Comparison of k-ary n-cube and de Bruijn Overlays in QoS-constrained Multicast Applications

Richard West, Gerald Fry and Gary Wong

Computer Science Department
Boston University
Boston, MA 02215
{richwest,gfry,gtw}@cs.bu.edu

Abstract

Research on the construction of logical overlay networks has gained significance in recent times. This is partly due to work on peer-to-peer (P2P) systems for locating and retrieving distributed data objects, and also scalable content distribution using end-system multicast techniques. However, there are emerging applications that require the real-time transport *of data from various sources to potentially many thousands of subscribers, each having their own quality-of-service (QoS) constraints. This paper primarily focuses on the properties of two popular topologies found in interconnection networks, namely* k-ary n-cubes *and de Bruijn graphs. The regular structure of these graph topologies makes them easier to analyze and determine possible routes for real-time data than complete or irregular graphs. We show how these overlay topologies compare in their ability to deliver data according to the QoS constraints of many subscribers, each receiving data from specific publishing hosts. Comparisons are drawn on the ability of each topology to route data in the presence of dynamic system effects, due to end-hosts joining and departing the system. Finally, experimental results show the service guarantees and physical link stress resulting from efficient multicast trees constructed over both kinds of overlay networks.*

1 Introduction

This work addresses the problem of delivering real-time data streams on an Internet-scale, from one or more publishers to potentially many thousands of subscribers, each having their own service constraints. Target applications for this work include multimedia streaming of live video broadcasts, interactive distance learning and the exchange of time-critical data sets in large-scale scientific applications [22].

Many researchers are currently investigating end-system or application-level multicast techniques [11, 2] to distribute content in a scalable manner. This work is partly motivated by the lack of widespread deployment of IP multicast at the network-level and the inability of routers to employ application-specific stream processing services. Recent research on end-system multicast [8, 16, 4] has focused on the construction of logical meshes (or overlays) over which multicast trees or routes may be constructed, for the scalable delivery of data. Primary to these systems is the need to minimize physical link stress, or the duplication of data routed across a logical overlay network, while maximizing bandwidth. Our work is centered around the study of scalable overlay topologies [19, 14, 23, 3, 5] constructed over physical networks for content distribution of real-time streams to subscribers with their own quality-of-service (QoS) constraints. Depending on the application, these constraints may be in terms of throughput, delay, jitter or loss probability.

NARADA is one of the first end-system multicast approaches to construct a logical mesh for routing data. Physical hosts map to nodes in this logical mesh, or graph, depending on whether or not they aid in the construction of suitable paths between end-systems. Unfortunately, approaches such as NARADA suffer from scalability problems due to the overheads of adapting the mesh topology and considering all possible communication paths between end-hosts. This has motivated us to consider more regular topologies for logical interconnection networks between end-hosts, over which real-time data may be multicast. Specifically, parallel computer architectures such as the SGI Origin 2/3000 use k-ary n-cube topologies to connect processing elements, thereby yielding a trade-off in

the connectivity versus hop count between processing elements [10, 6, 7, 15]. Similarly, peer-to-peer (P2P) systems construct logical overlay topologies as a result of various distributed hashing techniques that include hypercubes [24], more generalized toroidal structures [20, 1] and *k-ary n-cubes* [21].

While many P2P systems make use of overlay networks to locate and retrieve distributed objects using $O(\log M)$ message exchanges (given a system of M hosts), there are no guarantees on the timely delivery of data. In contrast, there is a growing interest in applications that require the real-time *transport* of data streams to potentially many thousands of end-users, each with their own service constraints. It seems reasonable, therefore, to consider overlay topologies for routing real-time data streams in a manner that achieves low end-to-end delay. A carefully chosen overlay structure is important to: (1) minimize the hop distance over which data must travel, (2) allow for flexibility in the choice of paths available for multicast communication, without making routing choices excessively complex, and (3) avoid undue stress on the underlying physical network. Also, using end-hosts for real-time transport of data allows application-specific stream processing to be performed, which would otherwise be difficult in existing routers. For example, various packets of a given stream could be discarded at an intermediate host to reduce the bandwidth demand along subsequent network links to the destination.

In prior work, we investigate *k-ary n-cube* graphs as structures for overlay networks over which arbitrary messages are routed between hosts [12]. Part of this work studies the costs of adapting an overlay topology in a dynamic setting, as end-hosts join and depart the system over time [13]. In comparison, recent work by others analyzes various graph properties that are associated with delay and fault tolerant metrics from logical networks of various types, including those modeled after *de Bruijn* graphs [18].

Although *de Bruijn* graphs achieve a diameter closer to the Moore bound (mentioned in [18]), the *k-ary n-cube* structure is more flexible for routing QoS-constrained data streams in the sense that more shortest-path routes are available between a given pair of nodes. Thus, with *k-ary n-cube* networks, there is a larger range of *physical* end-to-end costs and a greater chance of finding routes that more closely fit the service constraints for each subscriber. Furthermore, it is unclear how an arbitrary message can be routed over a *de Bruijn* topology if there is a node failure on the path of the message while it is *en route* to its destination, whereas *k-ary n-cube* networks do not require a routing path to be determined completely at the time of transmission from the source.

Contributions: This paper provides a comparative analysis of *k-ary n-cube* and *de Bruijn* overlay networks, particularly considering properties of the logical topologies relevant for applications that involve real-time multicast communications and self-organization of a high volume of peers. Formulae are presented for the average and worst-case number of shortest paths between pairs of nodes in *k-ary n-cube* and *de Bruijn* graphs, in order to evaluate the topologies with respect to QoS route availability. We investigate the efficiency of each overlay structure for multicast communications and propose a method for dynamically handling host joins and departures in *k-ary n-cube* systems, similar to that used by systems such as Chord [24]

The rest of the paper is organized as follows. In Section 2, various properties of *k-ary n-cube* and *de Bruijn* graphs relating to route availability and fault resilience are introduced. Particular attention is paid to topological characteristics important for the delivery of real-time data streams. Section 3 includes a series of simulations that compare the performance (in terms of service guarantees and physical link stress) of various multicast routing approaches across overlay networks. Dynamic characteristics of overlay topologies are investigated in Section 4, detailing how to manage physical host to logical node mappings, and related state information, in the presence of system joins and departures. This is followed by a description of related work in Section 5. Finally, conclusions and future work are described in Section 6.

2 Overlay Routing

We prefer topologies that support routing policies yielding a large number of routes between any source and destination pair for two reasons. Firstly, the characteristics of physical links corresponding to logical edges in the overlay change dynamically, and our hypothesis is that a large number of routes to choose from allows for a selection of paths that fulfill as many QoS requirements as possible. Secondly, redundant routes increase the probability that a destination remains reachable, even if some intermediate nodes become unavailable either due to host failure or changes in system membership.

While there are many factors that influence acceptable routing efficiency or fault tolerance, it is important to characterize the numbers of routes between pairs of nodes using different overlay topologies. Specifically, we compare the availability of routes across *k-ary n-cube* and *de Bruijn* graphs, which are popular topologies for interconnection networks in various parallel computing architectures. For *k-ary n-cubes* and *de Bruijn* graphs with $M = k^n$ nodes, each node is represented by a logical identifier consisting of n base-k digits. Two nodes are connected in a *k-ary n-cube* iff their identifiers have $n-1$ identical digits, except for the

ith digit in both identifiers, which differ by exactly 1 modulo k. In contrast, in a *de Bruijn* there is a *directed* edge from node A to B iff the last $n-1$ digits of A match the first $n-1$ digits of B. Figures 1 and 2 show an example of each topology, where $k=2$ and $n=3$. Observe that these figures show examples of failed nodes. In our application domain, we envision a distributed system with dynamically-changing end-host membership. This factor contributes significantly to the importance of alternative routes between end-hosts, as will be seen below.

2.1 Route Availability

***k*-ary *n*-cubes:** Since we want to enforce the property that forwarding proceeds only along those edges which reduce the hop count to the destination, we consider only those routes that reach the destination with the minimal length, which we know is no more than $h = \lfloor \frac{k}{2} \rfloor n$.

In the usual case, the number of routes between maximally distant nodes, R, is given by the following:

$$R = \frac{\left(\lfloor \frac{k}{2} \rfloor n\right)!}{\left(\lfloor \frac{k}{2} \rfloor !\right)^n} \quad (1)$$

If $k > 2$ and even, we have:

$$R = \frac{2^n \left(\lfloor \frac{k}{2} \rfloor n\right)!}{\left(\lfloor \frac{k}{2} \rfloor !\right)^n} \quad (2)$$

Intuitively, Equation 1 is derived by first considering there are at most h hops between a pair of nodes, and there are $h!$ permutations of these hops that could make up possible paths. However, not all such permutations are valid. Specifically, of the $\lfloor \frac{k}{2} \rfloor$ hops traversed in a given dimension only one path is valid, so the denominator of Equation 1 eliminates all but one of the $(\lfloor \frac{k}{2} \rfloor)!$ permutations in each of the n dimensions. From Equation 1, when $k=3$ (which is optimal for minimizing h [12]), R reduces to $n!$ routes between maximally distant nodes. For the case when $k > 2$ and even, Equation 2 factors 2^n into the numerator of R to account for the doubling in possible paths in each of the n dimensions.

The local route redundancy (that is, the number of available neighbors which reduce the distance to the destination) is simply the number of dimensions in which the current node's address differs from the destination. Since there are n dimensions, the best case local redundancy is n, and the worst case is 1 (when the destination is an immediate neighbor). The average case is $\frac{n(k-1)}{2k}$.

de Bruijn graphs: In a de Bruijn graph of degree k and diameter $n = \log_k M$, if the number of hops remaining to some destination is less than n, then only one of the k neighbors leads to a lower hop count to the same destination, so in this sense there is no route redundancy in either the local or the global case. However, if we relax the restriction that messages are forwarded only along routes that reduce the distance to the destination for the first (source) node, then we can find (in general) $k-1$ non-overlapping routes each of hop-count distance $n+1$ [18].

***c*-redundant de Bruijn graphs:** In a standard de Bruijn graph, once a destination is specified, each node has exactly one neighbor whose hop count to the destination is one smaller, therefore there is no redundancy nor flexibility in routing at intermediate nodes. However, if we are willing to increase the degree of the graph from k to ck for some constant $1 \leq c \leq k$, and allow node x to forward to the set of neighbors $(ik^{n-1}+kx+j) \bmod k^n$ for all $0 \leq i < c$ and $0 \leq j < k$, we achieve a topology that we designate as a *c-redundant de Bruijn* graph. In such a structure, each node (other than the k nodes neighboring the destination) has exactly c neighbors to choose from, each of which reduces the distance to the destination. This increases the upper bound on the number of routes through the graph to c^{n-1}.

Properties of some of the above types of graphs with various parameters are shown in Table 1.

2.2 Fault resilience

We now consider the case where a node along a shortest path from source to destination fails, and the destination must be reached via an alternative route. Since we do not expect any node to have global knowledge about the state of the network, we assume that the current forwarding node is H [1] hops away from the ultimate destination, and the routing agent at this host has detected that the preferred neighbor $H-1$ hops from the sink is unavailable. We presume that $H > 1$, ruling out the case in which the destination itself fails.

***k*-ary *n*-cubes:** We are now left with $(H-1)(H-1)!$ shortest paths. There are only $H-1$ instead of the usual H neighbors available at the current hop. However, there remain the usual $(H-1)!$ paths from the next hop onward.

Various routing policies involving k-ary n-cube topologies can be employed to route around a failed node along the path from source to destination by eliminating the failed node from the set of next-hop alternatives. For example, given the resulting set of available neighbors of the forwarding host, a routing decision is made based on physical proximity metric optimization, or by left-/right-shifting successive digits in the logical identifier until the ID of an active neighbor is found. These routing approaches will be considered further in Section 3.

de Bruijn graphs: In this case, the shortest path is unique, so there is no backup path as short as the original.

[1] Where H is the Hamming distance between source and destination node IDs.

				Hop count		Local routes			Global routes	
		Nodes	Degree	Med	Max	Min	Med	Max	Med	Max
k	n	k-ary n-cubes								
2	20	1M	20	10	20	2	5	20	3.6M	2×10^{18}
3	13	1.6M	26	9	13	2	5	13	363K	6.2G
	k	de Bruijn graphs								
	2	1M	2	19	20	1	1	(2)	1	(2)
	3	1.6M	3	13	13	1	1	(3)	1	(3)
	4	1M	4	10	10	1	1	(4)	1	(4)
	5	2M	5	9	9	1	1	(5)	1	(5)
	20	3.2M	20	5	5	1	1	(20)	1	(20)
	26	12M	26	5	5	1	1	(26)	1	(26)
k	c	c-redundant de Bruijn graphs								
2	2	1M	4	19	20	2	2	2	256K	512K
3	3	1.6M	9	13	13	3	3	3	531K	531K
4	2	1M	8	10	10	2	2	2	512	512
4	4	1M	16	10	10	4	4	4	256K	256K
5	4	2M	20	9	9	4	4	4	64K	64K
5	5	2M	25	9	9	5	5	5	390K	390K
10	2	1M	20	6	6	2	2	2	32	32
13	2	5M	26	6	6	2	2	2	32	32

Table 1. A comparison of k-ary n-cubes versus de Bruijn graphs

c-redundant de Bruijn graphs: In this topology there will be $(c-1)c^{H-1}$ minimal length routes, because we have $c-1$ instead of the usual c neighbors at the current hop, and the expected c^{H-1} paths from the subsequent node to the destination.

If so many failures have occurred that no path of ideal length still exists, the question still arises whether it is possible to find other backup routes which are longer but still reach the destination. We now consider this issue for each topology as follows:

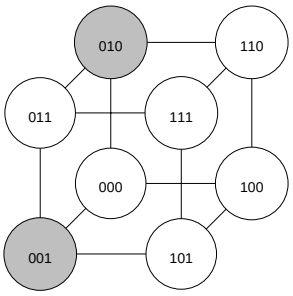

Figure 1. A k-ary n-cube containing failed nodes

k-ary n-cubes: If each node maintains only local knowledge about the graph in a *k-ary n-cube*, then the failure of just two intermediate nodes can make a destination unreachable within $h = \lfloor \frac{k}{2} \rfloor n$ hops. For instance, in Figure 1, nodes 001 and 010 have failed, leaving nodes 000 and 011 unable to reach each other in $h = 3$ hops or less. This problem occurs in a *k-ary n-cube* graph when a pair of nodes separated by a Hamming distance of 2 fail simultaneously.

At least two nodes must fail before all paths of preferred length become unavailable, but once this happens, our standard routing policy fails (as described in Section 3). In theory there may exist a path to the destination by routing in the "wrong" direction along some dimension, which would increase the hop count by at least 2 if $k = 2$ and at least 1 if $k = 3$, but a sophisticated routing policy is needed to guarantee convergence, because normally we rely on decreasing the hop count at each node to avoid cycles.

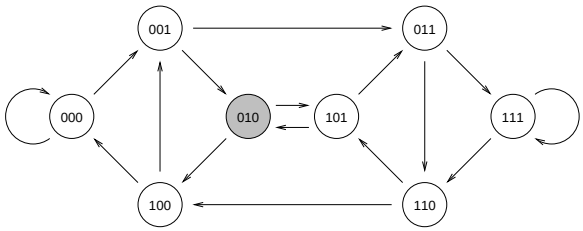

Figure 2. A de Bruijn graph containing a failed node

***de Bruijn* graphs:** At each node there are $k-1$ backup (non-overlapping) paths with worst case hop count n to any destination, so if the source is aware of a failure along the primary path (which has maximum hop count $n-1$), it can choose one such backup path, increasing the hop count from $n-1$ to n in the worst-case.

In a standard de Bruijn graph, a single failed node along the primary path between any pair of nodes increases the path length. Consider the path from node 001 to 101 in Figure 2. If all nodes are operating, then the route $001 \rightarrow 010 \rightarrow 101$ is optimal. However, if node 010 fails, then the best available path is $001 \rightarrow 011 \rightarrow 110 \rightarrow 101$.

If each node maintains only local knowledge of the network, then it is the responsibility of intermediate nodes to detect the failures, and the worst case hop count increases from $n-1$ to $x+n$, where x is the number of hops between the source and the first node that is capable of detecting the failure of the path with optimal hop count.

c-redundant de Bruijn graphs: This topology is resilient enough that at least c nodes must fail before the length of the preferred route increases. In this scenario, the result is

similar to the standard de Bruijn case, except we now have $ck-1$ backup paths (though some will overlap). Again, the worst case hop count increases from $n-1$ to $x+n$.

3 Multicast Tree Construction

The efficiency with which data can be multicast to a subset of hosts is dependent upon the overlay topology, since the placement of logical edges between nodes determines the paths that are used in the underlying physical network. Furthermore, the availability of multiple distinct routes between nodes and the ability to redirect via alternate nodes en-route to a destination provide the mechanisms for constructing multicast trees in a decentralized fashion. We evaluate methods for building efficient multicast trees using the following metrics:

- *Relative delay penalty*: This is calculated by dividing the total cost to route a message over a given set of edges in the overlay, by the cost of routing between the same end-points using unicast routing across the physical network.
- *Average link stress*: Communication between hosts is restricted to the set of edges supported by a logical topology that independently maps each logical link to a *path* in the underlying physical network. Since we are interested in constructing multicast trees using such *virtual* topologies, it is possible for distinct edges involved in transmission of multicast data to overlap in the physical links that they represent. Therefore, in multicasting a single message from a publisher to a set of subscribers, there are a number of message duplications that travel along a single link in the physical network. *Average link stress* is defined as the number of message transmissions over all physical links divided by the number of physical links involved in multicasting a single message to each destination host. In building efficient multicast trees using overlay networks, we prefer regular topologies which minimize *average link stress*, thus placing less strain on physical communication resources.
- *Average normalized lateness*: We assume that each subscriber host specifies a real-time service constraint on the data it receives in the form of a maximum end-to-end cost metric, such as a latency deadline. For a given subscriber host, $s \in S$, and its corresponding latency constraint, c, a *normalized lateness* value, $L(s,c)$, is calculated using the following formula:

$$L(s,c) = \begin{cases} 0 & \text{if } s.cost(p) \leq c \\ \frac{s.cost(p)-c}{c} & \text{if } s.cost(p) > c \end{cases},$$

  where $s.cost(p)$ denotes the total cost of routing a message along the logical network from publisher host p to subscriber s. The lateness values are normalized in order to eliminate bias towards subscribers with large latency constraints, relative to other subscriber hosts in the group, and all subscriber hosts with satisfied constraints are assigned a *normalized lateness* of zero. The *average normalized lateness* is defined as the mean of *normalized lateness* values over all hosts in S.
- *Success ratio*: For a group of subscribers, S, the success ratio is the fraction of all members in S that receive data streams in accordance with their service requirements.

Other metrics for evaluating the efficiency of multicast trees include *multicast tree weight*. Given a weighted graph, $G = (V, E, w)$, a publisher node, $p \in V$, and a set of subscriber nodes $S \subset V$, the multicast tree, $T = (V', E', w)$, spanning all nodes in $S \cup \{p\}$ and rooted at node p, is associated with a weight, $\delta(T) = \sum_{e \in E'} w(e)$. Note that T may contain nodes in $V - S$ that correspond to *intermediate* hosts responsible for forwarding messages towards subscribers. It makes sense to minimize the value of $\delta(T)$ while keeping the complexity of tree construction algorithms manageable for highly dynamic systems on an Internet scale. The optimal solution is a *Steiner* tree over points S, though the problem of finding such a tree given an arbitrary graph is known to be NP-hard [17]. Considering this paper focuses on QoS-constrained routing via multicast trees built using overlays, we do not consider the multicast tree weight metric any further. Instead, we compare the multicast trees generated from de Bruijn graphs with trees formed using policies for routing in *k-ary n-cube* overlay networks using the other metrics described above.

Experimental Results: A series of experiments were conducted to quantitatively characterize the support for multicast communications inherent in *k-ary n-cube* and *de Bruijn* overlay networks. In each experiment, multicast trees are constructed in the *k-ary n-cube* according to three policies:

- *Combined ODR paths:* A tree is constructed as a combination of paths computed using the *ordered dimension routing* (ODR) scheme described in [12]. Simply, at each hop along a path to a specific destination, a message is forwarded in a fixed ordering of dimensions that reduces the remaining hop distance.
- *Combined Greedy paths:* Local physical proximity information is used to construct paths as described in [12]. In this scheme, a greedy approach is taken in the sense that the lowest cost neighbor of a forwarding node is chosen as the next hop in the logical path towards a subscriber.
- *Combined Random paths:* A message is forwarded to a random neighbor of a given hop along a path to the destination, as long as it reduces the remaining hop distance. Further details can be found in our previous

work [12].

In directed *de Bruijn* graphs, paths between destination nodes and the publisher are computed according to the *de Bruijn* routing scheme mentioned in the work on ODRI [18], and these routes are combined to form a multicast tree. Specifically, each hop along a given path has a n-digit base-k identifier and the next hop is selected by left-shifting this value by one digit. For example, in a de Bruijn graph with $k = 2$ and $n = 3$, the path between nodes 000 and 111 would be $000 \rightarrow 001 \rightarrow 011 \rightarrow 111$. While this does not consider physical proximity information as in the case of greedy routing over k-ary n-cubes, future work will involve a similar proximity-based routing scheme over de Bruijn graphs. As will be seen in our experiments on QoS service guarantees, we consider an undirected de Bruijn graph and choose a next hop based on whichever neighbor has the lower link cost, while also reducing the hop count to the destination. In any case, the union of the paths between a given publisher and all corresponding subscribers is the basis for a de Bruijn multicast tree.

Each of the multicast tree construction policies mentioned above specifies the method of selecting paths from which to build a tree in the logical network that spans the set of subscriber vertices. Given the subscriber set, S, a publisher node, p, and a graph, G, a tree, T, that minimizes the logical hop-count between p and each member of S is built according to the following algorithm:

(1) Add node p to T
(2) For each node $s \in S$, use *combined ODR paths*, *combined random paths*, *combined greedy paths*, or *combined de Bruijn paths* to route from vertex s *towards* p in G, adding each node and edge traversed to T, until a node is reached that already exists in T.

A transit-stub graph consisting of 5050 nodes was generated using GT-ITM and the Stanford GraphBase graph library [25] to simulate a physical router infrastructure. On average, each transit node is connected to 10 stub domains, and a pure random model is used to generate edges within each of the top level, transit, and stub graphs. The transit-stub network consists of 10 transit domains, which are connected with random graph parameters $\alpha = 0.5$, $\beta = 1.0$. There are an average of 5 nodes per transit domain and 10 nodes per stub domain. Random graph parameters within transit domains are $\alpha = 0.6$, $\beta = 1.0$, and stub graphs are given parameters $\alpha = 0.42$, $\beta = 1.0$. In the top level graph and each transit domain, edges are generated along dimensions scaled to a length of 20, whereas in the stub graphs the scaling parameter is 10. Finally, each host is associated with a randomly chosen physical router.

Relative delay penalty: For the first experiment, relative delay penalties of multicast tree routes over *de Bruijn* and *k-ary n-cube* topologies are compared for parameters $k = 2$ and $n = 16$. In addition to the routing schemes mentioned above, shortest path trees (with respect to physical costs) are generated by applying Dijkstra's algorithm to each of the weighted graphs used to model the two overlay structures. For each subscriber host, the experiment records the total end-to-end physical cost, c_i, of routing a message between the publisher node, p, and the subscriber, s_i. Figure 3 presents the cumulative distribution of delay penalties with respect to unicast routing for the ODR, de Bruijn, and greedy routing schemes.

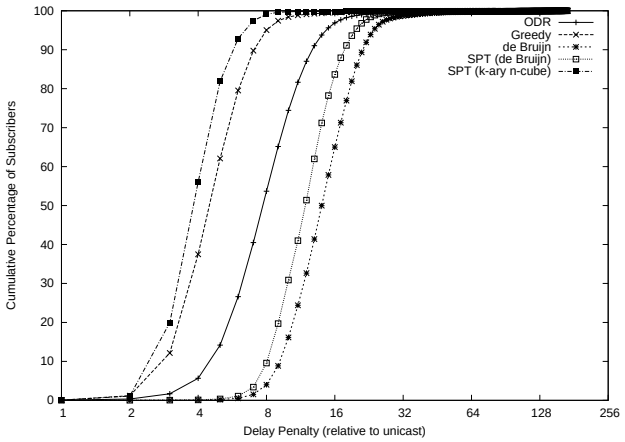

Figure 3. Cumulative distribution of delay penalties (relative to unicast) in *k-ary n-cube* and *de Bruijn* topologies ($k = 2$, $n = 16$).

As in the work on NARADA, the relative delay penalties in all cases suggest that overlay routing is a plausible alternative to using IP multicast. In our case, regular overlay topologies provide a scalable solution to end-system multicasting of QoS-sensitive streams. Moreover, a simple greedy routing scheme proves nearly as good as Dijkstra's shortest path scheme, with the advantage that it is fully decentralized and is largely unaffected by dynamic changes to the underlying distributed system. Even if Bellman-Ford routing were used, it would require potentially multiple message exchanges between neighboring hosts to converge to shortest routes relative to a given host. Such convergence may not be possible when hosts frequently join and depart.

Average link stress: Under the simulation scenario involving the transit-stub physical model described above, values for *average link stress* are computed for group sizes of $3^i \mid i \in [5 \cdots 12]$, using both a *3-ary 13-cube* and a de Bruijn graph with a radix of 10 and a dimension of 6. Observe that a *3-ary 13-cube* has 3^{13} nodes while the de Bruijn topology supports one million nodes. Likewise, the diameter of the de Bruijn graph is 6 compared to 13 in the case of the *3-ary 13-cube*. These two configurations where chosen because: (a) $k = 3$ minimizes the diameter and average hop distance in a k-ary n-cube (as shown in our earlier work [12]) and, (b) the total nodes in each graph is on the scale of the sort of distributed systems we envision being built on the Internet.

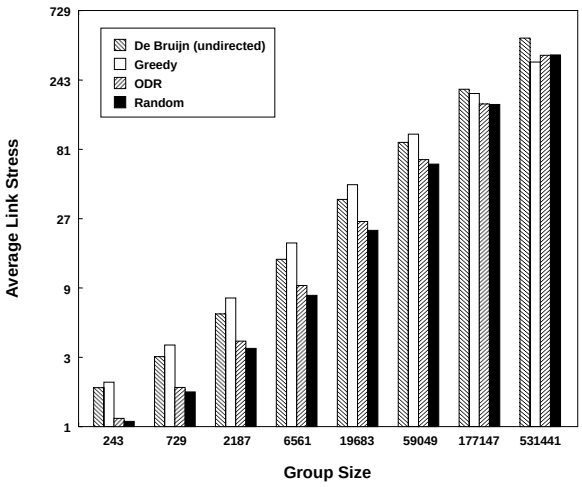

Figure 4. Physical link stress versus group size for each routing algorithm's resultant multicast tree

Figure 4 shows the average link stresses for the aforementioned routing schemes over the two overlay topologies. As can be seen, the link stress increases as a function of the group size for overlay multicast trees. This increase is due to the sharing of physical links among multiple logical edges of each overlay. Interestingly, the average link stress using greedy routing over k-ary n-cubes is higher than in all other cases for small group sizes. However, as the group size grows, the link stress with greedy routing is lower. Specifically, the rate of increase in link stress with greedy routing is less than in all other cases, as the group size increases.

Observe that the multicast tree formed from the union of greedy routes is more likely to have common edges in the logical overlay when the number of subscribers increases. This factor contributes to the lower physical link stress for larger group sizes. We envision that greedy routing will do even better if we employ an adaptive node reassignment scheme as in our previous work [12], to ensure publishing hosts and corresponding subscribers are clustered within short hop distances over a given overlay. Given that in our current approach hosts are randomly assigned to logical positions in an overlay topology, it makes sense to reassign node IDs to physical hosts when publishers and corresponding subscribers are known. For this reason, we are currently investigating an adaptive node reassignment scheme as in Figure 5, where $i.cost(P)$ denotes the total end-to-end cost of routing a message between hosts P and i along a given path in the overlay. Experimental results using this adaptive scheme are outside the scope of this paper but such approaches have been shown to be beneficial in our earlier work [12].

```
Subscribe(Subscriber S, Publisher P, Depth d) {

 // recursively apply algorithm up to D hops from P
 if (d = D) return;

 Find a neighbor i of P such that
 i.cost(P) is maximal for all neighbors;
 Find a neighbor j of P such that
 j.cost(P) is minimal for all neighbors;

 if (S.cost(P) < i.cost(P)) {
   swap logical positions of i and S;
   Subscribe (i, S, d+1);
 }
 else Subscribe(S, j, d+1)
}
```

Figure 5. Adaptive node re-assignment algorithm

QoS guarantees: The next experiment considers the likelihood of satisfying subscriber-specific real-time service constraints in *de Bruijn* and *k-ary n-cube* overlay topologies. For simplicity, we assume each subscriber is associated with a service constraint representing the maximum latency that can be tolerated in receiving the multicast message (although the results can be generalized to any additive or linear cost metric). Such deadlines are generated according to a uniform random distribution, ranging from the minimum latency on a physical link to the maximum latency along a physical link multiplied by the worst-case hop count in the *3-ary 13-cube* graph representing an overlay topology.

We simulate greedy routing over a *3-ary 13-cube* from the source node with logical ID 0 to each subscriber host present in randomly selected groups of various sizes, and a *normalized lateness* value is calculated for each receiver in each group. For the *undirected de Bruijn* topology, there exist two possible routes over which a message can be forwarded for each destination host. For this experiment, we consider paths resulting from *right shifting* or *left shifting* the logical identifier of the forwarding node to move one hop closer to the destination. In actuality, reverse routes from each subscriber to the publisher node are computed (this can be done since we assume physical link costs are symmetric), and the shifting direction is based on the latency corresponding to the logical link adjacent to the publishing host. If *right shifting* yields a lower first-hop link cost, then this method is used to construct the routing path, otherwise, *left shifting* is performed. This method of routing provides a policy analogous to greedy routing in the *k-ary n-cube* network, except the decision can be made only once for each path in the *de Bruijn* case.

Normalized Lateness values are calculated for subscriber hosts in the *de Bruijn* overlay as in the simulated *k-ary n-cube* network. For each group and each topology, the mean of the *normalized lateness* measurements is computed, and the *average normalized* lateness values are charted in Fig-

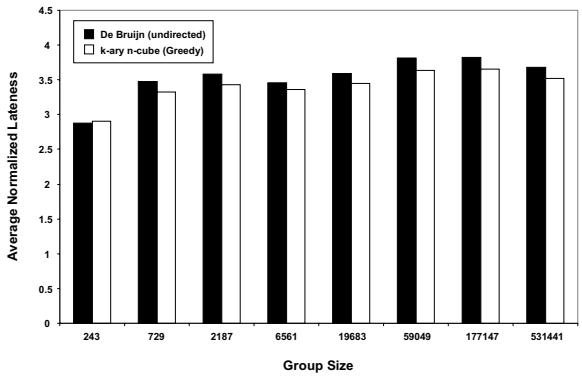

Figure 6. Average normalized lateness versus group size in k-ary n-cube and *de Bruijn* overlays

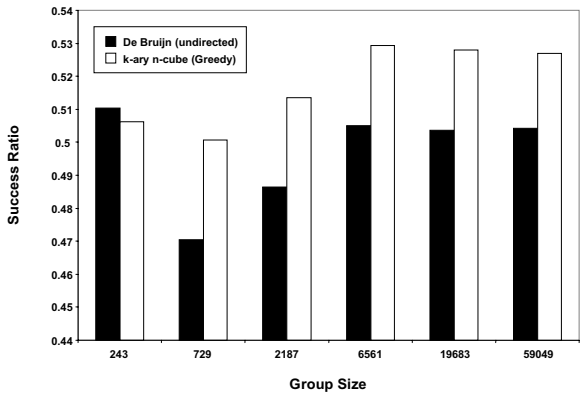

Figure 7. Success ratio versus group size in k-ary n-cube and *de Bruijn* overlays

ure 6. The results show that the *k-ary n-cube* topology provides routes found by the greedy routing policy that reduce *average normalized* lateness in comparison with the simulated *de Bruijn* overlay, for each group considered, except for the subscriber set containing 243 hosts.

In addition to the data recorded for *average normalized lateness* over the multicast simulations described above, we compute the *success ratios* for multicast groups containing 3^5, 3^6, 3^7, 3^8, 3^9, and 3^{10} subscriber hosts. The *success ratio* corresponding to a particular group is calculated by counting the number of hosts for which service deadlines are not exceeded and dividing by the number of subscribers in the corresponding group. The resulting data is shown in Figure 7, and it is apparent from the graph that greedy routing in the k-*ary* n-*cube* overlay performs slightly better than the *de Bruijn* structure (except in the group containing 243 hosts), even though the worst-case logical hop count between nodes in the k-*ary* n-*cube* is significantly greater than in the *de Bruijn* case.

The values presented in Figures 6 and 7 are intended to provide a measure of the performance of *de Bruijn* routing *relative* to greedy routing over a k-*ary* n-*cube* topology. The average normalized lateness and success ratio for each group size in the simulations are dependent upon the parameters used in generating subscriber constraints and physical link costs, and are not intended to represent the absolute behavior in an actual system. However, the experiments demonstrate the likelihood of satisfying subscriber-specific QoS constraints in k-*ary* n-*cube* overlays when compared with *de Bruijn* topologies.

Summary: The results from the experiments described in this section provide insights into the characteristics of overlay topologies for multicast data distribution. Regardless of the overlay topology, it appears that regular graph structures are potentially suitable for the large-scale delivery of QoS-sensitive streams between publishers and many thousands of subscribers. Applications such as Internet TV, supporting the delivery of live video and audio broadcasts could benefit from routing over regular overlays.

More sophisticated routing algorithms than the ones investigated above may consider multiple kinds of service constraints (e.g., jitter, bandwidth, or window-constraints [26]) and/or base forwarding decisions on more information about the underlying physical network. However, we believe that it is necessary for such policies to exhibit low complexity, so that physical measurements remain accurate in a rapidly changing network environment.

4 Dynamic Characteristics

A major advantage of the simple greedy routing algorithm is that it requires only local knowledge of a given graph, which means that nodes may join and leave a group without broadcasting group management information to all existing members. In effect, the group membership information is itself distributed throughout all hosts in the system. This property is essential to allow groups to scale to very large sizes, for which managing $O(M)$ state at each node becomes prohibitive.

To apply the overlay topology onto the multicast group, we are considering a scheme which is essentially the same as ODRI [18], and heavily influenced by Chord [24].

In a k-*ary* n-*cube*, the neighbors of node x are the set:

$$x - k^i \left(\left\lfloor \frac{x}{k^i} \right\rfloor \bmod k \right) + k^i \left(\left\lfloor \frac{x}{k^i} \pm 1 \right\rfloor \bmod k \right)$$

for all $0 \leq i < n$, and are connected with undirected edges.

In a standard de Bruijn graph of degree k and diameter n, the neighbors to which node x may forward are those nodes numbered $(kx + i) \bmod k^n$ for all $0 \leq i < k$.

Instead of assigning each physical host a single identifier x as above, we allow hosts to maintain groups of identifiers, which simplifies the handling of node joins and departures. In this approach, we keep the logical ID space

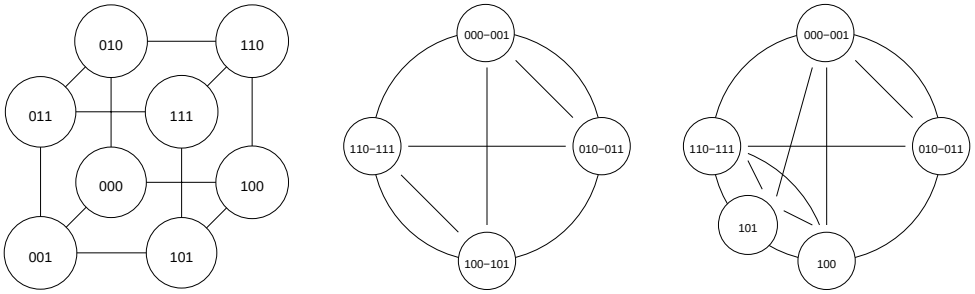

Figure 8. (a) An example k-ary n-cube graph, and (b) a corresponding overlay network with 4 physical hosts, followed by (c) the same network after a 5th host joins

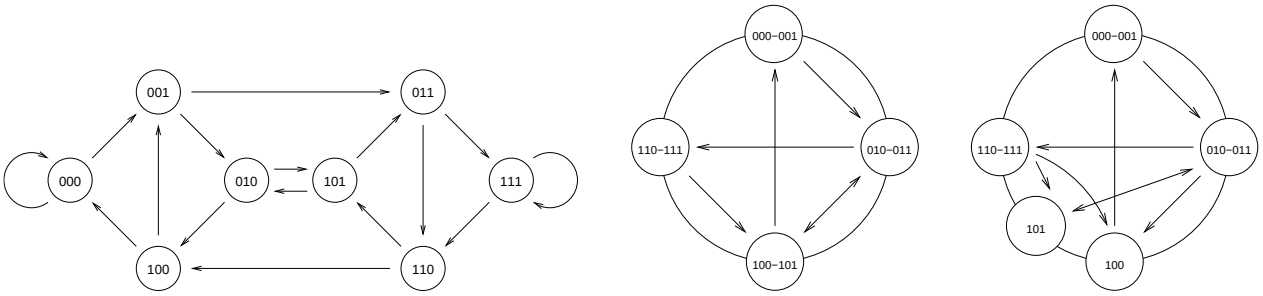

Figure 9. (a) An example *de Bruijn* graph, and (b) a corresponding overlay network with 4 physical hosts, followed by (c) the same network after a 5th host joins

constant regardless of the current number of physical hosts in the system. Specifically, for both k-*ary* n-*cubes* and *de Bruin* graphs we limit the maximum number of hosts in a group to k^n, by choosing a suitable value for n. We then assign each physical host a contiguous subset of the ID space $[x_1, x_2], x_2 \geq x_1$, such that each identifier belongs to exactly one host. It is vital that this property holds over the lifetime of the group. Therefore, a new host joins the group by taking over one or more contiguous identifiers from a randomly chosen existing node. Likewise, when a host leaves the system (or fails), its identifiers must be reassigned to one of the two physical hosts adjacent to it in logical ID space. The neighborhood rules given above are then generalized to the rule that there is an edge between host x (with ID range $[x_1, x_2]$) and host y (with ID range $[y_1, y_2]$) if and only if there exists some logical edge (u, v) such that $x_1 \leq u \leq x_2$, $y_1 \leq v \leq y_2$. Most of the properties of k-*ary* n-*cube* and *de Bruijn* graphs still apply (or apply in the average case) under this scheme for dynamic overlay management [18]. Examples of this procedure are shown in Figures 8 and 9, for k-*ary* n-*cube* and *de Bruijn* overlays, respectively. Note that the arcs connecting the nodes in a circle are not part of the interconnection graphs but are merely there to represent the ring of logical IDs.

Observe that when a host departs a distributed system using the above scheme, it is possible to find a route between two end-points because routing state information is redistributed to remaining hosts. This means there is always a route through the nodes of a logical overlay but the actual end hosts involved in that route may change over time. One may therefore ask why it is important to have as many alternative routes through an overlay, as discussed in Section 2. The reason is that there is a delay due to the redistribution of routing state information amongst remaining hosts, to update mappings of physical hosts to logical node IDs. Given that we envision a system whereby real-time streams may have partially traversed a logical path through an overlay, it is important for a given host to be able to find reachable next-hops towards the destination before its neighbor (routing) tables have been updated as a result of dynamic changes to the system. This is certainly an area of further investigation, and we plan to study the lag effects of exchanging state information in the presence of host joins and departures while streams are being propagated to subscribers.

5 Related Work

Much of the recent work in P2P systems focuses on the construction of *virtual* network topologies that allow hosts to communicate assuming some existing underlying unicast routing service. Such overlay networks allow data to be distributed and processed without compromising the flexibility required for application-specific stream processing at inter-

mediate nodes along routing paths. The predominant literature in the area of overlay topology construction can be divided into overlays modeled around regular graph structures and logical networks in which links are formed based on random message exchanges.

Systems such as Pastry, Chord, CAN, and those based on *de Bruijn* graphs fall into the category of regular overlay networks, whereas NARADA attempts to randomly restrict logical connections based on metrics that affect QoS constraints [21, 24, 20, 8]. It has been shown that regular overlay topologies are more scalable than the latter approach, but the restriction to a fixed set of logical links introduces the risk of neglecting links with low enough cost to satisfy service constraints for real-time applications. However, in systems involving millions of hosts and rapidly changing network characteristics, the lower complexity of policies employed to service host joins, departures, and routing of multicast messages over regular topologies results in easier implementation and an increase in predictability.

Overlays based on *k-ary n-cube* graphs exploit the above mentioned advantages of a regular structure while providing low-complexity routing policies in which decisions can be made on a hop-by-hop basis. Such policies may take advantage of proximity metrics along physical paths that map to logical edges while a message is *en route* from source to destination. Furthermore, *k-ary n-cube* overlays are densely connected compared to other topologies in the sense that the number of available routes with optimal hop count is $O(n!)$, where $n = \log_k M$ represents the number of dimensions in a system supporting M hosts.

Support for multicast at the application level has been shown to be advantageous for applications involving many receivers with QoS constraints. The NARADA protocol, for example, shows how data can be streamed over a randomly constructed overlay mesh [2]. Other protocols for application-layer multicast include M-RTP, which constructs trees consisting of multiple unicast paths [9].

An additional body of work is centered around the optimization of costs based on nonlinear metrics (e.g., bandwidth) involved in multicast applications. SplitStream balances the responsibility of message forwarding among peers involved in a forest of multicast trees by splitting stream data into multiple *stripes*, and Bullet is concerned with maximizing the achievable bandwidth at which data can be transmitted to subscribers over pre-computed multicast trees [4, 16]. Each of these systems assume the existence of an underlying network structure responsible for maintaining connections and routing state at a lower level of host communication, such as a physical infrastructure or a *virtual* topology. For example, SplitStream leverages the Pastry/Scribe routing services for computing individual multicast trees. Thus, the contributions in overlay construction and analysis can be considered complementary to the design of systems involving multiple channels of streaming data.

Since it is desirable for multicast trees to exhibit a low sum of weights on edges, some work proposes approximating *Steiner* trees for data distribution. Algorithms exist for approximating solutions to the *Steiner* tree problem, such as Kou's algorithm [17]. Other work seeks to adjust the tradeoff between multicast trees with shortest paths and low weight. For our purposes, multicast routing policies must be of low complexity in order to maintain a high degree of scalability and fault tolerance. Hence, we focus on algorithms that do not assume that each node has complete information concerning the structure of the interconnection topology.

6 Conclusions and Future Work

This paper compares several well-known graph topologies for scalable QoS-constrained overlay routing. The relative delay penalties of logical overlay routing, with respect to unicast routing over a physical network, suggest that regular graph topologies may be suitable for the delivery of soft real-time streams on the scale of the Internet. We observed that a simple greedy routing scheme, that does not require global state information at each node, performs almost as well as an optimal shortest path routing scheme. In the latter case, a distributed form of Dijkstra's algorithm, using a Bellman-Ford technique would be required in practice, but this is not particularly desirable in a setting where hosts can dynamically join and depart a large-scale distributed system. Given that Bellman-Ford routing requires each host to potentially exchange multiple messages with its neighbors, a shortest path route may not be determined before the membership of the system changes. For this reason, simple decentralized routing strategies over regular overlay topologies are desired. Our results suggest the costs of such approaches are potentially acceptable for many real-time and multimedia streaming applications, such as live video broadcasting.

Of the topologies we have considered, if the degree of nodes is fixed, then k-ary n-cubes yield the greatest route redundancy, but standard de Bruijn graphs minimize the distances between pairs of nodes. It appears that de Bruijn graphs are the most efficient structure in terms of end-to-end hop count, but that k-ary n-cubes might be preferred in practice when fault tolerant routing and QoS requirements are considered. By fault tolerant, we mean that if a real-time stream has already traversed part of a path, it is still capable of reaching the destination given a host along the remainder of the path either fails or departs the system.

We argue that with k-ary n-cubes, a path bypassing a failed node is more likely to meet end-to-end QoS requirements, because (a) there are more alternative paths to

choose, and (b) the hop distance is less likely to increase compared to the original path. However, c-redundant de Bruijn graphs warrant further investigation, since for sufficient values of c, they can approach the average case route redundancy (and exceed the worst case) of k-ary n-cubes with equivalent node degree, while potentially reducing the distance to the destination.

As part of our future and ongoing research, our vision is to build an Internet-wide distributed system for processing and delivery of real-time data streams to many thousands of subscribers, each having their own QoS constraints. This work not only encompasses the design and analysis of logical overlay networks to interconnect end-hosts for purposes of data transport, but also the design of efficient end-host architectures that may safely be extended with application-specific stream processing agents.

References

[1] D. Banerjee, B. Mukherjee, and R. Suryanarayan. The multidimensional torus: Analysis of average hop distance and application as multihop lightwave network. *Proc. ICC '94, Vol. 3*, pages 1675–1680, May 1994.

[2] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *Proceedings of ACM SIGCOMM*, August 2002.

[3] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller. Construction of an efficient overlay multicast infrastructure for real-time applications. In *IEEE INFOCOM*, San Francisco, CA, April 2003.

[4] M. Castro, P. Druschel, K. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: high-bandwidth multicast in cooperative environments. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, volume 37, 5 of *Operating Systems Review*, pages 298–313, New York, Oct. 19–22 2003. ACM Press.

[5] Y. Chawathe. *Scattercast: An Architecture for Internet Broadcast Distribution as an Infrastructure Service*. PhD thesis, University of California, Berkeley, December 2000.

[6] A. A. Chien. A cost and speed model for k-ary n-cube wormhole routers. *IEEE Transactions on Parallel and Distributed Systems*, 9(2):150–162, 1998.

[7] A. A. Chien and J. H. Kim. Planar-adaptive routing: Low-Cost adaptive networks for multiprocessors. *Journal of the ACM*, 42(1):91–123, 1995.

[8] Y.-H. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *ACM SIGMETRICS 2000*, pages 1–12, June 2000.

[9] R. Cohen and C. Kaempfer. A unicast-based approach for streaming multicast. In *IEEE Infocom*, pages 440–448, 2001.

[10] W. J. Dally. Performance analysis of k-ary n-cube interconnection networks. *IEEE Transactions on Computers*, 39(6):775–785, 1990.

[11] P. Francis. Yoid: Extending the multicast Internet architecture, 1999. On-line documentation: http://www.aciri.org/yoid/.

[12] G. Fry and R. West. Adaptive routing of QoS-constrained media streams over scalable overlay topologies. In *Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, May 2004.

[13] G. Fry and R. West. Dynamic characteristics of k-ary n-cube networks for real-time communication. In *Proceedings of the 5th International Conference on Communications in Computing*, June 2004.

[14] J. Jannotti, D. Gifford, K. Johnson, M. Kaashoek, and J. O'Toole. Overcast: Reliable multicasting with an overlay network. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation*, October 2000.

[15] M. Kang, C. Yu, H. Y. Youn, B. Lee, and M. Kim. Isomorphic strategy for processor allocation in k-ary n-cube systems. *IEEE Transactions on Computers, Vol. 52, No. 5*, pages 645–657, May 2003.

[16] D. Kostić, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: high bandwidth data dissemination using an overlay mesh. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, volume 37, 5 of *Operating Systems Review*, pages 282–297, New York, Oct. 19–22 2003. ACM Press.

[17] L. Kou, G. Markowsky, and L. Berman. A fast algorithm for Steiner trees. *Acta Informatica*, 15:141–145, 1981.

[18] D. Loguinov, A. Kumar, V. Rai, and S. Ganesh. Graph-theoretic analysis of structured peer-to-peer systems: Routing distances and fault resilience. In *Proceedings of the ACM SIGCOMM '03 Conference*, Karlsruhe, Germany, August 2003.

[19] N.J.A.Harvey, M. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet: A scalable overlay network with practical locality properties. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*, March 2003.

[20] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Computer Communication Review*, volume 31, pages 161–172. Dept. of Elec. Eng. and Comp. Sci., University of California, Berkeley, 2001.

[21] A. Rowstron and P. Druschel. Pastry: scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Nov. 2001.

[22] SETI@home: http://setiathome.ssl.berkeley.edu/.

[23] S. Shi and J. Turner. Routing in overlay multicast networks. In *IEEE INFOCOM*, June 2002.

[24] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, D. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for Internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17–32, Feb. 2003.

[25] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee. How to model an internetwork. In *IEEE INFOCOM*, volume 2, pages 594–602, San Francisco, CA, March 1996.

[26] Y. Zhang, R. West, and X. Qi. A virtual deadline scheduler for window-constrained service guarantees. In *Proceedings of the 25th IEEE Real-Time Systems Symposium (RTSS)*, December 2004.