# Unexpected Complexity: Experiences Tuning and Extending CAN

Michael B. Jones, Marvin Theimer,
Helen Wang, Alec Wolman

December 2002

Technical Report
MSR-TR-2002-118

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

# Unexpected Complexity: Experiences Tuning and Extending CAN

Michael B. Jones, Marvin Theimer, Helen Wang, Alec Wolman

*Microsoft Research, Microsoft Corporation*
*One Microsoft Way*
*Redmond, WA 98052*

{mbj, theimer, helenw, alecw}@microsoft.com

## Abstract

As part of evaluating options for the design and implementation of a scalable application-level multicast system, we produced an independent implementation of CAN, experimented with tuning it, and also extended it to improve its performance and in some cases, correctness. We were able to reproduce most of the results in the original CAN paper, providing independent validation of their results. However, we encountered far more complexity with tuning and extending CAN than anticipated. The best set of parameter choices appeared to be highly scenario-specific and non-intuitive, at least for some of the parameters supported. Tuning CAN involved a lot of trial-and-error navigation through the CAN parameter space. Furthermore, the "anycast" semantics provided by CAN turned out to make our application design more complex than was originally anticipated.

## 1. Introduction

The Content Addressable Network (CAN) [Ratnasamy et al. 01a] is a self-organizing, peer-to-peer overlay network that can be used to build scalable distributed applications. Other such overlay networks currently in use include Chord [Stoica et al. 01], Tapestry [Zhao et al. 01], and Pastry [Rowstron and Druschel 01].

Scalable application-level multicast is a function that can take advantage of the capabilities provided by such overlay networks. As part of evaluating options for the design and implementation of a scalable application-level multicast system we experimented with both CAN and Pastry, with results reported in [Castro et al. 03]. This paper reports on the lessons we learned from trying to tune CAN for our application as well as improve its performance and, in some cases, correctness.

One of our earliest steps was to produce an independent implementation of CAN. To validate our implementation we then set about reproducing the results presented in the original CAN paper. In nearly all cases we were able to reproduce these results within a few percent of the original values, thereby providing independent validation of their results.

We then set about tuning CAN for our multicast application. Our original validation efforts involved tuning CAN for simple unicast message-passing and we had intended to use the resultant parameter settings as a starting point for tuning two different multicast designs. Unfortunately these settings did not provide much help, as it turned out that the best set of parameter choices appeared to be highly scenario-specific and changes in parameter settings resulted in non-linear performance effects.

Along the way we discovered and fixed a bug in the published CAN multicast flooding algorithm [Ratnasamy et al. 01b] and also extended the design with three additional features in an effort to increase CAN's performance. Two of the three features, *network-based routing metric* and *transit-stub topological node placement*, ended up improving performance while one, *corner neighbors* did not.

An unexpected complication we encountered concerned itself with the fact that some of CAN's more important parameter settings, *multiple nodes per zone* and *multiple realities*, cause it to deliver messages in anycast-style to any of a set of eligible nodes, rather than to a single destination node. As a result, we ended up having to add an extra layer of distributed synchronization to our application.

The net result of our efforts and experiments was that we were able to validate the overall performance claims that have been published in the prior literature but that we encountered an unexpectedly large amount of complexity while trying to tune CAN for our application as well as when extending it to improve its performance and correctness. The remainder of this paper briefly summarizes the CAN design, describes the new features we added to it, describes a bug in the published CAN flooding algorithm and our fix for the bug, and our experiences with trying to tune and use CAN.

## 2. CAN Design

### 2.1 Basic Design

The Content Addressable Network (CAN) [Ratnasamy et al. 01a] overlay network design organizes nodes of an overlay into a *d*-dimensional hypercube. Each node takes ownership of a specific hyper-rectangle in the space, such that the collection of hyper-rectangles covers the entire space. Each node tracks who its immediately adjacent neighbors are and routes messages to them. Nodes join the hypercube by routing a join message to a randomly chosen point in the space, causing the node owning that region of space to split its region into two, giving half to the new node and retaining half for itself. Message routing consists of choosing one of the current node's neighbors closer in CAN space to the destination than the current node is, and forwarding the message to that neighbor, repeating this process until the message reaches the node whose region contains the destination address.

### 2.2 Previously Published CAN Features

Beyond the basic CAN algorithm, described above, CAN adds a number of "knobs" that can be used to improve its routing performance. While these were described in [Ratnasamy et al. 01a], we summarize them here:

**Dimensions:** The number of dimensions of the CAN hypercube.

**Ratio-Based Routing:** Vanilla CAN routes to the neighbor closest to the destination in CAN space. Ratio-based routing examines the ratio between the network delay to each neighbor and the progress made in CAN space by routing to that neighbor, choosing to route to the neighbor with the best ratio of CAN distance progress to network cost.

**Multiple Nodes per Zone:** This knob allows more than one node to inhabit the same hyper-rectangle. CAN delivers messages to any one of the zone inhabitants in an anycast manner.

**Multiple Realities:** This knob allows multiple CAN hypercubes to co-exist at once, with the same nodes occupying each, but with completely different assignments of hyper-rectangles to nodes in each. Messages can switch between realities at each hop. Messages are delivered to a zone containing the destination CAN address in any one of the realities.

**Uniform Partitioning:** If enabled, when a node joins the CAN network, once its join message reaches a node containing its target CAN address, rather than immediately splitting the region in two, all the node's neighbors are examined. If a neighbor's zone is larger than the current zone, the join message is forwarded to the neighbor, which will then apply the same test. Once a local maximum neighbor size is reached that zone is split in two, with the new node obtaining half the split zone.

**Landmark-Based Placement:** Landmark-based placement causes nodes, at join time, to probe a set of well known "landmark hosts", estimating each of their network distances. Each node measures its round-trip-time to the landmark machines, and orders the landmarks from the nearest to the most distant in the underlying network. Nodes with the same landmark ordering are clustered into a bin. Rather than choosing a random CAN addresses at which to join, the CAN space is divided into evenly sized bins, and the CAN join address is then chosen from within the bin area. The effect is that nodes with the same landmark ordering end up closer to each other in CAN space.

### 2.3 New CAN Features

In the course of our investigations we developed several additional knobs beyond those previously published in an attempt to further improve the achievable CAN performance along several dimensions. We summarize them here:

**Network-Based Routing Metric:** Network-based routing chooses to route a message to the neighbor with least network cost, subject to the message still being closer to the destination. This is like the previously-published Ratio-Based Routing except that only network cost is factored into the routing decisions.

**Corner Neighbors:** We extended the CAN implementation to support routing through corner neighbors. Our motivation for adding corner neighbors was to increase the number of available routing choices. In a traditional CAN, a node is only considered a neighbor if the coordinate spans overlap along *d*-1 dimensions and are adjacent along 1 dimension. With our extension, a node will be considered a corner neighbor if the coordinate spans are adjacent along 2 or more dimensions, and overlap along all remaining dimensions. In CANs with a large number of dimensions, the number of corner neighbors has the potential to grow extremely large. To offset this effect, we implement an option where nodes can randomly select a fixed number of corner neighbors from the set of possible corner neighbors.

**Transit-stub Topological Node Placement:** The basic CAN construction is ignorant of the under-

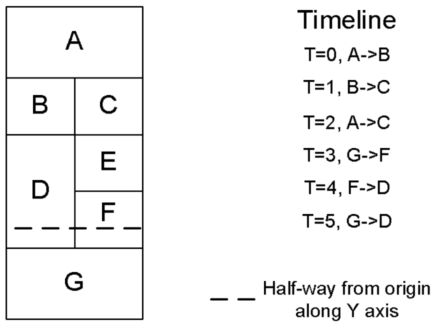

**Figure 1:** Illustration of the race condition that affects the CAN efficient flooding algorithm. If the timing of messages follows the timeline specified in this figure, then Node *E* never receives the flooded message.

lying network topology, namely, two adjacent nodes in the CAN coordinate space may be far from each other in terms of the IP network distance. One of our key interests was to understand how much performance benefit there is to constructing overlays using network topology information. Consequently we wanted to see how alternative assignment strategies would perform. In addition to the landmark-based placement described above, we constructed a placement technique based on the Georgia-Tech transit-stub Internet topology model supported by our simulator [Zegura et al. 96]. Here, we divide the CAN space into *T* equal sized bins, where *T* is the total number of transit networks in the topology. Within a given bin, we randomly choose CAN addresses for each stub network that attaches to the corresponding transit network. This ensures that all stubs that connect to the same transit network will be relatively near each other in CAN space. Furthermore, nodes connected to the same stub network will also end up close to each other in CAN space.

## 2.4 CAN Flooding as Previously Published

The multicast algorithm we implemented for CAN is based on the efficient flooding algorithm described in [Ratnasamy et al. 01b], with some significant modifications. We begin by summarizing the published algorithm, and then we present our modifications.

The naive approach to implement flooding for a CAN overlay network is for each node that receives a message to forward that message to all of its neighbors. Nodes filter out duplicate messages by maintaining a cache of previously received message-ids. The problem with the naive strategy is that it can lead to a large number of duplicate messages. To reduce the number of duplicates, the [Ratnasamy et al. 01b] study presents an efficient flooding algorithm that exploits the structure of the CAN coordinate space to limit the directions in which each node will forward messages. Nodes use the following five rules to decide whether to forward a message, and to decide to which neighbors to forward the message.

1. *Origin Forwarding Rule*: The multicast origin node forwards the message to all neighbors.
2. *General Forwarding Rule*: A node receives a message from a neighboring node adjacent along dimension *i*. The node forwards that message to all adjacent neighbors along dimensions 1 through *i*-1. The node also forwards the message to those adjacent neighbors along dimension *i* in the opposite direction from where it received the message.
3. *Duplicate Filter Rule*: A node caches the message-ids of all received messages. When a node receives a duplicate, it does not forward the message.
4. *Half-Way Filter Rule*: A node does not forward a message along a particular dimension if that message has already traveled at least half-way across the space from the origin coordinate in that dimension.
5. *Corner Filter Rule*: Along the lowest dimension (dimension 1), a node *N* only forwards to a neighbor *A* if a specific corner of *A* ($C_A$) is in contact with *N*. $C_A$ is defined to be the corner of *A* that is adjacent to *N* along dimension 1 and has the lowest coordinates along all other dimensions. Note that this rule eliminates certain messages that would otherwise be sent according to the two forwarding rules.

## 2.5 Improvements to CAN Flooding

We discovered and fixed two flaws with the above algorithm. The first flaw is an ambiguity in the half-way filter rule specified above. The authors state that the above algorithm ensures there will be no duplicate messages if the CAN coordinate space is evenly partitioned (i.e. all CAN nodes have equal sized zones). The following change to the half-way filter rule is needed to ensure that this property actually holds. When deciding whether or not to forward to a neighbor *N*, if *N* contains the point that is half-way across the space from the source coordinate in that dimension, then we only forward to *N* that neighbor from the positive direction.

The second flaw we discovered is a race condition that can lead to certain nodes never receiving the flooded message. This race condition arises because when a node receives a duplicate message, it does not forward that message. Therefore, the order in which a node receives a message from its neighbors may determine the directions in which that message is forwarded. To demonstrate this problem, Figure 1 illustrates a situation where one of the nodes does not receive the multicast message. This figure shows a small portion of a 2-dimensional CAN, where the dashed line in the figure is the location along the *y* axis that is half-way from the origin. The sequence of message delivery times listed in the timeline portion of Figure 1 causes node *E* to never receive the message. Note that a different ordering of message reception either at node *C* or at node *D* would have led to proper message delivery at node *E*. For example, if we switch the order of messages at times *T*=1 and *T*=2, then the message from *A* to *C* is delivered before the message from *B* to *C*, which means that node *C* will forward the message to *E*.

The idea behind our fix to the flooding algorithm is to make static forwarding decisions based on the relative position of a node to the multicast origin, rather than dynamic forwarding decisions based on the order of incoming messages. The new algorithm breaks up the forwarding process into two stages. In first stage, a node decides which dimensions and directions to the forward message along. In the second stage, a node applies a second set of rules to filter the subset of neighbors that satisfy the first stage rules.

The stage one forwarding rules are:

1. If a node's region overlaps the origin along all dimensions less than or equal to *i*, then this node will forward the message in both the positive and the negative directions along dimension *i*.
2. If a node's region overlaps the origin along all dimensions less than *i*, then this node will forward the message only in one direction along dimension *i*. The direction to forward the message will be away from the origin coordinate, towards the half-way point.
3. For the lowest dimension (dimension 1), always forward only in one direction. As before, the forwarding direction will be away from the origin coordinate, towards the half-way point.

The stage two filtering rules are:

1. For all dimensions greater than 1, only forward to a neighboring node along dimension *i* if that neighbor's region overlaps the origin coordinates for all dimensions less than *i*.
2. The half-way filter rule from the original algorithm, with our modification described above.
3. The corner filter rule from the original algorithm.

Although the rules for this algorithm look somewhat different from the original algorithm, the way that messages flow through the CAN coordinate space is quite similar to the original algorithm. An important side effect of the modified flooding algorithm is a significant reduction in the number of duplicate messages, due to the first rule in the filtering stage of the new algorithm.

## 3. Experiments and Results

### 3.1 Experimental Context and Methodology

We built an independent implementation of CAN in a discrete-event simulator provided by the Pastry authors. This work was part of our effort to evaluate the effectiveness of building scalable application-level multicast using overlay networks [Castro et al. 03]. To the best of our knowledge, that study conducted the first head-to-head comparison of CAN-style versus Pastry-style overlay networks, using multicast communication workloads running on an identical simulation infrastructure.

In this paper, we report results from two different sets of simulations. The first set of simulations uses a workload that consists of randomly selected nodes in the overlay performing unicast RPC-style communication patterns. The second set of simulations was performed as part of the aforementioned multicast study, and the communication workloads consist of both the CAN efficient flooding algorithm [Ratnasamy et al. 01b] and the Scribe tree-based multicast algorithm [Rowstron et al. 01] running on top of CAN.

Our simulations ran on a network topology with 5050 routers. We used a random graph generated with the Georgia Tech transit-stub Internet topology model [Zegura et al. 96]. Our simulator models latency for each of the network links, and queuing delay at each of the application-level overlay hops. It does not model queuing delay at the internal routers or packet losses because modeling these would prevent simulation of large networks.

To evaluate performance, our primary metrics are relative delay penalty (RDP) [Chu et al. 00] and neighbor state overhead. RDP is the ratio of the average delay across the overlay network and the average delay across the underlying IP network. Neighbor state overhead is a measure of the number of neighbors that each overlay node communicates with, averaged across all nodes that participate in the overlay. This measure is important because

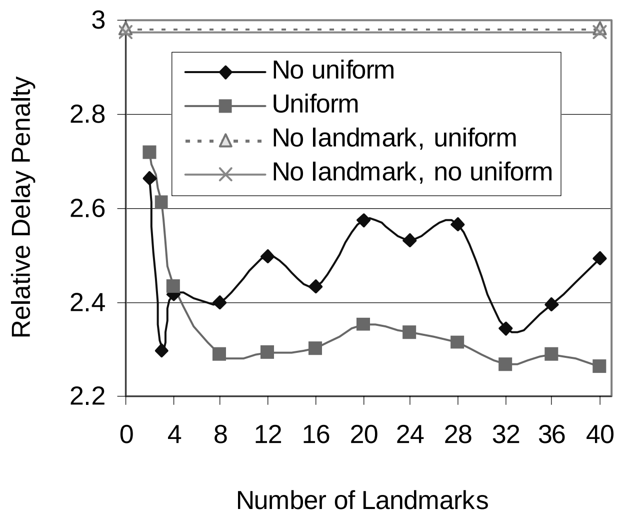

**Figure 2:** The impact of the landmarks on RDP for 10,000 nodes, 10 dimensions, and 5 nodes per zone.

neighbors must communicate on a regular basis to maintain the structure of the overlay and to implement estimates of the network delay between nodes.

## 3.2 CAN Parameter Complexity

CAN has a very large number of parameters that can be used to tune its performance. We performed an extensive exploration of this parameter space, attempting to understand which combinations of parameters lead to the best RDP values for unicast communication. The intent was to use the best parameter combination as a starting point for tuning our application-level multicast system.

We varied the following parameters during our exploration: the number of dimensions; the number of nodes per zone; the number of realities; enabling/disabling uniform partitioning; enabling/disabling routing through corners; the maximum number of corner neighbors considered; choosing a node assignment policy from among random, topological, or landmark-based; and choosing a routing policy from among CAN distance, CAN ratio, or network distance. To allow direct comparisons between different CAN configurations, we measured the average amount of neighbor state that each node maintains and only compared instances of CAN that used similar amounts of neighbor state.

Our primary conclusion is that the CAN parameter space is difficult to navigate. Configurations that work well with one state size budget do not scale to a different one in a linear fashion. Therefore, discovering the best configuration with 50 neighbors does not provide immediate insight into which parameters will give the best configuration with 100 neighbors. To illustrate the difficulties encountered in exploring the CAN parameter space, Figure 2 shows how varying a single parameter, the number of landmarks, can lead to interesting non-linear behavior in terms of RDP.

The difficulty of tuning CAN also manifested itself in our application-level multicast work. The best choice of parameters we were able to find from our unicast experiments did not prove particularly useful for our multicast experiments, forcing us to re-explore the parameter space to tune multicast performance. Table 1 lists a set of representative configurations yielding lowest RDP values for an 80,000 node CAN multicast system at a variety of different state overheads. Note the unpredictable variation of even the most basic CAN parameter, namely dimensionality of the hyperspace.

| State | Dimensions ($d$) | Nodes Per Zone ($z$) | Realities | Uniform Partitioning |
|---|---|---|---|---|
| 18 | 10 | 1 | 1 | enabled |
| 29 | 9 | 2 | 1 | enabled |
| 38 | 12 | 3 | 1 | enabled |
| 59 | 10 | 5 | 1 | enabled |
| 111 | 8 | 10 | 1 | enabled |

**Table 1:** Representative good configurations, as a function of neighbor state, for an 80,000 node CAN.

Nonetheless, we can provide some general guidelines, at least for the workloads that we have explored. Enabling some form of topological assignment (either landmark-based or transit-stub-based) provided the largest improvement in RDP out of all the CAN parameters. For example, a typical transit-stub-based placement configuration saw a 32% improvement in RDP and an 11% decrease in neighbor state compared to a random node placement configuration. We found that transit-stub-based node placement performed comparably to landmark-based node placement once sufficient numbers of landmarks were employed to be effective.

Enabling uniform partitioning often provides a significant reduction in terms of the neighbor state overhead, especially for the two topological assignment schemes where nodes often end up clustered close together in the CAN space. Furthermore, uniform partitioning never causes RDP to become significantly worse.

The network distance routing metric appears to perform consistently better than the other routing metrics. For a typical configuration, network-based routing provides a 24% improvement in RDP over ordinary CAN routing, and a 15% improvement over ratio-based routing.

Both corner routing and multiple realities don't seem to improve RDP significantly without large increases in neighbor state overhead. For example, when we limit the number of corner neighbors considered so that it requires only a moderate amount of state, the improvement in RDP is extremely small. For example, we see a 0.1% improvement in RDP versus a 27% increase in neighbor state.

### 3.3 Topologically Aware Node Assignment Complexity

To implement both landmark-based placement and transit-stub topological placement, one must devise a technique for dividing the CAN coordinate space evenly into a fixed number of bins, where the number of bins is not known in advance. A desirable property for these bins is that the length in each dimension be comparable (e.g. to avoid long and thin rectangles).

Our initial binning approach frequently lead to the creation of star topologies – a single node occupying a large portion of the CAN space with a very large number of neighbors (each of which occupies a small portion of the CAN space). This sort of topology defeats the purpose of constructing the overlay in the first place. A combination of refinements to our binning algorithm and enabling uniform partitioning were critical to reducing the magnitude of this effect, although we were unable to completely eliminate the problem.

### 3.4 Application Programming Model Complexity

An important distinction to understand between CAN and overlay networks such as Pastry, Chord, and Tapestry is that they deliver a message tagged with a given key to exactly that single node whose node ID is numerically closest to the key in the namespace. In contrast, CAN delivers such a message to any node in the overlay that belongs to the same zone that the key is contained by in some reality. Thus, CAN provides "anycast" semantics whereas its competitors provide unicast semantics. The provision of anycast instead of unicast semantics has significant implications for the applications that use the overlay network since they are now responsible for maintaining consistency between state maintained on all nodes belonging to the same anycast equivalence class.

Many of the original peer-to-peer papers used distributed hash tables storing immutable content as a driving application. For this application, anycast delivery semantics are fine. In contrast, when using application-level multicast as a driving application we observed that CAN imposed a significant semantic burden on the application.

## 4. Conclusion

The primary contributions of this work are twofold: we have provided an independent validation of the originally published CAN results and we have learned about some of the difficulties of using CAN as a distributed systems building block. Along the way we both extended the CAN design and fixed a few design bugs. Our validation is based on having done a completely separate implementation of CAN, using a different simulator for testing CAN, and creating a separate set of experiments for testing and using CAN. Our experience with CAN is based on having both performed unicast messaging experiments as well as building two different kinds of application-level multicast.

While gratified that CAN seems capable of achieving the performance and cost levels for which it was designed we were surprised by the amount of complexity we encountered when using and tuning it. In particular:

- We found that changing tuning parameters did not yield easily predictable changes in CAN behavior.
- Several of the tuning parameters force anycast communication semantics on the application. While this may be quite suitable for some applications it may add a non-trivial burden to the design, implementation, and behavior of others.
- Several of CAN's features had significant complexity hidden inside their design. In particular, we found a subtle bug in CAN's broadcast flooding algorithm and had considerable difficulty and only partial success in getting topologically aware node assignment to work as intended.

## Acknowledgments

We thank Sylvia Ratnasamy for answering our many questions concerning CAN, and for the topology files used in their experiments. Without her help we could not have validated our CAN implementation. We likewise thank Antony Rowstron, Miguel Castro, and Anne-Marie Kermarrec for allowing us to use their network simulator.

## References

[Castro et al. 03] Miguel Castro, Michael B. Jones, Anne-Marie Kermarrec, Antony Rowstron, Marvin Theimer, Helen Wang, Alec Wolman. An Evaluation of Scalable Application-Level Multicast Built Using Peer-To-Peer Overlay Networks. In *Proceedings of IEEE INFOCOM*, San Francisco, CA, April 2003.

[Chu et al. 00] Yang-hua Chu, Sanjay G. Rao and Hui Zhang. A Case for End System Multicast. In *Proceedings of ACM SIGMETRICS*, Santa Clara, CA, pp. 1-12, June 2000.

[Ratnasamy et al. 01a] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content-Addressable Network. In *Proceedings of ACM SIGCOMM*, San Diego, CA, pp. 161-172. August 2001.

[Ratnasamy et al. 01b] Sylvia Ratnasamy, Mark Handley, Richard Karp, and Scott Shenker. Application-level Multicast using Content-Addressable Networks. In *Proceedings of Third International Workshop on Networked Group Communication*, UCL, London, UK, November 2001.

[Rowstron and Druschel 01] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of IFIP/ACM Middleware 2001*, Heidelberg, Germany, November 2001.

[Rowstron et al. 01] Antony Rowstron, Anne-Marie Kermarrec, Miguel Castro, and Peter Druschel. SCRIBE: The design of a large-scale event notification infrastructure. In *Proceedings of Third International Workshop on Networked Group Communication*, UCL, London, UK, November 2001.

[Stoica et al. 01] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In Proceedings of ACM SIGCOMM, San Diego, CA, pp. 149-160. August 2001.

[Zegura et al. 96] Ellen W. Zegura, Kenneth L. Calvert, and Samrat Bhattacharjee. How to Model an Internetwork. In *Proceedings of IEEE Infocom '96*, San Francisco, CA, April 1996.

[Zhao et al. 01] Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. U. C. Berkeley Technical Report UCB/CSD-01-1141, April, 2001.