SelectCast - A Scalable and Self-Repairing Multicast Overlay Routing Facility *

Adrian Bozdog Robbert van Renesse
Department of Computer Science
Cornell University, Ithaca, NY 14853
<adrianb,rvr>@cs.cornell.edu

Dan Dumitriu
School of Computer and Comm. Sciences
EPFL, Lausanne, Switzerland
danmihai.dumitriu@epfl.ch

ABSTRACT

In this paper we describe SelectCast, a self-repairing multicast overlay routing facility for supporting publish/subscribe applications. SelectCast is a peer-to-peer protocol, and leverages Astrolabe, a secure distributed information management system. SelectCast uses replication to recover quickly from transient failures, as well as Astrolabe's aggregation facilities to recover from long-term failures or adapt to changes in load or QoS requirements. In order to evaluate the scalability and performance of SelectCast, and compare these with other multicast facilities, we built a multicast testing facility on NetBed. This paper reports latency and load results for SelectCast, compared to both native IP multicast and Yoid.

1. INTRODUCTION

Many distributed applications require some form of multicast. Examples include collaborative applications such as teleconferencing and games, news delivery services such as a stock ticker, locate services such as expanding ring search, as well as video distribution services. Unfortunately, IP-level multicast routing is badly supported in today's Internet. There are various reasons for this. Perhaps most importantly, multicast addresses do not aggregate as well as do unicast addresses, and the mapping of multicast addresses to locations is much more dynamic than for unicast addresses. As a consequence, routing tables that support multicast may grow very large while being highly dynamic. Also, as flow and congestion control for IP-level multicast routing is not well understood. ISPs are not eager to deal with such problems.

Peer-to-peer, application-level multicast (ALM) routing is therefore an important alternative to IP multicast, because ALM protocols do not require the cooperation of ISPs, and because they can support various forms of multicast Quality-of-Service beyond those provided by IP multicast, including flow and congestion control, buffering, retransmission, and filtering. Several ALM protocols have been shown to perform nearly as well as IP multicast in terms of latency, bandwidth, and stress on network links [9, 7].

All current Application-Level Multicast Routing Protocols (ALMRPs) route messages along trees in order to get logarithmic scaling behavior with respect to the number of receivers (assuming the tree has some bounded maximum branching factor and is reasonably well balanced). These protocols put an uneven load on the hosts and networks, as most hosts (at the leaves of the tree) only receive messages, while some hosts, which we call *routers*, have to forward copies of each message to some set of peers. In an ALMRP capable of filtering, these routers may also have to analyze the content of messages in order to decide what links to forward the messages to. For satisfactory performance, robustness, and scale, it is important to select well-provisioned, dependable hosts for routers, and to recover quickly from their failures.

Many ALMRPs have been proposed. Our protocol, SelectCast, goes beyond previous work on ALMRPs in the following two ways:

- SelectCast offers users great flexibility in how to select routers. For example, routers can be selected to mini-

*This research was funded in part by DARPA/AFRL-IFGA grant F30602-99-1-0532, in part by a grant under NASA's REE program administered by JPL, and in part by NSF-CISE grant 9703470. The authors are also grateful for support from the AFRL/Cornell Information Assurance Institute.

George W. Johnson Center, George Mason University '03 , Fairfax, VA

mize latency or maximize throughput. The default selects routers based on longevity, thereby attempting to optimize robustness. The selection may be changed at run-time.

- SelectCast allows senders to specify the set of intended destination hosts through the use of a SQL condition on selected attributes of such hosts. Note that this is different from publish/subscribe systems in which the subscribers specify what messages they are interested in receiving, either by topic or by a predicate on messages. Our strategy is strictly more powerful than traditional topic and content-based publish/subscribe mechanisms.

Like several other ALMRPs, our approach is built upon a peer-to-peer infrastructure. This infrastructure, Astrolabe [14], provides a topology-aware domain hierarchy and secure aggregation of attributes associated with the domains. We will see that this infrastructure is essential to providing the innovative properties of SelectCast.

This paper is organized as follows. In Section 2 we describe the SelectCast protocol in detail. An experimental evaluation of SelectCast appears in Section 3. Section 4 surveys related work. Section 5 concludes.

2. SELECTCAST

In this section we describe the SelectCast system. As SelectCast makes extensive use of Astrolabe in order to maintain the dissemination tree, we start out with a quick review of Astrolabe. Readers are referred to [14] for an in-depth description and analysis of this secure and highly scalable peer-to-peer infrastructure. We then provide an overview of the basic concepts of the SelectCast protocol. In the following subsections, we describe various details of the implementation.

2.1 Astrolabe

Astrolabe can most easily be thought of as a peer-to-peer implementation of a DNS-like directory service which supports on-the-fly aggregation. That is, the hosts are organized in a domain hierarchy, and each domain has a set of attributes. The attributes of leaf domains (*i.e.*, hosts) are writable, but the attributes of a non-leaf domain are generated by summarizing the attributes of its child domains. For example, the domains may have an attribute called "uptime." Hosts report how long they have been up in their respective leaf domain attributes. The "uptime" attributes of internal domains could be calculated by, say, taking the minimum, and thus report the shortest time that a host has been up in that domain. Astrolabe supports an extensive set of summarizing functions based on SQL, and these aggregation queries can be installed on-the-fly in a secure fashion.

The Astrolabe service runs an agent on each host. Such an agent maintains a *domain table* for every non-leaf domain that it is in. A domain table contains a row for each child domain, and a column for each attribute. In each domain table of an agent, one of the rows is the agent's own. Except for the leaf domain row, which is written directly by the corresponding agent, the agent produces its own rows by aggregating the tables of the corresponding child domains. The rows not owned by the agent are learned through an epidemic protocol known as *gossip*, and which is secured using public key cryptography.

Each domain has an attribute "contacts", containing a small set of addresses of agents representing the domain. The contact attribute itself is calculated by choosing the addresses of the k *best* agents among the contacts of the child domains, where k is a small integer (typically 3), and *best* based on some metric such as minimum load or longest uptime. The contacts attribute of a leaf domain is the set of the addresses of the corresponding agent, which is typically a singleton set. However, an agent may have more than one address if it is multi-homed, that is, attached to more than one network.

A separate gossip protocol instance runs for every domain. The protocol is run among the contacts of the child domains of the domain. Each contact, at regular intervals, chooses a sibling domain at random, and then starts a message exchange with a randomly chosen contact for the sibling domain. The message exchange contains essentially their respective current versions of the parent domain table. The tables are merged based on timestamps, and at the end of the exchange the resulting tables in the respective agents are identical.

If the gossip protocol is run every T seconds, a contact will be involved, on average, in two gossip exchanges per T seconds. Due to the randomization, there is some variance in this load. Also, in a reasonably balanced tree an agent may be a contact of as many as $O(\log N)$ domains, where N is the total number of agents. Although fairly constant, these overheads are not negligible and are apparent in the performance measurements provided in Section 3.

The gossip protocol detects the absence of gossips from certain hosts. If this is the case, those hosts are removed from the system and the aggregation is automatically recalculated. This possibly results in new contacts being selected for domains. Even if all contacts for a domain were to fail simultaneously, new contacts will be selected and communication with the rest of the Astrolabe hierarchy restored.

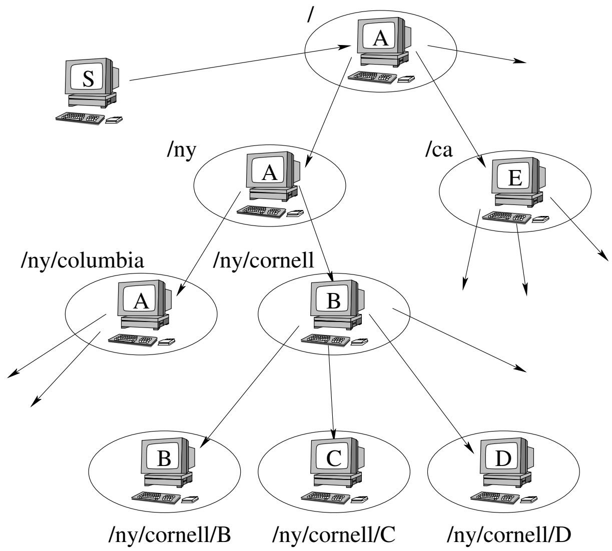

Figure 1: Routing in SelectCast.

```
on receipt(msg):
  for each child of msg.domain
  do
    if msg.filter(child.attrs)
    then
      msg2 := new message;
      msg2.domain := msg.domain + '/'
        + child.id;
      msg2.filter := msg.filter;
      msg2.data := msg.data;
      send msg2 to
        BEST(child.attrs("routers"));
    fi
  done
```

Figure 2: The SelectCast forwarding algorithm.

2.2 SelectCast Overview

SelectCast uses Astrolabe in order to decide how to route multicast messages. We will show how we build a single multicast forwarding tree, but SelectCast can be instantiated many times (say, dozens) on a single Astrolabe instance. The basic idea is as follows. Each domain has an attribute called "router" containing the address of a host that will act as a message forwarding agent for that domain. For leaf domains, the attribute contains the address of the corresponding host. For internal domains, the "router" attribute of one of the child domains is chosen. (We will show different ways of doing so below.) To multicast a message, it is first sent to the root domain's router. The router of each domain forwards the message to the routers of each of the domain's child domains, and so the message is disseminated to all hosts.

For example, in Figure 1, a partial picture of an Astrolabe hierarchy is shown. There are six hosts: a sender S and receivers A through E. For each domain, we show the router for that domain, which in this example is chosen to be the router with the lowest name in alphabetical order. The sender S sends the message to the root domain's router, which is A. A forwards the message to its child domain's routers, which are E and itself. And so on until the leaf domains are reached. The corresponding tree of routers forms the multicast forwarding tree.

Messages may contain predicates that the routers use to forward messages to child domains selectively. This is how SelectCast got its name. The routers apply the predicate to the attributes of its child domains. For example, assume each domain has an attribute called *uptime* that contains the minimum uptime of the hosts in that domain. A message with the condition "uptime $<$ 180" is then forwarded only to those hosts that have been up less than three minutes. Note that, in fact, the message is forwarded only to those domains that contain such hosts. This is a generalization of the topic-based publish/subscribe paradigm. As we will see later, we can support this efficiently for arbitrarily complex queries, and even content-based filtering can be expressed this way.

2.3 Tolerating Churn

As described above, a router receives and forwards messages on behalf of a domain. In case a router fails, Astrolabe will automatically select a new router based on an aggregation query. However, this process can take a relatively long time. In order to recover from failed routers quickly, a domain may have more than one router. Astrolabe's flexible aggregation facility allows users to specify how router selection is done. For example, if there is a lot of "host churn" (hosts joining and leaving rapidly), the user may specify to use the three hosts with the largest uptime in a domain. This is the default selection. If load balancing is a concern, a user may instead specify to use the three hosts with the least load. If security is a concern, selection may be based on the physical security of hosts. This aggregation query can even be changed on-the-fly if requirements change. Note that Astrolabe updates the set of routers not only as hosts fail and recover, but also as domain attributes such as load and uptime change.

Typically, the selection is of the form where each non-leaf domain selects the k *best* routers from the routers of the child domains. Here k is a small positive integer that determines the level of failure masking, and *best* is some *aggregation condition* such as "maximum uptime," "minimum load," or "maximum security." It is even possible to have multiple sets of routers for each domain, so that different messages can be routed differently. For example, high throughput traffic could use the least loaded routers, while low volume but high security traffic could use the most secure routers.

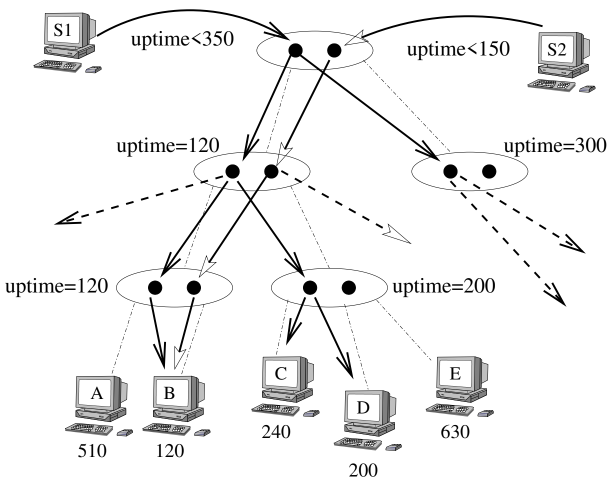

Figure 3: Forwarding with multiple routers and filters.

2.4 Forwarding

Messages are forwarded from the root domain down to the leaf domains. A host may be a router for more than one domain, and so when a message arrives at a host, the host needs to know for which domain the message is intended in order to forward the message to the correct subdomains. For example, in Figure 1, host B represents two domains. In order to achieve correct forwarding, each message has two attributes:

- **domain:** the parent domain from which the message is distributed, initially the root domain "/";
- **filter:** the predicate that is checked before forwarding the message to some child domain;

Any host can act as a router, and executes the algorithm of Figure 2. On receipt of a message `msg`, the host uses Astrolabe to iterate over all child domains of `msg.domain`. For each such child domain, the host applies the predicate in the message to the attributes of the child domain. If this predicate evaluates to `true`, a new message is forwarded to one of the child domain's routers. Each SelectCast router maintains statistics about the child domain's routers in order to select the most reliable one.

The forwarded message is the same as the received message, except that the identifier of the child domain is appended to the `domain` attribute of the message. Thus, if the parent domain was "/ny", and the child domain's local identifier was "cornell", then the new `domain` attribute will be "/ny/cornell." The destination host is obtained by taking the "routers" attribute of the child domain, and applying the function `BEST` to it in order to select one of the routers. (`BEST` is as described in Section 2.3.)

Figure 3 shows the forwarding of two messages from two hosts, $S1$ and $S2$ in a partial depiction of an Astrolabe hierarchy, in which two routers are elected within each domain. Five hosts are shown with their "uptime" attributes. In order for this example to work, Astrolabe has to be configured to aggregate the uptime attribute in the domains by taking the minimum. (In the next subsection we show how to make this work for arbitrary queries.) The message sent by host $S1$ uses the filter "uptime $<$ 350" (so that only those hosts that have been up for less than 350 seconds are targeted), while the message sent by host $S2$ contains the "uptime $<$ 150" predicate. The figure shows how the messages are forwarded to only their intended recipients.

Note that the forwarding algorithm works no matter what initial host is used. Although it may seem natural to send a message to one of the root domain's routers first, as suggested in the overview, it is advantageous to send the message to the local host. This way the root routers will not become bottlenecks. Effectively, instead of having a single forwarding tree, there is a forwarding tree for each child of the root domain, resulting in significantly better performance and scalability. For example, in Figure 1, S would have sent the message to E as well as to A, thereby off-loading A with the responsibility of the first level of forwarding.

2.5 Filtering

Publish/subscribe services usually support either topic-, or content-based subscriptions. In a topic-based subscription, each subscriber specifies the list of topics that the subscriber is interested in, while the publisher specifies for each message it sends what the topic is. The simplest way to support this in Astrolabe is to have a boolean attribute per topic, which is aggregated by logical OR. If there are many attributes, it may be much more efficient to use Bloom filters [4]. This solution uses a single attribute that contains essentially a fixed-size bit map that is aggregated using bitwise OR. Topic names are hashed to a bit in this bit map. The condition tagged to the message is "`BITSET(HASH(topic))`". In the case of hash collisions, this solution may lead to messages being routed to more destinations than strictly necessary, thus the size of the bitmap should grow dynamically so that the rate of collisions will be acceptably low.

This technique can easily be generalized for arbitrarily complex filters. Given an arbitrary predicate P on attributes of domains, each receiver sets a bit corresponding to a hash of P in the Bloom filter in case P evaluates to true at that receiver. As a Bloom filter is essentially of constant size, this technique scales quite well. Nevertheless, if many predicates are used, the Bloom filter has to grow accordingly in order to be an effective filter in the top-level domains. The size of the Bloom filter controls the trade-off between precision of filtering and the space used inside Astrolabe.

The operations and their results are cached, and recalculated only as attributes change or members come or go. Thus, the overhead of filtering in SelectCast is negligible compared to the overhead of forwarding.

Note that the predicates above are predicates on Astrolabe attributes. In a content-based publish/subscribe system, subscribers specify which messages they are interested in by using a predicate on messages. This can be supported as follows in SelectCast. Each subscriber enters its predicate in a specific attribute, say *interest*. These attributes are aggregated by Astrolabe by OR-ing them together. (Our SQL engine supports a general aggregation operator `FOLD()`, that takes two arguments: the attribute to be aggregated, and a binary operation.)

Publishers add the following condition onto their messages: "`EVAL(`*`interest`*`(this))`". This condition applies the code in the domain's *interest* attribute to the message. (`EVAL` is not a standard SQL operator, but a variety of popular SQL engines do support this functionality.)

In order to make this scale, the aggregated *interest* attribute should not be too complex. We intend to add a `SIMPLIFY()` operation that conservatively simplifies the *interest* expression (returning `TRUE` in the limit). The *interest* attribute would then be generated by "`SELECT SIMPLIFY(FOLD(`*`interest`*`,` **or**`)) AS` *`interest`*". Essentially, `SIMPLIFY()` assumes the role of the Bloom filter, and should return a result of maximum size. For example, "uptime $<$ 30 OR uptime $<$ 50" can be simplified to be "uptime $<$ 50". Again, if many predicates are in use, this filter is likely relatively ineffective in the top-level domains, but can still be quite effective near the leaves where most communication takes place.

2.6 Flow Control

So far we have talked about routing with no regard for reliability of message delivery. If we do not implement retransmission and/or congestion control, our protocol is unlikely to be useful. Rather than building our own mechanisms for this, SelectCast currently uses standard TCP connections for forwarding data between routers. For efficiency, each host maintains a cache of TCP connections to other hosts.

Our implementation uses sockets, which provide some fixed amount of buffering in case the send window is full. Back-pressure when this buffer is full is provided through `EWOULDBLOCK` error notifications when trying to send a message. If this occurs when forwarding a message to some domain's router, SelectCast will attempt to send the message to another router for the same domain. Only when sending fails for all routers of a domain, SelectCast will give up forwarding to that domain until Astrolabe selects new routers. Applications have limited control over this back-pressure (by setting the socket's send buffer size).

In spite of these efforts, messages can get lost in a variety of ways as SelectCast does not buffer any messages on its own, and does not implement any end-to-end flow control mechanism. With limited buffering, such a mechanism would slow the rate of dissemination down to at most what the slowest receiver can accept, and therefore we have rejected this notion. SelectCast should therefore be considered a best-effort multicast routing mechanism, consistent with an end-to-end approach to building reliable applications [13].

We do intend to offer message logging services in the near future to aid applications with requirements for message recovery. Astrolabe can be used to locate and manage such logging servers.

3. EXPERIMENTS

In this section, we evaluate the performance of SelectCast, and compare it to the performance of IP multicast and another ALMRP, namely Yoid.[1] The experiments were run using implementations on an actual network. Rather than looking at maximum message rate, which is a measure that depends heavily on the underlying network infrastructure and the hosts, we measure the load on the network and the hosts. We also present latency numbers for the experimental set-up that we used. We ran our experiments on NetBed, the Utah Network

[1]The latest version of the protocol, which is the one that we used, differs quite substantially from the one described in [8]. A description of the version of Yoid that we used can be found in a full paper on SelectCast [5].

Testbed [3] (formerly known as *Emulab*). In our experiments, we used Astrolabe hierarchies with a branching factor of no more than two (*i.e.*, each domain had no more than two members). SelectCast did not use any filter for the experiment messages.[2]

All Yoid experiments used Yoid's default configuration parameters. One of the members was used as Yoid's rendezvous server. The experiments used different numbers of nodes up to 64 nodes and three different types of network topologies. The first topology was a *LAN* topology in which all experiment nodes were connected through a LAN. The other two topologies had two node LANs (*topo_of_2nodeLAN*) and four node LANs (*topo_of_4nodeLAN*) connected through a backbone, in which one router was assigned to each LAN.

The LANs were configured with no message delays and no loss probability. All nodes were workstation-class computers (600 to 850 MHz Pentium with 256-512 MByte RAM and 100 MBit Ethernet interfaces) and ran Linux. IP-Multicast experiments were only conducted on the LAN topology, while SelectCast and Yoid experiments were conducted on all topologies. The experiments used 4, 6, 8, 10, and 14 senders. Each sender had to send 500 100-byte messages, while the inter-message period was 100 ms. (Again, we are only interested in load, as maximum message rate is highly dependent on the platform used.) The senders resided in different LANs of the topo_of_2nodeLAN topology, and were distributed in pairs per LAN for experiments made with the topo_of_4nodeLAN topology. The bandwidth used and the network load were measured using tcpdump [2]. The network load is the number of packets sent and received by a node per second.

Below is a summary of the performance results. For a full treatment, see [5].

3.1 Network Load Results

We ran experiments using up to 64 nodes for both LAN and topo_of_4nodeLAN topologies. Because a router has to be assigned to each LAN, there were not enough nodes available to run experiments with 64 members using the topo_of_2nodeLAN topology on NetBed.

Node network load results take into consideration two types of load: *total load* and *user load*. The total load represents the total number of network packets sent or received by a node per second, while the user load only considers those packets that contain a payload. In other words, the user data of a node is the number of packets sent or received by the node's ALM system. Most of the remaining packets contain TCP acknowledgement messages, and do not generate a direct load on the ALM system proper.

In the experiments below, we observed that SelectCast's total load is typically about double its user load, while Yoid's total load is typically about the same as its user load. This is because SelectCast uses TCP for message forwarding, and the inter-packet time is so large that separate acknowledgement packets are returned for each data message. The acknowledgements do not impose much of a load on the hosts, however.

For each experiment, we have computed the mean, maximum, and standard deviation for both load types across all members. In Figure 4, we show both the maximum user load and the maximum total load among all nodes in various topologies with four senders. Because the computed load values from the SelectCast experiments are the same for all topologies, the figure contains only the SelectCast results for one topology. Most importantly, the loads appear to grow very slowly with the size of the membership. We can observe that the maximum total load for SelectCast and Yoid is about the same, except that for one topology with 16 nodes Yoid has unstable behavior during which we see an explosion of TCP-level acknowledgements. (Yoid uses a mixture of UDP and TCP in order to maintain the tree and improve its latency, although user-level data itself uses UDP exclusively.)

(We observed unstable behavior in Yoid in a variety of experiments. In all these experiments the number of members is at most 20 and the number of senders is at most 8. In these experiments, the rendezvous server has the maximum total load, but there are other nodes with unexpectedly large total load values as well. For most experiments there is another node which has its load close to the rendezvous server load. We observed that in at least some of these situations the high loads are caused by communication between the rendezvous server and two or three other nodes.)

In a dissemination tree, each host receives and forwards on average 2 messages for every message sent. As in this experiment there are four senders each sending 10 messages per second, on average each host transmits and receives a total of 80 messages per second. If we look at mean load in Figure 5, we observe that both SelectCast and Yoid behave mostly as expected (ignoring TCP-level acknowledgements). For IP-multicast this load is halved, as hosts do not have to forward packets. Figure 6 shows the standard deviation of both user and total load. The standard deviation is generally fairly low, although we can again observe the unstable behavior in Yoid, while SelectCast has consistent stable behavior.

[2]Because of caching, there is no measurable performance overhead for filtering.

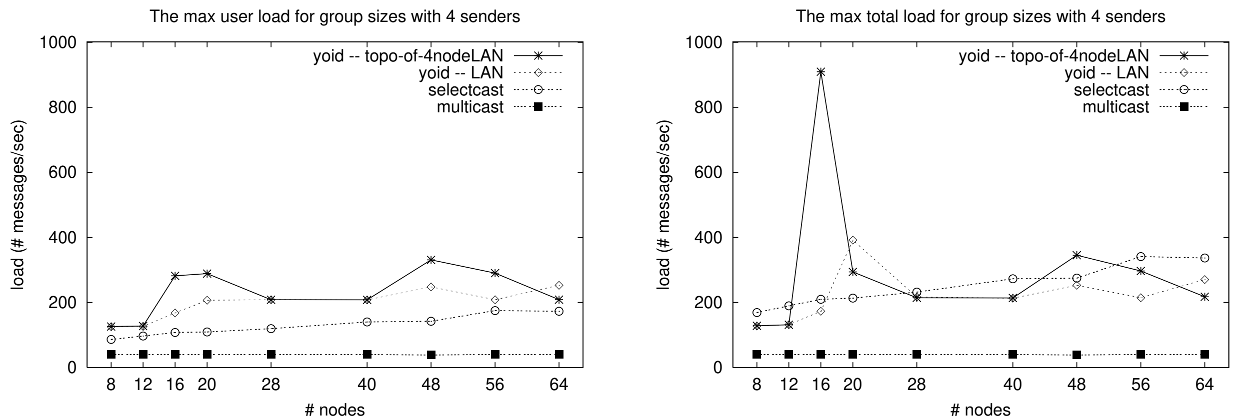

Figure 4: The maximum load for 4 senders

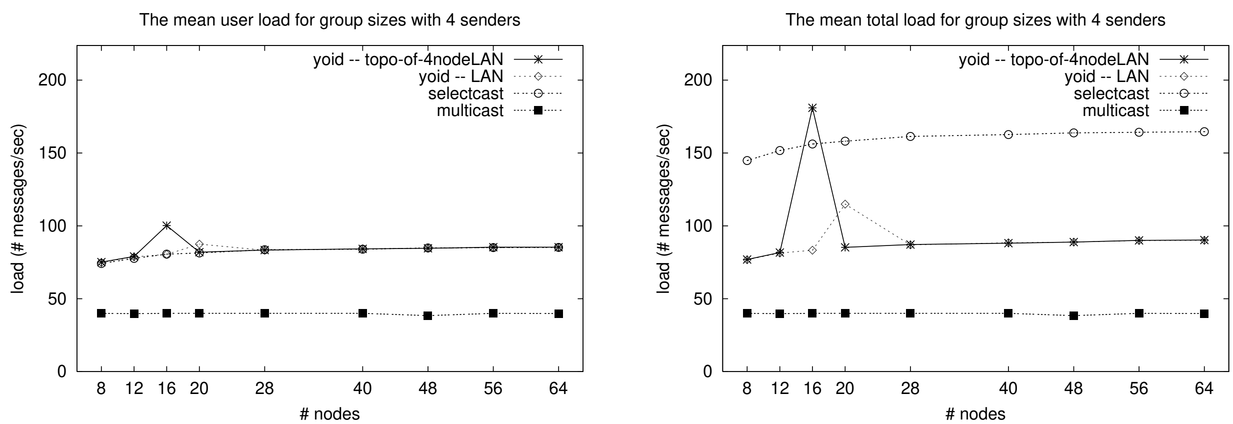

Figure 5: The mean load for 4 senders

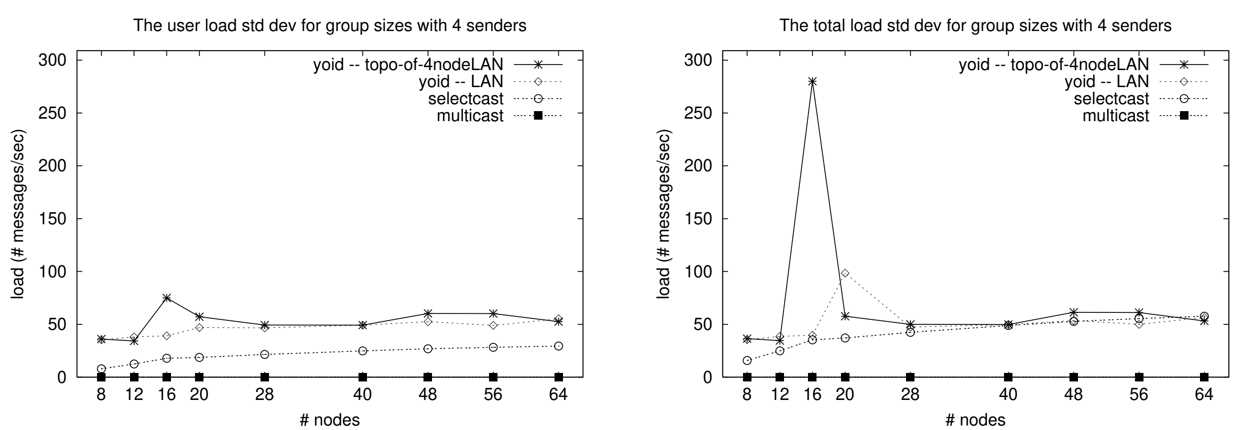

Figure 6: The standard deviation of the load for 4 senders

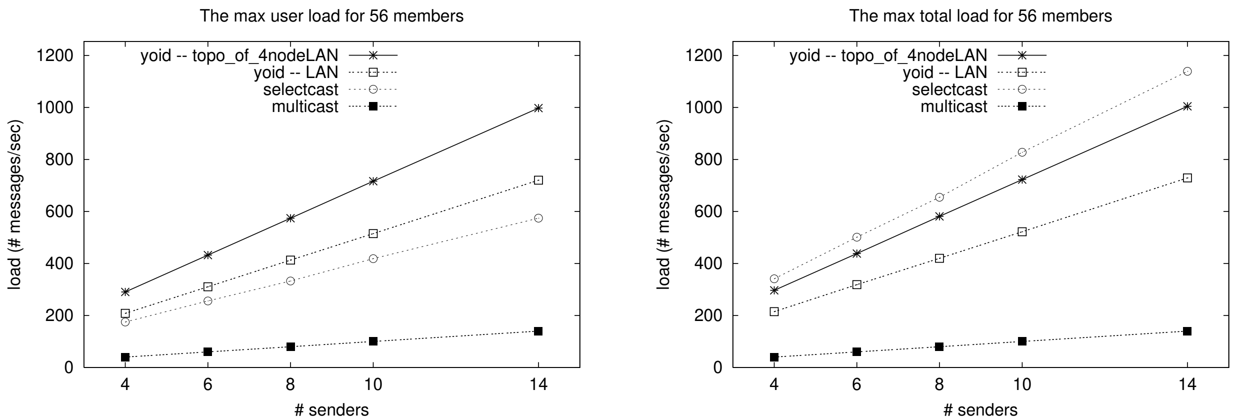

Figure 7: Results from experiments with fixed number of members

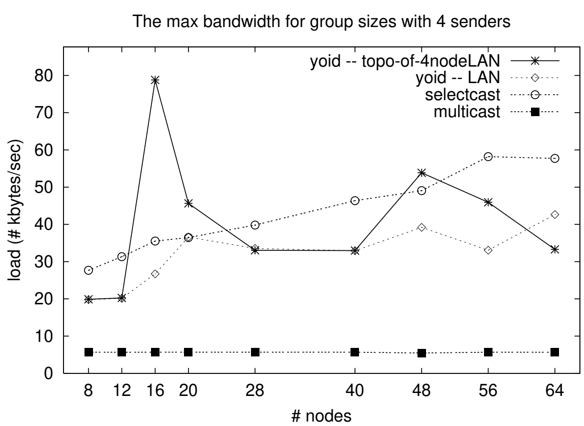

Figure 8: Maximum bandwidth results

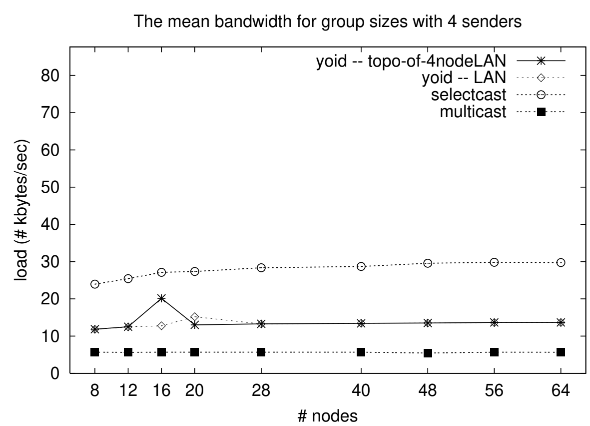

Figure 9: Mean bandwidth results

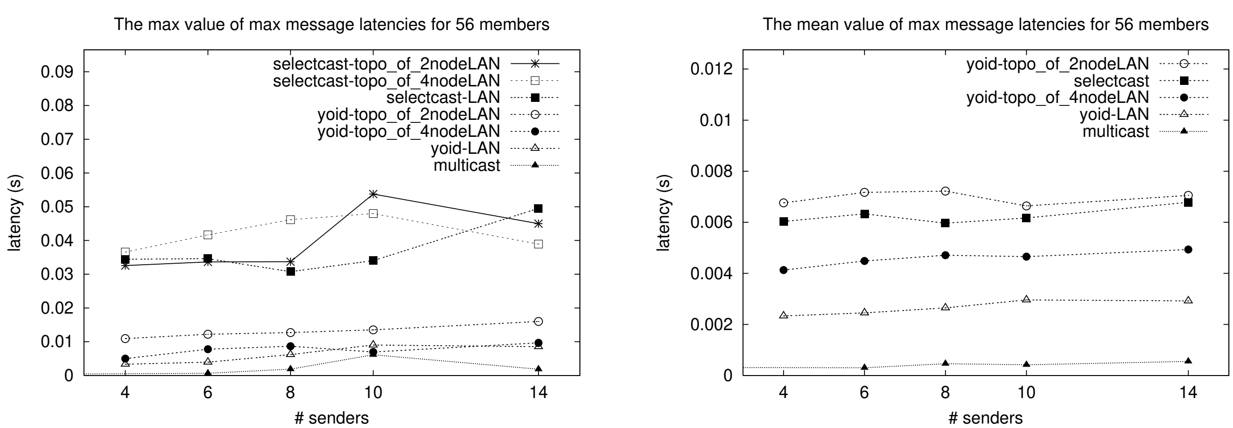

Figure 10: Latency results for experiments with 56 members

In Figure 7 we show the load results of experiments with a variable number of senders. As in previous results, SelectCast values are not topology specific. SelectCast puts a slightly higher load on the network than Yoid, but SelectCast's load on the ALM system is lower than Yoid's. In all cases, the load appears to grow linearly with the number of senders, as expected and desired.

3.2 Bandwidth Results

The bandwidth load represents the number of kbytes sent or received by a node per second. We measured the bandwidth load for all experiments described in Section 3.1. For each experiment, we computed the mean, maximum and standard deviation of bandwidth load across all members. The computed values from SelectCast experiments were the same for all topologies.

In Figure 8, we show the maximum bandwidth load in the LAN, and topo_of_4nodeLAN topologies with four senders. We can see that the differences between maximum bandwidth values for different topologies behave similarly as those between the maximum of total load values shown in Figure 4. As observed for network load results, the bandwidth load grows slowly with the size of the membership.

Figure 9 shows the mean bandwidth load. SelectCast's bandwidth load is twice as large as Yoid's load because SelectCast uses TCP for forwarding where each TCP message triggers an acknowledgement while Yoid uses UDP. The load imposed by TCP acknowledgements per host for experiments with 4 senders is around 6 kbytes per second. Moreover, Astrolabe's overhead is 30% out of SelectCast's load. This explains clearly the difference between SelectCast's bandwidth load and Yoid's load for packets that contain data.

3.3 Latency Results

We measured maximum message latencies for experiments with 56 members on all topologies. A message's maximum latency is the time between sending the message and the last receipt. For this, members' clocks were synchronized using the Network Time Protocol service [1]. Prior to each experiment, 25 messages were sent by each member to warm up the employed ALM system. For example, SelectCast establishes connections between its domains when the first message is sent, and caches information about them at every router.

Figure 10 plots the mean and maximum message latencies for experiments made with a variable number of senders for all test topologies. The first observation is that the latencies do not depend much on the number of senders. From studying the data we collected, we conjecture that the small anomalies are caused by variances in the delay of NTP resynchronization messages (received from a central server). For LAN topologies, SelectCast does significantly worse than Yoid. This is because SelectCast does not adjust its hierarchy automatically like Yoid does, and thus a message has to travel through many hops (up to seven in the given configuration).

4. RELATED WORK

In this section we describe the most relevant ALM systems that we are aware of. An extended survey of related work is included in the full paper [5].

CAN multicast (CAN-ALM) [12] is an ALM system built on top of CAN (Content-Addressable Networks) [11], a peer-to-peer object location system that uses a multi-dimensional Cartesian coordinate space. CAN-ALM forward messages only based on the coordinates of sources and neighbors, without the need of any additional routing information. CAN-ALM has an important advantage over SelectCast by having routing tables of constant size at all group members . SelectCast's capabilities for filtering and to adapt to network conditions is an important advantage over CAN-ALM.

Bayeux [16] and Scribe [6] are ALMRPs based on peer-to-peer Plaxton routing [10] and take network locality into account. One advantage of Bayeux and Scribe over SelectCast is the usage of servers from employed routing technologies, which eliminate the need of implementing routing protocols at the members. Unlike Bayeux and Scribe, SelectCast supports filtering, user-driven router selection, and applications with multiple senders using a tree for each sender.

Overcast [9] consists of a central source node, and a number of Overcast nodes with permanent storage spread throughout the Internet. Overcast organizes its nodes into a distribution tree rooted at the source that maximizes each of its nodes' bandwidths from the source. Overcast has more flexibility in selecting routers than SelectCast and may create a more optimal dissemination tree. On the other hand, SelectCast can optimize the dissemination tree for various different metrics, and supports filtering.

Narada [7] is a mesh-based ALM protocol intended for relatively small groups. Narada attempts to optimize the mesh continually so that for any pair of members the shortest path latency in the mesh is comparable to the unicast latency. To route multicast messages, Narada uses a protocol which is similar to the Distance Vector Multicast Routing Protocol (DVMRP) [15]. Because of Narada's $O(n^2 lg(N))$ control overhead, it does not scale as well as SelectCast. On the other hand, Narada may find more optimal configurations for small systems. Narada does not support in-network filtering.

5. CONCLUSIONS

In this paper, we presented the SelectCast self-organizing publish/subscribe facility. We also presented an initial performance evaluation of SelectCast when compared to both native IP-multicast and Yoid, another self-organizing application-level multicast facility. By looking at SelectCast's performance results in live experiments with up to 64 members, we have gained confidence in that SelectCast will scale well to much larger number of members.

Currently, SelectCast does not make use of hardware multicast capabilities. We plan to make information about domain multicast available through Astrolabe so that SelectCast can decide whether the hosts in a particular domain are covered by hardware multicast.

Acknowledgements

We would like to thank Paul Francis, Ken Birman, Werner Vogels and the anonymous reviewers for extensive comments on this paper, and Yuri Pryadkin for his help with the description of the Yoid protocol as implemented and distributed. We also like to thank Suman Banerjee, Yang-hua Chu, Peter Druschel, David Helder, John Janotti, Sharad Mittal, and Sylvia Ratnasamy for comments on the Related Work section.

6. REFERENCES

[1] Network Time Protocol. http://www.eecis.udel.edu/ ntp/. University of Delaware ECE/CIS.

[2] tcpdump. http://www.tcpdump.org.

[3] The Utah Network Testbed. http://www.emulab.net.

[4] B. Bloom. Space/time tradeoffs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, July 1970.

[5] A. Bozdog, R. van Renesse, and D. Dumitriu. Selectcast: Scalable and self-repairing multicast overlay routing. *In submission*, 2004.

[6] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralized Application-Level Multicast infrastructure. *IEEE Journal on Selected Areas in Communications (JSAC)*, 20(8), 2002.

[7] Y. Chu, S. Rao, and H. Zhang. A case for end system multicast. In *Proceedings of ACM SIGMETRICS*, pages 1–12, Santa Clara, CA, June 2000.

[8] P. Francis, S. Ratnasamy, R. Govindan, and C. Alaettinoglu. Yoid project. http://www.icir.org/yoid/.

[9] J. Jannotti, D. Gifford, K. Johnson, M. Kaashoek, and J. Jr. Overcast: Reliable multicasting with an overlay network. In *Proceedings of the Fourth Symposium on Operating System Design and Implementation (USENIX OSDI)*, pages 197–212, Santa Clara, CA, October 2000.

[10] C. Plaxton, R. Rajaraman, and A. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 1997.

[11] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of SIGCOMM 2001*, UC San Diego, California, USA, August 2001.

[12] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-Level Multicast using Content-Addressable Networks. In *Proceedings of the Third International Workshop on Networked Group Communication*, London, UK, November 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):277–288, November 1984.

[14] R. van Renesse, K. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management and data mining. *ACM Transactions on Computer Systems*, 21(2), May 2003.

[15] D. Waitzman, C. Partridge, and S. Deering. Distance Vector Multicast Routing Protocol. RFC-1075, 1988.

[16] S. Zhuang, B. Zhao, A. Joseph, R. Katz, and J. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of the Eleventh International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV 2001)*, Port Jefferson, New York, June 2001.