Optimal broadcast in a distributed memory model of parallel computation

Ramesh Subramonian
Division of Computer Science,
University of California, Berkeley,
Berkeley CA 94720
subramon@cs.berkeley.edu

Narayan Venkatasubramanyan
American Airlines Decision Technologies,
Dallas/Fort Worth Airport, TX 75261-9616
narayan@aadt.com

March 12, 1993

Abstract

A commonly arising problem in many parallel algorithms is broadcast. Many multi-processors do not have dedicated hardware for this purpose. Therefore, a broadcast must be effected by point-to-point message passing. In this paper, we address the problem of effecting broadcast by point-to-point message transmission, where a *source* processor has *many* messages which it wishes to disseminate to $P-1$ other processors. In a step, a processor can send at most one item from among those in its possession and receive at most one item. An item is received at most L steps after it is sent. The goal is to find a schedule that achieves the broadcast in minimum time.

We improve on previous results by (i) providing an integer programming formulation whose optimal solution represents an optimal k-broadcast schedule, without making any assumptions about L or P; and (ii) developing a simpler but marginally sub-optimal solution which also makes no assumptions about L or P. In addition, we (i) refine the LogP model to capture network behaviour more precisely; and (ii) provide a bound on the performance degradation when the latency is allowed to vary.

1 Introduction

1.1 Background

The development of parallel computing has been hampered by the lack of simple models of parallel computation that accurately model a variety of different machines. The PRAM model fails to adequately model communication issues. Network models are too closely tied to specific topologies and hence are inadequate as models of general-purpose parallel computation.

This motivated the development of the *LogP* model [1]. The model has four parameters:

1. P, the number of *processors*.
2. L. This models the *latency*, which is an upper bound on the time to deliver a message from one processor to another.
3. g. This models the *bandwidth*. It is a lower bound on the time between two successive message transmissions by a processor.
4. o. This models the *overhead*. This is the amount of time spent by a processor in sending or receiving a message.

A cogent technological argument for this model is provided in [1]. The authors also discuss when simpler models, derived by ignoring one or more of the above parameters, are appropriate.

Referring the reader to [1] for more details, we highlight one of the features of the LogP model: the assumption that the processors are connected in a *complete graph*. This means that all processors are equi-distant from one another. There are several reasons why we made this simplifying assumption. In many multiprocessors, the overhead of the network interface tends to dominate and drown out minor gains in exploiting neighbourhood properties, especially for small messages. We believe this will continue to be the case because, while network technology is improving, it is not driven by the same market forces as microprocessor and memory technology. Therefore, the three aspects that characterize today's communication networks - high latency, high overhead and low bandwidth - will continue to be problems. Also, for better routing, many multiprocessors actually transmit small messages, breaking down large messages into a sequence of small messages. The necessity of providing fault-tolerance in large multiprocessors has led designers to discourage programming which takes advantage of the relative location of different processors [**?**]. Possibly the most important rationale for this choice is that it does not degrade performance significantly and allows the design of portable algorithms and programs.

1.2 The Problem

A problem that occurs often in many parallel algorithms is *broadcast*. In particular, consider the k-broadcast problem in which one processor (called the *source*) possesses k items which it wishes to disseminate to the remaining $P-1$ processors. Dedicated hardware for this purpose might not be available on all machines. This makes it important to provide a message-passing simulation of broadcast. We shall study this problem in the context of the LogP model (see Section 2).

A restricted version of this problem is the *continuous broadcast* problem where the life-cycle of the source is constrained to follow the pattern: send item i in step i. Here the assumption made is that, k, the number of items is infinite. The *delay* of an item is the time between it is sent by the source and the time all processors receive that item. The objective is to minimize the maximum, over all items, of the delay of an item. The motivation behind studying continuous broadcast is if the schedule so derived is truncated and used for finite k, it is not unduly inefficient.

1.3 Related Work

Since broadcast is an important primitive, it has received considerable attention in the field. We point the reader to the following relevant references [2, 3] but would like to focus the reader's attention on two papers which have attempted essentially the same problem.

Let $B(P, L)$ be the minimum delay in broadcasting 1 item, resident on a *source* processor, to $P-1$ other processors.

In [4, 5], a solution which requires $2B(P)+k+O(L)$ time was derived on the very similar postal model. In [6], an optimal solution for continuous broadcast is derived. However, their solution works only for very restricted values of L and P.

An important characteristic of real systems ignored in both these papers, is the effects of varying latencies and, as a consequence, messages being received out of order. In [6], the problem of what happens when two messages arrive at a processor at the same time is considered to some extent. However, we feel that their assumption that "each processor p can examine items in its buffer at time step t and determine which item it will receive for this time step" at *no cost* seems a little too optimistic.

1.4 New Results

We have considered various aspects of this problem and our results are as follows.

1. In Section 4, we provide an integer programming formulation whose optimal solution yields an optimal schedule for the k-broadcast problem. Our attempt to achieve absolute optimality (there are no hidden constant factors, neither additive nor multiplicative, in our solution) might be construed as overkill. However, since the Achilles'

heel of most multi-processors is communication, it is important to extract the utmost of what is usually a severely limited communication bandwidth. Therefore, although integer programming could, in the worst case, require exponential time, our decision to use it was based on the following:
(i) the integer program needs to be solved only once for a given computer model, and
(ii) integer programming is a well-understood combinatorial optimization technique for which a variety of mature tools and packages exist.

2. In Section 5, we focus on a restricted version of the general broadcast problem called "continuous broadcast". In this case, k is assumed to be infinite and the source is restricted to sending item i at time i. For this case, we show that *if* the lower bound can be achieved, certain additional constraints can be imposed and the number of variables drastically reduced. This greatly speeds up the time required to solve the integer program.

3. In Section 6, we derive a schedule for the continuous broadcast problem. It is a very simple schedule but is slightly sub-optimal. In this section we diverge somewhat from the model of analysis used by [5] and [6] by taking into account varying latencies (Section 2). The delay is $B(P-1, L+1) + L$ which is slightly worse than the lower bound of $B(P-1, L) + L$.

4. In Section 7, we study the effect of selectively varying latency. In the analysis of Sections 4 through 6, we make the standard assumption that that each message incurs exactly the worst case latency L. Although it appears counter-intuitive, the schedule may actually be *delayed* if the latency of some messages is *reduced*. We provide a bound on the delay that can be introduced if an adversary is allowed to look at the schedule and then specify the latency of each message as some integer between 1 and L.

2 Model

We will use the LogP model [1] for analysis. Since capacity considerations and overheads are not germane to this discussion, we shall assume $o = 0$ and shall normalize by g. This reduces to the following assumptions:

1. processors operate in lock-step synchrony,
2. a processor can send at most one message per step,
3. a processor can receive at most one message per step,
4. a message sent at time t will be received no later than $t + L$.

However, we must make the LogP model more precise in order to specify what happens if two processors send a message to the same processor in the same step. We do this as follows:

A message is said to be *outstanding* if it has been sent but has not been received. Each processor has a pool of outstanding messages. If p_1 sends a message to p_2 at time t, it is added to the pool of outstanding messages at p_2, and is tagged with some time instant between $t+1$ and $t+L$; this time instant represents the earliest time at which this item is available for receipt by p_2. The exact value of the tag is specified by an adversary.

When a processor executes a receive, it receives one of the items in its pool, which is available for receipt. If there is more than one such item, then the item with lowest tag is received. If there is more than one item with the same lowest tag, then an adversary is allowed to specify which one is received.

3 Definitions and Background Results

Definition 3.1 *The delay of an item is the length of the interval between the time it is sent by the source and the time all processors receive that item.*

Definition 3.2 *Let $B(P, L, o, g)$ be the minimum delay in broadcasting 1 item, resident on a* source *processor, to $P-1$ other processors.*

Definition 3.3 *Let $L > 0$ be a fixed integer. For a 1-item broadcast, let f_i be the number of processors that possess the item at time i. The sequence $\{f_i\}$ is:*

1. $f_i = 0$ *for* $i < 1$
2. $f_i = 1$ *for* $1 \leq i \leq L$
3. $f_i = f_{i-1} + f_{i-L}$ *otherwise.*

Definition 3.4 *P is said to be a "nice" value if $P = f_i$ for some i.*

Theorem 3.1 *[6] In continuous broadcast, the minimum delay of an item is $B(P-1, L) + L$.*

Definition 3.5 *A* universal optimal broadcast tree*, $\mathcal{B}$, is an infinite tree created as follows. Create a root node with receive-label 1. The receive-label of a node is the time at which the processor assigned to that node receives the message. For each node with label l, create an infinite number of children with receive-labels $l + (L), l + (L+1), l + (L+2), \ldots$.*

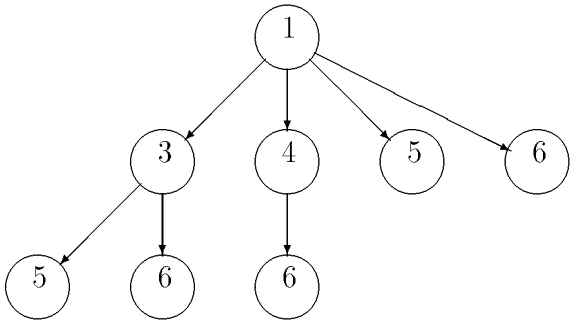

Figure 1: Restricted Broadcast tree for $P = 8, L = 2$.

Definition 3.6 *The universal tree is defined for infinite P. If P is finite, then the* restricted broadcast tree *is the subtree of $\mathcal{B}$ that has P nodes and the maximum receive label in the resultant tree is the minimum over all possible subtrees. Further, among subtrees that have the above property, we arbitrarily choose a subtree of minimum depth (The reasons for doing so will be apparent in Section 7). If P is "nice" then the subtree is unique.*

Theorem 3.2 *[6] Let $\mathcal{B}(\mathcal{P})$ be a restricted broadcast tree with P nodes. Then, $\mathcal{B}(\mathcal{P})$ is optimal for single item broadcast.*

Theorem 3.3 *[6] Let r be the largest index such that $f_r < P-1$, so that $B(P-1, L) = r+1$. Let $k^* = \lfloor \sum_{i=1}^{i=r} \frac{f_i}{(P-1)} \rfloor$. Any k-broadcast algorithm requires at least $T_{lb} = B(P-1, L) + L + (k-1) - k^*$ steps.*

Definition 3.7 *Let $d(P)$ be the minimum depth of a restricted broadcast tree with P nodes.*

Lemma 3.1 $d(P) = \min d : (d+1)L + 1 > B(P)$

Proof. Consider the left-most path of the tree. It is obvious that for any node along this path, it was the first node at that depth to receive an item. Therefore, the minimum depth of the tree will be minimum depth of the node along the left-most path which must be included in the broadcast tree i.e., whose receive label is less than or equal to the broadcast time. The receive label of the node at depth i along the left-most path is $(iL) + 1$. The proof follows. $\square$

4 Integer Programming Formulation for k-broadcast

We shall set up an integer programming problem, whose solution will give us the desired schedule. We introduce the variables: $f(k', p', t', Send)$ and $f(k', p', t', Receive)$ [1]. These are

[1]We have denoted k', p', t', Send and Receive as arguments for cleaner typesetting, although they should really be thought of as subscripts.

Figure 2: Possible restricted broadcast trees for $P = 4, L = 2$. The one on the right will be chosen because it has smaller depth.

the number of items of type k' sent and received respectively by processor p' at time t'. So, $k' \in \{1,..,k\}$, $p' \in \{1,..,P\}$, $t' \in \{1,..,T_{max}\}$. We shall specify the value of T_{max} later. Further, let $X_{t'}$ represent a binary variable which takes on a value of 1 when any processor sends any item at time t', and 0 otherwise.

The source processor is processor 1. All k items are resident on the source processor.

The constraints introduced are as follows:

1. A processor can send at most one item in a step. Further, a processor may send an item at time t' only if sends have been enabled at that instant. We formulate this as:

$$1 \leq p' \leq p, 1 \leq t' \leq T_{max}, \sum_{k'=1}^{k'=k} f(k', p', t', Send) \leq X_{t'}$$

   Since we know that sends must be enabled up to time $T_{lb} - L$ (T_{lb} is defined in Theorem 3.3), we can simplify this as

$$1 \leq p' \leq p,\ 1 \leq t' \leq T_{lb} - L,\ \sum_{k'=1}^{k'=k} f(k', p', t', Send) \leq 1$$

$$1 \leq p' \leq p,\ T_{lb} - L < t' \leq T_{max} - L,\ \sum_{k'=1}^{k'=k} f(k', p', t', Send) \leq X(t')$$

2. To ensure that there are no "gaps" in the broadcast, we impose the following requirement:

$$1 \leq t' \leq T_{max} - 1,\ X_{t'} \geq X_{t'+1}$$

3. A processor can receive at most one item in a step. We formulate this as:

$$2 \leq p' \leq p,\ 1 \leq t' \leq T_{max},\ \sum_{k'=1}^{k'=k} f(k', p', t', Receive) \leq 1$$

4. It is easy to argue that there exists an optimal broadcast in which a processor receives any given item at most once. We formulate this as:

$$2 \leq p' \leq p,\ 1 \leq k' \leq k,\ \sum_{t'=1}^{t'=T_{max}} f(k', p', t', \text{ Receive }) = 1$$

5. A processor can send only items that it has received. We formulate this as:

$$2 \leq p' \leq p,\ 1 \leq k' \leq k,\ \sum_{\hat{t}=1}^{\hat{t}=t'} f(k', p', \hat{t}, \text{ Receive }) \geq f(k', p', t', \text{ Send })$$

6. An item sent in a step will be received L steps later. We formulate this as:

$$1 \leq t' \leq T_{max} - L,\ 1 \leq k' \leq k,\ \sum_{p'=1}^{p'=p} f(k', p', t', \text{ Send }) = \sum_{p'=1}^{p'=p} f(k', p', t' + L, \text{ Receive })$$

7. Recognizing that sends must be enabled up to time $T_{lb} - L$, the objective function to be minimized is:

$$\sum_{t'=T_{lb}-L+1}^{t'=T_{max}} X_{t'}$$

8. For the solution to be meaningful, all variables must be $\{0, 1\}$-valued.

Theorem 4.1 *There exists a k-broadcast schedule which takes time T_{max}, or less, if and only if there exists a feasible solution to the above integer programming problem. Further, if that integer program is feasible, the optimal solution will correspond to the optimal k-broadcast.*

Proof. Any broadcast can be represented as a $(0, 1)$ vector, each component corresponding to the variables $f(., ., ., .)$ ordered in some suitable manner. Similarly, any solution to the system of constraints can be represented as a $(0, 1)$ vector. We need to show that the set of vectors that represent broadcasts (β) is equal to the set of vectors that satisfy the system of constraints (κ).

It is easily verified that a vector representing a feasible broadcast will satisfy all the constraints in the mathematical formulation. Hence, β is a subset of κ.

Let us assume that there exists a $(0, 1)$ vector which satisfies all the constraints but is not a feasible broadcast. Such a $(0, 1)$ vector would correspond to a broadcast scheme which violates one or more the restrictions imposed in Section 2. It is easily verified that any such violation would require that one or more of the constraints in the integer program be violated, too. Hence, by contradiction, it follows that κ is a subset of β.

We need to prove that the solution of the integer program achieves a complete k-broadcast in the shortest possible time. Consider two solutions, $s_1, s_2 \in \kappa$, such that the last send in s_1

is at time t_1, the last send in s_2 at time t_2 and $t_1 < t_2$. Then, by the non-increasing nature of $X_{t'}$, it is obvious that the value of the objective function for s_1 is less than the value of the objective function for s_2.

To ensure feasibility, T_{max} is chosen to be some provable upper bound on the time required to achieve a k-broadcast (see Lemma 4.1).

The solution to the integer program is not strictly a schedule. What it does tell us is for each time step, which processor is sending or receiving which item. A schedule is derived by arbitrarily matching up, for each item, the processors sending an item k' at time t with the processors receiving item k' at time $t + L$. □

Lemma 4.1 $T_{max} \leq L + (k-1) + L + (p-1)$.

Proof. This is easily observed by considering a simple strategy where the source processor sends the k items to different processors. As soon as a processor receives an item from the source, it broadcasts that item to the other processors by sending the item to $p_1, p_2, \ldots$. *All processors* follow the same order of sends. A send from p_i to p_i is simply not performed. It is easy to see that this scheme satisfies all the requirements to be a legal schedule (see Section 2).

The time at which the last item is first received by a non-source processor is $L + (k-1)$. $p-1$ steps later it will have been dispatched to all other processors and a further L steps it will have been received by all processors. □

5 Meeting Lower Bound on Continuous Broadcast

It remains an open problem to decide for an arbitrary L and P whether the lower bound on continuous broadcast can be met. In this section, we show that **if** the lower bound (Theorem 3.1) can be met, then

1. certain additional constraints, which could potentially improve the running time of the integer programming solution, can be imposed; and
2. the number of variables and constraints can be significantly reduced.

5.1 Introducing Additional Constraints

We start by defining a $\{\hat{f}\}$ sequence such that $\hat{f}_i$ is the number of processors that send item 1 at time i.

Definition 5.1 *Let R be the largest value of r such that $f_r \leq P$.*

1. $\hat{f}_i = f_i$ *if* $i \le R - L$

2. $\hat{f}_{R-L+1} = (P-1) - \sum_{j=1}^{j=R-L} f_j$

3. $\hat{f}_i = 0$ *if* $i > R - L + 1$

In this section, we shall shift the time axis by L and assume that the first time a non-source receives item k' is at time k'. Given this and the $\{\hat{f}_i\}$ sequence, it is simple to calculate the number of items of each type that must be sent and received in each step.

Observation 5.1 *Let $Num\_send(k', t')$ = the number of processors that send item k' at time t'. Let $Num\_receive(k', t')$ = the number of processors that receive item k' at time t'. $Num\_send(k', t') = \hat{f}_{t'-(k'-1)}$ and $Num\_receive(k', t') = Num\_send(k', t' - L)$.*

Let $Last\_send(k')$ be the last instant of time when item k' is sent by any processor. All processors receive item k' by time $B(P-1, L) + (k'-1)$. Hence, $Last\_send(k') = B(P-1, L) + (k'-1) - L$, since that is the last time at which a processor could have sent item k'. If one looks at the broadcast tree for a single item (see Figure 1), one notices that once a processor starts sending a particular item, it continues doing so until time $Last\_send(k')$. If the number of processors is not "nice", then some of the the processors that have item k' at time $Last\_send(k')$ will send it and some will not.

Observation 5.2 *$Last\_send(k') = k' + B(P-1, L) - L$. If processor p' sends item k' at time t', then it sends item k' at all subsequent time steps up to and including $Last\_send(k') - 1$.*

The additional constraints are:

1.
$$\forall k', \sum_{p'=2}^{p'=p} f(k', p', t', \text{ Send }) = Num\_send(k', t')$$
$$\forall k', \sum_{p'=2}^{p'=p} f(k', p', t', \text{ Receive }) = Num\_receive(k', t')$$

2.
$$\forall t', \ \forall p', \ k' \le t' \le Last\_send(k') - 1, \ f(k', p', t', Send) \le f(k', p', t'+1, Send)$$

3. The source sends item k' in step $k' - L$. Hence,
$$\forall k', f(k', 1, k' - L, Send) = 1$$

5.2 Reducing the Number of Constraints/Variables

We now show how the number of constraints and variables can be significantly reduced. Let T_1 be the first time at which all processors have received the first item. Then, we can reduce the range of k' to $\{1,..,T_1\}$ from $\{1,..,k\}$ and can reduce the range of t' to $\{1,..,T_1\}$ from $\{1,..,T_{max}\}$. We no longer need the objective function since all we are interested in finding is whether a feasible solution exists.

If a feasible solution exists, then extending the schedule so defined for the rest of the items is trivial. This is because the situation when the (T_1+1)th item is injected into the system is the same as that when the T_1th item was injected into the system. Therefore, the same processor assignment strategy used at the T_1th time step can be repeated at the (T_1+1)th time step (modulo some processor re-numbering).

6 A Simple Sub-optimal Continuous Broadcast Schedule

In this section, we present a simple schedule for continuous broadcast which is almost optimal. We first create a restricted broadcast tree (Definition 3.6) with $P-1$ nodes assuming that the latency is $L+1$ (This particular choice of latency is explained in Theorem 6.1). The source will only feed the root of the broadcast trees of the different items, which is why the tree has $P-1$ nodes instead of P.

Definition 6.1 *G_j is the number of children of node j in the broadcast tree.*

Definition 6.2 *T_j is the receive-label of node j in the broadcast tree of item 1.*

We label the internal nodes arbitrarily with the labels $1, 2, \ldots$. We assign the first group of G_1 processors to node 1, the next group of G_2 processors to node 2, and so on. The kth processor of the jth group will receive item $xG_j + k$ in steps $T_j + xG_j + (k-1)$ and send the item in G_j steps, starting from the step in which it receives the item.

For every processor, we have completely specified which item will be sent in which step. By construction, we are guaranteed that a processor that is required to send an item at time t receives that item at or before t. We must now specify when which items are received. We shall first concern ourselves only with non-terminal receives. Therefore, Lemma 6.1 deals only with the assignment of processors to internal nodes of the broadcast trees.

Definition 6.3 *A receive is said to be a "terminal receive" if the item received is not sent by the receiving processor; else, it is a "non-terminal receive".*

Lemma 6.1 *The processor assignment algorithm achieves the following: (i) all sends required for the continuous broadcast are effected; (ii) no processor is required to send more than one item per step; (iii) no processor is required to receive more than one non-terminal receive per step.*

Proof. To simplify the proof, we convert the requirements into a graph colouring problem. Consider a graph G consisting of infinite sequence of subgraphs, the ith subgraph being the restricted broadcast tree of the ith item. Node j of subtree i has a receive-label $(i-1)+T_j$ (Definition 6.2) . Node j of the ith subtree has a set of send-labels which are $\{(i-1)+T_j,(i-1)+T_j+1,..,(i-1)+T_j+(G_j-1)\}$. (If node j has send-labels $\{x_1,x_2,..x_{G_j}\}$ in tree i it means that the processor assigned to node j in tree i will send item i in steps x_1,s_2,x_{G_j}.) We connect two nodes with an edge if (i) they belong to the same subgraph, or (ii) they have the same receive-label or (iii) they have a common send-label.

The colouring scheme is as follows: The colours used are $1,\ldots,P-2$. Node j of broadcast i is coloured $\sum_{l=1}^{l=j-1} G_l + ((i-1) \textbf{ mod } G_j)+1$. The assignment of colour c to node j in broadcast tree i means that processor c will play the role of node j in the broadcast tree of item i.

1. All sends are effected because we have specified how all internal nodes are coloured and the colouring uses $\sum G_j = P-2$ colours whereas we have $P-1$ non-source processors available.
2. To see that no processor is required to send more than one item per step, we show that no two nodes with the same send-label have the same colour. There are two cases:
   (a) Node numbers are different. These nodes must be assigned different colours because they are assigned processors from different groups.
   (b) Node numbers are the same. The jth internal node of the ith tree is connected to the jth internal nodes of the following trees: $\{i-(G_j-1),i-(G_j-2),\ldots i-2,i-1,i+1,i+2,i+(G_j-2),i+(G_j-1)\}$, none of which can be assigned the same colour as i.
3. The fact that no processor is required to receive more than one non-terminal receive per step follows trivially from above. □

Destinations of terminal sends. For a simpler exposition, we initially assume that a processor can receive two items in the same step. We will remove this assumption later.

Note that $\sum_j G_j = P-2$. Therefore, there is one processor that does not belong to any group. We take care of this by assigning one processor to be a pure-receive processor: it simply receives an item in each step. In the subsequent discussion, we shall ignore this processor, since satisfying its needs is trivial.

We introduce a few definitions.

Definition 6.4 *Let*

- $\hat{R}_i$ = *one less than the first time at which a terminal receive of item* i *occurs.*
- $\hat{S}_i$ = *one less than the first time at which a terminal send of item* i *occurs.*
- $x(i,k)$ = *number of terminal sends of item* i *at time* $\hat{S}_i + k$.
- $y(i,j,k)$ = *number of terminal receives of item* i *in group* j *at time* $\hat{R}_i + k$.

Observation 6.1 $\forall i, \forall k, \sum_j y(i,j,k) = x(i,k)$

Remark. This is simply the observation that the number of sends of item i at step $\hat{R}_i + k$ must be equal to the number of receives of item i at step $\hat{S}_i + k$. Notice that the values of $\hat{R}_i, \hat{S}_i, x(i,k)$ are immediate consequences of the choice of broadcast tree.

Observation 6.2 $\forall i, \forall j, \sum_k y(i,j,k) = G_j - 1$

Remark. Upto and including time $\hat{R}_i$, only one processor in each group has received item i. This item was received as a non-terminal receive. All other processors in the group will receive the item as terminal receives. This observation merely requires us to ensure that each group receives each item sufficiently many times. Of course, these receives must be distributed evenly if each processor is to get each item.

We now fix the y's such that the equalities of Observations 6.1 and 6.2 are satisfied. Further, we will require for reasons which will become apparent in the proof of Lemma 6.3, that for any two items i_1, i_2, $\forall j, \forall k,\ y(i_1,j,k) = y(i_2,j,k)$.

Lemma 6.2 *It is always possible to find an assignment of values to the y's consistent with the equalities of Observations 6.1 and 6.2.*

Proof. Follows simply from the fact that any allocation of incoming items to the different groups suffices as long as no group receives more items than it needs. □

We have now specified **how many** processors in a given group receive a given item in a given step. The following Assignment protocol specifies the identity of these processors. Consider the terminal receives of the jth group at time $\hat{R}_{i_2} + 1$. At this instant, the jth group will receive:

$y(i_2, j, 1)$ items of type i_2;
$y(i_2 - 1, j, 2)$ items of type $i_2 - 1$;
$y(i_1, j, i_2 - i_1 + 1)$ items of type i_1; and so on

In general, the jth group will receive $y(i', j, (i_2 - i' + 1))$ items of type i', $i' \in \{i_1, \ldots, i_2\}$.

We assign processors to items, starting with i_1 and working our way up to i_2. We start with none of the G_j processors in this group marked. For a given item, $i', i_1 \leq i' \leq i_2$, arbitrarily assign $y(i', j, (i_2 - i' + 1))$ processors to receive item i' and mark them, subject to the constraints:

1. none of the processors have received i' in a prior step; and

2. none of the processors are marked.

If it is not possible to assign the desired number of processors, the Assignment protocol "fails".

Lemma 6.3 *The Assignment protocol never fails.*

Proof. Consider the reception of item $i', i_1 \leq i' \leq i_2$ in the jth group at time $\hat{R}_{i_2}+1$. The number of processors that have received i' prior to this instant is $1 + \sum_{l=1}^{l=i_2-i'} y(i', j, l)$. The "+1" is because in each group, there is exactly one processor that has a non-terminal receive of i; all other processors receive i as a terminal receive. When the Assignment Protocol begins to assign processors to item i', the number of processors that are already "marked" is $\sum_{i=i_1}^{i=i'+1} y(i, j, i_2 - i + 1)$ which is equal to $\sum_{l=i_2-i'+2}^{l=i_2-i_1+1} y(i', j, l)$ because of our earlier requirement that for any two items $\hat{i}, \bar{i}$, $y(\hat{i}, j, k) = y(\bar{i}, j, k)$. Therefore, the number of processors free to receive i' (those which have not received i' and which are not "marked") is at least $G_j - (1 + \sum_{l, l \neq i'} y(i', j, i_2 - i' + 1))$ which, by Observation 6.2, is equal to $y(i', j, i_2 - i' + 1)$. This is as many processors as is needed. □

Theorem 6.1 *The processor assignment described above satisfies the requirements of a k-broadcast and achieves a delay of $L + B(P-1, L+1)$.*

Proof. The additive term of L is dues to the time taken by the processor assigned to the root of the broadcast tree to receive that item from the source. The rest of the delay follows trivially from Lemmas 6.1 and 6.3 except for one fact: we may have allowed a processor to receive one terminal receive and one non-terminal receive in a step. However, we have designed the schedule under the assumption that the latency is $L+1$. Therefore, every item actually *arrives* in a step *earlier* than it is scheduled to arrive in the broadcast tree with latency $L+1$. We have shown that at most two items can arrive in a step: a terminal receive and a non-terminal receive. No matter how the adversary orders such receives, an item will be *received* no later than the step in which it is scheduled to be received, provided that there are no outstanding items at that processor at the instant these two arrivals occur.

This will never happen because if we break the time line arbitrarily into intervals of length G_j, each such interval will see exactly the same pattern of G_j receives. Therefore, if two items arrive in a step, then there must be a time instant at which no item arrives, before the next time at which two items arrive simultaneously. □

Remark. From an implementation point of view, the fixed communication pattern of the schedule we have derived has a significant advantage. This is because it allows each processor to set up channels to the processors to which it will communicate. On machines like the CM-5 (from Thinking Machines) or the iWARP (from Intel) this leads to signficant savings.

7 The Effect of Varying Latency

Definition 7.1 *A message is said to be "propagating" if the processor that receives it must send the item received; else, it is non-propagating.*

In the analysis of preceding sections, we have assumed that each message incurs exactly the worst case latency L. Although this appears to be a very pessimistic assumption, the performance can actually deteriorate if an adversary is allowed to *reduce* the latency of certain messages. To see how this can happen, consider a propagating message (Definition 7.1) that is scheduled to arrive at time t and a series of non-propagating messages that would have arrived in the L subsequent time slots had they all incurred a latency of exactly L. By setting the latencies of the subsequent messages as $L-1, L-2, \ldots$, the adversary could force all L messages to arrive at the same time. The queueing discipline will queue the messages in some arbitrary order. Thus, it is conceivable that the propagating message is received L steps after it would have been received under the constant latency assumption. This effect could ripple through the schedule, effectively doubling the latency. We now bound the time by which an adversary could delay the schedule of Section 6.

Theorem 7.1 *The maximum additional delay for the continuous broadcast schedule of Section 6 caused by an adversary being allowed to set the delays of each message as some integer between 1 and L is* $(L-1) \times d(P-1)$ *where* $d(P)$ *is defined in Definition 3.7.*

Proof. Let P_i be processors assigned to internal nodes of depth i. An item is said to have arrived at a processor at time t if it was sent at time $\leq t-l$ where l is the latency incurred by it. An item is said to have been received by a processor at time t if it reaches the head of the queue of that processor at time $t-1$.

Processors in P_0 are fed by the source, whose sends cannot be delayed. Hence, propagating messages will *arrive* on time. However they may not be received at that very instant due to queueing effects.

It remains to argue that a propagating message will be *received* no more than $L-1$ steps after it has arrived. The adversary can speed up messages (whose arrival time has been determined under the fixed latency of L assumption) by no more than $L-1$. Therefore, the adversary can cause at most $L-1$ messages to be received before the propagating message.

Processors in P_i receive propagating items from processors in P_{i-1}. An easy inductive argument establishes that their propagating items are delayed by at most $L \times i$ steps due to the delay in message arrival and a further L due to delays in message reception. $\square$

8 Conclusion and Future Work

In this paper, we provide a simple, almost optimal solution for the continuous broadcast problem which improves on earlier work. We refine the LogP model[1] to better capture the behaviour of the network. We point out that the delay calculated under the assumption that all messages incur the worst case latency can actually increase if the latency of some messages is selectively decreased. We provide an upper bound on the maximum degradation in performance by varying latency selectively.

We introduce the notion of a *robust* broadcast schedule. This is one which achieves broadcast in the minimum time regardless of how an adversary chooses to reduce the latency of certain items. It should be noted that it is possible that given two schedules S_1 and S_2, such that S_1 is better than S_2 under under the fixed latency assumption, S_1 could be worse than S_2 when latencies are allowed to vary. The problem of finding a robust broadcast schedule remains an open one.

References

[1] D. Culler, R. Karp, D. Patterson, E. Santos, A. Sahay, K. Schauser, R. Subramonian, and T. von Eicken. Towards a realistic model of parallel computation. *Proc. of Principles and Practices of Parallel Programming*, 1993.

C. Leiserson, Z. S. Abuhamdeh, D. Douglas, C. R. Feynmann, M. Ganmukhi, J. Hill, W. D. Hillis, B. Kuszmaul, M. St. Pierre, D. Wells, M. Wong, S-W Yang, and R. Zak. The network architecture of the connection machine cm-5. In *Proceedings of the 4th Symposium on Parallel Algorithms and Architectures*, pages 272–285, 1992.

[2] S. Bertsekas and J. M. Tsitkilis. *Parallel and Distributed Computation: Numerical Methods.* Prentice-Hall, 1989.

[3] S. M. Hedetniemi, S. T. Hedetniemi, and A. L. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18(4):319–349, 1988.

[4] A. Bar-Noy and S. Kipnis. Designing broadcasting algorithms in the postal model for message-passing systems. *Proc. 4th Symp. on Parallel Algorithms and Architectures*, pages 11–22, 1992.

[5] A. Bar-Noy and S. Kipnis. Multiple message broadcasting in the postal model. Technical Report RC 18230, IBM, November 1992.

[6] R. M. Karp, A. Sahay, E. Santos, and K. Schauser. Optimal broadcast and summation in the logp model. *Proc. 5th Symposium on Parallel Algorithms and Architectures*, pages 142–153, 1993.