Scalable application-level anycast for highly dynamic groups

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

[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 distributed 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 [12] 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. In this paper, we are interested in the application of anycast to manage distributed resources in decentralized peer-to-peer systems. This is a challenging application for anycast because groups can vary in size from very small to very large and membership can be highly dynamic.

We note that resource discovery based on anycast is a powerful building block in many distributed systems. For example, it can be used to manage dynamic distributed resources (e.g., processing, storage, and bandwidth) 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.

Unfortunately, existing anycast proposals do not support this application. Network level anycast (e.g., [10]) does not work well with highly dynamic groups and its deployment has been hampered by concerns over security, billing, and

* Supported in part by Texas ATP (003604-0079-2001) and by NSF (ANI-0225660).

scalability with the number of groups. Previous application-level anycast proposals [1, 9] 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 powerful building block in large scale peer-to-peer applications.

Our anycast system builds a per-group proximity-aware spanning tree. It efficiently supports both small and very large groups. Moreover, it supports highly dynamic groups, because nodes join and leave a group with low overhead and all membership maintenance is decentralized. The spanning 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. The tree is the union of the paths from the group members to a node acting as the root of the tree. When a member joins a group, it merely routes a join request towards the root of the tree using the overlay, adding overlay links to the tree as needed. When the join request reaches a node that is already a member of the tree, the request is not propagated any further. Thus, membership maintenance is fully distributed. The same overlay network can support many different independent groups, with the advantage that 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 and 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 [4]. This striping achieves increased robustness to node failures, spreads the forwarding load across all participating nodes, and allows SplitStream to accommodate participating nodes with widely differing network bandwidth. It uses anycast to locate nodes with spare network bandwidth.

The contributions of this paper are to describe a scalable application-level implementation of anycast for highly dynamic groups, and to articulate its power as a tool for resource management in distributed systems.

The rest of this paper is organized as follows. Section 2 explains group membership management and spanning tree construction. 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 performance results in Section 5. Related work is described in Section 6, and we conclude in Section 7.

2 Group management

We use Scribe [18, 6], an application-level multicast system, to manage groups and to maintain a spanning tree connecting the members of each group. Scribe can handle group sizes varying from one to millions, and it supports rapid changes in group membership efficiently. All nodes join Pastry, a structured peer-to-peer overlay [16, 3]. 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 [6].

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 [8] from the root. It has been shown that Scribe multicast achieves low delay and induces low link and node stress [6].

Scribe's approach to group management could be implemented on top of other structured peer-to-peer overlay networks like CAN [14], Chord [20], or Tapestry [23]. In fact, Scribe has been implemented on top of CAN [7].

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

2.1 Peer-to-peer overlay

Pastry is fully described in [16, 3]. Here, we provide a brief overview to understand how we implement group management and anycast.

In Pastry, numeric *keys* represent application objects and are chosen from a large identifier 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. The routing table maintained in each Pastry node is created and maintained in a manner that enables Pastry to exploit network locality in the underlying network. There are two particular properties of Pastry routes 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 underlying network, is usually below two [3]. These simulations also show that the routes for messages sent to the same key from nearby nodes in the underlying network converge at a nearby node after a small number of hops.

2.2 Group trees

Scribe uses Pastry to name a potentially large number of groups. Each group has a key called the *groupId*, which could 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 the Pastry overlay routes ensure that this mechanism produces a tree.

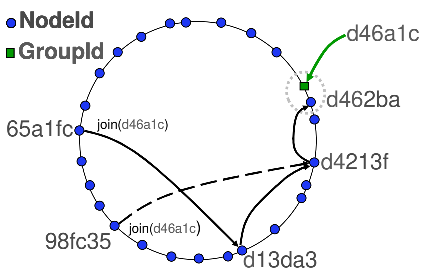

Fig. 1. Example formation of a group tree.

Figure 1 demonstrates how the trees are built using an example tree with the groupId $d46a1c$. Node $d462ba$ acts as the root for the tree, as it has the numerically closest nodeId to the groupId. Initially node $65a1fc$ routes a JOIN request using the groupId. Pastry routes the request via node $d13da3$, which takes node $65a1fc$ as a child and then forwards the request towards the groupId. Node $d4213f$ receives the request, takes $d13da3$ as a child and forwards the request. Node $d462ba$ receives the request, and takes $d4213f$ as a child. As it is the root for this group tree it does not forward the request. Subsequently, node $98fc35$ joins the group. It routes a JOIN using the groupId. Node $d213f$ receives the request and takes $98fc35$ as a child. As it is already a member of the tree the request is not forwarded to any other nodes.

If a node wishes to leave a group and it has entries in its children table for that group then it marks itself as no longer a member and does nothing else. As it has entries in its children table it is required to receive and forward messages for the group. If the departing node has no entries in its children table then it sends a LEAVE message to its parent in the tree. Upon receiving a LEAVE message the parent node removes the child node from its children table. If the parent node subsequently has an empty children table and it is not a member of the group, it sends a LEAVE message to its parent. This process continues recursively up the tree until a node is reached that still has entries in the children table after removing the departing node, or the root is reached. Scribe's mechanisms to tolerate node failures, including root failures, and several optimizations are described in [6].

Group management scales well because Pastry ensures that the trees are well balanced and that join and leave requests are localized and do not involve any centralized node. For example, join requests stop at the first node on 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 only when implemented on top of other proximity-aware structured overlays, for example Tapestry.

3 Anycast and manycast

Our anycast system allows a node to send a message to a member of a group. It relies on Scribe to create a Scribe tree for the group. If Scribe uses a proximity-aware overlay like Pastry [3] or Tapestry [23], the message will be delivered to one of the closest group members in the underlying 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, which is built as described in Section 2.2. This is efficient and scalable: anycasts 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 because anycasts from different senders start at different nodes.

Each group has an associated *groupId*. To anycast a message to a group, a sender routes a message through Pastry using the groupId as the key. The message is routed towards the node that acts as the root of the group tree. At each hop, the local node checks whether it is part of the tree for the specified groupId. If it is, then the anycast message is not forwarded; instead, a depth-first search of the group tree is initiated from that 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) without success, the node itself checks if it is in the group. If so, the message is delivered to the node and the search terminates. Otherwise, the anycast continues after the node adds its identifier to the list of visited nodes in the message. The identifiers of its children are then removed to keep the list small.

It is possible for the group to become empty while an anycast 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 anycast until the message has been delivered to the desired number of members, tracking the number of recipients in the message.

Both anycast and manycast can be augmented such that the message is delivered only to group members that satisfy a given 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 given 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 predicate associated with the group.

Our anycast and manycast primitives can be secured from malicious participating nodes in the overlay, by extending earlier work on secure routing in Pastry [2]. However, the details are beyond the scope of this paper.

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 [11]. 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 [17] 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 [5, 4], 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 default parent for a particular stripe has no spare bandwidth, the node anycasts to the spare capacity group, to find a nearby parent that can adopt it. The spare capacity group initially contains all the nodes.

In general, applications may customize the anycast primitive by controlling the DFS tree traversal order, by providing a boolean predicate on attributes of group members, and by specifying the desired number of receivers. Moreover, the low overhead of group creation and membership operations enables the use of many, potentially overlapping groups to increase the efficiency of complex queries. For instance, in addition to joining a group that indicates available storage capacity, nodes with free storage can join groups associated with more fine-grained attributes, such as a range of available storage, the expected availability of the storage node or the node's query load. Anycast queries can then be directed towards specific groups whose members are known or likely to satisfy certain attribute values, thus increasing efficiency.

5 Experimental results

We performed an evaluation of our anycast implementation using simulations of a Pastry overlay with 100,000 nodes on a transit-stub topology [22] with 5050 routers. Each router had an attached 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 four messages.

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 0.15% 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 [12]. GIA [10] 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, 9] 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 does not scale to large and highly dynamic groups.

Like Scribe, Bayeux [24], and CAN-Multicast [15] are group communication systems built on top of structured peer-to-peer overlays, namely Tapestry [23] and CAN [14]. Neither Bayeux nor CAN-multicast support an anycast primitive.

While such a primitive could be added, neither protocol can support highly dynamic groups efficiently. CAN-multicast builds a separate overlay for each group, which makes joining and leaving a group relatively expensive. In our approach, the same overlay is used to host many different groups, thus amortizing the cost of maintaining the overlay.

Like Scribe, Bayeux builds per-group multicast trees on top of a peer-to-peer overlay. In Bayeux, however, a tree is the union of the overlay routes from the root to each member. This forces Bayeux to forward each join and leave request to the root, thus preventing it from handling large, dynamic groups efficiently.

The object location approach used in Tapestry and in Plaxton's work [13] can be viewed as a special case of our anycast mechanism. In these systems, the tree formed by the overlay routes from each replica holder of an object to the object's root is annotated with pointers to the nearest replica holder. Lookup messages are routed towards an object's root; when the message intercepts the tree, it uses the replica pointer to retrieve a copy of the object.

Tapestry and Plaxton use this mechanism only to locate objects. Since the trees are not fault-tolerant, objects must be periodically reinserted. Our system supports a fully general anycast primitive, supports predicate-based anycast, and maintains the group membership despite node failures. Moreover, we articulate the power of such a primitive for dynamic resource management in large-scale distributed systems.

The Internet Indirection Infrastructure (i3) [19] 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. In its simplest form, the i3 group membership mechanism is not scalable, because a single node maintains all members of a group. To support large groups, i3 uses a hierarchy of triggers. However, i3 has to explicitly construct and balance these trees of triggers, while in our approach, the spanning trees are formed naturally by the existing overlay routes from the group members to the group's root. Thus, our approach can handle large, highly dynamic groups with greater efficiency.

Astrolabe [21] 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 for managing dynamic resources at very small time scales.

7 Conclusions

This paper presents an efficient anycast implementation that supports large and highly dynamic groups. The properties of our anycast system enable management and discovery of resources in large-scale distributed systems at much smaller time scales than previously possible. Our systems takes a decentralized approach to group membership management, and amortizes the cost of maintaining a single overlay network over many groups and group membership changes. The system is built on top of Pastry, a scalable, proximity-aware peer-to-peer overlay and it supports multicast as well as anycast. Finally, we discuss applications that can use our anycast implementation to schedule jobs, manage storage, and manage bandwidth for content distribution. These applications demonstrate that our anycast implementation provides a powerful 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, A. Ganesh, A. Rowstron, and D. Wallach. Secure routing for structured peer-to-peer overlay networks. In *Proc. OSDI'02*, Dec. 2002.
3. 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.
4. M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth content distribution in a cooperative environment. In *SOSP'03*, 2003.
5. M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth content distribution in a cooperative environment. In *IPTPS'03*, February 2003.
6. 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.
7. 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*, 2003.
8. Y. K. Dalal and R. Metcalfe. Reverse path forwarding of broadcast packets. *Communications of the ACM*, 21(12):1040–1048, 1978.
9. 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.
10. D. Katabi and J. Wroclawski. A Framework for Scalable Global IP-Anycast (GIA). In *Proc SIGCOMM'00*, 2000.
11. M. Mutka and M. Livny. Scheduling remote processing capacity in a workstation-processing bank computing system. In *Proc. of ICDCS'87*, 1987.
12. C. Partridge, T. Menedez, and W. Milliken. Host anycasting service. In *RFC 1546*, November 1993.
13. C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proc. 9th ACM Symp. on Parallel Algorithms and Architectures*, pages 311–320, June 1997. Newport, Rhode Island, USA.
14. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *Proc. SIGCOMM'01*, Aug. 2001.
15. S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level multicast using content-addressable networks. In *Proc. of NGC'01*, Nov. 2001.

16. A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. Middleware'01*, 2001.
17. 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.
18. 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.
19. I. Stoica, D. Adkins, S. Ratnasamy, S.Shenker, S. Surana, and S. Zhuang. Internet indirection infrastructure. In *Proc of ACM SIGCOMM*, 2002.
20. 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.
21. R. van Renesse and K. Birman. Scalable management and data mining using Astrolabe. In *IPTPS '02*, 2002.
22. E. Zegura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. In *INFOCOM96*, 1996.
23. 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.
24. S. Zhuang, B. Zhao, A. Joseph, R. Katz, and J. Kubiatowicz. Bayeux: An Architecture for Scalable and Fault-tolerant Wide-Area Data Dissemination. In *Proc. NOSSDAV'01*, June 2001.