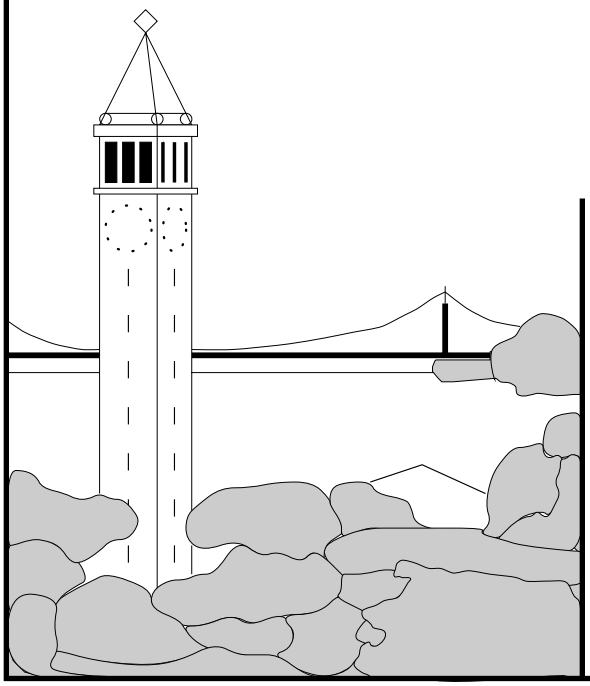
Content-Based Multicast: Comparison of Implementation Options

Ryan Huebsch



Report No. UCB/CSD-03-1229

February 2003

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Content-Based Multicast: Comparison of Implementation Options[1]

Ryan Huebsch

UC Berkeley
huebsch@cs.berkeley.edu

Abstract

This paper is an attempt to quantify the performance differences for content-based multicast implemented inside the overlay routing algorithm or built on top of the simple API provided by the routing layer. We focus on overlay networks designed for peer-to-peer distributed hash table (DHT) applications where content-based multicast is most applicable. In particular we study the Content Addressable Networks (CAN) and Chord routing algorithms. It is our conjecture that similar results would be obtained through other protocols such as Pastry and Tapestry.

We show that it is feasible and in some ways more flexible to provide content-based multicast above the routing layer with only a modest gain in latency.

1 Introduction

Distributed hash tables (DHTs) can be used to distribute, store and retrieve data among many nodes in a network. A basic DHT provides only two simple primitives, `get(key)` and `put(key, value)`. Given a particular key a DHT is able to retrieve the associated value from the proper node. For some applications this is sufficient, i.e. when the application knows the exact key(s) for the item(s) it needs. However, another set of applications [8, 10] require greater search capabilities. More complex queries such as range or relational (SQL-like) queries are difficult, if not impossible, to perform over a basic DHT with just two primitives.

These enhanced queries usually require broadcasting or flooding a request to a group of nodes in the network, which can be inefficient when only using the `put` and `get` primitives. The group of nodes to be contacted is based on the keys they are responsible for storing. In order to provide increased performance for applications that require partial flooding of the content space, an efficient multicast function should be provided.

A DHT is composed of multiple components which are connected through standard interfaces. It is important to consider where specific functionality should be implemented in a layered system. Adding functionality to core components often makes them more complicated, prone to bugs, and may decrease efficiency for more common tasks. In the network community, the end-to-end argument [13] is often used as design principle that argues for pushing most functionality higher in the protocol stack unless it can be correctly (and significantly more efficiently) implemented at the lower level.

This paper is an attempt to quantify the performance differences for content-based multicast implemented in two different layers of the DHT. We use Content Addressable Networks (CAN) [11] and Chord [14] as the routing algorithm for the DHT. It is our conjecture that similar results would be obtained through other protocols such as Pastry [6] and Tapestry [15].

We discuss related work in Section 2, followed by the general architecture in Section 3. Section 4 explains the details of the algorithms we have implemented. Section 5 evaluates the performance differences and we conclude the paper in Section 6.

2 Related Work

Multicast was first introduced as an IP-based solution [5]. Several research projects [3, 4, 7, 9] have argued instead for application level multicast as a more practical alternative to IP multicast, citing the end-to-end argument. One example is Narada [1], which is an overlay network that provides small-scale multicast groups. End systems in Narada self-organize into an efficient mesh structure using a distributed protocol. Source rooted shortest delay spanning trees are then explicitly constructed for the end hosts within a multicast group. As a result of using global routing and tree formation algorithms, systems like Narada do not scale beyond hundreds of nodes nor do they operate efficiently under dynamic conditions.

To ensure better scalability and handling of dynamic groups and network failures, application level multicast has been implemented using peer-to-peer routing algo-

[1]This research was funded in part by NSF/IRIS (`http://project-iris.net/`) project Cooperative Agreement No. ANI-0225660 and in part by a NSF Information and Data Management grant No. IIS-0209108.

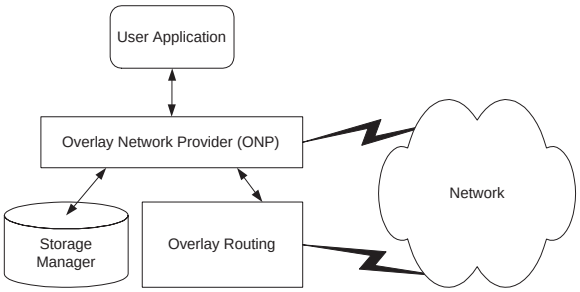


Figure 1: The system model is composed of four components, the user application which interfaces the with overlay network provider (ONP). The ONP manages the storage manager and locates other ONPs through key lookups using routing layer.

rithms. For example, Bayeux [16] is implemented on top of Tapestry, and organizes multicast receivers into a distribution tree routed at the source. In Bayeux, nodes explicitly join and leave a multicast session by notifying the source node. The service model is limited to a single source.

An alternative to Bayuex has been proposed in [12], in which multicast facilities are provided within the routing layer itself. In this proposed solution (called CAN-based multicast), a multicast group forms a "mini" CAN that is reachable from a bootstrapping node in the underlying base CAN. Multicast is then achieved within this mini CAN via directed flooding. Unlike Bayeux, CAN-based multicast does not restrict the service model to a single source.

A major difference between our system, and the above multicast solutions are that our multicast membership is content-based rather than enrollment based. In our service model, group members are implicitly subscribed to a multicast group determined by the content (keys) that they are responsible for storing. The group is sender specified, but the sender does not know which specific nodes will receive the message. This is in contrast to other content-based multicast such as [1, 2] which are based on publish-subscribe model, in which receivers subscribe to multicast groups as determined by content publishers, and receive event notifications whenever the subscribed content changes.

The main contribution of this paper is to evaluate content-based multicast implementations at two different layers: inside the routing algorithm (such as CAN-based multicast) and on top of the routing layer (such as Bayeux).

3 Architecture

Our architecture is based on a design with three main components as shown in Figure 1. In this model the overlay network provider (ONP), routing layer, and the (non persistent) storage manager are commonly grouped together and called a DHT. We focus on the interaction between two components, the ONP and the routing layer. For the purpose of this investigation, storage is ignored and the application can be any user-level application that requires a multicast primitive.

In this model the ONP provides a number of functions to a user application, including `put` and `get` which insert and retrieve data items from the network. The routing layer provides a simple `lookup` function to determine which node is responsible for a particular key. Once the routing algorithm determines the proper node, the ONP is able to directly contact the corresponding ONP on the other node and transfer the data accordingly.

Although the routing layer only provides a simple primitive, it is responsible for maintaining a number of important properties including load balancing and recovery from faults. Routing algorithms have been designed for peer to peer DHT based networks including [11, 14, 6, 15].

Data in the overlay network is uniquely identified by two values, a namespace and resource identifier. These two values are transformed into a *location identifier* by the ONP using a well-known function (usually a hashing function). The location identifier is the search key used with the routing layer. We address the class of functions where the high order bits of the location identifier are based solely on the namespace, while the lower order bits are based on resource identifier (and possibly the namespace as well). This allows data within the same namespace to be assigned to nodes that are close in the logical space (and possibly physically close depending on the routing layer).

The multicast primitive we wish to provide is designed to deliver a message (data) to all nodes in the network that are responsible for some group of content. The group of content is comprised of all the data in a particular namespace. Thus, the multicast message is designed to go to all nodes that are responsible for a particular prefix of location identifiers. It is expected that nodes will usually only send one message at a time (as opposed to streaming data over a period of time) and that any node in the system may send a message to any multicast group.

A multicast message contains the location prefix (corresponding to the namespace), mask, and a payload, which is the data to be delivered to each node. Using the prefix and mask, the lower and upper bounds of the location identifier space can be determined; this range is referred to as the *multicast range*.

Unlike many multicast service models, nodes do not specifically join a multicast group, instead a message is automatically delivered to all nodes in the group.

4 Multicast Algorithms

We examine multicast using two different routing algorithms CAN and Chord. We have implemented two versions of multicast in CAN and two versions in Chord. We then compare these routing layer implementations with a general ONP level multicast, which works on top both routing algorithms (and in general, it will work on top of any routing algorithm).

4.1 CAN Multicast

CAN maps location identifiers to a point in d-dimensional space, where d is a system-wide parameter. The logical space wraps around the edges to create a torus (continuous space).

Each node in CAN is responsible for a zone or region of the logical space. Every node maintains a pointer for each of its neighbors in the logical space. On average each node has $2d$ neighbors, although uneven partitioning of the space can result in fewer or more neighbors for some nodes.

A message is routed through the logical space by using a greedy algorithm. A node will forward the message to whichever neighbor is closest based on Euclidean distance in the logical space. On average $\frac{d}{4}n^{\frac{1}{d}}$ hops are required (where n is the number of nodes in the system) for a message to reach its destination.

A multicast message is sent to a *multicast zone*, described by two coordinates (the lower and upper bounds of the multicast range), in the logical space where all identifiers of interest are mapped. This zone can intersect one or more nodes.

Flooding of the multicast message begins by the message being delivered to any node in the multicast zone. This can be achieved by issuing a lookup for a random identifier within the multicast range. Our system uses the lower bound of the multicast range as the starting point.

The first multicast method implemented is the naïve algorithm. For this algorithm, each node iterates through its neighbors and decides whether to forward the message if the following conditions are met:

1. The neighbor is not the same node that the message came from on the previous hop.

2. The neighbor's zone intersects with the multicast zone

Each node also maintains a list of messages that it has previously received and forwarded. Messages are only forwarded the first time they are received. If this list was lost, due to a failure or error, the correctness would not be affected, however extra messages could be generated.

The second method, referred to as *smart* (compared to the naïve algorithm), is described in [12] where it was first introduced as *directed flooding*. It follows the same basic rules as the naïve method but adds two additional conditions:

1. If the message arrived from a node that abuts this node in the i^{th} dimension, then the message is only forwarded to neighbors who abut on dimensions less than i or in the opposite direction (in dimension i) than from where it was received.

2. The message is not forwarded along a particular dimension if it has traveled more than halfway around the total logical space (not just the multicast zone) in that dimension from the source node.

Figure 2 shows how a sample smart multicast is disseminated.

The smart algorithm is designed to predict whether another node is responsible for sending the multicast message to a particular neighbor based on the direction the multicast message is traveling. Using the smart algorithm, nodes should only receive duplicate messages when the logical space is not divided evenly among the nodes. The naïve method, only prevents forwarding the message to the neighbor it was received from. This means that using the naïve method, nodes (in the worst case) could receive a multicast from each of their neighbors.

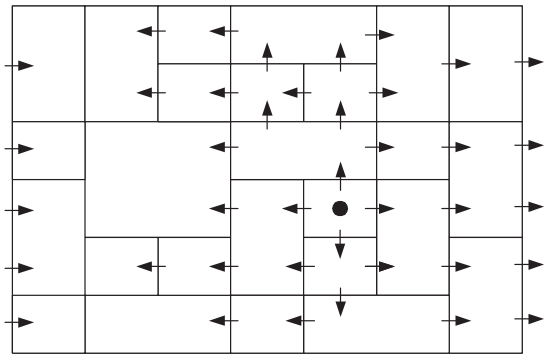


Figure 2: Smart multicast in CAN. The dot represents the starting node.

Both algorithms forward messages only to logical neighbors, which correspond to one logical hop. This provides the maximum efficiency for routing in the overlay network. On average the multicast message could reach all nodes in the group after $\frac{d}{4}n^{\frac{1}{d}}$ hops.

In a peer-to-peer environment, robustness is an important property. Neither algorithm is particularly designed for robustness. The naïve algorithm sends more duplicate messages, therefore the likelihood that at least one of the messages reaches each node is higher than the smart algorithm, increasing the robustness of the naïve algorithm.

4.2 Chord Multicast

Chord's routing mechanism is based on a circle (or a one-dimension logical space). Each node in the network is assigned a point on the circle (also known as its id) and is responsible for any keys that map to the portion of circle before it's id and after the previous node's id (usually visualized as the arc going counter-clockwise around the circle). Routing can be achieved as long as each node knows its predecessor and successor on the circle. For efficiency each node also maintains pointers (known as a fingers) to nodes throughout the circle. A finger is maintained for the node responsible for $id+2i$, for each i such that $0 < i < m$ and m is the number of bits used to specify an identifier.

A message is routed through the system in either an iterative or recursive style. In the iterative style, a node will contact the node with the highest preceding value (compared to the target location identifier) within its finger table. That node will either accept the message (if it is responsible for the target location identifier) or return the message with its best guess of the predecessor based on its finger table. The process repeats till the proper successor node is located. The recursive style is similar except that the intermediate nodes contact the next node directly instead of sending contact information back to the initiator. The

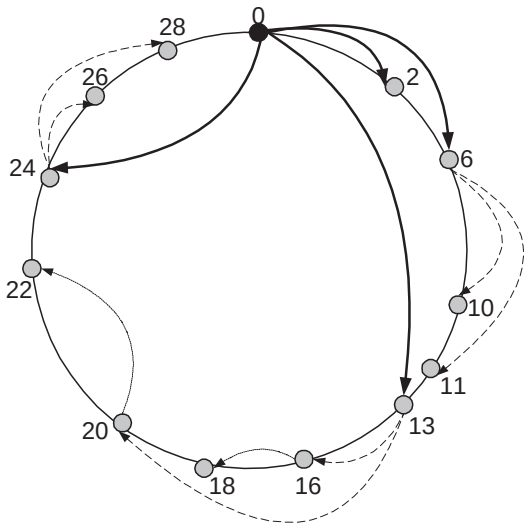


Figure 3: Smart multicast in Chord. Solid lines represent messages sent by the first node, dashed lines show messages sent by those nodes, and finally the dotted lines show the final messages sent. Fingers along which no message was sent are not shown for clarity. Node 0 forwards the message to nodes 2, 6, 13, and 24. Node 6 only forwards the message to nodes that lie from 6 to 13.

recursive style reduces latency by about one half over the iterative style. On average a message requires log(n) hops in the overlay network.

A multicast message in Chord is disseminated in a similar fashion as is done in CAN. The message is first sent to the node at the lower bound of the multicast range. The naïve method will forward the message to every finger in the node's finger table, including the successor that lies in the multicast range. A list of each message forwarded is kept to prevent a node from re-forwarding the same message.

An improved forwarding algorithm, called smart, also sends the message to each node in the finger table, but adjusts the range of multicast before sending to limit the number of duplicate messages. The lower bound of the range is set the remote node's identifier. The upper bound is set to the minimum of the identifier of the next local finger and the multicast range. In effect, the node partitions the multicast range among its fingers. Figure 3 shows an example of the smart distribution.

The same tradeoffs are made with the Chord multicast algorithms as are made with the CAN multicast algorithms. The smart multicast method reduces the total number of messages sent but sacrifices robustness if there are failures or losses. In the average case, in the naïve method a node receives the message from each node that has a finger to that node, or approximately $\log n$ messages. The smart algorithm is designed to produce no duplicate messages since the logical space is partitioned into non-overlapping ranges. The time for distribution is the same for both algorithms and is approximately $\log n$.

4.3 ONP Multicast

The ONP level algorithm is based on a simple spanning tree method, although no tree is explicitly created or saved between messages.

The algorithm will work for any routing algorithm that satisfies one property with respect to the mapping of location identifiers: for any location identifier bit prefix, if the lower bound of the prefix and the upper bound of the prefix map to the same node, so will all identifiers in between. This is not as strict as a requirement saying any continuous range must map to the same node. Since the range is specified as a bit prefix, the bounds of the prefix are powers of two as opposed to any arbitrary value.

Chord naturally satisfies this property since each node has a continuous range (or arc) along the circle. CAN also guarantees this property for any number of dimensions since a zone is defined using a bit prefix.

As with the routing layer level multicast algorithms, the message is first delivered to a node within the multicast range. This node is the root of the tree and processes the message like any other node.

The node creates two messages, one with location prefix extended with a 0 (on the right leaving the high order bits the same), and one message with the location prefix extended with a 1. The mask for each of the messages is reduced by one bit. Figure 4 shows this process for the last two rounds of communication.

Each of the messages is then forwarded to the node that is responsible for the key at the lower end of the new (reduced) range. If the current node is responsible for the lower bound location identifier, then the upper bound location identifier is tested to see if that value is also local. If both values are local, a leaf on the tree has been reached, and the message is discarded. Otherwise the message is processed locally following the same process.

A multicast message will be recursively processed by nodes as the mask is reduced until the base case is reached when the mask is empty and all possible prefixes have been tested or a leaf is reached. Descending to the bottom of the tree should rarely occur since the tree should be relatively sparse (there are more location identifiers than nodes).

The method as described creates a binary spanning tree. A quad or higher degree tree could also be created. Instead of creating two messages and reducing the mask by one bit, the mask can be reduced by more than one bit at each level. The number of messages created at each node becomes 2^x where x is the number of bits removed from the mask.

Because messages are always sent to the low end of the range at each level, the leftmost child is the same node as the parent. In this case, only the high end of the range needs to be checked to determine if the node is a leaf.

The binary tree will have a maximum height of m, where m is the number of bits in the identifier. Consider the case when the routing algorithm maps a single continuous region of the identifier space to a node (as is the case with CAN and Chord), then each node may only appear twice at each level unless it is a leaf. This bounds the worst

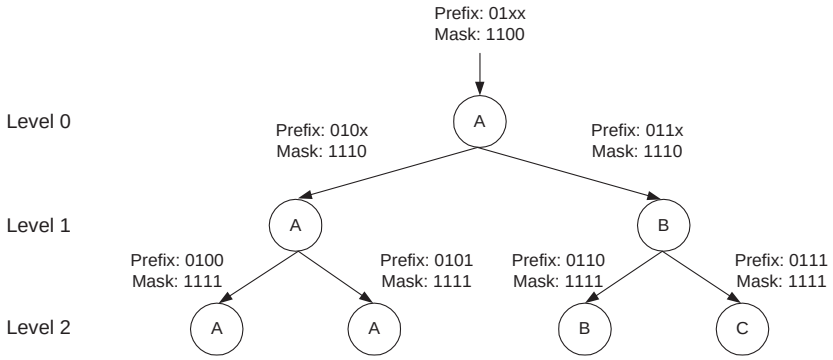


Figure 4: ONP multicast. The message arrives at node A at level 0. At level 1 on the left side of the tree, node A will detect it has reached a leaf, since both its children map to itself.

case fan-out at $2m$ (two messages per level), and the worst case number of duplicates is also $2m$ (up to two messages per level). Nodes that receive messages from themselves (they are their own parent) do not count as duplicate messages since these messages were never transmitted over the network.

The worst case time for dissemination is m times the cost of the average lookup. In the average case, the lookup only requires one hop for each level since the parent and child are usually close in the identifier space. On average the tree will only have $\log n$ levels.

Higher degree trees should increase the speed at which the message is disseminated (since there are fewer levels in the tree); however it also increases the fan-out since each node has more children. Higher degree trees are also likely to produce more duplicate messages. At each level the range for the multicast is divided into smaller ranges. It becomes more likely that two or more of these ranges will map to the same node since the ranges are smaller and the tree should be relatively sparse.

5 Experimental Results

A number of metrics are used to evaluate the multicast methods. The primary metric is the relative delay penalty (RDP). RDP is defined as ratio of the actual delay before a node receives the message and the unicast delay if the source sends the message directly to the recipient on the physical network. The RDP shows the relative speed of the multicast message dissemination

In conventional IP multicast, each node has a RDP of 1, since it is the most efficient distribution and follows the unicast path. Overlay networks generally introduce much higher RDPs since neighbors in the overlay network are not necessarily neighbors in the physical network.

The 50% RDP represents the RDP of the median, or the RDP after 50% of the nodes have received the message. Likewise, 90% RDP represents the RDP after 90% of the nodes have received the message.

The average number of multicasts received by each node is also considered to see how effective each algorithm is in preventing duplicate delivery.

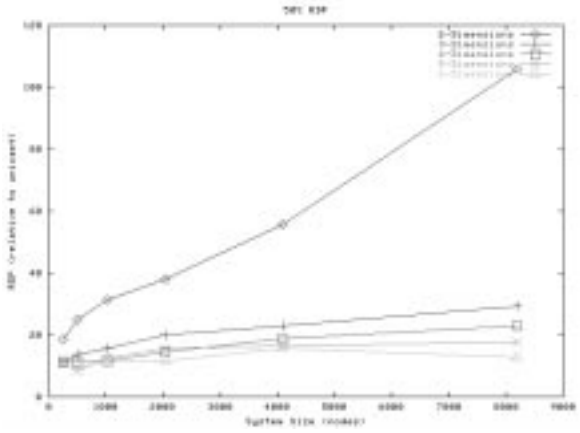


Figure 5: 50% RPD with CAN at various dimensions.

All of our experiments were conducted using a discrete event simulator. The simulator was setup with a basic star topology network. Every node is assigned a half latency and bandwidth. When communicating with another node, the latency of the communication is the sum of the two half latencies. The bandwidth of the communication is the minimum of the two nodes.

For all of our experiments each node has a half latency of 100 ms and a bandwidth of 64 KBps. This results in a 200 ms latency and 64 KBps connection between any two nodes. All flows use UDP with no packet loss. This simple network allows the algorithms to be easily analyzed.

For each test, nodes join the network one after the other with a small delay between joins (500 ms). Each node uses the first node as the landmark to bootstrap into the network. The system is run with no queries, joins, or leaves till the system stabilizes. Experiments using Chord used the iterative style for lookups.

The first set of tests focused on the setting the number of dimensions used for CAN when using the ONP binary-tree multicast. CAN was tested with two through six dimensions with network sizes starting at 256 nodes through 8192 nodes. Figures 5-7 shows the results.

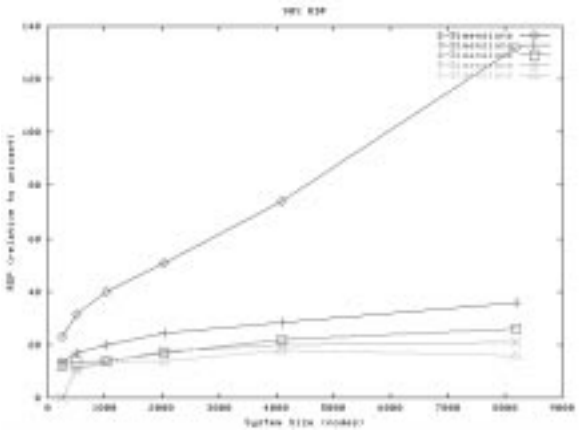


Figure 6: 90% RPD with CAN at various dimensions.

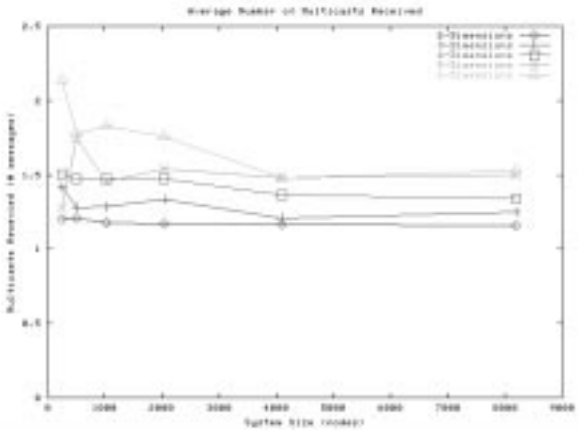


Figure 7: Number of duplicate messages with CAN at various dimensions.

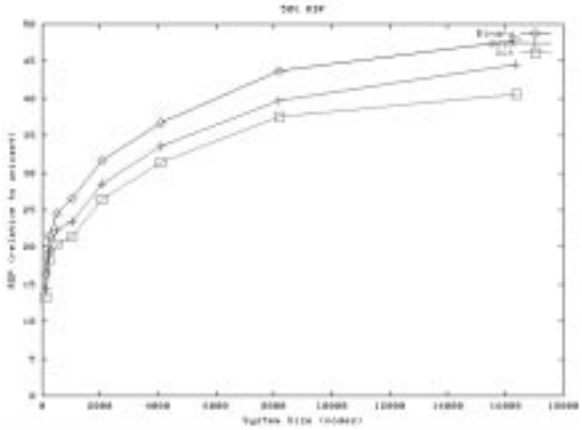


Figure 8: 50% RPD with ONP multicast with varying degree trees.

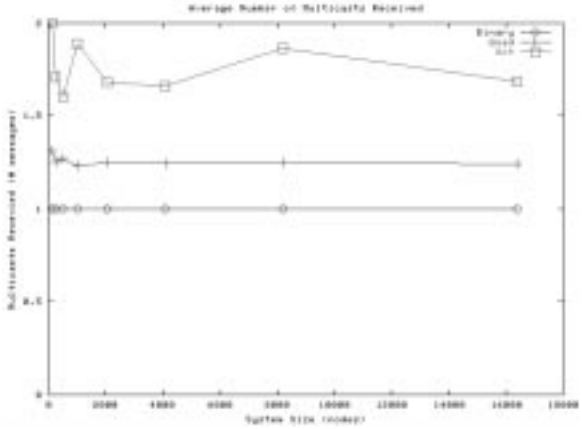


Figure 9: Number of duplicate messages with ONP multicast with varying degree trees.

As the number of dimensions increases, the average path length decreases which corresponds to the decrease in the 50% RDP. The 90% RDP shows the same trend. The average number of messages is slightly higher (more duplicates) with more dimensions. The differences diminish slightly as the size of the network increases. For the remaining tests, the dimensionality of CAN is set to four.

The second set of tests (Figures 8-9) examines the effect of degree of the ONP spanning tree. In this experiment the network size goes up to 16384 nodes. Binary, quad-, and oct-trees are graphed. The 50% RDP graph shows that the higher degree trees are able to disseminate the message faster and scale better with larger systems. The 90% RDP shows the same trends and is not shown. With higher degree trees the average number of messages received also increases, leading to poor use of network resources.

The next set of tests (Figures 10-12) compares the different multicast methods. The 50% and 90% RDP graphs show that the naïve method is slightly faster than the smart multicast. The ONP binary-tree multicast is about twice as

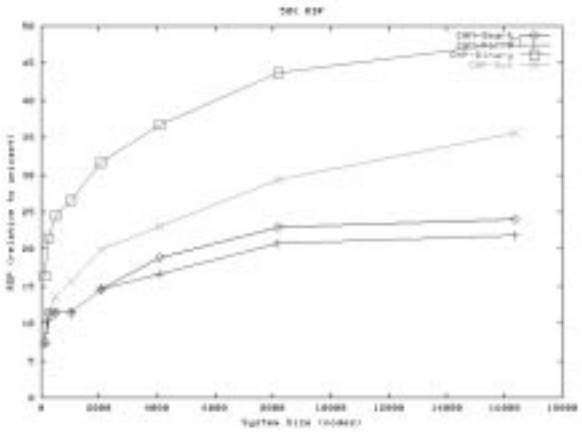


Figure 10: 50% RPD with CAN and ONP multicast implementations.

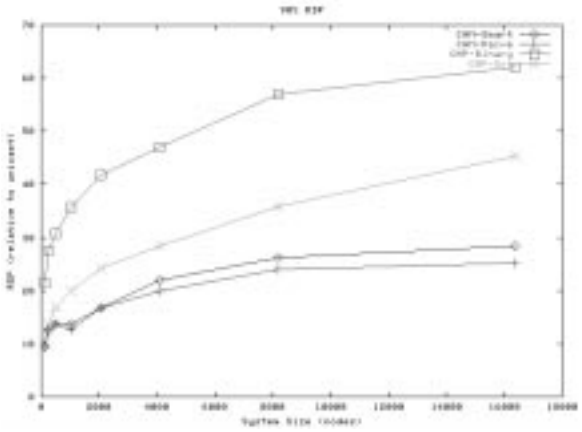


Figure 11: 90% RPD with CAN and ONP multicast implementations.

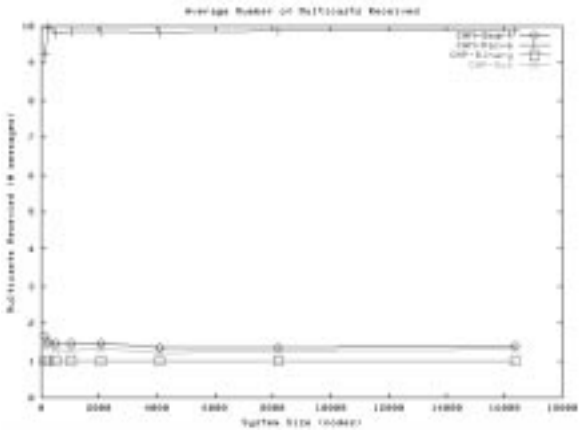


Figure 12: Number of duplicate messages with CAN and ONP multicast implementations.

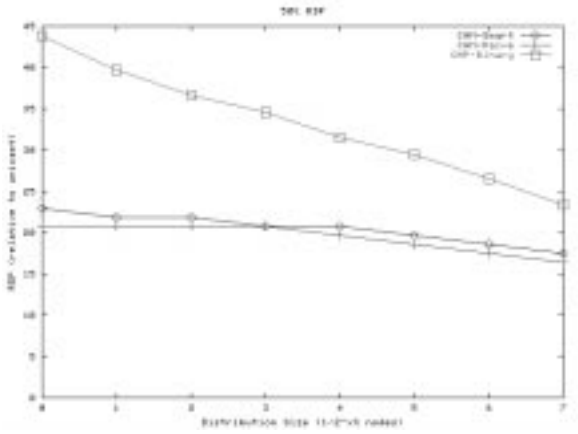


Figure 13: 50% RPD with varying multicast distribution sizes.

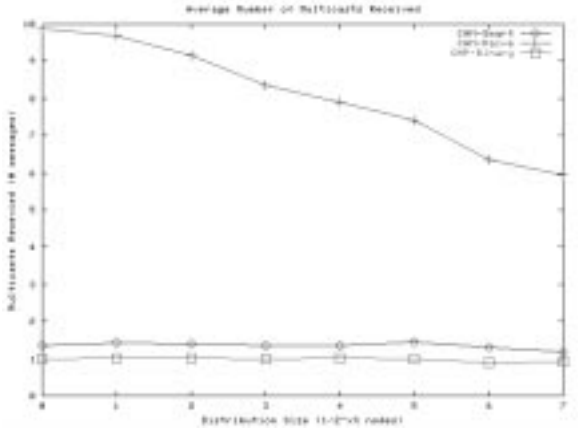


Figure 14: Number of duplicate messages with varying multicast distribution sizes.

slow. The oct-tree approaches the performance of the native CAN algorithms. The data also shows that the ONP binary-tree multicast delivers only one message per node, slightly better than the smart and oct-tree methods. The naïve method, as expected, produces a large number of average messages received per node.

The final set of tests with CAN (Figures 13-14) shows how the various multicast algorithms perform with varying size multicast groups. The number of nodes in the system is fixed at 8192. Along the x-axis is the size of the group (represented as $\frac{1}{2^x}$).

As the group becomes smaller, the ONP binary-tree multicast is able to achieve a linear decrease in RDP. The CAN multicast algorithms improve with smaller groups, but not significantly. This is due to the fact that CAN multicast methods can only follow neighbors and as the group size decreases the message can only be routed to some neighbors. For the naïve algorithm this has the positive side effect of reducing the number of duplicates.

A comparison of the various multicast algorithms in Chord is shown in Figures 15-17. Similar to the CAN re-

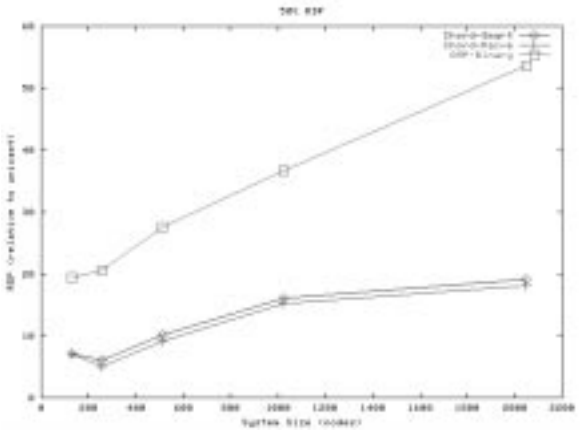


Figure 15: 50% RPD with Chord and ONP multicast implementations.

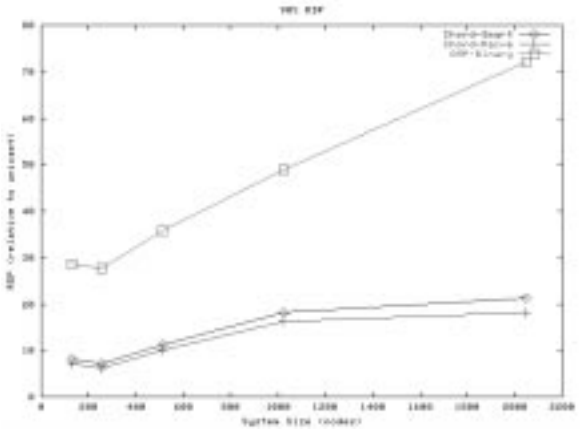


Figure 16: 90% RPD with Chord and ONP multicast implementations.

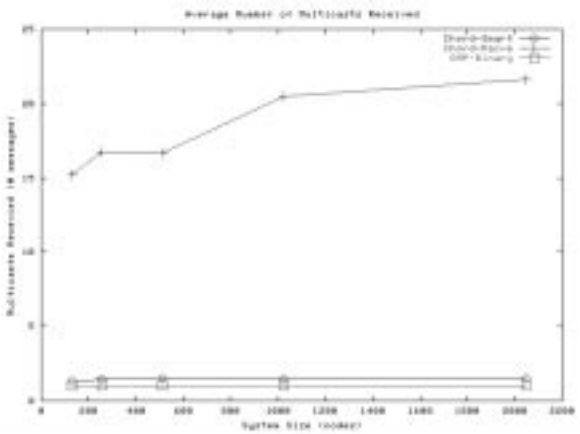


Figure 17: Number of duplicate messages with Chord and ONP multicast implementations.

sults, the ONP binary-tree multicast is about twice as slow as the native Chord algorithms. This is primarily due to the cost of the ONP issuing the lookups through Chord rather than being able to directly transfer the data to the nodes. The naïve and smart algorithms distributed the multicast at about the same pace.

The naïve Chord algorithm generated a significant number of duplicates. This is due to the fact that the message is forwarded to all fingers without partitioning the logical space among them. Again, both the smart Chord algorithm and ONP binary-tree multicast both achieved near perfect distribution producing few duplicates.

6 Conclusion

The various multicast methods performed as expected resulting in the traditional tradeoff. By moving the multicast function lower in the stack, a substantial performance benefit is realized. However this adds additional complexity to the routing layer. The same tradeoff is made between IP Multicast and application level multicast implementations.

An important result from the tests is to realize that a general ONP level multicast is feasible and can perform reasonably well. The ONP multicast is able to keep the number of duplicates very low, usually better than routing layer level multicasts. The ONP multicast also has the added ability to make the tradeoff between fast distribution and number of duplicate messages. This tradeoff can be made on a message-by-message basis by the user application. This shows the added flexibility of an ONP level implementation.

For applications that require very high performance multicast, implementation at the lower level is better. However to provide a general multicast on top of many different routing algorithms, the ONP multicast is a fair compromise and should be considered a viable alternative for most systems.

7 Acknowledgements

Our thanks to Ion Stoica, Boon Thau Loo, Scott Shenker, and Joe Hellerstein for their input and support of this project. We would also like to thank Sylvia Ratnasamy for assistance with CAN.

References

[1] G. Banavar, M. Chandra, B. Nagarajaro, R. Strom, and C. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In *Proceedings of ICDCS*, 1998.

[2] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized publish-subscribe infrastructure, 2001.

[3] Y. Chawathe, S. McCane, and E. Brewer. An architecture for internet content distribution as an infrastructure service, Feb. 2000.

[4] Y.-H. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *Proceedings of ACM SIGMETRICS*, Santa Clara, CA, June 2000.

[5] S. Deering and D. Cheriton. Multicast routing in internetworks and extended lans. In *Proceedings of ACM SIGCOMM*, 1998.

[6] P. Druschel and A. Rowstron. Past: Persistent and anonymous storage in a peer-to-peer networking environment. In *Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems*, 2001.

[7] P. Francis. Yoid: Extending the internet multicast architecture, 2000.

[8] M. Harren, J. Hellerstein, R. Huebsch, B. Loo, S. Shenker, and I. Stoica. Complex queries in dht-based peer-to-peer networks. In *Proceedings of 1st International Workshop on Peer-to-Peer Systems*, Cambridge, USA, Mar. 2002.

[9] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole, Jr. Overcast: Reliable multicasting with an overlay network. In *In Proceedings of OSDI*, 2000.

[10] J. Li, B. T. Loo, J. Hellerstein, F. Kaasheok, D. Karger, and R. Morris. On the Feasibilty of Peer-to-Peer Web Indexing and Search. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, Feb 2003.

[11] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proceedings of ACM SIGCOMM*, 2001.

[12] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level multicast using content-addressable networks. *Lecture Notes in Computer Science*, 2233, 2001.

[13] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4), Nov. 1984.

[14] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM*, 2001.

[15] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, Apr. 2001.

[16] S. Zhuang, B. Zhao, A. Joseph, R. Katz, and J. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant widearea data dissemination, 2001.