Communicating Process Architectures 2005
Jan Broenink, Herman Roebbers, Johan Sunter, Peter Welch, and David Wood (Eds.)
IOS Press, 2005

Improving TCP/IP Multicasting with Message Segmentation

Hans Henrik HAPPE and Brian VINTER

Dept. of Mathematics & Computer Science, University of Southern Denmark, DK-5230 Odense M, Denmark.

{hhh , vinter} @imada.sdu.dk

Abstract. Multicasting is a very important operation in high performance parallel applications. Making this operation efficient in supercomputers has been a topic of great concern. Much effort has gone into designing special interconnects to support the operation. Today's huge deployment of NoWs (Network of Workstations) has created a high demand for efficient software-based multicast solutions. These systems are often based on low-cost Ethernet interconnects without direct support for group communication. Basically TCP/IP is the only widely supported method of fast reliable communication, though it is possible to improve Ethernet performance at many levels – i.e., by-passing the operating system or using physical broadcasting. Low-level improvements are not likely to be accepted in production environments, which leaves TCP/IP as the best overall choice for group communication.

In this paper we describe a TCP/IP based multicasting algorithm that uses message segmentation in order to lower the propagation delay. Experiments have shown that TCP is very inefficient when a node has many active connections. With this in mind we have designed the algorithm so that it has a worst-case propagation path length of $O(\log n)$ with a minimum of connections per node. We compare our algorithm with the binomial tree algorithm often used in TCP/IP MPI implementations.

Keywords. Multicasting, NoW, HPC

1. Introduction

Message multicasting is a highly investigated topic[1], partly because of the importance of multicasting performance in parallel programming[2]. In [3] it is found that 9.7% of all calls to the MPI[4] layer in the NAS[5] benchmark-suite are broadcast operations. In fact the only operations that are more frequent are the point-to-point send and receive operations.

Most work into multicast algorithms are very analytical and consider theoretical performance using quite simplified hardware models. Previous work has shown that at the system level, the optimal topology for broadcast algorithms are quite different from the theoretical findings[6].

In this work we move the findings from research in wormhole routed interconnects[7] into the software stack. In section 2 we describe the existing algorithms for multicasting in computational clusters. In section 2.1 we introduce the concept of segmenting messages in the software stack and in section 3 we show how segmentation may be used for multicasting. In this section we also introduce a multicasting tree that only requires each process to send the same amount of data as the size of the multicasted message, at the most. In section 4 the model is implemented and tested in two Ethernet based clusters.

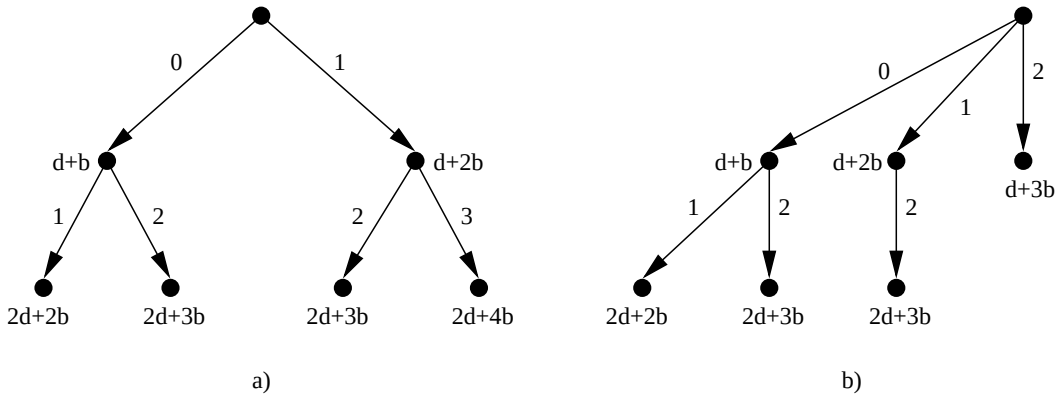

Figure 1. Multicast trees. Node expressions is the time at which the node receives the message and edge numbers are the step in which communication takes place. a) Binary multicast tree. b) Binomial multicast tree.

2. Point-to-Point Multicasting

Multicasting a message by means of point-to-point communication must be done in accordance to some algorithm. Basically this algorithm must select which destination processes should aid in spreading the message by forwarding to other destinations.

The optimal message path, when multicasting using point-to-point communication, will yield a tree where the root is the source process. This is easily realized from the fact that all other processes only need to receive the message once. In [8] an optimal multicast solution in the logP model[9] has been given. The tree structure in this solution depends very much on the parameters of the model. In real systems these parameters can be very dynamic and therefore a more practical approach is used to define a suitable tree structure.

In the following we will give a simple analysis of some of the classical tree structures often used in real systems. This analysis will be based on at simple network model where the time t to send a message from one process to another is defined as:

$$t = d + b \tag{1}$$

$$d = delay \tag{2}$$

$$b = \frac{message\ size}{max\_bandwidth}. \tag{3}$$

d is the one-byte latency and b is the time imposed by the maximum point-to-point bandwidth. The letter m will be used to denote the time for the whole multicast and n is the number of processes involved (source and destinations).

A simple way to implement multicasting is to have the source process send the message directly to all destination processes. This will give a multicast time of $m = (n-1)b + d$ which scales very poorly for large messages. For small messages this seams to be a good solution, but the source process will be very busy sending which means it cannot attend to other matters (i.e. calculation).

The next logical step is to make a higher tree with a constant fanout f as shown in Figure 1.a. Here $m \leq h(d + fb)$, where $h = \lfloor \log_f n \rfloor$ is the height of the tree. Because the height of the tree is a logarithmic function, this tree will give $O(\log n)$ time complexity. Given d, b and n it is possible to find the optimal fanout. An advantage of this multicast tree is that the work of sending is shared among many processes (those with children in the tree).

A binomial multicast tree is an optimization of a binary multicast tree. When a process in a binary multicast tree is done sending to its children it considers the multicast finished, while the children are still sending. Instead it could continue sending to other processes, which is

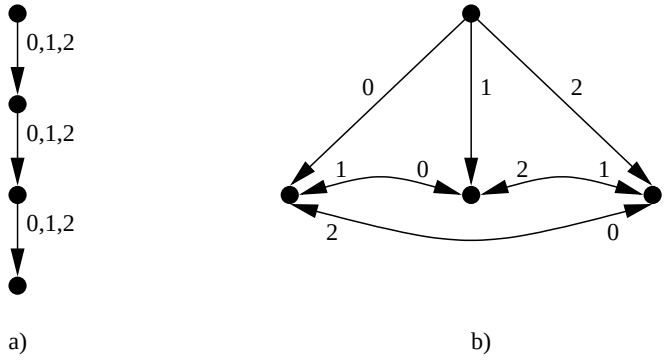

Figure 2. Segmentation multicast graphs. Edge numbers are the segments transfered along the path. a) Sequential tree (segmented seqtree). b) Basic idea of the optimal algorithm.

the idea behind the binomial tree (Figure 1.b). The structure of this tree ensures that every process that has received the message, participates in sending to those processes that have not received the message. As illustrated in the figure this gives better results than a plain binary tree because more processes work in each step. Trees with a constant fanout $f > 2$ could give better results for small messages, because the height of a binomial tree is $h = \lfloor \log_2 n \rfloor$. The uneven work that each process has to do is a disadvantage of binomial multicast trees.

2.1. Message Segmentation

A major problem with point-to-point multicasting is that the maximum multicast bandwidth cannot be more than half the maximum point-to-point bandwidth, when there is two or more destinations. Either the root sends to both destinations or one destination forwards to the next. This is only true when all of a message is received at a destination before it is forwarded. Message segmentation deals with this problem by splitting messages into smaller segments. Now forwarding can begin when the first segment has been received and together with the right multicast algorithm it is possible to multicast with a bandwidth that exceeds half the maximum point-to-point bandwidth.

The segment size s dictates the delay of relaying a message through a process. Theoretically s should be as small as possible in order to minimize this delay, but the actual splitting and forwarding imposes an overhead. These trade-offs has to be considered when choosing a proper segment size.

When using message segmentation the classical multicast trees are far from optimal. The problem is that some processes send the message more than once. This sets an upper bound $max\_bandwidth/f_{max}$ on the multicast bandwidth, where f_{max} is the maximum fanout in the multicast tree. In order to achieve multicast bandwidths near the maximum point-to-point bandwidth, the multicast algorithm has to ensure that the message size is the upper bound on how much data each process must transmit.

A sequential tree (a fanout of one) is the obvious choice for a multicast structure that utilizes segmentation (Figure 2.a). This structure ensures that all processes at most receive and send messages once. Theoretically the multicast time of this structure would be $m = (n-1)(d+b_s)+b$, where $b_s = s/max\_bandwidth$ is the time imposed by the maximum point-to-point bandwidth when transmitting a segment. This gives a time complexity of $O(n)$ which might be negligible for large messages (b will dominate), but for small messages the propagation delay $(n-1)(d+b_s)$ will dominate m. In the following this algorithm will be called “segmented seqtree”.

In [8] an optimal solution for multicasting a number of segments in the logP model is presented. The basic idea of this solution is that the source process scatters the segments to the destination processes in a round-robin fashion. When a destination process receives a segment

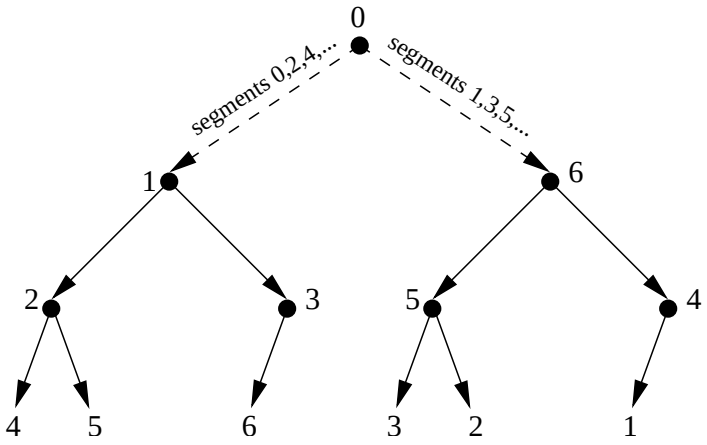

Figure 3. Segmented bintree algorithm. Process 0 is the source process.

it becomes the root of the segment and sends the segment to all other destination processes, by means of an optimal single segment multicast tree (Figure 2.b). A final multicast structure for each segment will depend on the logP parameters and the number of segments. Also, there can be special cases where those processes that are done can aid in finishing the multicast (end-game).

3. Segmented Multicasting in TCP/IP Networks

All the information needed in order to construct an optimal point-to-point multicast is hard to obtain at runtime. We have therefore tried a more general algorithm inspired by the optimal solution described in [8].

Figure 2.b illustrates how the algorithm works. The source process spreads out segments evenly to the destination processes. Each destination process sends the received segments to all other destination processes. The message size is an upper bound on how much data each process must send in this algorithm and each segment is only forwarded by one process.

We have evaluated the algorithm in an Ethernet based cluster using TCP/IP. Results showed that the segmented seqtree algorithm performed much better for large messages. Without further investigation we believe that TCP/IP does not handle multiple connections well. With this algorithm all processes have multiple connections that constantly are switching between activity and inactivity. This puts much pressure on the TCP subsystem in the form of buffer management which again increase memory activity (i.e. cache misses). It might also be TCP congestion control that cannot adapt fast enough. This is a subject of further research.

TCP connections will always use extra memory, so limiting the number of connections is preferable. The segmented seqtree algorithm has this feature but the path from the source to the last destination is $O(n)$ long. This will result in poor performance as the message size decreases.

We have devised an algorithm, we call “segmented bintree”, that falls in between the characteristics of two above. This algorithm uses a minimum of connections given these constraints:

1. The path length must be $O(\log n)$.
2. The message size is an upper bound on how much data a process has to forward.

Figure 3 illustrates how it works. Two binary trees containing all destination processes are build with the constraint that the total sum of children for each process is at most two. When multicasting the source process sends every other segment to the root of each of these

two trees. The segments are forwarded all the way down to the leaves with the result that all processes receives the message.

The first constraint obviously hold with this algorithm. Because of the manor in which the two trees are created, the second constraint also holds. Each process will at most have four active connections which is a minimum given the constraints. In the general case, the first constraint dictates that the source process or one of its descendants must forward to at least two processes. We will call this process x. x also has to have a descendant y that has to forward to at least two processes. The second constraint dictates that x cannot forward the whole message to y, hence y has to receive the remaining part from some other process z. z will not always be a child of y, because some z will have two or more children of its own. Therefore, there will exist a process y that has at least four connections given the constraints.

4. Experiments

The binomial, segmented seqtree and segmented bintree algorithms have been implemented using TCP sockets for communication. The algorithms have then been compared in two different clusters.

4.1. Clusters

The clusters used have the following configurations:

- *Gigabit:* 64 nodes, Intel Pentium 4 3.2 GHz CPUs, switched gigabit Ethernet network. Network adapter is connected to the CPU via the 32bit PCI bus, Linux 2.6.
- *CSA Gigabit:* 13 nodes, Intel Pentium 4 2.6 GHz CPUs, switched gigabit Ethernet network. Network adapter is connected to the CPU via Intel's Communications Streaming Architecture (CSA), Linux 2.6.

Note that the plain 64-node gigabit cluster is a production cluster which we had very limited access to (shared with other by means of a job queue). Therefore it was not possible to investigate irregularities in the following results. Also, the 32bit PCI bus connection to the network adapter makes full-duplex gigabit communication impossible.

The interconnect consisted of a set of 24-port switches with independent full-duplex 10 gigabit connections between them. We did not try to compensate for this heterogeneity when laying out the multicast trees, but all tests were run on same set of nodes. This issue should only affect the segmented algorithms, because these utilize parallel communication extensively. The fact that the results do not show any evidence of this heterogeneity, suggests that the 32bit PCI issue insured that the inter-switch links were not saturated.

The small CSA gigabit cluster has been included to test the algorithms in a gigabit cluster where close to maximum full-duplex gigabit bandwidth is possible.

4.2. Results

Both the segmented algorithms have been run with a segment size of 8KB throughout the tests. In general this size has proved to give good results, though it might be a subject of further study.

Time measurements have been carried out by letting all destination nodes notify the source node when the full message had been received. We have not compensated for this extra notification delay in the following results.

In all the results the multicast bandwidth is the bandwidth of the actual multicast and not the accumulated bandwidth of all communication lines.

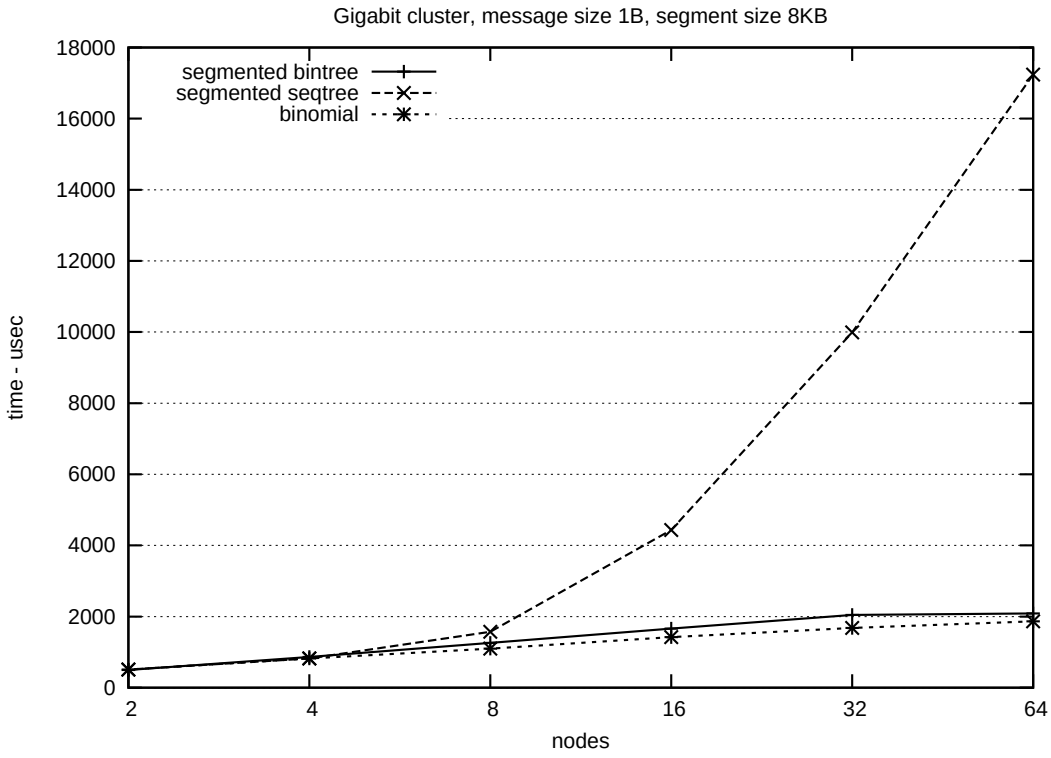

Figure 4. Multicast time/number of nodes. Latency in the gigabit cluster for one byte messages.

Overall the segmented bintree algorithm performs just as well or better than the binomial, for 32KB or larger messages. The segmented seqtree algorithm needs even larger message sizes before it generally outperforms the binomial, which was expected.

Figure 4 shows the segmented seqtree algorithm's problem with small messages. The binomial algorithm performs slightly better than the segmented bintree. This was expected because the segmented bintree algorithm has a slightly longer path to the last destinations, due to the final communication between the subtrees.

With 256KB messages in Figure 5 the segmented bintree algorithm generally outperforms the others. With 64 nodes it is 196% faster than the binomial algorithm. Also, when comparing the two and four node runs, we start to see that the nodes cannot handle full-duplex gigabit communication.

In Figure 6 with 8MB messages the segmented bintree algorithm scales very well. The bandwidth decrease is very small as the number nodes increases, while additional nodes has a much greater impact on the binomial algorithm. With 64 nodes the segmented bintree algorithm is 320% faster than the binomial. The performance of the segmented seqtree algorithm should be close to that of the segmented bintree with 8MB messages, but this is not the case. It must be an issue with the specific cluster because we do not see the same result in other smaller clusters (Figure 8).

Figure 7 shows the results for different message sizes with 64 nodes. The segmented bintree algorithm follows the binomial up to a message size of 32KB. As the message size increases beyond 32KB we see the effect of segmentation which makes it possible to increase its multicast bandwidth all the way up to the maximum, given the 32bit PCI issue (see the result for four nodes in Figure 6).

The importance of using nodes capable of full-duplex gigabit communication becomes very clear when looking at the results from the CSA gigabit cluster (Figure 8). Here the multicasting bandwidth reaches 82.6MB/s which is 74.4% of the maximum point-to-point TCP bandwidth, which has been measured to 111MB/s.

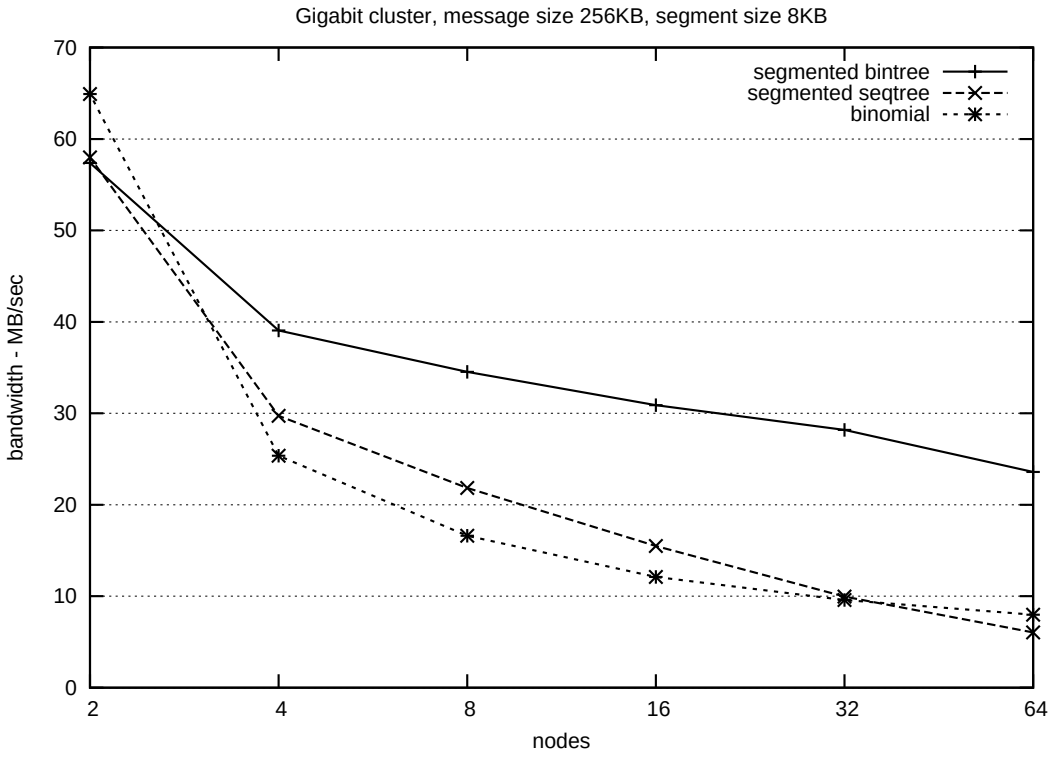

Figure 5. Multicast bandwidth/number of nodes. Multicast bandwidth in the gigabit cluster with 256KB messages.

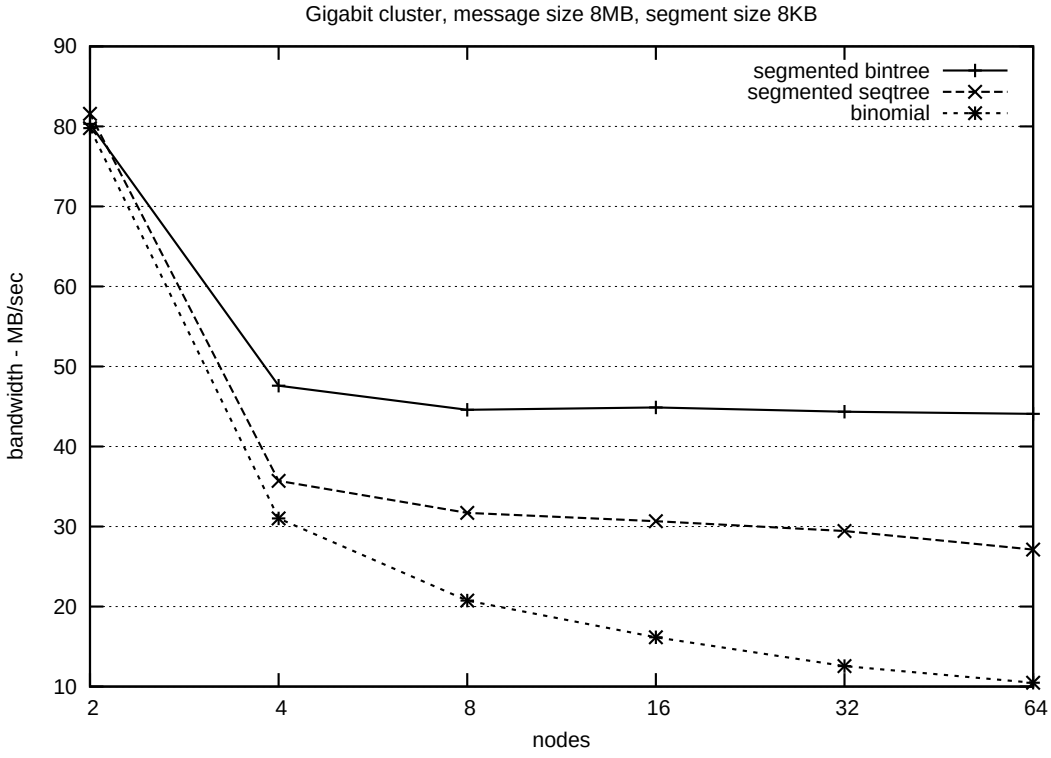

Figure 6. Multicast bandwidth/number of nodes. Multicast bandwidth in the gigabit cluster with 8MB messages.

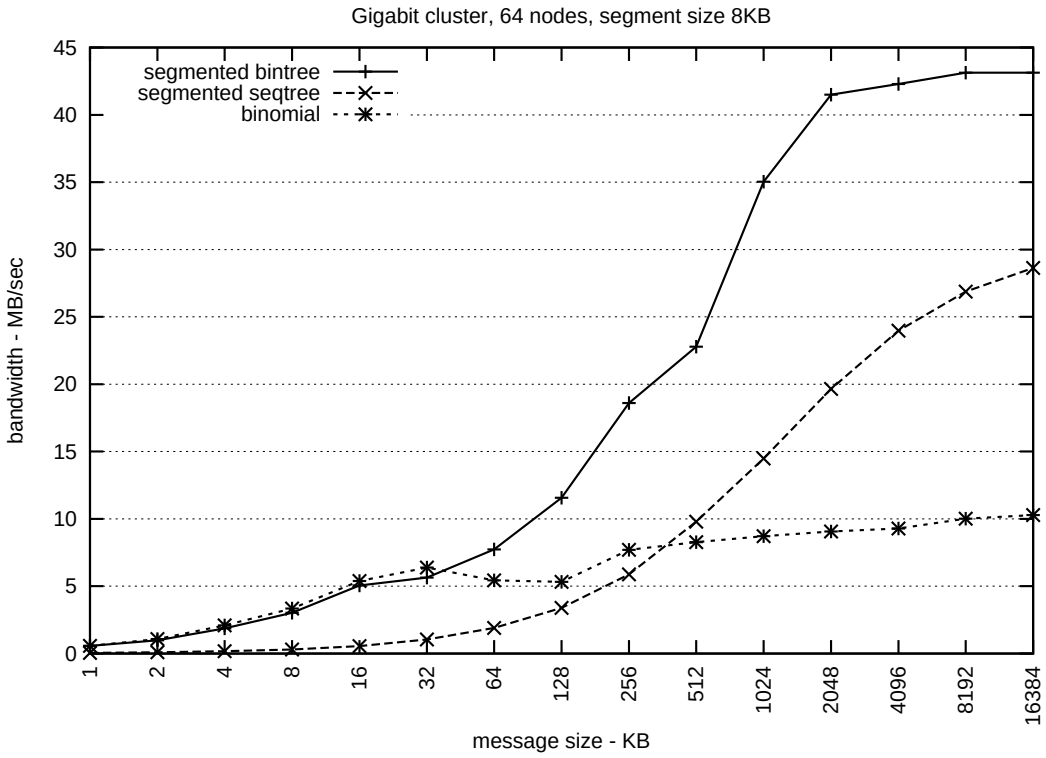

Figure 7. Multicast bandwidth/message size. Multicast bandwidth in the gigabit cluster with 64 nodes.

5. Conclusion

The goal of this work was to improve software-based point-to-point multicasting, by means of message segmentation. Tests has shown that minimizing the number of active connections reduces TCP/IP's communication overhead considerably. With this in mind, we have devised an algorithm that theoretically has an $O(\log n)$ time complexity, while only using four or less connections per process. This algorithm utilizes message segmentation in order to achieve multicasting bandwidths, close to the maximum point-to-point bandwidth. The algorithm can do this because no process sends more data than the size of the multicasted message. This also distributes the work evenly among the involved processes.

We have compared the algorithm with a more obvious segmentation algorithm (sequential tree) and the widely used binomial tree algorithm. The results have shown that our algorithm generally outperforms the binomial algorithm with 32KB or larger messages and in some test it were up to 320% faster. For messages smaller than 32KB the binomial algorithm wins with a small margin. Using another algorithm in this case could easily solve this problem.

References

[1] Taxonomy and Survey. Total order broadcast and multicast algorithms.
[2] Andrew S. Tanenbaum, M. Frans Kaashoek, and Henri E. Bal. Parallel programming using shared objects and broadcasting. *IEEE Computer*, 25(8):10–19, 1992.
[3] Ted Tabe and Quentin F. Stout. The use of the MPI communication library in the NAS parallel benchmarks. Technical Report CSE-TR-386-99, 17, 1999.
[4] Message Passing Interface Forum. MPI: A message-passing interface standard. Technical Report UT-CS-94-230, 1994.

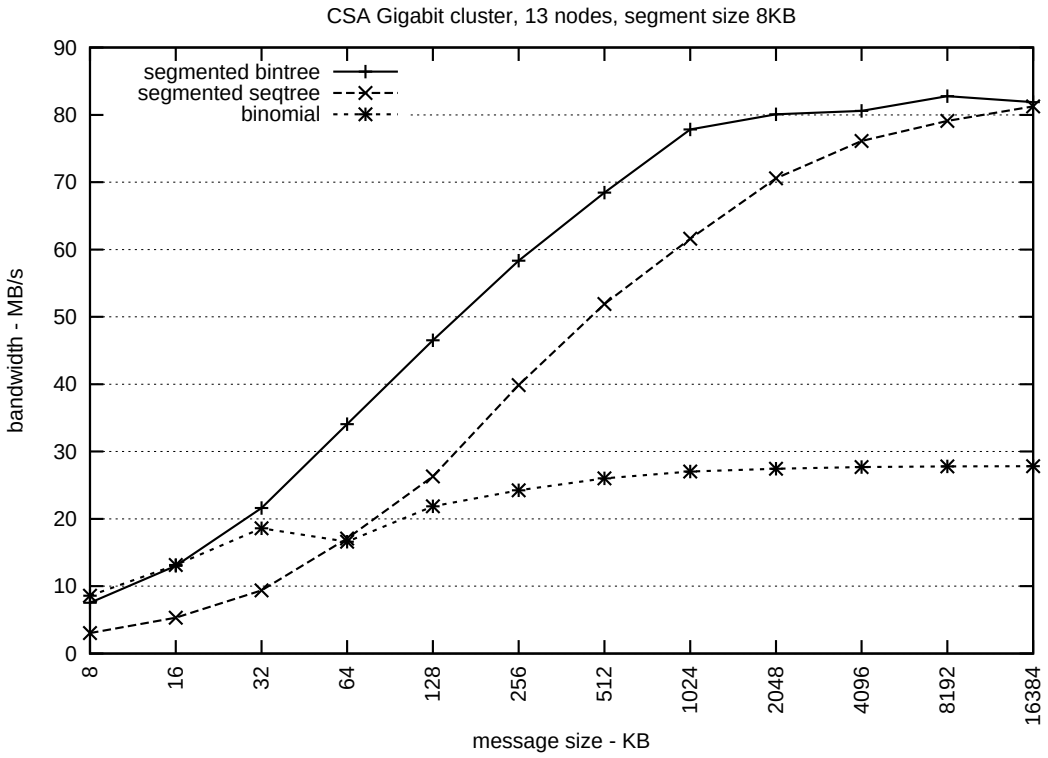

Figure 8. Multicast bandwidth/message. Multicast bandwidth in the CSA gigabit cluster with 13 nodes.

[5] D. H. Bailey, L. Dagum, E. Barszcz, and H. D. Simon. Nas parallel benchmark results. In *Supercomputing '92: Proceedings of the 1992 ACM/IEEE conference on Supercomputing*, pages 386–393. IEEE Computer Society Press, 1992.

[6] John Markus Bjørndalen, Otto J. Anshus, Tore Aarsen, and Brian Vinter. Configurable Collective Communication in LAM-MPI. In James Pascoe, Roger Loader, and Vaidy Sunderam, editors, *Communicating Process Architectures 2002*, pages 123–134, 2002.

[7] Lionel M. Ni and Philip K. McKinley. A survey of wormhole routing techniques in direct networks. *Computer*, 26(2):62–76, 1993.

[8] Richard M. Karp, Abhijit Sahay, Eunice E. Santos, and Klaus E. Schauser. Optimal broadcast and summation in the logP model. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 142–153, 1993.

[9] David E. Culler, Richard M. Karp, David A. Patterson, Abhijit Sahay, Klaus E. Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten von Eicken. LogP: Towards a realistic model of parallel computation. In *Principles Practice of Parallel Programming*, pages 1–12, 1993.