Internet Indirection Infrastructure*

Ion Stoica, Dan Adkins, Sylvia Ratnasamy, Scott Shenker, Sonesh Surana, Shelley Zhuang
{*istoica, dadkins, sylviar, sonesh, shelleyz*}*@cs.berkeley.edu, shenker@icsi.berkeley.edu*

1 Introduction

Today's Internet is designed around the point-to-point communication abstraction. This simple abstraction is one of the main reasons behind the scalability and the efficiency of the Internet. However, as the Internet evolves into a global communication infrastructure, there is an increasing need to implement other communication primitives such as multicast and anycast, and to support end-host mobility. Unfortunately, despite years of intense research, these services have yet to be deployed in the Internet. The main difficulty resides in that the point-to-point communication abstraction – which assumes one sender and one receiver placed at well-known and fixed network locations – is *not* appropriate for these services. For example, mobility requires one to remove the assumption that end-hosts are fixed, multicast requires one to remove the assumption that there is only one sender and one receiver, and anycast requires one to remove the assumption that the receiver's location is known.

To get around this problem, existing solutions use a simple but powerful technique: *indirection*. These solutions assume a physical or a logical indirection point interposed between the sender and the receiver(s) that relays the traffic between them. By communicating through the indirection point rather than directly to the end-host, a sender can abstract away the location and the number of receivers. For instance, mobile IP assumes a home agent that hides the end-host mobility, while IP multicast assumes a logical indirection point (address) that hides the number of receivers and their locations.

While indirection can enable these services, efficiently implementing them at the IP layer has proven difficult. Existing IP multicast solutions [1, 5, 7, 8, 12] scale poorly with the number of groups, as every router has to maintain state for each group whose distribution tree passes through the router. Furthermore, because the amount of resources consumed by a multicast session is difficult to predict, ISPs have little incentive to enable the multicast service and carry the multicast traffic of other ISPs. Mobile IP [16, 17] suffers from inefficient routing, as each packet to the mobile host is forwarded through a home agent which is physically attached to the mobile host's home address. In addition, routers that implement ingress filtering [9] will not forward the packets sent by a mobile host in a foreign network. To get around these problems several solutions have been proposed to deploy these services at the application level [4, 14, 20]. While these proposals achieve the desired functionality, they do so in a very disjointed fashion in that solutions for one service are not solutions for other services; *e.g.*, proposals for application-layer multicast don't address mobility, and vice-versa. As a result, many similar and largely redundant mechanisms are required to achieve these various goals.

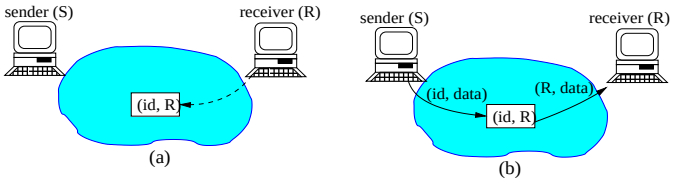

Figure 1: Example illustrating communication between two nodes. (a) The receiver R inserts trigger (id, R). (b) The sender sends packet $(id, data)$.

To avoid this replication, and to provide greater generality, we propose a unified indirection-based solution for multicast, anycast and mobility. The next Section describes the basics of our solution.

2 Rendezvous Based Communication

The key idea of our solution is to replace the point-to-point communication abstraction which dominates existing network architectures, with a *rendezvous-based communication abstraction*: instead of explicitly sending a packet to a given receiver, each packet is associated

*This research was partially supported by NSF under grant number NSF-ITR 0085879.

with an identifier, which is then used by the receiver to get the packet.

A rendezvous-based network exports two primitives: sending a packet p, $send(p)$, and inserting a trigger t, $insert(t)$. Triggers are used by nodes to receive packets. A trigger consists of two fields $(id, addr)$, where id represents the trigger identifier, and $addr$ represents a node address. A trigger $(id, addr)$ specifies the fact that the node with address $addr$ will receive packets identified by id. Figure 1 illustrates the communication between two nodes, where receiver R wants to receive packets identified by id. The receiver inserts the trigger (id, R) into the network. Upon receiving a packet $(id, data)$, the network searches for the trigger whose identifier matches id and forwards the packet to receiver R.

Thus, id represents a rendezvous point between the sender's packets and the receiver's trigger, hence the name of the communication abstraction. This level of indirection decouples the sender and the receiver behaviors. The fact that the receiver moves and changes its address in the process or that there are multiple receivers listening to the same packets is *transparent* to the sender. Similarly, the fact that the sender changes its address or that there are multiple senders sending packets with the same identifier is transparent to the receiver. This decoupling is the key that enables most of the desirable properties exhibited by rendezvous-based networks such as providing mobility and multicast.

To understand the rendezvous-based communication model better, it is useful to contrast it with the IP model. The packet identifier is similar to the IP destination field. However, while the IP destination field contains, in general, topological information in the form of the end-host address, the packet identifier is semantics free: from the network's viewpoint, an identifier is just an uninterpreted string of bits. Triggers are similar to routing entries. However, while in the Internet, routing entries are updated and maintained by special routing protocols, triggers are explicitly maintained by end hosts. This gives end-hosts maximum flexibility and control in choosing the identifiers and the "paths" along which packets are propagated.

To demonstrate the feasibility of our solution, we are implementing an overlay network on top of IP, called *Internet Indirection Infrastructure* ($i3$), that implements the rendezvous-based abstraction.

3 Internet Indirection Layer ($i3$)

$i3$ is an overlay network which consists of a set of servers that store triggers and forward packets between end-points (end-processes). The address of an end-point consists of an IP address and a port number. Packet and trigger identifiers are represented by strings of m bits.

We assume that each end-host knows a list of $i3$ servers, which is obtained via a bootstraping mechanism when the end-host joins the $i3$ system. When an end-host wants to send a packet, it hands the packet to an $i3$ server. Upon receiving a packet, an $i3$ server searches for the trigger matching the packet. If such a trigger is found, the packet is forwarded via IP to the end-point whose address is stored by the matching trigger. The packets are not stored in $i3$; they are only forwarded.

At its basis, $i3$ implements a best-effort service like today's Internet. $i3$ implements neither reliability nor ordered delivery on top of IP. End-hosts use periodic refreshing to maintain their triggers into $i3$. This soft-state approach allows for a simple and efficient implementation. When an end-host fails, its triggers are automatically deleted from $i3$. If a trigger is lost – for example, as a result of an $i3$ server failure – the trigger will be reinserted, possibly at another server, the next time the end-host refreshes it.

To find the trigger that matches a given packet, $i3$ relies on one of the recently proposed lookup services [18, 19, 22, 27]. A lookup service maps an identifier space to a set of servers. Given an identifier id, the lookup service finds the server responsible for that id. This primitive makes it easy to implement the matching procedure. A trigger $(id, addr)$ is stored at the server responsible for id. In turn, a packet $(id, data)$ is forwarded based on id through the overlay network to the same server. Then, the packet is matched to the trigger, and forwarded to $addr$ via IP.

Public and Private Triggers: Without loss of generality, we assume two types of triggers: public and private. The identifiers of public triggers are known by all end-hosts in the system. An example is a web server that maintains a public trigger to allow any client to contact it.[1] Public triggers are long lived, typically days or months. In contrast, private triggers are chosen cooperatively by a small number of end-hosts and they are short lived. Typically, private triggers exist only during the

[1] A public trigger can be defined as a the hash of the host's DNS name, of a web address, or of the public key associated to a web server.

duration of a flow.

To illustrate the difference between public and private triggers better, consider the possible scenario of a client accessing a web server. First, the client chooses a private trigger identifier id_c and sends it to the web server via the server's public trigger. Once contacted, the server will create a private trigger identifier id_s and send it back to the client. The client and the server will then insert the private triggers $(id_c, addr_c)$, and respectively $(id_s, addr_s)$ into $i3$, and use them to communicate. Once the communication terminates, the private triggers are destroyed. Section 3.2 discusses the security vulnerability associated with public triggers, and outlines a possible solution.

3.1 Services Supported by $i3$

In this section, we use several examples to demonstrate the power and the flexibility of $i3$. We start with a basic model in which the matching procedure reduces to exact matching, and show how this model can support mobility and multicast. We then extend $i3$ to support more complex functionalities such as anycast and service composition.

Mobility: Since end-hosts use packet identifiers rather than node addresses to communicate, $i3$ provides natural support for mobility. A mobile host that changes its address from $addr_1$ to $addr_2$ as a result of moving from one subnetwork to another can preserve the end-to-end connectivity by simply updating each of its existing triggers $(id, addr_1)$ to $(id, addr_2)$.

With any scheme that supports mobility, efficiency is a major concern [21]. $i3$ employs several techniques to achieve efficiency. First, each packet is routed based on its identifier to the server that stores its trigger. This means that no additional operation needs to be invoked when the sender or the receiver moves. Second, the address of the server storing the trigger is cached at the sender, and subsequent packets are forwarded directly to that server via IP. This way, most packets are forwarded along a one-hop route in the overlay network. Third, to alleviate the triangle routing problem, end-hosts can use off-line heuristics to sample the $i3$ identifier space to place the triggers close to themselves.

Multicast: $i3$ provides native support for multicast. Creating a multicast group is equivalent to having all members of the group register triggers with the same identifier id. As a result, any packet that matches id is forwarded to all members of the group.

Note that unlike IP multicast, with $i3$ there is no difference between unicast or multicast packets, in either sending and receiving. Such an interface gives maximum flexibility to the application. An application can switch on-the-fly from unicast to multicast by simply having more hosts insert triggers with the same identifier. For example, in a telephony application this would allow multiple parties to seamlessly join a two-party conversation. In contrast, with IP, an application has to at least change the IP destination address in order to switch from unicast to multicast.

Anycast: Anycast ensures that a packet is delivered to at most one receiver in a group. Anycast enables server selection, a basic building block for many of today's applications.

To implement anycast we assume a more general matching rule. Instead of exact matching, we consider the longest prefix matching with the restriction that the k most significant bits should match exactly. More precisely, a packet $(id, data)$ matches the trigger $(id', addr)$ if and only if (1) the k most significant bits of id and id' are identical, and (2) the $m - k$ least significant bits of id' represent the longest prefix match for the $m - k$ least significant bits of id among all trigger identifiers stored in the network. We choose k large enough to eliminate (for all practical purposes) the possibility of erroneously delivering a packet to the wrong destination, or of an adversary guessing the first k bits of the packet identifier by scanning the identifier space.

This matching rule makes it straightforward to implement anycast. All hosts in an anycast group maintain triggers which are identical in the k most significant bits. These k bits play the role of the anycast group identifier. To send a packet to an anycast group, a sender associates with each packet an identifier which shares the k most significant bits with the identifiers of the receiver triggers. The packet is delivered to the member of the group whose trigger identifier best matches the packet identifier.

The last $m - k$ bits of the identifier can be used to encode application preferences. For instance, assume that there are n web servers and the goal is to balance the client requests among these servers. This goal can be achieved by setting the $m - k$ least significant bits of both trigger and packet identifiers to random values. Note that by changing the semantics of the information stored in the $m - k$ least significant bits it is possible to achieve other goals than load balancing. For example, if we encode the host location in those bits (e.g., zip code),

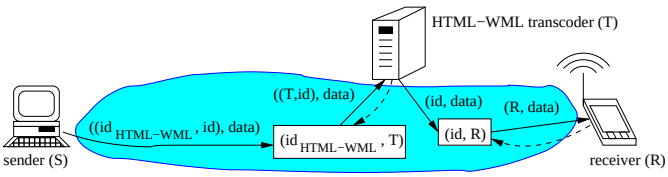

Figure 2: Service composition example. Sender sends HTML format; receiver can display only WML format. Server T performs HTML-WML transcoding.

packets will be delivered to a group member that is geographically close to the sender.

Identifier Stacks (Supporting Service Composition): Some applications may require third parties to process the data before it reaches the destination [11]. An example is a wireless application protocol (WAP) gateway translating HTML web pages to WML for wireless devices [25]. WML is a lightweight version of HTML designed to run on wireless devices with small screens and limited capabilities. In this case, the server can forward the web page to a third-party server T that implements the HTML-WML transcoding, which in turn processes the data and send it to the destination via WAP.

This section presents an alternative to implement service composition in $i3$. The idea is to extend $i3$ by replacing a packet identifier with a *stack* of identifiers. $i3$ then tries to find the trigger that matches the first identifier in the stack. If such a trigger is found, the identifier at the head of the packet's stack is replaced with the trigger's address and the process continues. The process terminates when the packet's stack becomes empty.[2] Stacks give the sender the possibility to control the path of the packets in $i3$. Note that using stacks to forward packets is similar to source routing in IP. $i3$ goes one step further by also allowing the receiver to control the path. To achieve this, we replace the address of a trigger with a stack of identifiers. When a packet matches a trigger, the identifier at the head of the packet's stack is replaced with the stack of the trigger. Figure 2 shows an example in which the sender controls the transcoding of the data packets from HTML to WML by using the identifier stack $(id_{HTML-WML}, id)$.

3.2 Security

Unlike IP, where an end-host can only send and receive packets, in $i3$ end-hosts are also responsible for maintaining the routing information through triggers. While this allows flexibility for applications, it also (and unfortunately) creates new opportunities for malicious users. We now discuss several security issues and how $i3$ addresses them.

Eavesdropping: A user that knows a host's trigger can eavesdrop the traffic towards that host by inserting a trigger with the same identifier and its own address. We consider two cases: (a) private and (b) public triggers.

Private triggers are secretly chosen by the application end-points and are not supposed to be revealed to the outside world. The length of the trigger's identifier makes it very difficult for a third party to use a brute force attack.[3] Furthermore, end-points can periodically change the private triggers associated with a flow.

With $i3$, a public trigger is known by all users in the system, and thus anyone can eavesdrop the traffic to such a trigger. To alleviate this problem, end-hosts can use the public triggers to choose a pair of private triggers, and then use these private triggers to exchange the actual data. To keep the private triggers secret, one can use public key cryptography. To initiate a connection, a host A chooses a private trigger id_a, encrypts it under the public key of a receiver B, and then sends it to B via B's public trigger. B decrypts A's private trigger id_a, then chooses its own private trigger id_b, and sends this trigger back to A over A's private trigger id_a. Since the sender's trigger is encrypted, a malicious user cannot impersonate B unless it knows B's private trigger.

Trigger hijacking: A malicious user can isolate a host by removing its public trigger. Similarly, a malicious user in a multicast group can remove other members from the group by deleting their triggers. While removing a trigger also requires to specify the IP address of the trigger, this address is, in general, not hard to obtain.

One possibility to guard against this attack is to add another level of indirection. Consider a server S that wants to advertise a public trigger with identifier id_p. Instead of inserting the trigger (id_p, S), the server can insert two triggers, (id_p, x) and (x, S), where x is an identifier known only by S. Since a malicious user has to know x in order to remove either of the two triggers, this simple technique provides effective protection against this type of attack.

DoS Attacks: The fact that $i3$ gives end-hosts control on

[2] To avoid loops, each packet carries also a hop count that is decremented every time the packet matches a trigger.

[3] While other application constraints such as storing a trigger at a server nearby can limit the identifier choice, the identifier is long enough (i.e., 256 bits), such that the application can always reserve a reasonable large number of bits (e.g., 64 bits) that are randomly chosen.

routing opens new possibilities for DoS attacks. We consider two types of attacks: (a) attacks on end-hosts, and (b) attacks on the infrastructure. In the former case, a malicious user can insert a hierarchy of triggers in which all leaf triggers point to the victim. Sending a single packet to the trigger at the root of the hierarchy will cause the packet to be replicated and all replicas to be sent to the victim. This way an attacker can mount a large scale DoS attack by simply leveraging the $i3$ infrastructure. In the latter case, a malicious user can create trigger loops, for instance by connecting the leaves of a trigger hierarchy to its root. In this case, each packet sent to the root will be exponentially replicated!

To alleviate these attacks, $i3$ uses three techniques:

1. **Challenges** $i3$ implicitly assumes that a trigger that points to an end-host R is inserted by the end-host itself. An $i3$ server can easily verify this assumption by sending a challenge to R the first time the trigger is inserted. The challenge consists of a random nonce that is expected to be returned by the receiver. If the receiver fails to answer the challenge the trigger is removed. As a result an attacker cannot use a hierarchy of triggers to mount a DoS attack (as described above), since the leaf triggers will be removed as soon as the server detects that the victim hasn't inserted them.

2. **Resource allocation** Each server uses Fair Queueing [6] to allocate resources amongst the triggers it stores. This way the damage inflicted by an attacker is only proportional to the number of triggers it maintains. An attacker cannot simply use a hierarchy of triggers with loops to exponentially increase its traffic. As soon as each trigger reaches its fair share the excess packets will be dropped. While this technique doesn't solve the problem, it gives $i3$ time to detect and to eventually break the cycles.

3. **Loop detection** When a trigger that doesn't point to an IP address is inserted, the server runs a procedure to detect whether the new trigger doesn't create a loop. A simple procedure is to send a special packet with a random nonce. If the packet returns back to the server, the trigger is simply removed. To increase the robustness, the server can invoke this procedure periodically after such a trigger is inserted.

4 Related Work

The rendezvous-based communication is similar in spirit to the tuple space work in distributed systems [2, 13, 26]. However, tuple spaces have richer semantics, and they guarantee persistence and atomicity. Providing these properties in very large scale distributed systems is, however, very difficult. In contrast, $i3$ trades these properties for a scalable and efficient implementation.

$i3$ shares many similarities with naming systems. This should come as no surprise, as identifiers can be viewed as semantic-less names. DNS maps hostnames to IP addresses [15]. DNS names are hierarchical while $i3$ identifiers are flat. DNS resolvers form a static overlay hierarchy, while $i3$ servers form a self-organizing overlay. $i3$ integrates identifier resolution with packet forwarding. Active Names (AN) maps a name to a chain of mobile code responsible for locating the remote service [23]. While AN names are used primary to describe services, $i3$ identifiers are used primary to abstract away the end-host location. Intentional Naming System (INS) is a resource discovery and service location system for mobile hosts [24]. $i3$ differs from INS in that from network's point of view an identifier does not carry any semantics. Another difference is that $i3$ allows end-hosts to explicitly control (via triggers) the application-level path followed by the packets.

The rendezvous-based abstraction is similar to the IP multicast abstraction [5]. An IP multicast address identifies the receivers of a multicast group in the same way an $i3$ identifier identifies the multicast receivers. However, unlike IP which allocates a special range of addresses (i.e., class D) to multicast, $i3$ does not put any restrictions on the identifier format. In addition, $i3$ has ability to support multicast groups with heterogeneous receivers.

TRIAD [3] and IPNL [10] have been recently proposed to solve the IPv4 address scarcity problem. Both schemes use DNS names rather than addresses for global identification. One difference between $i3$ and both TRIAD and IPNL is that the path of a packet is determined by end-hosts, instead of being determined during the DNS name resolution by network specific protocols.

5 Status

We have implemented an early prototype of $i3$ based on the Chord lookup service [22]. We use 256 bit identifiers, and the matching procedure requires exact match-

ing on the 128 most significant bits. This choice makes it very unlikely that a packet will erroneously match a trigger, and at the same time gives the applications up to 128 bits to encode application specific information such as the host location. The untuned prototype is able to forward about 35,000 packets per second, and to handle 80,000 trigger insertions/refreshes per second on a 700 MHz Pentium III processor. By choosing the set of Chord fingers based on the network proximity, we are able to reduce the routing latency between any two servers in $i3$ within a factor of two (on the average) of the optimal routing in the underlying network.

6 Acknowledgments

We thank Kevin Lai, Karthik Lakshminarayanan and Ananthapadmanabha Rajagopala-Rao for insightful comments that helped to refine the $i3$ design.

References

[1] A. J. Ballardie, P. F. Francis, and J. Crowcroft. Core based trees. In *Proc. of ACM SIGCOMM'93*, pages 85–95, San Francisco, 1993.

[2] N. Carriero. *The Implementation of Tuple Space Machines*. PhD thesis, Yale University, 1987.

[3] D. R. Cheriton and M. Gritter. TRIAD: A new next generation Internet architecture, March 2000. http://www-dsg.stanford.edu/triad/ triad.ps.gz.

[4] Y. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *Proceedings of ACM SIGMETRICS'00*, pages 1–12, Santa Clara, CA, June 2000.

[5] S. Deering and D. R. Cheriton. Multicast routing in datagram internetworks and extended LANs. *ACM Transactions on Computer Systems*, May 1990.

[6] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *Journal of Internetworking Research and Experience*, pages 3–26, October 1990.

[7] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei. Protocol independent multicast – sparse mode (pim-sm) : Protocol specification, June 1997. Internet RFC 2117.

[8] D. Estrin, D. Farinacci, V. Jacobson, C. Liu, L. Wei, P. Sharma, and A. Helmy. Protocol independent multicast – dense mode (pim-dm) : Protocol specification. *Work in Progress*.

[9] P. Ferguson and D. Senie. Network ingress filtering, January 1998. Internet RFC 2267.

[10] P. Francis and R. Gummadi. IPNL: A nat extended internet architecture. In *Proc. ACM SIGCOMM'01*, pages 69–80, San Diego, 2001.

[11] S. Gribble, M. Welsh, R. von Behren, E. Brewer, D. Culler, N. Borisov, S. Czerwinski, R. Gummadi, J. Hill, A. Josheph, R. Katz, Z. Mao, S. Ross, and B. Zhao. The ninja architecture for robust internet-scale systems and services, 2000.

[12] H.W. Holbrook and D.R. Cheriton. IP multicast channels: EXPRESS support for large-scale single-source applications. In *Proceedings of ACM SIGCOMM'99*, Cambridge, Massachusetts, Aug. 1999.

[13] Java Spaces. http://www.javaspaces.homestead.com/.

[14] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and Jr. J. W. O'Toole. Overcast: Reliable multicasting with an overlay network. In *Proceedings of the 4th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2000)*, San Diego, California, October 2000.

[15] P. Mockapetris and K. Dunlap. Development of the Domain Name System. In *Proc. ACM SIGCOMM*, Stanford, CA, 1988.

[16] C. E. Perkins. IP mobility support, October 1996. Internet RFC 2002.

[17] C. E. Perkins and D. B. Johnson. Mobility support in IPv6. In *Proceedings of ACM/IEEE MOBICOM'96*, pages 27,37, November 1996.

[18] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. ACM SIGCOMM*, pages 161–172, San Diego, 2001.

[19] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, November 2001.

[20] A. C. Snoeren and H. Balakrishnan. An end-to-end approach to host mobility. In *Proceedings of ACM/IEEE MOBICOM'99*, Cambridge, MA, August 1999.

[21] Alex C. Snoeren, Hari Balakrishnana, and M. Frans Kaashoek. Reconsidering internet mobility. In *Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, Elmau/Oberbayern, Germany, May 2001.

[22] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. ACM SIGCOMM'01*, pages 149–160, San Diego, 2001.

[23] A. Vahdat, M. Dahlin, T. Anderson, and A. Aggarwal. Active names: Flexible location and transport. In *Proceedings of USENIX Symposium on Internet Technologies & Systems*, October 1999.

[24] W. Adjie-Winoto and E. Schwartz and H. Balakrishnan and J. Lilley. The design and implementation of an intentional naming system. In *Proc. ACM Symposium on Operating Systems Principles*, pages 186–201, Kiawah Island, SC, December 1999.

[25] WAP wireless markup language specification (WML). http://www.oasis-open.org/cover/wap-wml.html.

[26] P. Wyckoff, S. W. McLaughry, T. J. Lehman, and D. A. Ford. T Spaces. *IBM System Journal*, 37(3):454–474, 1998.

[27] Ben Zhao, John Kubiatowicz, and Anthony Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, Computer Science Division, U. C. Berkeley, April 2001.