Strategies for Using Multicasting to Locate Resources

Craig E. Wills
Department of Computer Science
Worcester Polytechnic Institute
Worcester, MA 01609

Abstract

One issue in distributed computing systems is how to locate and manage resource information that is inherently distributed within the computing environment. This work investigates the use of a technique called multicasting for location of such distributed information.

In this paper we present three different strategies for using the resources contained by a machine to dynamically compute the multicast addresses on which the machine should listen for requests of the information. The central goal of these strategies is to filter out requests for resource information at the network interface of a machine so that a machine only receives a request for information if it contains the resource.

1 Introduction

One issue in distributed computing systems is how to locate and manage information that is inherently distributed within the computing environment. This issue is particularly important as environments dynamically change and grow in size. A specific example of a distributed resource is the set of users logged into a distributed environment. Another example is in a distributed file system where the complete set of files is distributed between many file servers. In a similar manner, we can view each machine as a potential computation server willing to provide a portion of the total computational services available in the network. Rather than view each of these problems separately, we view them as a related class of problems in which solutions found for one problem may be applicable to other problems.

This work investigates the use of a technique called multicasting [8] for location of such distributed information. Multicasting supports efficient delivery of messages to a subset of machines, as opposed to broadcast, which delivers messages to all machines, or unicast, which delivers messages from one machine to another. This paper investigates strategies for using multicast to locate resources that are inherently distributed within a local area network. For this work we assume machines are interconnected by broadcast bus technology, such as Ethernet

Multicasting shows much promise for resource location because it allows each message to be sent to a subset of machines in the network. The broadcast bus technology works by sending a single message on the wire addressed to either a specific machine, or a group (multicast) address. Only the machines listening for that address actually process the message. The question is how to map resource location requests to multicast addresses. In the ideal case a machine receives a resource request only when it contains information about the resource.

In this paper we present three different strategies for using the resources contained by a machine to dynamically compute the multicast addresses on which the machine should listen for requests of the information. The first strategy assumes that the space of total multicast addresses is large and attempts to map each resource to a unique multicast address. The second strategy realizes the number of multicast addresses that any single machine can recognize is limited so it maps each resource to a limited range of addresses. In each case a client seeking information about a resource simply uses the appropriate mapping function to compute the multicast address for sending the request. The third strategy combines the previous two strategies in a two-tiered approach. If a machine con-

\footnote{Ethernet is a registered trademark of the Xerox Corporation.}
tains just a few resources then it maps each resource to a unique address, while if it contains many resources then it switches strategies and maps to a fixed set of addresses. A client seeking information sends a request to addresses computed from both mapping functions, but any machine receives at most one request.

For each strategy we present expected results through simulation and analysis. In each case we compare against a single multicast group (broadcast) to understand the benefits gained with each multicasting strategy. After presenting and analyzing the different strategies we elaborate specific applications that we are exploring for the use of multicasting. These applications include: specific resource problems such as locating users in a distributed environment; deciding when it is appropriate to use central name servers rather than storing the information at each node and using the strategies discussed; and using multicasting to locate resources by their attributes rather than simply by their name.

2 Related Work

Much work has been done on different approaches to locating resources in a distributed environment. One approach is to store all information at central name servers, to which any request for information or update of information is sent. Grapevine [3] and Clearinghouse [13] are examples of systems that use this technique for managing information in large, distributed environments.

Rather than centralize information in one, or a few, nodes, resource information can be stored locally at each node in the network. In this approach, access to the information is simple because it is stored locally, but as resource information changes it must be communicated to all other machines in the network. This technique is used by the rwhod protocol in the UNIX\(^2\) operating system to manage user and machine status information [5]. However, because this information changes frequently and is communicated using broadcasting, this protocol is typically not used as the number of machines on a network grows.

Another approach to manage resource information is simply to store it at each node that has information and provide the information as it is requested by clients. This approach is useful when the information changes frequently in comparison to how often it is used. Typically broadcast requests are sent to locate such information. This approach was used in Clouds [6] for locating remote objects. The Sun RPC rstat and rusers programs use this technique to collect information about machine and user status in a network [15].

Other techniques, such as caching, have been used with these approaches to reduce the cost of accessing data. Cheriton and Mann [4] have successfully combined caching along with multicasting in the study of a decentralized naming facility. In previous work, we have investigated different approaches for managing information depending on the nature of the information [17].

In addition to resource location, multicasting is useful for communication between distributed processes. In the ISIS [14] and Amoeba [11] systems, work has been done to provide reliable multicasting to ensure that each participating process receives all messages in the same order.

3 Approach

For this work we want to explore strategies for keeping all information at the server nodes that supply information, and use multicasting to access this information. To understand the potential benefits of multicasting it is important to understand the cost of network communication on a machine. Figure 1 shows a machine connected via a network interface to a bus network.

![Network Interface of a Machine](image)

For each packet that is broadcast on the bus, there are three levels at which the packet may reach:

---

\(^2\)UNIX is a registered trademark of AT&T.
1. Network interface. All packets are received by the network interface on each machine. By default the network interface accepts all packets sent to its bus address (a unicast message) and all packets sent to the broadcast address for the bus (a broadcast message). The interface can also be configured to accept packets addressed to a multicast group. All packets not broadcast, unicast to this host, or multicast to a group in which this host is participating, are rejected at the network interface level.

2. Operating system kernel. All packets accepted by the network interface cause the operating system kernel to be interrupted to handle the packet. Thus, a consequence of broadcasting is that each machine on the network must handle the broadcast packet regardless if any user-level process is listening for it. If no user-level process is listening for a packet then it is discarded by the kernel.

3. User process. All packets addressed to an active process on the machine cause the kernel to make a context switch to the process to handle the received packet. In terms of resource location if the packet corresponds to a request for a resource known by the server process then the process replies with the information, however, if resource is not known by this server process then many CPU cycles have been wasted on handling this packet.

Ideally a request is either rejected at the network interface or it is successfully honored by a process on the host. Any other action wastes machine resources. With broadcasting, all requests reach at least the operating system level, even on hosts that are not participating in the service. Broadcasting to an active process causes a context switch to occur on each machine. As machines become more powerful, interrupts and context switching become relatively more expensive [2], therefore increasing the importance of reducing the occurrence of these activities.

Given the problem that a client machine wishes to locate a resource that is contained by a known number of machines on the network, the goal is to match resource requests with request packets handled by a host. If a host has a requested resource then it must participate. If it does not have the resource then the request is rejected at the network interface level. The question addressed by this paper is how close to the ideal can we come using multicasting. Specifically with the use of dynamically computed multicast addresses where the address used for locating a resource is computed from the name of the resource itself.

4 System Model

We assume that a server process exists on each machine in the local area network and knows about the resources offered by that machine. Rather than assume each resource exists at one and only one machine in the network, we assume the more general case that at any time a resource \( r \in R \) may exist at any number of nodes within the network. Because we assume the network is a broadcast bus, each packet that is sent on the network is received by the network interface of all machines, but is only passed on to the operating system if the address of the packet matches an address configured for the interface. Each packet contains a request for one resource.

We assume that the multicast address space is large. For example, the Ethernet hardware reserves 47 bits for multicast addresses. The IP multicasting software [7], which maps IP network addresses into hardware addresses has 28 bits available for multicast addresses, although only the low-order 23 bits of the IP multicast address are mapped into a special range of Ethernet multicast addresses.

Processes may selectively join and leave multicast groups. In the IP multicasting software processes join and leave groups by issuing systems calls that communicate with the network interface. The network interface knows which multicast addresses are currently in use and passes packets addressed to these addresses to the kernel, which passes them along to the appropriate process.

To compare different strategies for using multicasting we will use our goal of trying to match resource requests with request packets handled by a host. If a request is received by a host and the host has the requested resource then the host is correctly “hit.” Otherwise we consider this request a “miss” for the given host. An ideal strategy would have would have no misses.

To illustrate our approach we can look at a simple use of multicasting—using a single multicast address for all machines participating in the management of a given resource. In this approach each server process registers at a well-known multicast address and all requests are sent to this address. The advantage of a multicast address versus using the broadcast address is that machines not participating are not bothered (packets are rejected by their network interface). In Figure 2 the line denoted by “\( G = 1 \)” shows the percentage of misses for a resource plotted against the
number of resources contained by the host. For the example, we assume that there are \( R = 100 \) resources (we can also view this number as the percent of total resources contained by the host).

The figure shows the expected result that if a machine contains only one resource then 99% of the requests will result in a miss. If a machine contains all resources then no misses will occur.

5 Dynamic Multicast Address Strategies

Use of a static multicast address may eliminate uninterested machines from having to participate in a resource location protocol, but that approach does not take full advantage of the opportunities presented by multicasting. Rather we wish to explore strategies for multicasting by dynamically computing multicast addresses based upon resource names.

5.1 Unique Multicast Addresses

Since the range of multicast addresses is large (approximately eight million unique addresses provided by IP multicasting) the simplest approach is to map each resource name to a unique address. We define a hashing function \( \text{T} \) such that for each resource \( r \in R \) we compute the multicast address \( a \) for that resource by computing \( a = \text{T}(r) \). If the function \( \text{T} \) is ideal then each resource will be mapped into a unique multicast address and the percentage of misses, regardless of the number or resources contained by a machine, will be zero.

Using this approach the routines needed to manage the resource information are straightforward. \text{AddResource}(r), called by a server when a resource is added to the set of resources for that machine, simply uses the function \( \text{T} \) to compute the corresponding multicast address and instructs the network interface to join this multicast group. \text{DeleteResource}(r), called by a server when a resource is removed from the set of resources for the machine, computes the multicast address of the resource and instructs the network interface to leave this multicast group. Finally, \text{FindResource}(r), called by a client routine to locate information about a resource, computes the multicast address and sends a single request to this address. It then retrieves responses from server machines who are members of this multicast group. Depending on the nature of the request and resource the client will wait for a fixed number of responses (perhaps one) or time out waiting for responses. For enhanced reliability the request may be retransmitted.

An important question with this approach is how to define the hashing function \( \text{T} \). As an initial direction we have explored the use of polynomial code or cyclic redundancy code (CRC) checksums to use in computing a hash value [16]. This code is normally used to compute message checksums for error detection in network transmission, but it is also attractive in this application because it can be computed on an arbitrary string of bytes (resource name), and by varying the degree of the polynomial the size (in bits) of the resulting checksum can be changed.

To illustrate the use of a CRC polynomial checksum computation we have done some preliminary investigation and have some results based on the polynomial \( z^{20} + z^{18} + z^{17} + z^{16} + z^{12} + z^5 + z^4 + 1 \). This polynomial will result in checksums of 20 bits, which we choose as a large hash space. The first resource space we tried was the set of all user login ids on the central computer here at WPI. There are 2883 such login ids. Since the polynomial is of degree 20 there are \( 2^{20} \) or approximately one million unique addresses that are available.

Table 1: Mapping User Login Ids to Multicast Address Slots

<table>
<thead>
<tr>
<th>Number of entries: 2883, slots: 1048576</th>
</tr>
</thead>
<tbody>
<tr>
<td>0: 1045697</td>
</tr>
<tr>
<td>1: 2875</td>
</tr>
<tr>
<td>2: 4</td>
</tr>
</tbody>
</table>

Expected results:

<table>
<thead>
<tr>
<th>Number of entries: 2883, slots: 1048576</th>
</tr>
</thead>
<tbody>
<tr>
<td>0: 1045697.96</td>
</tr>
<tr>
<td>1: 2875.08</td>
</tr>
<tr>
<td>2: 3.95</td>
</tr>
</tbody>
</table>

As the results show there are 1045697 address slots with zero ids mapped to the slot, 2875 slots with one id mapped, and 4 slots where two different ids would map to the same address. The second part of the table shows the expected result of a hash function where all entries are equally likely to be selected. The result is a binomial distribution, which is approximated with a Poisson distribution in the table. Comparing the two sets of numbers indicates that the hashing function is good for the set of resource values.

As a second example, we repeated the same experiment using the set of words used by the spell checker...
on our UNIX system. This list contains 25144 words with the results of mapping this "word" resource to the 20-bit address space shown in Table 2.

Table 2: Mapping Words to Multicast Address Slots

<table>
<thead>
<tr>
<th>Number of entries: 25144, slots: 1048576</th>
</tr>
</thead>
<tbody>
<tr>
<td>0: 1023709</td>
</tr>
<tr>
<td>1: 24596</td>
</tr>
<tr>
<td>2: 267</td>
</tr>
<tr>
<td>3: 5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Expected results:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0: 1023731.07</td>
</tr>
<tr>
<td>1: 24548.24</td>
</tr>
<tr>
<td>2: 294.32</td>
</tr>
<tr>
<td>3: 2.35</td>
</tr>
<tr>
<td>4: 0.01</td>
</tr>
</tbody>
</table>

In this case our experiment hash function gives slightly better results than the expected values in mapping each resource to a unique slot. In both cases the number of resources hashing to the same address is small, indicating that the percentage of misses for any given machine would be small, particularly if a machine was expected to contain no more than a small fraction of the total resources. These results indicate that a good hash function for a large address space is available. Additional work needs to be done to examine other hashing functions and to see how much the results deteriorate if fewer address slots are available.

The problem with mapping each resource name to its own multicast address is there is a practical limit on the number of multicast addresses that a single network interface may support. For example, the Digital UNIBUS Network Adapter, DUENA\textsuperscript{3} \cite{9}, supports a maximum of 10 multicast addresses. Other network interfaces support more addresses, but at some point the set of addresses available to a machine will be limited. In addition, the number of addresses that can be supported for any process by the operating system may be limited. For example, in the IP multicasting implementation for the UNIX operating system there is a default limit of 20 multicast address groups per socket (the network access mechanism for a process). These limits indicate that for machines maintaining a large number of resources, it is not practical to try and map each resource to a unique multicast address because the software and network interface cannot support the total range of multicast addresses.

5.2 Group Multicast Addresses

Rather than try to map each resource to a unique multicast group across a large address space, another strategy is to map resource names into an address space as large as a server process can support. If we represent this limit as $M$ then we can

---

\textsuperscript{3}UNIBUS and DEUNA are trademarks of Digital Equipment Corporation
define a group hashing function $\Gamma$ that maps a resource into one of $M$ multicast addresses. Using this approach, the $\text{AddResource}(r)$, $\text{DeleteResource}(r)$, and $\text{FindResource}(r)$ routines are the same as defined previously except the hashing function $\Gamma$ is used in place of the function $T$.

Assuming that a server process can join no more than $M = 20$ multicast groups at any time, Figure 2 shows the expected miss rate for $G = M = 20$ multicast groups as compared against one group. As shown the miss rate increases until at approximately 50 resources all 20 multicast groups have at least one resource mapped to them and the server process will receive all requests, just as the case for only one multicast group address. The area between the two curves is the percentage of requests that are rejected at the network interface level. Figure 3 shows the same result for groups of size 1, 5, 20, and 50. As expected the fewer the groups, the more quickly the curve merges with the single group ($G = 1$) curve.

The results shown in the previous two figures assume that the mapping function evenly distributes the entire domain of resources over the range of multicast addresses. One approach for improving the percentage of misses is to examine if resources provided by a particular machine can all be mapped to a small number of address groups. To explore the desirability of such an approach, Figure 4 shows results based upon the "goodness" of the hashing function $\Gamma$ for 20 groups. The solid curve indicates random hashing with the three other curves representing different levels of goodness. The $P = 0$ curve is the worst case meaning that each new resource added at a machine maps to a previously unused multicast address if one exists. The other extreme is the ideal $P = 100$ curve, which means that each new resource always maps to a previously unused multicast address if a previously used address is not filled (assuming $100/20 = 5$ resources per address). The $P = 50$ curve indicates a 50% chance of mapping to a new group if one exists. Although such hashing functions can be hypothesized, finding one, particularly an ideal one that is good for all resources on all machines is difficult, unless some order exists among how the resources are distributed.

The use of a group mapping function has also been investigated in other work. Ahamed, et al [1] outline a procedure for managing distributed resources over a set of $L$ nodes (assumed not to fail). In their work they assume that each machine can maintain up to $M$ multicast addresses. Each resource is then hashed into one of $K \leq L \cdot M$ (rather than $M$) distinct groups. When a machine contains resources mapping into more than $M$ addresses the resource information, not the resource, is moved to another node. Because of the invariant that $K \leq L \cdot M$, it is guaranteed that another node can be found which already has the address group or can add another address group. In addition to the possible network overhead for adding resources, two types of problems can arise if machines do crash. By separating the resource from information about the resource it is possible for the resource machine to go down while information about the resource (available from another machine) may indicate the resource is available. The other, more problematic, case is that a resource may be available, but the host containing information about the resource may have crashed. Additional overhead must be incurred to handle these cases.

5.3 Merging of Unique and Group Mapping Functions

The advantage of limiting the group hash function to $M$ groups is that information about a resource is maintained by machines that actually contain the resource. However, the problem with any group hashing function is that it may map just a few resources contained at a machine into different multicast addresses, thus incurring the cost of receiving mismatched requests for information. On the other hand the unique mapping function $T$ can only be used if the number of resources at a host is always less than the number of supported multicast addresses $M$. If the number of resources grows larger then resources become lost.

To allow for handling a few resources efficiently, and still be able to handle any number of resources at a machine we postulate a two-tiered approach that combines the two previous approaches. In this approach, we use the unique mapping function $T$ as long as the number of resources at a node is less than $M$. Once the number of resources increases beyond $M$ then all previous multicast addresses are flushed and the multicast addresses for a machine are computed based upon the group mapping function $\Gamma$. This strategy is used by the $\text{AddResource}(r)$ and $\text{DeleteResource}(r)$ routines. It is possible that a server process will switch between these two mapping functions as the number of resources changes.

The principal change is that the requestor of information must now send two requests. One request to the address computed by the unique mapping function $T$, and one request to the address computed by the group mapping function $\Gamma$. However because at any time a server process will be using only one of the mapping functions, it is guaranteed that a server
Figure 3: Group Hash Function (Varying Number of Groups)

Figure 4: Varying Goodness of Group Hash Functions (20 Groups)
process will receive no more than one request.

Figure 5 shows the expected results of this strategy for \( M = 10, 20, \) and 50, compared against a single multicast address. As shown, if fewer than \( M \) resources are contained by a machine then the number of misses is zero (or near zero), while this mechanism can still accommodate any number of resources contained by an individual machine. In all cases the percentage of misses is fewer than for a single address or for using only the group hash function. The increased cost of this approach is that the client has to send two packets addressed to different multicast addresses for each request. In an environment where the scope of the requests is limited and network traffic is not a problem, this strategy is the most effective at relieving the burden of unwanted interruptions on the server machines.

6 Applications

Given these strategies and basic analysis on the use of multicasting for locating resources in a distributed environment, there are many directions for future work. Some of this work is currently underway using the IP multicasting software as a basis on which to build applications.

User Location Problem. One type of resource is the set of users logged onto a particular machine. The “user location problem” can be described as follows.

I am a user logged into some machine in the network. I wish to locate another user who may or may not be logged into one or more machines in the network. If a user is logged into a machine, I want to know how long they have been idle. If the user is not logged into a machine, I want to know when the user last logged out.

As networks grow with more and more single-user workstations and perhaps a few timeshared machines, this problem appears a good match for use with the two-tiered multicast strategy. Almost all machines will contain just a few resources while a few of the timeshared machines may still support many users. In addition, the information itself changes frequently in relation to the frequency that it is requested so it is advantageous to keep the information at its source.

Service Location Problem. Related to previous work we have done on the use of computational services in a distributed environment [18], we plan to examine the use of multicasting for locating such services. This problem is different than the user location problem because the services that are offered by a machine change slowly while requests for information are more frequent. These parameters may push us to another solution for managing the information.

Use of Central Name Servers. The strategies we proposed for use of multicasting all assume the information to be stored at the source of the information and retrieved upon request. Another direction of work is to explore when it is appropriate to use central name servers to store the information, and how to do so with multicasting. One problem with multicasting for information to all machines is that a client does not know how many responses to expect. With a central name server, which contains all information, only one response would be needed. Central servers can also respond better to requests containing wildcards and partial names. Sending to multicast, rather than machine, addresses could be used to reduce administrative overhead as central servers are moved between machines.

Self-Adapting Information Servers. An interesting approach is to examine servers, which adapt to the nature of the information in managing it. Based on how often the information is updated, how often it is accessed, its distribution, and other factors, the servers themselves would move resource information around while clients would only need to send requests to the appropriate multicast address for the resource in order to retrieve information. This idea is related to the previous one in that servers may organize themselves to maintain all information at just a few nodes.

Attribute-Based Addresses. Not only can multicasting be used to locate resources by name, but we can also look at it to find resources by attribute. For example, by associating a multicast address with a certain machine type or with machines having active users. This idea can be extended to allow combinations of attributes to be embedded into addresses. For example, all active users on workstations in a particular room. A practical problem with this approach may be that more attribute-based addresses are generated than can be supported by the multicast hardware and software. In such cases a two-tiered approach as discussed in this paper may be used.

Load Sharing. Related to attribute-based addresses is to use multicasting for finding machines with particular load attributes. This idea seems promising to avoid interrupting machines that are already busy for queries about their load. In their work on load sharing, Eager, Lazowska, and Zahorjan [10] investigate the use of probing machines to find one suitable for executing
a computation. With the availability of multicasting we can envision servers that register with a multicast address based upon their load. A client seeking to find a machine would then send a probe to the address for nonloaded machines, and if no response is received, then lightly loaded machines, and so forth. This approach allows lightly loaded machines to be found without ever bothering machines that are already busy.

7 Summary

The central goal of our resource location strategies is to dynamically compute multicast addresses based on the resource name to reduce, if not eliminate, the number of resource requests received, but not honored, by a machine. The three strategies we examined each are an improvement on broadcasting in this regard. If the number of resources at a specific node is small then use of a good hashing function with a large multicast address space can basically eliminate any mismatched requests from reaching a server process. By combining this strategy with a second strategy of mapping all resources into a limited range of multicast addresses, we were able to develop a strategy that is efficient for a small number of resources, yet can handle any number of resources at a machine.

The basic idea of our approach is to embed as much information about the resources contained by a machine into the multicast addresses supported by the machine so as many requests as possible can be filtered out by the network interface. One question to be addressed with further research is how the rate and cost to modify the set of multicast addresses compares with the processing saved from fewer interrupts. Another consideration is the scalability, beyond a local area network, of a strategy where multicast group membership may change frequently. In this case the cost of maintaining group membership over a wide area network may become expensive. Further work will indicate what type of resource location problems are appropriate to attack with the dynamic multicast address strategies we have discussed in this paper.
References


