A Scalable Control Topology for Multicast Communications [*]

Jörg Liebeherr[†] *Bhupinder S. Sethi*[‡]

[†] Polytechnic University
Department of Electrical Engineering
6 MetroTech Center
Brooklyn, NY 11201
Phone: (718) 260-3493, Fax: (718) 260-3074
email: `jorg@catt.poly.edu`

[‡] Department of Computer Science
University of Virginia
Charlottesville, VA 22903
email: `bss4k@cs.virginia.edu`

Abstract

Multimedia collaborative 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. For this it is crucial that there exist mechanisms to efficiently exchange control information between the members of a group. 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. We use actual data traces of the group membership in an MBONE conference to gain insight into the transient changes of the load at each node. We present a set of protocol mechanisms that maintain the hypercube topology in a soft-state fashion without requiring any entity to have global state information.

Key Words: *IP Multicast, Multicast Communications, Implosion Problem, Hypercubes.*

[*]This work is supported in part by the National Science Foundation under Grant No. NCR-9624106. Corresponding author is **Jörg Liebeherr.**

1 Introduction

Recently emerging interactive multi-user applications 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 [7].

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. Services that include 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, the management of control information in a multicast group is a non-trivial problem, especially, if group sizes are large. Without careful management of control information, a multicast protocol can only support small group sizes.

Consider, 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* [6, 13].

In recent years, many techniques and protocol mechanisms have been proposed to cope with the amount of control information that is exchanged between members of a multicast group [2, 5, 9, 12, 13, 16, 17, 19, 20, 22, 28, 24, 25, 29], mostly in the context of providing a reliable multicast service. Early proposals were targeted at local area networks and exploited the broadcast capabilities in such networks [22, 24]. In packet-switching networks, we find two approaches to deal with the volume of control traffic. In one approach, control information is broadcast to all members of the multicast group; the volume of control traffic is limited based on the volume of data traffic or on the size of the multicast groups [3, 9, 23]. The drawback of this approach is that the each multicast member has to reduce its control traffic as the multicast group grows. In the second approach, the 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 [5, 28] and trees [12, 17, 19, 29].

This paper proposes a new approach for disseminating control information between the members of a multicast groups. 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

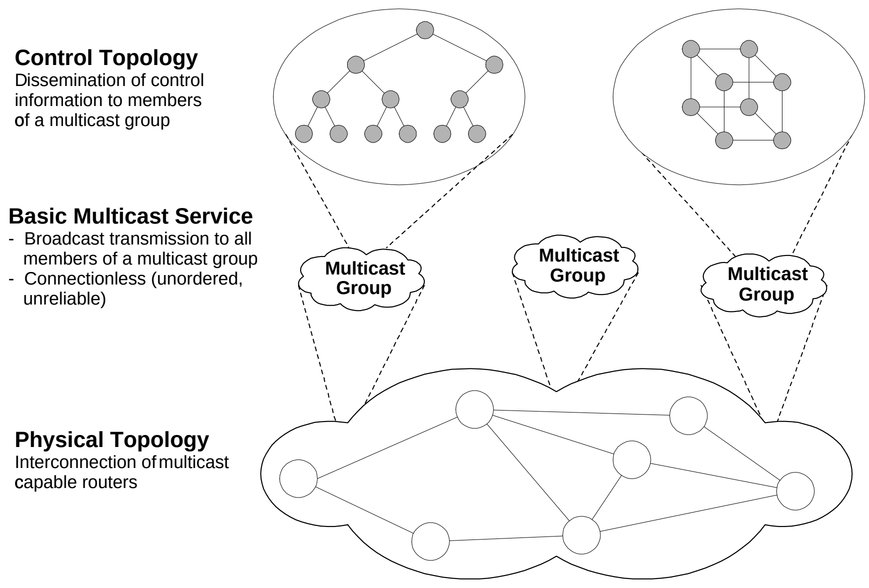

Control Topology
Dissemination of control information to members of a multicast group
Basic Multicast Service
- Broadcast transmission to all members of a multicast group
- Connectionless (unordered, unreliable)
Multicast Group
Multicast Group
Multicast Group
Physical Topology
Interconnection of multicast capable routers

Figure 1: Multicast Framework.

contributions of this study are as follows:

- We show how to construct a hypercube control topology with simple boolean operations.
- 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.
- 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 disseminating 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 the basic protocol mechanisms for a soft-state implementation of the proposed architecture. In Section 6 we present an empirical evaluation of various control topologies using actual traces from a large multicast session on the MBONE. In Section 7 we present our conclusions and discuss future work.

2 Control Topologies for Multicast Communications

In this section we review currently used control topologies for disseminating control information in multicast groups, focusing on the ability of these topologies to support large multicast groups. We also introduce some terminology that is used throughout this paper.

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. In contrast, an asymmetric groups is characterized by few members of the group generating most or all of the traffic.[1]

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 $V = \{1, 2, \ldots, N\}$. We assume that each node can directly communicate with any other member of the group.[2]

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 [23] limits the total amount of control traffic to 5% of the data traffic. In [3], 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* [9, 22] or *damping* [24]. 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 have to 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 [4, 8, 10]. Due to the high load at the controlling station, the scalability of such protocols is strictly limited.

[1]A multicast web server is an extreme example of an asymmetric group; here, the server is the only member of the group that generates traffic.

[2]This can be accomplished by a multicast message which is read only by the destination, or via unicast communication.

2.2 Ring Topology

Ring control topologies have been implemented to provide a reliable multicast service with total ordering of messages [5, 28] . 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

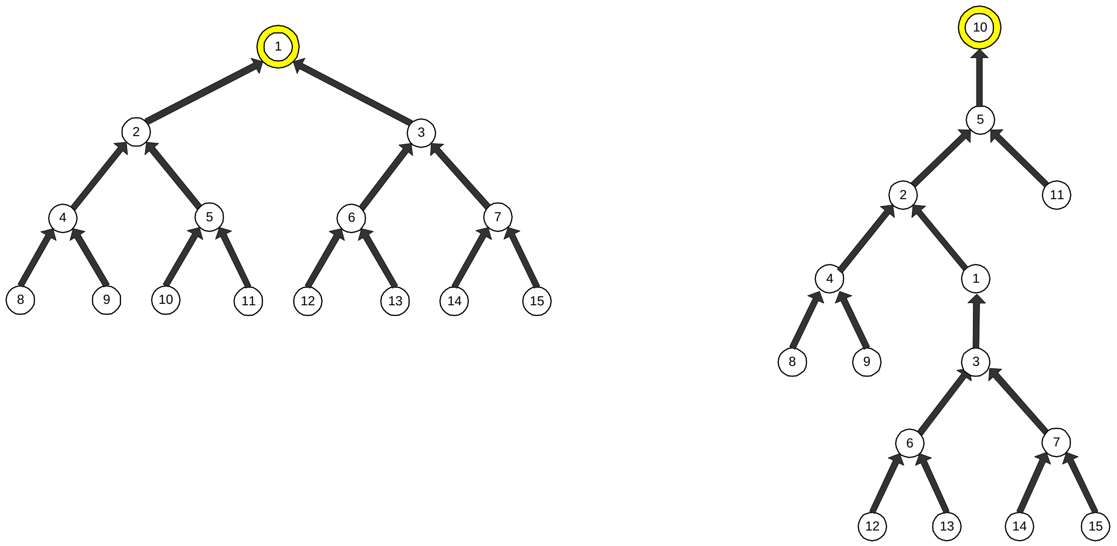
Tree topologies assume that control information is transmitted along the edges of a *rooted spanning tree*. There is a spanning tree for each multicast member. We use T_k to denote the spanning tree with node k as root. Node l transmits a control message to the root node k by passing the message to its 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 [11, 17, 19, 22, 29], mostly for multicast groups with only a single sender. We discuss the shared K-ary tree topology proposed in [17] 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 [17]. 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).

Among the currently considered topologies, tree-based topologies seem to be most suited to support large multicast groups. However, tree-based topologies are not without problems since re-hanging a shared tree destroys the balance for processing control information among nodes. In the next section we propose a new control topology, derived from a hypercube, that offers better load balancing, and as a result, has better scalability properties than tree-based



(a) Original Tree Rooted at Node 1. (b) Re-hung Tree Rooted at Node 10.

Figure 2: Shared Binary Tree.

solutions.

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.

We demonstrate that the hypercube topology is well-suited to support multicast communication in very large groups. In this section we present our method for embedding control trees in a hypercube. In Section 4 we compare the hypercube against the a control topology that is derived from a shared K-ary tree. In Section 5 we present a soft-state protocol for maintaining a hypercube topology in a dynamic multicast group.

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 [14, 15, 21] 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 [15] for an overview). The goal of these 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 [26, 27], 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* hypercube 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. We refer to such trees as *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.

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 $Bin(a) := a_n \ldots a_1$ $(a_i \in \{0, 1\})$. With the ordering imposed by the numbers, compactness can be achieved by ensuring that in a multicast group with N members the positions

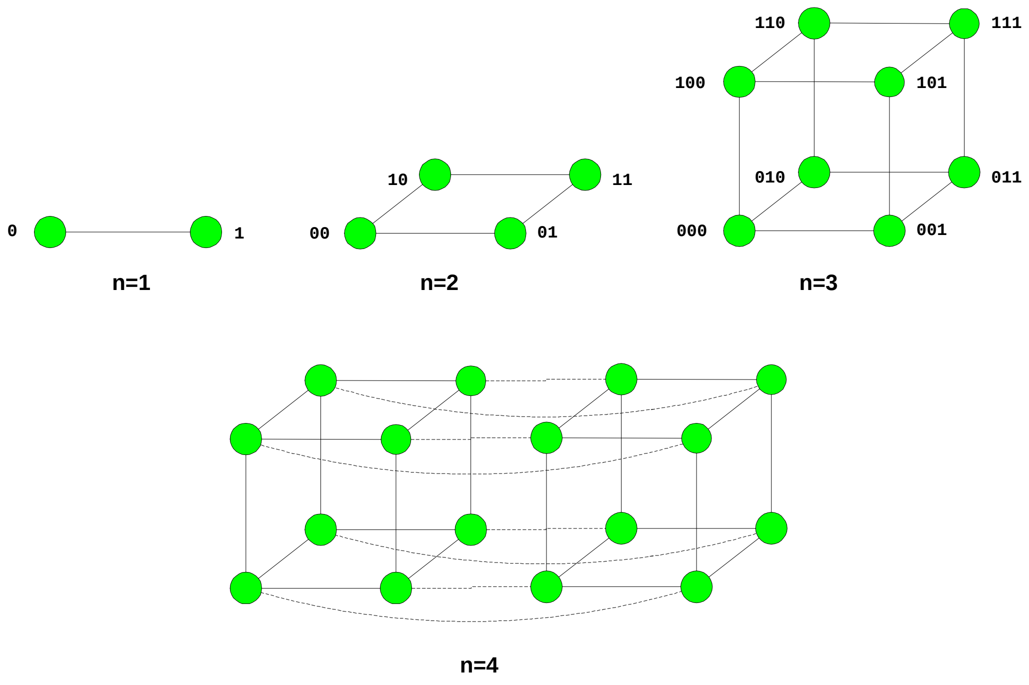


Figure 3: n-dimensional Hypercubes.

$Bin(0), Bin(1), \ldots, Bin(N-1)$ are always occupied. However, using this ordering there 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 an ordering obtained by interpreting node labels using a *Gray codes.* A Gray code, denoted by '$G(\cdot)$', is defined via the following properties [21]:

- 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 [15].

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 $G(0), G(1), \ldots, G(K-1)$.

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 $G(i)$:

Position Number 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 $G(r)$. The ability to calculate the embedded tree in a distributed fashion will be exploited in Section 5 where we present protocol mechanisms to maintain a hypercube topology.

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. Intentionally, we have depicted an incomplete hypercube with $N = 7 < 2^3$ nodes.

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.

```
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:
            $G(r) := R = R_n \dots R_2 R_1$.
Output: Label of the parent node of node $I$ in the embedded tree rooted at $R$.
------------------------------------------------------------------------------
 1. Procedure Parent $(I, R)$
 2.     If $(G^{-1}(I) < G^{-1}(R))$
 3.         // Flip the least significant bit where I and R differ.
 4.         Parent := $I_n I_{n-1} \dots I_{k+1}(1 - I_k)I_{k-1} \dots I_2 I_1$
 5.             with $k = \min_i (I_i \neq R_i)$.
 6.     Else // $(G^{-1}(I) > G^{-1}(R))$
 7.         // Flip the most significant bit where I and R differ.
 8.         Parent := $I_n I_{n-1} \dots I_{k+1}(1 - I_k)I_{k-1} \dots I_2 I_1$
 9.             with $k = \max_i (I_i \neq R_i)$.
10.     Endif
```

Figure 4: Tree Embedding Algorithm.

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. [3]

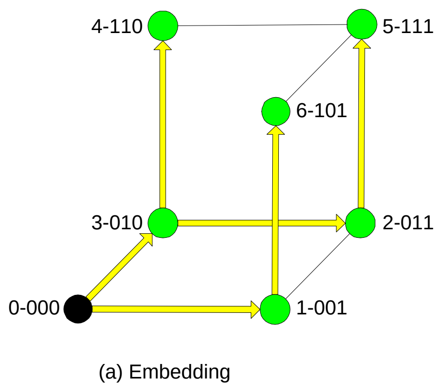
Property 4: If **Parent** (I, R) is the p-th node in the Gray encoding, then the following holds: $p \leq \max\{i, r\}$.

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

[3] A *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 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 [21]).



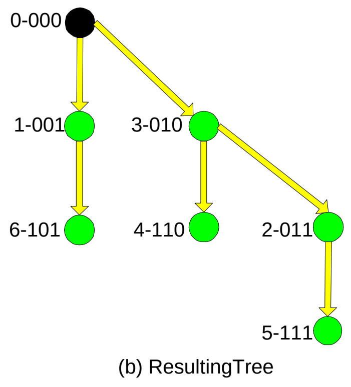
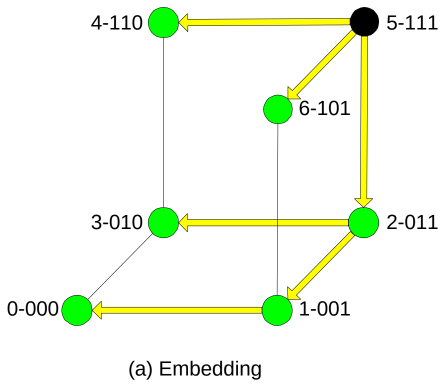


Figure 5: Embedding a Tree with Root $r = 1$.



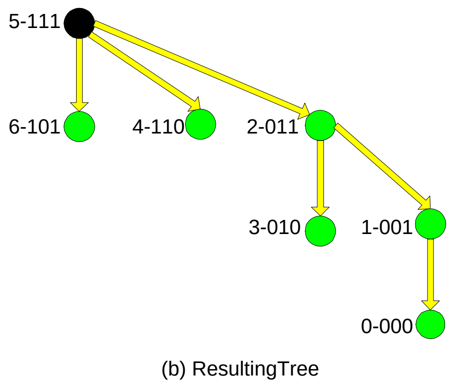


Figure 6: Embedding a Tree with Root $r = 5$.

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 [20].

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

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

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

$v_k(T_l) :=$ Number of descendants of node $k \in V$ in tree T_l (including node k), where the *descendants* of node k in tree T_l are the nodes that have node k 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:

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

Based on the 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:

$w_k := \frac{1}{N}\sum_{l=1}^{N} w_k(T_l)$	Average number of direct children of node $k \in V$ in a spanning tree.
$v_k := \frac{1}{N}\sum_{l=1}^{N} v_k(T_l)$	Average number of descendants of node $k \in V$.
$p_k := \frac{1}{N}\sum_{l=1}^{N} p_k(T_l)$	Average path length from node k to the root.

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

$$\overline{w} := \frac{1}{N}\sum_{k=1}^{N} w_k \qquad \overline{v} := \frac{1}{N}\sum_{k=1}^{N} v_k \qquad \overline{p} := \frac{1}{N}\sum_{k=1}^{N} p_k$$

$$w_{max} := \max_k w_k \qquad v_{max} := \max_k v_k \qquad p_{max} := \max_k p_k$$

We use $\overline{w}$ and $\overline{v}$ as indicators of the average processing load at a node. For $\overline{w}$, our rationale is that the load of processing control information directly correlates to the number of children of a node. Somewhat differently, $\overline{v}$ correlates the load to the total number of descendants of a node. We interpret $\overline{p}$ 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 $w_{max}/\overline{w}$, $v_{max}/\overline{v}$, and $p_{max}/\overline{p}$ 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 [18] 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 [17]. 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, i.e., there exists a $d \geq 0$ with $N = \frac{K^{d+1}-1}{K-1}$. Then we obtain [18]:[4]

$$\begin{aligned} \overline{w} &= 1 & \qquad w_{max} &= K \\ \overline{v} &= 2d - \frac{k+3}{k-1} + 2 & \qquad v_{max} &= \begin{cases} \frac{5}{8}N + \frac{1}{4} & \text{if } k = 2 \\ \frac{k-1}{k}N + \frac{2}{k} & \text{if } k > 2 \end{cases} \\ \overline{p} &= 2d - \frac{4}{k-1} & \qquad p_{max} &= 2d - \frac{3}{k-1} \end{aligned}$$

4.3 Analysis of the n-dimensional Hypercube

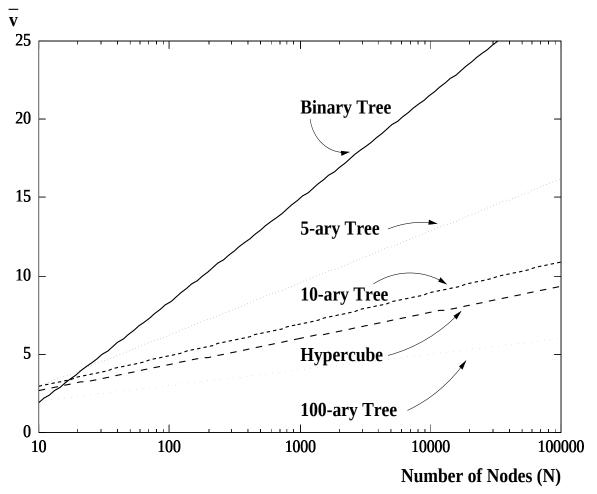
Calculating the performance measures for the hypercube, where trees are embedded as described in Section 3, requires considerable effort [18]. For a complete hypercube, that is, $N = 2^n$, we obtain [18]:

$$\begin{aligned} \overline{w} &= 1 & \qquad w_{max} &= 2 \\ \overline{v} &= \tfrac{1}{2}\log_2 N + 1 & \qquad v_{max} &= \tfrac{1}{8}(\log_2 N)^2 + \tfrac{3}{8}\log_2 N + 1 \\ \overline{p} &= \tfrac{1}{2}\log_2 N & \qquad p_{max} &= \tfrac{1}{2}\log_2 N \end{aligned}$$

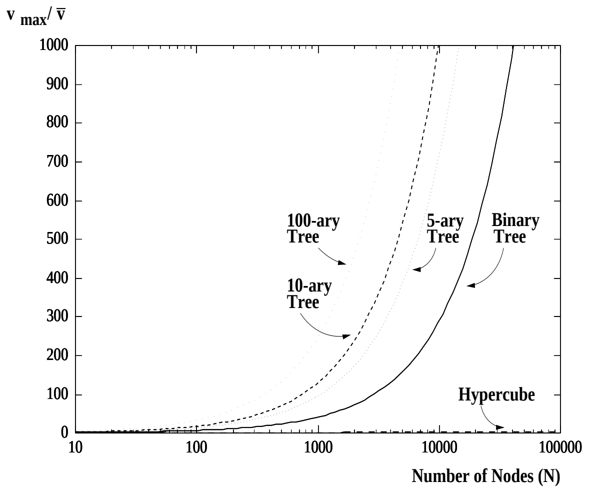
4.4 Discussion

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

[4]The expressions shown here have an error term of order $O(1/N)$. We refer to [18] for the exact expressions.



(a) Results for $\overline{v}$.



(b) Results for $v_{max}/\overline{v}$.

Figure 7: Comparison of Average Number of Descendants.

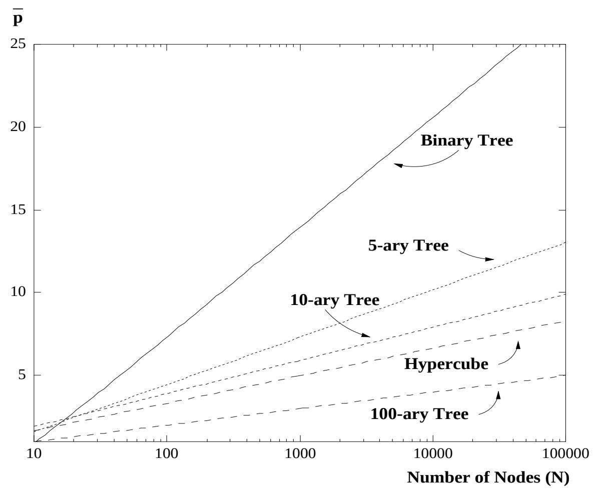
Let us first examine the number of children of a node, $\overline{w}$ and w_{max}. Since, for all spanning trees, the average number of children is $\overline{w} \leq 1$, i.e., on the average a node in a control tree has only one child, we only compare the ratios $w_{max}/\overline{w}$:

K-ary Tree:	$w_{max}/\overline{w} = K$
Hypercube:	$w_{max}/\overline{w} = 2$

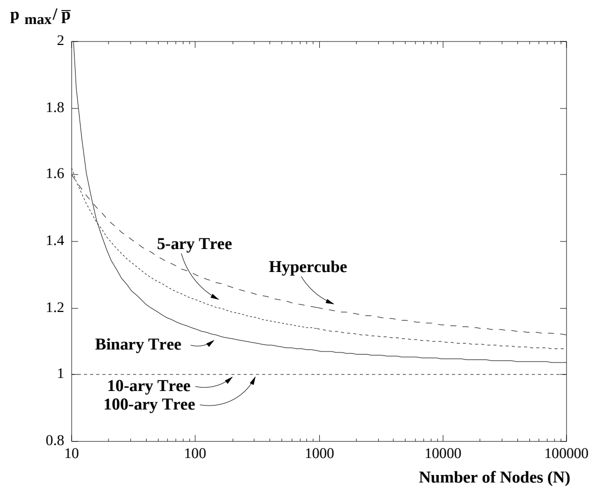
Thus, there is a node in a K-ary tree that has K times as many children as the average node resulting in a highly unbalanced load. The hypercube, in contrast, is better load-balanced. Here, the difference between the worst-case and the average case is only a factor of 2. (In the K-ary tree, the maximum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

In Figure 7(a) we present a graph where we plot the values for $\overline{v}$ 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, $\overline{v}$ decreases for increasing values of K. Note that, over the entire range of values, $\overline{v}$ for the hypercube is smaller than the $\overline{v}$ values for the 10-ary tree.

The comparison of the ratios $v_{max}/\overline{v}$, depicted in Figures 7(b), reveals a problem with load balancing for the shared tree topologies. Since v_{max} increases linearly in N, all shared trees have a bottleneck at the node were the maximum is attained. For large values of K, scalability problems arise even for small group sizes. Even for the binary tree, the maximum-



(a) Results for $\overline{p}$.



(b) Results for $p_{max}/\overline{p}$.

Figure 8: Comparison of the Average Path Lengths.

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 $v_{max}/\overline{v}$ 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 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 $p_{max}/\overline{p}$ quickly approaches 1 in all topologies.

In summary, the hypercube appears very suitable to support large multicast groups. A comparison with shared K-ary tree showed 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 $w_{max}/\overline{w}$ and $v_{max}/\overline{v}$ 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 Protocol Mechanisms for Maintaining a Hypercube Control Topology

We now present a set of protocol mechanisms for maintaining a hypercube control topology. We only discuss the most basic operations for entering a new node ('Join') and eliminating an existing node ('Leave') from the hypercube.

The control topologies considered in this study are built on top of the basic IP multicast service. Recall, that a message sent to an IP multicast address is broadcast to all users listening to this address. No guarantees are made on successful delivery or order of delivery.

The protocol mechanisms to be presented are all *soft-state*; that is, all state information is refreshed periodically without requiring a consistent state at all times.

5.1 Protocol Goals

The goal of the protocol is to organize a group of nodes in a logical hypercube. Every node is aware only of its neighbors in the hypercube. No entity in the system has a notion of the complete state information.[5]

We assume that each multicast group has a single multicast channel, referred to as *control channel.* Every member can send and receive on this channel. Obviously, scalability requirements demand that the traffic on this channel be kept minimal.[6]

Every node in the hypercube has a *physical address*, given by an IP address/port pair and a *logical address*, given by the node label discussed in Section 3. The logical addresses are totally ordered using the Gray code from Subsection 3.2. The hypercube is said to be in a *stable state* if it satisfies the following three criteria:

1. *Consistent:* No two nodes have the same logical address.
2. *Compact:* In a multicast group with N nodes, the dimension of the hypercube is given by $\lceil log_2 N \rceil$.
3. *Connected:* Each node knows the physical address of each of its neighbors in the hypercube.

[5]This is in contrast protocols, such as RMP [28], where every node has information about every member in the group, but also different from protocols, such as SRM [9], where no node has any information about the group.

[6]This requirement can be relaxed so that only a small subset of nodes is listening on the control channel at any given point of time. But, currently, we will assume that every member of the group is listening.

Faults and the mere fact that the protocol is soft-state may lead to a violation of the above conditions, in which case the hypercube becomes unstable. Our protocol returns an unstable hypercube to a stable state. Due to space limitations, we do not discuss recovery from network or node failures.

5.2 Neighborhoods and Leaders

The *neighbors* of a node are those nodes with logical addresses that differ from the logical address of the node in exactly one bit location. Note that in an m-dimensional hypercube, every node has a maximum number of m neighbors. Every node maintains a table with the physical addresses of all its neighbor; this table defines the node's *neighborhood.* A node is said to have a *complete neighborhood* if it knows the physical address of each neighbor that is part of the hypercube. In a stable state, a node can determine whether it has a complete neighborhood, as long as it knows the largest logical address (according to the ordering of the Gray code) in the group.

If, in an unstable hypercube, two or more nodes have the same logical address, we use the physical address to establish a total ordering. Given any node, its successor in the Gray's ordering is defined to be its *ancestor.* In a stable hypercube, every node except the one with the largest logical address has one ancestor. A node without any neighbor with a logical address greater than its own is defined to be a *leader.* A stable hypercube has exactly one leader.

5.3 Basic Message Formats

The maintenance operations of the hypercube are based upon a set of messages, called *basic messages.* The protocol has a total of 5 basic messages. A node transmits a message, either by unicasting to one or all of its neighbors, or by multicasting on the control channel. We do not assume transmission of basic messages to be reliable.

Much of the hypercube maintenance is based on the following two basic messages which are repeatedly sent at a given frequency:

- **Ping Message:** Every node periodically *ping*s its neighbors by sending them a short unicast message. A *ping* message contains the logical and physical address of the sender and the receiver of the message. If a node with an incomplete neighborhood receives a *ping* from a new neighbor, it updates its neighborhood table with the physical address of the new neighbor, and begins *ping*ing this new node. If a neighbor of a node does not send *ping*s for an extended period of time, the node assumes the neighbor to have failed, and removes the entry from its neighborhood.

- **Beacon Message:** A *beacon* is a message that is multicast on the control channel. The *beacon* message contains the current neighborhood of the sender. A node can transmit a *beacon* message only if it (1) considers itself to be a leader, or, (2) determines that it does not have a complete neighborhood. Every node uses the *beacon* messages sent by leader(s) to form an estimate of the largest logical address in the group. This information is sufficient to determine whether it has a complete neighborhood.

  Our protocol uses the *beacon* to accomplish the join procedure of new nodes. (The *beacon* messages are also used for stabilizing the hypercube after network or node failures.)

The remaining three basic messages are used for specific tasks:

- **Leave Message:** A node sends a *leave* message when it wants to release its logical address, either because it wants to leave the group or because it is moving to a new logical address. The node sends this message to all its neighbors, which - upon reception - remove the node from their neighborhood.

  REMARK: Since a leave message is not reliable, a neighbor may not receive a *leave* message. In this case, it will notice the absence of the node through missing responses to its *ping* messages.

- **Kill Message:** A *kill* messages is issued to resolve conflicts when two or more nodes happen to have the same logical address. If a node detects that another node has the same logical address as itself, it compares its physical address with the physical address of the other node. If the other node has a lower physical address, it sends a *kill* message, on receiving which, the first node will leave and rejoin the group. If however, its own physical address is lower, it behaves as if it had received *kill* message.

- **Bid Message:** The *bid* message is used by nodes in response to a *beacon* message and contains both the logical and physical address of the sender. A *bid* indicates that a node is willing to move to the neighborhood of the node which sent the *beacon*.

5.4 Node Behavior

The protocol imposes the following rules on each node.

- Every node listens to the control channel and keeps track of the largest logical address in the hypercube (according to the ordering of the Gray code).

- Every node sends a *ping* to its neighbors at a frequency F_1. If a node determines itself to be a leader, it will start to send multicast *beacon*s to the control channel at frequency F_2. A node with an incomplete neighborhood will also send *beacon*s, albeit at a lower frequency $F_3 (<< F_2)$. The frequencies should be set according to the rate of network faults and node failures.

- Each leader node strives to move to a logical address which is lower than its own in the Gray's ordering. Whenever a node can take a new position with a smaller logical address in the hypercube without creating a hole, it will do so. We will call this behavior the *address minimization principle*. More precisely, if a node receives a *ping* directed at its physical address, but to a logical address lower than its own, it will assume the lower logical address contained in the *ping*.

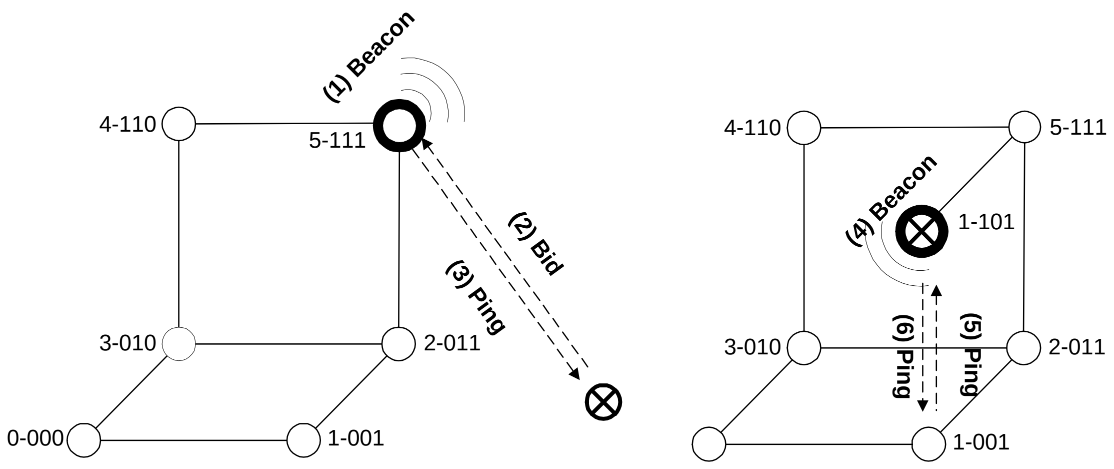
- If a node receives a message from a node with a logical address identical to its own, it resolves the conflict by removing itself from the hypercube, or by sending a *kill* message to the node, as explained previously.

5.5 The Join Procedure

Figure 9 illustrates the steps that are performed when a new node joins the hypercube. The join procedure is completed in six steps; Figure 9(a) shows the first three steps, and Figure 9(b) shows the last three steps. A node that wants to join starts listening to the multicast channel until it encounters a *beacon* packet from the leader node. The node responds with a *bid* packet to the leader. If the leader receives the *bid* packet, it installs the node into its neighborhood and starts *ping*ing the new node with its new logical address. The logical address of the new node is the successor of the old leader in the ordering of the Gray code; hence, the new node becomes the new leader and begins to *beacon* on the control channel (see second picture in Figure 9). As soon, as the neighbors of the new node see the *beacon*, they will update their neighborhood table and start *ping*ing the new node. The new node, will return the *ping*, thereby completing its neighborhood view.

5.6 The Leave procedure

The fact that a node has left a position in the hypercube is detected by its neighbors through receipt of a *leave* message, or because no *ping* message was received for an extended period of time. In either situation, the ancestor of the departing node assumes responsibility for restoring the compactness of the hypercube.



(a) Steps (1) to (3). (b) Steps (4) to (6)..

Figure 9: Join Procedure.

To restore compactness, the ancestor waits for a *beacon* from a leader on the control channel. At that point, it installs the leader in its neighborhood, and begins *ping*ing the leader. Because of the address minimization principle, the leader will try to take the empty position. Then, the (old) leader will find its other neighbors in the new neighborhood in a similar fashion as a newly joining node.

The above discussion contains only some of the protocol mechanisms necessary for maintaining the hypercube control topology. Due to space limitations, we have simplified the discussion and refer to [18] for a complete description of the protocol.

6 Empirical Evaluation of Dynamic Behavior

In this section, we present an empirical evaluation of the hypercube control topology. We measure the dynamic behavior of the hypercube control topology, and compare the results with the K-ary shared tree topology.

The basis for our empirical evaluation are data traces of an MBONE session obtained with the *Mlisten* tool [1]. The authors of [1] have made available to us detailed traces of the multicast group membership of MBONE sessions. The traces that are the basis for our evaluation are the video transmission sessions for the *NASA's STS-80C space shuttle mission* measured over a period of 27 days (657 hours) from November 8, 1996 to December 4, 1996.

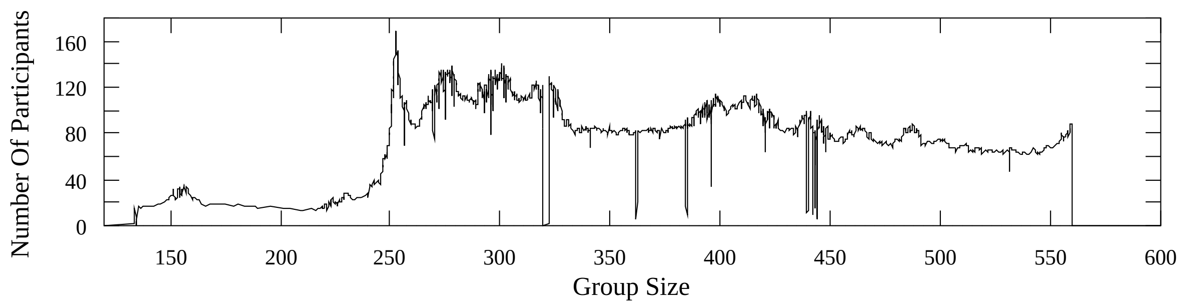


Figure 10: Group Size of the MBONE Session.

The varying size of the group membership over the measurement period from 120 hours to 600 hours is shown in Figure 10. Over the entire length of the experiment, the MBONE sessions had 1,874 different hosts participating with a maximum of 169 simultaneous users (at $t = 252$ hours into the experiment). Note in Figure 10 the occasional drops in the group size, e.g., at $t = 319$ hours. Even though these drops are likely due to artifacts of the measurement experiments, we assume that Figure 10 reflects the actual group size.

We use this trace to track the dynamic behavior of the performance measures from Subsection 4.1 which indicate the average load and the degree of load balancing. [7]

As before, we perform a comparison between the hypercube topology and the shared K-ary tree. In all topologies, we assume that, whenever the group membership has changed, the resulting topology is as compact as possible. In tree topologies, we keep the tree depth minimal, and in the hypercube topology, we attempt to minimize the hypercube dimension. For a hypercube this means we assume that compactness is always maintained. For the tree, our scheme ensures that all nodes at level 0 (root) to d are filled before filling any node at level $d + 1$. This assumption ensures that we always present the most optimistic results for the shared K-ary tree. Actual algorithms for restructuring shared K-ary trees, e.g., [17], may yield worse results.

Next we discuss the outcome of the experiment:

- **Number of children of a node**: In Figure 11 we depict the ratio $w_{max}/\overline{w}$ which reflects the skewness of load balancing. [8] Note that the ratio is lowest for the hypercube. For K-ary trees, the ratio increases as the maximum number of children of a node is increased.
- **Number of descendants of a node:** In Figure 12 we depict the values for the ratio

[7]We emphasize again that, in this paper, we do not take into consideration any routing aspects.

[8]We mentioned before, that $\overline{w}$ approaches 1 as the number of nodes increases, and is, therefore, not very interesting.

$v_{max}/\overline{v}$. (The values of $\overline{v}$ are ≤ 4 for the hypercube, ≤ 10 for the binary tree, ≤ 6 for the 5-ary tree, and ≤ 3 for the Star network). The hypercube clearly emerges as the topology with the best load balancing properties.

• **Path Length in Control Topology:** Since for all topologies considered, the maximum value p_{max} is within a constant from the average $\overline{p}$ (see Section 4), we only show the values for $\overline{p}$. In Figure 13 we see that the average path length in the hypercube topology is less than 4 hops, less than that for the 5-ary tree.

7 Conclusions

We have presented a new approach for disseminating control information between the members of a multicast groups. In our approach, we organize the members of a multicast group in a logical hypercube 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. We outlined a set of protocol mechanism for maintaining the hypercube topology in a packet-switching networks. The protocol was soft-state and did not require global knowledge of the hypercube topology at any network entity.

Our future work on the hypercube control topology will be concerned with both conceptual and practical issues. On a conceptual level, we will attempt to account for the geographical location of a host when inserting a node into a hypercube. As an experimental project, we are currently working on a Java implementation of the protocol mechanisms. The implementation will enable us to measure the overhead incurred by the protocol and the rate at which the protocol can recover from network failures.

8 Acknowledgments

We gratefully acknowledge the help of Prof. Almut Burchard from the Department of Mathematics at Princeton University with devising the tree embedding algorithm. We thank Prof. Mostafa Ammar and Dr. Kevin Almeroth from the College of Computing at Georgia Tech for making available to us the traces of the MBONE measurements.

References

[1] K. Almeroth and M. Ammar. Multicast Group Behavior in the Internet's Multicast Backbone (MBone). *IEEE Communications Magazine*, 35(6), June 1997.

[2] S. Armstrong, A. Freier, and K. Marzullo. Multicast Transport Protocol. Technical Report RFC 1301, Internet Engineering Task Force, February 1992.

[3] J. Bolot. End-to-End Packet Delay and Loss Behavior in the Internet. *Proc. ACM Sigcomm '93*, 23(4):289–298, September 1993.

[4] C. Bormann, J. Ott, H. Gehrcke, T. Kerschat, and N. Seifert. MTP-2: Towards Achieving the S.E.R.O. Properties for Multicast Transport. In *Proc. ICCN '94, San Francisco*, 1994.

[5] J. M. Chang and N. F. Maxemchuck. Reliable Broadcast Protocols. *ACM Transactions on Computing Systems*, 2(3):251–273, August 1984.

[6] J. Crowcroft and K. Paliwoda. A Multicast Transport Protocol. In *Proc. ACM Sigcomm '88*, pages 247–256, August 1988.

[7] 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.

[8] M. F. Kaashoek et. al. An Efficient Reliable Broadcast Protocol. *Operating Systems Review*, 23(4):5–20, October 1989.

[9] 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.

[10] A. Frier and K. Marzullo. MTP: An Atomic Multicast Transport Protocol. Technical report, Cornell University, 1990.

[11] H. Garcia-Molina and A. Spauster. Ordered and Reliable Multicast Communication. *ACM Transactions on Computer Systems*, 9(3):242–271, August 1991.

[12] 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.

[13] 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.

[14] S. Lakshmivarahan and S. K. Dhall. *Analysis and Design of Parallel Algorithms: Arithmetic and Matrix Problems.* McGraw-Hill, New York, 1990.

[15] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufman Publishers, San Mateo, 1992.

[16] B. N. Levine and J.J. Garcia-Luna-Aceves. A Comparison of Known Classes of Reliable Multicast Protocols. In *Proc. IEEE International Conference on network Protocols (ICNP '96)*, 1996.

[17] 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.

[18] J. Liebeherr and B. S. Sethi. Towards Super-Scalable Multicast. Technical report, University of Virginia, August 1997. Technical report is in preparation. Sections that are referenced in the text are available at: `http://www.cs.virginia.edu/~bss4k/research`.

[19] A. Paul, K. K. Sabnani, J. C.-H. Lin, and S. Bhattacharyya. Reliable Multicast Transport Protocol (RMTP). *IEEE Journal on Selected Areas in Communications. Special Issue for Multipoint Communications*, 15(3):407 – 421, April 1997.

[20] 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.

[21] M. J. Quinn. *Parallel Computing: Theory and Practice.* McGraw-Hill, New York, 2nd edition, 1994.

[22] 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.

[23] 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.

[24] W. T. Strayer, B. D. Dempsey, and A. C. Weaver. *XTP: The Xpress Transfer Protocol.* Addison-Wesley Publishing, 1992.

[25] R. Talpade and M. H. Ammar. Single Connection Emulation: An Architecture for Providing a Reliable Multicast Transport Service. In *Proc. 15th IEEE Intl Conf on Distributed Computing Systems (ICDS-15)*, June 1995.

[26] N.-F. Tzeng and H.-L. Chen. Structural and Tree Embedding Aspects of Incomplete Hypercubes. *IEEE Transactions on Computers*, 43(12):1434–1438, December 1994.

[27] N.-F. Tzeng and H. Kumar. Traffic Analysis and Simulation Performance of Incomplete Hypercubes. *IEEE Transactions on Parllel and Distributed Systems*, 7(7):740–754, July 1996.

[28] B. Whetten, S. Kaplan, and T. Montgomery. A High Performance Totally Ordered Multicast Protocol. In *Proc. Infocom '95*, 1995.

[29] R. Yavatkar, J. Griffioen, and M. Sudan. A Reliable Dissemination Protocol for Interactive Collaborative Applications. In *Proceedings of ACM Multimedia 95*, November 1995.

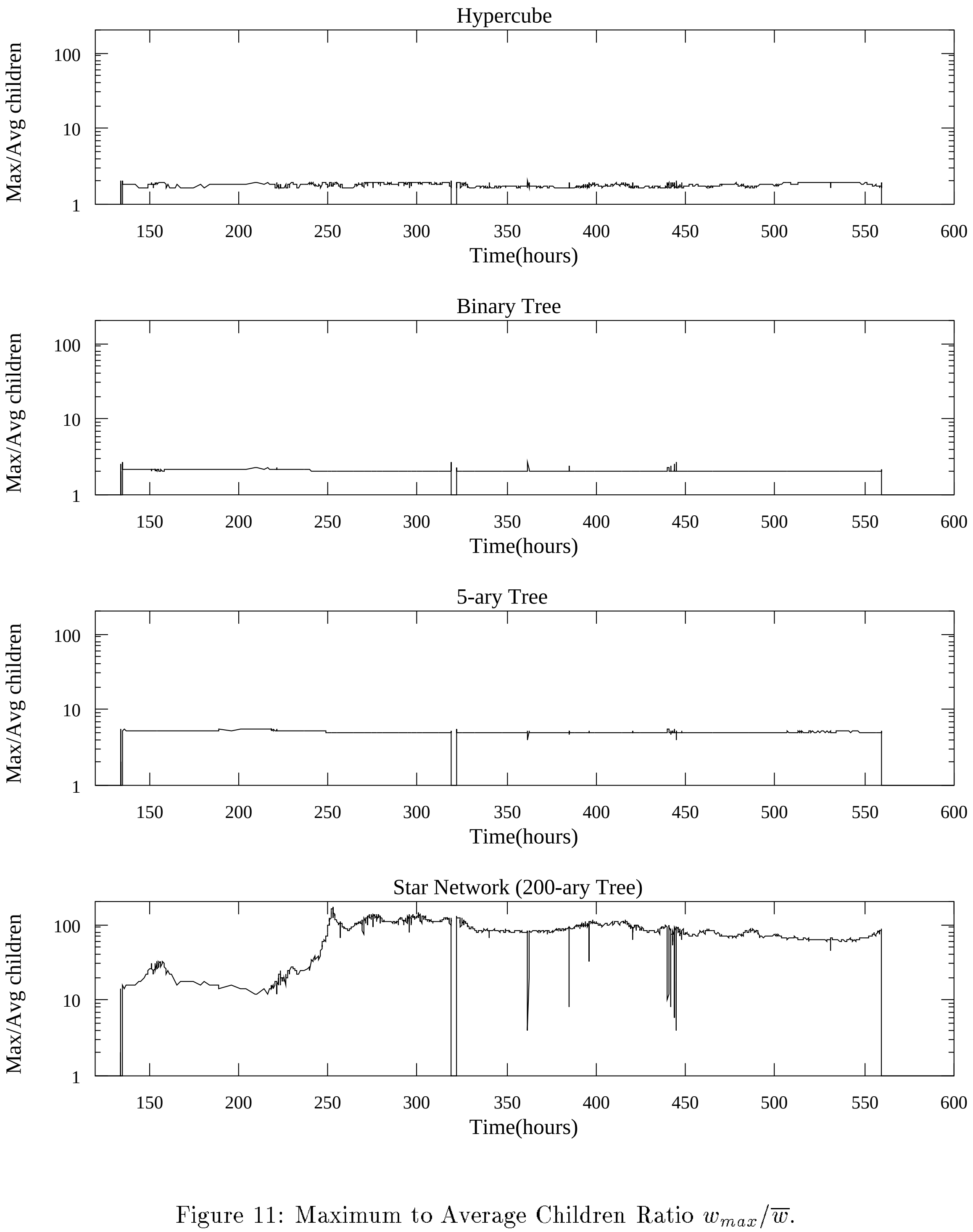


Figure 11: Maximum to Average Children Ratio $w_{max}/\overline{w}$.

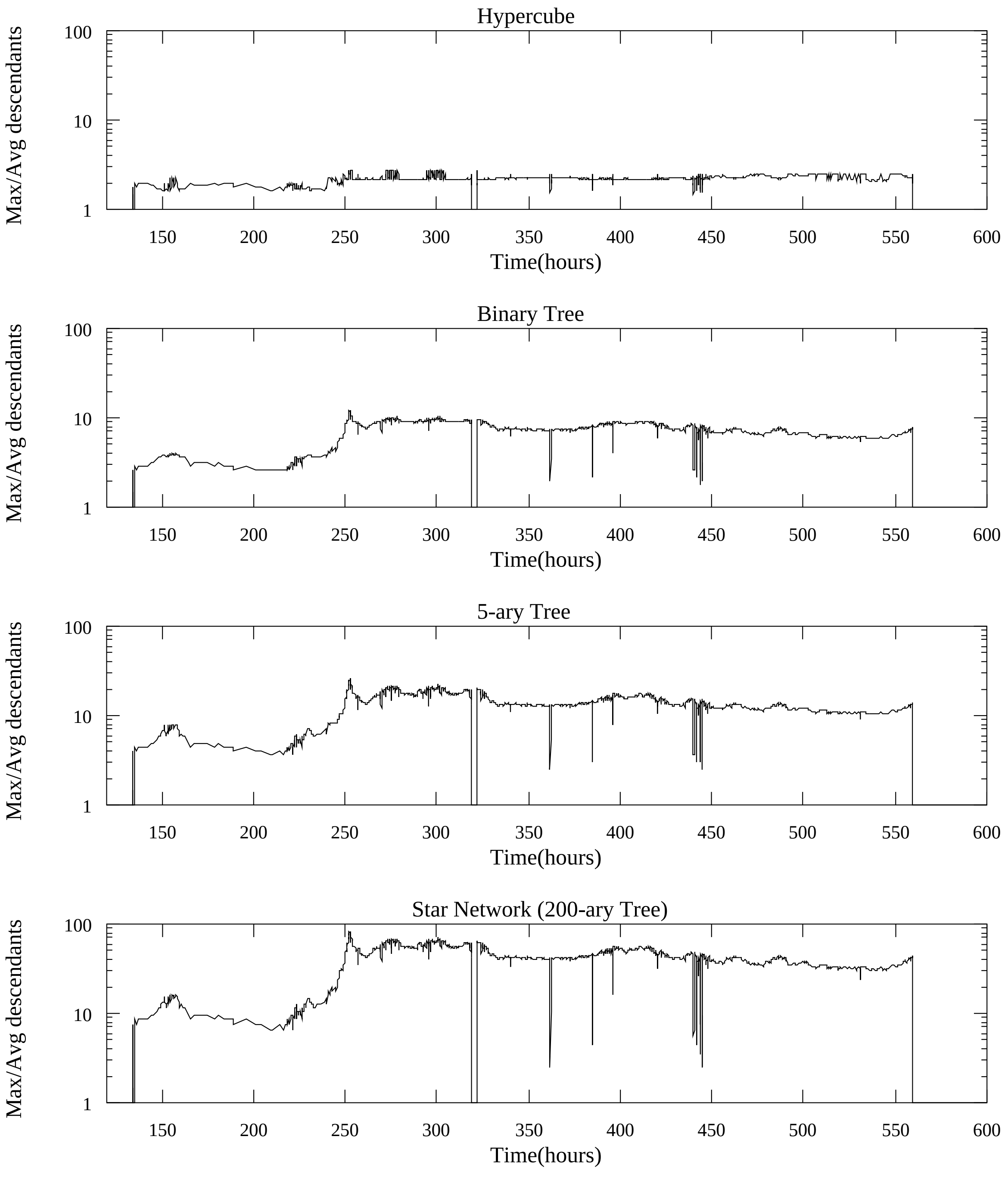


Figure 12: Maximum to Average Descendant Ratio $v_{max}/\overline{v}$.

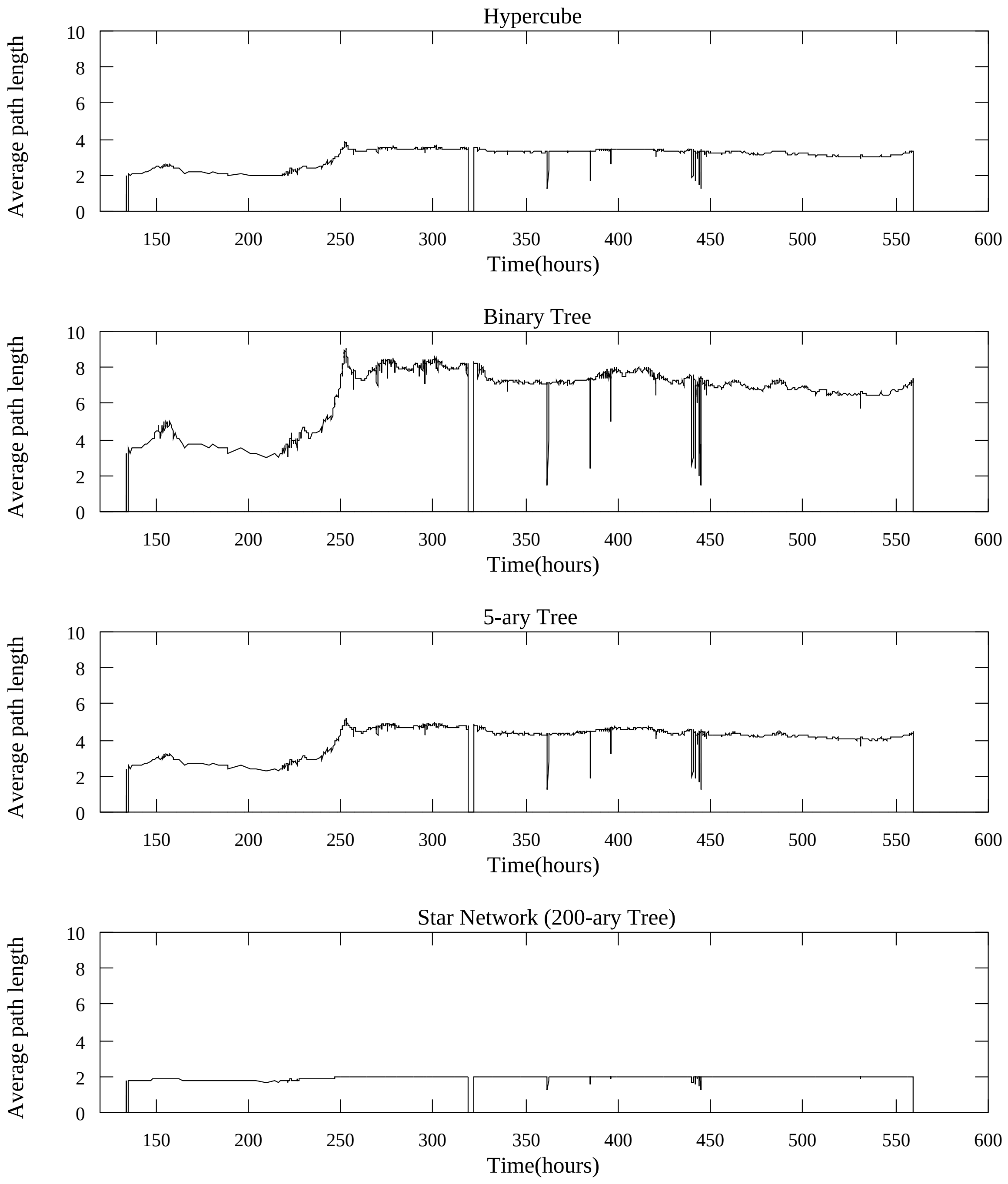


Figure 13: Average Path Length $\overline{p}$.