

A Scalable Control Topology for Multicast Communications *

Jörg Liebeherr

Department of Electrical Engineering
Polytechnic University
Brooklyn, NY 11201

Bhupinder S. Sethi

Department of Computer Science
University of Virginia
Charlottesville, VA 22903

Abstract

Large-Scale multicast applications for the Internet require the availability of multicast protocols that enhance the basic connectionless IP Multicast service. A critical requirement of such protocols is their ability to support a large group of simultaneous users. In this paper, we present a new approach for distributing control information within a multicast group. The goal of our approach is to scale to very large group sizes (in excess of 100,000 users). Multicast group members are organized as a logical n -dimensional hypercube, and all control information is transmitted along the edges of the hypercube. We analyze the scalability of the hypercube control topology and show that the hypercube balances the load per member for processing control information better than existing topologies.

1 Introduction

Recently emerging large-scale multicast applications [17] have increased the need for advanced multicast services on the Internet. These services are implemented on top of the basic connectionless IP Multicast service which does not guarantee reliable or in-sequence delivery [5].

In the basic multicast service, a user joins a multicast group simply by indicating interest in receiving data sent to that group. Any packet that is transmitted to a multicast group is forwarded to all members of the group. Error control, rate control, or in-sequence delivery are not part of the basic service. To implement these advanced services, multicast group members must exchange control information with each other. However, adding even relatively low-level functions, such as error control or flow control, to the basic multicast service may introduce severe scalability problems.

A major impediment for scalability of multicast applications is the need of multicast group members to exchange control information with each other. Con-

sider, for example, the implementation of a reliable multicast service. A unicast protocol with a single sender and a single receiver requires the receiver to send positive or negative acknowledgment packets (ACKs and NACKs) to the sender to indicate reception or loss of data. If the same mechanisms are applied to large groups, the sender would soon be flooded by the number of incoming ACK or NACK packets; this is referred to as *ACK Implosion* [4, 11].

In recent years, numerous techniques and protocol mechanisms have been proposed to cope with the amount of control information that is exchanged between members of a multicast group, mostly in the context of providing a reliable multicast service. In the more recent proposals, multicast group members are organized in a logical graph, henceforth called *control topology*. Only those group members which are neighbors in the logical graph can exchange control information. By merging control information received from their neighbors, the dissemination of control information can be made efficient. Control topologies that have been considered in the literature are rings [3, 22] and trees [10, 13, 15, 23].

This paper proposes a new approach for disseminating control information between the members of a multicast group. We present a topology that is derived from an *n -dimensional hypercube*. We claim that the hypercube topology has excellent scalability properties, making it the preferred choice for multicast applications with very large group sizes. The key contributions of this paper are twofold. First, we show how to construct a hypercube control topology with simple boolean operations. Second, we show that the hypercube balances the load for processing control information at multicast group members. Thus, the creation of bottlenecks in the control topology is avoided. In [14] we present a soft-state protocol that maintains the hypercube control topology without requiring any network entity to have global knowledge.

The remainder of the paper is structured as follows. In Section 2 we review the existing proposals for dis-

*This work is supported in part by a National Science Foundation CAREER Grant (NCR-9624106).

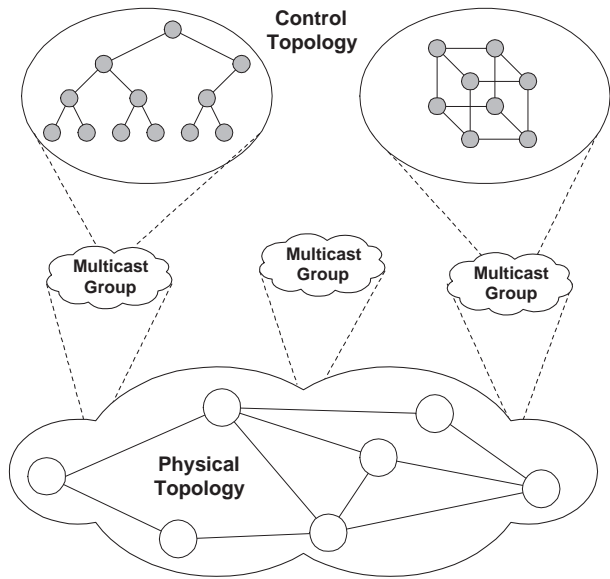


Figure 1: Multicast Framework.

seminating control information to the members of a multicast group. In Section 3 we present the hypercube as a new solution to disseminate control information in a multicast group. In Section 4 we analyze the scalability properties of a hypercube and compare them with other control topologies. In Section 5 we present our conclusions.

2 Control Topologies for Multicast Communications

In this section we review currently used control topologies for disseminating control information in multicast groups. An underlying assumption of our work is that communication within a multicast group is *symmetric*, i.e., on the average, each member of the group generates the same amount of traffic.

It is convenient to view the members of a multicast group as a set of nodes V . Nodes are numbered in an arbitrary sequence, that is $V = \{1, 2, \dots, N\}$. We assume that each node can directly communicate with any other member of the group.

2.1 No Control Topology

Several protocols that extend the basic IP Multicast service do not provide a topology for disseminating control information. Instead, the control information from any node is broadcast to all other nodes in the group. Clearly, such a protocol must restrict the volume of control information, since otherwise the scalability is severely impeded. The RTP protocol [20] limits the total amount of control traffic to 5% of the data

traffic. In [1], feedback from receivers to the senders is adapted to the size of the multicast group.

Among reliable multicast protocols, the most popular approach to contain control traffic without a control topology is a method known as *NACK suppression* [7, 19] or *damping* [21]. Here, a multicast group member with a control packet to send is forced to queue this packet for a random time interval before it can be transmitted. If a member receives a control packet which matches a queued packet, it cancels the transmission. For large group sizes, however, the random queueing time must be large, resulting in slow feedback times for the control information.

Yet another set of protocols without a control topology employ a central controlling station which coordinates ordering and reliability [2, 6, 8]. Due to the high load at the controlling station, the scalability of such protocols is strictly limited.

2.2 Ring Topology

Ring control topologies have been implemented to provide a reliable multicast service with total ordering of messages [3, 22]. In these protocols, the multicast group is structured as a logical ring, and a token is passed around the ring. Control messages are unicast between the current holder of the token and other nodes. The scalability of ring topologies is only moderate since control messages are always directed to the token holder, thus creating a bottleneck at that node. Also, the time to pass the token around the ring increases for large group sizes, resulting in decreased overall throughput.

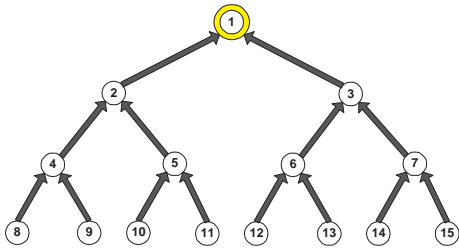
2.3 Tree Topology

Tree topologies assume that control information is transmitted along the edges of a *rooted spanning tree*. There is a spanning tree for each multicast member. We use T_k to denote the spanning tree with node k as root. Node l transmits a control message to the root node k by passing the message to its immediate ancestor in T_k , the tree rooted at k . Tree topologies achieve scalability by exploiting the hierarchical structure of a tree. A drawback of tree-based topologies is the overhead in constructing and maintaining the tree. Note that the tree must be dynamically modified both in response to host failures and to members joining and leaving the tree.

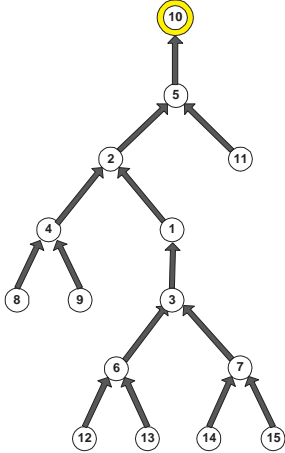
Several tree-based control topologies have been proposed for transmission of control information [9, 13, 15, 19, 23], mostly for multicast groups with only a single sender. We discuss the shared K -ary tree topology proposed in [13] which explicitly targets symmetric multicast groups. The shared tree topology is derived from a single balanced K -ary tree with root r . If

some other node $k \neq r$ becomes the root, then the tree is *re-hung* with node k as new root [13]. Re-hanging trees with a new root is illustrated in Figure 2. In Figure 2(a) we show a binary (2-ary) tree with node 1 as root. Figure 2(b) depicts the same tree, ‘re-hung’ for node 10 as root node.

Re-hanging a tree does not increase the number of children of each node. However, after re-hanging, the tree may no longer be balanced. Note that the longest path to the root in the re-hung tree in Figure 2(b) is twice as long as in the original tree in Figure 2(a).



(a) Original Tree Rooted at Node 1.



(b) Re-hung Tree Rooted at Node 10.

Figure 2: Shared Binary Tree.

3 The Hypercube Control Topology

In this section we propose the hypercube as a new control topology for multicast communications. We propose organizing the members of a multicast group as the nodes of a logical n -dimensional hypercube. We present a method for embedding spanning trees into the hypercube and use these spanning trees for disseminating control information.

3.1 Hypercube and Tree Embeddings

An n -dimensional hypercube is a graph with $N = 2^n$ nodes where each node is labeled by a bit string $k_n \dots k_1$ ($k_i \in \{0, 1\}$). Nodes in the hypercube are

connected by an edge if their bit strings differ in exactly one position. In Figure 3 we depict hypercubes for dimensions $n = 1$ to $n = 4$.

Hypercubes have been studied extensively by the parallel computing community; they are deemed attractive as a multiprocessor architecture because of their symmetry, the short distances between nodes, and the number alternative routes. The literature on hypercubes is rich, and we refer to [12, 18] as excellent sources on the topic.

In the following we show that the properties of the hypercube topology can be exploited to address the problem of disseminating control information in large multicast groups. We propose to organize the members of a multicast group as the nodes of a hypercube. Then we embed spanning trees into the hypercube and disseminate control information along the edges of the spanning trees.

Past research on parallel algorithms has produced numerous algorithms for embedding trees in hypercubes (see [12] for an overview). The goal of these algorithms is to assign a parallel computation, represented as a tree, into a multicomputer with an hypercube interconnection network. These algorithms make a number of assumptions that are not applicable in the context of a multicast group. First, most algorithms assume a static hypercube. Second, with few exceptions these algorithms assume a complete hypercube, i.e., $N = 2^n$. Both assumptions are not realistic for multicast groups with a dynamically changing membership.

Our goal is to exploit the strong symmetry of the hypercube in the context of multicast communications. To achieve this we must devise methods that address the problems of actual multicast applications. First, we have to consider *incomplete* hypercubes with $N < 2^n$ nodes. When embedding spanning trees in an incomplete hypercube with nodes V , we want to make sure that all spanning trees only contain nodes in V . Such trees are said to be *completely contained* in the incomplete hypercube. Second, we have to consider that the multicast group membership changes dynamically. Since adding and removing nodes may degenerate the compact structure of a hypercube, we need to have mechanisms in place that keep the dimension of the hypercube as small as possible; we refer to this property as *compactness*.

3.2 Gray Ordering of Hypercube Nodes

The key to ensure complete containment of all spanning trees and maintain compactness of the incomplete hypercube is the selection of a particular ordering of the nodes.

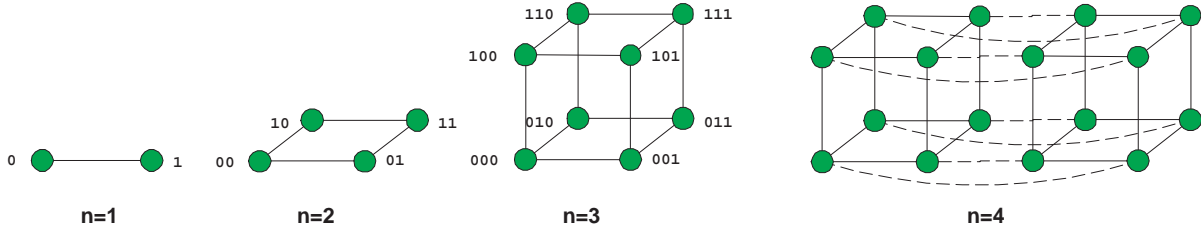


Figure 3: n -dimensional Hypercubes.

The standard ordering of hypercube nodes interprets the label of a node as a binary number. Specifically, the number $a = \sum_{i=1}^n a_i \cdot 2^{i-1}$ is associated with the node labeled $Bin(a) := a_n \dots a_1$ ($a_i \in \{0, 1\}$). With the ordering imposed by the numbers, compactness can be achieved by ensuring that in a multicast group with N members the positions $Bin(0), Bin(1), \dots, Bin(N-1)$ are always occupied. However, using this ordering it is not clear how to construct spanning trees that satisfy the complete containment condition.

As a solution we propose to use a different ordering of the nodes, which is based on interpreting node labels using a *Gray code*. A Gray code, denoted by ' $G(\cdot)$ ', is defined via the following properties [18]:

- The values are unique. That is, if $G(i) = G(j) \Rightarrow i = j$.
- $G(i)$ and $G(i+1)$ differ in only one bit, for $0 \leq i < 2^{d-1} - 1$.
- $G(2^{d-1} - 1)$ and $G(0)$ differ in only one bit.

In other words, a Gray code corresponds to a Hamiltonian walk on the hypercube [12]. Let i be a number and $Bin(i)$ its binary representation, it is easy to verify that following generates a Gray code:

$$G(i) := Bin(i) \otimes Bin(i/2)$$

where ' \otimes ' is the XOR operator and ' $x/2$ ' is an integer division by 2. We use $G^{-1}(\cdot)$ to denote the inverse of $G(\cdot)$, that is, $G^{-1}(G(i)) = i$.

By interpreting the node labels in the hypercube as Gray codes, the relationship between i and $G(i)$ defines an ordering of the nodes. Clearly, with this ordering, compactness can be enforced by ensuring that the members of a hypercube with K nodes occupy positions $G(0), G(1), \dots, G(K-1)$.

Example: Consider the ordering of nodes in a 3-dimensional hypercube. Refer to Figure 3 for the labeling of nodes. In the following table, we show the position number i , the binary interpretation $Bin(i)$, and the Gray code $G(i)$:

i	0	1	2	3	4	5	6	7
$Bin(i)$	000	001	010	011	100	101	110	111
$G(i)$	000	001	011	010	110	111	101	100

Thus, using the standard ordering $Bin(i)$ a multicast group with $K = 5$ members would occupy the following positions in the hypercube: 000, 001, 010, 011, 100. In contrast, using a Gray code $G(i)$ occupies the following positions: 000, 001, 011, 010, 110.

3.3 Tree Embedding in Gray-ordered Hypercubes

We now present an algorithm to embed spanning trees into hypercubes that use Gray codes for ordering the nodes. The embedding of spanning trees in the hypercube is calculated locally: A node with label $G(x)$ directly obtains the address of its parent node in the tree with root $G(r)$.

Most importantly, for a Gray-ordered hypercube which preserves compactness as shown in the previous subsection, our algorithm always generates a *completely contained* spanning tree. The algorithm is presented in Figure 4. Given two node labels I and R , the algorithm computes the label of the parent node of node I in the spanning tree that is rooted at node R . If each node performs the procedure *Parent* for a root node R , we obtain a spanning tree with root R embedded into the hypercube.

In Figures 5 and 6 we show the embeddings of the spanning trees in a 3-dimensional hypercube for root nodes 1 and 5, respectively.

All trees that are constructed by procedure *Parent* have the following set of properties. The properties follow directly from the procedure *Parent* and are shown without proof.

Property 1: A node and its parent always have a Hamming distance of 1.

Property 2: The path length between a node and a root is given by their Hamming distance.

Property 3: In a hypercube with N nodes, all trees have a depth of $\lceil \log_2(N) \rceil$. If $N = 2^n$, the embedding results in a binomial tree.¹

¹A *binomial tree* of height 0 is a single node. For all $i > 0$, a binomial tree of height i is a tree formed by connecting the

<p>Input: Label of the i-th node in the Gray encoding: $G(i) := I = I_n \dots I_2 I_1$, and the label of the r-th node ($\neq i$) in the Gray encoding: $G(r) := R = R_n \dots R_2 R_1$.</p> <p>Output: Label of the parent node of node I in the embedded tree rooted at R.</p>
<p>Procedure Parent (I, R)</p> <p>If ($G^{-1}(I) < G^{-1}(R)$) // Flip the <i>least significant bit</i> // where I and R differ. Parent := $I_n I_{n-1} \dots I_{k+1} (1 - I_k) I_{k-1} \dots I_2 I_1$ with $k = \min_i (I_i \neq R_i)$.</p> <p>Else // ($G^{-1}(I) > G^{-1}(R)$) // Flip the <i>most significant bit</i> // where I and R differ. Parent := $I_n I_{n-1} \dots I_{k+1} (1 - I_k) I_{k-1} \dots I_2 I_1$ with $k = \max_i (I_i \neq R_i)$.</p> <p>Endif</p>

Figure 4: Tree Embedding Algorithm.

Property 4: If $Parent(I, R)$ is the p -th node in the Gray encoding, then the following holds: $p \leq \max\{i, r\}$.

Property 4 ensures that the algorithm in Figure 4 guarantees complete containment of all embedded trees. Next we analyze the properties of the proposed hypercube control topology and compare it against tree-based solutions.

4 Comparison of Scalability Properties

To gain insight into the scalability property of the hypercube control topology, we conduct a performance comparison with the K -ary shared tree discussed in Subsection 2.3. Our analysis should be seen in the light of our assumption that communication within a multicast group is *symmetric*, that is, on the average each member of the group generates the same amount of traffic.

The scope of our investigation is limited to quantitative aspects of disseminating control information. We do not consider the effects of protocol processing or routing issues. Furthermore, we consider generic transmission of control messages without assumptions on a particular control function (flow control, error control, etc.) as in [16].

4.1 Performance Measures

We define a set of performance measures which capture the load for control processing incurred at each member of a multicast group. In all configurations considered, control information is transmitted along

roots of two binomial trees of height $i - 1$ with an edge and designating one of these roots to be the root of the new tree (cited from [18]).

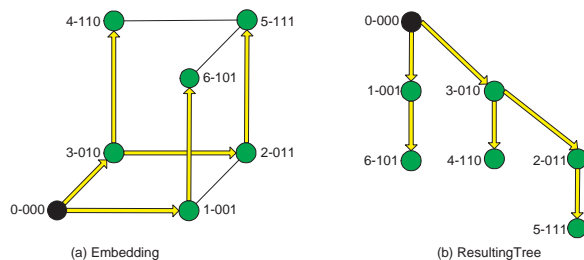


Figure 5: Embedding a Tree with Node 1 as Root.

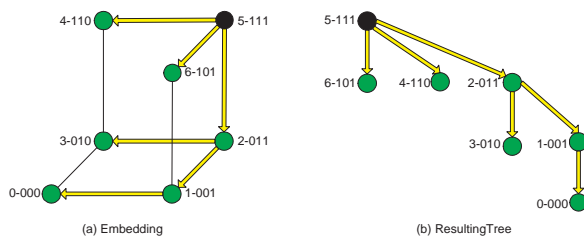


Figure 6: Embedding a Tree with Node 5 as Root.

the edges of a *rooted spanning tree*. Recall that we use T_l to denote the spanning tree with node $l \in V$ as root.

For most control functions, a good indicator for the load at a node in a control tree is the number of direct children. We define:

$$w_k(T_l) := \text{Number of children of node } k \in V \text{ in tree } T_l.$$

Since control functions may incur a load at a node that is proportional to the number of nodes in the subtree below it, we define:

$$v_k(T_l) := \text{Number of descendants of node } k \in V \text{ in tree } T_l \text{ (including node } k), \text{ where the descendants of node } k \text{ in tree } T_l \text{ are the nodes that have node } k \text{ on their path to the root node } l.$$

Finally, since the path lengths in a control tree indicate delays of passing control information in the tree, we define a measure that expresses this delay:

$$p_k(T_l) := \text{Length of the path from node } k \text{ to root node } l \text{ in } T_l.$$

Based on these notions we define more concise measures by taking the average over all trees T_l with $l \in V$, denoted as w_k , v_k , and p_k . These measures are defined as follows:

$$w_k := \frac{1}{N} \sum_{l=1}^N w_k(T_l) \quad \text{Average number of direct children of node } k \in V \text{ in a spanning tree.}$$

$$v_k := \frac{1}{N} \sum_{l=1}^N v_k(T_l) \quad \text{Average number of descendants of node } k \in V.$$

$$p_k := \frac{1}{N} \sum_{l=1}^N p_k(T_l) \quad \text{Average path length from node } k \text{ to the root.}$$

To further condense the amount of data, we take the averages and maxima of the above values and obtain:

$$\begin{aligned} w_{avg} &:= \frac{1}{N} \sum_{k=1}^N w_k & w_{max} &:= \max_k w_k \\ v_{avg} &:= \frac{1}{N} \sum_{k=1}^N v_k & v_{max} &:= \max_k v_k \\ p_{avg} &:= \frac{1}{N} \sum_{k=1}^N p_k & p_{max} &:= \max_k p_k \end{aligned}$$

We use w_{avg} and v_{avg} as indicators of the average processing load at a node. For w_{avg} , our rationale is that the load of processing control information directly correlates to the number of children of a node. Somewhat differently, v_{avg} correlates the load to the total number of descendants of a node. We interpret p_{avg} as a measure for the delays that occur in disseminating control information; the longer the path length to the root node, the higher the expected delay.

We use the values of w_{max} , v_{max} , and p_{max} to calculate measures for the degree of load balancing of a topology. Specifically, we use the ratios w_{max}/w_{avg} , v_{max}/v_{avg} , and p_{max}/p_{avg} to compare the worst-case node and average node for a control topology. Our expectation is that a control topology with good scalability properties must balance the load incurred at a node, which is reflected in the need for low maximum-to-average ratios.

In the following we present the results of the above measures for the K -ary tree and the hypercube. Since some derivations are quite long, they cannot be included in this paper. The reader is referred to [14] for the complete set of derivations.

4.2 Analysis of the Shared K -ary Tree

Recall that the control trees in the shared K -ary tree are obtained from a single K -ary by re-hanging this tree with different nodes as root [13]. In Figure 2 we showed an example of re-hanging a binary tree.

As K is increased, the maximum path length from a node to the (re-hung) root decreases. This, however, increases the load on the node that is the root in the original tree. In the extreme case, we have $N = K$ and obtain a *star topology* where re-hanging the tree always results in $N - 1$ nodes hanging off the original root of a star topology.

Let us assume for simplicity that all leaves of the tree are occupied, that is, $N = \frac{K^{d+1}-1}{K-1}$ for some $d \geq 0$. Then we obtain [14]:²

$$\begin{aligned} w_{avg} &= 1 \\ w_{max} &= K \\ v_{avg} &= 2d + \frac{k-5}{k-1} \end{aligned}$$

²The expressions shown here have an error term of order $O(1/N)$. We refer to [14] for the exact expressions.

$$\begin{aligned} v_{max} &= \begin{cases} \frac{5}{8}N + 1/4 & \text{if } k = 2 \\ \frac{k-1}{k}N + 2/k & \text{if } k > 2 \end{cases} \\ p_{avg} &= 2d - \frac{4}{k-1} \\ p_{max} &= 2d - \frac{3}{k-1} \end{aligned}$$

4.3 Analysis of the n-dimensional Hypercube

Calculating the performance measures for the hypercube, where trees are embedded as described in Section 3, requires considerable effort [14]. For a complete hypercube, that is, $N = 2^n$, we obtain [14]:

$$\begin{aligned} w_{avg} &= 1 \\ w_{max} &= 2 \\ v_{avg} &= 1/2 \log_2 N + 1 \\ v_{max} &= 1/8(\log_2 N)^2 + 3/8 \log_2 N + 1 \\ p_{avg} &= 1/2 \log_2 N \\ p_{max} &= 1/2 \log_2 N \end{aligned}$$

4.4 Discussion

We now use the measures from the previous subsections to demonstrate how the control topologies scale when the number of nodes grows large.

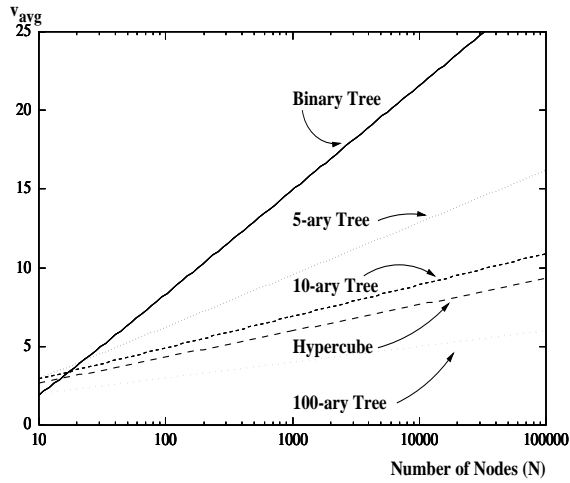
Let us first examine the number of children of a node, w_{avg} and w_{max} . Since, for all spanning trees, the average number of children is $w_{avg} \leq 1$, i.e., on the average a node in a control tree has only one child, we only compare the ratios w_{max}/w_{avg} :

$$\begin{aligned} \text{K-ary Tree:} & \quad w_{max}/w_{avg} = K \\ \text{Hypercube:} & \quad w_{max}/w_{avg} = 2 \end{aligned}$$

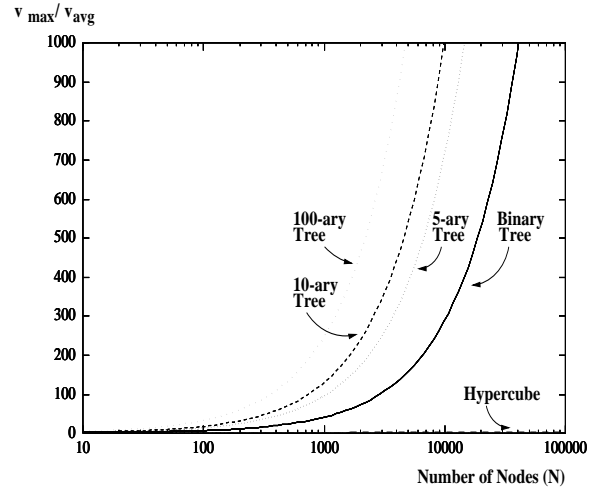
Thus, there is a node in a K -ary tree that has K times as many children as the average node resulting in a highly unbalanced load. The hypercube, in contrast, is better load-balanced. Here, the difference between the worst-case and the average case is only a factor of 2. (In the K -ary tree, the maximum is attained for the root in the original K -ary tree. The hypercube attains the maximum at the node with label $00\dots 0$.)

In Figure 7(a) we present a graph where we plot the values for v_{avg} by varying the number of nodes N . We present results for the shared K -ary tree (with $K = 2, 5, 10, 100$) and the hypercube. In the figure we see that, in a K -ary shared tree, v_{avg} decreases for increasing values of K . Note that, over the entire range of values, v_{avg} for the hypercube is smaller than the v_{avg} values for the 10-ary tree.

The comparison of the ratios v_{max}/v_{avg} , depicted in Figures 7(b), reveals a problem with load balancing

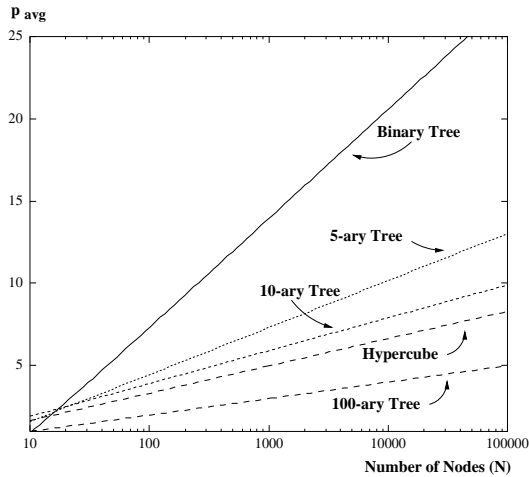


(a) Results for v_{avg} .

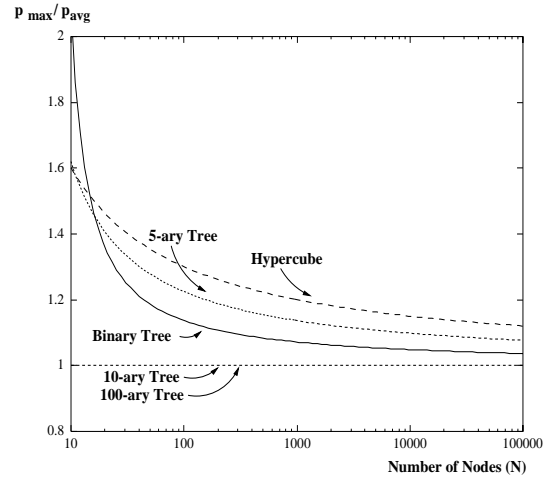


(b) Results for v_{max}/v_{avg} .

Figure 7: Comparison of Average Number of Descendants.



(a) Results for p_{avg} .



(b) Results for p_{max}/p_{avg} .

Figure 8: Comparison of the Average Path Lengths.

for the shared tree topologies. Since v_{max} increases linearly in N , all shared trees have a bottleneck at the node where the maximum is attained. For large values of K , scalability problems arise even for small group sizes. Even for the binary tree, the maximum-to-average ratio exceeds 100 when the number of nodes has only a few thousand nodes. In a direct comparison with the K -ary tree, the value of v_{max}/v_{avg} for the hypercube appears almost insignificant.

In Figures 8(a) and 8(b) we present the results for the path lengths. It is interesting to note that the average path to the root is shorter in the hypercube than in a 10-ary tree. Figure 8(b) shows that load balancing is not an issue when considering the path lengths. As the size of the multicast group is increased, the ratio p_{max}/p_{avg} quickly approaches 1 in all topologies.

In summary, the hypercube appears very suitable to support large multicast groups. A comparison with shared K -ary trees has shown that the hypercube has all the advantages, and none of the disadvantages of the shared K -ary tree. Particularly, the hypercube provides an excellent balance of the average and worst-case load at the nodes. The load-balancing indicators w_{max}/w_{avg} and v_{max}/v_{avg} clearly demonstrates that the shared tree topology has problems when scaled to very large group sizes. In contrast, the hypercube topology does not have these scalability problems.

5 Conclusions

We have presented a new approach for disseminating control information between the members of a multicast group. In our approach, we organize the members of the multicast group in a logical hyper-

cube and assign each multicast group member a number that is derived from a Gray code. The Gray encoding enables each node to locally calculate the next hop for transmitting control information. We analyzed the scalability properties of our approach in symmetric multicast groups, i.e., where each group member is a sender. In a comparison with an K -ary shared tree control topology we showed that the hypercube is superior in balancing the load of control information among all nodes. As future work, we will implement the proposed approach. We refer to [14] for a set of soft-state protocol mechanisms that maintain the hypercube topology in a packet-switching networks.

6 Acknowledgments

We gratefully acknowledge the help of Prof. Almut Burchard from the Department of Mathematics at Princeton University with devising the tree embedding algorithm.

References

- [1] J. Bolot. End-to-End Packet Delay and Loss Behavior in the Internet. *Proc. ACM Sigcomm '93*, 23(4):289–298, September 1993.
- [2] C. Bormann, J. Ott, H. Gehrcke, T. Kersch, and N. Seifert. MTP-2: Towards Achieving the S.E.R.O. Properties for Multicast Transport. In *Proc. ICCN '94, San Francisco*, 1994.
- [3] J. M. Chang and N. F. Maxemchuk. Reliable Broadcast Protocols. *ACM Transactions on Computing Systems*, 2(3):251–273, August 1984.
- [4] J. Crowcroft and K. Paliwoda. A Multicast Transport Protocol. In *Proc. ACM Sigcomm '88*, pages 247–256, August 1988.
- [5] S. E. Deering and D. R. Cheriton. Host Groups: A Multicast Extension to the Internet Protocol. Technical Report RFC 966, Internet Engineering Task Force, December 1985.
- [6] M. F. Kaashoek et. al. An Efficient Reliable Broadcast Protocol. *Operating Systems Review*, 23(4):5–20, October 1989.
- [7] S. Floyd, V. Jacobson, McCanne S, C.-G. Liu, and L. Zhang. A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing. In *Proc. ACM Sigcomm '95*, pages 342–356, August 1995.
- [8] A. Frier and K. Marzullo. MTP: An Atomic Multicast Transport Protocol. Technical report, Cornell University, 1990.
- [9] H. Garcia-Molina and A. Spauster. Ordered and Reliable Multicast Communication. *ACM Transactions on Computer Systems*, 9(3):242–271, August 1991.
- [10] H. W. Holbrook, S. K. Singhal, and D. E. Cheriton. Log-based Receiver-Reliable Multicast for Distributed Interactive Simulation. In *Proc. ACM Sigcomm '95*, August 1995.
- [11] M. G. W. Jones, S. A. Sorensen, and S. Wilbur. Protocol Design for Large Group Multicasting: The Message Distribution Protocol. *Computer Communications*, 14(5):287–297, 1991.
- [12] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufman Publishers, San Mateo, 1992.
- [13] B. N. Levine, D. B. Lavo, and J.J. Garcia-Luna-Aceves. The Case for Reliable Concurrent Multicasting Using Shared Ack Trees. In *Proc. ACM Multimedia '96*, November 1996.
- [14] J. Liebeherr and B. S. Sethi. Towards Super-Scalable Multicast. Technical report, Polytechnic University, 1998. <http://aida.poly.edu/~jorg/stuff>.
- [15] A. Paul, K. K. Sabnani, J. C.-H. Lin, and S. Bhattacharyya. Reliable Multicast Transport Protocol (RMTP). *IEEE Journal on Selected Areas in Communications*, 15(3):407 – 421, April 1997.
- [16] S. Pingali, D. Towsley, and J. F. Kurose. A Comparison of Sender-Initiated and Receiver-Initiated Reliable Multicast Protocols. In *Proc. ACM Sigmetrics '94*, May 1994.
- [17] M. Pullen, M. Myjak, and C. Bouwens. Limitations of Internet Protocol Suite for Distributed Simulation in the Large Multicast Environment. IETF Internet-Draft, March 1997.
- [18] M. J. Quinn. *Parallel Computing: Theory and Practice*. McGraw-Hill, New York, 2nd edition, 1994.
- [19] S. Ramakrishnan and B. N. Jain. A Negative Acknowledgment With Periodic Polling Protocol for Multicast over LANs. In *Proc. IEEE Infocom '87*, pages 502–511, March/April 1987.
- [20] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. Technical Report RFC 1889, Internet Engineering Task Force, January 1996.
- [21] W. T. Strayer, B. D. Dempsey, and A. C. Weaver. *XTP: The Xpress Transfer Protocol*. Addison-Wesley Publishing, 1992.
- [22] B. Whetten, S. Kaplan, and T. Montgomery. A High Performance Totally Ordered Multicast Protocol. In *Proc. Infocom '95*, 1995.
- [23] R. Yavatkar, J. Griffioen, and M. Sudan. A Reliable Dissemination Protocol for Interactive Collaborative Applications. In *Proceedings of ACM Multimedia 95*, November 1995.