Scalable application-level anycast for highly dynamic groups

Miguel Castro[1] Peter Druschel[2] Anne-Marie Kermarrec[1] Antony Rowstron[1]

mcastro@microsoft.com *druschel@cs.rice.edu* *annemk@microsoft.com* *antr@microsoft.com*

[1]Microsoft Research, 7 J J Thomson Avenue, Cambridge, CB3 0FB, UK.
[2]Rice University, 6100 Main Street, MS-132, Houston, TX 77005, USA.

Abstract

We present an application-level implementation of anycast for highly dynamic groups. The implementation can handle group sizes varying from one to the whole Internet, and membership maintenance is efficient enough to allow members to join for the purpose of receiving a single message. Key to this efficiency is the use of a proximity-aware peer-to-peer overlay network for decentralized, lightweight group maintenance; nodes join the overlay once and can join and leave many groups many times to amortize the cost of maintaining the overlay. An anycast implementation with these properties provides a key building block for peer-to-peer applications. In particular, it enables management and location of dynamic resources in large scale peer-to-peer systems. We present several resource management applications that are enabled by our implementation.

1 Introduction

Anycast [10] is a service that allows a node to send a message to a nearby member of a group, where proximity is defined using a metric like IP hops or delay. It is a useful building block in many distributed applications. For example, it can be used to manage dynamic distributed resources (e.g., processing, storage, and bandwidth). This can be achieved using the following pattern. First, anycast groups are created for each resource type. Nodes join the appropriate group to advertise the availability of a resource and leave the group when they no longer have the resource available. Nodes request nearby resources by anycasting messages to the appropriate groups.

We are particularly interested in the application of anycast to manage distributed resources in decentralized peer-to-peer systems. This is a challenging application because groups can vary in size from very small to very large and membership can be highly dynamic.

Previous anycast proposals do not support this application. Network level anycast (e.g., [8]) does not work well with highly dynamic groups and its deployment has been hampered by concerns over security, billing, and scalability with the number of groups. Previous application-level anycast proposals [1, 7] are easy to deploy but assume that groups are small and that membership is relatively stable. We present an application-level implementation of anycast that can be used as a bulding block in large scale peer-to-peer applications.

We implemented anycast as an extension of Scribe [14, 4]. Scribe builds a proximity-aware spanning tree for each group. It supports both small and very large groups, and it allows nodes to join and leave a group with low overhead. The trees are used to anycast messages efficiently: messages are delivered to a nearby group member with low delay and link stress. This enables very fine-grained resource management in large scale distributed systems.

The key to the efficiency of our anycast implementation is that group trees are embedded in a structured, proximity-aware overlay network. When a node joins a group, it merely routes towards the group spanning tree in the overlay, adding overlay links to the tree as needed. Thus, membership maintenance is fully distributed and the overhead of maintaining the proximity-aware overlay network can be amortized over many different group spanning trees (and other applications). This is the key to lightweight membership management and the resulting ability to support large, highly dynamic groups.

We show a number of uses of our anycast implementation to manage resources in real applications. SplitStream, for instance, is a peer-to-peer streaming content distribution system that stripes content across multiple multicast trees [3]. This striping achieves increased robustness to node failures, spreads the forwarding load across all participating nodes, and allows SplitStream to accomodate participating nodes with widely differing network bandwidth. It uses anycast to locate nodes with spare forwarding capacity.

The rest of this paper is organized as follows. In Section 2, we explain how Scribe manages group membership and builds spanning trees. In Section 3, we show how to implement anycast and manycast using these trees. Section 4 discusses applications of our anycast implementation. We present some preliminary performance results in Section 5. Related work is described in Section 6, and we conclude in Section 7.

2 Group management

We use Scribe [14, 4] to manage groups and to maintain a spanning tree connecting the members of each group. Scribe can handle group sizes varying from one to the whole Internet, and it supports rapid changes in group membership efficiently. All nodes join Pastry, a structured peer-to-peer overlay. Subsequently, nodes may join and leave potentially many groups many times to amortize the cost of building the Pastry overlay. This, combined with Pastry's already scalable algorithms to join the overlay is the key to Scribe's ability to support highly-dynamic groups efficiently.

Per-group trees are embedded in the Pastry overlay, and are formed as the union of the overlay routes from the group members to the root of the tree. In addition to supporting anycast, group trees can be used to multicast messages to the group using reverse path forwarding [6] from the root. Multicast achieves low delay and induces low link and node stress.

Scribe's approach to group management could be implemented on top of other structured peer-to-peer overlay networks like CAN [11], Chord [16], or Tapestry [19]. In fact, Scribe has been implemented on top of CAN [5].

Next, we describe in more detail Pastry and the construction of group trees.

2.1 Peer-to-peer overlay

Pastry is described in [12, 2]. Here we provide a brief overview to understand how we implement group management and anycast.

In Pastry, *keys* represent application objects and are chosen from a large space. Each participating node is assigned an identifier (*nodeId*) chosen randomly with uniform probability from the same space. Pastry assigns each object key to the live node with nodeId numerically closest to the key. It provides a primitive to send a message to the node that is responsible for a given key. The overlay network is self-organizing and self-repairing, and each node maintains only a small routing table with $O(log(n))$ entries, where n is the number of nodes in the overlay. Each entry maps a nodeId to its IP address. Messages can be routed to the node responsible for a given key in $O(log(n))$ hops. Additionally, Pastry routes have two properties that are important for efficient anycast: *low delay stretch* and *local route convergence*. Simulations with realistic network topologies show that the delay stretch, i.e., the total delay experienced by a Pastry message relative to the delay between source and destination in the Internet, is usually below two [2]. These simulations also show that the routes followed by messages sent from nearby nodes in the network topology towards the same destination converge on a nearby node after a small number of hops.

2.2 Group trees

Scribe is described in detail in [4]. We provide a brief overview in order to explain how anycast works.

Scribe uses Pastry to name a potentially large number of groups. Each group has a key called the *groupId*, which can be the hash of the group's textual name concatenated with its creator's name. To create a group, a Scribe node asks Pastry to route a CREATE message using the groupId as the key. The node responsible for that key becomes the root of the group's tree.

To join a group, a node routes a JOIN message through the overlay to the group's groupId. This message is routed towards the root of the tree by Pastry. Each node along the route checks whether it is already in the group's tree. If it is, it adds the sender to its children table and stops routing the message any further. Otherwise, it creates an entry for the group, adds the source node to the group's children table, and sends a JOIN message for itself. The properties of overlay routes ensure that this mechanism produces a tree.

To leave a group, a node simply records locally that it left the group. If there are no other entries in the children table, it sends a LEAVE message to its parent in the tree. The message proceeds recursively up the tree, until a node is reached that still has entries in the children table after removing the departing child. Scribe's approach to tolerate node failures, including root failures, and a couple of optimizations are described in [4].

Group management scales well because Pastry ensures that the trees are well balanced and join and leave requests are localized and do not involve any centralized node. For example, join requests stop at the first node in the route to the root that is already in the group tree. This property also holds when Scribe is implemented on top of CAN, Chord, or Tapestry.

The local route convergence property of Pastry also helps group management scalability because join and leave requests tend to traverse a small number of physical network links. Similarly, the low delay stretch property results in low latency for joins. Scribe would retain these properties when implemented on top of Tapestry but not when implemented on top of the other overlays.

3 Anycast and manycast

Anycast allows a node to send a message to a member of a group [10]. In proximity-aware overlays like Pastry [2] and Tapestry [19], the message will be delivered to one of the closest group members in the network topology. The sender does not need to be a member of the group.

Anycast is implemented using a distributed depth-first search of the group tree. This is efficient and scalable: searches complete after visiting a small number of nodes ($O(log(n))$ where n is the number of elements in the overlay), and load is balanced by starting searches for different senders at different nodes.

To anycast a message to a group, the sender routes the message towards the group's identifier using the overlay. The message is forwarded until it reaches the first node in the route that is already in the tree. The depth-first search of the tree starts with this node. The search is fairly standard except that children edges are explored before parent edges and applications can define the order in which children edges are explored. For example, the order can be random or round-robin to improve load balancing, or it can use some application specific metric like an estimate of the load on each child.

We append the identifiers of the nodes that have already been visited to the message to guide the search. After all of a node's children have been visited (if any), the node checks if it is in the group. If so, the message is delivered to the node and the search ends. Otherwise, the search continues after the node adds its identifier to the list of visited nodes in the message and removes the identifiers of its children to keep the list small.

It is possible for the group to become empty while a search is in progress. This is detected when the root is visited after all its children have been visited and it is not in the group. In this case, an error message is sent back to the sender.

The search is efficient and scalable. In the absence of membership changes during the search, it completes after exploring a single path from the start node to a leaf. Our implementation uses the overlay properties to balance the load imposed by searches on the nodes that form the group tree: different senders are unlikely to explore the same path unless the group is very small. Therefore, our approach scales to a large number of concurrent senders.

In overlays with the local route convergence property (Tapestry and Pastry), the nodes in the subtree rooted at the start node are the closest in the network to the sender, among all the group members. Therefore, the message is delivered to a group member that is close to the sender in the network topology. Additionally, this leads to low link stress and delay for anycast routing.

We can also support *manycast*, i.e., the message can be sent to several group members. This is done by continuing the search until enough satisfying members are found and tracking the number found in the message.

Both anycast and manycast can be augmented with predicates such that the message is delivered to a group member that satisfies a predicate. The algorithm above works efficiently when most group members satisfy the predicate. If this is not the case, the group can be split in subgroups such that most of the elements in each subgroup satisfy a predicate. The overlay can be used to name and locate the group that should be used for a given predicate. Additionally, nodes can leave the group when they stop satisfying the anycast predicate associated with the group.

4 Distributed resource management

Existing anycast systems have been limited to small and relatively static sets of group members; accordingly, applications have been limited to tasks like server selection. Enabling anycast for highly dynamic groups whose size and membership change rapidly enables a much larger set of applications.

Applications can exploit such an anycast facility to discover and manage a much larger quantity of resources at much smaller time scales. For example, a group can be created whose members advertise some spare resource, e.g., storage capacity, bandwidth, or CPU cycles. They join the group in order to advertise availability of the resource; when they receive an anycast message from a node that seeks to consume the resource, they assign some resource units to that node. They leave the group when they no longer have any resource units available.

We next discuss some example application scenarios.

CPU cycles: Many workstations are idle for significant periods of the day, and many systems offer their spare CPU cycles for compute intensive tasks, e.g., Condor [9]. Such systems need to match spare CPU resources to jobs that need to be executed.

One approach to schedule such jobs is to create a group consisting of nodes that have pending compute jobs. Essentially, such a group acts as a job queue. Nodes with spare CPU cycles anycast to the group, thus matching nodes with spare cycles to nodes with pending jobs. When a node has no jobs remaining it removes itself from the group. A second approach is to create a group whose members are nodes with spare CPU cycles. A node with pending jobs anycasts to the group to find a node with spare compute cycles.

The first approach works best when demand for CPU cycles is greater than supply and the second works best when the reverse is true. The two approaches can be combined by using two groups rooted at the same node.

Storage capacity: Storage systems often need to locate a node that has the spare capacity to store a replica of an object. For example, in an archival file storage system like PAST [13] a node with high storage utilization may wish to find an alternate node to store a replica on its behalf. In such a system, a group can be created consisting of nodes with spare disk capacity. A node seeking an alternate node to store a replica anycasts to the group.

Furthermore, a group can be created that consists of the nodes that store a given object. Nodes that wish to access the object anycast to the group to locate a nearby replica. The same group can also be used to multicast updates or coherence messages in order to keep the replicas consistent.

Bandwidth: Similar techniques can be used to locate nodes with spare network bandwidth. SplitStream [3], for instance, is a peer-to-peer content streaming system that stripes content over multiple multicast groups. This striping allows SplitStream to spread the forwarding load across the participating nodes, takes advantage of nodes with different network bandwidths, and yields improved robustness to node failures. Nodes with spare network bandwidth join a spare capacity group.

A node tries to join the groups associated with each stripe. If its potential parent for a particular stripe has no spare bandwidth, the node anycasts on the spare capacity group, to find a nearby parent that can adopt it. The spare capacity group initially contains all the nodes.

5 Preliminary results

We performed a preliminary evaluation of our anycast implementation using simulations of a Pastry overlay with 100,000 nodes on a transit-stub topology [18] with 5050 routers. Each router had a LAN and each Pastry node was randomly assigned to one of these LANs.

We started by measuring the cost of creating 10 different anycast groups varying in size from 2,500 to 25,000 nodes. All group members joined at the same time. The average number of messages per join was 3.5 with 2,500 members and 2.3 with 25,000 members. The average cost per join decreases with the group size because the number of hops before a JOIN message reaches the group tree decreases. The cost of joining an empty group is approximately 4.1 messages. The joining load was evenly distributed, e.g., 83% of the nodes processed a single message when creating the largest group.

Then, we simulated 1,000 anycasts from random sources to each group with the predicate *true*. The average number of messages per anycast was 2.9 with 2,500 members and 2.1 with 25,000 members. The average anycast cost decreases with the group size for the same reason that the join cost decreases. The cost of anycasting to an empty group is also approximately 4.1 messages. The load to process anycasts was also evenly distributed, e.g., 99% of the nodes that processed an anycast in the group with 25,000 members only processed one or two messages and none processed more than 4.

To evaluate whether our implementation delivers an anycast to a nearby group member, we measured the network delay between the sender and the receiver of each anycast and ranked it relative to the delays between the sender and other group members. For 90% of the anycasts to the group with 2,500 members, less than 7% of the group members were closer to the sender than the receiver. The results improve when the group size increases because the depth first search of the tree during the anycast is more likely to start at a node closer to the sender. Less than 0.1% of the group members were closer to the sender than the receiver for 90% of the anycasts to the group with 25,000 members.

6 Related work

Related work includes IP and application-level anycast, and global resource management systems. The main contribution of our approach is the ability to perform anycast efficiently in large and highly dynamic groups.

Anycast was first proposed in RFC 1546 [10]. GIA [8] is an architecture for global IP-anycast that is scalable but shares the drawbacks characteristic of network-level approaches: it requires router modifications, and it cannot exploit application-level metrics to select the destination of anycast messages.

The approach to application-level anycast proposed in [1, 7] builds directory systems. Given a query, the service returns the unicast address of the closest server satisfying the query. The approach is similar to Round Robin DNS. It assumes a small and static number of servers per group and would not scale to large-scale highly dynamic groups.

The Internet Indirection Infrastructure (I3) [15] proposes a generic indirection mechanism supporting unicast, multicast, anycast and, mobility. I3 uses *triggers* to represent indirection points between senders and receivers. A trigger is represented by a unique id. Receivers subscribe to triggers, and senders publish to triggers. When a message arrives at a trigger, it is forwarded to one or more of the receivers registered at the trigger. The current implementation uses Chord to map triggers to nodes. I3 does not currently provide a scalable membership mechanism for triggers supporting multicast and anycast. Our groups provide an indirection similar to triggers with a scalable, efficient implementation.

Astrolabe [17] provides a very flexible distributed resource management system. It supports generic queries on the state of the system using gossiping and it limits the rate of gossiping to achieve better scalability at the expense of increased delays. The increased delays make it unsuitable to manage dynamic resources at very small time scales.

7 Conclusions

The paper presented an anycast implementation that supports large and highly dynamic groups. We argued that the properties of our system enable management and discovery of resources in large scale distributed systems at much smaller time scales than previously possible. The properties of our system derive in part from Scribe's decentralized approach to group membership management. Scribe is built on top of Pastry, a scalable, proximity-aware peer-to-peer overlay and it supports multicast as well as anycast. Finally, we discussed applications that use our anycast implementation to schedule jobs, manage storage, and manage bandwidth for content distribution. These applications show that our anycast implementation provides a useful building block to construct large-scale peer-to-peer applications.

References

[1] S. Bhattachargee, M. Ammar, E. Zegura, N. Shah, and Z. Fei. Application layer anycasting. In *Proc IEEE Infocom'97*, 1997.

[2] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron. Exploiting network proximity in peer-to-peer overlay networks, 2002. Technical report MSR-TR-2002-82.

[3] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth content distribution in a cooperative environment, 2002. Submitted to IPTPS'03.

[4] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE JSAC*, 20(8), Oct. 2002.

[5] M. Castro, M. Jones, A.-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang, and A. Wolman. An evaluation of scalable application-level multicast built using peer-to-peer overlay networks. In *Proc. of INFOCOM'03*.

[6] Y. K. Dalal and R. Metcalfe. Reverse path forwarding of broadcast packets. *Communications of the ACM*, 21(12):1040–1048, 1978.

[7] Z. Fei, S. Bhattachargee, M. Ammar, and E. Zegura. A novel server technique for improving the response time of a replicated service. In *Proc IEEE Infocom'98*, 1998.

[8] D. Katabi and J. Wroclawski. A Framework for Scalable Global IP-Anycast (GIA). In *Proc SIGCOMM'00*, 2000.

[9] M. Mutka and M. Livny. Scheduling remote processing capacity in a workstation-processing bank computing system. In *Proc. of ICDCS'87*, 1987.

[10] C. Partridge, T. Menedez, and W. Milliken. Host anycasting service. In *RFC 1546*, November 1993.

[11] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *Proc. SIGCOMM'01*, Aug. 2001.

[12] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. Middleware'01*, 2001.

[13] A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Proc. SOSP*, Oct. 2001.

[14] A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. Scribe: The design of a large-scale event notification infrastructure. In *Proc. NGC'01*, Nov. 2001.

[15] I. Stoica, D. Adkins, S. Ratnasamy, S.Shenker, S. Surana, and S. Zhuang. Internet indirection infrastructure. In *Proc of ACM SIGCOMM*, 2002.

[16] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. SIGCOMM'01*, 2001.

[17] R. van Renesse and K. Birman. Scalable management and data mining using astrolabe. In *IPTPS '02*, 2002.

[18] E. Zegura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. In *INFOCOM96*, 1996.

[19] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-resilient wide-area location and routing. Technical Report UCB//CSD-01-1141, U. C. Berkeley, Apr. 2001.