Providing content-based services in a peer-to-peer environment

Ginger Perng, Chenxi Wang, Michael K. Reiter
Carnegie Mellon University
Pittsburgh, PA, USA
gperng@ece.cmu.edu, chenxi@cmu.edu, reiter@cmu.edu

Abstract

Information dissemination in wide area networks has recently garnered much attention. Two differing models, publish/subscribe and rendezvous-based multicast atop overlay networks, have emerged as the two leading approaches for this goal. Event-based publish/subscribe supports content-based services with powerful filtering capabilities, while peer-to-peer rendezvous-based services allow for efficient communication in a dynamic network infrastructure. We describe Reach*, a system that integrates these two approaches to provide efficient and scalable content-based services in a dynamic network setting.*

1 Introduction

Event-based publish/subscribe (pub/sub) services have been widely studied as a basis for information dissemination in large networks (e.g., [3, 4, 6]). In this communication model, a *subscriber* registers a long-standing *subscription* to the pub/sub service and receives published messages that match that subscription. Pub/sub services are typically supported by a fixed infrastructure of routers that disseminate subscriptions and forward messages based on their *content* rather than the addresses of the subscribers. Current pub/sub systems are capable of supporting rich subscription languages and provide powerful content-based services.

A different model for information dissemination is rendezvous-based communication services layered atop structured peer-to-peer overlays (e.g., as implemented using Distributed Hash Tables (DHTs))). Motivated by a separate set of goals, the DHT-based information services typically do not support content-based routing but instead focus on providing efficient communication in a highly dynamic network where servers can join and leave at will. In this model, overlay nodes serve as logical meeting points for senders and receivers to exchange information. The overlay layer is responsible for efficiently locating the corresponding rendezvous node for a particular message and forwarding the message to interested subscribers (e.g., [5, 9, 13, 15]).

The pub/sub and rendezvous models present dramatically different approaches to information dissemination, each with its advantages. The pub/sub models exploit information content for routing, which supports highly expressive subscription and filtering capabilities. In contrast, peer-to-peer rendezvous-based services—notably DHTs—route in a way that ignores content and thus provides limited information services. However, the rendezvous-based services are built on top of peer-to-peer networks which have the ability to adapt dynamically via members joining and leaving the service. This permits a degree of flexibility and resource sharing that conventional pub/sub systems do not exploit.

Despite these contrasts, we argue that peer-to-peer rendezvous-based services and pub/sub communication can be integrated in such a way that gains the advantages of both. Our goal in this paper is to examine using rendezvous-based communication as a primitive for implementing pub/sub services. Specifically, we describe the design of a system we call *Reach*, which supports an important class of content-based services—namely content-based multicast—atop a peer-to-peer service, using the rendezvous abstraction. At a high level, the rendezvous service is the means by which subscriptions are stored in the network (like *triggers* in [12]), and by which published messages are directed to "find" the subscriptions they match. This rendezvous node is then an entry point into a "subset tree" of nodes hosting other, weaker (more general) subscriptions, and thus to which this message should also be routed. This tree is implemented in such a way that it offers join-and-leave flexibility and maximum efficiency as the nodes in the tree are nearby neighbors in the overlay.

We describe the basics of our design in Section 2 and 3, discuss research issues in Section 4, present related work in Section 5, and conclude in Section 6.

2 System Model and Design

In Reach, we assume that the universe of information content is characterized according to n enumerated attributes. Each distinct event (publication) bears an n-bit identifier, where a "1" in the i-th bit indicates that the event pertains to the i-th attribute; we denote the identifier for event e by id_e. Subscriptions in Reach are expressed in the same manner—each subscription contains an n-bit identifier in which each bit set indicates an interest in the corresponding attribute. A subscription indicates a conjunction of the indicated interests, i.e., an event matches a subscription if for every set bit in the subscription, the bit in the event identifier is set, as well.

This encoding scheme defines an identifier hierarchy — id_1 is said to be a parent of id_2 if and only if $id_1 \supset id_2$. For example, as shown in Figure 1, identifier 0011 is a parent for both 0001 and 0010 . More intuitively, a parent identifier contains at least all the attributes of a child identifier. This hierarchy is a fundamental concept in Reach and is the basis for content-based multicasting. The precise benefits of this hierarchy will become clear in Section 3.

Reach consists of a network of overlay nodes. Each node serves as a rendezvous point for some subscriptions and messages. They are also responsible for forwarding messages to interested clients. We say that a node i is the rendezvous node for message e if i hosts id_e (the same is true for subscriptions). For illustration purposes, we first discuss how identifiers are mapped onto physical nodes when the network size is a power of two. The other cases are discussed in Section 3.3. Recall that n is the size of the attribute space. Assume that we have a network with 2^m nodes, $m \leq n$, where each node can be identified by an m-bit string (we call this the node ID). An identifier, id, is mapped to node i if the lower-order m-bits of id coincide with the node ID of i. For example, node 01 in a network of four nodes would host identifiers 0001, 0101, 1001, and 1101 from an 4-bit attribute space. For convenience, we denote that node 01 hosts **01.

Messages and subscriptions entering Reach are routed to their designated rendezvous node. A subscription is stored at its rendezvous node where messages are matched to the subscriptions. Once a match occurs, the message is forwarded to the interested client. A message e is said to match a subscription f if and only if $id_e \supseteq id_f$. Note that not all subscriptions f's satisfying $id_e \supseteq id_f$ are located on the same rendezvous node. Therefore, the challenge is to efficiently locate all nodes that host f's such that $id_e \supseteq id_f$. The routing and forwarding algorithms described in Section 3 are designed specifically with this goal in mind.

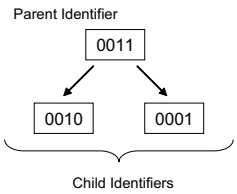


Figure 1. Parent/child relationship.

3 Routing and Content-Based Multicasting

Reach is a rendezvous-based network that supports content-based multicast. To achieve this, Reach uses a unique peer-to-peer lookup service that maintains high-level semantic relations within the overlay. We begin with a description of Reach's basic peer-to-peer routing infrastructure. We then describe content-based multicasting on top of this infrastructure. In Section 3.3 we discuss how Reach handles dynamically changing network sizes. We start our discussions assuming a power of two network size; this restriction is later relaxed.

3.1 Point-to-Point Routing

In Reach, we use a Hamming-distance based routing scheme. More specifically, each Reach node maintains a routing table that contains the addresses of all the nodes whose identifiers are one Hamming distance away from its own. For instance, node 1011 would have a neighbor set containing 0011, 1001, 1010, and 1111. In a network of 2^m nodes, each Reach node has a routing table with m entries. We note that the neighboring relationship in Reach is in the overlay layer and therefore is entirely logical.

The Reach routing algorithm is straightforward: a message is incrementally forwarded to its destination in such a fashion that each hop puts the message one Hamming distance closer to its destination. Consequently, the number of overlay hops between a pair of nodes is exactly the Hamming distance between their respective node IDs.

For a network of 2^m nodes, the average point-to-point path length in Reach is $O(m)$. This is similar to other peer-to-peer lookup services such as [13, 15, 10]. However, we stress that our contribution is not in Reach's ability to act as a routing infrastructure, but rather in its ability to support content-based multicasting.

3.2 Content-based Multicasting

To support content-based multicasting, Reach must locate all nodes who host subscriptions that are subsets of the event identifier of a particular message. For example, a message with identifier 1001 needs to reach the node hosting 1001, as well as nodes hosting 1000 and 0001. To achieve this, our high-level strategy is simple: each event

message that enters Reach is first routed to its rendezvous node (1001 in our example). From there the event is progressively forwarded to nodes that host subsets of the event identifier (1000 and 0001). Each traversed node forwards the event to its subscribers if matching subscriptions are found. Routing toward the rendezvous point is simply a point-to-point overlay communication that follows the algorithm described in Section 3.1. Discussions in this section focus on the post-rendezvous dissemination operations. We call this part *subset routing* (see Figure 2).

Recall the definition of hierarchical event types in Section 2. This hierarchy defines a superset-subset relationship that is fundamental to achieving subset routing. Informally, subset routing follows a dissemination tree rooted at the rendezvous node. The ID of every child node in the tree is one Hamming distance away from its parent and has Hamming weight less by one. The shaded nodes in Figure 2 illustrate a subset routing tree. As shown, node 0111 has three immediate children, 0011, 0101, and 0110. When an event reaches its rendezvous node (in Figure 2, event 1100111 reaches node 0111), the event is recursively forwarded to the nodes one level down the tree until it reaches all nodes in the tree. In the example in Figure 2, event 1100111 is sent to 0110, 0101, 0011, and from there to 0010, 0100 and 0001. The ways in which routing tables are constructed in Reach guarantee that children nodes are always in the parent's routing table (and vice versa). In a subset routing tree, every non-root node hosts identifiers that are subsets of the event identifier. Thus, an event eventually reaches all subset subscriptions by starting from the rendezvous node and traversing down the tree.

One might observe that the subset routing tree is not unique (e.g., in Figure 2, 0110 can also be parent to 0100). We use a deterministic algorithm to build a tree and at the same time, eliminate potential duplicate messages. The details of the subset routing algorithm are as follows: When a message e reaches its rendezvous node i, i finds the children of id_i and forwards the event to each such child. When a child node j, such that $id_j \subset id_i$, receives this message, it generates the children of id_j and forwards the message to them, and so forth. In the meantime, each node uses a deterministic algorithm to eliminate duplicate messages to children in common with other nodes. A possible algorithm would be to use a public hash of the message identifier as an indexing mechanism to determine who is responsible for the common children.

At a more intuitive level, the content-based multicasting mechanism in Reach ensures that a message is forwarded to each subscription that requests all or a subset of the attributes present in the message content. The multicast algorithm generates $2^b + l$ messages where b ($b \leq m$) is the number of marked attributes in the m-bit suffix of the message identifier, and l is the number of overlay hops to reach the rendezvous node (i.e. the root of the tree).

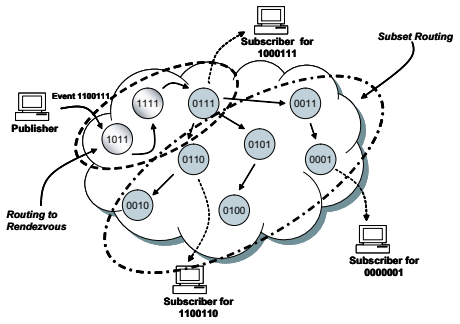


Figure 2. Dissemination of event 11000111 in a network of size 2^4 (m = 4, n = 8).

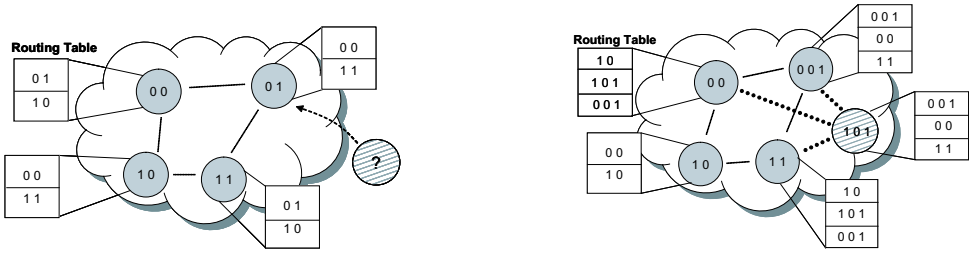
In contrast to other DHT-based overlay networks, Reach identifiers encode meaningful application-level information, i.e., attributes. As a result, the neighboring relation in Reach reflects a semantic relation that is not preserved in standard peer-to-peer overlay networks. This semantic relationship in the overlay layer is the key to an efficient implementation of content-based multicast. Without it, a separate overlay message would be required to reach each potentially distant subset ID in the overlay layer.

3.3 Dynamic Networks

We have thus far described Reach when the network size is a power of two. In this section we relax this assumption and discuss arbitrary network sizes. Assuming Reach bootstraps from a network of 2^m nodes, we discuss node joins and leaves in turn.

Node joining: We assume that a new node joins Reach by contacting an existing Reach node. The Reach node, upon receiving a *join* request, divides the set of identifiers that it hosts between the new node and itself. Consider the scenario of a four-node network. Suppose node 01 receives a *join* request, it first creates two new node IDs, 001 and 101, by adding a leading bit to its original 2-bit node identifier. It then updates its own ID to 001 and labels the new node 101. Node 001 will continue to host identifiers whose 3-bit suffix match 001, and give the rest to the new node, 101. As a last step, 001 adds 101 to its routing table, informs its original neighbors, 11 and 00, of its new ID, and introduces 101 as a new neighbor to them. Node 00 and 11 update their routing table accordingly. Figure 3 depicts an example node join scenario.

It should be noted that a Reach node processes a *join* request only when its own ID has equal or fewer number of bits than all of its neighbors. If the node's ID has more bits than any of its neighbors, it simply forwards the *join* request to a neighbor whose ID has fewer bits than its own. Note



(a) Reach network before a node joins.

(b) Reach network after the node joins.

Figure 3. Example of a node joining a Reach network with 2^2 existing Reach nodes.

that in some cases a node will need to drop neighbors from its routing table after a *join* operation. Consider the example of a split at node 001, whose ID is being updated to 0001. A neighbor 1011 before the split is now two Hamming distances away from 0001. As a result, 0001 simply drops 1011 from its neighbor set.

Assuming k is the network size, on average $O(\log k)$ nodes are affected for each *join* operation (i.e., having to update their routing tables). At the end of a *join* operation, each node hosts a set of identifiers that has the node ID in its suffix.

Node leaving: Our algorithm for node leaving is similar to node joining. If node i with an $m+1$-bit identifier is to leave the network, it contacts node j who shares the m-bit suffix with i.[1] j will then host all of i's virtual identifiers and update its node ID to the m-bit suffix. i will inform all of its neighbors to drop i from their routing tables and to add j if it's not present. j will update its neighbors on its new node identifier. Again, $O(\log k)$ nodes will be affected by a single leave operation.

We note that a network with varying length node IDs has a size that is not a power of two. It should be clear that the routing and event dissemination mechanisms in Reach can carry on exactly in the same manner even with varying length node IDs—Hamming-distance based routing with different length IDs would simply route on the largest suffix. The average path length in such a network is still $O(\log k)$ when the network size is between k and $2k$. The joining and leaving operations can be repeated as many times as the attribute space allows—for an n-bit attribute space, the network can grow to at most 2^n nodes where each node hosts a single virtual identifier.

Concurrent Joins/Leaves: The join/leave algorithm described thus far assumes that nodes join and leave the network in a serialized manner. In a scenario with concurrent joins and leaves, nodes in the network may operate on out-of-date neighbor tables. We define the *global gap* as the largest difference in the ID length of any pair of nodes in the network. As shown in [1], networks with large global gaps have longer average path lengths than those that do not. Handling concurrent joins and leaves with out-of-date routing tables can result in a large global gap. A possible remedy to reduce the global gap in Reach is by imposing a limit on the number of join/leave requests a node can process between subsequent heartbeat messages from neighbors. Assuming the heartbeat messages carry up-to-date IDs from the neighbors, a node can update its routing table before processing subsequent requests. A detailed discussion and analysis of concurrent joins/leaves is out of the scope of this paper.

[1] There should only be one such node in the network and it is in i's routing table.

4 Discussion

4.1 Naming

In Reach, we assume a globally-static attribute space and use a bit vector for naming. This naming scheme preserves high-level application semantics, which gives rise to a content-based abstraction that is fundamentally more powerful than standard DHTs. An interesting question is whether information content can be effectively represented by such a bit-vector representation. For example, if the application in question consists of querying the content of newspaper articles, using a separate bit to represent every newspaper ever published is clearly not an option. A more rational approach would call for some kind of hierarchical attribute space for which high-level filtering and routing are followed by filtering on more fine-grained attributes.

We emphasize that the attribute representation problem is independent of the Reach architecture and routing scheme. Use of a hierarchical attribute space, for instance, may better facilitate certain applications, but it will not result in fundamental changes to the Reach infrastructure design.

4.2 Finer-Grain Filtering

Currently, Reach only allows clients to express interest in particular attributes. A more powerful subscription language would allow filtering based on the values of attributes. Augmenting Reach's functionality to achieve this is simple: the routing scheme (including point-to-point and subset routing) in Reach need not change, but subscriptions can be accompanied by arbitrarily complex filters on the values of the attributes. When a message is being matched to the subscriptions, the finer-grain filters on attributes values will be invoked. This way, an event matches a subscription if and only if the event identifier matches the subscription identifier *and* the value-based filter covers the event attribute values.

4.3 Fault Tolerance

Our current design does not specifically deal with fault tolerance. We note that link failures at the physical network layer can be masked with similar mechanisms as those in [16, 11]. However, our routing scheme will break down if a physical intermediate node fails. Since every Reach node serves as a rendezvous point for some particular content, straightforward replication of all the rendezvous nodes is not a viable option. We note that Reach falls in the general vein of hypercube models, and fault tolerance can potentially be dealt with by adopting a hypercube fault-tolerance mechanism (e.g.,[8]). Further investigation is needed to determine if these mechanisms can adapt to dynamic networks such as Reach. We are also investigating other more fault tolerant methods for traversing the subset tree.

4.4 Load Balancing

In a random subscription and event workload, Reach's mapping of event IDs to rendezvous node is evenly distributed between all nodes in the system. However, in reality, it is possible that certain rendezvous nodes will become overloaded due to popular event identifiers. Reach currently does not have specific mechanisms to deal with load balancing. A potential load-balancing scheme could replicate subscriptions up the dissemination tree. A parent node can then probablistically decide whether to fulfill the children's subscriptions. Another scheme could allow an overloaded node, A, to push an identifier, id, to another less overloaded node, B. A would send all subscriptions that match id to B. When an event labeled with id is published, A would then forward the event to B to let B fulfill the subscriptions for id. This is similar to load balancing mechanisms as found in [12].

4.5 Quenching

Another potential optimization for Reach is incorporating a quenching mechanism to reduce unnecessary communication due to lack of subscriber interest. A quenching mechanism would be as follows: when a subscription reaches its rendezvous node, the node sends an *interest* bit to all its neighbors whose IDs are supersets to his ID. Quenching is performed recursively; that is, an interest bit from a child node will cause an interest bit to propagate upward to the parents and superset nodes only if the *interest* bit has not been communicated previously. Therefore, an event will only be delivered to children nodes that have indicated interest prior to the publication of the event. This is similar to quenching performed by Siena[4].

5 Related Work

As Reach is inspired by supporting content-based services with dynamic server infrastructures, it is important to contrast our design with others that have similar goals.

5.1 Publish/Subscribe Networks

Current pub/sub networks such as [2, 4] are capable of supporting powerful content-based services. Compared to Reach, they provide more expressive subscription languages and finer grain content-based services.

However, many pub/sub implementations require broadcasting subscriptions to the entire network. As network size increases, subscription registration becomes prohibitively expensive. There exist pub/sub designs that explore covering relationships between subscriptions to reduce broadcast traffic [4], but such designs only benefit if the workload lends itself well to such optimizations. In contrast, subscription registration in Reach does not require broadcast and traverses only $log(n)$ of the network nodes.

In general, pub/sub services are built on top of a static network infrastructure, and indeed many routing and forwarding algorithms rely on this static nature. We note that many applications may benefit from more general network settings including dynamically changing server populations. Reach, as other peer-to-peer systems, supports a dynamic network infrastructure and as a result allows the network to scale gracefully as the server population changes.

A pub/sub system that allows for dynamic server populations is presented in [14]. However, as with other pub/sub systems, broadcasts to the network are necessary to establish the content-based routing paths.

5.2 Application Level Multicast

Information services such as Bayeux [16], Scribe [11] and *i*3 [12] are built on top of structured overlay networks.

These systems provide application-level multicast services using the rendezvous-based communication model. In these systems, information is associated with an identifier that is a hash of the data content. Each data identifier is mapped to the rendezvous node whose ID is the closest match to the identifier. These systems only allow exact (or near exact) matching of subscriptions and data IDs and are incapable of supporting more sophisticated content-based operations.

Furthermore, application-level multicast services require the explicit creation of multicast groups and setup of rendezvous nodes. As [4] points out, there does not appear to be a universally optimal mapping from multicast groups to recipient interests. In Reach, multicast groups are implicitly defined by the superset/subset relationship, and thus do not need explicit management. As a result, subscribers only receive messages matching their interests, and publishers need not duplicate messages to reach all relevant multicast groups.

A system that shares similar goals with Reach is Hermes [7]. Hermes implements a "type- and attribute-based" routing scheme that extends the expressiveness of subscriptions and supports event hierarchies. Hermes incorporates event hierarchies in the system by explicitly notifying "ancestor" identifers of the existence of new "descendant" identifiers. Thus, a published event with the ancestor identifier will also be forwarded to the rendezvous node of the descendant identifier. This scheme adds support for event hierarchies. However, nodes in a Hermes event hierarchy are more likely to be scattered throughout the network, thereby resulting in a large overhead for propagating an event to all of its descendants. In contrast, Reach attains efficiency by constructing a network wherein the super/subset identifiers are immediate neighbors in the overlay infrastructure.

6 Conclusion

In this paper, we presented a design for content-based multicasting atop a peer-to-peer rendezvous communication abstraction. Our approach offers benefits from both pub/sub and rendezvous models. From pub/sub, we implement an important class of information services, namely content-based multicasting. We adopt an expressive form of subset matching, and route publications efficiently on "subset trees" by utilizing subset relationships in the routing protocol. From the peer-to-peer model, we accomodate dynamic joins and leaves, and thereby gain benefits of resource sharing that have not been previously exploited in pub/sub systems. We showed that by exploiting semantic relationships in the overlay structure, we are able to achieve an efficient implementation of content-based multicast.

References

[1] I. Abraham, B. Awerbuch, Y. Azar, Y. Bartal, and D. Malkhi. A Generic Scheme for Building Overlay Networks in Adversarial Scenarios. *International Parallel and Distributed Processing Symposium* (Nice, France), April 2003.

[2] M. Aguilera, R. Strom, D. Sturman, M. Astley, and T. Chandra. Matching events in a content-based subscription system. *Symposium on Principles of Distributed Computing*, pages 53-61, 1999.

[3] G. Banavar, M. Kaplan, K. Shaw, R. E. Strom, D. C. Sturman, and W. Tao. Information flow based event distribution middleware. *International Conference on Distributed Computing Systems*, pages 114–121. IEEE Computer Society, 1999.

[4] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, **19**(3):332–383. ACM Press, August 2001.

[5] N. J. A. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. SkipNet: A Scalable Overlay Network with Practical Locality Properties. *Fourth USENIX Symposium on Internet Technologies and Systems* (Seattle, WA), March 2003.

[6] T. S. Inc. TIBCO Rendezvous. http://www.tibco.com/solutions/products/active_enterprise/rv/.

[7] P. R. Pietzuch and J. M. Bacon. Hermes: A Distributed Event-Based Middleware Architecture. *International Workshop on Distributed Event-Based Systems*, 2002.

[8] P. Ramanathan and K. G. Shin. Reliable Broadcast in Hypercube Multicomputers. *IEEE Transactions on Computers*, **37**(12):1654–1657, December 1988.

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

[10] A. Rowstron and P. Druschel. Pastry: scalable, decentralized object location and routing for large-scale peer-to-peer systems. *IFIP/ACM International Conference on Distributed Systems Platforms*, pages 329–350, 2001.

[11] A. Rowstron, A.-M. Kermarrec, P. Druschel, and M. Castro. SCRIBE: The design of a large-scale event notification infrastructure. *3rd International Workshop on Networked Group Communications*.

[12] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet Indirection Infrastucture. *ACM SIGCOMM Conference* (Pittsburgh, PA, August 2002), 2002.

[13] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Conference*. Published as *Computer Communication Review*, **31**(4):149–160, 2001.

[14] W. W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, and A. P. Buchmann. A Peer-to-Peer Approach to Content-Based Publish/Subscribe. *International Workshop on Distributed Event-Based Systems*, 2003.

[15] B. Y. Zhao, J. Kubiatowicz, and A. D. Joseph. *Tapestry: an infrastructure for fault-tolerant wide-area location and routing*. UCB Technical Report UCB/CSD–01–1141. Computer Science Division (EECS) University of California, Berkeley, April 2001.

[16] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. Kubiatowicz. Bayeux: An Architecture for Scalable and Fault-tolerant Wide Area Data Dissemination. *International Workshop on Networking And Operating System Support for Digital Audio and Video*, June 2001.