Willow: DHT, Aggregation, and Publish/Subscribe in One Protocol

Robbert van Renesse, Adrian Bozdog
Dept. of Computer Science, Cornell University

***Abstract*—This paper describes a new peer-to-peer protocol that integrates DHT routing, aggregation, all-to-all multicast, as well as both topic- and content-based publish/subscribe. In spite of this extensive set of features, the Willow protocol is simple, scalable, balances the load well across the members, is proximity-aware, adapts to network conditions, and recovers quickly and gracefully from network partitions and subsequent repairs.**

I. Introduction

In recent years, many application-level protocols have been designed for resource location, point-to-point and multicast routing, publish/subscribe, and aggregation. This paper introduces a new protocol, Willow, that provides all of these features. More specifically, Willow supports DHT-based routing, standing SQL aggregation queries on attributes of the nodes, Application-Level Multicast (ALM), and multicast filtering capabilities strong enough to support topic- and content-based pub/sub and more.

As with previous protocols, memory requirements on the nodes grow $O(\log N)$, while latency grows $O(\log N)$ ($O(\log^2 N)$ for aggregation). Willow is proximity-aware, and prefers short hops over long ones. Other than many previous proximity-aware protocols, Willow adapts to link latencies changing over time. A particularly important feature in the Willow protocol is its *zippering* mechanism by which separate Willow instances can be merged efficiently (in $O(\log N)$ parallel steps) into a single instance. The zippering mechanism is an important factor to Willow's stability both in the face of network partitions and in the face of churn.

Willow borrows some of its design from Astrolabe [1], [2]. While Astrolabe was intended to do aggregation only, Astrolabe can in fact be configured to function as a DHT as follows. Rather than manually assigning the Astrolabe domain names, they would be generated by concatenating, say, 32 4-bit digits. DHT routing can then be performed by walking the Astrolabe hierarchy in a straightforward manner. Using the SelectCast protocol [3] that runs on Astrolabe, pub/sub can be supported on this infrastructure as well.

[0]This work was funded in part by DARPA/AFRL-IFGA grant F30602-99-1-0532, and by the AFRL/Cornell Information Assurance Institute.

Willow, however, is closer in design to a traditional DHT. At the heart of the Willow protocol is a standard Plaxton-routing [4] infrastructure much like that of Kademlia [5]. But where other Plaxton-based DHTs hide aggregation facilities such as supported by Plaxton's original design, Willow exposes them. Compared to Astrolabe, Willow can support more queries, and answers them more quickly. Most importantly, Willow spreads load evenly across the members.

In this overview of the Willow protocol, we discuss related work in Section II, the Willow model in Section III, and the implementation in Section IV. A short look at results from simulation experiments are presented in Section V, and Section VI concludes.

II. Related Work

Due to space limitations, we limit ourselves to discussing only the most closely related projects.

Aggregation is important for supporting queries more complex than DHT lookup operations, as well as for scalable monitoring applications. In the area of peer-to-peer aggregation protocols, the most closely related projects besides Astrolabe are DASIS [6], Cone [7], SDIMS [8], SOMO [9], and PIER [10]. DASIS uses a Kademlia-like structure and aggregates information about the members in a way quite similar to Willow. In DASIS, information thus collected is used in the join algorithm to balance the P2P topology better than is typically achieved through random placement.

Cone augments a ring-based DHT with a trie, one for each attribute and aggregation operation. Cone can then support range queries over those attributes.

The SDIMS design exploits the fact that each key in Plaxton-based DHT identifies a tree consisting of the routes from each other node to the root node for that key. In SDIMS, each attribute and aggregation operation is hashed onto a key and then the aggregation is performed along the corresponding tree. The SDIMS implementation extends the Pastry protocol. In order to allow for an administrative hierarchy as in Astrolabe, Pastry was modified to have a leaf set for each administrative domain, rather than a single one.

Rather than augmenting a DHT, SOMO layers over a DHT. Even though Willow is an augmented DHT, SOMO aggregates information up a tree and then multicasts the results back down using the same tree much like in Willow. While Willow uses only a single tree, SOMO has a tree per key like in SDIMS.

In PIER, a DHT is used as a database index, and maps database keys to nodes that store the corresponding tuples. The DHT is augmented in order to allow for enumeration of tuples at nodes so that selection queries can be implemented. Most of the work in PIER so far has focussed on distributed joins rather than on how to support aggregation queries efficiently.

All DHT-based multicast protocols that we are aware of, such as Bayeux [11] and SplitStream [12], are layered on top of a DHT. In those systems, a key is associated with each multicast group, and one or more trees are build on a per-key basis, and these trees follow the DHT routes for those keys. Willow, in contrast, does not associate any keys with groups, but uses filtering in order to send messages to particular subsets of members. This leads to two advantages. Willow has many more routing options for multicast than do previous DHT-based multicast schemes, potentially resulting in better performance and load balancing, and more addressing options, strong enough to support even content-based publish/subscribe.

III. Willow Model

Before we show how the Willow protocol is implemented, we will first describe what Willow looks like once deployed.

A. Willow Tree

Each agent chooses a random 128-bit identifier. For the remainder of this paper, we assume that this results in each agent having a unique identifier, although agents can easily detect if their identifiers conflict and resolve this situation by choosing a new random identifier. The identifier determines a path in a virtual binary tree of 129 levels. Starting at the first bit of the identifier and the root of the tree, a 0 bit determines the left child, and a 1 bit the right child, and so on. Vice versa, each node in the tree can be named using a bit string. For example, the root is named by the empty bit string, and its right child is named "1". We call each node in the tree a *domain,* and all the agents whose identifiers start with the domain's identifier are considered members of that domain. In particular, all agents are members of the root domain, and each agent is a member of the leaf domain consisting of only the agent.

Each domain has attributes in addition to its identifier. In the case of a leaf domain, these attributes and their values are written directly by its corresponding agent. In the case of non-leaf domains (aka. *internal domains*), the attributes are determined by the two child domains. If both the left and right child domains are empty (contain no agents), then the set of attributes of the parent domain is empty as well. If only one of the child domains is empty, the attributes are the same as those of the non-empty child domain.

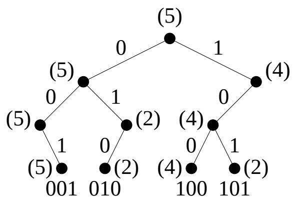

Fig. 1. Willow tree with four agents using three bit identifiers. The maximum load in each domain is indicated between braces.

The interesting case is when a domain has two non-empty child domains. We call such domains *branching domains*. The attributes of the parent domain are then determined by an aggregation function over the attributes of the child domains. In Willow this is done using SQL. Imagine the two sets of attributes forming two rows in a relational table. One or more SQL aggregation queries over the table are used to compute the attributes of the parent domain. For example, the aggregation query might be "SELECT MAX(maxload) AS maxload." This specifies that the parent domain will have an attribute *maxload* which is computed by taking the maximum of the *maxload* attributes of both child domains. All nodes share the same aggregation functions, and therefore the root domain will have an attribute *maxload* containing the global maximum load. Figure 1 shows an example Willow tree.

Each time the attributes of a leaf domain change, all the attributes of domains on the path up from the leaf domain to the root are recalculated automatically, much like in dependent cells in a spreadsheet. The aggregates are also recalculated in case of membership changes, or when aggregation functions are installed, updated, or removed, all of which can be done on the fly. We note that such updates are not instantaneous, in that there is a latency involved in the dissemination of queries and attribute updates to the various agents. Willow also does not guarantee consistency between the attributes that different agents observe, but all converge quickly to the same state in the absence of updates.

B. Willow Operations

Willow supports the following operations:

- *DHT routing*: route a message to an agent with an identifier nearest to a specified key in terms of the XOR metric [5];

- *monitoring*: using a JDBC query, specify what data needs to be reported by the agents into the Willow tree, and using an SQL query, how the data should be aggregated;
- *publish/subscribe*: route a message down the (logical!) Willow tree to all domains satisfying an SQL predicate specified within the header.

Monitoring is guided by two queries. (Both are installed and aggregated themselves as "executable attributes" within the Willow tree.) Once an agent installs such queries, it will take some time for them to propagate to all agents, and then some time before all the necessary information has been retrieved, reported, and aggregated. In order to determine when a query has completed, Willow has a permanent query installed that reports the total number of members of a domain in the *nmembers* attribute.

For example, say a user wants to determine which machine has the least load. Rather than reporting just the identifier and load of an agent, the user's query would really report $(agent, load, count)$, where $agent$ and $load$ are the least loaded agent for the corresponding domain, and $count$ is total number of agents that have reported their load for this query. The user waits until $count$ approaches $nmembers$. (Note that unless continuous updates are required, he or she should also remove the query at this time.)

There are queries that do not lend themselves well to aggregation. For example, the agent with the median load would be impossible to determine in the way described above. Such queries will require more than one pass to answer. Any query may be answered simply by collecting all the necessary information of all agents, but often more efficient approaches exist. In the example above, the agent with the median load can be determined using a binary search.

Willow also supports multicast and publish/subscribe. Given any domain, Willow supports a mechanism to forward a message to both child domains. When applied recursively, this mechanism multicast the message to all agents in the domain. The message may contain an SQL condition which is applied to the attributes of each child domain in order to determine whether to forward the message to the child domain or not. For example, it would be possible to send a message to all agents with a load less than 3, as long as a query is installed that reports the minimum load in each domain. Using a Bloom filter, traditional topic-based publish/subscribe can be efficiently supported this way, but the Willow mechanism is quite powerful and can support content-based publish/subscribe as well (both are based on the work in [3] and described in detail there).

level	friend	child	contact	candidate	maxload
0 (root)	101	0	001	010	5
		1	100	101	4
1	010	0	001	001	5
		1	010	010	2
2		0			
		1	001	001	5

Fig. 2. Data maintained by agent 001 in Figure 1. In actuality, not the identifiers of other agents are stored, but their IP addresses and boot times.

For this, each agent would specify a query as one of its attributes which is applied to attributes of messages. The queries are aggregated using logical `OR`. The Willow SQL query engine supports an `EVAL` function that allows such queries to be evaluated. In order for this to scale, the queries have to be conservatively simplified. For example, if there are too many terms in an aggregated query, the query can be replaced with `TRUE`. This has the effect that the message is broadcast in the higher levels of the Willow tree, and filtered in the lower levels. (Note that the same thing happens with Bloom filters, which also conservatively simplify the membership, leading to harmless false positives in the higher levels of the tree.)

IV. Willow Implementation

The internal architecture used by Willow is close to that of Kademlia [5], even though the tree maintenance is very different. As the Willow tree itself is virtual, each Willow agent maintains domain information for each of the 128 domains that it is a member of. Given a particular domain $\langle d \rangle$ (a prefix of the agent's identifier of length d bits), the domain information for $\langle d \rangle$ contains the following:

- a small (possibly empty) set of *friends*, which are the "fingers" or "neighbors" used for P2P routing. The friends are members of $\langle d \rangle 0$ if the agent is in $\langle d \rangle 1$, or $\langle d \rangle 1$ if the agent is in $\langle d \rangle 0$;
- the attributes of both the left and right child domains, $\langle d \rangle 0$ and $\langle d \rangle 1$ respectively, one of which (the one that the agent is not in) may be empty;

Figure 2 shows the data maintained by agent 001 in the example of Figure 1.

DHT routing uses friends for DHT routing exactly as in Kademlia. But in Willow, friends are also used for multicast routing as follows. Multicast messages contain an integer specifying in which domain they need to be forwarded. Initially, this integer is zero. To forward a multicast message with integer d, an agent considers all the branching domains that it is a member of with a name of d bits or longer. For each such domain, it forwards the message to one of the friends in the corresponding peer child domain (assuming they satisfy the SQL condition attached to the message), replacing the integer to contain the length of the child domain's identifier.

Note that the branching factor of the resulting multicast dissemination tree is $O(\log N)$, and thus the number of hops $O(\log N/\log\log N)$. Also note that in spite of there being only one logical tree, there is a different physical multicast tree from every agent, and this contributes to good load balancing. Most importantly, if agents are connected to the Internet with only a single link, and can only send one message at a time, the approach can be shown to maximize parallelism for relatively long messages.

A. *Attribute Propagation*

Consider an agent and some domain $\langle d\rangle$. The agent is either in the left or right child domain, say $\langle d\rangle 0$. In that case, the agent derives the attributes of $\langle d\rangle 0$ from the attributes in the domain information of $\langle d+1\rangle$, while it learns the attributes of $\langle d\rangle 1$ through communication with other agents. This proceeds as follows.

Each domain elects one of its agents to be the *contact* of its domain. The default election strategy favors older, presumably more stable, agents to represent larger domains. This is done using the Willow aggregation facility, which elects both a contact and a *candidate* for each domain. The contact of a branching domain is the younger one of the candidates of its child domains, while the candidate of a domain is the older of the candidates of its child domains. The contact and candidate of leaf domains are both the agent itself. Note that all agents are contacts of exactly one internal domain, except for the oldest agent which is contact only of its leaf domain. The election strategy can be changed as needed simply by installing another aggregation query.

The contact of a domain is responsible for sending the attributes of the domain to the corresponding peer domain. That is, the contact of domain $\langle d\rangle 0$ sends updates of its corresponding attributes to a friend in $\langle d\rangle 1$, which then disseminates the update in its domain through multicast. Although higher level contacts have usually more attributes to aggregate and disseminate, the variance of higher level attributes tends to be low and so updates are often significantly less common than those of lower level domains. This depends of course on the choice of aggregation queries, but in practice we have seen relatively little load on higher level contacts.

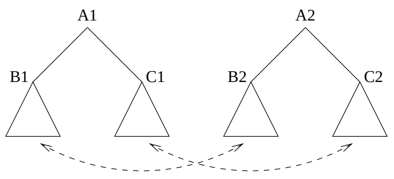

Fig. 3. Two distinct Willow trees are merged recursively and in parallel.

B. *Efficiency*

An important aspect of the Willow protocol is how friends are determined, as these determine how well Willow exploits network locality. Currently, Willow maintains only a single friend per peer domain. At regular intervals (currently, once a minute), each agent probes a random agent in each peer domain (determined using a DHT lookup to a random key in that domain). If the random agent exhibits better latency than the current friend, the friend is replaced with the new agent. In Section V we show that this is an effective strategy.

In the Willow implementation, all communication is through TCP. As TCP connections do not lose any data, only diffs need to be exchanged over these pipes, which reduces communication overhead. TCP takes care of congestion control. Willow further limits the rate of sending updates in order to control load on the network. If a TCP pipe is full or the maximum rate has been exceeded, diffs can be "saved up." Newer updates will typically overwrite parts of older updates, and therefore the amount of backlog that builds up this way is limited. Note that each agent only has to maintain one TCP connection per friend, plus at most two for each of the domains the agent is contact of. Thus the total number of TCP connections per agent is $O(\log N)$.

C. *Tree Maintenance*

So far we have tacitly assumed that there are no membership changes. Willow supports a Tree Maintenance Protocol (TMP) to maintain a single instance of the Willow tree in which all agents have a consistent view of this tree. The TMP is a recursive protocol in which left and right child branches of a domain are repaired in parallel. Even disjoint trees merge quickly once communication between any two agents in the respective trees is established. Such initial contact between separated trees can be established through a rendezvous host, IP multicast or broadcast. This is also how new agents join the Willow tree. Most other DHTs use DHT routing in order to add new agents, but this does not work efficiently when merging trees or fixing broken DHT structures.

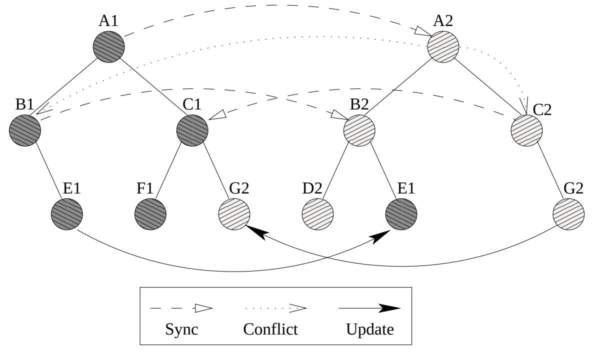

Fig. 4. Example of the zippering protocol.

In Figure 3 we show two disjoint Willow trees (or subtrees). In order to merge domains $A1$ and $A2$, we first recursively merge $B1$ and $B2$ first in parallel with $C1$ and $C2$, and then fix the top-level domains. The TMP uses two types of messages. A `sync` message contains the name of a (non-leaf) domain and the attributes of both child domains as known by the sender of the message (which is an agent in that domain). It is used when one agent discovers another agent with a different top-level contact. A `conflict` message contains the level of a domain and the address of a contact, and is used to repair inconsistencies.

On receipt of a `sync` message, agent R determines if it is in the specified domain. If not, R returns to the sender a normal update message containing the attributes of the smallest common domain. If R is in the domain, it compares both the attributes of the left child domain and the right child domain with those of its own. Both cases are handled in the same way, so we look at just the left child domain. If the attributes of the left child domain were null, or if the existing attributes share the same contact, then R adopts the attributes into its corresponding domain information. If the contact in the domain information is different from the contact in the message, R sends a `conflict` message to one of the two contacts, containing the address of the other one. On receipt of a `conflict` message, the agent determines the branching domain from which the contact was calculated, and sends a `sync` message for that domain to the conflicting contact.

For example, in Figure 4, $A1$'s contact sends a `sync` message to $A2$'s contact. Then, after comparing the attributes of its child domains with those of $A1$'s child domains, $A2$'s contact detects conflicts in both child domains. It then sends a `conflict` message to the contacts of both $B1$ and $C2$, starting two parallel merges. The message to $B1$'s contact contains $B2$'s contact, and on receipt $B1$'s contact transmits to $B2$'s contact the attributes of $E1$ using another `sync` message. On the receipt of the `sync` message, $B2$'s contact adopts $E1$ as its right child domain. In parallel with all this, $C1$'s contact adopts $G2$ as its right child. After this, the normal update protocol fixes all attributes for all the agents involved.

When applied to two separate Willow trees, the TMP protocol *zippers* both trees together in $O(\log N)$ parallel steps. But the TMP protocol also fixes internal inconsistencies between agents. In particular, when an update message arrives from a peer domain with a conflicting contact, the TMP protocol is started to fix the inconsistency.

Not all inconsistencies can be detected this way. For example, it is possible for an agent in a domain X to know about a peer domain Y, while the contact for X does not. The agents in the peer domain may not know about X, or even if they did, the contact for Y may not. Thus there is no communication between domains X and Y.

The problem is resolved as follows. The Willow attribute update protocol periodically sends update messages at configured intervals, even in the absence of updates (in which case the message will have an empty set of diffs). Currently, Willow is configured to do so every 10 seconds. If an agent in X has not received an update (either directly from an agent in Y, or through multicast in the local domain) for more than 20 seconds, it sends an update for X to a friend in Y. This friend will multicast the attributes in Y, and so the contact for Y will learn about X. That contact in turn will send an update to the contact of X, which will then multicast the attributes of Y within X. After this, all agents in X and Y will know about each other.

An agent that has not received an update for more than 20 seconds will continue to sends updates to the peer domain every 10 seconds. However, if more than 60 seconds have passed, the agent will remove the attributes for the peer domain, and considers it failed.

V. Evaluation

A fully operating Java implementation of Willow is currently available. In order to investigate the scaling issues of Willow, we conducted a study of Willow on a simulated network. In each experiment we placed nodes uniformly at random on a 250x250 millisecond Euclidean plane. The network latency between any two nodes is determined by their Euclidean distance. In these simulations we assume that the bandwidth is constant across the network.

In Figure 5 we show the maximum end-to-end delay of multicasting a small message from a random

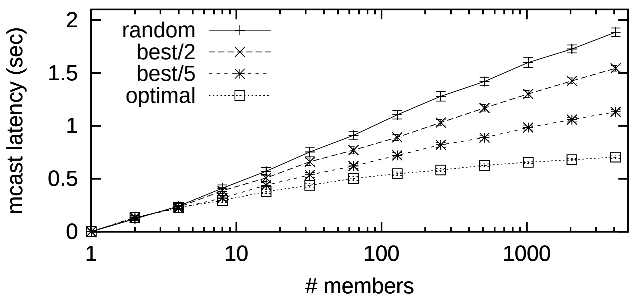

Fig. 5. Multicast latency as a function of the number of agents for different friend selection strategies. The error bars denote 99% confidence intervals.

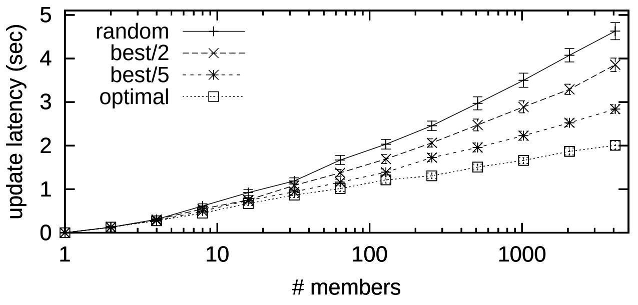

Fig. 6. Update latency as a function of the number of agents for different friend selection strategies.

node to all other nodes, which we expect to grow $O(\log N/\log\log N)$. We used five different strategies for agents selecting friends in peer domains. They are

- *random*: use a random agent in the peer domain.
- *best/2*: use the closest of two random agents.
- *best/5*: use the closest of five random agents.
- *optimal*: use the closest agent in the peer domain.

The *optimal* friend selection strategy is not practical, as there is no cost-effective way for all agents to determine their nearest-by peer agent, but serves as a base line. (In fact, the simulation study was limited by investigating this aspect.) We can see, however, that trying random peer agents over time and maintaining the one with the lowest latency converges fairly quickly to an optimal latency.

In Figure 6 we show the end-to-end delay of updates, that is, how long it takes between the time that an update is made at a random agent and the time at which all other agents have learned the new root-level aggregate. We simulate a worst-case scenario in which the root-level aggregate indeed changes. Depending on the application this can in fact be a rare event. From a complexity analysis study we expect the latency to grow $O(\log^2 N/\log^2\log N)$. Load studies omitted here show that load is well balanced across the agents.

VI. Conclusion

The Willow protocol represents two contributions over previous work. First, Willow supports all of DHT routing, multicast, publish/subscribe, and aggregation in one simple, location-aware protocol. The aggregation facilities allow for a wide range of queries over the data. Second, Willow includes an efficient tree merging protocol that allows disjoint trees to merge in $O(\log N)$ parallel steps, and also repairs Willow trees efficiently when damaged by churn.

Willow is implemented in 2300 lines of Java code, excluding the SQL parser and engine (representing close to 10,000 lines of Java). Initial experience and simulation results indicate that Willow scales well.

References

[1] R. van Renesse, "Scalable and secure resource location," in *Proc. of the Thirty-Third Annual Hawaii Int. Conf. on System Sciences*, Los Alamitos, CA, Jan. 2000, IEEE, IEEE Computer Society Press.

[2] R. van Renesse, K. Birman, and W. Vogels, "Astrolabe: A robust and scalable technology for distributed system monitoring, management and data mining," *ACM Transactions on Computer Systems*, vol. 21, no. 2, May 2003.

[3] A. Bozdog, R. van Renesse, and D. Dumitriu, "SelectCast: A scalable and self-repairing multicast overlay routing facility," in *Proc. of the First ACM Workshop on Survivable and Self-Regenerative Systems*, Fairfax, VA, Nov. 2003.

[4] C. Greg Plaxton, Rajmohan Rajaraman, and Andrea W. Richa, "Accessing nearby copies of replicated objects in a distributed environment," in *ACM Symposium on Parallel Algorithms and Architectures*, 1997, pp. 311–320.

[5] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the XOR metric," in *Proc. of the First International Workshop on Peer-to-Peer Systems (IPTPS'02)*, Cambridge, MA, Mar. 2002.

[6] K. Albrecht, R. Arnold, and R. Wattenhofer, "Join and leave in peer-to-peer systems: The DASIS approach," Tech. Rep. 427, Dept. of Computer Science, ETH Zurich, Nov. 2003.

[7] R. Bhagwan, G. Varghese, and G.M. Voelker, "Cone: Augmenting DHTs to support distributed resource discovery," Tech. Rep. CS2003-0755, UC, San Diego, July 2003.

[8] P. Yalagandula and M. Dahlin, "A scalable distributed information management system," 2003, In submission.

[9] Z. Zhang, S.-M. Shi, and J. Zhu, "SOMO: Self-Organized Metadata Overlay for resource management in p2p DHT," in *Proc. of the Second Int. Workshop on Peer-to-Peer Systems (IPTPS'03)*, Berkeley, CA, Feb. 2003.

[10] R. Huebsch, J.M. Hellerstein, N. Lanham, B.T. Loo, and S. Shenker, "Querying the Internet with PIER," in *Proc. of the 19th Int. Conf. on Very Large Databases (VLDB)*, Sept. 2003.

[11] 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. of the 11th Int. Workshop on Network and Operating System Support for Digital Audio and Video*, Port Jefferson, NY, June 2001.

[12] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-bandwidth multicast in a cooperative environment," in *Proc. of the 19th ACM Symp. on Operating Systems Principles*, Bolton Landing, NY, Oct. 2003.