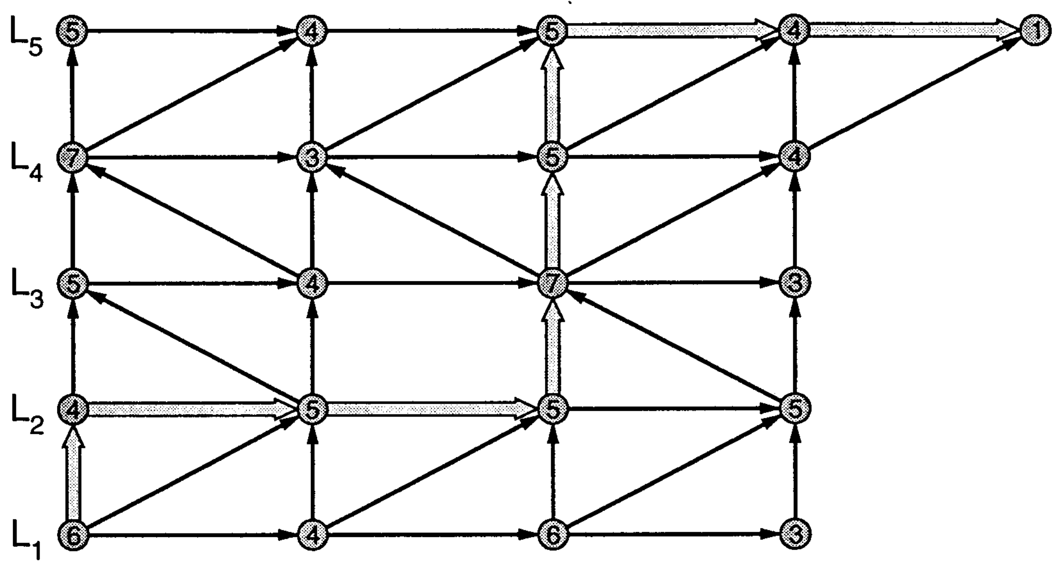


Figure 2.13: The long path found in Figure 2.4.

2.4 The processor bound model

We now turn our attention to the considerably more restrictive processor bound model. We are still interested in the problem of sending a message of size n over m links in the minimum time. Following the notation of Fraigniaud [7], we let $S_1(n, m)$ represent this minimum time under the processor bound constraint. Again, a packet of size k will take $\beta + k\tau$ time to transmit, and a processor must completely receive an entire packet before forwarding any portion of the information contained in that packet. But a processor is now restricted to either sending a message to a neighbour, or receiving a message from a neighbour: both may not occur simultaneously.

We obtain an upper bound by examining a pipelined transmission. As in the link bound case, we select a packet size k, and divide the n message units into $\lceil n/k \rceil$ sub-messages. Sending these sub-messages over two links takes time $2\lceil n/k \rceil \beta + 2n\tau$, since the middle processor will first receive then send each of the $\lceil n/k \rceil$ sub-messages, and will both send n message units and receive n message units. As illustrated in Figure 2.14, each additional edge only adds the time to send a single packet. We define $T_1(n, m, k)$ to be the time to send a message of size n over m links in a pipeline

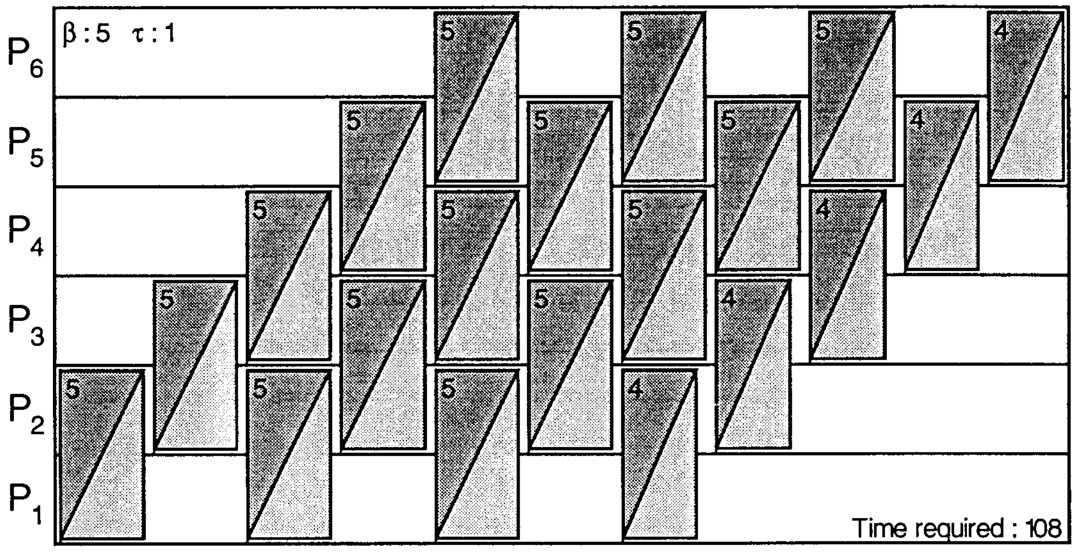


Figure 2.14: A pipelined transmission: $n = 19$, $m = 5$.

composed of packets of size k, namely

$$T_1(n,m,k) = \left(2\left\lceil\frac{n}{k}\right\rceil + m - 2\right)\beta + ((m-2)\,k + 2n)\,\tau.$$

By the definition of $S_1(n,m)$ it must be the case that $S_1(n,m) \leq T_1\,(n,m,k)$ for any value of k. We will demonstrate the corresponding lower bound:

$$S_1(n,m) \geq \min_{1\leq k\leq n} T_1\,(n,m,k)\,,$$

for the problem of sending—with any pattern of arbitrarily sized packets—a message of size n over m links in a manner permitted by the processor bound model.

We use essentially the same notation as the link bound case: Processors are numbered P_1 through P_{m+1}. For any processor P_i, we again denote the number of packets sent by $\nu(i)$, the j^{th} packet sent (for $1 \leq j \leq \nu(i)$) by $\lambda(i,j)$, the size of that packet by $|\lambda(i,j)|$ and the time required to send the packet $t(\lambda(i,j)) = \beta + |\lambda(i,j)|\,\tau$ If we assume that a transmission begins at time zero, we can assign a *start time* to each packet that corresponds to the elapsed real time before the packet begins sending. The start time of $\lambda(1,1)$ is thus zero for any transmission, and we label the start time of any packet $\lambda(i,j)$ by $st(\lambda(i,j))$. So that we may conveniently refer to the temporal relationship between packets, we define two-parameter successor and predecessor functions: by $\operatorname{succ}(\lambda(i,j),A)$ we mean the packet with the smallest start time sent by a processor whose index is in A that starts no earlier than when $\lambda(i,j)$ finishes transmitting. If no such packet exists, $\operatorname{succ}(\lambda(i,j),A)$ is undefined. The predecessor function is similarly defined: by $\operatorname{pred}(\lambda(i,j),A)$ we mean the packet with the largest start time sent by a processor whose index is in A that finishes being sent no later than $\lambda(i,j)$ starts transmitting. If no such packet exists, $\operatorname{pred}(\lambda(i,j),A)$ is undefined.

By employing these successor and predecessor operators, we can define the *packet time digraph G* for any processor bound transmission as follows:

- For the nodes of the digraph, we employ a new symbol: $\mu(i,j)$. The node set of the packet content digraph was just the collection of packets of the underlying transmission. In the case of the packet time digraph, however, the nodes do not represent the transmission of packets over communication links, but rather the sending and receiving of packets by processors. As many as two nodes may represent a packet in the digraph: one corresponding to the processor that sent it; the other corresponding to the processor that receives it. The nodes of the digraph are thus

  $$V(G) = \{\mu(i,j) \mid 2 \leq i \leq m,\ 1 \leq j \leq \nu(i-1) + \nu(i)\}.$$

  The majority of packets—all but those either received by P_{m+1} or sent by P_1—are represented twice in this node set. For some fixed i $(1 < i < m+1)$, the nodes $\mu(i,j)$ represent those packets that P_i either sends or receives sorted by start time. The apparent asymmetry (beginning at 2, that is) with the link-bound case is a result of the packet content digraph being defined against the packets that a *link* transmits, while the packet time digraph is defined against the packets seen (that is, sent or received) by a processor. We make the mapping from vertices of the digraph to packets explicit by defining the map $\mathrm{p}\colon V(G) \to \Lambda$ from $V(G)$ to the set of all packets $\Lambda = \{\lambda(\hat{\imath},\hat{\jmath}) \mid 1 \leq \hat{\imath} \leq m,\ 1 \leq \hat{\jmath} \leq \nu(i)\}$ as follows:

$$\begin{aligned} \mathrm{p}(\mu(i,1)) &= \lambda(i-1,1) \\ \mathrm{p}(\mu(i,j+1)) &= \mathrm{succ}\left(\mathrm{p}(\mu(i,j)),\{i-1,i\}\right). \end{aligned}$$

- The edge set is generated from the predecessor and successor operators as follows:

  Adjacent packet edges that connect pairs of vertices that correspond to pairs of packets that are consecutively active (either as a send or a receive) from the point of view of a single processor:

  $$\bigcup_{2\le i\le m} \{(\mu(i,j),\mu(i,j+1) \mid 1\le j<\nu(i-1)+\nu(i))\}$$

  or equivalently

  $$\bigcup_{2\le i\le m} \{(\mu(i,j),\mu(i,j')) \mid \mathrm{p}(\mu(i,j')) = \mathrm{succ}\left(\mathrm{p}(\mu(i,j)),\{i-1,i\}\right)\}.$$

  Successor edges—joining vertices that correspond to packet–successor pairs:

  $$\bigcup_{2\le i<m} \{(\mu(i,j),\mu(i+1,j')) \mid \mathrm{p}(\mu(i+1,j')) = \mathrm{succ}\left(\mathrm{p}(\mu(i,j)),\{i,i+1\}\right)\}.$$

  And *predecessor* edges which join vertices that correspond to predecessor–packet pairs:

  $$\bigcup_{2<i\le m} \{(\mu(i-1,\bar{\jmath}),\mu(i,j)) \mid \mathrm{p}(\mu(i-1,\bar{\jmath})) = \mathrm{pred}\left(\mathrm{p}(\mu(i,j)),\{i-2,i-1\}\right)\}.$$

Rather than describing the physical constraints of the packet contents, this digraph describes the physical constraint of the packet start times. But the underlying principle of the two digraphs are the same: nodes representing packets q and q' are connected by an edge (q,q') if q must finish sending before q' may start sending.

As an illustration of the packet time digraph, consider the example of Figure 2.15 and Figure 2.16. In the digraph, the lighter vertices are those whose corresponding to

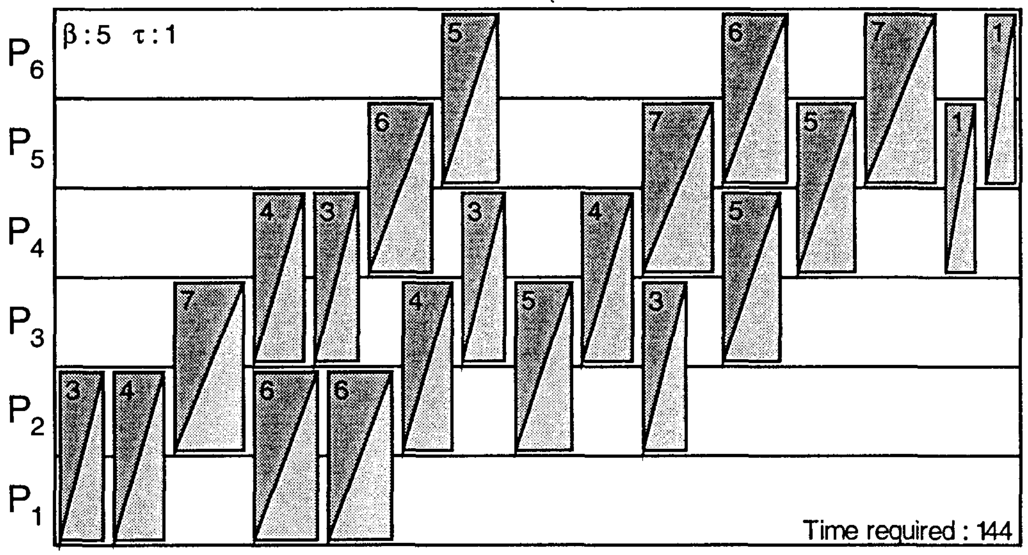


Figure 2.15: A random processor bound transmission.

packets sent, while the dark vertices correspond to packets received. And again, the node labels are the sizes of the corresponding packets.

For a packet time digraph G containing a path $\mathcal{A}$, we define $t(G)$ and $t(\mathcal{A})$ in concordance with the definitions of the link bound case:

$$t(\mathcal{A}) = \sum_{q \in \mathcal{A}} t(\mathrm{p}(q))$$

$$t(G) = \max_{\mathcal{A} \in G} t(\mathcal{A}).$$

The $\mathrm{p}(q)$ in the first definition is pedantic: q is a vertex of G, not a packet; we had only defined $t()$ for packets. In fact, it will be convenient to blur the distinction as much as possible—the vertices **are** packets, save that most are represented twice. Accordingly, we will drop the $\mathrm{p}()$ function wherever possible, and speak casually of $|q|$, $st(q)$ and the like.

For any transmission (λ, ν, st) with corresponding packet time digraph G, the time required to send all the packets is $st(\lambda(m, \nu(m))) + t(\lambda(m, \nu(m)))$. By the definition of the successor and predecessor functions, it must be the case that for any edge (q, q')

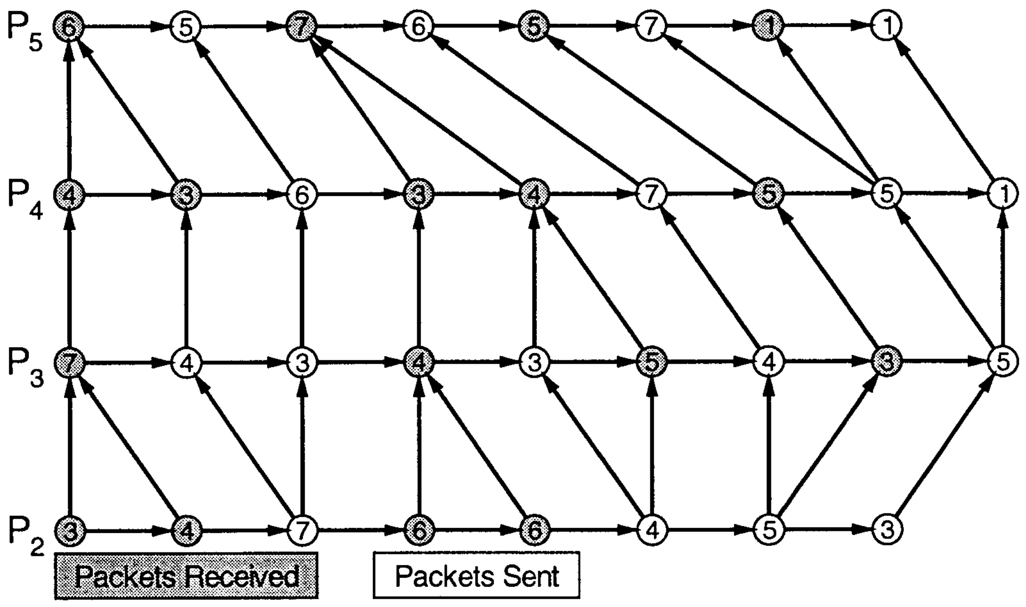


Figure 2.16: The packet content digraph of a random processor bound transmission.

in G, the start times of q and q' are related by $st(q) + t(q) \leq st(q')$. As a result,

$$t(G) \leq st(\lambda(m, \nu(m))) + t(\lambda(m, \nu(m))).$$

In order to demonstrate the lower bound, we will need to show, for some packet time digraph G corresponding to an arbitrary transmission and some integer k, that $t(G) \geq T_1(n, m, k)$.

As an illustration of the digraph mechanism, we demonstrate that if G_p is a packet time digraph corresponding to a pipelined scheme that sends a message of length n over m links in packets of size k (a transmission that is certainly not arbitrary) then the relation holds: $t(G_p) \geq T_1(n, m, k)$

Proposition 2.4.1 *(By analogy to Proposition 2.3.1) For positive integers n and m, and for any integer k, $1 \leq k \leq n$, the packet time digraph of a processor bound transmission that pipelines a message of size n over m links in packets of size k contains a path $\mathcal{B}_u$ that requires*

$$t(\mathcal{B}_u) \geq T_1(n, m, k).$$

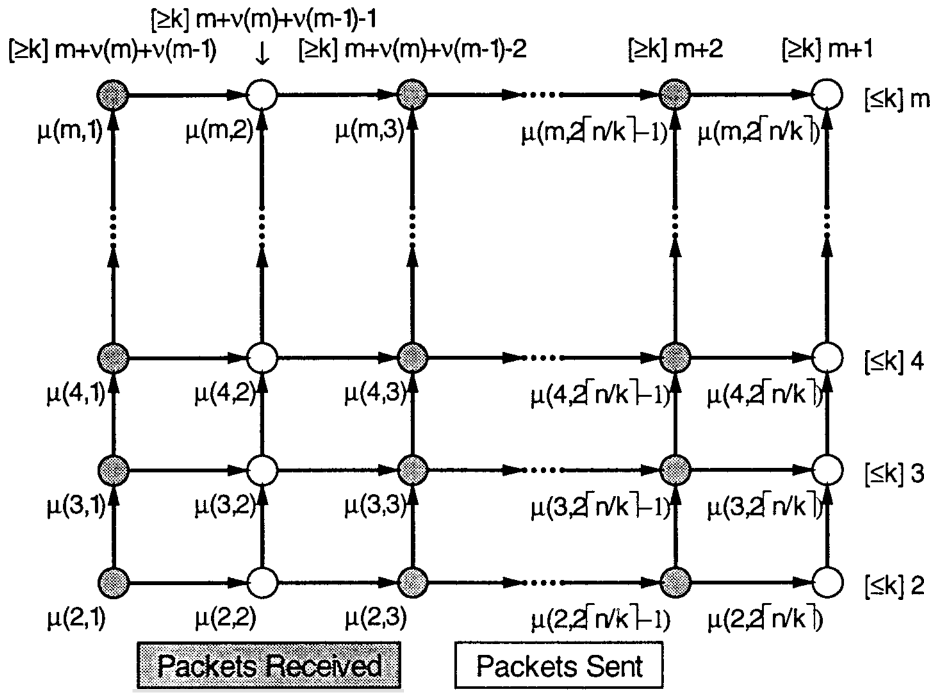


Figure 2.17: The packet time digraph of an equal-packet transmission.

Proof: Consider a transmission as illustrated in Figure 2.14: a message of size n divided into $\lceil n/k \rceil$ sub-messages, the first $\lceil n/k \rceil - 1$ of which are of size k. The sub-messages are sent along each link in the same order. The start time of any two packets $\lambda(i,j)$ and $\lambda(i',j')$ will be the same when $i+2j = i'+2j'$, and the packet time digraph (in its canonical projection into the Cartesian coordinates) forms a $2\lceil n/k \rceil$ by $m-1$ grid graph.

The transmission takes time equal to the time for the path

$$\mu(2,1)\mu(3,1)\ldots\mu(m,1)\mu(m,2)\mu(m,2\lceil n/k \rceil)$$

which is

$$T_1(n,m,k) = (2\lceil n/k \rceil + m - 2)\beta + ((m-2)k + 2n)\tau.$$

□

In order to demonstrate the general relation, we will again employ an algorithm to wander through the digraph and find a long path. We will proceed much as in the link bound case: we will first show that the packet time digraph is planar. Yet in the link bound case, we used the canonical order together with the physical constraints of packet content in proving the loop assertion of algorithm **SearchForward**. In this case, the additional time information maintained in G allow us to dispense with the canonical order.

Lemma 2.4.1 *The packet time digraph of a transmission is an acyclic planar digraph. The straight line embedding $\mu(i,j) \mapsto (i,j)$ into the Cartesian plane is a plane embedding.*

Proof: Let G be the packet time digraph in question.

G must be acyclic, since every edge is of the form (v,w) where $st(v) < st(w)$.

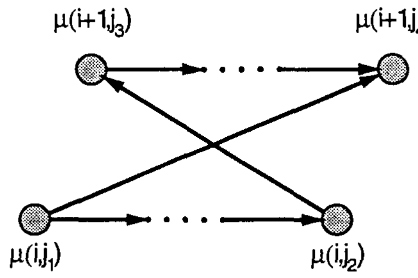


Figure 2.18: An impossible lack of planarity.

Suppose the embedding were not planar. The only possible crossing would concern some vertices $\mu(i,j_1)$, $\mu(i,j_2)$, $\mu(i+1,j_3)$ and $\mu(i+1,j_4)$ with $j_1 < j_2$ and $j_3 < j_4$ if there were edges $(\mu(i,j_1),\mu(i+1,j_4))$ and $(\mu(i,j_2),\mu(i+1,j_3))$, as shown in Figure 2.18.

By the construction of the edges, it's necessary that $st(\mu(i,j_1)) < st(\mu(i,j_2)) < st(\mu(i+1,j_3)) < st(\mu(i+1,j_4))$. The edge $(\mu(i,j_1),\mu(i+1,j_4))$ is either a successor edge from $\mu(i,j_1)$ or a predecessor edge from $\mu(i+1,j_4)$. If the first, $\mu(i+1,j_3)$ would have been chosen as destination in preference to $\mu(i+1,j_4)$; and if the second, $\mu(i,j_2)$ would have been chosen as source in preference to $\mu(i,j_1)$. As a consequence, edge $(\mu(i,j_1),\mu(i+1,j_4))$ cannot appear in any context of this form, and the digraph must be planar. □

We define several more auxiliary functions for use with the packet digraph. First, for any vertex $\mu(i,j)$ in the digraph, we are interested in being able to refer to both the number of message units **received** by processor P_i and the number of message units **sent** by the same processor prior to packet referred to by $\mu(i,j)$ being sent. With no regard at all for the finer sensibilities of the reader, we simply label these quantities, respectively, $f_r(\mu(i,j))$ and $f_s(\mu(i,j))$.

As an example, consider the vertex $\mu(3,6)$ in Figure 2.16 (The grey node labeled 5 in the row associated with P_3.) Before the packet associated with $\mu(3,6)$ is sent, P_3 has received two packets ($\mu(3,1)$ and $\mu(3,4)$) containing a total of 11 message units, and P_3 has send three packets ($\mu(3,2)$, $\mu(3,3)$, and $\mu(3,5)$) containing a total of 10 message units. Therefore, $f_s(\mu(3,6)) = 10$ and $f_r(\mu(3,5)) = 11$.

We'll use the quantities $f_r(\mu(i,j))$ and $f_s(\mu(i,j))$ in the statement of the assertion of the non-deterministic algorithm for this case. In order to manipulate the values, we need the following lemma.

Lemma 2.4.2 *For any vertex q of G, the following hold:*

1. $f_r(q) \geq f_s(q)$
2. *If (q,r) is an adjacent packet edge of G,* $f_r(q) + f_s(q) + |q| = f_r(r) + f_s(r)$
3. *If (q,r) is a successor edge of G,* $f_r(q) + f_s(q) \geq f_r(r) + f_s(r)$

Proof: We prove each point in turn:

1. $f_r(q)$ is the number of message units sent by some processor P_{i-1} at time $st(q)$, while $f_s(q)$ is the number of units sent by P_i at the same time. Since each message unit sent by P_i must first be received from P_{i-1}, $f_r(q) \geq f_s(q)$
2. Suppose $q = \mu(i,j)$, then the only information transfer done by processor P_i between time $st(q)$ and $st(r)$ is exactly packet q; thus, either:

q is received by P_i: $f_r(q) + |q| = f_r(r)$ and $f_s(q) = f_s(r)$ or

q is sent by P_i: $f_r(q) = f_r(r)$ and $f_s(q) + |q| = f_s(r)$.

and in either case, $f_r(q) + f_s(q) + |q| = f_r(r) + f_s(r)$.

3. We must have $f_s(r) \leq f_r(r)$ and $f_s(q) \leq f_r(q)$. We again distinguish two cases:

   q is received by P_i: Then P_i does not transfer any information to P_{i+1} between $st(q)$ and $st(r)$. As a result, $f_r(r)$—the number of bits P_{i+1} has received at time $st(r)$—must be equal $f_s(q)$—the number of bits P_i has sent at time $st(q)$. Therefore $f_s(r) \leq f_r(r) = f_s(q) \leq f_r(q)$.

   q is sent by P_i: The only transfer of information between P_i and P_{i+1} between time $st(p)$ and $st(r)$ is q itself. As a result, $f_r(r) = f_s(q) + |q|$. But since P_i was able to send q at time $st(p)$, it must be the case that $f_s(q) + |q| \leq f_r(q)$. And in addition, P_{i+1} will not have been able to forward any of the information in q: thus $f_s(r) + |q| \leq f_r(r)$.

   In either case, $f_r(q) + f_s(q) \geq f_r(r) + f_s(r)$.

The lemma follows. □

We will omit the presentation of the deterministic version of the algorithm to search a packet time digraph. For any packet time digraph G of a link bound transmission, there must exist some input $\hat{k}$ so that **SearchFwdLink**$(G, \hat{k})$ has returns p and p', where $p \leq m < p'$.

```
Algorithm SearchFwdLink(G, k):
  INPUT:  packet time digraph G and positive integer k.
  OUTPUT:  positive integer p.
  let $q \leftarrow \mu(2,1)$,   $\mathcal{A} \leftarrow \phi$
     (We will assume that $i, j$ are defined so that $q \equiv \mu(i,j)$)
  loop
     ASSERT:  $t(\mathcal{A}) \geq \left(\left\lceil \frac{1}{k} f_r(q) \right\rceil + \left\lceil \frac{1}{k} f_s(q) \right\rceil + i - 2\right) \beta$
                        $+ \; ((i-2)k + f_r(q) + f_s(q))\, \tau$
     let $\mathcal{A} \leftarrow \mathcal{A}q$
     NDcase $(|q|)$
```

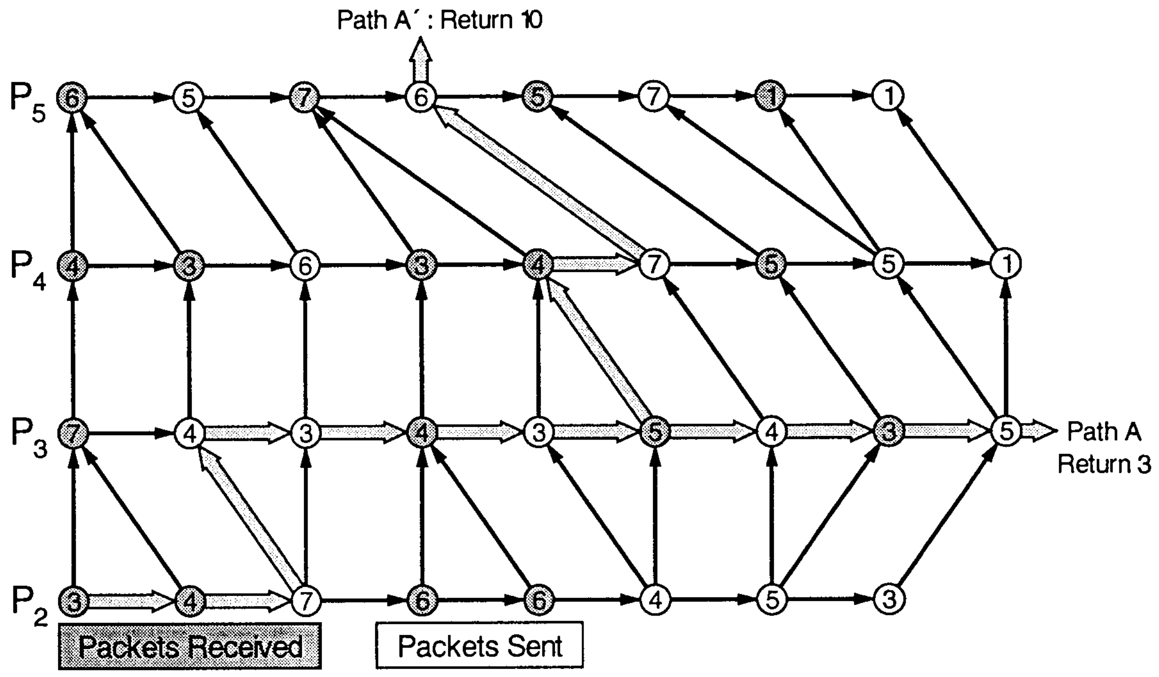


Figure 2.19: Two searches in the digraph of Figure 2.16.

```
        [≤ k]   do if $j = \nu(i-1) + \nu(i)$ then return $i$
                    let $q \leftarrow \bar{q}$ where $(q, \bar{q})$ is the adjacent packet edge leaving $q$.
                end do
        [≥ k]   do if $i = m$ then return $m + 1 + \nu(m-1) + \nu(m) - j$
                    let $q \leftarrow \bar{q}$ where $(q, \bar{q})$ is the successor out-edge from $q$.
                end do
        end NDcase
    end loop
```

As an example, Figure 2.19 shows two possible results of running **SearchFwdLink** on the transmission that provides Figure 2.16 with input $k = 5$. (In fact, the path $\mathcal{A}$ corresponds to the path traced by the missing deterministic algorithm with an input of $k = 6$, and the path $\mathcal{A}$ to the deterministic result when $k = 5$.)

Lemma 2.4.3 *The loop invariant assertion of Algorithm* **SearchFwdLink** *is true.*

Proof: We need to prove

$$t(\mathcal{A}) \geq \left(\left\lceil\frac{f_r(q)}{k}\right\rceil + \left\lceil\frac{f_s(q)}{k}\right\rceil + i - 2\right)\beta + ((i-2)\,k + f_r(q) + f_s(q))\,\tau$$

for any iteration of the algorithm.

This is initially true: $\mathcal{A} = \phi$ and the expression evaluates to zero. Assume that it remains true through some iterations of the algorithm.

Let q (with associated i, j) and $\mathcal{A}$ be the values the variables held at the beginning of the loop where the assertion is true, and let $\bar{q}$ (with associated $\bar{\imath}, \bar{\jmath}$) and $\bar{\mathcal{A}} = \mathcal{A}q$ be the values at the end of the same iteration.

Suppose first that the $[\leq k]$ portion of the non-deterministic case was executed, so $(q, \bar{q})$ is an adjacent packet edge, and $\bar{\imath} = i$. We employ the second part of Lemma 2.4.2.

$$\begin{aligned}
t(\bar{\mathcal{A}}) &= t(\mathcal{A}) + t(q) \\
&\geq \left(\left\lceil\frac{f_r(q)}{k}\right\rceil + \left\lceil\frac{f_s(q)}{k}\right\rceil + i - 2\right)\beta + \beta \\
&\quad + ((i-2)\,k + f_r(q) + f_s(q))\,\tau + |q|\,\tau \\
&= \left(\left\lceil\frac{f_r(q)}{k}\right\rceil + \left\lceil\frac{|q|}{k}\right\rceil + \left\lceil\frac{f_s(q)}{k}\right\rceil + i - 2\right)\beta \\
&\quad + ((i-2)\,k + f_r(q) + f_s(q) + |q|)\,\tau \\
&\geq \left(\left\lceil\frac{f_r(\bar{q})}{k}\right\rceil + \left\lceil\frac{f_s(\bar{q})}{k}\right\rceil + i - 2\right)\beta \\
&\quad + ((i-2)\,k + f_r(\bar{q}) + f_s(\bar{q}))\,\tau \\
&= \left(\left\lceil\frac{f_r(\bar{q})}{k}\right\rceil + \left\lceil\frac{f_s(\bar{q})}{k}\right\rceil + \bar{\imath} - 2\right)\beta + ((\bar{\imath}-2)\,k + f_r(\bar{q}) + f_s(\bar{q}))\,\tau
\end{aligned}$$

If the $[\geq k]$ portion of the non-deterministic case was executed, then $(q, \bar{q})$ is a successor edge, and $\bar{\imath} = i + 1$. We employ the third part of Lemma 2.4.2.

$$\begin{aligned}
t(\bar{\mathcal{A}}) &= t(\mathcal{A}) + t(q) \\
&\geq t(\mathcal{A}) + \beta + k\tau \\
&\geq \left(\left\lceil\frac{f_r(q)}{k}\right\rceil + \left\lceil\frac{f_s(q)}{k}\right\rceil + i - 2\right)\beta + \beta + ((i-2)\,k + f_r(q) + f_s(q))\,\tau + k\tau
\end{aligned}$$

$$\begin{aligned}
&= \left(\left\lceil \frac{f_r(q)}{k} \right\rceil + \left\lceil \frac{f_s(q)}{k} \right\rceil + \bar{\imath} - 2\right)\beta + ((\bar{\imath} - 2)\,k + f_r(q) + f_s(q))\,\tau \\
&\geq \left(\left\lceil \frac{f_r(\bar{q})}{k} \right\rceil + \left\lceil \frac{f_s(\bar{q})}{k} \right\rceil + \bar{\imath} - 2\right)\beta + ((\bar{\imath} - 2)\,k + f_r(\bar{q}) + f_s(\bar{q}))\,\tau
\end{aligned}$$

Thus, in either case, the assertion is still true at the end of the loop. Therefore, by induction, the assertion must hold for all complete iterations of the algorithm. □

As before, there is an input value $\hat{k}$ for which the algorithm produces two possible return values p and p', where $p \leq m < p'$. The two paths corresponding to these nondeterministic returns bound the "upper right" corner of the packet time digraph—upper right, that is, in the canonical embedding. In our example Figure 2.19, a value of 5 gives the two paths. We employ a "backward" analogue of algorithm **SearchFwdLink**, (one that follows the edges backward, and selects between the adjacent packet neighbour and the predecessor edge neighbour) beginning at $\mu(m, \nu(m-1)+\nu(m))$. The path traced by the backward algorithm will necessarily intersect one of the two forward paths.

In the same way that $f_r(\mu(i,j))$ and $f_s(\mu(i,j))$ refer to the number of message units received by P_i and the number of message units sent by P_i at the moment that the packet associated with $\mu(i,j)$ begins sending, we would like to be able to speak of the number of message units that remain to be received or sent **after** a packet has finished sending. By analogy to f_r and f_s, we label these quantities, respectively $\ell_r(\mu(i,j))$ and $\ell_s(\mu(i,j))$. Accordingly, for any vertex q of the packet time digraph, either q is sent by P_i, and

$$f_r(q) + \ell_r(q) = n \quad \text{and} \quad f_s(q) + \ell_s(q) = n - |q|$$

or q is received by P_i, and

$$f_r(q) + \ell_r(q) = n - |q| \quad \text{and} \quad f_s(q) + \ell_s(q) = n.$$

In either case, we have:

$$f_r(q) + f_s(q) + |q| + \ell_r(q) + \ell_s(q) = 2n. \tag{2.1}$$

Again, consider the vertex $\mu(3,6)$ in Figure 2.16 (The grey node labeled 5 in the row associated with P_3.) After the packet associated with $\mu(3,6)$ is sent, P_3 must still receive one packet ($\mu(3,8)$) containing a total of 3 message units, and P_3 must still send two packets ($\mu(3,7)$ and $\mu(3,9)$) containing a total of 9 message units. Therefore, $\ell_s(\mu(3,6)) = 9$ and $\ell_r(\mu(3,5)) = 3$. In addition, identity 2.1 holds:

$$f_r(\mu(3,6)) + f_s(\mu(3,6)) + |\mu(3,6)| + \ell_r(\mu(3,6)) + \ell_s(\mu(3,6))$$

$$= 11 + 10 + 5 + 3 + 9 = 2 \cdot 19 = 2n.$$

We specify the "transposed" algorithm **SearchBackLink** explicitly:

```
Algorithm SearchBackLink($(G, k)$):
  INPUT:  packet time digraph $G$ and positive integer $k$.
  OUTPUT:  positive integer $p$.
  let $q \leftarrow \mu(m, \nu(m) + \nu(m-1))$,   $\mathcal{B} \leftarrow \phi$
     (We will assume that $i, j$ are defined so that $q \equiv \mu(i,j)$)
  loop
      ASSERT:   $t(\mathcal{B}) \geq \left(\left\lceil \frac{1}{k}\ell_r(q)\right\rceil + \left\lceil \frac{1}{k}\ell_s(q)\right\rceil + m - i\right)\beta$
                       $+ \; ((m-i)\,k + \ell_r(q) + \ell_s(q))\,\tau$
      let $\mathcal{B} \leftarrow q\mathcal{B}$
      NDcase $(|q|)$
      $[\leq k]$     do if $j = 1$ then return $i$
                    let $q \leftarrow \bar{q}$ where $(\bar{q}, q)$ is the adjacent packet edge entering q.
                 end do
      $[\geq k]$     do if $i = 2$ then return $2 - j$
                    let $q \leftarrow \bar{q}$ where $(\bar{q}, q)$ is the predecessor in-edge into $q$.
                 end do
      end NDcase
  end loop
```

Continuing with our small example, Figure 2.20 shows a possible backward path traced when $k = 5$.

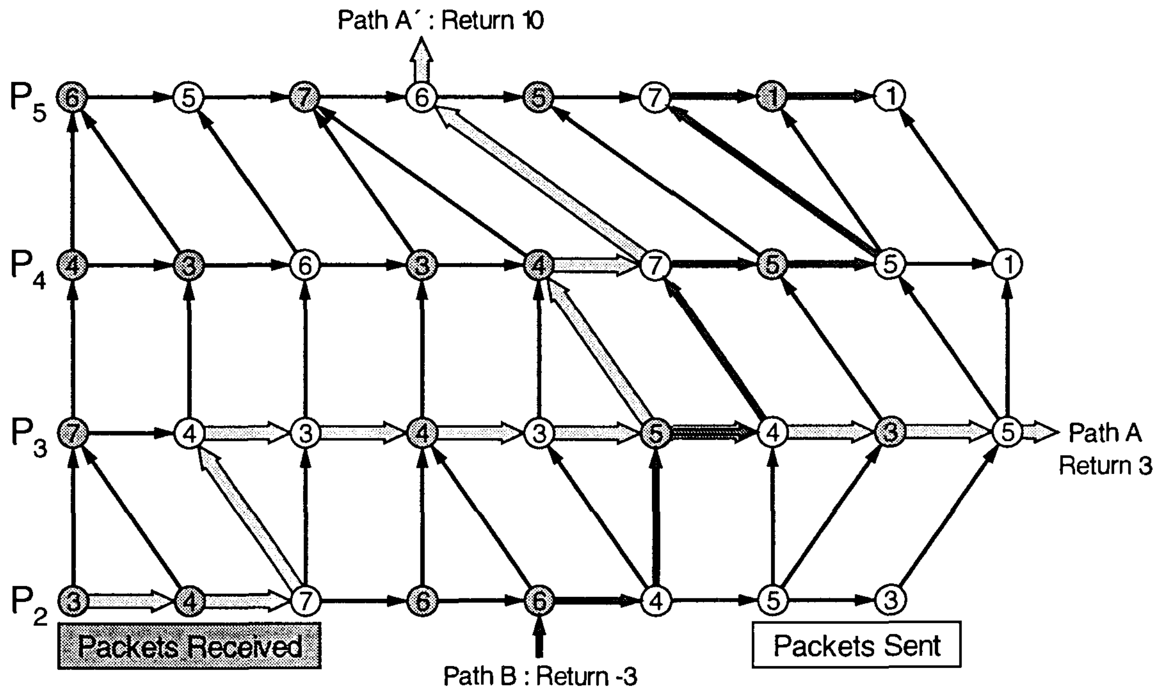


Figure 2.20: A backward search in the digraph of Figure 2.16.

Following the example of the previous section, we prove the loop invariant, and then demonstrate that the "backward" path has an intersection with one of the two forward paths in a small packet — one of size no more than $\hat{k}$. In our example there are three nodes in the intersection, and two are small.

Lemma 2.4.4 *For any vertex q of G, the following hold:*

1. $\ell_r(q) \leq \ell_s(q)$
2. *If (r,q) is an adjacent packet edge of G,* $\ell_r(q) + \ell_s(q) + |q| = \ell_r(r) + \ell_s(r)$
3. *If (r,q) is a predecessor edge of G,* $\ell_r(q) + \ell_s(q) \geq \ell_r(r) + \ell_s(r)$

Proof: As in the proof of Lemma 2.4.2, we suppose that vertex q is $\mu(i,j)$ in each case.

1. ℓ_r is the number of message units remaining to be received by P_i, while ℓ_s is the number of units remaining to be sent by P_i. Since P_i may never have sent more message units than it has received, $\ell_r(q) \leq \ell_s(q)$.

2. Let (r, q) be the adjacent packet in-edge into q. From Lemma 2.4.2 we have: $f_r(r) + f_s(r) + |\mathrm{p}(r)| = f_r(q) + f_s(q)$. Using identity 2.1—$f_r(q) + f_s(q) + |q| + \ell_r(q) + \ell_s(q) = 2n$—we get $\ell_r(q) + \ell_s(q) + |q| = \ell_r(r) + \ell_s(r)$.

3. We must have $\ell_r(r) \leq \ell_s(r)$ and $\ell_r(q) \leq \ell_s(q)$. We distinguish two cases:

   q is sent by P_i: Then P_{i-1} does not transfer any information to P_i between the time r finishes transmitting and q finishes transmitting. As a result, $\ell_s(r) = \ell_r(q)$: that is, the number of message units P_i has still to receive when q finishes is the same as the number of message units that P_{i-1} needs to send after r is done.

   As a consequence, $\ell_r(r) \leq \ell_s(r) = \ell_r(q) \leq \ell_s(q)$.

   q is received by P_i: The the only transfer of information between P_{i-1} and P_i between the time that r finishes sending and the time that q finishes transmitting is represented by q itself. As a result, $\ell_s(r) = |q| + \ell_r(q)$ (or, in words: the number of message units that P_{i-1} must still send after r completes its transfer is $|q|$ more than the number of message units that P_i must still receive after q has finished sending.)

   Since P_{i-1} was able to send q, though, $\ell_r(r) + |q| \leq \ell_s(r)$ and so $\ell_r(r) \leq \ell_r(q)$. Similarly, after receiving q, $\ell_r(q) + |q| \leq \ell_s(q)$. Thus $\ell_s(r) \leq \ell_s(q)$.

   In either case, $\ell_r(q) + \ell_s(q) \geq \ell_r(r) + \ell_s(r)$.

The lemma holds. □

The only characteristics of f_r and f_s used in the proof of Lemma 2.4.3 were exactly those proved in (the latter two parts of) Lemma 2.4.2. Since Lemma 2.4.4 proves the same characteristics for ℓ_r and ℓ_s, the proof of the loop invariant assertion for the backward search will be essentially identical to that for the forward. As a result, we will only state the lemma.

Lemma 2.4.5 *The loop invariant assertion of algorithm* **SearchBackLink** *holds.*

Moreover, the proof that the path $\mathcal{B}$ traced by algorithm **SearchBackLink** intersects one of the two forward paths $\mathcal{A}$ and $\mathcal{A}'$ traced by algorithm **SearchFwdLink** in a small packet is also basically identical to that of Lemma 2.3.4. The differences are all simply syntactic: here, the vertices are labeled with μ, while in the link bound proof they are labeled λ; and here, the "vertical" edges are either successor or predecessor edges, while in the link bound proof, they are overlapping packet edges. The vast majority of the argument simply invokes the planarity of the underlying digraph: and that, at least, is constant. We again only state the lemma.

Lemma 2.4.6 *$\mathcal{B}$ intersects either $\mathcal{A}$ or $\mathcal{A}'$ in a vertex q where $|q| \leq \hat{k}$.*

Let $\hat{\mathcal{A}}$ be the path (either $\mathcal{A}$ or $\mathcal{A}'$) that contains the vertex of intersection q. Following the example of the link bound section, we label by $\mathcal{A}_1$ the maximal sub-path of $\hat{\mathcal{A}}$ containing $\mu(2,1)$ that does not contain q, and by $\mathcal{B}_1$ the maximal sub-path of $\mathcal{B}$ containing $\mu(m, \nu(m-1)+\nu(m))$ and also not containing q. By the respective loop invariants of algorithms **SearchFwdLink** and **SearchBackLink**:

$$t(\mathcal{A}_1) \geq \left(\left\lceil \frac{f_r(q)}{\hat{k}} \right\rceil + \left\lceil \frac{f_s(q)}{\hat{k}} \right\rceil + i - 2 \right) \beta + \left((i-2)\,\hat{k} + f_r(q) + f_s(q) \right) \tau$$

and

$$t(\mathcal{B}_1) \geq \left(\left\lceil \frac{\ell_r(q)}{\hat{k}} \right\rceil + \left\lceil \frac{\ell_s(q)}{\hat{k}} \right\rceil + m - i \right) \beta + \left((m-i)\,\hat{k} + \ell_r(q) + \ell_s(q) \right) \tau.$$

The path $\mathcal{P} = \mathcal{A}_1 q \mathcal{B}_1$ in G will thus provide the lower bound:

$$\begin{aligned} t(\mathcal{P}) &= t(\mathcal{A}_1) + t(q) + t(\mathcal{B}_1) \\ &\geq t(\mathcal{A}_1) \geq \left(\left\lceil \frac{f_r(q)}{\hat{k}} \right\rceil + \left\lceil \frac{f_s(q)}{\hat{k}} \right\rceil + i - 2 \right) \beta \end{aligned}$$

$$\begin{aligned}
& + \left((i-2)\,\hat{k} + f_r(q) + f_s(q)\right)\tau \\
& + \beta + \hat{k}\tau \\
& + \left(\left\lceil \frac{\ell_r(q)}{\hat{k}} \right\rceil + \left\lceil \frac{\ell_s(q)}{\hat{k}} \right\rceil + m - i\right)\beta \\
& + \left((m-i)\,\hat{k} + \ell_r(q) + \ell_s(q)\right)\tau \\
= & \left(\left\lceil \frac{f_r(q)}{\hat{k}} \right\rceil + \left\lceil \frac{f_s(q)}{\hat{k}} \right\rceil + i - 2 + 1 + \left\lceil \frac{\ell_r(q)}{\hat{k}} \right\rceil + \left\lceil \frac{\ell_s(q)}{\hat{k}} \right\rceil + m - i\right)\beta \\
& + \left((i-2)\,\hat{k} + f_r(q) + f_s(q) + \hat{k} + (m-i)\,\hat{k} + \ell_r(q) + \ell_s(q)\right)\tau \\
\geq & \left(\left\lceil \frac{f_r(q)}{\hat{k}} \right\rceil + \left\lceil \frac{f_s(q)}{\hat{k}} \right\rceil + \left\lceil \frac{\hat{k}}{\hat{k}} \right\rceil + \left\lceil \frac{\ell_r(q)}{\hat{k}} \right\rceil + \left\lceil \frac{\ell_s(q)}{\hat{k}} \right\rceil + m - 2\right)\beta \\
& + \left((m-2)\,\hat{k} + 2n\right)\tau \\
\geq & \left(2\left\lceil \frac{n}{\hat{k}} \right\rceil + m - 2\right)\beta + \left((m-2)\,\hat{k} + 2n\right)\tau \\
= & T_1(n, m, \hat{k}).
\end{aligned}$$

To complete the parallel, we return one last time to our example Figure 2.16 in order to check the long path is indeed longer than the 108 time units required by the pipelined transmission of Figure 2.14. The highlighted path in Figure 2.21 has fifteen packets, which contain a total of 63 message units. The pipeline transmission used $\beta = 5$ and $\tau = 1$: with those values, the highlighted path requires 138 time units.

We have thus shown:

Theorem 2.4.1 *The minimum time for sending a message of size n over m links in the processor bound model is*

$$S_1(n,m) = \min_{1 \leq k \leq n} \left(\left(2\left\lceil \frac{n}{k} \right\rceil + m - 2\right)\beta + ((m-2)\,k + 2n)\,\tau\right).$$

We again provide closed forms:

Corollary 2.4.1

$$\begin{aligned}
\left(\sqrt{(m-2)\,\beta} + \sqrt{2n\tau}\right)^2 & \leq S_1(n,m) \\
S_1(n,m) & < \left(\sqrt{(m-2)\,\beta} + \sqrt{(2n-2)\,\tau}\right)^2 + 2\beta + m\tau
\end{aligned}$$

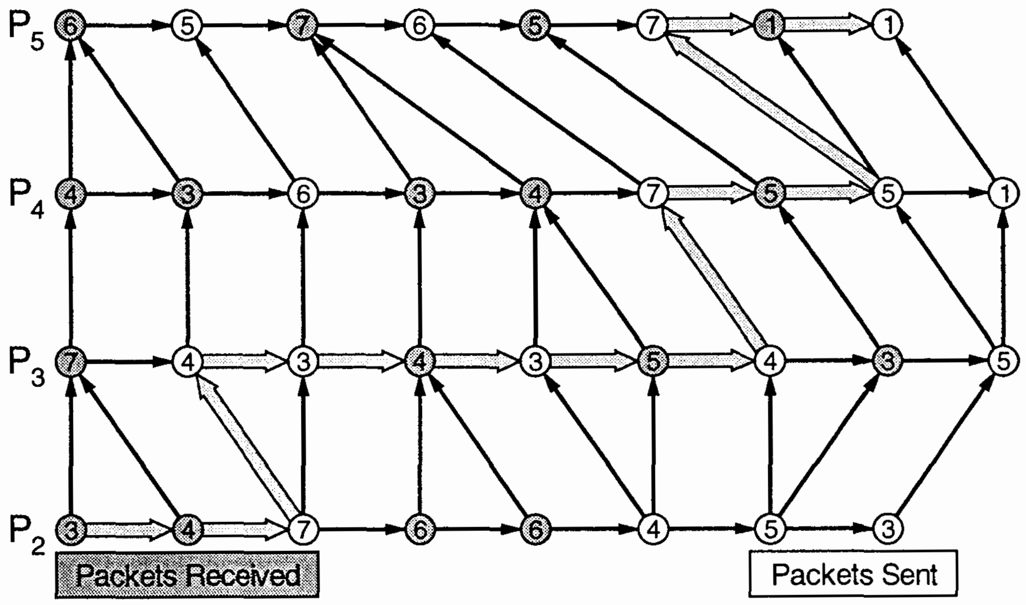


Figure 2.21: The long path found in Figure 2.16.

The appropriate algebra may be found in Appendix A.2.2.

2.5 Sending along multiple paths

We have examined the problem of sending a message of length n over m communication links. Another way to think of the problem is to send a message between two processors at distance m from one another in the path, or unidirectional ring. Assuming the link bound for simplicity, what happens when the topology is more interesting?

The easiest solution to the presence of a general network is to break the problem down into the subproblem already studied, and consider a collection of *disjoint paths* over which the message may be sent. If we wished to send a message of size n in the hypercube Q_m between $\vec{0}$ and $\vec{1}$, for example, we could find m disjoint paths of length m, and thus take no more than $S_*(\lceil n/m \rceil, m)$ time.

This is in fact the approach that Saad and Schultz [22] employed as they examined sending messages in torus networks. They demonstrate that for any two processors P_1 and P_2 in a two dimensional torus, if the distance from P_1 to P_2 is d, there exist four edge disjoint paths between P_1 and P_2 whose maximum length is no more than $d+6$. As a consequence, they argue, sending a message of size n from P_1 to P_2 takes no more than $S_*(\lceil n/4 \rceil, d+6)$ time.

The parameters under which the use of disjoint paths is optimal are not completely clear. In cases, such as the ring, where the only set of paths between two processors are disjoint, a disjoint path solution will be trivially optimal. We present three scenarios, to illustrate the more general case.

Consider sending a message of size n in the ring C_p from P_0 to P_m in minimum time. We denote this minimum time by $S_*(C_p, n, m)$. There are two paths available for use: the "forward path" through P_1 and P_{m-1}, and the "backward" path through P_{p-1} and P_{m+1}. Regardless of how we choose to send the information along those paths, any scheme that gets n message units to P_m will send some n_0 units along the forward path, and $n-n_0$ units along the backward path. From Section 2.3, we know that the minimum time required by this division is simply $\max\{S_*(n_0, m), S_*(n-n_0, p-m)\}$, and that this time is achievable by simply employing the appropriate minimum time pipelines in each direction. As a result, the disjoint paths provide an optimal solution: Observation 2.5.1.

Observation 2.5.1

$$S_*(C_p, n, m) = \min_{0 \le n_0 \le n} \max\{S_*(n_0, m), S_*(n-n_0, p-m)\}$$

A closed form for Observation 2.5.1 can be found in Appendix A.3.1.

Consider sending a message from $P_{0,0}$ to $P_{2,2}$ in the 3×3 grid graph. If we employ two disjoint paths, the time to send a message of size n is $S_*(\lceil n/2 \rceil, 4)$. A more interesting scheme for some parameter k is:

- $P_{0,0}$ divides the message in half, and sends half to $P_{1,0}$ in packets of size $2k$ and the other half to $P_{0,1}$ in packets of size $2k$.

- $P_{0,1}$ divides each packet it receives in half, and sends half to $P_{1,1}$ and half to $P_{0,2}$. $P_{1,0}$ similarly divides the packets it receives in half, and sends the halves to $P_{2,0}$ and $P_{1,1}$.

- $P_{0,2}$ and $P_{2,0}$ simply forward the packets they receive on to $P_{1,2}$ and $P_{2,1}$ respectively.

- $P_{1,1}$ forwards each packet it receives, as well. Anything it gets from $P_{0,1}$ it sends to $P_{1,2}$, and anything from $P_{1,0}$ it sends to $P_{2,1}$.

- $P_{2,1}$ and $P_{1,2}$ receive pairs of packets in lock-step. They amalgamate the pairs, and send a single packet of size $2k$ on to $P_{2,2}$ for each pair they receive.

Rather than taking the

$$\min_{1 \le k \le \lceil n/2 \rceil} \left\{ \left(\left\lceil \frac{n}{2k} \right\rceil + 3 \right) \beta + \left(6k + \left\lceil \frac{n}{2} \right\rceil \right) \tau \right\}$$

time required by the disjoint paths, this scheme only requires

$$\min_{1 \le k \le \lceil n/4 \rceil} \left\{ \left(\left\lceil \frac{n}{4k} \right\rceil + 3 \right) \beta + \left(4k + \left\lceil \frac{n}{2} \right\rceil \right) \tau \right\}$$

time, which is strictly smaller. Use of simple disjoint paths, as opposed to (for lack of a better term) recursive disjoint paths, is decidedly sub-optimal in this case.

For a more ambiguous case, consider sending a message from $P_{0,0}$ to $P_{2,1}$ in the 2×3 grid graph as follows:

- $P_{0,0}$ divides the message into two unequal "halves" — two-fifths and three-fifths of the original, and sends the larger "half" to $P_{1,0}$ in packets of size $3k$ and the smaller "half" to $P_{0,1}$ in packets of size $2k$. Despite the fact that the packets are not the same size, the pairs of packets are sent in lock-step.

- $P_{0,1}$ forwards each packet it receives on to $P_{1,1}$; similarly, $P_{2,0}$ forwards each packet it receives on to $P_{2,1}$.

- $P_{1,0}$ receives packets of size $3k$ from $P_{0,0}$, which it splits into thirds, sending two thirds to $P_{2,0}$ and one third to $P_{1,1}$

- $P_{1,1}$ receives two packets of size k and $2k$ in lock-step. (Yes, indeed: both arrive $4\beta + 4k\tau$ after their respective parent packets were sent by $P_{0,0}$). It amalgamates the packets, and sends a single packet of size $3k$ to $P_{1,2}$ for each pair it receives.

The disjoint path solution in this case requires

$$\min_{1 \le k \le \lceil n/2 \rceil} \left\{ \left(\left\lceil \frac{n}{2k} \right\rceil + 2 \right) \beta + \left(2k + \left\lceil \frac{n}{2} \right\rceil \right) \tau \right\}$$

time, while the other sending scheme takes

$$\min_{1 \le k \le \lceil n/5 \rceil} \left\{ \left(\left\lceil \frac{n}{5k} \right\rceil + 2 \right) \beta + \left(4k + \left\lceil \frac{3n}{5} \right\rceil \right) \tau \right\}$$

time. For appropriate values of n, β and τ, either solution could be better than the other.

One could consider the alternate sending scheme from the last example as using not disjoint paths, but rather overlapping paths that are partially disjoint. Essentially, there are three paths used: $P_{0,0}P_{0,1}P_{1,1}P_{2,1}$ and $P_{0,0}P_{1,0}P_{2,0}P_{2,1}$ over which a pipeline in packets of size $2k$ is sent, and $P_{0,0}P_{1,0}P_{1,1}P_{2,1}$ which uses a pipeline in packets of size k. In fact, for appropriate values of n, β and τ, an additional pipeline of some small packet size along the path $P_{0,0}P_{0,1}P_{1,1}P_{1,0}P_{2,0}P_{2,1}$ would improve the time further, since that path is underutilized in comparison to the other links.

While not likely sufficient to obtain a lower bound, an approach that might bear investigation is the notion of *contracting* subgraphs. The 3×3 grid widget, for example, could be considered a "path of length four" over which τ has three-eighths its normal value. The difficulty with such an approach, of course, is trying to deal with the strange boundary conditions. The three central nodes of the path are simply fictions; they won't be legal targets for a broadcast, for example. In addition, the "path of length four" will behave peculiarly when fed a packet whose length is not divisible by four. These difficulties are not insurmountable, but would require a great deal of care to treat accurately.

In short summary, multiple paths and their treatment could bear a great deal more investigation.

Chapter 3

Broadcasting

3.1 The problem

The general *broadcast* problem is fairly simply expressed: given an interconnection network with some topology and an identified processor within that network, find a communication scheme that will allow the processor to send a piece of information to all its peers in the minimum time. The broadcast problem may also cast as a construction problem: given a time bound, what topologies allow all (or some, or one) of their processors to broadcast in time no larger than the bound. We will not consider the latter sense, but refer the interested reader to [3], [19] or even [20].

Despite its simplicity of specification, broadcasting (in the first sense) has proven to be a rich area for research. Johnson and Garey showed that the problem of calculating the time required to broadcast from a specific processor in an arbitrary graph is $\mathcal{NP}$-complete. Authors have examined (and this is only a sampling) the time required to broadcast: in bounded degree graphs [18]; in the presence of faults [12]; in specific topologies, such as the hypercube [16] or de Bruijn graph [2]; or using circuit switched communications [21] rather than the store-and-forward we employ.

The limiting topologies we will consider for the broadcast problem are the complete graph, in which each processor is connected to every other, and the ring: the minimal processor-transitive connected graph. We begin with the ring.

3.2 Broadcasting in the ring

We consider rings with some number p processors P_0 to P_{p-1} where each processor P_i is connected only to P_{i-1} and P_{i+1} (processor indices taken modulo p.) We will consider rings in two flavours: bidirectional, in which processor P_i may both send or receive a message from either P_{i-1} or P_{i+1}, and unidirectional, where processor P_i receives messages from P_{i-1} and sends to P_{i+1}.

Clearly, the results of Chapter 2 apply immediately to broadcasting in the unidirectional or "oriented" ring. In a unidirectional ring of p processors if P_0 originates a broadcast, it must send a message to P_{p-1}; this informs all other processors along the way.

Observation 3.2.1 *For a processor in a unidirectional ring with p processors to broadcast a message of size n to all other processors, $S(n, p-1)$ time is necessary and sufficient. That is,*

$$b_*(\vec{C}_p, n) = S_*(n, p-1)$$

and

$$b_1(\vec{C}_p, n) = S_1(n, p-1).$$

The interesting ring is the bidirectional one. Fraigniaud and Lazard [8] enumerate the following bounds for us:

$$\begin{aligned}
b_{F*}(C_p, n) &\leq S_*(\left\lceil \frac{n}{2} \right\rceil, p-1) \\
b_{F1}(C_{2m}, n) &\leq S_*(n, 2m-1) \\
b_{F1}(C_{2m+1}, n) &\leq S_*(\left\lceil \frac{3n}{2} \right\rceil, 4m) \\
b_{H*}(C_p, n) &\leq S_*(n, \left\lfloor \frac{p}{2} \right\rfloor) \\
b_{H1}(C_p, n) &\leq S_1(n, \left\lfloor \frac{p}{2} \right\rfloor)
\end{aligned}$$

In contrast, for the full duplex cases we will show:

$$b_{F*}(C_{2m}, n) = S_*(\lceil n/2 \rceil, m)$$

$$\begin{aligned}
b_{F*}(C_{2m+1},n) &= \min_{1\le k\le n} T_*(n-\left\lfloor\frac{n+k}{2}\right\rfloor, m+1, k)\\
b_{F1}(C_{2m},n) &\le S_*(n,m)\\
b_{F1}(C_{2m+1},n) &\le \min_{1\le k\le n} T_*(n+k\left\lceil\frac{k(m-1)+n}{2mk}\right\rceil, m, k).
\end{aligned}$$

In each case, the effective depth of the pipelined scheme has been at least halved. And in the processor bound even ring, the depth of the pipeline has been reduced to one quarter of the original, while the amount of information sent has also decreased.

The half-duplex bounds do not show quite the same improvement:

$$\begin{aligned}
b_{H*}(C_{2m},n) &\le \min_{0\le k\le n} \max\left\{T_*(\left\lceil\frac{n}{2}\right\rceil, m, k), T_*(n, m-1, k)\right\}\\
b_{H*}(C_{2m+1},n) &\le \min_{0\le k\le n} \max\{T_*(n,m-1,k), t_0\} \quad \text{where}\\
t_0 &= \min_{0\le n_0\le n} \max\{T_*(n-n_0,m,k), T_1(n_0,m+2,k)\}\\
b_{H1}(C_{2m},n) &\le \min_{0\le k\le n} \max\{T_*(n,m,k), T_1(n,m-1,k)\}\\
b_{H1}(C_{2m+1},n) &\le \min_{0\le k\le n} \max\{T_1(n,m-1,k), t_0'\} \quad \text{where}\\
t_0' &= \min_{0\le n_0\le n} \max\left\{\begin{array}{l} T_1(n_0,m,k)\\ T_*(n-n_0,\ m-2+3\lceil(n-n_0)/k\rceil + \\ \qquad \left\lceil\frac{1}{m}(2\lceil(n-n_0)/k\rceil+1)\right\rceil \quad ,k)\end{array}\right.
\end{aligned}$$

The link bound even cycle is a strict improvement over the original. All the other bounds will perform identically to the originals in the worst case: when $n\tau \ll m\beta$.

As in the previous section, we look first at the link bound case.

3.2.1 The link bound full duplex ring

The ring with an even number of processors is much easier to work with than the odd ring. In the even ring, the broadcast originator P_0 has an antipodal processor P_m that must receive the entire message. Since the paths $P_0, P_1, \ldots, P_m$ and $P_0, P_{2m-1},$-$P_{2m-2}, \ldots, P_m$ are completely independent in the link bound model, a simple lower bound on the broadcast is

$$\min_{1\le n_0\le n} \{\max(S_*(n_0,m), S_*(n-n_0,m))\}$$

resulting from sending n_0 units along one path, and the remainder along the other. The next theorem demonstrates a matching lower bound.

Theorem 3.2.1

$$b_{F*}(C_{2m}, n) = S_*(\left\lceil \frac{n}{2} \right\rceil, m)$$

We prove the theorem by a series of relatively uninteresting lemmas. First, a lemma describing the properties of the function $T_*(n,m,k)$: the minimum time required to send n units over m links in "packets of size k".

Lemma 3.2.1 *Let n_0, n_1, m, and k be positive integers.*

1. *If $k < n_0$, then $T_*(n_0,m,k) = T_*(n_0+k,m-1,k) = T_*(n_0-k,m+1,k)$.*
2. *If $k \leq n_0$ and $k \leq n_1$, then $T_*(n_0,m,k) < T_*(n_1,m,k)$ if and only if $n_0 < n_1$.*

Proof: We prove each point in turn.

1. We invoke the definition of $T_*(n,m,k)$, and apply a bit of algebra:

$$\begin{aligned} T_*(n_0,m,k) &= (\lceil n_0/k \rceil + m - 1)\,\beta + ((m-1)\,k + n_0)\,\tau \\ &= (\lceil n_0/k + 1 \rceil + m - 2)\,\beta + ((m-2)\,k + n_0 + k)\,\tau \\ &= (\lceil (n_0+k)/k \rceil + m - 2)\,\beta + ((m-2)\,k + n_0 + k)\,\tau \\ &= T_*(n_0+k, m-1, k) \end{aligned}$$

and

$$\begin{aligned} T_*(n_0,m,k) &= (\lceil n_0/k - 1 \rceil + m)\,\beta + (mk - k + n_0)\,\tau \\ &= (\lceil (n_0-k)/k \rceil + m)\,\beta + (mk + n_0 - k)\,\tau \\ &= T_*(n_0-k, m+1, k). \end{aligned}$$

2. If we pretend that $T_*(n,m,k)$ takes real parameters, and take the partial derivative with respect to n, we find $\frac{\partial}{\partial n}T_*(n,m,k) = \tau$ almost everywhere, and that its value is unbounded and positive on the remaining points. Since the partial derivative is always positive, $T_*(n,m,k)$ is monotone increasing with respect to n.

Both points are validated: the lemma holds. □

The previous lemma demonstrates that for fixed values of m and k, $T_*(n,m,k)$ is monotone increasing with n. We prove a similar result for the function $S_*(n,m)$: the minimum time required to send n units over m links.

Lemma 3.2.2 *For any positive integers n_0, n_1, and m,*

$$S_*(n_0,m) < S_*(n_1,m)$$

if and only if $n_0 < n_1$.

Proof: Let $n_0 < n_1$. There exists a value k such that $S_*(n_1,m) = T_*(n_1,m,k)$, by the definition of $S_*(n,m)$. The second point of Lemma 3.2.1 gives, $S_*(n_0,m) \leq T_*(n_0,m,k) < T_*(n_1,m,k) = S_*(n_1,m)$. Thus, $S_*(n_0,m) < S_*(n_1,m)$.

Suppose $S_*(n_0,m) < S_*(n_1,m)$. There exists a value k' such that

$$S_*(n_0,m) = T_*(n_0,m,k').$$

By the definition of $S_*(n,m)$,

$$S_*(n_0,m) = T_*(n_0,m,k') < S_*(n_1,m) \leq T_*(n_1,m,k'),$$

and by Lemma 3.2.1 , $n_0 < n_1$. □

Finally, a lemma that indicates how one might employ these characteristics of $T_*(n,m,k)$ and $S_*(n,m)$ to prove Theorem 3.2.1

Lemma 3.2.3 *Given an optimal transmission that sends n units from P_0 to P_m in packets of size k, the transmission may be easily extended to append suffixes to the message, and send, in exactly the same amount of time, as follows:*

- *$n+k$ units to P_{m-1}*
- *$n+2k$ units to P_{m-2}*
- *and in general, $n+ik$ units to P_{m-i} for $1 \le i \le m-1$*

and in addition send prefixes of the message as follows:

- *the first $n-k$ units to P_{m+1}*
- *the first $n-2k$ units to P_{m+2}*
- *and in general, the first $n-jk$ units to P_{m+j} for $1 \le j \le \lfloor (n-1)/k \rfloor$*

Proof: The only changes required to the standard pipeline scheme are with the originator P_0, and with the receiver P_m.

Processor P_0, instead of terminating the transmission after n units are sent, continues to send information in packets of size k.

Processor P_m sends the information in each packet it receives on down the line, rather than simply listening to P_{m-1}.

By the first point of Lemma 3.2.1, we see that it is sufficient to consider $m = 1$ in the statement of the lemma.

Let $k > 0$. We prove by induction on n. If $n < k$, then the statement of the lemma is null: true by default.

Assume that the lemma holds for all $n < N$. Consider a transmission of N units in packets of size k over a single link. By our standard pipelined scheme, we send the "odd" sized packet last: a packet q of size $N + k - k\lceil N/k \rceil$. Consider the state of the transmission just prior to the time that P_0 sends q.

By induction, at time $(\lceil N/k \rceil - 1)\beta + k(\lceil N/k \rceil - 1)\tau$ the transmission scheme has sent:

- $k(\lceil N/k \rceil - 1)$ units to P_1
- $k(\lceil N/k \rceil - 2)$ units to P_2
- and in general, $k(\lceil N/k \rceil - i)$ units to P_i for $0 \le i < \lceil N/k \rceil$

If at this time, each of these processors—that is, P_0 to $P_{\lceil N/k \rceil -1}$—sends a packet of $N + k - k\lceil N/k \rceil$ units along to its neighbour, the scheme will have sent:

- N units to P_1
- $N - k$ units to P_2
- and in general, $N - ik$ units to P_{i+1} for $0 \leq i \leq \lceil N/k \rceil - 1$

thus satisfying the lemma for $n = N$. By the principle of mathematical induction, the lemma holds, for some particular $k > 0$ for all $n > 0$. □

All that remains to demonstrate the proof of Theorem 3.2.1 are the following two observations.

1. Since according to Lemma 3.2.2, $S_*(n, m)$ is monotone increasing with n,
$$S_*(\left\lceil \frac{n}{2} \right\rceil, m) = \min_{n_0 < n} \max \{S_*(n - n_0, m), S_*(n_0, m)\}.$$
It follows that $S_*(\lceil n/2 \rceil, m)$ is a lower bound on the time required to send n units to P_m along both paths, and therefore necessarily a lower bound on the time required to broadcast n units to all processors.

2. Since the paths
$$P_0, P_1, P_2, \ldots, P_{2m-1}$$
and
$$P_0, P_{2m-1}, P_{2m-2}, \ldots, P_1$$
are completely independent of one another, we can assign different schemes to the two paths without any possible interference. If we employ an "extended" pipelined transmission (after Lemma 3.2.3) that sends $\lceil n/2 \rceil$ units along the first path to P_m, and another extended transmission (with reversed message order) that sends $\lfloor n/2 \rfloor$ units to P_m along the second path, then in time $S_*(\lceil n/2 \rceil, m)$ each processor will have received the n message units. If we additionally constrain P_0 to send no more than n units in each direction, no processor will receive the same information from both directions!

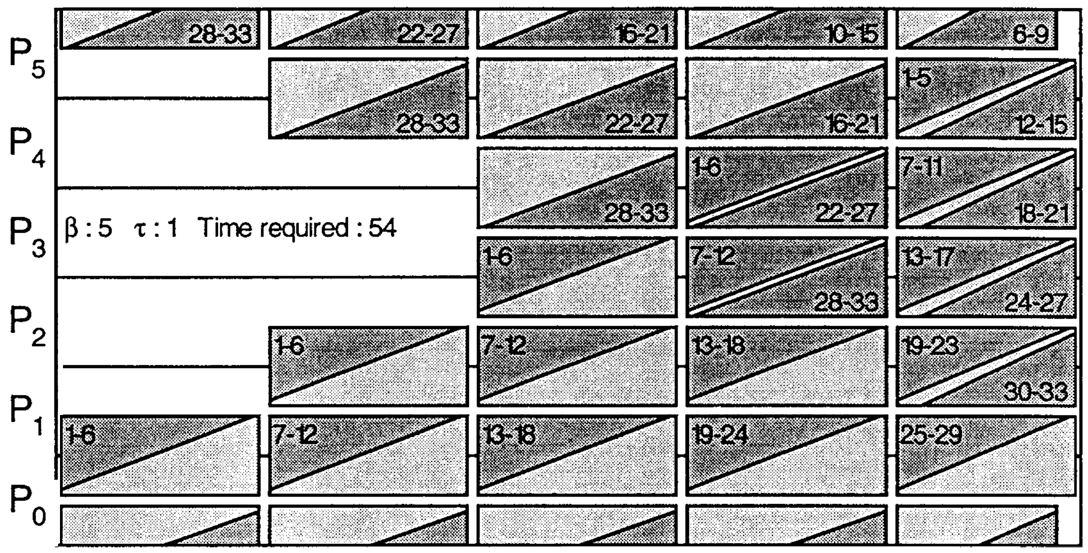


Figure 3.1: Example: Broadcasting a message of size 33 in C_6

As an example, consider Figure 3.1. For the values $\beta = 5$ and $\tau = 1$, the optimal value for k is 6.

We proceed immediately to the odd case.

Theorem 3.2.2

$$b_{F*}(C_{2m-1}, n) = \min_{1 \leq k \leq n} T_*(n - \left\lfloor \frac{n+k}{2} \right\rfloor, m, k)$$

We prove the upper and lower bounds separately. First, we examine two lemmas that together demonstrate the upper bound.

Lemma 3.2.4 *To broadcast a message of n units to a bidirectional ring of $2m-1$ processors,*

$$\min_{\substack{0 \leq n_0 \leq n \\ 0 \leq k \leq n}} \{\max \{T_*(n_0, m-1, k), T_*(n - n_0, m, k)\}\}$$

time is sufficient.

Proof: This upper bound should be immediate: for some values of n_0 and k, send an extended pipelined scheme along the path of increasing index that will send n_0 units to P_{m-1} in packets of size k, and send $n - n_0$ units to P_{m-1} along the path of decreasing index (modulo $2m - 1$) in packets of the same size k. As in the proof of Theorem 3.2.1, this will provide at least n units to each processor at time $\max\{T_*(n_0, m-1, k), T_*(n-n_0, m, k)\}$. □

Lemma 3.2.5 *Lemma 3.2.4 provides the desired upper bound for Theorem 3.2.2. Specifically, for positive integers n, m, and $1 \leq k \leq n$,*

$$\min_{0 \leq n_0 \leq n} \max\{T_*(n_0, m-1, k), T_*(n-n_0, m, k)\} = T_*\left(n - \left\lfloor \frac{n+k}{2} \right\rfloor, m, k\right).$$

Proof: By Lemma 3.2.1, we have $T_*(n_0, m-1, k) = T_*(n_0 - k, m, k)$. We are therefore interested in minimizing:

$$\min_{0 \leq n_0 \leq n} \{\max\{T_*(n_0 - k, m, k), T_*(n - n_0, m, k)\}\},$$

which is the same as minimizing $|(n - n_0) - (n_0 - k)|$. This occurs when either $n_0 = \lceil (n+k)/2 \rceil$ or $n_0 = \lfloor (n+k)/2 \rfloor$.

Since $\lceil (n+k)/2 \rceil - k = n - \lfloor (n+k)/2 \rfloor$ and $\lfloor (n+k)/2 \rfloor - k = n - \lceil (n+k)/2 \rceil$, it is sufficient to consider only $T_*(n - \lceil (n+k)/2 \rceil, m, k)$ and $T_*(n - \lfloor (n+k)/2 \rfloor, m, k)$ in finding the maximum. Applying Lemma 3.2.1 again,

$$T_*\left(n - \left\lceil \frac{n+k}{2} \right\rceil, m, k\right) \leq T_*\left(n - \left\lfloor \frac{n+k}{2} \right\rfloor, m, k\right),$$

so

$$\min_{0 \leq n_0 \leq n} \max\{T_*(n_0, m-1, k), T_*(n-n_0, m, k)\} = T_*\left(n - \left\lfloor \frac{n+k}{2} \right\rfloor, m, k\right).$$

□

For purpose of illustration, consider the broadcast scheme in Figure 3.2. With $\beta = 5$ and $\tau = 1$, the smallest upper bound that can be achieved is with $k = 7$.

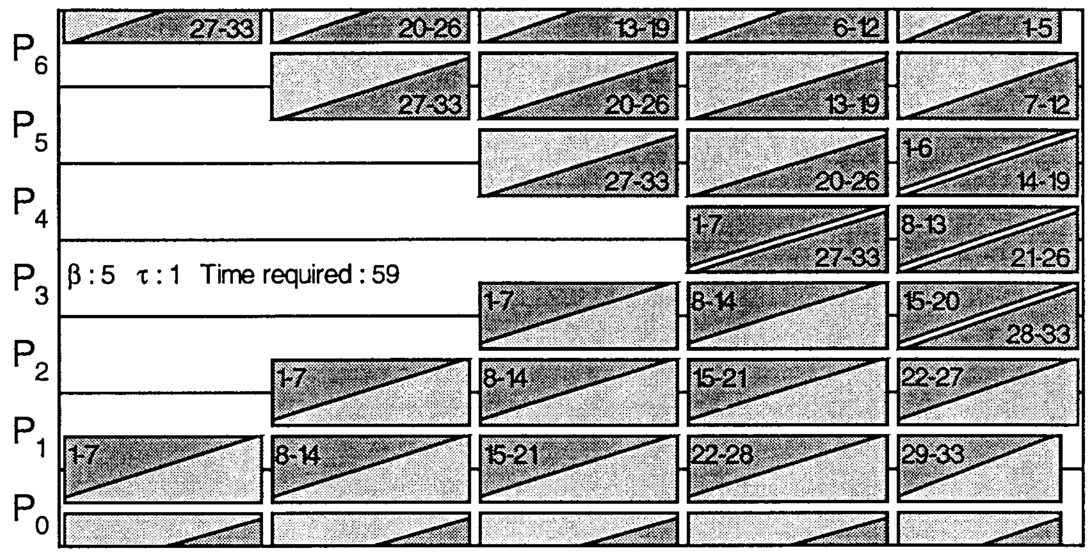


Figure 3.2: Example: Broadcasting a message of size 33 in C_7

Processor P_0 sends 13 message units to P_4 in the same time that it sends 20 message units to P_3.

Rather than dealing with the lower bound directly, we resort to a more primitive technique: that of Chapter 2. Consider the problem of sending a single message along a path from P_0 to P_m in minimum time so that the total of the units received by P_m and the units received by P_{m-1} is n.

Lemma 3.2.6 *Let n and m be positive integers. Let M be a message of length n. In a path of processors $P_0P_1 \ldots P_m$, the minimal time for P_0 to simultaneously send the first n_0 units of M to P_{m-1} and the last $n - n_0$ units of M to P_m (where n_0 is some value $0 \leq n_0 \leq n$ that makes this time as small as possible) is at least*

$$\min_{1 \leq k \leq n} \left\{ T_* \left(n - \left\lfloor \frac{n+k}{2} \right\rfloor, m, k \right) \right\}.$$

Proof: We extend the notion of the *packet content digraph* that we saw in Section 2.3. First, we are now considering transmissions that send both in the "forward" direction

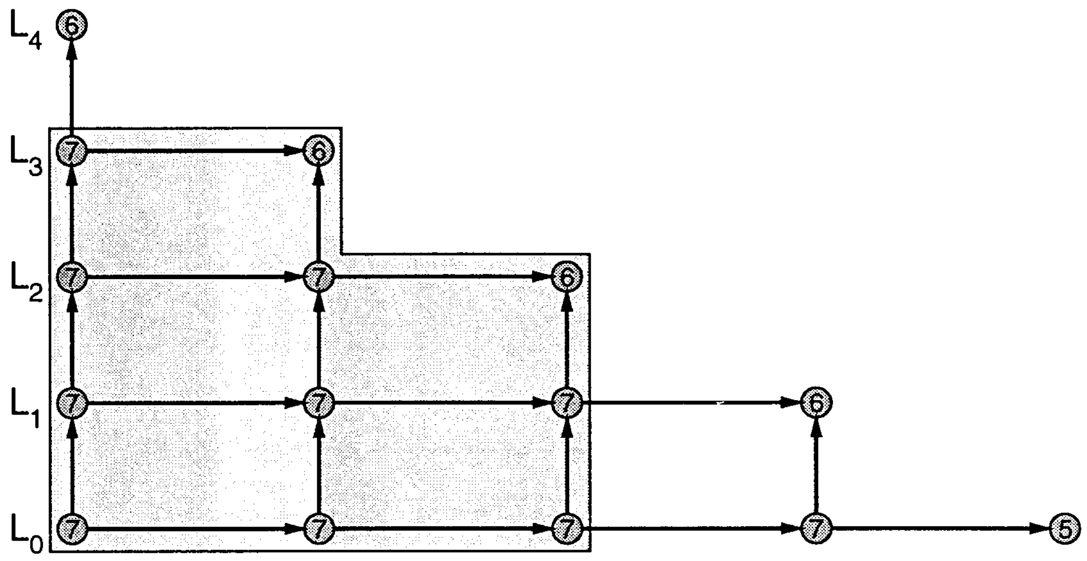


Figure 3.3: Example: The digraph of the forward component of Figure 3.2

(from P_i to $P_{(i+1)\bmod p}$) and in the "backward" direction (from P_i to $P_{(i+1)\bmod p}$). The link-bound model makes these two directions independent: we thus imagine that we have two separate unidirectional transmissions (ν_f, λ_f) and (ν_b, λ_b).

Consider the packet content digraph of a forward transmission (ν_f, λ_f) from P_0 that simultaneously sends prefixes of n_0 units to P_{m-1} and $n - n_0$ units to P_m. The standard pipeline will satisfy the requirements for $n_0 = \lfloor n/2 \rfloor$ in time $S_*(\lceil n/2 \rceil, m)$: we're only interested in better schemes; as a result, $n_0 \geq \lceil n/2 \rceil$.

We're particularly interested in the long paths that we can find that end at either $\lambda_f(m-1, \nu_f(m-1))$ or $\lambda_f(m, \nu_f(m))$. We delete from the packet content digraph all nodes that do not have a path to either of these nodes. As an example, consider Figure 3.3. The highlighted portion of the digraph is all that remains after this deletion.

We extend (slightly) the specification of algorithm **SearchForward** in Section 2.4 to deal with the fact that there are packets $\lambda(m-1, j)$ that have no overlapping packet neighbour. We would still wish to return either m or $m+1$ on leaving packet $\lambda(m, \nu(m))$ and to return $m-1$ on leaving $\lambda(m-1, \nu(m-1))$—as long as

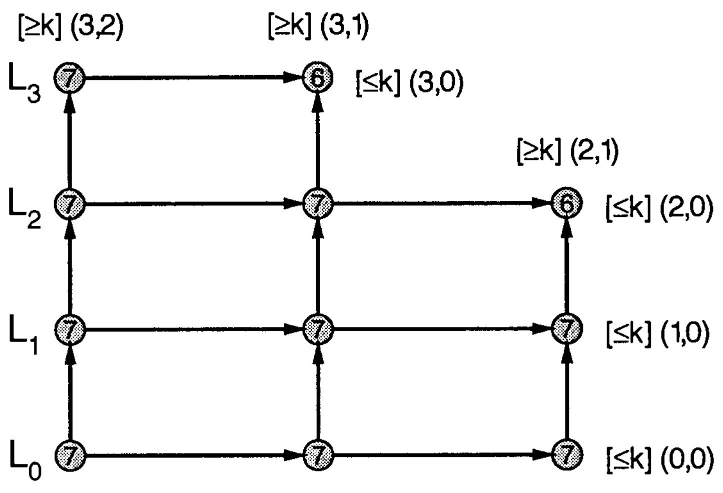


Figure 3.4: Example: The digraph of the forward component of Figure 3.2

$|\lambda(m-1, \nu(m-1))|$ is no larger than the input parameter k. Accordingly, these new returns must have values between $m-1$ and m. We try to make the presence of these extra returns obvious by having the algorithm return a pair of integers, rather than just one:

- If $|\lambda(i,j)| \geq k$, return $(i, \nu(i)+1-j)$
- If $|\lambda(i,j)| \leq k$, return $(i, 0)$

We use lexicographic ordering on these return pairs.

Continuing with our illustration, Figure 3.4 contains the relevant portion of the digraph we have been considering. The possible return codes are marked adjacent the nodes on the upper and right hand boundaries.

As in chapter 2, there must exist some integer k so that **SearchForward**(ν, λ, k) has returns ρ and ρ, where $\rho \leq (m-1, 0) < \rho'$. We examine two cases:

(The easy case.) Suppose $\rho' \geq (m, 0)$. Then the respective paths $\mathcal{A}$ and $\mathcal{A}'$ traced by **SearchForward** in producing the returns ρ and ρ' bound the region containing both $\lambda(m-1, \nu(m-1))$ and $\lambda(m, \nu(m))$

We can run **SearchBackward** from both of these packets, and thus produce two long paths: the one from $\lambda(m, \nu(m))$ requiring $T_*(n_0, m-1, k)$ time, and the

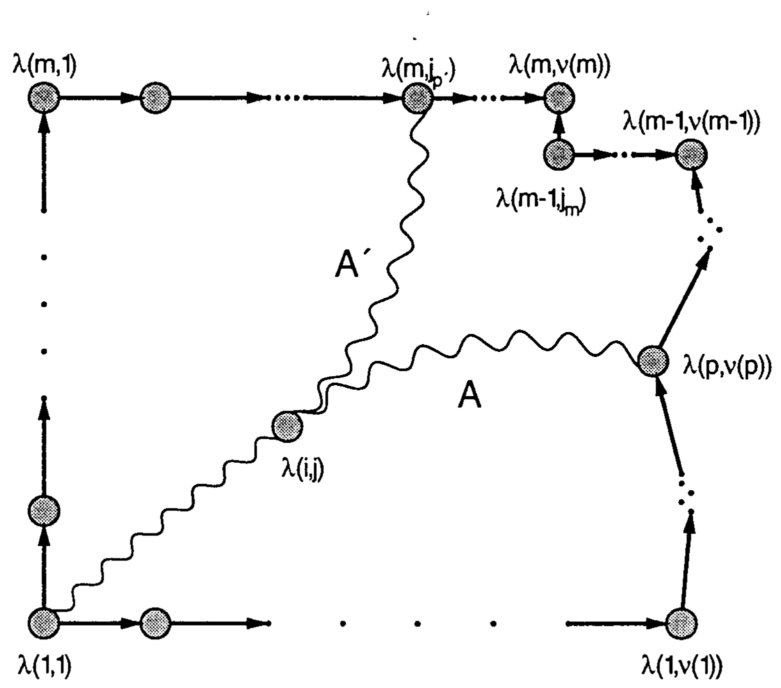


Figure 3.5: Easy case: Two paths traced by **SearchForward**(ν, λ, k)

other requiring $T_*(n - n_0, m, k)$ time. Since the time required for any path in the transmission is a lower bound on the time required for the transmission as a whole, this dual transmission requires time at least

$$\min_{1 \le k \le n} \{\max\{T_*(n_0, m-1, k), T_*(n - n_0, m, k)\}\}.$$

(The hard case.) We have $m - 1 \le \rho' < m$. Again let $\mathcal{A}$ and $\mathcal{A}'$ be the paths that were traced to generate the returns ρ and ρ' respectively. Since $\lambda(m-1, \nu(m-1))$ is contained in the region bounded by $\mathcal{A}$ and $\mathcal{A}'$, we can find a long path that requires $T_*(n_0, m-1, k)$ time by running *SearchBackward* from $\lambda(m-1, \nu(m-1))$. To complete the proof, we will show $t(\mathcal{A}') \ge T_*(n - n_0, m, k)$.

Let $\lambda(m-1, j_m)$ be the rightmost (in the embedding) packet that has $\lambda(m, \nu(m))$ as an overlapping packet neighbour, and let $\lambda(m-1, j_{p'})$ be the last packet in $\mathcal{A}'$ corresponding to the return $\rho' = (m-1, \nu(m-1) + 1 - p')$. We note that both

$$\sum_{\ell=1}^{j_m} |\lambda(m-1, \ell)| \ge n - n_0$$

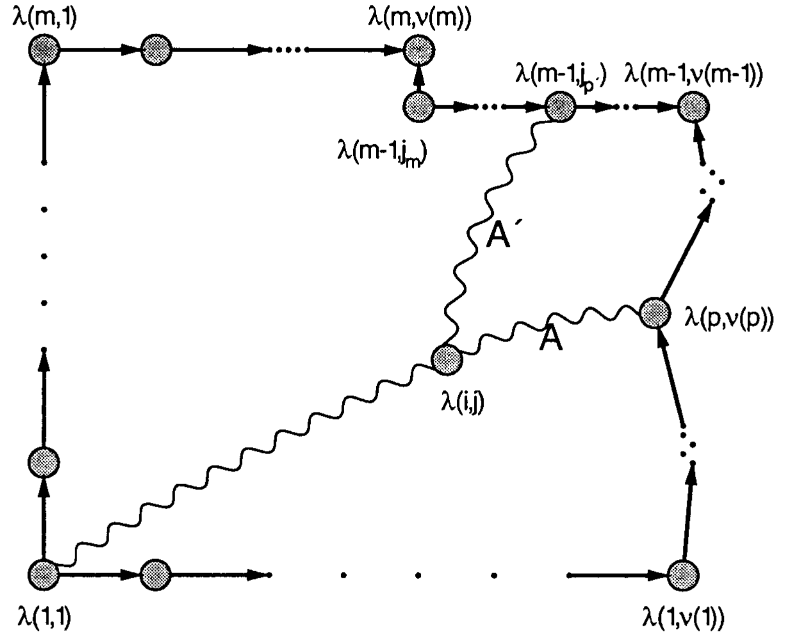


Figure 3.6: Hard case: Two paths traced by **SearchForward**(ν, λ, k)

and $j_{p'} > j_m$.

The assertion of *SearchForward* provides that

$$\begin{aligned}
t(\mathcal{A}) &\geq T_*\left(\sum_{\ell=1}^{j_{p'}-1} |\lambda(i,\ell)|, m-1, k\right) + t(\lambda(m-1, j_{p'})) \\
&\geq T_*\left(\sum_{\ell=1}^{j_m} |\lambda(i,\ell)|, m-1, k\right) + \beta + k\tau \\
&\geq T_*(n-n_0, m-1, k) + \beta + k\tau \\
&= T_*(n-n_0, m, k)
\end{aligned}$$

Since this case also has two long paths of lengths $T_*(n_0, m-1, k)$ and $T_*(n-n_0, m, k)$, it must be the case that any transmission from P_0 that sends prefixes of n_0 units to P_m and $n-n_0$ units to P_{m+1} requires time at least

$$\min_{\substack{0 \leq n_0 \leq n \\ 0 \leq k \leq n}} \{\max\{T_*(n_0, m-1, k), T_*(n-n_0, m, k)\}\}.$$

By Lemma 3.2.5, we see that this bound is exactly

$$\min_{1 \le k \le n} T_* \left(n - \left\lfloor \frac{n+k}{2} \right\rfloor, m, k \right).$$

□

The lower bound for the proof of Theorem 3.2.2 follows as a corollary.

Corollary 3.2.1 *To broadcast a message of n units to a bidirectional ring of $2m-1$ processors,*

$$\min_{1 \le k \le n} T_* \left(n - \left\lfloor \frac{n+k}{2} \right\rfloor, m, k \right)$$

time is necessary.

Proof: Let M be a message of n units. Use a minimum time extended pipeline scheme that sends the first $\lfloor (n+k)/2 \rfloor$ units of M to P_{m-1} and the first $n - \lfloor (n+k)/2 \rfloor$ units of M to P_m in the "forward" direction of a bidirectional ring of $2m-1$ processors. In addition, send a similar scheme in the "backward" direction: the last $\lfloor (n+k)/2 \rfloor$ units of M to P_m and the last $n - \lfloor (n+k)/2 \rfloor$ units of M to P_{m-1}.

This composite scheme will necessarily be a minimum scheme that sends M to both P_m and P_{m-1} from P_0. Since it also sends M to all other processors in the same time, this scheme will also provide a lower bound for the problem of broadcasting n units to each processor in a bidirectional ring of $2m-1$ processors. □

3.2.2 The processor bound full duplex ring

The processor bound case suffers the same limitations as the link bound case: specifically, that the even and odd cases are quite different, and require separate analysis. As before, we turn our attention first to the even ring with $p = 2m$ processors.

A broadcast scheme that gives a good upper bound is reasonably simple. We employ a synchronous scheme that sends packets of size k and employs $r = m - 1 + \lceil n/k \rceil$ rounds. During odd numbered rounds, each processor P_{2i} communicates with P_{2i+1}; in even rounds P_{2i} communicates with P_{2i-1}—all indices (as usual) modulo p.

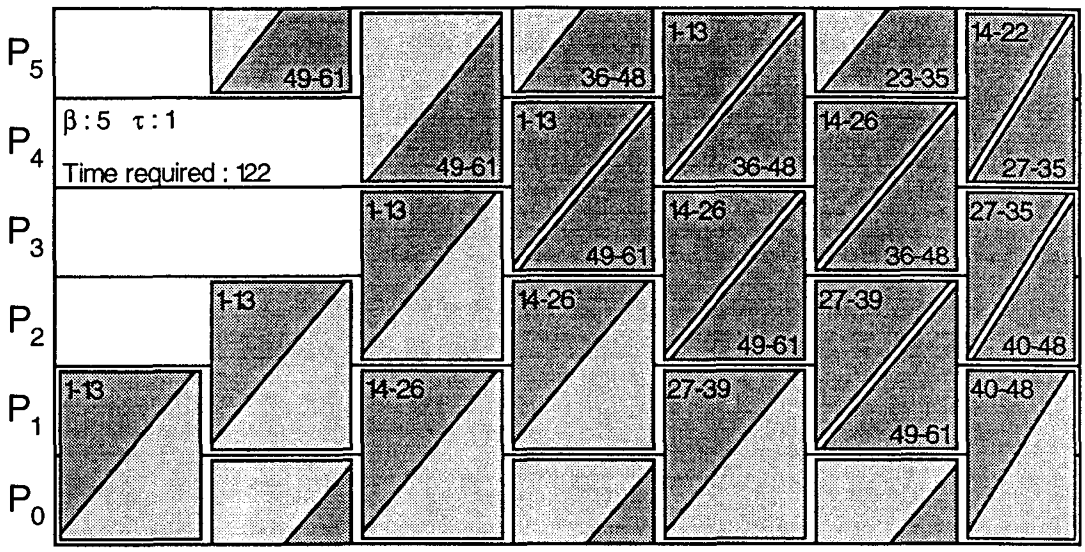


Figure 3.7: Example: Sending a message of size 61 in C_6.

Processor P_0 sends packets beginning at the front of the message to P_1, and beginning at the back of the message to P_{p-1}; in each round, a processor other than P_0 will simply send whatever it received in the previous round. If a processor has either seen no packets, or seen nothing in the previous round, it does not send anything. And as a wrinkle, only $n + k - k\lceil n/k \rceil$ units are sent in the last round.

Consider the example of Figure 3.7. For the values $\beta = 5$ and $\tau = 1$, the optimal value for k is 13. The first 6 rounds are all of size 13, while the last round is only size 9.

Proposition 3.2.1 *The given broadcast scheme provides an upper bound of:*

$$b_{F1}(C_{2m}, n) \leq S_*(n, m).$$

Proof: Consider P_m (equivalently, P_{m+1}): it will begin receiving its i^{th} packet at time $(m - 2 + i)(\beta + k\tau)$. Another processor P_{m-j} (equivalently, P_{m+1+j}) will start receiving its first processor at time $(m - 1 - j)(\beta + k\tau)$, and initially receive only one packet every other round. Processor P_{m-j} (P_{m+1+j}) will start to receive its $(j+1)^{\text{th}}$ packet at time $(m - 1 - j + 2j)(\beta + k\tau)$—the same time that P_m (P_{m+1}) begins to receive its $(j+1)^{\text{th}}$ packet. What's more, P_{m-j} (P_{m+1+j}) receives its first packet from the other direction immediately thereafter: beginning at time $(m+j)(\beta+k\tau)$. As a consequence, for any i and j, $1 \le i \le r$ and $1 \le j < 2m$, processor P_j will begin to receive its i^{th} packet no later than P_m begins to receive its i^{th} packet. So all processors will have received $\lceil n/k \rceil$ packets no later than P_m; the time bound follows.

It remains to show that each processor receives the entire message. Suppose a processor P_j receives ℓ packets from P_0 via P_1 and the remaining $\lceil n/k \rceil - \ell$ packets from P_0 via P_{2m-1}. There are two cases:

1. The the last packet P_j receives is via P_1.

   Then P_j receives units
   $$b_0 b_1 \ldots b_{(n-(\lceil n/k \rceil - \ell)k)}$$
   via P_1, and units
   $$b_n b_{n-1} \ldots b_{(n+1-(\lceil n/k \rceil - \ell)k)}$$
   via P_{2m-1}.

2. Alternatively, P_j gets its last packet via P_{2m-1}.

   Then P_j gets units $b_0 b_1 \ldots b_{k\ell}$ via P_1, and units $b_n b_{n-1} \ldots b_p$ via P_{2m-1}, where
   $$p = n + 1 - (\lceil n/k \rceil - \ell - 1)\,k - (n - (\lceil n/k \rceil - 1)\,k) = k\ell + 1.$$

In either case, P_j receives exactly the units in the message. The proposition follows.

□

The odd case is only a bit more involved. Suppose we're interested in broadcasting to a ring of $2m+1$ processors. We again employ packets of size k; have P_0 send packets beginning at the front of the message to P_1, and in reverse order from the end of the message to P_{2m}. In the first round, round 1, we have processor P_0 send to P_1 while P_{2m} is idle. And in general, we design the sending pattern for round j based on letting processor P_{2m+1-j} be idle: P_{2i-j} and P_{2i-1-j} (for $i = 1 \ldots m$) will exchange a packet. As an example, consider Figure 3.8.

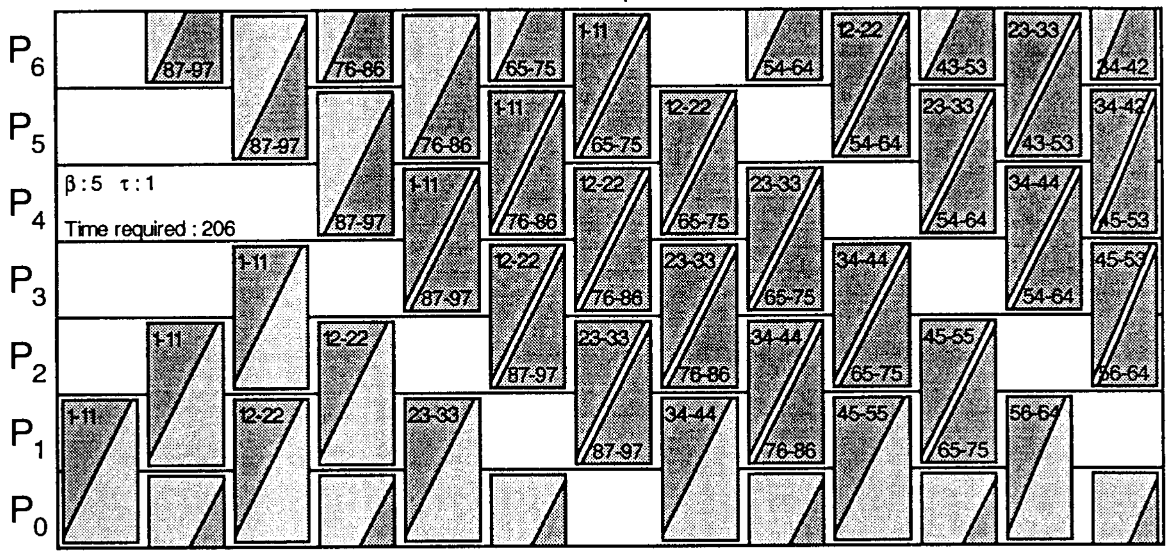


Figure 3.8: Example: Sending a message of size 97 in C_7.

Proposition 3.2.2 *The given broadcast scheme provides an upper bound of*

$$b_{F1}(C_{2m+1}, n) \leq \min_{1 \leq k \leq n} T_*(n + k\left\lceil \frac{k(m-1)+n}{2mk} \right\rceil, m, k).$$

Proof: Processor P_{m+1} begins receiving its first message at time $m(\beta + k\tau)$, and continues to receive a sub-message each round until it is next idle, in round $3m+1$. In much the same way as we argued for the even case, we note that for $1 \leq j \leq m$, P_{m+1-j} (P_{m+1+j}) will both begin to receive its $(j+1)^{\text{th}}$ sub-message at the same time that P_{m+1} starts to receive its $(j+1)^{\text{th}}$, and will receive its i^{th} sub-message (where $i \leq j$) no later than P_{m+1} receives its i^{th}. Thus, if $\lceil n/k \rceil \leq m+1$, P_{m+1} will receive its last sub-message no earlier than any other processor.

If the message comprises more than $m+1$ sub-messages, though, processor P_{2m} is the laggard: P_{2m} begins to receive sub-message $m+1+j$ at time $(2m+j+\lceil j/2m \rceil)(\beta + k\tau)$.

Since a total of $\lceil n/k \rceil$ sub-messages comprise the message, we need a function that compactly represents:

$$\text{rounds} = \begin{cases} m + \lceil n/k \rceil & \text{if } \lceil n/k \rceil \leq m+1 \\ m + \lceil n/k \rceil + \left\lceil \frac{\lceil n/k \rceil - m - 1}{2m} \right\rceil & \text{if } \lceil n/k \rceil > m+1 \end{cases}$$

The function is:

$$\text{rounds} = m - 1 + \lceil n/k \rceil + \left\lceil \frac{k(m-1)+n}{2mk} \right\rceil.$$

The total time required by the scheme is no more than $\beta + k\tau$ times the number of rounds. We note that, as a result of the abbreviated final sub-message, the last $\lceil n/k \rceil$ rounds send a total of n units. Accordingly, the total time is no more than

$$\left(m - 1 + \lceil n/k \rceil + \left\lceil \frac{k(m-1)+n}{2mk} \right\rceil\right)\beta + \left(\left(m - 1 + \left\lceil \frac{k(m-1)+n}{2mk} \right\rceil\right)k + n\right)\tau,$$

or more compactly,

$$T_*(n + k\left\lceil \frac{k(m-1)+n}{2mk} \right\rceil, m, k).$$

The value of k that minimizes this quantity provides the upper bound.

The proof that each processor receives the entire message is essentially the same as the even case; we decline to reproduce it. □

The example of Figure 3.8 demonstrates an minimal transmission of a message of size 97 in C_7, since for $\beta = 5$ and $\tau = 1$, the optimal value of k is 11.

3.2.3 The half duplex ring

The bounds for these half duplex cases are due to Fraigniaud and Lazard [8]. The algorithms that produce them cut the ring C_p between $P_{\lfloor p/2 \rfloor}$ and $P_{\lfloor p/2 \rfloor + 1}$, and then broadcast to the two independent paths that result. While the bounds are quite good, we will provide an improvement for each one by failing to cut the ring, and employing the extra communication capacity.

We could have employed the full-duplex results of Sections 3.2.1 and 3.2.2 by simply doubling the number of rounds. What's more, we could quite easily enhance those naive results by noting that in the full duplex schemes for the ring C_p, the first $\lceil p/2 \rceil$ rounds do not employ any edges in more than one direction. Only the subsequent rounds would thus need be doubled.

Since $2T_*(n', m', k') - m'(\beta + k'\tau) = T_1(n', m', k')$, the resulting bounds would be:

$$\begin{aligned}
b_{H*}(C_{2m}, n) &\leq S_1(\lceil n/2 \rceil, m) \\
b_{H*}(C_{2m+1}, n) &\leq \min_{1 \leq k \leq n} T_1(n + k - \left\lfloor \frac{n+k}{2} \right\rfloor, m, k) \\
b_{H1}(C_{2m}, n) &\leq S_1(n, m) \\
b_{H1}(C_{2m+1}, n) &\leq \min_{1 \leq k \leq n} T_1(n + k \left\lceil \frac{k(m-1)+n}{2mk} \right\rceil, m, k)
\end{aligned}$$

None of these bounds are tighter than those of Fraigniaud and Lazard. (And, in fact, only the upper bound for the processor bound even ring is as good.)

We employ an enhanced version of Fraigniaud and Lazard's scheme: rather than splitting the ring, we consider each edge to be unidirectional with the orientation initially defined so that P_0 has out-degree 2, $P_{\lceil p/2 \rceil}$ has in-degree 2, and all other processors have both out-degree and in-degree 1. We reverse the orientation on edges adjacent a processor (other than P_0, of course) that has seen the entire message. And we proceed in greedy fashion: the objective is to pipeline the message to all processors with in-degree 2 as quickly as possible.

We begin with the link bound ring with an even number $p = 2m$ of processors. We select a integer $0 \leq k \leq n$. The transmission scheme is divided into discrete phases.

In the first phase—phase 0 for convenience—the message is sent to P_m: we send an extended pipeline along the two paths to P_m in packets of size k. The "backward" pipeline uses the reverse of the message order employed by the "forward" pipeline. Time required: $T_*(\lceil n/2 \rceil, m, k)$. We label this time t_0.

Phase 1 sees the message sent to P_{m-1} and P_{m+1}: we continue to employ the extended pipeline scheme from P_0; yet processor P_m will simply send one packet to P_{m-1} and P_{m+1} respectively to provide the balance of the message. P_{m-1} and P_{m+1} will thus have received the entire message at time

$$t_1 = \min_{0 \leq n_1 \leq \lfloor n/2 \rfloor} \max \{T_*(n - n_1, m - 1, k), T_*(\lceil n/2 \rceil, m, k) + \beta + n_1\tau\}.$$

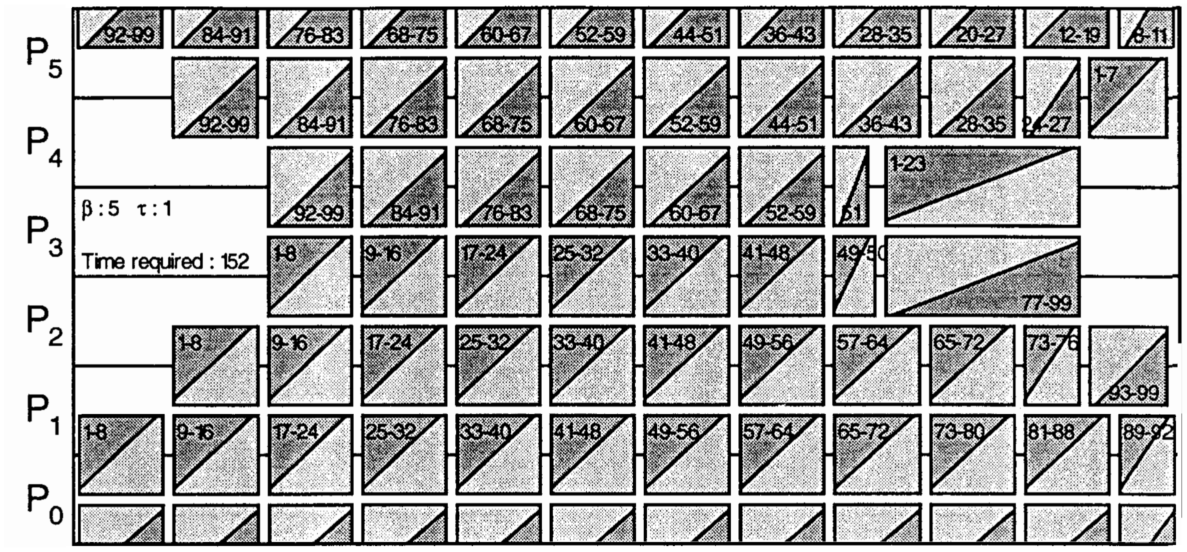


Figure 3.9: Example: Sending a message of size 99 in C_6.

The j^{th} phase sees P_{m+1-j} and P_{m-1+j} sending one packet to P_{m-j} and P_{m+j} respectively. If t_{j-1} was the time at which P_{m+1-j} (P_{m-1+j}) received the entire message, then

$$t_j = \min_{0 \le n_j \le n_{j-1}} \max \{T_*(n - n_j, m - j, k), t_{j-1} + \beta + n_j\tau\}.$$

The scheme stops (and fails to execute phase j) when $T_*(n, m - j, k) \le t_{j-1}$.

As an example, consider Figure 3.9 in which a message of size 99 is broadcast by this scheme to C_6 with $k = 8$. Unlike our previous examples, $k = 8$ is not the optimal value.

The transmission scheme will employ between one and m phases. If m phases are required, then P_1 will be informed from both P_0 and P_2. But more importantly, the total time to broadcast will have been no more than $T_*(n, 1, k)$ by the construction of time t_{m-1}. The other extreme case is when only one or two phases are required. If one phase is used, then $t_1 < t_0$ and the total time to broadcast is no more than $t_0 = T_*(\lceil n/2 \rceil, m, k)$. If three phases are used, then $t_1 = T_*(n, m - 1, k)$ is an upper bound. Proposition 3.2.3 follows from an optimal choice for k.

Proposition 3.2.3

$$b_{H*}(C_{2m}, n) \leq \min_{0 \leq k \leq n} \max\{T_*(\lceil n/2 \rceil, m, k), T_*(n, m-1, k)\}$$

While we make no claims of optimality for the last algorithm, it is interesting to note the ways in which it might be improved. The first inclination is to flip the direction of more than just two edges at the end of each phase: rather than broadcasting to P_{m-1} and P_{m+1} in phase 1, for example, choose P_{m-j} and P_{m+j} for some appropriate value of j. While it does not make for a proof, we have exhaustively examined all choices for j under the assumption that the packet size k does not change (except, as usual, in the last round): the choice $j = 1$ takes no more time than any other choice. The algorithm was then designed to exploit the adjacency of the targets of successive phases. There may indeed be some algorithm that does not have adjacent targets for the phases that also alters the size of the packets. We have not, by any means, exhausted all possibilities.

For the odd ring C_{2m+1}, the broadcast scheme is very similar to what we have just seen. But rather than broadcasting to P_m alone in phase 0, both P_m and P_{m+1} are targets. For this first phase, we use the transmission scheme that gave us Theorem 3.2.2. We ignore transfers of information toward P_0 from P_m or P_{m+1}, and we force transfers between P_m and P_{m+1} to alternate direction. If P_m and P_{m-1} exchange n_0 message units, that portion of the transmission ends in time $T_1(n_0, m+2, k)$, while the remainder of the message from P_0 requires $T_0(n-n_0, m, k)$ time. As a consequence,

$$t_0 = \min_{0 \leq n_0 \leq \lfloor n/2 \rfloor} \max\{T_*(n-n_0, m, k), T_1(n_0, m+2, k)\}.$$

Phase j, then, sees P_{m-j} (P_{m+1+j}) send a single packet back to P_{m-j-1} (P_{m+2+j}). The definition of t_j is unchanged. Figure 3.10 contains our next example: for parameters $\beta = 5$ and $\tau = 1$, the values $k = 12$ and $n_0 = 27$ are merely illustrative.

We construct the upper bound in the same manner as for the even cycle. (Note: we need $T_1(0, m', k')$ defined to be zero in order to deal correctly with the boundary case $\beta = 1$ and $\tau = 0$. If $\beta = 1$ and $\tau = 0$, the time to broadcast is m, not $m + 2$! The other boundary case — when $\beta = 0$ and $\tau = 1$ behaves nicely.)

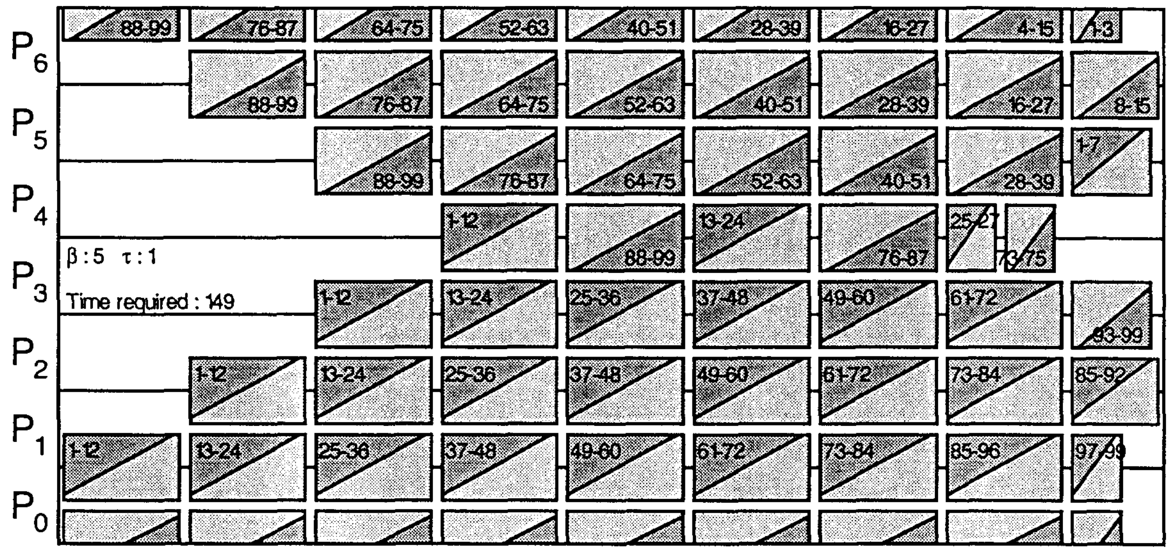


Figure 3.10: Example: Sending a message of size 99 in C_7.

Proposition 3.2.4 *If, for some integers n, m, and k*

$$t_0 = \min_{0 \le n_0 \le \lfloor n/2 \rfloor} \max\{T_*(n - n_0, m, k), T_1(n_0, m + 2, k)\}$$

then

$$b_{H*}(C_{2m+1}, n) \le \min_{0 \le k \le n} \max\{T_*(n, m - 1, k), t_0\}\,.$$

The processor bound ring now acquires our attention. Again, we make the division between the even and odd cases. The processor bound constraint prohibits the "single large packet" approach of the link bound examples. As a consequence, all transfers of information will be constrained to be packets of size k—except where that would result in sending redundant information. Aside from that, the even case is not a great departure from the link bound even ring. The odd case is only the expected synergy of the processor bound full duplex "rotating wait" scheme with the link bound half duplex scheme.

For the even ring C_{2m} we proceed exactly as in the link bound case, and send first to P_m. For simplicity, rather than counting the time t_j to the end of each phase, we

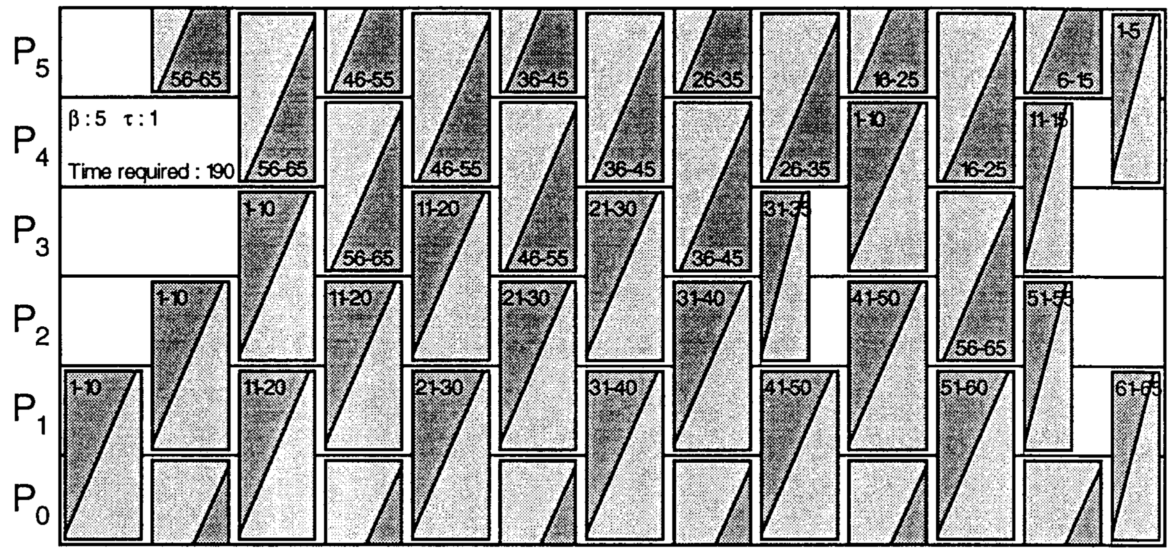


Figure 3.11: Example: Sending a message of size 65 in C_6.

only count the number of packets r_j that have been sent. (As a physical justification, the node(s) that have been informed by a phase cannot take any other action until its neighbours finish transmitting their packets, too.) From our count r_j we may obtain t_j, thus

$$t_j = r_j\beta + ((r_j - \lceil n/k \rceil)k + n)\tau.$$

The numbers of packets after each phase are:

$$\begin{aligned} r_0 &= m - 1 + \lceil n/k \rceil \\ r_1 &= \min_{0 \le n_1 \le \lfloor n/2 \rfloor} \max \left\{ r_0 + \lceil n_1/k \rceil, 2\left\lceil \frac{n-n_1}{k} \right\rceil + m - 3 \right\} \\ r_j &= \min_{0 \le n_j \le n_{j-1}} \max \left\{ r_{j-1} + \lceil n_j/k \rceil, 2\left\lceil \frac{n-n_j}{k} \right\rceil + m - 2 - j \right\} \end{aligned}$$

Figure 3.11 continues the set of example broadcast schemes. The choice of $k = 10$ is definitely sub-optimal: $k = 13$ gives a faster broadcast for the choice of parameters.

Again, the times t_0 (in case of the pathology $\tau = 0$) and t_1 (when $n_1 = 0$) form an upper bound on all transmissions of this form. The proposition follows from a choice of k to minimize the bound.

Proposition 3.2.5

$$b_{H1}(C_{2m}, n) \leq \min_{0 \leq k \leq n} \max \{T_*(n, m, k), T_1(n, m-1, k)\}$$

The scheme used for the odd ring C_{2m+1} bears the same resemblance to the even case that the scheme for the full duplex processor bound odd ring bears to the corresponding even ring. The algorithm proceeds, as before, in phases: phase 0 informing P_m and P_{m+1}; phase 1 informing P_{m-1} and P_{m+1}; phase j informing P_{m-j} and P_{m+j}. Moreover, packets are sent synchronized in rounds numbered beginning with 0: in round i, processor $P_{i \bmod (2m+1)}$ is idle.

The first phase is slightly complicated by the edge between P_m and P_{m+1}. Even in the full duplex processor bound schemes we have seen, this edge would only be used m times in every $2m+1$ rounds. This scheme is no different; but the edge is alternatively used in the two directions until the phase is complete.

Processor P_{2m} should again be the laggard, as it always suffers at least as many idle rounds as any other processor. We thus examine the P_{m+1} to P_{2m} half of the ring, in forming the time bounds. Following the example of the preceding even ring, we keep a count of the number of rounds.

Processor P_{m+1} receives its i^{th} packet from P_{m+2} in round $2i + m - 2 + \lceil i/m \rceil$. And P_{m+1} will receive its ℓ^{th} packet from P_m over the shared link in round $4\ell + m - 3 + \lceil (2\ell - 1)/m \rceil$. Phase 0 is thus complete in round

$$r_0 = \min_{\lceil n/2 \rceil \leq n_0 \leq n} \max \left\{ \begin{array}{l} 2\left\lceil \dfrac{n_0}{k} \right\rceil + m - 2 + \left\lceil \dfrac{n_0}{km} \right\rceil \\ 4\left\lceil \dfrac{(n-n_0)}{k} \right\rceil + m - 3 + \left\lceil \dfrac{1}{m}\left(2\left\lceil \dfrac{(n-n_0)}{k} \right\rceil - 1\right) \right\rceil \end{array} \right.$$

Note: n_0 is not a throw-away value. We will use it (and its colleagues n_j) to construct the phase limits r_j. The round in which processor P_{m+1} is first able to send a packet to P_{m+2} is exactly the round in which P_{m+1} would have received its $(n_0+1)^{\text{th}}$ packet from P_{m+2}, had the direction on the edge not been reversed.

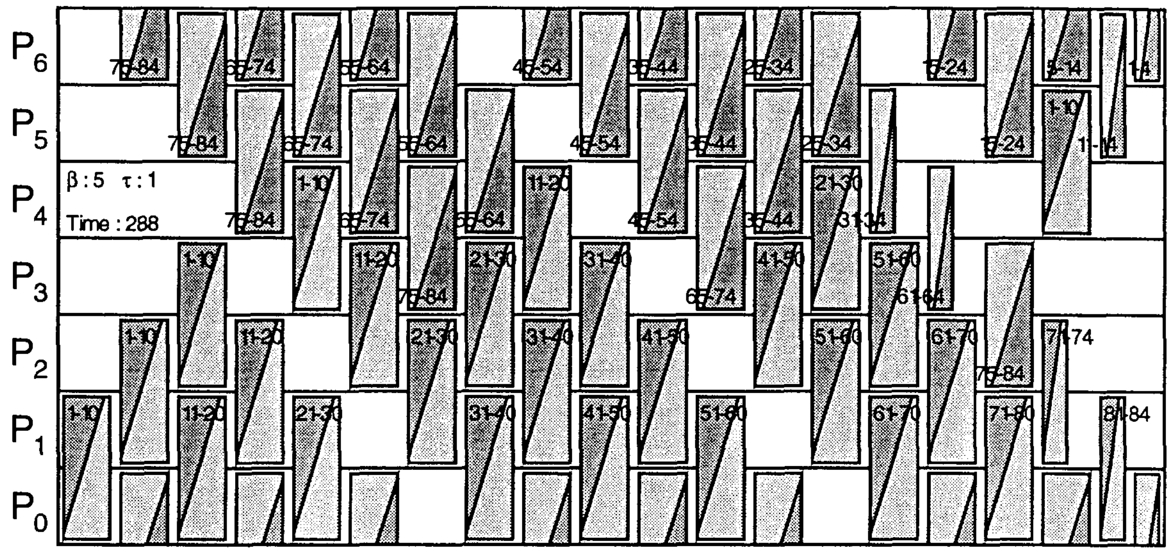


Figure 3.12: Example: Sending a message of size 84 in C_7.

In a similar manner, processor P_{m+1+j} receives its i^{th} packet from P_{m+2+j} in round $2i + m - 2 - j + \lceil i/m \rceil$. So phase j may complete in round

$$r_j = \min_{n_{j-1} \le n_j \le n} \max \begin{cases} 2\left\lceil \frac{n_j}{k} \right\rceil + m - 2 - j + \left\lceil \frac{n_j}{km} \right\rceil \\ 2\left\lceil \frac{n_{j-1} + n - n_j}{k} \right\rceil + m - 1 - j + \left\lceil \frac{n_{j-1} + n - n_j}{km} \right\rceil \end{cases}$$

As an example, consider Figure 3.12. Again, the value $k = 10$ is quite suboptimal: a packet size of 28, for example, produces a faster broadcast.

The time bound is derived in the same manner as we have seen in the previous examples. The proposition follows from choosing k to minimize the upper bound.

Proposition 3.2.6 *If, for some integers n, m, and k*

$$t_0 = \min_{0 \le n_0 \le n} \max \begin{cases} T_1(n_0, m, k) \\ T_*(n - n_0, m - 2 + 3\left\lceil \frac{(n - n_0)}{k} \right\rceil + \left\lceil \frac{1}{m}\left(2\left\lceil \frac{(n - n_0)}{k} \right\rceil + 1\right) \right\rceil, k) \end{cases}$$

(where the second option evaluates to 0 if $n = n_0$), then

$$b_{H1}(C_{2m+1}, n) \le \min_{0 \le k \le n} \max \{T_1(n, m - 1, k), t_0\}.$$

How do these broadcast schemes stack up against the schemes of Fraigniaud and Lazard? The schemes from this section for even rings are strictly better, while the broadcast schemes for the odd rings are no worse than the originals. In the odd case, our schemes employ communication capacity forbidden to those of Fraigniaud and Lazard: when that capacity is unnecessary (such as when $\tau = 0$) the schemes will perform identically; in any case where the pipeline employed by Fraigniaud and Lazard's schemes are nontrivial (that is, more than one packet is sent over some edge) our schemes will be faster. We do not, however, make any claims of optimality for these schemes.

3.3 Broadcasting in the complete graph

For the complete graph, we have only upper bounds to present. While Fraigniaud and Lazard [8] present good bounds (and, in fact label all four cases as 'Done') the following bounds are incommensurable in the full duplex processor bound case, and strictly better in the other three cases.

Recall that:

$$S_*(n,m) = \min_{1 \le k \le n} T_*(n,m,k)$$

is the minimum time to send a message of size n over m links under the link-bound model, where

$$T_*(n,m,k) = (\lceil n/k \rceil + m - 1)\beta + ((m-1)k + n)\tau.$$

Fraigniaud and Lazard present:

$$\begin{aligned}
b_{F*}(K_p, n) &\le S_*\left(\left\lceil \frac{n}{p-1} \right\rceil, 2\right) \\
b_{H*}(K_p, n) &\le S_*\left(\left\lceil \frac{n}{\lfloor p/2 \rfloor} \right\rceil, 3\right) \\
b_{F1}(K_p, n) &\le S_*(n, p-1) \\
b_{H1}(K_p, n) &\le S_*(2n, 2p-5)
\end{aligned}$$

While we will see:

$$b_{F*}(K_p,n) \leq \min_{1\leq k\leq n} T_*\left(\left\lceil\frac{n-k}{p-1}\right\rceil, 2, k\right)$$

$$b_{H*}(K_p,n) \leq \begin{cases} \min\limits_{1\leq k\leq n}\left(\left(2\left\lceil\frac{n-k}{pk}\right\rceil+1\right)\beta+\left(2\left\lceil\frac{n-k}{p}\right\rceil+k\right)\tau\right) \\ \min\limits_{1\leq k\leq n}\left((2r+1)\beta+\left(2\left\lceil\frac{n-k-r\lfloor\beta/\tau\rfloor}{p}\right\rceil+k\right)\tau\right) \\ \qquad \text{where } r = o\left(\log\left(\left\lfloor\frac{n-k}{p(k+\lfloor\beta/\tau\rfloor)}\right\rfloor+1\right)\right) \end{cases}$$

$$b_{F1}(K_p,n) \leq \begin{cases} (p=2^m) & S_*(n,\log_2 p) \\ (p=2m) & S_*(n,p/2) \\ (p=2m+1) & \min\limits_{1\leq k\leq n} T_*(n, \lfloor p/2\rfloor + \left\lceil\frac{\lfloor p/2\rfloor - 1 + \lceil n/k\rceil}{p-1}\right\rceil, k) \\ (p\neq 2^m) & S_*(n,\lfloor\log_2 p\rfloor)+\beta+n\tau \end{cases}$$

$$b_{H1}(K_p,n) \leq \begin{cases} (p=2^m) & S_*(n,\log_2 p) \\ (p\neq 2^m) & S_*(n,\lfloor\log_2 p\rfloor)+\beta+n\tau \end{cases}$$

3.3.1 The full duplex link bound complete graph

The first bound is the result of refining the analysis of Fraigniaud and Lazard.

Proposition 3.3.1

$$b_{F*}(K_p,n) \leq \min_{1\leq k\leq n} T_*\left(\left\lceil\frac{n-k}{p-1}\right\rceil, 2, k\right)$$

Proof: The scheme that provides the upper bound is synchronous, as follows:

- Choose a packet size k, and reserve a sub-message of size k.
- Divide the remaining $n-k$ message units into $p-1$ sub-messages, as evenly as possible; further divide these sub-messages into $r = \lceil (n-k)/(p-1)k\rceil$ chunks of size k—where we allow the last chunk to be smaller. Let $m(i,j)$ refer to the j^{th} chunk of the i^{th} sub-message, and $|m(i,j)|$ denote the size of $m(i,j)$. We note that the construction may result in some $m(i,r)$ having zero size. That will not cause any difficulty.

- We pad each chunk $m(i,r)$ to size k with the reserved sub-message giving $m'(i,r)$; and we create an additional chunk (possibly of size zero) for each processor $m'(i,r+1)$ with the remainder of the reserved sub-message. This construction ensures that $|m(i,r)| = |m'(i,r+1)|$.

- The chunks are sent in $r+1$ rounds, as follows:

  1. P_0 sends $m(i,1)$ to P_i, $1 \le i \le p-1$.
  2. P_0 sends $m(i,2)$ to P_i and
     P_i sends $m(i,1)$ to each P_j, $1 \le j \le p-1$, and $i \neq j$.

     $\vdots$

  $r-1$. P_0 sends $m(i,r-1)$ to P_i, $1 \le i \le p-1$ and
     P_i sends $m(i,r-2)$ to each P_j, $1 \le j \le p-1$, and $i \neq j$.

  r. P_0 sends $m'(i,r)$ to P_i, $1 \le i \le p-1$ and
     P_i sends $m(i,r-1)$ to each P_j, $1 \le j \le p-1$, and $i \neq j$.

  $r+1$. P_0 sends $m'(i,r+1)$ to P_i, $1 \le i \le p-1$ and
     P_i sends $m(i,r)$ (not the padded version!) to each P_j, $1 \le j \le p-1$, and $i \neq j$.

The first round takes time $\beta + k\tau$, and the remaining r rounds require a total of $r\beta + \lceil n - k/(p-1) \rceil \tau$; the total time required by this scheme is

$$\left(\left\lceil \frac{n-k}{(p-1)k} \right\rceil + 1 \right) \beta + (\left\lceil \frac{n-k}{p-1} \right\rceil + k)\tau.$$

The best choice for k gives the bound of the proposition.

What's more, the scheme ensures that each processor has received all n units of the message by the end of the $(r+1)^{\text{th}}$ round: P_i will receive $m(j,k)$ in round $k+1$ if $j \neq i$, and round k otherwise; the k units of the reserved sub-message will always be received in the r^{th} and $(r+1)^{\text{th}}$ rounds. □

As an example, consider the problem of sending a message of size 83 in K_5; processor P_0 wishes to broadcast to four other processors. Letting $\beta = 5$ and $\tau = 1$ for simplicity, as in other chapters, the optimal value of k is 10.

- Reserve $b_{74} \dots b_{83}$.
- Divide the remaining message into four sub-messages: $b_1 \dots b_{18}$, $b_{19} \dots b_{36}$, $b_{37} \dots b_{54}$, and $b_{55} \dots b_{73}$.
- Split the sub-messages into chunks of size k:

$$\begin{array}{ll} m(1,1) = b_1 \dots b_{10} & m(1,2) = b_{11} \dots b_{18} \\ m(2,1) = b_{19} \dots b_{28} & m(2,2) = b_{29} \dots b_{36} \\ m(3,1) = b_{37} \dots b_{46} & m(3,2) = b_{47} \dots b_{54} \\ m(4,1) = b_{55} \dots b_{64} & m(4,2) = b_{65} \dots b_{73} \end{array}$$

- Make the extra chunks m':

$$\begin{array}{ll} m'(1,2) = b_{11} \dots b_{18} b_1 b_2 & m'(1,3) = b_3 \dots b_{10} \\ m'(2,2) = b_{29} \dots b_{36} b_1 b_2 & m'(2,3) = b_3 \dots b_{10} \\ m'(3,2) = b_{47} \dots b_{54} b_1 b_2 & m'(3,3) = b_3 \dots b_{10} \\ m'(4,2) = b_{65} \dots b_{73} b_1 & m'(4,3) = b_2 \dots b_{10} \end{array}$$

- Send the chunks, in 3 rounds:
  1. P_0 sends $m(i,1)$ to P_i, for each $i = 1,2,3,4$.
  2. P_0 sends $m'(i,2)$ to P_i, for each $i = 1,2,3,4$ while P_i sends $m(i,1)$ to P_j for $j = 1,2,3,4$ and $j \neq i$.
  3. P_0 sends $m'(i,3)$ to P_i, for each $i = 1,2,3,4$ while P_i sends $m(i,2)$ to P_j for $j = 1,2,3,4$ and $j \neq i$.

  The first two rounds send packets of size 10, while the last round sends packets of size at most 9.

3.3.2 The half duplex link bound complete graph

For the half duplex case, Fraigniaud and Lazard demonstrate an upper bound of

$$b_{H*}(K_p, n) \leq S_* \left(\left\lceil \frac{n}{\lfloor p/2 \rfloor} \right\rceil, 3 \right).$$

While we could perform a slight enhancement of their analysis (following the example of the previous proposition) to obtain

$$b_{H*}(K_p, n) \le \min_{1\le k\le n}\left(\left(2\left\lceil\frac{n-k}{pk}\right\rceil+1\right)\beta+\left(2\left\lceil\frac{n-k}{p}\right\rceil+k\right)\right),$$

we will employ a slightly different technique to further reduce the bound.

Proposition 3.3.2 *For fixed values of n, p, and k, let r be the smallest integer so that*

$$p\left((2^r-1)k+(2^r-r-1)\lfloor\beta/\tau\rfloor\right)+k+r\lfloor\beta/\tau\rfloor \ge n,$$

then

$$b_{H*}(K_p, n) \le \min_{1\le k\le n}\left((2r+1)\beta+\left(2\left\lceil\frac{n-k-r\lfloor\beta/\tau\rfloor}{p}\right\rceil+k\right)\tau\right).$$

Proof: Again, the upper bound is synchronous. In contrast to the full-duplex case, though, the size of the rounds is not more-or-less constant; rather, each successive round will transmit slightly more than twice as many message units as the previous one. Moreover, we will need to employ the term *phase* rather than *round* since each phase (other than the first) will contain pairs of packets that use the same link, in series.

We fix k to some positive integer less than n, and we let r be the largest integer subject to the relation in the statement of the proposition. The transmission scheme will take $r+1$ phases.

We begin by reserving a sub-message of size $\lceil(n-k-r\lfloor\beta/\tau\rfloor)/p\rceil+k+r\lfloor\beta/\tau\rfloor$, leaving no more than $\lceil(n-k-r\lfloor\beta/\tau\rfloor)(p-1)/p\rceil$ message units. We divide the remaining units into $p-1$ sub-messages, as evenly as possible; each of these sub-messages is thus of size greater than $(2^{r-1}-1)k+(2^{r-1}-r)\lfloor\beta/\tau\rfloor$ and no larger than $(2^r-1)k+(2^r-r-1)\lfloor\beta/\tau\rfloor$ units by the construction.

Each sub-message is divided further into r chunks, each slightly more than twice the size of the previous (with the exception of the last chunk, which merely contains all the remaining units in the message): the first chunk is of size k, the i^{th} (for $1\le i<r$) is of size $2^{i-1}k+(2^{i-1}-1)\lfloor\beta/\tau\rfloor$, and the r^{th} has size $\lceil(n-k-r\lfloor\beta/\tau\rfloor)/p\rceil-(2^{r-1}-$

$1)k - (2^{r-1} - r)\lfloor\beta/\tau\rfloor$. As in the $F*$ case before, we let $m(i,j)$ refer to the j^{th} sub-sub-message in the i^{th} sub-message.

We extend each chunk $m(i,r)$, giving $m'(i,r)$ by padding it to length $(2^{r-1} - 1)k + (2^{r-1} - r)\lfloor\beta/\tau\rfloor$ from the reserved sub-message; each of the $p-1$ new chunks $m'(i,r+1)$ are simply the remaining portion of the reserved sub-message.

The sending scheme is in $r+1$ phases, as follows:

1. P_0 sends $m(i,1)$ to P_i, $1 \le i \le p-1$.

2. P_0 sends $m(i,2)$ to P_i, $1 \le i \le p-1$ while:

   (a) P_i sends $m(i,1)$ to each P_j, $1 \le j < i$ then

   (b) P_i sends $m(i,1)$ to each P_j, $i < j \le p-1$

⋮

r. P_0 sends $m'(i,r)$ to P_i, $1 \le i \le p-1$ while:

   (a) P_i sends $m(i,r-1)$ to each P_j, $1 \le j < i$ then

   (b) P_i sends $m(i,r-1)$ to each P_j, $i < j \le p-1$

$r+1$. P_0 sends $m'(i,r+1)$ to P_i, $1 \le i \le p-1$ while:

   (a) P_i sends $m(i,r)$ (unpadded version) to each P_j, $1 \le j < i$ then

   (b) P_i sends $m(i,r)$ (unpadded version) to each P_j, $i < j \le p-1$

Each processor receives all n units of the message by the end of the $(r+1)^{\text{th}}$ phase: P_i will receive $m(j,k)$ in phase $k+1$ if $j \ne i$, and phase k otherwise; and the reserved units will always be received directly from P_0 in the r^{th} and $(r+1)^{\text{th}}$ phases.

The first phase takes time $\beta + k\tau$. The remaining r phases are dominated by the time used by processors other than P_0: each P_i sends $2r$ packets that contain a total of $2\lceil(n-k-r\lfloor\beta/\tau\rfloor)/p\rceil$ units. The total time for the scheme is:

$$\left((2r+1)\beta + \left(2\left\lceil\frac{n-k-r\lfloor\beta/\tau\rfloor}{p}\right\rceil + k\right)\tau\right).$$

Selecting the best value of k provides the bound in the proposition. □

For illustration, consider the problem of sending a message of size 180 in K_5 under the half duplex link bound constraint. Letting $\beta = 5$ and $\tau = 1$, the optimal values for k and r are 10 and 2 respectively.

- Reserve a message of size 52: $b_{129} \ldots b_{180}$.
- Divide the remaining message into four sub-messages: $b_1 \ldots b_{32}$, $b_{33} \ldots b_{64}$, $b_{65} \ldots b_{96}$, and $b_{97} \ldots b_{128}$.
- Split the sub-messages into chunks:

$$\begin{array}{ll} m(1,1) = b_1 \ldots b_{10} & m(1,2) = b_{11} \ldots b_{32} \\ m(2,1) = b_{33} \ldots b_{42} & m(2,2) = b_{43} \ldots b_{64} \\ m(3,1) = b_{65} \ldots b_{74} & m(3,2) = b_{75} \ldots b_{96} \\ m(4,1) = b_{97} \ldots b_{106} & m(4,2) = b_{107} \ldots b_{128} \end{array}$$

- Make the extra chunks m'—$|m'(i,2)| = 25$ and $|m'(i,3)| = 22$:

$$\begin{array}{ll} m'(1,2) = b_{11} \ldots b_{32} b_{129} b_{130} b_{131} & m'(1,3) = b_{132} \ldots b_{180} \\ m'(2,2) = b_{43} \ldots b_{64} b_{129} b_{130} b_{131} & m'(2,3) = b_{132} \ldots b_{180} \\ m'(3,2) = b_{75} \ldots b_{96} b_{129} b_{130} b_{131} & m'(3,3) = b_{132} \ldots b_{180} \\ m'(4,2) = b_{107} \ldots b_{128} b_{129} b_{130} b_{131} & m'(4,3) = b_{132} \ldots b_{180} \end{array}$$

- Send the packets, in 3 phases:
  1. (size 10) P_0 sends $m(i,1)$ to P_i, for each $i = 1,2,3,4$.
  2. (size 25) P_0 sends $m'(i,2)$ to P_i, for each $i = 1,2,3,4$ while
     (a) (size 10) P_i sends $m(i,1)$ to P_j for $1 \leq i < j \leq 4$ then
     (b) (size 10) P_i sends $m(i,1)$ to P_j for $4 \geq i > j \geq 1$
  3. (size 49) P_0 sends $m'(i,3)$ to P_i, for each $i = 1,2,3,4$ while
     (a) (size 22) P_i sends $m(i,2)$ to P_j for $1 \leq i < j \leq 4$ then
     (b) (size 22) P_i sends $m(i,2)$ to P_j for $4 \geq i > j \geq 1$

3.3.3 A digression: the full duplex processor bound hypercube

One of the ways to find an upper bound for processor bound complete graph is to employ a hypercube-like sending scheme. Accordingly, we turn our attention to the hypercube Q_m on 2^m nodes.

Proposition 3.3.3

$$b_{F1}(Q_m, n) \le S_*(n, m)$$

Proof: We employ what I'd have thought was the standard pipelined hypercube scheme. Imagine my surprise when I discovered that Fraigniaud and Lazard employ a labeling scheme of Johnsson and Ho [16] to provide $S_*(n, m+1)$ as their best hypercube bound under the full duplex processor bound constraint.

The bound I'd had in mind is again a synchronous scheme. The message is simply divided into $\lceil n/k \rceil$ sub-messages of size k (where we allow the last sub-message to be smaller). We label the i^{th} of these sub-messages (beginning the numbering, uncharacteristically, at 0) $\gamma(i)$. As as before, we let $|\gamma(j)|$ denote the size of the sub-message $\gamma(j)$.

Rather than sending the sub-messages

$$\{\gamma(i) \mid 0 \le i < \lceil n/k \rceil\},$$

we create a slightly different collection to send:

- $\mu(i) = \gamma(i)$, for $0 \le i < \lceil n/k \rceil - 1$.
- $\mu(\lceil n/k \rceil - 1)$ is the first k units of the concatenation $\gamma(\lceil n/k \rceil - 1)\gamma(\lceil n/k \rceil - 2)$.
- $\mu(\lceil n/k \rceil)$ is the last $|\gamma(\lceil n/k \rceil - 1)|$ units of $\gamma(\lceil n/k \rceil - 2)$.

We relabel the nodes so that the originator of the broadcast is node $\vec{0}$. For my convenience, I'll both label the neighbours of $\vec{0}$ by $\{e_0, e_1, e_{m-1}\}$ and denote sub-cubes by

$$Q\left\{e_{i_1}, e_{i_2}, \ldots, e_{i_p}, \bar{e}_{j_1}, \bar{e}_{j_2}, \ldots, \bar{e}_{j_q}\right\},$$

meaning the sub-cube where each of the indices $i_1, i_2, \ldots, i_p$ are 1, and each of the indices $j_1, j_2, \ldots, j_q$ are 0.

The sending scheme is fairly simple: there are $\lceil n/k \rceil + m - 1$ rounds, numbered from 0 to $r = \lceil n/k \rceil + m - 2$.

- In round $j < \lceil n/k \rceil$, $\vec{0}$ sends $\mu(j)$ to $e_{j \bmod m}$.
- In round j, where $\lceil n/k \rceil \le j < r$, $\vec{0}$ sends $\mu(\lceil n/k \rceil - 1)$ to $e_{j \bmod m}$.
- In round $j < r$, $\omega \neq \vec{0}$ sends the sub-message with the largest index it knows to $\omega \oplus e_{j \bmod m}$.
- In round r, $\omega \in Q\left\{e_{(\lceil n/k \rceil - 2) \bmod m}\right\}$ sends $\mu(\lceil n/k \rceil)$ to $\omega \oplus e_{r \bmod m}$.
- In round r, $\omega \in Q\left\{\bar{e}_{(\lceil n/k \rceil - 2) \bmod m}\right\}$ (this includes $\vec{0}$) sends $\gamma(\lceil n/k \rceil - 1)$ to $\omega \oplus e_{r \bmod m}$.

The third point is a bit vague for a proof. In order to specify it more precisely, we need one last bit of mechanics: let $\pi_j : V(Q_m) \to V(Q_m)$ be the permutation that maps

$$\omega = \omega_0 \omega_1 \ldots \omega_{m-1} \mapsto \omega_j \omega_{j+1} \ldots \omega_{m-1} \omega_0 \ldots \omega_{j-1}.$$

We want to be able to speak of *the leftmost* 1 in $\pi_j(\omega)$: we will say that the leftmost 1 of $\pi_j(e_i)$ is in position $(i - j) \bmod m$. The leftmost 1 of $\pi_3(e_2 + e_7)$, for example, is in position 4.

In round $j < r$, then, processors (other than $\vec{0}$) send a sub-message depending on the position of their leftmost 1 under the map π_j. Specifically, if the leftmost 1 of $\pi_j(\omega)$ is in position i, Processor ω sends sub-message $\mu(\min\{j - m + i, \lceil n/k \rceil - 1\})$, where sending a sub-message with a negative index is simply a no-op.

This rule also produces the last two points in the description of the scheme. In round r, a processor ω would send sub-message $\mu(\lceil n/k \rceil - 2)$ if and only if the leftmost 1 in $\pi_r(\omega)$ was in position 0 (as when $\omega \in Q\left\{e_{(\lceil n/k \rceil - 2) \bmod m}\right\}$) and otherwise send $\mu(\lceil n/k \rceil - 1)$. Since these two sub-messages overlap in $k - |\gamma(\lceil n/k \rceil)|$ units, there is no need to send these particular units again: as a consequence, we employ an abbreviated sending in the final step.

The first $\lceil n/k \rceil - 1$ rounds plus the last round see $\vec{0}$ send $\lceil n/k \rceil$ packets whose total size is n: these take time $\lceil n/k \rceil \beta + n\tau$. The remaining $m-1$ rounds each send k units, for a total of

$$(\lceil n/k \rceil + m - 1)\beta + ((m-1)k + n)\tau.$$

If we minimize this quantity over all values of k, the result is exactly $S_*(n, m)$. □

As a small example, consider sending a message of 19 bits to Q_4. Given $\beta = 5$ and $\tau = 1$, the optimal value of k is 5.

- Divide $b_1 \ldots b_{19}$ into four sub-messages: $\gamma(0) = b_1 \ldots b_5$, $\gamma(1) = b_6 \ldots b_{10}$, $\gamma(2) = b_{11} \ldots b_{15}$, and $\gamma(3) = b_{16} \ldots b_{19}$.
- Construct the alternate sub-messages μ: $\mu(0) = \gamma(0)$, $\mu(1) = \gamma(1)$, $\mu(2) = \gamma(2)$, $\mu(3) = b_{16} \ldots b_{19} b_{11}$, and $\mu(4) = b_{12} \ldots b_{15}$.
- Send the sub-messages in 7 rounds:

  0. Processor 0000 sends $\mu(0)$ to processor 1000.
  1. Processor 0000 sends $\mu(1)$ to processor 0100, while
     processor 1000 sends $\mu(0)$ to processor 1100.
  2. Processor 0000 sends $\mu(2)$ to processor 0010, while
     processor 0100 sends $\mu(1)$ to processor 0110,
     processor 1100 sends $\mu(0)$ to processor 1110, and
     processor 1000 sends $\mu(0)$ to processor 1010.
  3. Processor 0000 sends $\mu(3)$ to processor 0001, while
     processor 0010 sends $\mu(2)$ to processor 0011,
     each processor $01i_20$ sends $\mu(1)$ to processor $01i_21$, and
     each processor $1i_1i_20$ sends $\mu(0)$ to processor $1i_1i_21$.
  4. Each processor $000i_3$ sends $\mu(3)$ to processor $100i_3$, while
     each processor $001i_3$ sends $\mu(2)$ to processor $101i_3$,
     each processor $01i_2i_3$ sends $\mu(1)$ to processor $11i_2i_3$, and
     each processor $1i_1i_2i_3$ sends $\mu(0)$ to processor $0i_1i_2i_3$.

5. Each processor $i_0 00 i_3$ sends $\mu(3)$ to processor $i_0 10 i_3$, while each processor $i_0 01 i_3$ sends $\mu(2)$ to processor $i_0 11 i_3$, and each processor $i_0 1 i_2 i_3$ sends $\mu(1)$ to processor $i_0 0 i_2 i_3$.
6. Each processor $i_0 i_1 0 i_3$ sends $\gamma(3)$ to processor $i_0 i_1 1 i_3$, while each processor $i_0 i_1 1 i_3$ sends $\mu(4)$ to processor $i_0 i_1 0 i_3$.

With the exception of the last round, each round j doubles the number of processors that have received $\{\mu(i) \mid \max\{j-m, 0\} \leq i \leq \min\{j, \lceil n/k \rceil - 1\}\}$. And if not for the abbreviated sending scheme, the last round would be the same. But after the first 6 rounds in our example, each processor has received message unit b_{11}—as a result, b_{11} need not be transmitted by any processor in the final step.

As it turns out, this is not very different the processor-bound hypercube sending scheme of Johnsson and Ho [16]. The feature of this scheme missing in the original is that the originator stays busy until the very final round.

3.3.4 The full duplex processor bound complete graph

How can we use this scheme for the complete graph? Clearly, if $p = 2^m$, the previous scheme may be applied directly. For more general values of p, a naive solution would be to have K_p masquerade as the hypercube $Q_{\lceil \log_2 p \rceil}$. Each processor would have no more than two separate identities. In each round, every processor would be responsible for as many communications as it had identities: any maximal matching in the $\Delta = 2$ communication graph induced each round will partition the round into two parts which may be scheduled in series. As a result:

$$b_{F1}(K_p, n) \leq 2S_*(n, \lceil \log_2 p \rceil).$$

Much simpler, though, is to have the originator employ the hypercube scheme to broadcast to the largest sub-hypercube, then employ one additional step to inform the remaining nodes:

Corollary 3.3.1

$$b_{F1}(K_p, n) \leq S_*(n, \lfloor \log_2 p \rfloor) + \beta + n\tau$$

Interestingly, though, we can also employ the scheme for the full duplex processor bound ring that we saw in Subsection 3.2.2. That scheme differentiated between even and odd rings:

Corollary 3.3.2 *For even complete graphs K_{2m},*

$$b_{F1}(K_{2m}, n) \leq S_*(n, m).$$

Corollary 3.3.3 *For odd complete graphs K_{2m+1},*

$$b_{F1}(K_{2m+1}, n) \leq \min_{1 \leq k \leq n} T_*(n + k \left\lceil \frac{k(m-1)+n}{2mk} \right\rceil, m, k).$$

The upper bounds for complete graphs K_p with $p = 2^m$ or $p = 2m'$ are better than the bound of Fraigniaud and Lazard. For p odd, however, none of the three following bounds are clearly better than the others.

$$b_{F1}(K_p, n) \leq \begin{cases} S_*(n, p-1) & \text{Fraigniaud and Lazard} \\ S_*(n, \lfloor \log_2 p \rfloor) + \beta + n\tau & \text{Hypercube scheme} \\ \min_{1 \leq k \leq n} T_*(n + k \left\lceil \frac{k(m-1)+n}{2mk} \right\rceil, m, k) & \text{Cycle scheme} \end{cases}$$

3.3.5 The half duplex processor bound complete graph

Let Q_m^+ be the hypercube on 2^m nodes augmented by additional edges that join antipodal nodes.[1] This is normally called a *folded hypercube.* We can see that the scheme of Proposition 3.3.3 may be employed (with a small alteration) in the half duplex processor bound model:

Proposition 3.3.4

$$b_{H1}(Q_m^+, n) \leq S_*(n, m)$$

[1]For some vertex ω of the hypercube Q_m, the antipodal node is just $\bar{\omega}$—the unique node at distance m.

Proof: The first $m-1$ rounds of the hypercube scheme are already half-duplex: they need not change. In each other round, though, half the nodes of the hypercube are sending exactly the same sub-message. Specifically, in round j, all nodes ω whose leftmost one of $\pi_j(\omega)$ is in position 0 send sub-message $\mu(\min\{j-m+i, \lceil n/k \rceil - 1\})$. If those nodes employ the antipodal edge to send, rather than their dimension $j \bmod p$ edge, the scheme is unchanged in effect. □

The application is much the same as in the full duplex case. If the number of processors in the complete graph is a power of two, we may employ the folded hypercube scheme unchanged, and get

Corollary 3.3.4

$$b_{H1}(K_{2^m}, n) \le S_*(n, \log_2 p)$$

For a more arbitrary number of processors, we can first use the folded hypercube scheme to broadcast to $2^{\lfloor \log_2 p \rfloor}$ nodes. Then, in a single round, all other nodes can be informed in time $\beta + n\tau$.

Corollary 3.3.5

$$b_{H1}(K_p, n) \le S_*(n, \lfloor \log_2 p \rfloor) + \beta + n\tau$$

3.4 Summary

We have seen several new upper bounds for the broadcast problem in cycles and complete graphs (and, peripherally, for the full-duplex processor bound hypercube). In the three simplest cases—the processor bound and link bound unidirectional ring, and the full duplex link bound bidirectional ring—we have also provided matching lower bounds.

The bounds presented for the various flavours of ring differ quite sharply in quality. I think the upper bounds for the full duplex processor bound case are optimal, despite the fact that a proof has been consistently and annoyingly elusive. It is possible that the upper bound for the even half duplex ring under the link bound constraint is optimal as well, though I wouldn't go so far as to make that a formal conjecture. The remaining half-duplex bounds, however, are quite soft. In addition to being nowhere near optimal, they do not even tightly represent the transmission schemes that motivate them.

The bounds for the complete graph are similarly variable. The upper bound in the full duplex link bound case is quite good: of all the cases without lower bounds, it is in my opinion the most likely to acquire a matching lower bound. The scheme for the half duplex link bound complete graph also seems solid, though the expression of the upper bound could use some algebraic tidying. The pair of upper bounds for the processor bound constraint are fairly untidy, but that seems inherent in the problem. Nonetheless, they can probably stand some improvement.

The bounds for the hypercube and folded hypercube are incidental: means to an end, rather than an end in themselves. We do not mean to imply that these are the only hypercube results that could be brought to bear on the complete graph: rather, the opposite. There is likely much research in the area of hypercubes (and indeed, other topologies) that could be beneficially applied to even our simple cases.

For a tabular summary of the bounds, see Table 5.1.

Chapter 4

Gossiping

4.1 The problem

Gossiping is a natural extension of broadcasting, in which each processor simultaneously broadcasts it's piece of information to all others. The vast majority of authors that have addressed gossiping have employed a different cost metric than our linear one. See [17],[6] and [15] for two early papers and a recent survey.

There are several flavours of the linear cost gossip problem. One might, for example, look for the sparsest topologies that allow gossiping under some model in minimal time [10]. Alternatively, one could examine linear cost gossiping under other than our store-and-forward communication method. Fraigniaud and Peters [11], for example, compare store-and-forward algorithms for gossiping with circuit switched schemes. We will concern ourselves with the problem of finding good time bounds for gossiping in a fixed topology: the ring.

4.2 Gossiping in the ring

We will employ the same ring of p processors we have seen in previous chapters, and attempt to find upper bounds on the time $g(C_p, n)$ for the gossiping problem where each processor wishes to share a message of size n. In addition, and where possible, we will construct a matching lower bound. To distinguish the six cases, we continue using the notation of Fraigniaud and Lazard [8] and (wherever applicable) employ subscripts F and H for full and half duplex; and $*$ and 1 for link and processor bound, respectively.

4.2.1 The unidirectional link bound ring

For the unidirectional ring $\vec{C}_p$ of p processors, we will demonstrate matching upper and lower bounds that show:

$$g_*(\vec{C}_p, n) = (p-1)(\beta + n\tau).$$

The upper bound is synchronous, and due to Saad and Schultz [22]. The gossip scheme is divided up into rounds in which every processor sends a packet of size n to its neighbour. During the first round, each processor sends its own message, and in each subsequent round the processors send the information they received in the previous round. After $p-1$ rounds, each processor will have received $n(p-1)$ units of information: the corpus of all other processors.

The lower bound is a bit more involved. Like in previous chapters, we consider any scheme that gossips messages of size n from each processor to all others, and construct a digraph of the packets.

We establish some notation:

- The ring has p processors, P_0,P_1,...,P_{p-1}. Each processor P_i may send to P_{i+1}, and receives messages from P_{i-1}. We say that P_{i+1} is *after* P_i, and that P_{i-1} is *before* P_i in the ring.

- Each processor P_i begins with a message M_i of size n, $b(i,1)$,$b(i,2)$,...,$b(i,n)$.

- Each processor P_i sends $\nu(i)$ packets, $\lambda(i,1)$,$\lambda(i,2)$,...,$\lambda(i,\nu(i))$. As before, we denote the size of $\lambda(i,j)$ by $|\lambda(i,j)|$.

- Packet $\lambda(i,j)$ takes $t(\lambda(i,j)) = \beta + |\lambda(i,j)|\tau$ time to send. We extend the function $t()$ to paths $\mathcal{A}$ in the digraph: $t(\mathcal{A}) = \sum_{p \in \mathcal{A}} t(p)$.

- Each packet $\lambda(i,j)$ has a start time $st(\lambda(i,j)) \leq 0$. We let $st(\lambda(0,1)) = 0$, possibly by renumbering the processors as appropriate.
- We call two gossip schemes *equivalent* if there is a one to one mapping between their packet sets that preserves start time. Equivalent schemes will thus form an equivalence class.

We first establish that in a minimal scheme each processor may send the message units in, for lack of a better term, relative anti-lexicographic order.

Lemma 4.2.1 *Any minimal gossip scheme has an equivalent scheme in which each processor P_i sends the message units in order $b(i,1)$,$b(i,2)$,...,$b(i,n)$,$b(i-1,1)$,$b(i-1,2)$,...,$b(i-1,n)$,...,$b(i-p+2,n)$, where the first index of each message unit is considered modulo p.*

Proof: Suppose the contrary.

Consider a minimal gossip scheme that has no equivalent that sends in relative anti-lexicographic order. For each equivalent scheme $\mathcal{S}$, let $\rho(\mathcal{S},i)$ be the size of the longest prefix of message units sent by P_i in relative lexicographic order, and let $\rho(\mathcal{S}) = \min\limits_{0 \leq i < p} \rho(\mathcal{S},i)$. Let $\mathcal{S}'$ be an equivalent scheme that maximizes $\rho()$, and additionally has as few processors as possible that have the $(k+1)^{\text{th}}$ unit out of order, where $k = \rho(\mathcal{S}')$; and let P_i be a processor for which the $(k+1)^{\text{th}}$ unit is out of order in $\mathcal{S}'$.

Suppose this $(k+1)^{\text{th}}$ unit of P_i's is $b(i_1,j_1)$, and let the unit that would have appeared in that position in relative anti-lexicographic order be $b(i_2,j_2)$. Now $b(i_1,j_1)$ will never be sent by P_{i_1-1}, nor will $b(i_2,j_2)$ ever be sent by P_{i_2-1}. There will be at most $p-2$ processors that send both units, and these processors may be divided into two disjoint ranges. P_i will be a member of such a range of processors: it is well defined to speak of the processors before and after P_i in the range.

We alter $\mathcal{S}'$, by permuting the positions of $b(i_1,j_1)$ and $b(i_2,j_2)$ in the packets sent by P_i and all processors after it in the range that sends both units. The only possible problem that could arise in the permuted scheme is that P_i might not have received $b(i_2,j_2)$ by the time it should be sent.

In fact, $b(i_2, j_2)$ will have been known by P_i in time. If $k \leq n$, then $i_2 = i$, and $b(i_2, j_2)$ is part of P_i's original message. Otherwise, $b(i_2, j_2)$ will be the $(k - n + 1)^{\text{th}}$ unit sent by P_{i-1} in anti-lexicographic order: by our construction, we know it was sent in time. Moreover, $b(i_1, j_1)$ is not the unit that belongs as the $(k + 1)^{\text{th}}$ unit sent by P_i; it must be at least the $(k - n + 2)^{\text{th}}$ or subsequent message unit sent by P_{i-1}; and so if P_i has seen $b(i_2, j_2)$, it must also have seen $b(i_1, j_1)$. As a consequence, the permuted scheme must be a valid sending scheme, and equivalent to $\mathcal{S}'$.

In the relative anti-lexicographic order, $b(i_1, j_1)$ and $b(i_2, j_2)$ are both sent after the k^{th} unit by each processor in which they were permuted: in fact, they should be sent after the $(k+n)^{\text{th}}$ unit by each processor in which they were permuted other than P_i. Since the first k message units from each processor are sent in the correct order, the permutation will not change the order of the first k units on any processor. In addition, if the $(k + 1)^{\text{th}}$ unit from a processor is sent in the correct order, it will also not be altered by the permutation: a correct $(k + 1)^{\text{th}}$ units cannot be either $b(i_1, j_1)$ or $b(i_2, j_2)$.

This permuted scheme thus contradicts our construction of $\mathcal{S}'$. The permuted scheme has one fewer processor for which the $(k+1)^{\text{th}}$ unit is out of order. As a result of the contradiction, it must be the case that each gossiping scheme has an equivalent that sends the message units in relative anti-lexicographic order. □

Like in Section 2.3, we consider a minimal gossiping scheme in which the message units are sent in order, and construct the packet content digraph from the collection of packets, with adjacent packet edges $(\lambda(i, j), \lambda(i, j+1))$ where $j < \nu(i)$, and overlapping packet edges $(\lambda(i, j), \lambda(i + 1, j'))$ whenever $\lambda(i, j) \cap \lambda(i + 1, j') \neq \phi$.

Rather than running any strange algorithms, though, we employ a fairly pedestrian induction to demonstrate the lower bound.

Lemma 4.2.2 *In the packet content digraph of any minimal gossiping scheme that sends the message units in relative anti-lexicographic order, for each* $1 \leq k \leq p - 1$ *and each processor* P_i, *there is a path* $\mathcal{A}_{i,k}$ *that ends at the packet* $\lambda(i, j(i, k))$ *(the packet sent by* P_i *that contains unit* $b(i - k + 1, n)$*) such that* $t(\mathcal{A}_{i,k}) \geq k(\beta + n\tau)$.

Proof: Not surprisingly, we prove the result by induction on k.

The basis, where $k = 1$, is straightforward. Let $\mathcal{A}_{i,1} = \lambda(i,1)\dots\lambda(i,j(i,1))$. The path $\mathcal{A}_{i,1}$ must contain at least one packet, and transmits a total of n message units:

$$t(\mathcal{A}_{i,1}) \geq \beta + \sum_{\ell=1}^{j(i,1)} |\lambda(i,\ell)|\,\tau \geq \beta + n\tau.$$

Suppose the lemma holds for some particular value of k in the following way: there is a path $\mathcal{A}_{i,k}$ that ends at packet $\lambda(i,j(i,k))$ where

$$t(\mathcal{A}_{i,k}) \geq k\beta + \sum_{\ell=1}^{j(i,k)} |\lambda(i,\ell)|\,\tau \geq k(\beta + n\tau).$$

Let P_i be any processor of the ring, and $\lambda(i,j(i,k))$ the packet sent by P_i that contains $b(i-k+1,n)$, as in the statement of the lemma. If unit $b(i-k,n)$ is not contained in $\lambda(i,j(i,k))$, then (that is, if $j(i,k+1) \neq j(i,k)$)

$$\mathcal{A}_{i,k+1} = \mathcal{A}_{i,k}\lambda(i,j(i,k)+1)\dots\lambda(i,j(i,k+1)).$$

The sub-path $\lambda(i,j(i,k)+1)\dots\lambda(i,j(i,k+1))$ must contain at least one packet, and transmits sufficient message units so that

$$\mathcal{A}_{i,k+1} \geq (k+1)\beta + \sum_{\ell=1}^{j(i,k+1)} |\lambda(i,\ell)|\,\tau \geq (k+1)(\beta + n\tau).$$

If $\lambda(i,j(i,k))$ contains $b(i-k,n)$ (that is, $j(i,k) = j(i,k+1)$), then

$$\mathcal{A}_{i,k+1} = \mathcal{A}_{i-1,k}\lambda(i,j(i,k+1))$$

is a reasonable choice, since $b(i-k,n) \in \lambda(i-1,j(i-1,k)) \cap \lambda(i,j(i,k+1))$. In addition, since $|\lambda(i,j(i,k+1))| > n$,

$$n + \sum_{\ell=1}^{j(i-1,k)} |\lambda(i-1,\ell)| \leq \sum_{\ell=1}^{j(i,k+1)} |\lambda(i,\ell)|$$

and so

$$\mathcal{A}_{i,k+1} \geq (k+1)\beta + \sum_{\ell=1}^{j(i,k+1)} |\lambda(i,\ell)|\,\tau \geq (k+1)(\beta + n\tau)$$

in this case as well.

In either case, the induction hypothesis holds for value $k+1$. As a consequence of the principle of mathematical induction, the lemma must hold for $1 \le k \le p-1$. □

Theorem 4.2.1 follows as a corollary of the combination of the above lemma and the upper bound.

Theorem 4.2.1 *Under the linear cost model, $(p-1)(\beta + n\tau)$ time is necessary and sufficient for gossiping in the unidirectional link bound ring of p processors.*

4.2.2 The unidirectional processor bound ring

We will demonstrate two separate results. For even rings:

$$g_1(\vec{C}_p, n) = p\beta + 2(p-1)n\tau,$$

and for odd rings:

$$g_1(\vec{C}_p, n) = (p+1)\beta + 2pn\tau.$$

The upper bounds are both synchronous. In the even case, we have p rounds, the first and last of which send n units; all others send $2n$. In even rounds, all even numbered processors send; in odd rounds, the odd numbered processors send.

The odd case is only slightly more complicated. $p+1$ rounds, the first and last of which are of n units, and the rest of $2n$. Processor P_i is idle in round i. And in round i, processor P_{i+1-2k} receives a packet from P_{i-2k} for $0 \le k \le \frac{1}{2}((p-3))$. All indices, of course, modulo p. Figure 4.1 illustrates the scheme.

The lower bound for the even ring follows the pattern of the unidirectional ring in the previous section. But instead of employing the packet content digraph, we use the packet time digraph from Section 2.4.

First, we define the predecessor and successor operations on the set of packets.

$$\mathrm{succ}\,(\lambda(i,j), A) = \lambda(i_0, j_0)$$

where $\lambda(i_0, j_0)$ is the packet that achieves the minimum

$$\min\left\{st(\lambda(\hat{\imath},\hat{\jmath})) \mid \hat{\imath} \in A,\ st(\lambda(i,j)) + t(\lambda(i,j)) \le st(\lambda(\hat{\imath},\hat{\jmath}))\right\},$$

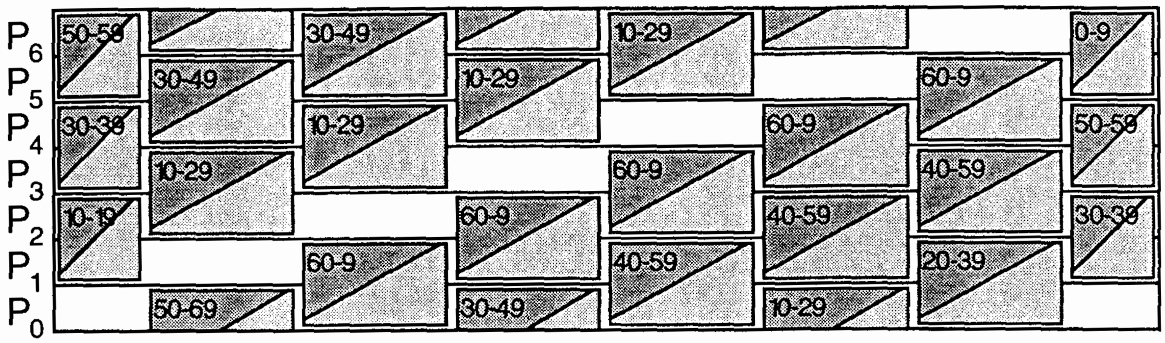


Figure 4.1: Example: Gossiping messages of size 10 in C_7

and

$$\text{pred}\,(\lambda(i,j),A) = \lambda(i_1,j_1)$$

where $\lambda(i_1,j_1)$ is the packet that achieves the maximum

$$\max\{st(\lambda(\hat{\imath},\hat{\jmath})) \mid \hat{\imath} \in A,\ st(\lambda(i,j)) \geq st(\lambda(\hat{\imath},\hat{\jmath})) + t(\lambda(\hat{\imath},\hat{\jmath}))\}.$$

We then construct the packet time digraph G.

••The nodes of the digraph are

$$V(g) = \{\mu(i,j) \mid 0 \leq i \leq p-1,\ 1 \leq j \leq \nu(i-1)+\nu(i)\},$$

where for a given processor P_i, the vertices $\mu(i,j)$ correspond to the packets either received or sent by P_i, sorted by start time. Each packet $\lambda(i,j)$ is represented twice: as $\mu(i,j_s)$ in the guise of a sent packet, and as $\mu(i+1,j_r)$ in the guise of a received packet. Formally, we use the same mapping $\text{p} : V(G) \to \Lambda$ from the node set of the digraph to the collection of packets.

$$\begin{aligned} \text{p}(\mu(i,1)) &= \text{either } \lambda(i-1,1) \text{ or } \lambda(i,1) \\ \text{p}(\mu(i,j+1)) &= \text{succ}\,(\text{p}(\mu(i,j)),\{i-1,i\}) \end{aligned}$$

Yet, we will tend to blur the distinction between the two, and treat a node $\mu(i,j)$ as if it were the packet it represents.

- The edges of G are the union of all:

  Adjacent packet edges $(\mu(i,j), \mu(i,j+1))$, for all processors P_i, and indices $1 \le j \le \nu(i-1) + \nu(i)$.

  Successor edges $(\mu(i,j), \mu(i+1,j'))$, where

  $$\mathrm{p}(\mu(i+1,j')) = \mathrm{succ}\left(\mathrm{p}(\mu(i,j)), \{i, i+1\}\right)$$

  and *predecessor* edges: $(\mu(i-1,\bar{j}), \mu(i,j))$; and

  $$\mathrm{p}(\mu(i-1,\bar{j})) = \mathrm{pred}\left(\mathrm{p}(\mu(i,j)), \{i-2, i-1\}\right).$$

(All processor indices, of course, are considered modulo p.)

In addition, we define two helper functions in order to more easily talk about the packets and their context. For each vertex $\mu(i,j)$, we let $f_r(\mu(i,j))$ denote the number of units P_i has received, by the time that $\mu(i,j)$ has finished transmitting, and similarly let $f_s(\mu(i,j))$ represent the number of message units that P_i has sent at that same point. For convenience, we let

$$f(\mu(i,j)) = f_r(\mu(i,j)) + f_s(\mu(i,j)) = \sum_{1 \le \ell \le j} |\mu(i,\ell)|.$$

The following observation results from noting that a processor is only able to send its own message, or units it has received.

Observation 4.2.1 *For any node μ in a packet time digraph $f_s(\mu) \le n + f_r(\mu)$.*

We again prove a technical lemma to obtain the lower bound. Note that while this lemma demonstrates a lower bound for both even and odd rings, the odd bound is not tight.

Lemma 4.2.3 *Let G be the packet time digraph of a processor bound scheme that gossips in minimal time. In addition, for each $1 \le k \le p-1$ and for each processor P_i, let $\mu(i,j(i,k))$ be the vertex with smallest start time for which both $f_r(\mu(i,j(i,k))) \ge kn$ and $f_s(\mu(i,j(i,k))) \ge kn$. There is then a path $\mathcal{B}_{i,k}$ of at least $k+1$ vertices and whose last vertex is $\mu(i,j(i,k))$ that contains a total of at least $f(\mu(i,j(i,k)))$ units*

Proof: As in the proof of Lemma 4.2.2, we prove by induction on k.

The basis is again straightforward. When $k = 1$, and for any processor P_i, let $\mathcal{B}_{i,1} = \mu(i,1)\ldots\mu(i,j(i,1))$. The path $\mathcal{B}_{i,1}$ must contain at least two packets: one to send, and one to receive; and there are exactly $f(\mu(i,j(i,1)))$ message units sent by packets corresponding to vertices of the path.

Suppose the lemma holds for some particular value of k. Then, for each processor P_i, there is a path $\mathcal{B}_{i,k}$ that ends at vertex $\mu(i,j(i,k))$ of at least k vertices containing at least $f(\mu(i,j(i,k)))$ units.

Let P_i be a processor in the ring, and let $\mu(i,j(i,k+1))$ be the vertex as defined in the statement of the lemma.

We divide into two cases:

1. Suppose $\mu(i,j(i,k+1)) \neq \mu(i,j(i,k))$. We let

   $$\mathcal{B}_{i,k+1} = \mathcal{B}_{i,k}\mu(i,j(i,k)+1)\ldots\mu(i,j(i,k+1))$$

   to satisfy the lemma: the path $\mu(i,j(i,k)+1)\ldots\mu(i,j(i,k+1))$ will necessarily be composed of at least one packet, and contain at least $f(\mu(i,j(i,k+1))) - f(\mu(i,j(i,k)))$ message units.

2. Suppose $\mu(i,j(i,k+1)) = \mu(i,j(i,k))$. Then by the construction of $\mu(i,j(i,k))$, it must be the case that either $f_r(\mu(i,j(i,k)-1)) < kn$ or $f_s(\mu(i,j(i,k)-1)) < kn$, but not both. The former cannot be true by Observation 4.2.1. As a consequence, $\mu(i,j(i,k))$ must be sent by P_i, and contain strictly more than n message units.

   Let $(\mu(i-1,j_1),\mu(i,j(i,k)))$ be the predecessor edge leading into $\mu(i,j(i,k))$. Vertex $\mu(i-1,j_1)$ must exist, since there must be at least one packet sending information into P_i to account for the $(k+1)n$ units P_i has received before $\mu(i,j(i,k))$ starts being transmitted! We note by Observation 4.2.1 that

   $$f_s(\mu(i-1,j_1)) = f_r(\mu(i,j(i,k)-1)) \geq (k+1)n$$

   and thus

   $$f_r(\mu(i-1,j_1)) \geq kn > f_s(\mu(i,j(i,k)-1)).$$

This implies both

$$f(\mu(i-1,j_1)) > f(\mu(i,j(i,k)-1))$$

and

$$f(\mu(i-1,j_1)) + |\mathrm{p}(\mu(i,j(i,k+1)))| > f(\mu(i,j(i,k+1))).$$

Vertex $\mu(i-1,j_1)$ is thus a candidate for $\mu(i-1,j(i-1,k))$, and can certainly occur no earlier in time. We let

$$\mathcal{B}_{i,k+1} = \mathcal{B}_{i-1,k}\mu(i-1,j(i-1,k)+1)\ldots\mu(i-1,j_1)\mu(i,j(i,k+1)).$$

The path $\mathcal{B}_{i-1,k}\mu(i-1,j(i-1,k)+1)\ldots\mu(i-1,j_1)$ is thus composed of at least $k+1$ vertices (more if $\mu(i-1,j_1) \neq \mu(i-1,j(i-1,k))$) containing no less than $f(\mu(i-1,j_1))$ units. Appending $\mu(i,j(i,k+1))$ adds a packet and $|\mathrm{p}(\mu(i,j(i,k+1)))|$ units, which ensures that $\mathcal{B}_{i,k+1}$ contains more than $f(\mu(i,j(i,k+1)))$ message units.

In both cases, the induction hypothesis is seen to be true. The lemma must hold, for all processors P_i and $1 \leq k \leq p-1$ □

The following corollary restates the result of the lemma in a more usable form.

Corollary 4.2.1 *In a unidirectional processor bound ring of p processors at least $p\beta + 2(p-1)n\tau$ time is required to gossip.*

Proof: The lemma provides that each final packet sent by any processor is part of a path that is composed of at least p packets which contain a total of no less than $2(p-1)n$ units. As a consequence, these final packets must finish transmitting no sooner than time $p\beta + 2(p-1)n\tau$. □

Theorem 4.2.2 thus follows as a corollary of the lemma and the upper bound.

Theorem 4.2.2 *For even integers p, and under the linear cost model, $p\beta + 2(p-1)\tau$ time is necessary and sufficient for gossiping in the unidirectional processor bound ring of p processors.*

We can tighten the bound for the odd case by a small post-analysis of the proof of Lemma 4.2.3. We employ the notion of "bandwidth dominance" [14] in the context of Lemma 4.2.3.

Each processor must send $(p-1)n$ units. Thus, a total of $p(p-1)n$ units must be sent, if all packets in the scheme are totaled. When p is odd, at most $(p-1)/2$ processors can be sending at the same time: thus, all paths of p packets (as found by the lemma) will require at least $p\beta + 2pn\tau$ time to complete. In addition, one of these paths must end "early", since at most $p-1$ of the paths can be active at the end of the gossiping scheme. This path may therefore be augmented by the size of the smallest "final" packet. This one augmented path will thus require at least $(p+1)\beta + 2pn\tau$ time to complete.

Together with the upper bound, this demonstrates Theorem 4.2.3.

Theorem 4.2.3 *For odd integers p, and under the linear cost model, $(p+1)\beta + 2p\tau$ time is necessary and sufficient for gossiping in the unidirectional processor bound ring of p processors.*

4.2.3 The full duplex link bound ring

Since we can pretend that a bidirectional link bound ring is two independent unidirectional rings, we could halve each message and send in the two directions. This provides an immediate upper bound of:

$$g_{F*}(C_p, n) \leq g_*(\vec{C}_p, \lceil n/2 \rceil) = (p-1)\left(\beta + \lceil n/2 \rceil \tau\right).$$

In fact, a better bound is possible. We will show

$$g_{F*}(C_p, n) = \lfloor p/2 \rfloor \beta + \left\lceil \frac{1}{2}((p-1)n) \right\rceil \tau.$$

The upper bound follows from Fraigniaud and Lazard [8]. From the point of view of a single processor, the scheme essentially halves the ring by sending the entire message in both directions rather than halving the message, and sending one half in each direction. Each message M_i thus only needs to be transferred over at most $\lfloor p/2 \rfloor$ links. Moreover, when p is even, $P_{p/2+i}$ need not receive M_i in total from both

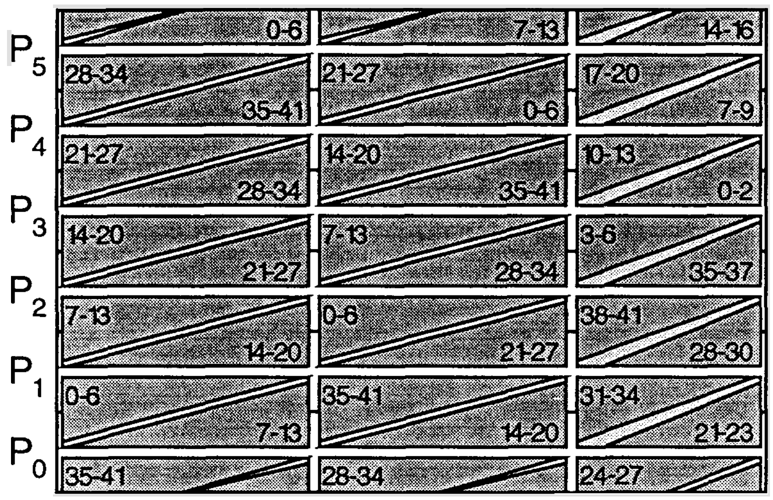


Figure 4.2: Example: Gossiping messages of size 7 in C_6

directions: as a consequence the last round need only be of size $\lceil n/2 \rceil$, rather than n. Figure 4.2 illustrates the scheme.

Consider a gossip scheme $\mathcal{S}$ that sends messages of size n from each processor that takes minimum time. The two directions are essentially independent: no transfer in one direction will ever affect a transfer in the other. As a consequence, we can examine the two unidirectional transfers separately.

Each processor P_i must receive a total of $n(p-1)$ units of information from its neighbours. It must thus receive at least $\left\lceil \frac{1}{2}(n(p-1)) \right\rceil$ message units from one of the neighbours. We build an induction argument based on a version of the packet time digraph to demonstrate that each processor must receive these $\left\lceil \frac{1}{2}(n(p-1)) \right\rceil$ units in at least $\lfloor p/2 \rfloor$ packets.

For the link bound case, we do not need to have two nodes corresponding to each packet of the transmission. We construct a packet time digraph on both the set of packets $\Lambda_f = \{\lambda_f(i,j) \mid 0 \leq i \leq p \text{ and } 1 \leq j \leq \nu_f(i)\}$ sent in the forward direction (P_i to $P_{(i+1)\text{mod}p}$) and the set of packets Λ_b sent in the backward direction (P_i to $P_{(i-1)\text{mod}p}$).

Our forward packet time digraph G_f is thus defined:

- The vertices of G_f are the set of packets Λ_f.
- For each packet $\lambda(i,j)$ we define: an *Adjacent packet* edge: $(\lambda(i,j), \lambda(i,j+1))$, if $j < \nu_f(i)$; and a *Predecessor* edge: $(\lambda(i-1,\hat{\jmath}), \lambda(i,j))$, if $j > 1$, where $\hat{\jmath}$ is the maximal index so that $st(\lambda(i-1,\hat{\jmath})) + t(\lambda(i-1,\hat{\jmath})) \leq st(\lambda(i,j))$.

  The edge set of G_f is simply the collection of all adjacent packet and predecessor edges for all packets.

The backward digraph is defined similarly.

For each processor P_i, we let $j_f(i,k)$ denote the index of the packet $\lambda_f(i, j_f(i,k))$ that contains the $\left\lceil \frac{n}{2}\left(p-1-2\left(\lfloor p/2 \rfloor - k\right)\right)\right\rceil^{\text{th}}$ message unit P_i sends in the forward direction. We define $j_b(i,k)$ similarly, as well.

The following lemma provides the lower bound as a corollary.

Lemma 4.2.4 *In the packet time digraph G_f (G_b) for any minimal gossiping scheme under the full duplex link bound constraint, for each $1 \leq k \leq \lfloor p/2 \rfloor$ and each processor P_i for which $j_f(i,k)$ $(j_b(i,k))$ is defined, there is a path $\mathcal{A}_{i,k}$ in G_f (G_b) that ends at the packet $\lambda_f(i, j_f(i,k))$ $(\lambda_b(i, j_b(i,k)))$ such that*

$$t(\mathcal{A}_{i,k}) \geq k\beta + \left\lceil \frac{1}{2}\left(n\left(p-1-2\left(\lfloor p/2 \rfloor - k\right)\right)\right)\right\rceil \tau.$$

Proof: We prove the lemma by induction on k. We will speak of the digraph G_f, since the proof for the backward graph only requires textual substitution.

For $k = 1$, The value $\left\lceil \frac{n}{2}\left(p-1-2\left(\lfloor p/2 \rfloor - k\right)\right)\right\rceil$ is at least $\lceil n/2 \rceil$. Each processor P_i will have had to have received at least one packet in order to have received even one message unit. The basis holds.

Suppose the lemma holds for some particular value of k.

Let P_i be any processor in the ring for which $j_f(i,k+1)$ is defined. In order to send $\lambda_f(i, j(i,k+1))$ on to P_{i+1}, processor P_i will have to have received $\left\lceil \frac{n}{2}\left(p-1-2\left(\lfloor p/2 \rfloor - k\right)\right)\right\rceil$ message units from P_{i-1}.

As a consequence, if $\lambda(i-1,\hat{\jmath})$ is the predecessor of $\lambda_f(i,j_f(i,k+1))$, then $\hat{\jmath} \geq j(i-1,k)$. So

$$\mathcal{A}_{i,k+1} = \mathcal{A}_{i-i,k}\lambda_f(i-1,j_f(i-1,k)+1)\ldots\lambda(i-1,\hat{\jmath})\lambda_f(i,j_f(i,k+1))$$

satisfies the lemma.

The induction hypothesis holds for value $k+1$. The lemma thus holds for all appropriate values of k, by the principle of mathematical induction. □

We've established that for every processor P_i that sends as many as $\left\lceil \frac{1}{2}\left(n\left(p-1\right)\right)\right\rceil$ message units to a neighbour, those units must have been packaged in at least $\lfloor p/2 \rfloor$ packets. Theorem 4.2.4 follows as a corollary.

Theorem 4.2.4 *Under the linear cost model,* $\lfloor p/2 \rfloor \beta + \frac{1}{2}\left(\lceil n\left(p-1\right)\rceil \tau\right)$ *time is necessary and sufficient for gossiping in the bidirectional link bound ring of p processors.*

4.2.4 The full duplex processor bound ring

Fraigniaud and Lazard [8] provide a bound of

$$g_{F1}(C_p,n) \leq \frac{1}{2}\left(p\beta\right) + (p-1)n\tau$$

for even p, and

$$g_{F1}(C_p,n) \leq \lceil 3p/4 \rceil \beta + \lceil 3pn/2 \rceil \tau$$

for odd p.

We can certainly halve the message, and use two simultaneous schemes for the unidirectional ring, where the "backward" scheme merely piggy-backs in the unused reverse capacity of the packet transfers of the "forward" scheme. In the case of the even ring, at least:

$$g_{F1}(C_p,n) \leq p\beta + 2(p-1)\lceil n/2 \rceil \tau.$$

But in the case of the odd ring, things are not so clear. The reverse scheme will not have finished at the same time as the forward one. The "rotating idle processor"

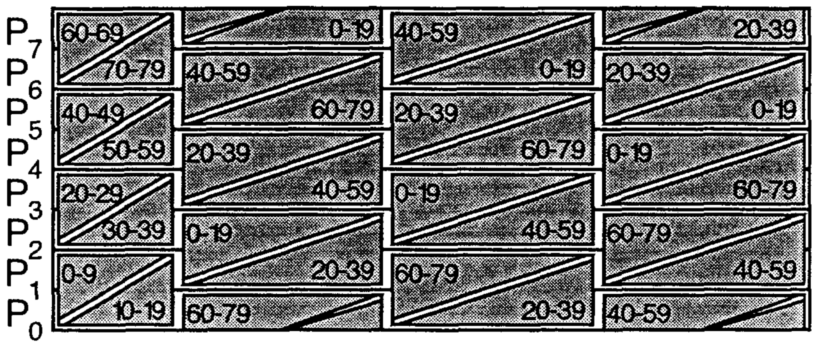


Figure 4.3: Example: Gossiping messages of size 10 in C_8

paradigm interferes with the progress of the scheme. Regardless, these schemes are not as good as the Fraigniaud and Lazard bounds.

For the even ring, the scheme that meets the bound in Fraigniaud and Lazard is due to Saad and Schultz [22], and is as follows:

- The scheme is divided into $p/2$ rounds, numbered 0 to $p/2 - 1$.
- In even rounds, P_i communicates with P_{i-1}, while in odd rounds, P_i and P_{i+1} exchange information.
- In each round, a processor sends every unit of information it knows that it has never sent before. Thus, round 1 sends packets of length n, while every other round sends packets of length $2n$.

Figure 4.3 illustrates the scheme.

For the odd ring C_p, a good scheme that takes time $(\lceil p/2 \rceil + 1)\beta + (p+1)n\tau$ is:

- The scheme is divided into $\lceil p/2 \rceil + 1$ rounds, numbered 0 to $\lceil p/2 \rceil$. Round 0 and $\lceil p/2 \rceil$ send packets of length n, while the other rounds send packets of length $2n$. We'll casually refer to the *length* of a round, and mean this size.
- In round i, processor $P_{i \bmod p}$ is idle. And thus, for $1 \le j \le \lceil p/2 \rceil$, processor $P_{(i+2j-1) \bmod p}$ communicates with $P_{(i+2j) \bmod p}$.

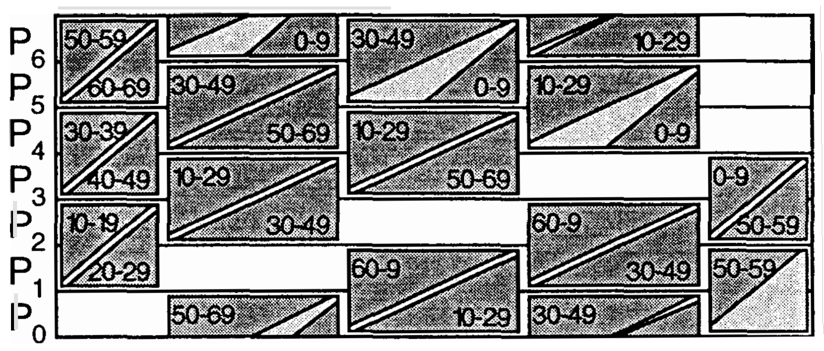


Figure 4.4: Example: Gossiping messages of size 10 in C_7

- A processor P_i keeps two queues for the information it must send: one for each direction. Initially, both queues contain M_i. Information received is pushed onto the end of the queue. And in each round, a processor sends either the entire contents of the queue for the relevant direction, or the first n or $2n$ (corresponding to the length of the round) units in the queue, whichever is smaller.

Figure 4.4 illustrates the scheme.

Observation 4.2.2

$$g_{F1}(C_{2m+1}, n) \leq (m+2)\beta + (2m+2)n\tau$$

We will provide what is a corresponding lower bound in the even case, and not a particularly good bound in the odd case.

$$g_{F1}(C_p, n) \geq \lceil p/2 \rceil \beta + (p-1)n\tau.$$

Suppose we have a minimal gossip scheme for the full duplex processor bound ring.

We need yet another variant on our digraph motif. We note that in a processor bound transmission, there is a natural twinning between packets in the forward and backward direction: when two packets are being sent along the same link at the same time. (In cases where a packet has no such twin, we create one by generating a packet of zero length as a place holder.) We label these pairs *transfers*, and let $\gamma(i,j)$ denote the j^{th} transfer (ordered by start time) over the link between P_i and $P_{(i+1)\bmod p}$.

We define, for a transfer $\gamma(i,j)$, some of the same collection of functions we have for packets. In particular:

- Size: $|\gamma(i,j)|$ denotes the length of the larger of the two packets that corresponds to the transfer.
- Start time: $st(\gamma(i,j))$ denotes the minimum of the two start times of the pair of corresponding packets. In fact, there's no reason to expect that they'd be different.

We let $\lambda'(\hat{\imath},\hat{\jmath})$ denote the $\hat{\jmath}^{\text{th}}$ packet that $P_{\hat{\imath}}$ sends (counting both directions) after adding the packets of zero length. Each λ' thus corresponds to a single transfer γ, and each γ contains two λ's: exactly the relationship that μ and λ had in the construction of the packet time digraph.

We thus construct the transfer time digraph G on the vertex set

$$\{\lambda'(i,j) \mid 0 \le i < p \text{ and } 1 \le j \le \nu'(i)\}.$$

For each vertex $\lambda'(i,j)$ we create three edges:

1. The adjacent transfer edge: $(\lambda'(i,j), \lambda'(i,j+1))$, where $j < \nu'(i)$.
2. The forward predecessor edge: $(\lambda'(i+1,\hat{\jmath}_f), \lambda'(i,j))$, where $\hat{\jmath}_f$ is as large as possible given $st(\lambda'(i+1,\hat{\jmath}_f)) < st(\lambda'(i,j))$
3. The backward predecessor edge: $(\lambda'(i-1,\hat{\jmath}_b), \lambda'(i,j))$, where $\hat{\jmath}_b$ is as large as possible given $st(\lambda'(i-1,\hat{\jmath}_b)) < st(\lambda'(i,j))$

The edge set of G is simply the collection of adjacent transfer edge, forward predecessor edge and backward predecessor edge for all vertices.

We still want to be able to speak of the total number of message units P_i has sent and received. Following the example of previous time digraphs, we let $f_s(\lambda'(i,j))$ $(b_s(\lambda'(i,j)))$ denote the total number of message units that P_i has sent in the forward (backward) direction, up to and including $\lambda'(i,j)$, and let $f_r(\lambda'(i,j))$ $(b_r(\lambda'(i,j)))$ denote the total number of units that P_i has received in the forward (backward) direction, up to and including the twin of $\lambda'(i,j)$. For convenience, we let $f(\lambda'(i,j)) = f_s(\lambda'(i,j)) + f_r(\lambda'(i,j))$ and $b(\lambda'(i,j)) = b_s(\lambda'(i,j)) + b_r(\lambda'(i,j))$.

For each processor P_i, we let $j_f(i,k)$ denote the minimal index for which both $f_r(\lambda'(i,j_f(i,k))) \geq kn$ and $f_s(\lambda'(i,j_f(i,k))) \geq kn$, and let $j_b(i,k)$ similarly denote the minimal index for which both $b_r(\lambda'(i,j_b(i,k))) \geq kn$ and $b_s(\lambda'(i,j_b(i,k))) \geq kn$.

Lemma 4.2.5 *In the transfer time digraph G of a minimal gossiping scheme for C_p under the full duplex processor bound constraint, for each $1 \leq k < \lceil p/2 \rceil$: for each processor P_i for which $j_f(i,k)$ is defined, there is a path $\mathcal{A}_{i,k}$ in G that ends at $\lambda'(i,j_f(i,k))$ that contains at least $k+1$ packets; and for each processor $P_{i'}$ for which $j_b(i',k)$ is defined, there is a path $\mathcal{B}_{i',k}$ in G that ends at $\lambda'(i,j_b(i,k))$ that contains at least $k+1$ packets.*

Proof: The forward and backward cases are entirely symmetric. Without loss of generality, we only prove the forward one. The proof follows that of Lemma 4.2.3.

The basis, when $k = 1$ is straightforward. In order for both $f_s(\lambda'(i,j_f(i,1))) > 0$ and $f_r(\lambda'(i,j_f(i,1))) > 0$, processor P_i must have been involved in two transfers. $\mathcal{A}_{i,1} = \lambda'(i,1)\ldots\lambda'(i,j_f(i,1))$ satisfies the lemma.

Suppose the lemma holds true for some specific value k.

Let P_i be any processor for which $j_f(i,k+1)$ is defined.

If $j_f(i,k+1) > j_f(i,k)$, then

$$\mathcal{A}_{i,k+1} = \mathcal{A}_{i,k}\lambda'(i,j_f(i,k)+1)\ldots\lambda'(i,j_f(i,k+1))$$

satisfies the lemma.

Suppose, then, that $j_f(i,k+1) = j_f(i,k)$. By the construction of $j_f(i,k+1)$, it must be the case that either $f_r(\lambda'(i,j_f(i,k+1)-1)) < kn$ while $f_s(\lambda'(i,j_f(i,k+1)-1)) \geq (k+1)n$ or $f_s(\lambda'(i,j_f(i,k+1)-1)) < kn$ while $f_r(\lambda'(i,j_f(i,k+1)-1)) \geq (k+1)n$. Since for any packet λ', $f_s(\lambda') \leq f_r(\lambda') + n$, the former case cannot occur. As a consequence, $\lambda'(i,j_f(i,k+1))$ must be sent by P_i to P_{i+1}.

Let $(\lambda'(i-1, j_1), \lambda'(i, j_f(i, k+1)))$ be the forward predecessor edge leading into $\lambda'(i, j_f(i, k+1))$. Node $\lambda'(i-1, j_1)$ must exist, since $f_r(\lambda'(i, j_f(i, k+1)-1)) \geq (k+1)n$.

Now, $f_s(\lambda'(i-1, j_1)) = f_r(\lambda'(i, j_f(i, k+1)-1)) \geq (k+1)n$, and thus $f_r(\lambda'(i-1, j_1)) \geq kn$. As a consequence, $j_f(i-1, k) \leq j_1$. And thus,

$$\mathcal{A}_{i,k+1} = \mathcal{A}_{i-1,k}\lambda(i-1, j(i-1,k)+1)\ldots\lambda(i-1,j_1)\lambda(i, j(i,k+1))$$

satisfies the lemma.

In either case, the induction hypothesis is verified. The lemma must hold for all processors P_i, and appropriate values of k. ☐

The lower bounds are not quite immediate. We need to demonstrate that there is a processor P_i that either both sends at least $(\lceil p/2 \rceil - 1)\,n$ message units in the forward direction and receives at least $(\lceil p/2 \rceil - 1)\,n$ units in the forward direction, or the equivalent statement with "backward" substituted for "forward".

In an attempt to simplify (or at least, compact) the notation, let $f_r(i)$ denote the total number of message units $f_r(\lambda'(i, \nu'(i)))$ processor P_i receives in the forward direction. Let $f_s(i)$, $b_r(i)$, and $b_s(i)$ be defined similarly. Then we need to demonstrate a processor for which

$$f_r(i) \geq (\lceil p/2 \rceil - 1)\,n \quad \text{and} \quad f_s(i) \geq (\lceil p/2 \rceil - 1)\,n$$

or

$$b_r(i) \geq (\lceil p/2 \rceil - 1)\,n \quad \text{and} \quad b_s(i) \geq (\lceil p/2 \rceil - 1)\,n.$$

This is equivalent to Lemma 4.2.6.

Lemma 4.2.6 *In any gossip scheme in a bidirectional ring C_p, for all pairs of processors P_i and $P_{(i+1)\bmod p}$ either*

$$f_r(i) \geq \lceil (p-2)n/2 \rceil \qquad \text{and} \qquad f_r((i+1) \bmod p) \geq \lceil (p-2)n/2 \rceil \tag{4.1}$$

or

$$b_r(i) \geq \lceil (p-2)n/2 \rceil \qquad \text{and} \qquad b_r((i+1) \bmod p) \geq \lceil (p-2)n/2 \rceil \tag{4.2}$$

hold. In addition, there exists a pair of processors $P_{i'}$ and $P_{(i'+1) \bmod p}$ for which either

$$f_r(i') \geq (\lceil p/2 \rceil - 1)\, n \qquad \text{and} \qquad f_r((i'+1) \bmod p) \geq (\lceil p/2 \rceil - 1)\, n \tag{4.3}$$

or

$$b_r(i') \geq (\lceil p/2 \rceil - 1)\, n \qquad \text{and} \qquad b_r((i'+1) \bmod p) \geq (\lceil p/2 \rceil - 1)\, n \tag{4.4}$$

Proof: Since each processor in a gossip scheme must receive $(p-1)n$ units of information, $b_r(i) + f_r(i) = (p-1)n$ for all processors P_i.

Regardless of the parity of p, we can color the processors: P_i black if $f_r(i) < \left\lceil \frac{1}{2}((p-2)n) \right\rceil$; P_i white if $b_r(i) < \left\lceil \frac{1}{2}((p-2)n) \right\rceil$; and P_i grey otherwise. Two adjacent processors both not colored white satisfy 4.2; and two adjacent processors both not colored black satisfy 4.1. We cannot have two adjacent processors colored respectively black and white: if P_i is black and P_{i+1} white then:

$$\lfloor pn/2 \rfloor \geq f_r(i) + n \geq f_s(i) = f_r(i+1) = (p-1)n - b_r(i+1) > \lfloor pn/2 \rfloor .$$

A similar contradiction arises when P_i is white and P_{i+1} black. Every adjacent pair of processors must thus satisfy either 4.1 or 4.2.

If p is even, then $\lceil p/2 \rceil - 1 = (p-2)/2$ and the lemma holds.

Suppose p is odd, and thus $\lceil p/2 \rceil - 1 = (p-1)/2$. We again color the processors: P_i white if $f_r(i) \geq (p-1)n/2$, and P_i black otherwise—when $b_r(i) \geq (p-1)n/2$. We must have two adjacent processors colored with the same color: if white, then 4.3 holds; if black then 4.4.

The lemma is thus established for any gossip scheme in C_p. □

Theorem 4.2.5 follows as a corollary of Lemma 4.2.6 and Lemma 4.2.5.

Theorem 4.2.5 *For even integers p, and under the linear cost model, $\frac{p}{2}\beta + (p-1)n\tau$ time is necessary and sufficient for gossiping in the bidirectional processor bound ring of p processors.*

We also obtain a lower bound for the odd case as a corollary:

Corollary 4.2.2

$$g_{F1}(C_{2m+1}, n) \geq (m+1)\beta + 2mn\tau$$

We do not believe this bound is optimal; rather, we expect it will be improved in future.

4.2.5 The half duplex bidirectional ring

The schemes for the unidirectional ring apply without change. And while we could employ the appropriate full duplex gossip schemes by replacing full duplex packets with a pair of half duplex ones, the bounds thus acquired are no better than simply employing the unidirectional bounds. We thus have the following upper bounds.

Observation 4.2.3

$$g_{H*}(C_p, n) \leq (p-1)(\beta + n\tau)$$

Observation 4.2.4

$$g_{H1}(C_p, n) \leq 2\lceil p/2 \rceil \beta + (4\lceil p/2 \rceil - 2)n\tau$$

While we have no corresponding lower bounds, we can provide one last upper bound for the even ring C_p:

$$g_{H*}(C_p, n) \leq (p/2+1)\beta + (p-1)n\tau.$$

- The gossip scheme is divided into $p/2+1$ phases numbered from 0 to $p/2$. Phase 0 sends packets of length n; phase $p/2$ sends packets of length $\lceil n/2 \rceil$ for processors with odd index, and $\lfloor n/2 \rfloor$ for processors with even index; phase $p/2-1$ sends packets of length $\lceil 3n/2 \rceil$ for processors with even index, and $\lfloor 3n/2 \rfloor$ for processors with odd index; while the remainder send packets of length $2n$. When we refer to the length of a phase, we mean the length of the packets sent in that phase.

  The last two phases are not quite disjoint. Processors thus make the decision of when to advance to the next phase independent of external synchronization.

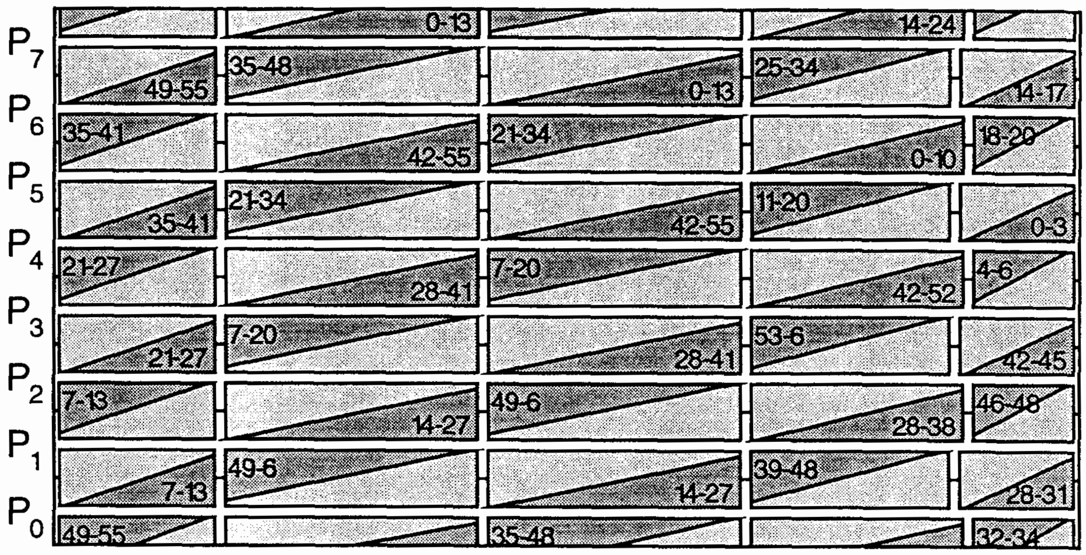


Figure 4.5: Example: Gossiping messages of size 7 in C_8

- In phase i, processors $P_{(i+2j) \bmod p}$ (for $0 \le j < p/2$) receive from their neighbours, while the other processors send to both neighbours.

- Each processor P_i considers the collection of all messages totally ordered: In the forward direction

$$M_i \prec M_{(i-1) \bmod p} \prec M_{(i-2) \bmod p} \prec \ldots \prec M_{(i+1) \bmod p},$$

  while within each message $b(j,n) \prec b(j,n-1) \prec \ldots \prec b(j,1)$. In the backward direction,

$$M_i \prec M_{(i+1) \bmod p} \prec M_{(i+2) \bmod p} \prec \ldots \prec M_{(i-1) \bmod p}$$

  while within each message $b(j',1) \prec b(j',2) \prec \ldots \prec b(j',n)$.

- If a processor P_i is sending in a given phase of length ℓ, it sends to both its neighbours. In each direction, it sends the first ℓ bits in the respective total order for that direction that have not already been sent.

Figure 4.5 illustrates the scheme.

4.3 Summary

We have examined the gossip problem for six variants of the cycle: unidirectional, full duplex or half duplex, and link bound or processor bound. Matching lower and upper bounds have been found for the two unidirectional rings, the link bound full duplex ring, as well as the full duplex even ring under the processor bound constraint. We have presented upper bounds for the remaining cases.

The upper bound for the full duplex odd ring under the processor bound constraint is a significant improvement over the existing best upper bound. It is entirely possible that a matching lower bound could be constructed. As it is, we present a lower bound, but not a good one: the lower bound is considerably softer than the upper.

The half duplex bounds are (with the exception of a not-quite-synchronous scheme for the full duplex even ring under the link bound constraint) merely the application of the unidirectional ring bounds to the half duplex ring. Yet while these unidirectional schemes give no poorer results than doubling the packets of the full duplex schemes, it is likely that improvement can be made.

For a tabular summary of the bounds, see Table 5.1.

Chapter 5

Conclusion

5.1 The results

The three preceding chapters have successively examined problems of sending a message between two processors, broadcasting a message from a single processor to all its peers, and gossiping messages from every processor to all other processors. In each of the three chapters, we present both lower and upper bounds. All of the lower bounds we present are new, while some of the upper bounds (typically those that we can match with a lower bound) are from the literature.

In one respect, the lower bounds are particularly significant. They are the first instances of exact bounds under the linear cost model for either broadcasting or gossiping, and after [9] and [10] the third instance of matching lower and upper bounds under the linear cost model for any communication problem.

In another respect, the lower bounds are not particularly interesting. It was well known that Saad and Schultz' pipelined schemes (at least for sending) were optimal within a multiplicative factor of two. Simply, the lower bounds themselves are nowhere near as important as the fact of their proof.

Our primary results, then, are the proofs of Theorems 2.3.1 and 2.4.1 in Chapter 2. These prove the matching bounds for the sending problems. In exact terms:

$$S_*(n,m) = \min_{1\le k\le n} \left\{ \left(\left\lceil \frac{n}{k} \right\rceil + m - 1 \right) \beta + ((m-1)k + n)\tau \right\}$$

$$S_1(n,m) = \min_{1\le k\le n}\left(\left(2\left\lceil\frac{n}{k}\right\rceil + m - 2\right)\beta + ((m-2)k + 2n)\tau\right)$$

and in closed form approximations:

$$\begin{aligned}\left(\sqrt{(m-1)\beta}+\sqrt{n\tau}\right)^2 &\le S_*(n,m)\\ S_*(n,m) &< \left(\sqrt{(m-1)\beta}+\sqrt{(n-1)\tau}\right)^2 + \beta + m\tau\\ &\text{and}\\ \left(\sqrt{(m-2)\beta}+\sqrt{2n\tau}\right)^2 &\le S_1(n,m)\\ S_1(n,m) &< \left(\sqrt{(m-2)\beta}+\sqrt{(2n-2)\tau}\right)^2 + 2\beta + m\tau.\end{aligned}$$

Secondary results are the bounds for the broadcasting and gossiping problems. These are derivative of the sending bounds in two ways. First, the upper bounds, particularly for broadcasting, employ the optimal time $T_X(n,m,k)$ to pipeline n bits over m links under model X in packets of size k—a concept designed to match the optimal sending bound for some value of k. Second, the proofs of the lower bounds for broadcasting and gossiping make extensive use of the mechanics behind the sending proofs.

Table 5.1 summarizes the contributions we have made to the three problems considered.

	Broadcast Bounds					Gossip Bounds		
Model	$\vec{C}_p$	C_{2m}	C_{2m+1}	K_{2m}	K_{2m+1}	$\vec{C}_p$	C_{2m}	C_{2m+1}
$F*$	[16]	3.2.1	3.2.1	3.3.1	3.3.1	[22]	[8]	[8]
	ℓ2.3	ℓ3.2.1	ℓ3.2.1			ℓ4.2.1	ℓ4.2.3	ℓ4.2.3
$F1$	2.4	3.2.2	3.2.2	3.3.4	[8],3.3.4	4.2.2	[8]	4.2.4
	ℓ2.4					ℓ4.2.2	ℓ4.2.4	
$H*$	($\equiv F*$)	3.2.3	[22]+	3.3.2	3.3.2	($\equiv F*$)	4.2.5	[22]
$H1$	($\equiv F1$)	3.2.3	[22]+	3.3.5	3.3.5	($\equiv F1$)	4.2.5	4.2.5

Table 5.1: Summary of Bounds

Entries in Table 5.1 refer, by default, to the best upper bound for the problem. They are either a reference to a section of this work, or a citation. A plus by a citation indicates some small enhancement to the bound cited on our part. An ℓ prior to a reference indicates a lower bound that matches the corresponding upper bound in the same cell of the table.

To attempt to ameliorate the difficulty in getting an intuitive feel for the bounds we have produced, we present the following tables of timings. We fix the parameters $\beta = 272\mu s$ and $\tau = 0.4\mu s$ per byte: values that are representative of the Intel *i*PSC/860 [5]. Entries in Table 5.2 and Table 5.3 represent the times required to broadcast messages of 1023 bytes and 32767 bytes in the given topologies, while the entries of Table 5.4 represent the times required to gossip messages of the same sizes. If a cell contains an entry of the form $a \rightarrow b$, the first value (a) represents the time that results from the best previously published bound, while the second value (b) represents the result of applying a bound in this work. The rest of the entries represent times that result from the bounds in the literature.

Model	Bytes	$\vec{C}_{10}$	C_{10}	C_9
$F*$	1023	4492.4	$3743.8 \rightarrow 2246.4$	$3369.4 \rightarrow 2041.6$
	32767	25967.6	$16284.8 \rightarrow 12984.0$	$15524.4 \rightarrow 12504.0$
$F1$	1023	5244.0	$4492.4 \rightarrow 2858.8$	$7861.8 \rightarrow 2858.8$
	32767	42448.4	$25967.6 \rightarrow 21748.4$	$41656.4 \rightarrow 23044.8$
$H*$	1023	4492.4	3267.2	2383.6
	32767	25967.6	$22914.8 \rightarrow 20390.4$	20464.0
$H1$	1023	5244.0	3813.6	2724.8
	32767	42448.4	$37984.8 \rightarrow 36403.2$	34310.4

Table 5.2: Broadcast times for C_p, in μs, given $\beta = 272\mu s$ and $\tau = 0.4\mu s$/byte.

Model	Bytes	K_{10}	K_9
$F*$	1023	$635.2 \rightarrow 626.4$	$646.4 \rightarrow 635.2$
	32767	$3000.8 \rightarrow 2886.0$	$3272.8 \rightarrow 3129.2$
$F1$	1023	$4492.4 \rightarrow 2588.0$	$4084.0 \rightarrow 2588.0$
	32767	$25967.6 \rightarrow 21748.4$	25005.6
$H*$	1023	$1021.2 \rightarrow 681.2$	$1123.2 \rightarrow 681.2$
	32767	$4909.6 \rightarrow 4126.8$	$6492 \rightarrow 4432.8$
$H1$	1023	$8168.0 \rightarrow 2588.0$	$8168.0 \rightarrow 2588.0$
	32767	$50008.8 \rightarrow 32371.2$	$50008.8 \rightarrow 32371.2$

Table 5.3: Broadcast times for K_p, in μs, given $\beta = 272\mu s$ and $\tau = 0.4\mu s$/byte.

Model	Bytes	$\vec{C}_{10}$	$\vec{C}_9$	C_{10}	C_9
$F*$	1023	6130.8	5449.6	3473.6	2724.8
	32767	120409.2	107030.4	60612.8	53515.2
$F1$	1023	10085.6	10085.6	5314.8	$7428.4 \rightarrow 5724.0$
	32767	238642.4	238642.4	119593.2	$178846.0 \rightarrow 132700.0$
$H*$	1023	6130.8	5449.6	$6130.8 \rightarrow 5586.8$	5449.6
	32767	120409.2	107030.4	$120409.2 \rightarrow 119865.2$	107030.4
$H1$	1023	10085.6	10085.6	10085.6	10085.6
	32767	238642.4	238642.4	238642.4	238642.4

Table 5.4: Gossip times for C_p, in μs, given $\beta = 272\mu s$ and $\tau = 0.4\mu s$/byte.

There are three additional minor results buried in the text that are not represented in the table. First, matching lower and upper bounds for the problem of sending a message in the bidirectional ring—Observation 2.5.1. Second, an improved upper bound for broadcasting in the hypercube under the full duplex processor bound constraint—Proposition 3.3.3. Last, an upper bound for broadcasting in the folded hypercube under the half duplex processor bound constraint–Proposition 3.3.4.

5.2 Future directions

Where does one go from here? The following avenues await:

- The lower bounds presented in Chapter 4 for the various flavours of the gossiping problem follow a common thread: one constructs a digraph, then applies an ugly technical lemma that proves a lower bound. I have crafted a different lemma for each case, yet they are all remarkably similar. A common lemma that would work for all cases would be a big improvement.

- In Table 5.1, there are many lower bounds that appear feasible if sufficient effort were expended. The lower bound for the $(H*, C_{2m})$ gossiping cell looks quite straightforward, for example. On the other hand, the $(F1, C_{2m})$ broadcasting cell has the appearance of tractability, but has resisted more than two (admittedly part-time) years of my best efforts.

- One might look for lower bounds on gossiping in the complete graph under the linear cost model. Or lower bounds for broadcasting in topologies other than the ring or complete graph.
- One might equally look to other communications modes: circuit switched transmissions, for example.
- Finally, an examination of the DMA-bound constraints Fk and Hk might prove interesting.

Appendix A

Extra Algebra

These appendices are not really intended for casual reading. The four sections provide explicit derivations of the closed forms employed in the text.

A.1 The upper bound of Saad and Schultz

Saad and Schultz [22] provide an upper bound on the problem of sending n bits over m links of

$$U(n,m) = \min_{1 \leq p \leq n} (p+m-1)\left(\beta + \left\lceil \frac{n}{p} \right\rceil \tau\right),$$

where p is the number of packets into which the message has been divided. Ignoring the "ceiling" function gives

$$U(n,m) \geq \min_{1 \leq p \leq n} (p+m-1)\left(\beta + \frac{n\tau}{p}\right).$$

Replacing the integral variable p with a continuous x, and differentiating:

$$\frac{\partial}{\partial x}\left((x+m-1)\left(\beta + \frac{n\tau}{x}\right)\right) = \beta - \frac{(m-1)\,n\tau}{x^2}.$$

The continuous version of the function, then, is minimized when

$$x = \sqrt{\frac{(m-1)\,n\tau}{\beta}},$$

which gives:

$$\begin{aligned}
U(n,m) &\geq \left(\sqrt{\frac{(m-1)\,n\tau}{\beta}} + m - 1\right)\left(\beta + \sqrt{\frac{n\beta\tau}{m-1}}\right) \\
&= (m-1)\,\beta + (m-1)\sqrt{\frac{n\beta\tau}{m-1}} + \beta\sqrt{\frac{(m-1)\,n\tau}{\beta}} \\
&\quad + \sqrt{\left(\frac{(m-1)\,n\tau}{\beta}\right)\left(\frac{n\beta\tau}{m-1}\right)} \\
&= (m-1)\,\beta + 2\sqrt{(m-1)\,n\beta\tau} + n\tau \\
&= \left(\sqrt{(m-1)\,\beta} + \sqrt{n\tau}\right)^2.
\end{aligned}$$

This value is a lower bound on Saad and Schultz' upper bound. As we will see in the next section, it is also a lower bound to the sending problem.

To achieve an upper bound, the ceiling function can be replaced:

$$U(n,m) \leq \min_{1\leq p\leq n} (p+m-1)\left(\beta + \frac{(n+p-1)\,\tau}{p}\right).$$

Again replacing the integral variable p with a continuous x, and differentiating:

$$\frac{\partial}{\partial x}\left((x+m-1)\left(\beta+\tau+\frac{(n-1)\,\tau}{x}\right)\right) = \beta+\tau-\frac{(m-1)\,(n-1)\,\tau}{x^2}.$$

The continuous version of the function is minimized when

$$x = \sqrt{\frac{(m-1)\,(n-1)\,\tau}{\beta+\tau}}.$$

The discrete version, then, will be no greater than when either

$$p = \left\lceil \sqrt{\frac{(m-1)\,(n-1)\,\tau}{\beta+\tau}} \right\rceil,$$

or

$$p = \left\lfloor \sqrt{\frac{(m-1)\,(n-1)\,\tau}{\beta+\tau}} \right\rfloor.$$

One could, in fact, argue that one of these values, 1, or n will be minimum: that won't be necessary for our purposes. In particular, we have $U(n,m)$

$$\begin{aligned}
&\leq \left(\left\lceil\sqrt{\frac{(m-1)(n-1)\tau}{\beta+\tau}}\right\rceil + m - 1\right)\left(\beta+\tau+\frac{(n-1)\tau}{\left\lceil\sqrt{\frac{(m-1)(n-1)\tau}{\beta+\tau}}\right\rceil}\right) \\
&= (m-1)(\beta-\tau)+(n-1)\tau+\frac{(m-1)(n-1)\tau}{\left\lceil\sqrt{\frac{(m-1)(n-1)\tau}{\beta+\tau}}\right\rceil} \\
&\quad + (\beta+\tau)\left\lceil\sqrt{\frac{(m-1)(n-1)\tau}{\beta+\tau}}\right\rceil \\
&< (m-1)(\beta-\tau)+(n-1)\tau+\frac{(m-1)(n-1)\tau}{\sqrt{\frac{(m-1)(n-1)\tau}{\beta+\tau}}} \\
&\quad + (\beta+\tau)\left(1+\sqrt{\frac{(m-1)(n-1)\tau}{\beta+\tau}}\right) \\
&= m(\beta+\tau)+(n-1)\tau+2\sqrt{(m-1)(n-1)(\beta+\tau)\tau} \\
&= \left(\sqrt{(m-1)(\beta+\tau)}+\sqrt{(n-1)\tau}\right)^2+\beta+\tau.
\end{aligned}$$

As m and n become large as compared to β and τ, this value becomes the asymptotic bound

$$\left(\sqrt{(m-1)\beta}+\sqrt{n\tau}\right)^2.$$

A.2 Bounds for sending in the path

A.2.1 The link bound path

From Theorem 2.3.1, our upper bound is

$$S_*(n,m) = \min_{1\leq k\leq n}\left\{\left(\left\lceil\frac{n}{k}\right\rceil+m-1\right)\beta+((m-1)k+n)\tau\right\}.$$

Removing the "ceiling" gives:

$$S_*(n,m) \leq \min_{1\leq k\leq n}\left\{\left(\frac{n-1}{k}+m\right)\beta+((m-1)k+n)\tau\right\}.$$

Replacing k with a continuous variable x, and differentiating:

$$\frac{\partial}{\partial x}\left(\left(\frac{n-1}{x}+m\right)\beta+\left((m-1)\,x+n\right)\tau\right)=(m-1)\,\tau-\frac{(n-1)\,\beta}{x^2}.$$

The continuous function, then, is minimized when

$$x=\sqrt{\frac{(n-1)\,\beta}{(m-1)\,\tau}},$$

and as a result, the discrete version will not be larger than when

$$k=\left\lceil\sqrt{\frac{(n-1)\,\beta}{(m-1)\,\tau}}\right\rceil.$$

Thus,

$$\begin{aligned}
S_*(n,m) &\leq \left(\frac{n-1}{\left\lceil\sqrt{\frac{(n-1)\beta}{(m-1)\tau}}\right\rceil}+m\right)\beta+\left((m-1)\left\lceil\sqrt{\frac{(n-1)\,\beta}{(m-1)\,\tau}}\right\rceil+n\right)\tau\\
&< \left(\frac{n-1}{\sqrt{\frac{(n-1)\beta}{(m-1)\tau}}}+m\right)\beta+\left((m-1)\left(\sqrt{\frac{(n-1)\,\beta}{(m-1)\,\tau}}+1\right)+n\right)\tau\\
&= m\beta+2\sqrt{(m-1)\,(n-1)\,\beta\tau}+(m-1+n)\,\tau\\
&= \left(\sqrt{(m-1)\,\beta}+\sqrt{(n-1)\,\tau}\right)^2+\beta+m\tau.
\end{aligned}$$

In a similar manner, we can deal with the lower bound:

$$S_*(n,m)\geq\min_{1\leq k\leq n}\left\{\left(\frac{n}{k}+m-1\right)\beta+\left((m-1)\,k+n\right)\tau\right\},$$

and we can certainly replace k with a continuous value:

$$S_*(n,m)\geq\min_{1\leq x\leq n}\left\{\left(\frac{n}{x}+m-1\right)\beta+\left((m-1)\,x+n\right)\tau\right\}.$$

Taking the derivative of this continuous function:

$$\frac{\partial}{\partial x}\left(\left(\frac{n}{x}+m-1\right)\beta+\left((m-1)\,x+n\right)\tau\right)=(m-1)\,\tau-\frac{n\beta}{x^2},$$

we can see that the minimum occurs when

$$x = \sqrt{\frac{n\beta}{(m-1)\tau}}.$$

We therefore have

$$\begin{aligned} S_*(n,m) &\geq \left(\frac{n}{\sqrt{\frac{n\beta}{(m-1)\tau}}} + m - 1\right)\beta + \left((m-1)\sqrt{\frac{n\beta}{(m-1)\tau}} + n\right)\tau \\ &= (m-1)\beta + 2\sqrt{(m-1)\,n\beta\tau} + n\tau \\ &= \left(\sqrt{(m-1)\beta} + \sqrt{n\tau}\right)^2. \end{aligned}$$

In closed form, then,

$$\begin{aligned} &\left(\sqrt{(m-1)\beta} + \sqrt{n\tau}\right)^2 \leq \\ &\qquad S_*(n,m) = \min_{1\leq k\leq n}\left\{\left(\left\lceil\frac{n}{k}\right\rceil + m - 1\right)\beta + ((m-1)\,k + n)\,\tau\right\} \\ &\qquad\qquad < \left(\sqrt{(m-1)\beta} + \sqrt{(n-1)\tau}\right)^2 + \beta + m\tau. \end{aligned}$$

A.2.2 The processor bound path

From Theorem 2.4.1 the upper bound for this processor bound case is:

$$S_1(n,m) = \min_{1\leq k\leq n}\left\{\left(2\left\lceil\frac{n}{k}\right\rceil + m - 2\right)\beta + ((m-2)\,k + 2n)\,\tau\right\}.$$

Removing the ceiling function:

$$\begin{aligned} S_1(n,m) &\leq \min_{1\leq k\leq n}\left\{\left(2\frac{n+k-1}{k} + m - 2\right)\beta + ((m-2)\,k + 2n)\,\tau\right\} \\ &= \min_{1\leq k\leq n}\left\{\left(\frac{2\,(n-1)}{k} + m\right)\beta + ((m-2)\,k + 2n)\,\tau\right\} \end{aligned}$$

Substituting a continuous variable x, and differentiating:

$$\frac{\partial}{\partial x}\left(\left(\frac{2\,(n-1)}{x} + m\right)\beta + ((m-2)\,x + 2n)\,\tau\right) = (m-2)\,\tau - \frac{2\,(n-1)\,\beta}{x^2}.$$

The continuous function is minimized when

$$x = \sqrt{\frac{2(n-1)\beta}{(m-2)\tau}},$$

so the discrete version is certainly no larger than when

$$k = \left\lceil \sqrt{\frac{2(n-1)\beta}{(m-2)\tau}} \right\rceil .$$

Therefore, $S_1(n,m)$

$$\begin{aligned}
&\leq \left(\frac{(2n-2)}{\left\lceil \sqrt{\frac{(2n-2)\beta}{(m-2)\tau}} \right\rceil} + m \right)\beta + \left((m-2)\left\lceil \sqrt{\frac{(2n-2)\beta}{(m-2)\tau}} \right\rceil + 2n \right)\tau \\
&< \left(\frac{(2n-2)}{\sqrt{\frac{(2n-2)\beta}{(m-2)\tau}}} + m \right)\beta + \left((m-2)\left(\sqrt{\frac{(2n-2)\beta}{(m-2)\tau}} + 1 \right) + 2n \right)\tau \\
&= m\beta + 2\sqrt{(m-2)(2n-2)\beta\tau} + (m+2n-2)\tau \\
&= \left(\sqrt{(m-2)\beta} + \sqrt{(2n-2)\tau} \right)^2 + 2\beta + m\tau.
\end{aligned}$$

We can deal with the lower bound similarly:

$$S_1(n,m) \geq \min_{1\leq k\leq n} \left\{ \left(\frac{2n}{k} + m - 2 \right)\beta + ((m-2)k + 2n)\tau \right\}.$$

Again, we can replace k with a continuous variable without invalidating the inequality. If we then differentiate:

$$\frac{\partial}{\partial x}\left(\left(\frac{2n}{x} + m - 2 \right)\beta + ((m-2)x + 2n)\tau \right) = (m-2)\tau - \left(\frac{2n\beta}{x^2} \right).$$

The minimum occurs when

$$x = \sqrt{\frac{2n\beta}{(m-2)\tau}}.$$

Substituting back into the continuous function:

$$\begin{aligned}
S_1(n,m) &\geq \left(\frac{2n}{\sqrt{\frac{2n\beta}{(m-2)\tau}}} + m - 2 \right)\beta + \left((m-2)\sqrt{\frac{2n\beta}{(m-2)\tau}} + 2n \right)\tau \\
&= (m-2)\beta + 2\sqrt{2(m-2)n\beta\tau} + 2n\tau \\
&= \left(\sqrt{(m-2)\beta} + \sqrt{2n\tau} \right)^2 .
\end{aligned}$$

So in closed form,

$$\begin{aligned}\left(\sqrt{(m-2)\beta}+\sqrt{2n\tau}\right)^2 &\le \\ S_1(n,m) = \min_{1\le k\le n}\left\{\left(2\left\lceil\frac{n}{k}\right\rceil+m-2\right)\beta+((m-2)k+2n)\tau\right\} \\ &< \left(\sqrt{(m-2)\beta}+\sqrt{(2n-2)\tau}\right)^2+2\beta+m\tau.\end{aligned}$$

A.3 Bounds for sending in the ring

A.3.1 The link bound ring

The bound here is:

$$S_*(C_p,n,m) = \min_{1\le n_0\le n}\left\{\max\left\{S_*(n_0,m),S_*(n-n_0,p-m)\right\}\right\}.$$

We're specifically interested in the lower bound, so:

$$S_*(C_p,n,m) \ge \min_{1\le n_0\le n}\left\{\frac{1}{2}\left(S_*(n_0,m)+S_*(n-n_0,p-m)\right)\right\}.$$

We can certainly replace invocations of the function $S_*(n,m)$ with the closed form lower bound in this context. We let n_0 become continuous, and differentiate:

$$\begin{aligned}&\frac{\partial}{\partial x}\left(\frac{1}{2}\left(\left(\sqrt{(m-1)\beta}+\sqrt{x\tau}\right)^2+\left(\sqrt{(p-m-1)\beta}+\sqrt{(n-x)\tau}\right)^2\right)\right)\\ =\ &\frac{1}{2}\left(2\left(\sqrt{(m-1)\beta}+\sqrt{x\tau}\right)\right)\left(\frac{\tau}{2\sqrt{x\tau}}\right)\\ &+\frac{1}{2}\left(2\left(\sqrt{(p-m-1)\beta}+\sqrt{(n-x)\tau}\right)\right)\left(\frac{-\tau}{2\sqrt{(n-x)\tau}}\right)\\ =\ &\frac{1}{2}\left(\sqrt{\frac{(m-1)\beta\tau}{x}}-\sqrt{\frac{(p-m-1)\beta\tau}{n-x}}\right).\end{aligned}$$

The minimum occurs when

$$\sqrt{\frac{m-1}{x}}=\sqrt{\frac{p-m-1}{n-x}},$$

or, more usefully,

$$x = \frac{(m-1)n}{p-2}.$$

Substituting this back into the continuous lower bound gives $S_*(C_p, n, m)$

$$\begin{aligned}
&\geq \frac{1}{2}\left(\sqrt{(m-1)\,\beta} + \sqrt{\frac{(m-1)n}{p-2}\tau}\right)^2 \\
&\quad + \frac{1}{2}\left(\sqrt{(p-m-1)\,\beta} + \sqrt{\left(n - \frac{(m-1)n}{p-2}\right)\tau}\right)^2 \\
&= \frac{m-1}{2}\left(\sqrt{\beta} + \sqrt{\frac{n\tau}{p-2}}\right)^2 + \frac{p-m-1}{2}\left(\sqrt{\beta} + \sqrt{\frac{n\tau}{p-2}}\right)^2 \\
&= \frac{p-2}{2}\left(\sqrt{\frac{(p-2)\,\beta}{p-2}} + \sqrt{\frac{n\tau}{p-2}}\right)^2 \\
&= \frac{1}{2}\left(\sqrt{(p-2)\,\beta} + \sqrt{n\tau}\right)^2.
\end{aligned}$$

The upper bound is nowhere near as tight:

$$S_*(C_p, n, m) \leq \min_{1 \leq n_0 \leq n} \left\{S_*(n_0, m) + S_*(n - n_0, p - m)\right\}.$$

We can again replace invocations of the function $S_*(n, m)$; this time with the closed form upper bound. We let n_0 become continuous, and differentiate:

$$\begin{aligned}
&\frac{\partial}{\partial x}\left(\left(\sqrt{(m-1)\,\beta} + \sqrt{x\tau}\right)^2 + \left(\sqrt{(p-m-1)\,\beta} + \sqrt{(n-x)\,\tau}\right)^2 + 2\beta + p\tau\right) \\
= \;& 2\left(\sqrt{(m-1)\,\beta} + \sqrt{x\tau}\right)\left(\frac{\tau}{2\sqrt{x\tau}}\right) \\
&+ 2\left(\sqrt{(p-m-1)\,\beta} + \sqrt{(n-x)\,\tau}\right)\left(\frac{-\tau}{2\sqrt{(n-x)\,\tau}}\right) \\
= \;& \sqrt{\frac{(m-1)\,\beta\tau}{x}} - \sqrt{\frac{(p-m-1)\,\beta\tau}{n-x}}
\end{aligned}$$

The minimum again occurs when

$$x = \frac{(m-1)n}{p-2}.$$

Substituting this back into the continuous lower bound gives $S_*(C_p, n, m)$

$$\begin{aligned}
&\leq \left(\sqrt{(m-1)\,\beta} + \sqrt{\left(\frac{(m-1)n}{p-2}\right)\tau}\right)^2 \\
&\quad + \left(\sqrt{(p-m-1)\,\beta} + \sqrt{\left(n - \frac{(m-1)n}{p-2}\right)\tau}\right)^2 + 2\beta + p\tau \\
&= (m-1)\left(\sqrt{\beta} + \sqrt{\frac{n\tau}{p-2}}\right)^2 + (p-m-1)\left(\sqrt{\beta} + \sqrt{\frac{n\tau}{p-2}}\right)^2 + 2\beta + p\tau \\
&= (p-2)\left(\sqrt{\frac{(p-2)\,\beta}{p-2}} + \sqrt{\frac{n\tau}{p-2}}\right)^2 + 2\beta + p\tau \\
&= \left(\sqrt{(p-2)\,\beta} + \sqrt{n\tau}\right)^2 + 2\beta + p\tau.
\end{aligned}$$

Again in closed form,

$$\begin{aligned}
\frac{1}{2}\left(\sqrt{(p-2)\,\beta} + \sqrt{n\tau}\right)^2 &\leq \\
S_*(C_p, n, m) = \min_{1 \leq n_0 \leq n} \{\max\{S_*(n_0, m), S_*(n - n_0, p - m)\}\} \\
&< \left(\sqrt{(p-2)\,\beta} + \sqrt{n\tau}\right)^2 + 2\beta + p\tau.
\end{aligned}$$

Bibliography

[1] Analog Devices, Inc. *ADSP-2106x SHARC User's Manual.* Analog Devices, Inc., Norwood, MA, 1995.

[2] J.-C. Bermond and P. Fraigniaud. Broadcasting and gossiping in de Bruijn networks. *SIAM Journal of Computing*, 23:212–225, 1994.

[3] J.-C. Bermond, P. Fraigniaud, and J.G. Peters. Antepenultimate broadcasting. *Networks*, 26:125–137, 1995.

[4] D.P. Bertsekas, C. Ozveren, G.D. Stamoulis, P. Tseng, and J.N. Tsitsiklis. Optimal communication algorithms for hypercubes. *Journal of Parallel and Distributed Computing*, 11:263–275, 1991.

[5] H. Charles and P. Fraigniaud. Scheduling a scattering-gathering sequence on hypercubes. *Parallel Processing Letters*, 3, 1993.

[6] R.C. Entringer and P.J. Slater. Gossips and telegraphs. *Journal of the Franklin Institute*, 307(6):353–360, 1979.

[7] P. Fraigniaud. *Communications intensives dans les architectures à mémoire distribuée et algorithmes parallèles pour la recherche de racines de polynômes.* PhD thesis, Ecole Normale Supérieure de Lyon, December 1990.

[8] P. Fraigniaud and E. Lazard. Methods and problems of communication in usual networks. *Discrete Applied Math.*, 53:79–133, 1994.

[9] P. Fraigniaud, S. Miguet, and Y. Robert. Scattering on a ring of processors. *Parallel Computing*, 13:377–383, 1990.

[10] P. Fraigniaud and J.G. Peters. Minimum linear gossip graphs and maximal linear (Δ, k)-gossip graphs. Technical Report 94-06, Simon Fraser University, Oct 1994.

[11] P. Fraigniaud and J.G. Peters. Structured communication in torus networks. In *28th Annual Hawaii International Conference on System Science*, pages 584–593, January 1995.

[12] L. Gargano, A.L. Liestman, J.G. Peters, and D. Richards. Reliable broadcasting. *Discrete Applied Math*, 53:135–148, 1994.

[13] S.M. Hedetniemi, S.T. Hedetniemi, and A.L. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18:319–349, 1988.

[14] C.-T. Ho. *Optimal communication primitives and graph embeddings on hypercubes.* PhD thesis, Yale University, New Haven, CT, 1990.

[15] J. Hromkovič, R. Klasing, B. Monien, and R. Peine. Dissemination of information in interconnection networks (broadcasting & gossiping). In F. Hsu and D.-Z. Du, editors, *Combinatorial Network Theory.* Science Press, AMS, 1993. To appear.

[16] S.L. Johnsson and C.-T. Ho. Optimum broadcasting and personalized communication in hypercubes. *IEEE Transactions on Computers*, 38:1249–1268, 1989.

[17] W. Knodel. New gossips and telephones. *Discrete Math*, 13, 1975.

[18] A.L. Liestman and J.G. Peters. Broadcast networks of bounded degree. *SIAM Journal of Discrete Math*, 1:531–540, 1988.

[19] A.L. Liestman and J.G. Peters. Minimum broadcast digraphs. *Discrete Applied Math*, 37/38:401–419, 1992.

[20] D.B. Peters and J.G. Peters. Bounded depth broadcasting. *Discrete Applied Math*, 1995. To appear.

[21] J.G. Peters and M. Syska. Circuit-switched broadcasting in torus networks. *IEEE Transactions on Parallel and Distributed Systems*, 1995. To appear.

[22] Y. Saad and M.H. Schultz. Data communication in parallel architectures. *Parallel Computing*, 11:131–150, 1989.

[23] Q. Stout and B. Wagar. Intensive hypercube communication, prearranged communication in link-bound machines. *Journal of Parallel and Distributed Computing*, 10:167–181, 1990.