On Broadcasting in Heterogenous Networks *

Samir Khuller † Yoo-Ah Kim ‡

Abstract

In this paper we study a widely used broadcasting heuristic for heterogenous networks of workstations, called *fastest node first*. We show that this heuristic produces an optimal solution for minimizing the sum of all completion times, and in addition produces a 1.5 approximation for the problem of minimizing the maximum completion time. We extend these results to show that the same bounds can be obtained for the multicast operation on such heterogenous networks. In addition we show that the problem of minimizing the maximum completion time is NP-hard, which settles the complexity of this open problem.

1 Introduction

Networks of Workstations (NOWs) are an extremely popular alternative to massively parallel machines and are widely used (for example the Condor project at Wisconsin [16]) and the Berkeley NOW project [15]. By simply using off-the-shelf PC's, a very powerful workstation cluster can be created, and this can provide a high amount of parallelism at relatively low cost. Since NOWs are put together over time, the machines tend to have different capabilities and this leads to a *heterogenous* collection of machines, rather than a *homogenous* collection, where all the machines have identical capabilities.

One fundamental operation that is used in such clusters, is that of *broadcast* (this is a primitve in many message passing systems such as MPI [1, 6, 8]). In addition it is used as a primitive in many parallel algorithms. The main objective of a broadcast operation is to quickly distribute the input data to the entire network for processing [18]. Another situation is when the system is performing a highly parallel search, then the successful processor needs to inform all other processors that the search has concluded successfully. Various models for heterogenous environments have been proposed in the literature. One general model is the one proposed by Bar-Noy et al [3] where the communication costs between links are not uniform. In addition, the sender may engage in another communication before the current one is complete. An approximation factor with a guarantee of $O(\log k)$ is given for the operation of performing a multicast. Other popular models in the theory literature generally assume an underlying communication graph, with the property that only nodes adjacent in this graph may communicate.

Broadcasting efficiently is an essential operation and many works are devoted to this (see [17, 9, 10, 4, 5] and references therein). In addition, for emergency notification an understanding of how to perform broadcast quickly is essential.

Most of the algorithmic (theoretical) work done on such problems is of little interest to practitioners, because the approximation algorithms tend to be fairly complex and slow, thus defeating the purpose of performing broadcast quickly. On the other hand, a simple model and algorithm was proposed by

*Research supported by NSF Awards CCR-9820965 and CCR-0113192.

†Department of Computer Science, University of Maryland, College Park, MD 20742. E-mail : samir@cs.umd.edu.

‡Department of Computer Science, University of Maryland, College Park, MD 20742. E-mail : ykim@cs.umd.edu.

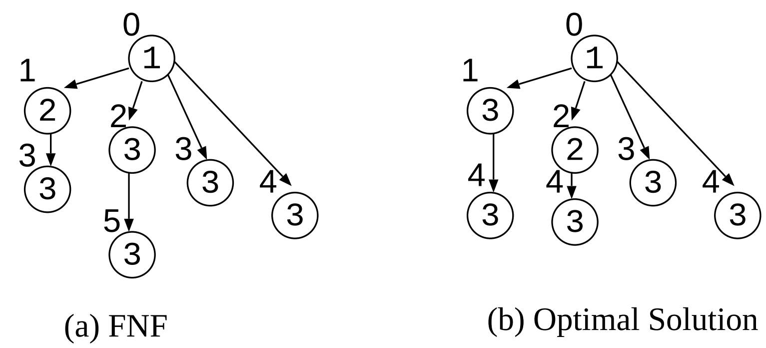


Figure 1: An example that FNF does not produce an optimal solution

Banikazemi et al [2]. In this model, heterogeneity among processors is modeled by a non-uniform speed of the sending processor. A heterogenous cluster is defined as a collection of processors $p_1, p_2, \ldots, p_n$ in which each processor is capable of communicating with any other processor. Each processor has a transmission time which is the time required to send a message to any other processor in the cluster. Thus the time required for the communication is a function of only the sender. Each processor may send messages to other processors in order, and each processor may be receiving only one message at a time.

Thus a broadcast operation is implemented as a broadcast tree. Each node in the tree represents a processor of the cluster. The root of the tree is the source of the original message. The children of a node p_i are the processors that receive the message from p_i. The completion time of a node is the time at which it completes receiving the message from its parent. The completion time of the children of p_i is $c_i + j \cdot t_i$, where c_i is the completion time of p_i, t_i is the transmission time of p_i and j is the child number. In other words, the first child of p_i has a completion time of $c_i + t_i$ $(j = 1)$, the second child has a completion time of $r_i + 2t_i$ $(j = 2)$ etc. See Figure 1 for an example.

A commonly used method to find a broadcast tree is referred to as the "Fastest Node First" (FNF) technique [2]. This works as follows: In each iteration, the algorithm chooses a sender from the set of processors that have received the message (set S) and a receiver from the set of processors that have not yet received the message (set R). The algorithm then picks the sender from $s \in S$ so that s can finish the transmission as early as possible, and chooses the receiver $r \in R$ as the processor with the minimum transmission time in R. Then r is moved from R to S and the algorithm continues. The intuition is that sending the message to fast processors first is a more effective way to propagate the message quickly. This technique is very effective and easy to implement. In practice it works extremely well (using simulations) and in fact frequently finds optimal solutions as well [2]. However, there are situations when this method also fails to find an optimal solution. A simple example is shown in Figure 1.

Despite several non-trivial advances in an understanding of the *fastest node first* method by Liu [12] (see also work by Liu and Sheng [14] in SPAA 2000), it was still open as to what the complexity of this problem is. For example, is there a polynomial time algorithm to compute a solution that minimizes the broadcast time? Can we show that in all instances the FNF heuristic will find solutions close to optimal? The second question is of interest, regardless of the complexity of the problem; for example, if there was a complex polynomial time algorithm to find the optimal solution, it is unlikely that it would be used in practice and an understanding of the FNF heuristic is still very useful.

Liu [12] (see also [14]) shows that if there are only two classes of processors, then FNF produces an optimal solution. In addition, if the transmission time of every slower processor is a multiple of the transmission time of every faster processor, then again the FNF heuristic produces an optimal solution.

So for example, if the transmission time of the fastest processor is 1 and the transmission time of all other processors are powers of 2, then the algorithm produces an optimal solution. It immediately follows that by rounding all transmission times to powers of 2 we can obtain a solution using FNF whose cost is at most twice the cost of an optimal solution. However, this still does not explain the fact that this heuristic does much much better in practice. Is there a better worst case guarantee on this simple method?

In this paper we show six results.

- We show that the FNF heuristic actually produces an optimal solution for the problem of minimizing the sum of completion times (Section 3).
- We use this to show that the FNF method has a performance ratio of at most 1.5 when compared to the optimal solution for minimizing broadcast time (Section 4).
- We show that the performance ratio of FNF is at least $\frac{25}{22}$ [1] (Section 5).
- As a corollary of the above approximation result, we are able to show that if the transmission times of the fastest $\frac{n}{2}$ processors are in the range $[1 \ldots C]$ then FNF produces a solution with makespan at most $T_{OPT} + C$ (Section 4).
- We extend the above results to the problem of multicasting (Section 6).
- We also establish that the problem is NP-hard, so unless $P = NP$ there is no polynomial time algorithm for finding an optimal solution (Section 7).

It is not surprising that the NP-completeness was open, since the proof is non-trivial. To show the NP-hardness, we have to introduce a variant of the well known 3-PARTITION problem and we first show that this problem is NP-complete by a sequence of two reductions starting from the 3 DIMENSIONAL MATCHING problem. From this variant of 3-PARTITION we are able to do a reduction to the problem of minimizing the broadcast time in a heterogenous network. We hope that this special case will be useful in proving the NP-hardness of other problems as well.

We show that the FNF heuristic actually produces an optimal solution for the problem of minimizing the sum of completion times. (This itself is a little surprising, since for most problems, minimizing the sum of completion times is usually harder than minimizing makespan.) We use this property of the algorithm to show that it actually maximizes the number of "fractional blocks[2]" in the schedule. This property leads to a lower bound on the optimal schedule based on the FNF schedule. Using this lower bound we can prove a guarantee of 1.5. We conjecture that there is a polynomial time approximation scheme for this problem (PTAS). However, the existence of such an algorithm is unlikely to convince practitioners to change what algorithm they use, especially since such an algorithm will likely be derived using dynamic programming and might be somewhat inefficient in practice.

2 Problem Definition

We are given a set of processors $(p_1, p_2, \ldots p_n)$ and there is one message to be broadcast to all the processors. Each processor p_i can send a message to another processor with transmission time t_i once it has received the message. Each processor can be either sending a message or receiving a message at any point of time. Without loss of generality, we assume that $t_1 \leq t_2 \leq \ldots \leq t_n$ and $t_1 = 1$. Also we assume that p_1 has the message at time zero.

[1] Observe that in the example shown in Figure 1 the performance ratio of FNF is $\frac{5}{4}$, but this example does not appear to generalize easily to arbitrarily large instances.

[2] This concept will be explained later.

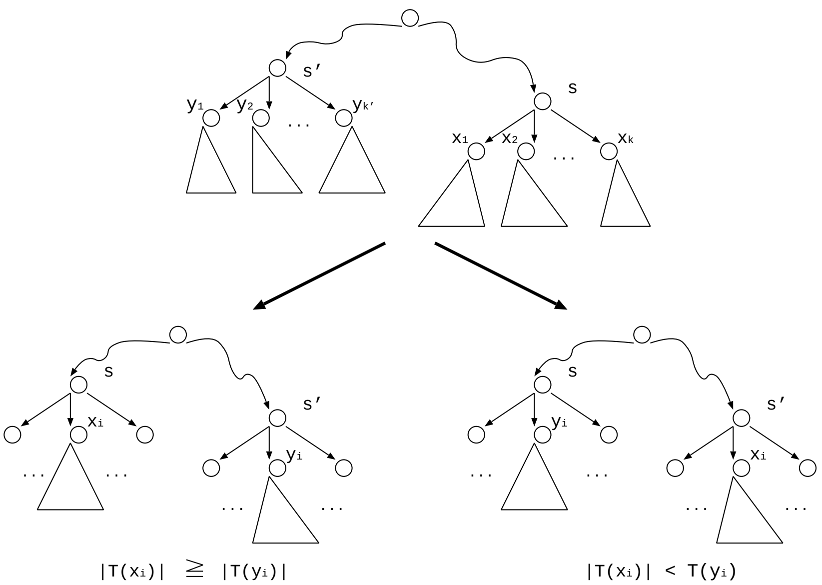


Figure 2: Figure shows how to modify the schedule of $T(s)$ and $T(s')$

We define the completion time of processor p_i to be the time when p_i has received the message. Our objective is to find a schedule that minimizes $C_{\max} = \max_i c_i$ where c_i is the completion time of processor p_i. In other words, we want to find a schedule that minimizes the time required to send the message to all the processors.

Our proof makes use of the following results from [2].

THEOREM 2.1. *[2] There exists an optimal broadcast tree in which all processors send messages without delay.*

THEOREM 2.2. *[2] There exists an optimal broadcast tree in which every processor has transmission time no less than its parent.*

3 Minimizing the Total Completion Time

In this section, we show that the FNF scheme finds an optimal schedule to minimize the total completion time of all the nodes, i.e., it minimizes $C_{\text{sum}} = \sum_i c_i$.

Suppose that we are given an (optimal) schedule of minimizing the total completion time and the total completion time is C. If there is a processor p_i whose completion time is later than processor p_j $(i < j)$, i.e., $c_i > c_j$ in the schedule, then we can find a schedule such that $c_i \leq c_j$ and the total completion time is no more than C.

Let us define a permutation $\pi : \{1, 2 \dots n\} \to \{1, 2, \dots n\}$. Then any schedule can be represented as an ordered list of processors $(p_{\pi(1)}, p_{\pi(2)}, \dots, p_{\pi(n)})$ by sorting in non-decreasing order of completion times (ties broken in accordance with their indices).

LEMMA 3.1. *Let $(p_{\pi(1)}, p_{\pi(2)}, \dots, p_{\pi(n)})$ be an optimal schedule for the problem and s be the smallest index such that $\pi(s) \neq s$. Then we can find a schedule with total completion time no more than the optimal and $i = \pi(i)$ for all $i \leq s$.*

Proof. In the given optimal broadcast tree, let us call the node corresponding to processor p_i as node i. We denote the subtree rooted at node i as $T(i)$. Consider subtrees $T(s)$ and $T(s')$ where $s' = \pi(s)$.

We know that s cannot be an ancestor of s' as $c_{s'} < c_s$. Also s' cannot be an ancestor of s as $t_s < t_{s'}$ by Theorem 2.2. Therefore, $T(s)$ and $T(s')$ are disjoint.

Let node s have children $x_1, x_2, \ldots, x_k$ and node s' have $y_1, y_2, \ldots, y_{k'}$ as shown in Figure2. We change the schedule as follows. First we exchange s and s'. In other words, the modified completion time of p_s becomes $c_{s'}$ and the completion time of $p_{s'}$ becomes c_s. Clearly, this does not increase the total completion time. For all i $(1 \le i \le \max(k, k'))$, we compare the size of subtree $T(x_i)$ and $T(y_i)$ and attach the bigger one to s and the smaller one to s' as i-th child. (if there does not exist a child, simply consider the size of the subtree as zero)

We can prove that this modification does not increase the sum of completion times. The difference of the total completion time for subtree $T(x_i)$ and $T(y_i)$ depends on which parent they are attached to. In case that $|T(x_i)| \ge |T(y_i)|$, the completion times of processors in $T(x_i)$ are decreased by $c_s - c_{s'}$ since the completion time of p_{x_i} is changed from $c_s + i \cdot t_s$ to $c_{s'} + i \cdot t_s$. The completion times of processors in $T(y_i)$ are increased by $c_s - c_{s'}$ since the completion time of p_{y_i} is changed from $c_{s'} + i \cdot t_{s'}$ to $c_s + i \cdot t_{s'}$. Therefore the difference is

$$(c_s - c_{s'})(|T(y_i)| - |T(x_i)|) \le 0$$

In case that $|T(x_i)| < |T(y_i)|$, the completion times of processors in $T(x_i)$ are increased by $i \cdot (t_{s'} - t_s)$ because the completion of x_i is modified from $c_s + i \cdot t_s$ to $c_s + i \cdot t_{s'}$. and the completion times of processors in $T(y_i)$ are decreased by $i \cdot (t_{s'} - t_s)$ because the completion of x_i is modified from $c_{s'} + i \cdot t_{s'}$ to $c_{s'} + i \cdot t_s$. Therefore the difference is

$$i \cdot (t_{s'} - t_s)(|T(x_i)| - |T(y_i)|) \le 0$$

Therefore we have the lemma.

By repeating the procedure in Lemma 3.1, we can find a schedule such that processors receive the message in non-decreasing order of their transmission times and the total completion time is no more than the optimal. We thus conclude:

THEOREM 3.1. *Algorithm FNF minimizes the total completion time.*

In fact, we can prove even stronger result by applying the same procedure only to a subset of processors p_i $(1 \le i \le k)$ for any $k \le n$.

COROLLARY 3.1. *Algorithm FNF minimizes the sum of completion times over all processors* p_i $(1 \le i \le k)$ *for any* $k \le n$.

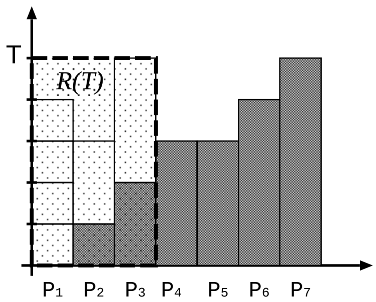
Proof. We do the same procedure as in Lemma 3.1 except that we only count processors p_i $(1 \le i \le k)$ when we compute the sizes of subtrees.

We will use this corollary in the next section to prove that the FNF scheme gives a 1.5-approximation for minimizing broadcast time.

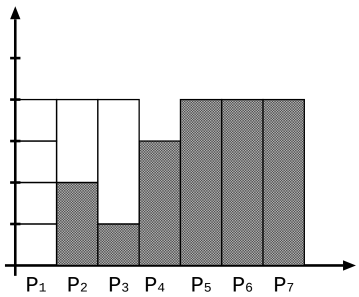
4 1.5-approximation for minimizing Broadcast Time

Liu [12] proved that FNF scheme gives 2-approximation for minimizing the broadcast time. In this section, we prove that FNF scheme gives 1.5-approximation.

Let us consider the bar chart as shown in Figure 3 (a), where processors are listed in non-decreasing order of transmission time in the horizontal line and each processor has a block whose height corresponds to its completion time in a schedule. (This figure corresponds to the schedule created on the instance specified in Figure 1.) We call these bars as *grey blocks*. Each processor p_i can start sending messages as soon as it receives the message, and send to one processor in each t_i time unit. Therefore, the height of a *white block* is same as the transmission times of the corresponding processors.



(a) FNF



(b) Optimal Solution

Figure 3: An example of bar charts corresponding to the schedules created by the instance specified in Figure 1

DEFINITION 4.1. *We define the number of* fractional blocks *as the number of (fractional) messages a processor can send by the given time. In other words, given a time T, the number of fractional blocks of processor p_i is $(T - c_i)/t_i$.*

For example, the number of fractional blocks of p_2 is 2.

DEFINITION 4.2. *We define $R(T)$ as the rectangular region bounded by time 0 and T and including processors from p_1 to $p_{\lfloor n/2 \rfloor}$ in the bar chart.*

An example is shown in Figure 3.
We only include processors $p_1, p_2, \ldots p_{\lfloor n/2 \rfloor}$ in $R(T)$ because of the following lemma.

LEMMA 4.1. *There is an optimal schedule in which only the $\lfloor n/2 \rfloor$ fastest processors send messages, that is, processor $p_i(\lfloor n/2 \rfloor + 1 \leq i \leq n)$ need not send any messages.*

Proof. We prove this by showing that there is an optimal broadcast tree in which every internal node has at least one leaf child. Suppose an internal node s does not have a leaf child. Then we move the processor which receives the message last in subtree $T(s)$ to a child of s. It is easy to see that this does not increase the makespan of the schedule by Theorem 2.2. By repeatedly applying this modification to the given broadcast tree, we can find an optimal broadcast tree satisfying the property.

Let the makespan of an optimal schedule and FNF be T_{OPT} and T_{FNF}, respectively. Then we have the following lemma.

LEMMA 4.2. *Algorithm FNF maximizes the number of fractional blocks in $R(T)$ for any $T_{OPT} \leq T \leq T_{FNF}$.*

To prove this lemma, we first prove the following proposition.

PROPOSITION 4.1. *We have $\sum_{i=1}^{m} a_i/b_i \geq \sum_{i=1}^{m} a_i'/b_i$ if $\sum_{i=1}^{l} a_i \geq \sum_{i=1}^{l} a'_i$ for all $1 \leq l \leq m$ and $0 < b_i \leq b_{i+1}$ for all $1 \leq i \leq m-1$.*

Proof. We will show that for all $1 \leq l \leq m$

$$\sum_{i=1}^{m} \frac{a_i}{b_i} \geq \sum_{i=1}^{l} \frac{a_i'}{b_i} + \sum_{i=l+1}^{m} \frac{a_i}{b_i} + \sum_{i=1}^{l} \frac{a_i - a_i'}{b_l}.$$

Then if we set l as m, we have the proposition as $\sum_{i=1}^{m} a_i \geq \sum_{i=1}^{m} {a'}_i$.

We prove this by induction. For $l = 1$, it is clearly true since

$$\sum_{i=1}^{m} \frac{a_i}{b_i} = \frac{a'_1}{b_1} + \sum_{i=2}^{m} \frac{a_i}{b_i} + \frac{a_1 - a'_1}{b_1}.$$

Suppose that it is true when $l = k$. Then

$$\begin{aligned}
\sum_{i=1}^{m} \frac{a_i}{b_i} &\geq \sum_{i=1}^{k} \frac{a'_i}{b_i} + \sum_{i=k+1}^{m} \frac{a_i}{b_i} + \sum_{i=1}^{k} \frac{a_i - a'_i}{b_k} \\
&\geq \sum_{i=1}^{k} \frac{a'_i}{b_i} + \sum_{i=k+1}^{m} \frac{a_i}{b_i} + \sum_{i=1}^{k} \frac{a_i - a'_i}{b_{k+1}} \\
&= \sum_{i=1}^{k+1} \frac{a'_i}{b_i} + \sum_{i=k+2}^{m} \frac{a_i}{b_i} + \sum_{i=1}^{k+1} \frac{a_i - a'_i}{b_{k+1}}.
\end{aligned}$$

Proof of Lemma 4.2. Let us denote the completion time of p_i in FNF and any other given schedule as c_i and c'_i, respectively. We need to show that $\sum_{i=1}^{\lfloor n/2 \rfloor} (T - c_i)/t_i \geq \sum_{i=1}^{\lfloor n/2 \rfloor} (T - c'_i)/t_i$. Since we have Proposition 4.1 and $t_i \leq t_{i+1}$, it is enough to show that $\sum_{i=1}^{l} (T - c_i) \geq \sum_{i=1}^{l} (T - c'_i)$ for all $1 \leq l \leq \lfloor n/2 \rfloor$. This is true because $\sum_{i=1}^{l} c_i \leq \sum_{i=1}^{l} c'_i$ for all $1 \leq l \leq \lfloor n/2 \rfloor$ by Corollary 3.1.

LEMMA 4.3. *The number of fractional blocks by FNF in $R(T_{FNF})$ is at most $3n/2 - 2$.*

Proof. If a processor receives the message from processor p_i, then it can be mapped to a white block of p_i. To finish the schedule, we should send the message to $n-1$ processors and therefore, there are $n-1$ white blocks in $R(T_{FNF})$. In addition, each processor $p_i (1 \leq i \leq \lfloor n/2 \rfloor)$ may have a fraction of block which is not finished by time T_{FNF}. But at least one processor should have no incomplete block since the makespan of FNF is T_{FNF}. Thus the number of fractional blocks is at most $(n-1) + \lfloor n/2 \rfloor - 1 \leq 3n/2 - 2$.

THEOREM 4.1. *We cannot have more than $(n-1)$ fractional blocks in $R(\frac{2}{3}T_{FNF})$ in any schedule.*

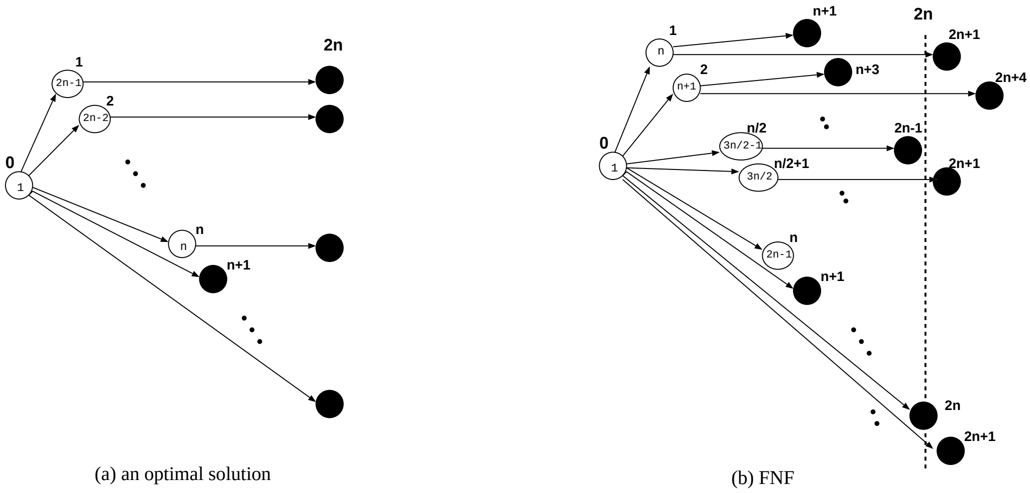
Proof. It is enough to show that FNF can have at most $(n-1)$ fractional blocks in $R(\frac{2}{3}T_{FNF})$ since FNF maximizes the number of blocks by Lemma 4.2. By time $\frac{2}{3}T_{FNF}$, each processor can have only $\frac{2}{3}$ of the number of blocks it has in $R(T_{FNF})$. Let f_i be the number of fractional blocks of processor p_i in $R(T_{FNF})$ and f'_i be the number of fractional blocks in $R(\frac{2}{3}T_{FNF})$. Since $f_i = \frac{T_{FNF} - c_i}{t_i}$ and $f'_i = \frac{\frac{2}{3}T_{FNF} - c_i}{t_i}$ we have $f'_i \leq \frac{2}{3}(\frac{T_{FNF} - \frac{3}{2}c_i}{t_i}) \leq \frac{2}{3}f_i$ Therefore, in $R(\frac{2}{3}T_{FNF})$ we can have at most $\frac{2}{3} \cdot (3n/2 - 2) \leq n - 1$ fractional blocks.

COROLLARY 4.1. *Algorithm FNF gives 1.5-approximation.*

Proof. Since we need send the message to $n-1$ processors in the optimal schedule, we should have $n-1$ blocks in $R(T_{OPT})$. It implies that $T_{OPT} \geq \frac{2}{3}T_{FNF}$.

When transmission times are in a small range, the FNF heuristic gives a better bound.

THEOREM 4.2. *Let $t_1 ... t_{\lfloor n/2 \rfloor}$ be the transmission times of the fastest $\lfloor n/2 \rfloor$ processors. Suppose $C' = (n/2 - 1)/\sum_{i=1}^{\lfloor n/2 \rfloor} 1/t_i$ then the FNF heuristic finds a solution of cost at most $T_{OPT} + C'$.*



(a) an optimal solution

(b) FNF

Figure 4: A bad example of FNF. The number inside a node is the transmission time of the processor and the number next to a node is the time it received the message. Black nodes are very slow processors. At time $2n$ (the dotted line), the optimal solution finished broadcasting but in FNF schedule, $\frac{n}{2}$ processors have not received messages yet.

Proof. In the bar chart of an optimal solution, the number of fractional blocks we can have between height T_{FNF} and T_{OPT} is $t/t_1 + t/t_2 + \ldots + t/t_{\lfloor n/2 \rfloor}$ where $t = T_{FNF} - T_{OPT}$. Since we have at least $n-1$ fractional blocks in $R(T_{OPT})$ and at most $3n/2 - 2$ fractional blocks in $R(T_{FNF})$, $\sum_{i=1}^{\lfloor n/2 \rfloor} t/t_i \leq n/2 - 1$. Therefore $T_{FNF} - T_{OPT} \leq (n/2-1)/\sum_{i=1}^{\lfloor n/2 \rfloor} 1/t_i$.

COROLLARY 4.2. *If the transmission times of the fastest $\lfloor n/2 \rfloor$ processors are in the range $[1 \ldots C]$, then the FNF heuristic finds a solution of cost at most $T_{OPT} + C$.*

COROLLARY 4.3. *If we have $\lfloor n/2 \rfloor$ processors with speed 1 then FNF gets a solution that is at most $T_{OPT} + 1$.*

5 Bad Example

Bhat et al [5] gave an example and proved that the broadcast time by FNF can be $\frac{17}{16}$ times the optimal in the example. In this section, we show that in fact, FNF gives the broadcast time of $\frac{25}{22}$ times the optimal in the same example.

Consider the example shown in Figure 4. We have the source with transmission time 1 and $2n$ processors with very slow transmission time. Also there are n processors with transmission times $n, n+1, \ldots, 2n-1$. In the optimal schedule, the source should send messages to processors with transmission time $2n-1, 2n-2, \ldots, n$, respectively. In other words, at time i the node with transmission time $2n-i$ receives the message from the source. Immediately after receiving the message, each of these processors send a message to one of the slow processors. The schedule completes at time $2n$.

In the FNF schedule, the source sends messages to processors with transmission time $n, n+1, \ldots, 2n-1$, respectively and again immediately after receiving the message, each of these processors send a message to one of the slow processors. At time $2n$, $\frac{n}{2}$ of the slow processors have not yet received the message. After time $2n$, processor with transmission time $n+i-1$ will send another message to a slow processor at time $2n + 3i - 2$. At the sam time, processor with transmission time $\frac{3n}{2} + i - 1$ send a message at time $2n + 2i - 1$. The source sends a message every time unit. Therefore, if t is the time needed to send messages to the remaining $\frac{n}{2}$ processors, then we have

$$\lfloor \frac{t+2}{3} \rfloor + \lfloor \frac{t+1}{2} \rfloor + t \geq \frac{n}{2}.$$

Therefore, we need at least $\frac{3n}{11} - \frac{7}{11}$ additional time. That means that the broadcast time of FNF can be $\frac{25}{22}$ times the optimal for large values of n.

6 Multicast

A multicast operation involves only a subset of processors. By utilizing fast processors which are not in the multicast group, we can reduce the multicasting time significantly. For example, suppose that we have m processors with transmission time t_1 and m more processors with transmission time t_2 where $t_1 < t_2$. Let we want to multicast a message to all processors with transmission time t_2. If we only use processors in the multicast group, it will take $t_2 \cdot \log m$ time. But if we utilize processors with transmission time t_1, we can finish the multicast in $t_1 \cdot (\log m + 1)$. Therefore, when $t_1 \ll t_2$, the speed-up is significant.

THEOREM 6.1. *Suppose that we have ρ-approximation algorithm for broadcasting. Then we can find a ρ-approximation algorithm for multicasting.*

Proof. Note that if an optimal solution utilizes k processors not in the multicast group, then those processors are the k fastest ones. Therefore, if we know how many processors participate in multicasting, we can use our ρ-approximation algorithm for broadcasting. By trying all possible k and taking the best one, we have ρ-approximation for multicasting.

7 NP-Completeness

We first show that the following problem is NP-complete. Then we show how to use this problem to derive a proof for the NP-hardness of minimizing the broadcast time.

The regular 3-PARTITION problem can be stated as follows: Given a finite set A of $3m$ elements, an integer B and a size $s(a)$ for each element in A, such that $\frac{B}{4} < s(a) < \frac{B}{2}$. Can A be partitioned into m disjoint sets $S_1, \ldots, S_m$ such that for $1 \leq i \leq m$, $\sum_{a \in S_i} s(a) = B$?

Notice that the constraints imply that each set S_i contains exactly three elements.

We can prove that the following special version of 3-PARTITION is NP-complete. We call it the SEEDED 3-PARTITION problem. Given a finite set A of $3m$ elements, where m is a power of 2, an integer B and a size $s(a)$ for each element in A, such that $\frac{B}{4} < s(a) < \frac{B}{2}$. Also specified is a (seed) subset A' of A, such that each element $a' \in A'$ has $s(a') < B(\frac{1}{4} + \epsilon)$ for a small positive constant ϵ, and each element $a \in A \setminus A'$ has $B(\frac{3}{8} - \epsilon) < s(a) < B(\frac{3}{8} + \epsilon)$. Can A be partitioned into m disjoint sets $S_1, \ldots, S_m$ such that for $1 \leq i \leq m$, $\sum_{a \in S_i} s(a) = B$ and each S_i contains exactly one element from A'?

We show how to reduce SEEDED 3-PARTITION to minimizing the broadcast time in heterogenous networks. Formally, given n processors, each with message transmission time t_i, is there a broadcast schedule to broadcast a message from one source processor to all other processors in T units of time?

The reduction works as follows: We create a set S of m processors each with transmission time 1. In addition we have a set S' containing $2m$ processors, m of them have transmission time $1.5 - 8\epsilon$, and m of them have transmission time $1.5 + 8\epsilon$. We also have a set S'' of m processors each with transmission time 2. Corresponding to each element in $a \in A$ we create a processor $p_a \in A$ with transmission time $t_a = \frac{s(a)}{B/4}$. For each element $a \in A'$ we create an additional set of processors $\bar{p}_a \in \bar{A}$ with transmission time $\bar{t}_a = \frac{B-2s(a)}{B/4}$. We also create a set M of $9m$ processors with extremely high transmission times. The processors in M cannot be used to send any other processors a message if the broadcast has to complete in time T. The source of the message is a processor in S. We set $T = \log m + 5$. (Since m is a power of 2, T is an integer.)

First observe that all the processors have transmission times between 1 and 2. For the processors in set A since $\frac{B}{2} > s(a) > \frac{B}{4}$, this follows. In fact, for the processors in A' the transmission times are strictly between 1 and $1 + 4\epsilon$. For the processors in $A \setminus A'$ the transmission times are strictly between $1.5 - 4\epsilon$ and $1.5 + 4\epsilon$. For the processors in $\bar{A}$, notice that the transmission times are almost 2.

We first show that if there is a solution to the SEEDED 3-PARTITION problem, then there is a solution to the broadcasting problem with $T = (\log m + 1) + 4$. The source message first broadcasts the message to processors in S. This takes $\log m$ transmission steps (each of which takes 1 unit of time. Suppose the solution of SEEDED 3-PARTITION is represented by sets S_i (see Figure 5). Let x_i^1, x_i^2, x_i^3 be the 3 elements in S_i, with $s(x_i^1) \leq s(x_i^2) \leq s(x_i^3)$. We can assume that $x_i^1 \in A'$ since this was a property of SEEDED 3-PARTITION. In the next transmission step, all the processors of S can simultaneously send messages to the processors corresponding to x_i^1 for $i = 1 \ldots m$. This takes another 1 unit of time. We have used up $\log m + 1$ time units. At this stage there are 4 time units remaining. We have processor $p_{x_i^1}$ send its first message to $p_{x_i^2}$, its second message sent to processor $\bar{p}_{x_i^1}$, which in turn sends a message to a processor in M. The third message sent by $p_{x_i^1}$ goes to a processor in M. Meanwhile $p_{x_i^2}$ sends its first message to $p_{x_i^3}$ and the second message to a processor in M. Also $p_{x_i^3}$ sends the message to a processor in M.

Since $s(x_i^1) + s(x_i^2) + s(x_i^3) = B$, this transmission chain takes exactly 4 units of time. Also note that $s(x_i^1) + 2s(x_i^2) \leq B$ and thus this chain has length at most 4 as well. The path from $p_{x_i^1}$ to $\bar{p}_{x_i^1}$ has length $2\frac{s(x_i^1)}{B/4} + \frac{B-2s(x_i^1)}{B/4} = 4$ as well.

Since there are 4 time units remaining at time T', each processor in S can send out 4 more messages. The first message goes to the faster processor in S'. Thus half the processors in S' have completion time $(1 + \log m) + 1$. These in turn then send the message to the other half (slower) processors in S'. Thus half the processors in S' have completion time $(1 + \log m) + 1 + 1.5 - 8\epsilon$. At this time the nodes in S' can send messages to $2m$ nodes from M in the remaining time.

The second message sent by processors in S goes to processors in S''. These have completion time $(1 + \log m) + 2$. In the remaining time, the processors in S'' can each send a message to a node in M.

The third and fourth messages from processors in S go to processors in M. Thus $2m$ processors in M get their messages directly from processors in S.

We now prove that if there is a solution to the broadcast problem in T time units, then there is a solution to the SEEDED 3-PARTITION problem. Let $T' = \log m + 1$. Consider an optimal broadcasting schedule that takes $T' + 4$ time units. We first recall the following theorem shown by Liu [12].

THEOREM 7.1. *[12] There exists an optimal broadcast sequence (ordered by completion times) in which all the fastest processors appear before all of the other processors.*

We break our schedule into two groups - Group I consists of the processors that receive the message strictly before time T'. Group II consists of the processors that receive the message at or after T' (and clearly before T). Using Theorem 7.1 it is easy to see that Group I consists of the processors in S in an optimal solution, since all these have transmission time 1. Group II consists of the processors in $S', S'', A, \bar{A}$ and M. For the Group II processors, we wish to argue that they have a particular form.

We analyze the subtree $T(i)$ rooted at a single processor $i \in S$ at time $T' - 1$. This subtree includes the set of processors that receive a message, either directly or indirectly through this processor in S. Since the transmission time is 1, at each time $t = T', \ldots, T' + 4$ some processor receives a message. We refer to the processor that receives a message at time $T' + j, j = 0 \dots 4$ as G_j (see Figure 6).

We now examine the maximum sizes each subtree rooted at G_j can have. Recall that the total number of processors in $S' \cup S'' \cup A \cup \bar{A} \cup M$ is $16m$. Thus the average number of processors in the subtree $T(i)$ is 16. We will argue that the maximum number of processors is exactly 16.

The subtree rooted at G_0 has size at most 8. This is because since all the processors in this subtree have transmission time > 1, we can only have 3 rounds of transmissions, and not 4.

Similarly, the subtree rooted at G_1 has size at most 4, since there are only 2 rounds of transmission, and not 3. The subtree rooted at G_2 has size at most 2, and the subtrees rooted at G_3 and G_4 each have size at most 1. This implies an upper bound of 16 on the number of processors in $T(i)$. Thus each subtree $T(i)$ must have exactly 16 processors.

This subtree has 9 leaves, and these must all be chosen from M as there are a total of $9m$ nodes in M, and 9 are chosen by each $T(i)$ subtree. (Recall that these processors cannot do any transmissions as they are very slow.)

Observe that a processor with speed 2 (from S''), must be chosen as G_2, otherwise we cannot complete the transmission to 16 processors in $T(i)$. In fact each $T(i)$ group can have only one processor from S'' for this reason. Processors G_3 and G_4 are both chosen from M, as are all the leaves in the subtree $T(i)$.

Note that only processors from A' can be G_0 since the transmission time of G_0 cannot be greater than $4/3$. Let us denote it as $p_{x_i^1}$. Then w.l.o.g, we can assume that $\bar{p}_{x_i^1}$ is the second child of $p_{x_i^1}$ (if not, we can exchange the second child with $\bar{p}_{x_i^1}$ and keep the broadcast time within $\log m + 5$).

In the subtree rooted at G_0, the first child and first grandchild of G_0 are not fixed and the broadcast time of this chain (starting from G_0 to M) should be at most 4. In the subtree rooted at G_1, G_1 and its first child is available and the broadcast time of this chain should be at most 3. In fact since we have to place all processors from A and S' in these two chains and the sum of transmission times of those processors is $7m$. We can show the chain starting from G_0 should be exactly 4 and the chain starting from G_1 exactly 3 (otherwise, we cannot include all processors).

Now we prove that processor from S' cannot be in the subtree rooted at G_0. This is because any combination of processors from A and S' cannot sum to exactly 4. Once we establish this, it is easy to see that all three processors corresponding to the 3 elements of the 3-set are all in one chain in the G_0 subtree. This implies that $\frac{s(x_i^1)+s(x_i^2)+s(x_i^3)}{B/4} = 4$. Thus the sum of the sizes is exactly B.

References

[1] *Message Passing Interface Form.* March 1994.

[2] M. Banikazemi, V. Moorthy and D. K. Panda. Efficient Collective Communication on Heterogeneous Networks of Workstations. *International Conference on Parallel Processing* , 1998.

[3] A. Bar-Noy, S. Guha, J. Naor and B. Schieber. Multicasting in Heterogeneous Networks. *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, 1998, pp. 448-453.

[4] A. Bar-Noy and S. Kipnis. Designing broadcast algorithms in the Postal Model for Message-passing Systems. *Mathematical Systems Theory*, 27(5), 1994.

[5] P. Bhat, C. Raghavendra and V. Prasanna. Efficient Collective Communication in Distributed Heterogeneous Systems. *Proceedings of the International Conference on Distributed Computing Systems*, 1999.

[6] J. Bruck, D. Dolev, C. Ho, M. Rosu and R. Strong, Efficient Message Passing Interface(MPI) for Parallel Computing on Clusters of Workstations. *J. Parallel Distributed Computing*, 40 (1997), pp. 19-34.

[7] M.R.Garey and D.S.Johnson Computer and Intractability: A Guide to the Theory of NP-completeness, Freeman, New York, 1979.

[8] W. Gropp, E. Lusk, N. Doss and A. Skjellum. A High-performance, portable Implementation of the MPI: a Message Passing Interface Standard. *Parallel Computing* 22(1996), pp. 789-828.

[9] S.M.Hedetniemi, S.T. Hedetniemi and A.L.Liestman. A Survey of Gossiping and Broadcasting in Communication Networks, *Networks* 18(1991), pp.129-134.
[10] R. Karp, A. Sahay, E. Santos and K.E.Schauser. Optimal Broadcast and Summation in the Logp Model. *Proceedings of 5th Annual Symposium on Parallel Algorithms and Architectures*, 1993 pp. 142-153.
[11] R. Kesavan, K. Bondalapati and D. Panda. Multicast on Irregular Switch-based Networks with Wormhole Routing. *Proceedings of the International Symposium on High Performance Computer Architectures*, 1997, pp. 48-57.
[12] P. Liu. Broadcasting Scheduling Optimization for Heterogeneous Cluster Systems. *Journal of Algorithms* 42, pp.135-152, 2002.
[13] P. Liu and D. Wang. Reduction Optimization in Heterogeneous Cluster Environments. *Proceedings of the International Parallel and Distributed Processing Symposium.* 2000, pp. 477-482.
[14] P. Liu and T-H. Sheng. Broadcasting Scheduling Optimization for Heterogeneous Cluster Systems. *ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, 2000, pp 129–136.
[15] D. A. Patterson, D. E. Culler and T. E. Anderson. A case for NOWs (Networks of Workstations). *IEEE Micro*, 15(1) (1995), pp.54–64.
[16] J. Pruyne and M. Livny. Interfacing Condor and PVM to Harness the Cycles of Workstation Clusters. *Journal on Future Generations of Computer Systems*, 12(1996).
[17] D. Richards and A. L. Liestman. Generalization of broadcasting and Gossiping. *Networks* 18(1988), pp. 125-138.
[18] A. Sussman, personal communication.

A NP-Completeness Proofs

The goal of this section is to prove that SEEDED 3-PARTITION is NP-complete.

First some definitions.

3-DIMENSIONAL MATCHING: Given a set $M \subseteq W \times X \times Y$, where W, X, Y are all disjoint sets each having q elements. Does M contain a matching, i.e., a subset $M' \subset M$ such that $|M'| = q$ and no two elements of M' agree in any co-ordinate?

We first show that the following problem is NP-complete by a reduction from 3-DIMENSIONAL MATCHING. This problem is a variation of the standard "4 PARTITION" problem.

SMALL-LARGE 4 PARTITION: Given finite sets A_1 and A_2 each having $2m$ elements, and a bound B and a size $s(a)$ for each element, such that $\frac{B}{5} < s(a) < \frac{B}{3}$. Moreover, $s(a) > \frac{B}{4}$ for $a \in A_1$ and $s(a) < \frac{B}{4}$ for $a \in A_2$. Can we find m disjoint sets $S_1, \ldots S_m$ such that $\sum_{a \in S_i} s(a) = B$ and each set S_i includes *exactly* two elements from each A_j?

The proof for the NP-completeness of the SMALL-LARGE 4 PARTITION problem is almost the same as the proof of the standard 4 PARTITION problem [7]. We have to change some essential parameters in the reduction. These changes are somewhat technical and omitted from this abstract.

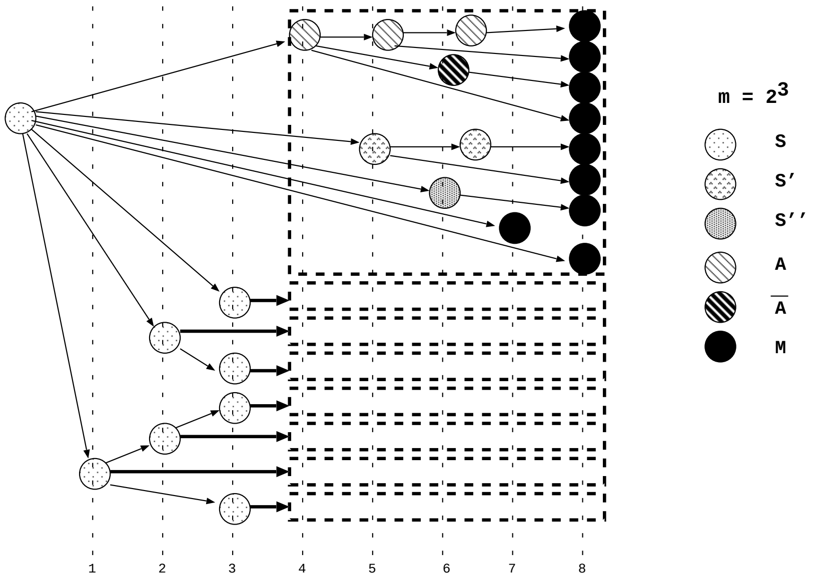


Figure 5: Reduction from SEEDED 3-PARTITION to minimizing the broadcast time in heterogeneous networks. Set S, S', S'' include processors with transmission time 1, $\{1.5 - 8\epsilon, 1.5 + 8\epsilon\}$, 2, respectively. Set A has processors with transmission time $t_{x_i^1}, t_{x_i^2}, t_{x_i^3}$ and processors in $\bar{A}$ have transmission time $\bar{t}_{x_i^1}$. M includes very slow processors.

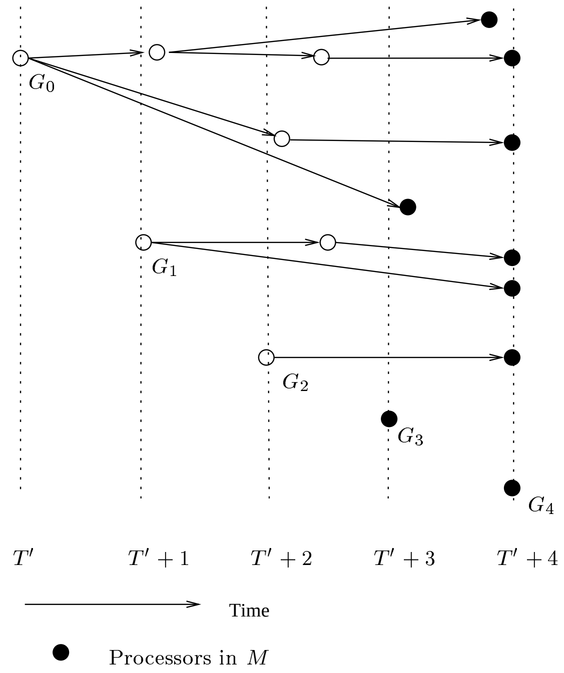


Figure 6: Figure to show last 4 steps of the broadcast schedule.