Application-Level Multicast using Content-Addressable Networks

Sylvia Ratnasamy[1,2], Mark Handley[2], Richard Karp[1,2], and Scott Shenker[2]

[1] University of California, Berkeley, CA, USA
[2] AT&T Center for Internet Research at ICSI

Abstract. Most currently proposed solutions to application-level multicast organize the group members into an application-level mesh over which a Distance-Vector routing protocol, or a similar algorithm, is used to construct source-rooted distribution trees. The use of a global routing protocol limits the scalability of these systems. Other proposed solutions that scale to larger numbers of receivers do so by restricting the multicast service model to be single-sourced. In this paper, we propose an application-level multicast scheme capable of scaling to large group sizes without restricting the service model to a single source. Our scheme builds on recent work on Content-Addressable Networks (CANs). Extending the CAN framework to support multicast comes at trivial additional cost and, because of the structured nature of CAN topologies, obviates the need for a multicast routing algorithm. Given the deployment of a distributed infrastructure such as a CAN, we believe our CAN-based multicast scheme offers the dual advantages of simplicity and scalability.

1 Introduction

Several recent research projects[8, 10, 7] propose designs for application-level networks wherein nodes are structured in some well-defined manner. A Content-Addressable Networks (CANs)[6] is one such system. Briefly,[1] a Content-Addressable Network is an application-level network whose constituent nodes can be thought of as forming a virtual d-dimensional Cartesian coordinate space. Every node in a CAN "owns" a portion of the total space. For example, Figure 1 shows a 2-dimensional CAN occupied by 5 nodes. A CAN, as described in [6], is scalable, fault-tolerant and completely distributed. Such CANs are useful for a range of distributed applications and services. For example, in [6] we focus on the use of a CAN to provide hash table-like functionality on Internet-like scales – a function useful for indexing in peer-to-peer applications, large-scale storage management systems, the construction of wide-area name resolution services and so forth.

This paper looks into the question of how the deployment of such CAN-like distributed infrastructures might be utilized to support multicast services and applications. We outline the design of an application-level multicast scheme built using a CAN. Our design shows that extending the CAN framework to support multicast comes at trivial additional cost in terms of complexity and added protocol mechanism. A key feature

[1] Section 2 describes the CAN design in some detail

of our scheme is that because we exploit the well-defined structured nature of CAN topologies (i.e. the virtual coordinate space) we can eliminate the need for a multicast routing algorithm to construct distribution trees. This allows our CAN-based multicast scheme to scale to large group sizes. While our design is in the context of CANs in particular, we believe our technique of exploiting the structure of these systems should be applicable to the Chord [8], Pastry [7] and Tapestry [10] designs.

In previous work, several research proposals have argued for *application-level* multicast[1, 3, 4] as a more tractable alternative to a network-level multicast service and have described designs for such a service and its applications. The majority of these proposed solutions (for example [1, 4]) typically involve having the members of a multicast group self-organize into an essentially random application-level mesh topology over which a traditional multicast routing algorithm such as DVMRP [2] is used to construct distribution trees rooted at each possible traffic source. Such routing algorithms require every node to maintain state for every other node in the topology. Hence, although these proposed solutions are well suited to their targeted applications,[2] their use of a global routing algorithm limits their ability to scale to large (many thousands of nodes) group sizes and to operate under conditions of dynamic group membership.

Bayeux[11] is an application-level multicast scheme that scales to large group sizes but restricts the service model to a single source. In contrast to the above schemes, CAN-based multicast can scale to large group sizes without restricting the service model to a single source.

In summary, we believe our CAN-based multicast scheme offers two key advantages:

- CAN-based multicast can scale to very large (*i.e.* many thousands of nodes and higher) group sizes without restricting the service model to a single-source. To the best of our knowledge, no currently proposed application-level multicast scheme can operate in this regime.
- Assuming the deployment of a CAN-like infrastructure, CAN-based multicast is trivially simple to achieve. This is not to suggest that CAN-based multicast by itself is either simpler or more complex than other proposed solutions to application-level multicast. Rather, our point is that CANs can serve as a building block in a range of Internet applications and services and that one such, easily achievable, service is application-level multicast.

The remainder of this paper is organized as follows: Section 2 reviews the design and operation of a CAN. We describe the design of a CAN-based multicast service in Section 3 and evaluate this design through simulation in Section 4. Finally, we discuss related work in Section 5 and conclude.

2 Content-Addressable Networks

In this Section, we present our design of a Content-Addressable Network. This paper gives only a brief overview of our CAN design; [6] presents the details and evaluation.

[2] The authors in [4], state that End System Multicast is more appropriate for small, sparse groups as in audio-video conferencing and virtual classrooms, while the authors in [1] apply their algorithm, Gossamer, to the self-organization of infrastructure proxies

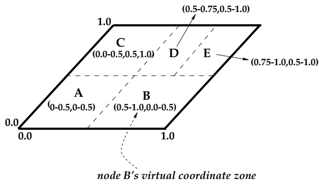

Fig. 1. *Example 2-d coordinate overlay with 5 nodes*

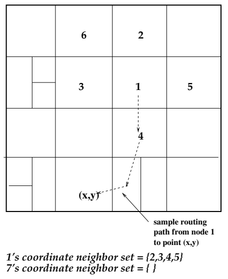

Fig. 2. *Example 2-d space before node 7 joins*

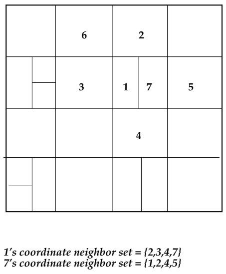

Fig. 3. *Example 2-d space after node 7 joins*

2.1 Design Overview

Our design centers around a virtual d-dimensional Cartesian coordinate space on a d-torus. [3] This coordinate space is completely logical and bears no relation to any physical coordinate system. At any point in time, the *entire* coordinate space is dynamically partitioned among all the nodes in the system such that every node "owns" its individual, distinct zone within the overall space. For example, Figure 1 shows a 2-dimensional $[0,1] \times [0,1]$ coordinate space partitioned between 5 CAN nodes. This coordinate space provides us with a level of indirection, since one can now talk about storing content at a "point" in the space or routing between "points" in the space where a "point" refers to the node in the CAN that owns the zone enclosing that point.

For example, this virtual coordinate space is used to store (key,value) pairs as follows: to store a pair (K_1,V_1), key K_1 is deterministically mapped onto a point, say (x, y) in the coordinate space using a uniform hash function. The corresponding key-value pair is then stored at the node that owns the zone within which the point (x, y) lies. To retrieve an entry corresponding to key K_1, any node can apply the same deterministic hash function to map K_1 onto point (x, y) and then retrieve the corresponding value from the point (x, y). If the point (x, y) is not owned by the requesting node or its immediate neighbors, the request must be routed through the CAN infrastructure until it reaches the node in whose zone (x, y) lies. Efficient routing is therefore a critical aspect of our CAN.

Nodes in the CAN self-organize into an overlay network that represents this virtual coordinate space. A node learns and maintains as its set of neighbors the IP addresses of those nodes that hold coordinate zones adjoining its own zone. This set of immediate neighbors serves as a coordinate routing table that enables routing between arbitrary points in the coordinate space.

We first describe the three most basic pieces of our design: CAN routing, construction of the CAN coordinate overlay, and maintenance of the CAN overlay and then briefly discuss the simulated performance of our design.

[3] For simplicity, the illustrations in this paper do not show a torus.

2.2 Routing in a CAN

Intuitively, routing in a Content Addressable Network works by following the straight line path through the Cartesian space from source to destination coordinates.

A CAN node maintains a coordinate routing table that holds the IP address and virtual coordinate zone of each of its neighbors in the coordinate space. In a d-dimensional coordinate space, two nodes are neighbors if their coordinate spans overlap along $d-1$ dimensions and abut along one dimension. For example, in Figure 2, node 5 is a neighbor of node 1 because its coordinate zone overlaps with 1's along the Y axis and abuts along the X-axis. On the other hand, node 6 is not a neighbor of 1 because their coordinate zones abut along both the X and Y axes. This purely local neighbor state is sufficient to route between two arbitrary points in the space: A CAN message includes the destination coordinates. Using its neighbor coordinate set, a node routes a message towards its destination by simple greedy forwarding to the neighbor with coordinates closest to the destination coordinates. Figure 2 shows a sample routing path.

For a d dimensional space partitioned into n equal zones, the average routing path length is thus $(d/4)(n^{1/d})$ and individual nodes maintain $2d$ neighbors. These scaling results mean that for a d dimensional space, we can grow the number of nodes (and hence zones) without increasing per node state while the path length grows as $O(n^{1/d})$.

Note that many different paths exist between two points in the space and so, even if one or more of a node's neighbors were to crash, a node would automatically route along the next best available path. If however, a node loses all its neighbors in a certain direction, and the repair mechanisms described in Section 2.4 have not yet rebuilt the void in the coordinate space, then greedy forwarding may temporarily fail. In this case, a node may use an expanding ring search to locate a node that is closer to the destination than itself. The message is then forwarded to this closer node, from which greedy forwarding is resumed.

2.3 CAN construction

As described above, the entire CAN space is divided amongst the nodes currently in the system. To allow the CAN to grow incrementally, a new node that joins the system must be allocated its own portion of the coordinate space. This is done by an existing node splitting its allocated zone in half, retaining half and handing the other half to the new node.

The process takes three steps:

1. First the new node must find a node already in the CAN.
2. Next, using the CAN routing mechanisms, it must find a node whose zone will be split.
3. Finally, the neighbors of the split zone must be notified so that routing can include the new node.

Bootstrap A new CAN node first discovers the IP address of any node currently in the system. The functioning of a CAN does not depend on the details of how this is done, but we use the same bootstrap mechanism as Yallcast and YOID [3]. As in [3] we assume that a CAN has an associated DNS domain name, and that this resolves to the IP address of one or more CAN bootstrap nodes. A bootstrap node maintains a partial list of CAN nodes it believes are currently in the system. Simple techniques to keep this list reasonably current are described in [3]. To join a CAN, a new node looks up the CAN domain name in DNS to retrieve a bootstrap node's IP address. The bootstrap node then supplies the IP addresses of several randomly chosen nodes currently in the system.

Finding a Zone The new node then randomly chooses a point (x, y) in the space and sends a JOIN request destined for point (x, y). This message is sent into the CAN via any existing CAN node. Each CAN node then uses the CAN routing mechanism to forward the message, until it reaches the node in whose zone (x, y) lies.

This current occupant node then splits its zone in half and assigns one half to the new node. The split is done by assuming a certain ordering of the dimensions in deciding along which dimension a zone is to be split, so that zones can be re-merged when nodes leave. For a 2-d space a zone would first be split along the X dimension, then the Y and so on. The (key, value) pairs from the half zone to be handed over are also transfered to the new node.

Joining the Routing Having obtained its zone, the new node learns the IP addresses of its coordinate neighbor set from the previous occupant. This set is a subset of the the previous occupant's neighbors, plus that occupant itself. Similarly, the previous occupant updates its neighbor set to eliminate those nodes that are no longer neighbors. Finally, both the new and old nodes' neighbors must be informed of this reallocation of space. Every node in the system sends an immediate update message, followed by periodic refreshes, with its currently assigned zone to all its neighbors. These soft-state style updates ensure that all of their neighbors will quickly learn about the change and will update their own neighbor sets accordingly. Figures 2 and 3 show an example of a new node (node 7) joining a 2-dimensional CAN.

As can be inferred, the addition of a new node affects only a small number of existing nodes in a very small locality of the coordinate space. The number of neighbors a node maintains depends only on the dimensionality of the coordinate space and is independent of the total number of nodes in the system. Thus, node insertion affects only *O(number of dimensions)* existing nodes which is important for CANs with huge numbers of nodes.

2.4 Node Departure, Recovery and CAN Maintenance

When nodes leave a CAN, we need to ensure that the zones they occupied are taken over by the remaining nodes. The normal procedure for doing this is for a node to explicitly hand over its zone and the associated (key,value) database to one of its neighbors. If the zone of one of the neighbors can be merged with the departing node's zone to produce a valid single zone, then this is done. If not, then the zone is handed to the neighbor whose current zone is smallest, and that node will then temporarily handle both zones.

The CAN also needs to be robust to node or network failures, where one or more nodes simply become unreachable. This is handled through a recovery algorithm, described in [6], that ensures one of the failed node's neighbors takes over the zone.

2.5 Design Improvements and Performance

Our basic CAN algorithm as described in the previous section provides a balance between low per-node state ($O(d)$ for a d dimensional space) and short path lengths with $O(dn^{1/d})$ hops for d dimensions and n nodes. This bound applies to the number of hops in the CAN path. These are *application level* hops, not IP-level hops, and the latency of each hop might be substantial; recall that nodes that are adjacent in the CAN might be many miles (and many IP hops) away from each other. In [6], we describe a number of design techniques whose primary goal is to reduce the latency of CAN routing. Of particular relevance to the work in this paper, is a distributed "binning" scheme whereby co-located nodes on the Internet can be placed close by in the CAN coordinate space. In this scheme, every node independently measures its distance (*i.e.* latency) from a set of well known landmark machines and joins a particular portion of the coordinate space based on these measurements. Our simulation results in [6] indicate that these added mechanisms are very effective in reducing overall path latency. For example, we show that for a system with over 130,000 nodes, for a range of link delay distributions, we can route with a latency that is well within a factor of three of the underlying IP network latency. The number of neighbors that a node must maintain to achieve this is approximately 28 (details of this test are in Section 4 in [6]).

3 CAN-based Multicast

In this section, we describe a solution whereby CANs can be used to offer an application-level multicast service.

If all the nodes in a CAN are members of a given multicast group, then multicasting a message only requires flooding the message over the entire CAN. As we shall describe in Section 3.2, we can exploit the existence of a well defined coordinate space to provide simple, efficient flooding algorithms from arbitrary sources without having to compute distribution trees for every potential source.

If only a subset of the CAN nodes are members of a particular group, then multicasting involves two pieces:

- the members of the group first form a group-specific "mini" CAN and then,
- multicasting is achieved by flooding over this mini CAN

In what follows, we describe the two key components of our scheme: group formation and multicast by flooding over the CAN.

3.1 Multicast Group Formation

To assist in our explanation, we assume the existence of a CAN C within which a subset of the nodes wish to form a multicast group G. We achieve this by forming an

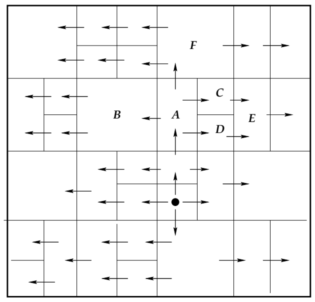

Fig. 4. *Directed flooding over the CAN*

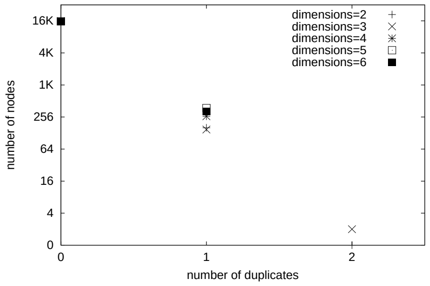

Fig. 5. *Duplicate messages using CAN-based multicast*

additional mini CAN, call it C_g, made up of only the members of G. The underlying CAN C itself is used as the bootstrap for the formation of C_g as follows: using a well-known hash function, the group address G is deterministically mapped onto a point, say (x, y), and the node on C that owns the point (x, y) serves as the bootstrap node in the construction of C_g. Joining group G thus reduces to joining the CAN C_g. This is done by repeating the usual CAN construction process with (x, y) as the bootstrap node. Because of the light-weight nature of the CAN bootstrap mechanisms, we do not expect the CAN bootstrap node to be overloaded by join requests. If this becomes a possibility however, one could use multiple bootstrap nodes to share the load by using multiple hash functions to deterministically map the group name G onto multiple points in the CAN C; the nodes corresponding to each of these points would then serve as a bootstrap node for the group G. As with the CAN bootstrap process, the failure of the bootstrap node(s) does not affect the operation of the multicast group itself; it only prevents new nodes from joining the group during the period of failure.

Thus, every group has a corresponding CAN made up of all the group members. Note that with this group formation process a node only maintains state for those groups for which it is itself a member or for which it serves as the bootstrap node. For a d-dimensional CAN, a member node maintains state for $2d$ additional nodes (its neighbors in the CAN), independent of the number of traffic sources in the multicast group.

3.2 Multicast forwarding

Because all the members of group G (and no other node) belong to the associated CAN C_g, multicasting to G is achieved by flooding on the CAN C_g. Different flooding algorithms are conceivable; for example, one might consider a naive flooding algorithm wherein a node caches the sequence numbers of messages it has recently received. On receiving a new message, a node forwards the message to all its neighbors (except of course, the neighbor from which it received the message) only if that message is not already in its cache. With this type of floodcachesuppress algorithm a source can reach every group member with requiring a routing algorithm to discover the network topology. Such an algorithm does not make any special use of the CAN structure and could in fact be run over any application-level topology including a random mesh topology as generated in [4, 1]. The problem with this type of naive flooding algorithm is that it can result in a large amount of duplication of messages; in the worst case, a node could receive a single message from each of its neighbors.

A more efficient flooding solution would be to exploit the coordinate space structure of the CAN as follows:

Assume that our CAN is a d-dimensional CAN with dimensions $1 \dots d$. Individual nodes thus have at least $2d$ neighbors; 2 per dimension with one to move forward and another to move in reverse along each dimension. *i.e.* for every dimension i a node has at least one neighbor whose zone abuts its own own in the forward direction along i and another neighbor whose zone abuts its own in the reverse direction along i. For example, consider node A in Figure 4: node B abuts A in the reverse direction along dimension 1 while nodes C and D abut A in the forward direction along dimension 1.

Messages are then forwarded as follows:

1. The source node (*i.e.* node that generates a new message) forwards a message to all its neighbors
2. A node that receives a message from a neighbor with which it abuts along dimension i forwards the message to those neighbors with which it abuts along dimension $1 \dots (i-1)$ and the neighbors with which it abuts along dimension i on the opposite side to that from which it received the message. Figure 4 depicts this directed flooding algorithm for a 2-dimensional CAN.
3. a node does not forward a message along a particular dimension if that message has already traversed at least half-way across the space from the source coordinates along that dimension. This rule prevents the flooding from looping round the back of the space.
4. a node caches the sequence numbers of messages it has received and does not forward a message that it has already previously received

For a perfectly partitioned (*i.e.* where nodes have equal sized zones) coordinate space, the above algorithm ensures that every node receives a message exactly once. For imperfectly partitioned spaces however, a node might receive the same message from more than one neighbor. For example, in Figure 4, node E would receive a message from both neighbors C and D.

Certain duplicates can be easily avoided because, under normal CAN operation, every node knows the zone coordinates for each of its neighbors. For example, consider once more Figure 4; nodes C and D both know each others' and node E's zone coordinates and could hence use a deterministic rule such that only one of them forwards messages to E. Such a rule, however, only eliminates duplicates that arise by flooding along the first dimension. The rule works along the first dimension because, *all* nodes forward along the first dimension. Hence even if a node, by applying some deterministic rule, does not forward a message to its neighbor along the first dimension, we know that some other node that does satisfy the deterministic rule will do so. But this need not be the case when forwarding along higher dimensions. Consider a 3-dimensional CAN; if a node by the application of a deterministic rule decides not to forward to a neighbor along the second dimension, there is no guarantee that any node will eventually forward it up along the second dimension because the node that does satisfy the deterministic rule might receive the packet along the first dimension and hence will not forward the message along the second dimension. [4] For example, in Figure 4 let us assume that node A decides (by the use of some deterministic rule) not to forward to node F. Because node C receives the message (from A) along the first dimension, it will not forward the message along the second dimension either and hence node F and the other nodes with Y-axis coordinates in the same range as F, will never receive the message. While the above strategy does not eliminate all duplicates, it does eliminate a large fraction of it because most of the flooding occurs along the first dimension. Hence, we augment the above flooding algorithm with the following deterministic rule used to eliminate duplicates that arise from forwarding along the first dimension:

– let us assume that a node, P, received a message along dimension 1 and that node Q abuts P along dimension 1 in the opposite direction from which P received the message. Consider the corner C_q of Q's zone that abuts P along dimension 1 and has the lowest coordinates along dimensions $2 \ldots d$. Then, P only forwards the message on to Q, if P is in contact with the corner C_q.

So, for example, in Figure 4, with respect to nodes C and D, the corner under consideration for node E would be the lower, leftmost corner of E's zone. Hence only D (and not C) would forward messages E in the forward direction along the first dimension.

For the above flooding algorithm, we measured through simulation the percentage of nodes that experienced different degrees of message duplication caused by imperfectly partitioned spaces. Figure 5 plots the number of nodes that received a particular number of duplicate messages for a system with 16,384 nodes using CANs with dimensions ranging from 2 to 6. In all cases, over 97% of the nodes receive no duplicate messages and amongst those nodes that do, virtually all of them receive only a single duplicate message. This is a considerable improvement over the naive flooding algorithm wherein *every* node might receive a number of duplicates up to the degree (number of neighbors) of the node.

It is worth noting that the naive flooding algorithm is very robust to message loss because a node can receive a message via any of its neighbors. However, the efficient flooding algorithm is less robust because the loss of a single message results in the breakdown of message delivery to several subsequent nodes thus requiring additional loss recovery techniques. This problem is however, no different than in the case of traditional IP multicast or other application-level schemes where the loss of a packet along a single link results in the packet being lost by all downstream nodes in the distribution tree. With both flooding algorithms, the duplication of messages arises because we do not (unlike most other solutions to multicast delivery) construct a single spanning tree rooted at the source of traffic. However, we believe that the simplicity and scalability gained by not having to run routing algorithms to construct and maintain such delivery trees is well worth the slight inefficiencies that may arise from the duplication of messages.

Using the above flooding algorithm, any group member can multicast a message to the entire group. Nodes that are not group members can also multicast to the en-

[4] By the second rule in the flooding algorithm.

tire group by first discovering a random group member and relaying the transmission through this random group member. [5] This random member node can be discovered by contacting the bootstrap node associated with the group name.

4 Performance Evaluation

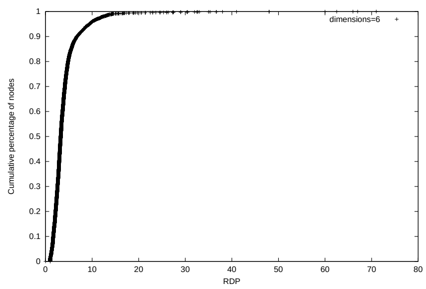

Fig. 6. *Cumulative distribution of RDP*

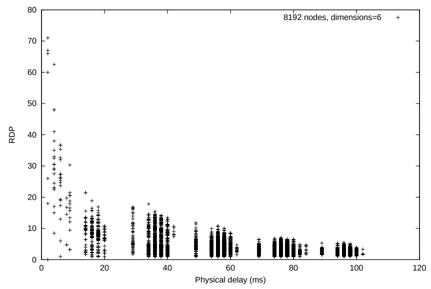

Fig. 7. *RDP versus physical delay for every group member*

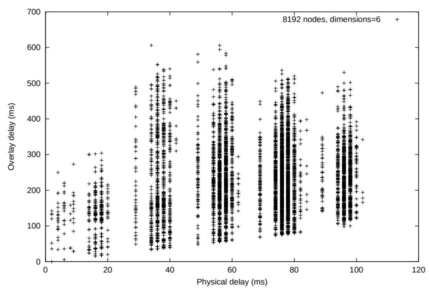

Fig. 8. *Delay on the overhead versus physical network delay*

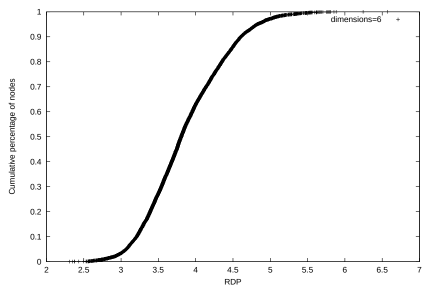

Fig. 9. *Cumulative distribution of RDP averaged over 100 traffic sources*

In this section, we evaluate, through simulation, the performance of our CAN-based multicast scheme. We adopt the performance metrics and evaluation strategy used in

[5] Note that relaying in our case is different from relayed transmissions as done in source specific multicast [11] because only transmissions from non-member nodes are relayed and even these can be relayed through any member node.

[4]. As with previous evalution studies of application-level multicasti schemes[4, 1, 11] we compare the performance of CAN-based multicast to native IP multicast and naive unicast-based multicast where the source simply unicasts a message to every receiver in succession. Our evaluation metrics are:

- **Relative Delay Penalty (RDP)**: the ratio of the delay between two nodes (in this case, the source node and a receiver) using CAN-based multicast to the unicast delay between them on the underlying physical network
- **Link Stress**: the number of identical copies of a packet carried by a physical link

Our simulations were performed on Transit-Stub (TS) topologies using the GT-ITM topology generator [9]. TS topologies model networks using a 2-level hierarchy of routing domains with transit domains that interconnect lower level stub domains.

4.1 Relative Delay Penalty

We first present results from a multicast transmission using a single source as this represents the performance typically seen across the different receiver nodes for a transmission from a single source. These simulations were performed using a CAN with 6 dimensions and a group size of 8192 nodes. The source node was selected at random. We used Transit-Stub topologies with link latencies of 20ms for intra-transit domain links, 5ms for stub-transit links and 2ms for intra-stub domain links.

Both IP multicast and Unicast-based multicast achieve an RDP value of one for all group members because messages are transmitted along the direct physical (IP-level) path between the source and receivers. Routing on an overlay network however, fundamentally results in higher delays. Figure 6 plots the cumulative distribution of RDP over the group members. While the majority of receivers see an RDP of less than about 5 or 6, a few group members have a high RDP. This can be explained [6] from the scatter-plot in Figure 7. The figure plots the relation between the RDP observed by a receiver and its distance from the source on the underlying IP-level, physical network. Each point in Figure 7 indicates the existence of a receiver with the corresponding RDP and IP-level delay. As can be seen, all the nodes with high values of RDP have a low physical delay to the source, i.e. the very low delay from these receivers to the source inflates their RDP. However, the absolute value of their delay from the source on the CAN overlay is not really very high. This can be seen from Figure 8, which plots, for every receiver, its delay from the source using CAN multicast versus its physical network delay. The plot shows that while the maximum physical delay can be about 100ms, the maximum delay using CAN-multicast is about 600ms and the receivers on the left hand side of the graph, which had the high RDP, experience delays of not more than 300ms.

The above results were all for a single multicast transmission using a single source; Figure 9 plots the cumulative distribution of the RDP with the delays averaged over multicast transmissions from a 100 sources selected at random. Because a node is unlikely to be very close (in terms of physical delay) to all 100 sources, averaging the results over transmissions from many sources helps to reduce the appearance of inflated

[6] The authors in [4] make the same observation and explanation

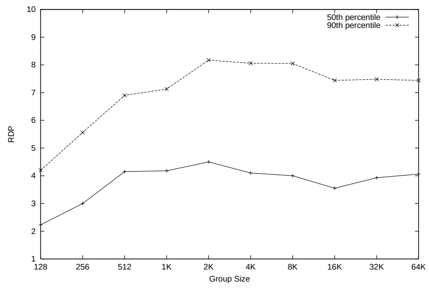

Fig. 10. *RDP versus increasing group size*

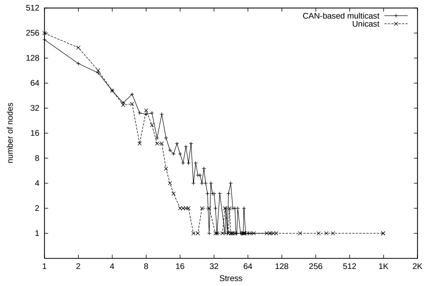

Fig. 11. *Number of physical links with a given stress*

RDPs that occurs when a receiver is very close to the source. From Figure 9 we see that, on an average, no node sees an RDP of more than about 6.0.

Finally, Figure 10 plots the 50 and 90 percentile RDP values for group sizes ranging from 128 to 65,000 for a single source. We scale the group size as follows: we take a 1,000 node Transit-Stub topology as before and to this topology, we add end-host (source and receiver) nodes to the stub (leaf) nodes in the topology. The delay of the link from the end-host node to the stub node is set to 1ms. Thus in scaling the group size from a 128 to 65K nodes, we're scaling the density of the graph without scaling the backbone (transit) domain. So, for example, a group size of 128 nodes implies that approximately one in ten stub nodes has an associated group member while a group size of 65K implies that every stub node has approximately 65 attached end-host nodes. This method of scaling the graph causes the flat trend in the growth of RDP with group size because for a given source the relative number of close-by and distant nodes stays pretty much constant. Further, at high density, every CAN node has increasingly many close-by nodes and hence the CAN binning technique used to cluster co-located nodes yields higher gains. Different methods for scaling topologies could yield different scaling trends.

While the significant differences between End-System Multicast and CAN-based multicast makes it hard to draw any direct comparison between the two systems; Figure 10 indicates that the performance of CAN-based multicast even for small group sizes is competitive with End-System multicast.

4.2 Link Stress

Ideally, one would like the stress on the different physical links to be somewhat evenly distributed. Using native IP multicast, every link in the network has a stress of exactly one. In the case of unicasting from the source directly to all the receivers, links close to the source node have very high stress (equal to the group size at the first hop link from the source). Figure 11 plots the number of nodes that experienced a particular stress value for a group size of 1024 for a 6-dimensional CAN. Unlike naive unicast where a small number of links see extremely high stress, CAN-based multicast distributes the stress much more evenly over all the links.

Figure 12 plots the worst-case stress for group sizes ranging from 128 to 65,000 nodes. The high stress in the case of large group sizes is because, as described earlier, we scale the group size without scaling the size of the backbone topology. For the above simulation, we used a transit-stub topology with a 1,000 nodes. Hence for a group size of 65,000 nodes, all 65,000 nodes are interconnected by a backbone topology of less than 1,000 nodes thus putting high stress on some backbone links. We repeated the above simulation for a transit-stub topology with 10,000 nodes, thus decreasing the density of the graph by a factor of 10. Figure 13 plots the worst-case stress for group sizes up to 2,048 nodes for all three cases (*i.e.* CAN-based multicast using Transit-Stub topologies with 1,000 and 10,000 nodes and naive unicast-based multicast). As can be seen, at lower density the worst-case stress drops sharply. For example, at 2,048 nodes the worst case stress drops from 169 (for TS1000) to 37 (for TS10000). Because, in practice, we do not expect very high densities of group member nodes relative to the Internet topology itself, worst-case stress using CAN-based multicast should be at a reasonable level. In future work, we intend looking into techniques that might further lower this stress value.

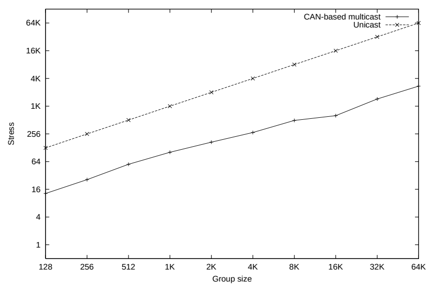

Fig. 12. *Stress versus increasing group size*

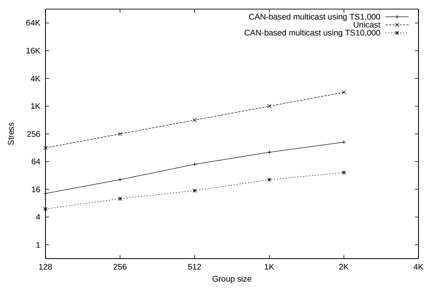

Fig. 13. *Effect of topology density on stress*

5 Related Work

The case for application-level multicast as a more tractable alternative to a network-level multicast service was first put forth in [4, 3, 1].

The End-system multicast [4] work proposes an architecture for multicast over small and sparse groups. End-system multicast builds a mesh structure across participating end-hosts and then constructs source-rooted trees by running a routing protocol over this mesh. The authors also study the fundamental performance penalty associated with such an application-level model. The authors in [1] argue for infrastructure support to tackle the problem of content distribution over the Internet. The Scattercast architecture relies on proxies deployed within the network infrastructure. These proxies self-organize into an application-level mesh over which a global routing algorithm is used to construct distribution trees. In terms of being a solution to application-level multicast, the key difference between our work and the End-System multicast and Scattercast work is the potential for CAN-based multicast to scale to large group sizes.

Yoid [3] proposes a solution to application-level multicast wherein a spanning tree is directly constructed across the participating nodes without first constructing a mesh structure. The resultant protocols are more complex because the tree-first approach results in expensive loop detection and avoidance techniques and must be made resilient to partitions.

Tapestry [10] is a wide-area overlay routing and location infrastructure that, like CANs, embeds nodes in a well-defined virtual address space. Bayeux [11] is a source-specific, application-level multicast scheme that leverages the Tapestry routing infrastructure. To join a multicast session, Bayeux nodes send *JOIN* messages to the source node. The source replies to a *JOIN* request by routing a *TREE* message, on the Tapestry overlay, to the requesting node. This *TREE* message is used to set up state at intermediate nodes along the path from the source node to the new member. Similarly, a *LEAVE* message from an existing member triggers a *PRUNE* message from the root, which removes the appropriate forwarding state along the distribution tree. Bayeux and CAN-based multicast are similar in that they achieve scalability by leveraging the scalable routing infrastructure provided by systems like CAN and Tapestry. In terms of service model, Bayeux fundamentally supports only source-specific multicast while CAN-based multicast allows any group member to act as a traffic source. In terms of design, Bayeux uses an explicit protocol to set-up and tear down a distribution tree from the source node to the current set of receiver nodes. CAN-based multicast by contrast, fully exploits the CAN structure because of which messages can be forwarded without requiring a routing protocol to explicitly construct distribution trees.

Overcast[5] is a scheme for source-specific, reliable multicast using an overlay network. Overcast constructs efficient dissemination trees rooted at the single source of traffic. The overlay network in Overcast is composed of nodes that reside within the network infrastructure. This assumption of the existence of permanent storage within the network distinguishes Overcast from CANs and indeed, from most of the other systems described above. Unlike Overcast, CANs can be composed entirely from end-user machines with no form of central authority.

6 Conclusion

Content-Addressable Networks have the potential to serve as an infrastructure that is useful across a range of applications. In this paper, we present and evaluate a scheme that extends the basic CAN framework to support application-level multicast delivery. There are, we believe, two key benefits to CAN-based multicast: the potential to scale to large groups without restricting the service model and the simplicity of the scheme under the assumption of the deployment of a distributed infrastructure such as a Content-Addressable Network.

Our CAN-based multicast scheme is optimal in terms of the distance (in terms of path length) in flooding messages over the CAN overlay structure itself. In future work, we intend looking into simple clustering techniques to further reduce the link stress caused by our flooding algorithm and understanding what the fundamental limitations there are. A number of important questions such as security, loss recovery, and congestion control remain to be addressed in the context of CAN-based multicast.

7 Acknowledgments

We thank Ion Stoica for his valuable input and Yan Chen and Morley Mao for sharing their data.

References

1. CHAWATHE, Y., MCCANNE, S., AND BREWER, E. An architecture for internet content distribution as an infrastructure service. available at http://www.cs.berkeley.edu/ yatin/papers, 2000.
2. DEERING, S. E. *Multicast Routing in a Datagram Internetwork*. PhD thesis, Stanford University, Dec. 1991.
3. FRANCIS, P. Yoid: Extending the internet multicast architecture. Unpublished paper, available at http://www.aciri.org/yoid/docs/index.html, Apr. 2000.
4. HUA CHU, Y., RAO, S., AND ZHANG, H. A case for end system multicast. In *Proceedings of SIGMETRICS 2000* (Santa Clara, CA, June 2000).
5. JANNOTTI, J., GIFFORD, D., JOHNSON, K., KAASHOEK, F., AND O'TOOLE, J. Overcast: Reliable multicasting with an overlay network. In *Proceedings of the Fourth Symposium on Operating Systems Design and Implementation* (San Diego, CA, Oct. 2000).
6. RATNASAMY, S., FRANCIS, P., HANDLEY, M., KARP, R., AND SHENKER, S. A Scalable Content-Addressable Network. In *Proceedings of SIGCOMM 2001* (Aug. 2001).
7. ROWSTRON, A., AND DRUSCHEL, P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. available at http://research.microsoft.com/ antr/PAST/, 2001.
8. STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M. F., AND BALAKRISHNAN, H. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of SIGCOMM 2001* (Aug. 2001).
9. ZEGURA, E., CALVERT, K., AND BHATTACHARJEE, S. How to Model an Internetwork. In *Proceedings IEEE Infocom '96* (San Francisco, CA, May 1996).
10. ZHAO, B. Y., KUBIATOWICZ, J., AND JOSEPH, A. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. available at http://www.cs.berkeley.edu/ ravenben/tapestry/, 2001.
11. ZHUANG, S. Q., ZHAO, B., JOSEPH, A., KATZ, R., AND KUBIATOWICZ, J. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of the Eleventh International Workshop on Network and OS Support for Digital Audio and Video* (New York, July 2001), ACM.