22 Mar 1990

A Fast O(k) Multicast Message Routing Algorithm

Thomas J. Sager

Bruce M. McMillin
Missouri University of Science and Technology, ff@mst.edu

A Fast O(k) Multicast Message Routing Algorithm

Thomas J. Sager and Bruce M. McMillin

CSC-90-2

March 22, 1990

Department of Computer Science
University of Missouri – Rolla
Rolla, MO 65401, U.S.A.
(314) 341-4491

This paper has been submitted for publication to the *Third Symposium on the Frontiers of Massively Parallel Computation* to be held at the University of Maryland, October 1990.

A Fast O(k) Multicast Message Routing Algorithm

Thomas J. Sager and Bruce M. McMillin
Department of Computer Science
University of Missouri-Rolla
Rolla, Mo. 65401 USA
phone: (314) 341-4856
email: tomsager@cs.umr.edu

Abstract

In many multicomputer applications is it necessary for one node to send an identical message to many nodes. One to many communications are called multicasts. Although broadcasts (one to all) and unicasts (one to one) have been widely implemented, multicasts, in spite of their importance to efficient use of multicomputer systems, have not received much attention.

Minimal traffic multicasting is equivalent to the minimal Steiner tree problem and is known to be $\mathcal{NP}$-complete. Therefore, a heuristic polynomial time approximation must be used; but, to take advantage of advances in second generation multicomputer hardware such as wormhole routing, a distributed multicast routing algorithm must have a low order of complexity, preferably $O(k)$ where k is the number of receiving nodes.

We present an $O(k)$ algorithm for multicast routing. It is simple enough to be easily implemented in hardware and has the advantage over previously presented $O(k)$ multicast routing algorithms that the choice of the channel on which to send to a particular destination depends only on statistical properties of those destinations already processed. Thus, a destination may be processed as soon as it arrives at a node. In addition, the algorithm we present appears to create in the mean less traffic than previously presented $O(k)$ multicast routing algorithms on a 10-cube topology.

Keywords and Phrases

Multicomputers, Heuristic Algorithms, $\mathcal{NP}$-completeness, Hypercube, Multicast Communication, Message Routing.

Acknowledgement

We would like to express our sincere appreciation to Dr. Lionel M. Ni of Michigan State University for presenting this problem to us and motivating us to research it.

1 Extended Abstract

1.1 Introduction

A *multicomputer* is composed of many interconnected processors. Each processor has its own local memory. There is no shared global memory. Interprocessor communication is accomplished entirely through message passing. A processor is directly connected to only a small number of other processors. Thus, a message may have to traverse several hops before arriving at its destination. The performance of a multicomputer is to a large extent dependent upon the hardware and software for message routing.

First generation multicomputers relied on the *store and forward* paradigm. In the store and forward paradigm, the entire message must be received and buffered before a decision can be made on what action to take (pass the message to host and/or forward to a particular set of neighboring nodes). Two other paradigms used in second generation multicomputers are *circuit switching* and *wormhole routing.* In circuit switching the path which a message will follow is determined before the message is sent. In wormhole routing a node receiving a message will scan the destination address and make a decision on how to forward the message before actually receiving the message. Thus, there is no need for the delay involved in buffering the message. The message merely passes through the routing circuitry without being buffered.

Routing algorithms may also be classified as either *centralized* or *distributed.* In centralized routing, routing decisions are made entirely by a single processor (perhaps the source node) whereas in distributed routing each node decides by itself on which channel to send or forward a message.

The manner in which the nodes or processors of a multicomputer are connected is called the *topology* of the multicomputer. One common topology is the *n-dimensional hypercube* or *n-cube* topology. In an n-cube, the multicomputer contains 2^n processors each of which has

a different n-bit address. Two processors are connected if and only if their addresses differ in exactly one bit position. Thus, each processor has exactly n neighbors. Hypercubes have become increasingly important and have been used in many different modern multicomputers such as the iPSC/2 [6] and the Symult Series 2010 [8].

Messages may also be classified according to the number of destinations. A message with a single destination is called a *unicast.* A message which must be sent to all nodes is called a *broadcast.* A message that has an arbitrary number of destinations is called a *multicast.* The limiting cases for a multicast are unicast and broadcast. Thus, multicast in a sense encompasses the entire spectrum. In the past, multicast has not received a lot of attention. Multicast messages have been implemented as broadcasts or as multiple unicast messages. The disadvantage of both of these methods is the quantity of unnecessary traffic generated which can lead to congestion and delay, or worse, deadlock or the necessity of discarding messages.

There are many applications for which multicasts are useful. One example is circuit simulation. The output of a gate may fan out and become the input to many other gates. On simulating the firing of a gate, its output should be multicast to all processors which simulate gates to which the output becomes an input.

Another application is the simulation of a global shared memory on a multicomputer containing only local memory. Here, writing on global memory is simulated by multicasting the written value to a subset of the total number of processors. Reading is performed by multicasting to a different subset of the processors requesting the desired value. Optimally, if there are N processors, each read set and each write set contains $\sqrt{N}$ processors and the intersection between each read set and each write set contains exactly one processor.

Multicast algorithms can be compared in various ways. One statistic is the *traffic* generated or total number of hops a multicast communication must traverse before arriving at all destinations. By minimizing traffic, congestion, delay and the possibility of deadlock

become less likely. Unfortunately minimizing traffic is equivalent to the minimal Steiner tree problem and is known to be $\mathcal{NP}$-complete [2, 5].

Another statistic is *destination-hops* or the sum of the number of hops the message travels to reach each destination separately. This can be minimized through the use of shortest path algorithms which are trivial on many popular architectures such as the hypercube. A third statistic is *routing delay* or the time each node must spend in routing a multicast message. This is to a large degree dependent upon the complexity of the routing algorithm, especially if it is to be implemented in hardware. Finally, we must be concerned with whether there is a possibility that the routing decisions might result in *deadlock.*

The literature contains only a few proposed multicast routing algorithms. Lan, Esfahanian and Ni [4] have proposed an $O(kn + n^2)$ algorithm on an *n-cube*, where k is the number of destinations of the multicast. This algorithm minimizes the number of destination-hops and has been implemented in hardware [3]. A drawback of this algorithm is that all destinations must be received before processing the destinations can begin. Also this algorithm can result in deadlock. The algorithm is distributed with local decisions being made at each node. The pseudo-code for this algorithm appears in Figure 1.

```
algorithm multicast routing of Lan et al.
begin
   receive destination-list, message
   while destination-list contains an address that differs from address of local node
                in at least one bit position do
      i ← bit position in which a maximal number of destinations
                on destination-list differ from local node
      remove from destination-list all destinations which differ from address of local node
                at bit position i and send on channel i with message
   if destination-list contains address of local node then
      send message to host
end multicast routing of Lan et al.
```

Figure 1: Pseudo-code for multicast routing algorithm of Lan et al.

Lin and Ni [5] propose a $O(k^2)$ algorithm based on a novel heuristic algorithm for the Steiner tree problem. The main drawback of this algorithm appears to be its relatively high complexity. Unlike the algorithm of Lan et al., it does not necessarily route to each destination by a shortest path, however it generates considerably less traffic than both the algorithm of Lan et al. and the algorithm we present here. This should not be unexpected considering its greater complexity. The algorithm of Lin and Ni is distributed and can result in deadlock.

Byrd, Saraiya and Delagi [1] have approached the multicast problem from the point of view of deadlock avoidance. They proposed a deadlock free algorithm which would appear to generate more traffic than the above two algorithms.

In this paper, we present the *bestfit* distributed multicast routing algorithm. It has $O(k \cdot \log n)$ complexity on an *n-cube*. In this algorithm we attempt to minimize the amount of traffic generated while maintaining the $O(k)$ complexity and routing to each destination over a shortest path. This algorithm could be particularly advantageous on second generation multicomputers which support wormhole routing. We envision that, if implemented in hardware, this algorithm could process the destinations of a multicast communication as they arrive and with little delay copy the bits of the following message as they arrive to however many channels necessary. Thus, we envision that the communication time between sending and receiving would be little greater than for a unicast.

We have programmed the *bestfit* algorithm and simulated it on various random sets of destinations on a 10-cube. We have found that in the mean for most destination sets of size 512 or less, our algorithm generates somewhat less traffic than the algorithm of Lan, Esfahanian and Ni [4], while also minimizing the number of hops between the source and each destination. However, the major advantage of this algorithm is not the slightly decreased amount of traffic, but the lower complexity and the corresponding increased speed with which a destination list may be processed and forwarding channels chosen.

1.2 The Bestfit Multicast Routing Algorithm

The *bestfit* multicast routing algorithm is shown in Figure 2. Its important features are:

1. It is distributed with complexity at each node of $O(k \cdot \log n)$ on an *n-cube* where k is the number of destinations.

2. The multicast communication travels to each destination along a shortest path.

3. The algorithm is simple enough to be easily implemented in hardware.

4. Since the decision on where to place a destination depends only on statistical properties of the destinations already processed, a destination may be processed as soon as it is received at a node.

5. It appears to generate less traffic in the mean than other $O(k)$ algorithms in the literature.

6. It will not necessarily avoid deadlock.

The algorithm basically tries to place each destination on the channel to which it *"fits best"*. That is, the current destination should look *"a lot like"* the *"average"* destination already assigned to the chosen channel. A destination is assigned to an empty channel if and only if its assignment to each non-empty channel would violate the constraint that for each channel there is at least one bit position for which all destinations assigned to that channel differ from the address of the local node.

parm is a small integer which is proportional to the original size of the multicast when it was created at the source. This parameter requires some further fine tuning. For multicasts of size less than 32, we set $parm = 1$. For multicast sets of size 32 through 400, we use $parm = 2$. For multicast sets of size 401 through 700, we use $parm = 8$ and for large

```
algorithm bestfit multicast routing
        Receive parm, destination list and message. Split destination list onto appropriate
        channels. Send parm, new destination lists and message on appropriate channels.
const n: size of hypercube;          localaddr: address of executing node
var
   last, winner: integer;            reladdr: address;
   accept, tohost: boolean;          message: string;
   channel: array [0..n − 1] of record
      acceptable: address;
      destlist: list of address;
      numbdest, score: integer;
      sum: array [0..n − 1, 0..1] of integer;
begin                                                                    Initialize
   last ← −1;          tohost ← false;
   for i ← 0..n − 1 in parallel do
      destlist ← empty;          acceptable ← 2^n − 1;
      for j ← 0..n − 1 in parallel do          sum[j, 0] ← 1;          sum[j, 1] ← 0;
                                                        Receive and Forward Multicast Message
   receive(parm);
   for i ← 0..n − 1 in parallel do channel[i].numbdest ← parm;
   while there are more destinations in multicast do
      receive(destination)
                                                                    Pick Winning Channel
      reladdr := destination xor localaddr;          accept ← false;
      if reladdr = 0 then tohost → true;
      else for i ← 0..last in parallel do
         with channel[i] do
            if (acceptable and reladdr) = 0 then score = −1
            else
               accept ← true;
               score ← (summation j ← 0..n − 1 in parallel sum[j, reladdr[j]])/numbdest;
      if accept then winner ← i ∈ 0..last ∋ maximal(score[i])
      else          last ← last + 1;          winner ← last;
                                                                    Update Winning Channel
      with channel[winner] do
         acceptable ← acceptable and reladdr;          incr(numbdest);
         for j ← 0..n − 1 in parallel do incr(sum[j, reladdr[j]]);
         enqueue(destlist, destination);
                                                                    Send Message
   while receiving(message) for i ← −1..last in parallel do
      if i = −1 and tohost then send message to host;
      else with channel[i] do
            choose j ∈ acceptable;          send parm, destlist, message on channel j;
end bestfit multicast routing;
```

Figure 2: Pseudo-code for *bestfit* multicast routing algorithm

multicasts of size 701 or greater, we set $parm = 16$. A multicast communication must therefore contain both *parm* and a *destination list* as well as a *message*.

In using this algorithm, we assume the destination list has first been sorted by the source node according to increasing distance from the source. Failure to do so causes the algorithm to seriously degrade. Note that this is easily done in a hypercube and does not add to the complexity of the total algorithm. Since there are only a small number, n, of possible distances, this sort can take place in $O(k)$ time as the source node creates the original destination list.

Note also that much of the algorithm is done in parallel. This is only natural as we intend it to be implemented in hardware. Within the "**while** there are more destinations" loop, every operation can be executed in linear time except for the summation of n numbers and the choice of the maximum of n numbers. Each of these operations requires $O(\log n)$ time. But, since n is small anyway, $n = log_2(N)$ where N is the number of processors, $\log n$ might be considered as virtually a constant.

If the number of destinations is very small, ≤ 4, then instead of using the algorithm in Figure 2, we exhaustively search for the best possible multicast tree. Since the number of reasonable candidates is quite small, they can be quickly processed in parallel and the best one chosen.

1.3 Results

We have programmed and tested both the *bestfit* algorithm and the algorithm of Lan et al. on a number of random sets of destinations on a *10-cube*. The random sets were all chosen using Park and Miller's minimal standard random number generator [7].

The destination set sizes which we tested were: 5 though 32, 64, 96, 128, 192, 256, 384, 512, 640, 768 and 896. 30 test cases were generated for each destination set size. The average traffic generated by each algorithm on the 30 random test cases is shown in Figure 3. The