NEVRLATE: Scalable Resource Discovery

Ajay Chander *
Computer Science Department
Stanford University
ajayc@cs.stanford.edu

Steven Dawson, Patrick Lincoln, David Stringer-Calvert †
Computer Science Laboratory
SRI International
{dawson,lincoln,davesc}@csl.sri.com

Abstract

A scalable and expressive peer-to-peer (P2P) networking and computing framework requires efficient resource discovery services. Here we propose NEVRLATE, for Network-Efficient Vast Resource Lookup At The Edge, an efficient organization of directories or directory mirrors, providing a scalable distributed resource discovery service. NEVRLATE organizes directory servers in an approximate two-dimensional grid, or a set of sets of servers, for registration to occur in one 'horizontal' dimension, and lookup to occur in the other 'vertical' dimension. The payoff of organizing n servers into a structure like this is to achieve $O(\sqrt{n})$ message complexity for registration, and nearly constant complexity lookup. At extra cost NEVRLATE can provide fault tolerance, high availability and security, anonymity, and privacy. The protocol described can be seen as a way to organize Gnutella supernodes, or as a performance extension of Freenet's architecture. In addition, it supports expressive lookup mechanisms, and may provide a basis for a truly scalable worldwide infrastructure for the semantic and the extended web.

1. Motivation

Peer-to-peer computing and networking, and more generally any massively distributed service, requires several layers of infrastructure. First, a distributed base of computing, storage, and communication capability must exist. Second, in order to have a thriving global structure, the parties controlling the digital and infrastructure resources must be motivated to participate in the P2P network (*e.g.*, millions of teenagers join in order to share MP3 files to listen to, or the hundred thousand employees of a multinational corporation join to further their employer's interest.) Finally, there must exist mechanisms that are aware of and utilize this global space of computational resources to provide facilities for efficient lookup, wide-area programming, security, fault tolerance and so on.

In this paper, we focus on distributed directories for distributed resources. Informally, we say that a directory is a mapping from some *description* of a resource, to the *location* or *other information* about a resource. The notion of location can be generalized to include any way of satisfying the request, but for our purposes we will focus on URI [14] as the notion of location. A resource description can include simple text strings like "King Lear", Jini [6] service descriptions, or hashes computed from such things. Directories are fundamental to daily life, the Internet, and just about any future service or computing platform. A commonly used directory, the telephone book, maps names to phone numbers. The Domain Name Service (DNS) maps domain names like `www.csl.sri.com` to IP addresses. Jini provides a LAN-scale lookup service. Google (`http://www.google.com/`) provides a lookup service from keywords to URLs. Napster provided a mapping from hashes of song names to IP addresses where those songs can be obtained. Unfortunately, none of these services provides all the features one might want in a global directory service:

• Distributed. There is no single point of failure and no single point to sue out of existence. Distributed services can be more scalable, more efficient, more available (through fault and intrusion tolerance), and more secure than localized services.

• Internet Scale. The system should have support for Internet scale communities. IPv6, JXTA [13], and Gnutella UUDI contemplate addressing up to 2^{128} devices or services.

• Multi-hierarchical. No single name based hierarchy is imposed on the structure of the network. Multi-hierarchical systems allow multiple conflicting viewpoints of organization to be superposed for efficiency, structure, and function. Hierarchies are good choices in certain contexts; DNS is a wonderful thing, and the GeoWeb (`http://www.dgeo.org/`) and The Open Directory (`http://dmoz.org/`)

* Supported in part by DARPA grant N66001-00-C-8015
† Supported in part by the Office of Naval Research under grant N00014-01-1-0837

0-7695-1582-7/02 $17.00 © 2002 IEEE
IEEE COMPUTER SOCIETY

provide orthogonal hierarchies with significant interest and merit, but not all services and uses fall into strict hierarchy tied to naming.

• Efficient. For network efficiency, utilizing relatively few messages of relatively small size is crucial. Another desirable is (local) storage efficiency. Based on the application, computational efficiency of the individual host may be a concern, but proxy-based architectures could preempt this issue.

• Reliable. When a resource is published by a member of the network, and after some time that resource is requested, will the resource always be found? Google is an example of a relatively reliable system of Internet scale: if Google has spidered a page, and you search for keywords in that page, you are very likely to obtain a pointer to that page, perhaps among many others.

• Responsive. When a resource is published, it becomes accessible after a relatively short delay. When a resource becomes unavailable, it is no longer returned in responses to queries after a relatively small delay. Ideally, one could imagine real-time responsiveness, but more realistically seconds of delay may be acceptable and achievable. Internet search-type lookup services tend to run with days or months of delay.

In this paper, we present NEVRLATE, for Network Efficient Vast Resource Lookup At The Edge, whose sweet spot is providing directory service for massive numbers of participants (millions and more) with resources that are

• Fairly stable (do not disappear, change, or move with higher frequency than queries are made)

• Simply described (the descriptions are small and structured)

Examples of domains where these assumptions hold include phone books, Internet sharing of files (e.g. KaZaA, Gnutella, Napster), business-to-business exchanges, service registries, open-source software repositories, and facilitated software agent systems.

1.1. Related Work

Gnutella [1] is the foremost large-scale, fully decentralized directory and distribution system running on the Internet. However, it faces serious scaling and reliability challenges. As Gnutella networks get large, about half of the network traffic sent over Gnutella is overhead to support the lookup of a resource. Gnutella requests have a time-to-live (TTL) feature that limits the virtual topology reach of a request to some small constant number of hops K, which may be 5 or 7. Thus, if a resource is published in a large network, it will not be found by lookups from most participants unless the neighbors of the publisher at distances at most $K, 2K, \ldots$ lookup that resource. A popular resource has a small effective K, and will be propagated around the network to most clients. For this to be a viable strategy, there must be a great number of common interests among Gnutella servers. That is, if you are to find something, it must be of interest to a K-hop neighbor of yours.

Freenet [5] is another directory and document sharing service with better scalability, where requests are sent along a chain. A Freenet server receiving a request forwards that request to one of its neighbors that is the most likely to have the resource in question. Freenet scales much more gracefully than Gnutella, but provides only a limited sort of directory service (which works well enough for static entities). Since paths are constructed probabilistically, and since no server has any responsibility for maintaining copies of resources, unpopular documents may get "lost" and the lookup process may traverse significant portions of the entire peer space. Since it uses hashes, Freenet also provides strong anonymity and privacy features that are beyond our immediate interests here.

Most Internet search-type lookup services fail to be responsive, and fail to be widely distributed. DNS is hierarchical, and is not rapidly responsive. Jini is not designed to scale beyond the LAN. File-distribution services that rely on a central index (*e.g.*, Napster) are efficient, but provide a single point of failure that can experience hardware failure, power outage, misconfiguration, and concerted legal action. In addition, this simple architecture doesn't scale.

Recent proposals for distributed peer-to-peer platforms have included Chord [7], Globe [11], OceanStore [10] and CAN [8]; we refer the reader to [7] for a comparison of their merits. NEVRLATE provides a greater degree of flexibility in adjusting publication and lookup costs based on the structure of resources (see Section 3.2), and has support for lookups with higher level semantics (Section 3.3).

1.2. Extreme versions of directory services

At one extreme of the design space of P2P directory services is an approach with no caching and no time-to-live constraints on searches. Let's call such a service GIntrovert. To publish a resource, one does nothing. To look up a resource, one sends a lookup message to all neighbors, which forward the message on to all their neighbors, and see if they have the resource in question. If they do have the resource, they send a response back to the looker. In a network of size n, the cost of publication in GIntrovert is constant, $O(1)$, and the cost of lookup is $O(n)$.

At the other extreme in this dimension of design space is a directory service where publication amounts to pushing the directory information to all servers, and lookups become entirely local. Call such a service GExtrovert. To publish a resource, one sends registration messages to all neighbors, which forward the message on to all their neighbors, storing all the directory information locally. In a network of size n,

0-7695-1582-7/02 $17.00 © 2002 IEEE

IEEE COMPUTER SOCIETY

the cost of publication in GExtrovert is $O(n)$, and the cost of lookup is constant, $O(1)$.

NEVRLATE, described below, plays in between these extremes. Section 2 explains the general idea behind NEVRLATE, and Section 3 describes the full system.

2. Basic NEVRLATE

The essential idea behind NEVRLATE is for directory servers to be organized into a logical two-dimensional grid, or a set of sets of servers, enabling registration in one 'horizontal' dimension, and lookup in the other 'vertical' dimension. The payoff of organizing n servers into a structure like this is to achieve $O(\sqrt{n})$ message complexity for registration, and $O(\sqrt{n})$ or better message complexity for lookup.

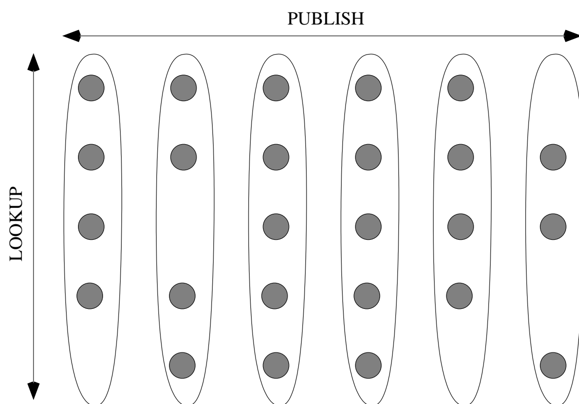

Figure 1. NEVRLATE **Server Organization**

In Figure 1 a set of sets of servers is shown. Each node in the graph is a directory server, which generally will be a single physical computer with good network connectivity. Each set of servers, represented as a vertical cloud, is a relatively well connected set of directory servers, all of which can reach each other member of the set. The set of sets of servers, represented as the entire rectangle, is the entire NEVRLATE network. Each NEVRLATE node maintains several data structures:

- ID (IP address)
- Local resource lookup table (list of resource descriptions and their associated URIs)
- List of server IDs in its set or vertical column
- List of server IDs in other sets, one per set
- Estimate of the total number of NEVRLATE nodes
- Estimate of the total number of sets
- List of estimates of the size of each set
- A peer to contact in case the current set is split.

A directory server may learn of a resource and its location (URI) through various means. For example, the directory server might itself contain some resource of interest, such as a digital copy of the 1681 printing of Shakespear's King Lear. Alternately, some other machine (not a NEVRLATE directory server) may contact the NEVRLATE node with a resource description and location, such as "2001 PVS manuals" ⇒ "`http://pvs.csl.sri.com/manuals.html`". The directory server then registers this resource and its URI in its own local lookup table, and sends the resource description and its location to one server in each other set. The receiving servers in the other sets do *not* forward on the new resource information to all members of their sets. Thus, a server might have a list of local resources that contains "1681 printing of Shakespear's King Lear" ⇒ "`http://NEVRLATE-17.csl.sri.com/KingLear.html`" and "2001 PVS manuals" ⇒ "`http://pvs.csl.sri.com/manuals.html`".

When a directory server is queried about the location of resources, say "What PVS manuals are available?", it sends the query out to all the servers in its set. Since every resource known to NEVRLATE is registered in some server in every set, every resource description is matched against the query and hits are returned.

Thus, in contrast to approaches relying on time-to-live (TTL) or 'geographic' restrictions on publication and query propagation, after time for propagation delay, once a resource description is submitted to NEVRLATE, all future related lookups will discover that resource.

3. NEVRLATE Protocols

We describe here the basic mechanisms of NEVRLATE; the interested reader may refer to Section 4 for pointers to a full specification and a prototype implementation.

3.1. Joining and Leaving NEVRLATE

To support the directory services needed in a dynamic P2P environment, the NEVRLATE network must itself be dynamic, accommodating both new nodes joining the network and existing nodes leaving the network. Although there is no requirement that NEVRLATE users also be NEVRLATE providers (servers), there is also no assumption of a static NEVRLATE network separate from the network of users. NEVRLATE is designed to grow smoothly as additional resources are made available and to adjust automatically as existing resources disappear. Moreover, the protocols that govern the expansion and contraction of a NEVRLATE network are designed to minimize the amount of administrative overhead necessary to maintain the network.

The major issues that must be addressed in the join/leave protocols for NEVRLATE are

- **Maintaining network balance.** As the total number n of nodes in the NEVRLATE network changes, the number

Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02)
0-7695-1582-7/02 $17.00 © 2002 IEEE

IEEE COMPUTER SOCIETY

of sets and the size of each set should be maintained near their targets for the network so that the costs of publish and lookup operations scale as expected.

Based on the number of lookups per publication, and maximum storage and other resource constraints, the optimal shape of the array (number of sets vs. nodes per set) will vary. Whatever the optimal shape, the number of sets must be maintained actively.

• **Sharing and maintaining resource knowledge.** A node that joins the NEVRLATE network must be populated with resource information so that it becomes useful to the network. When a node leaves the network, the remainder of its node set must still have access to the portion of the resource directory it maintained, so that the set can continue to answer all lookup requests directed to it; in addition, nodes (in other sets) for which the leaving node was the publish node must subsequently publish to some other node in the leaving node's set.

• **Maintaining network metadata.** When a node joins or leaves the NEVRLATE network, the network metadata maintained by the nodes (see Section 2) must be appropriately updated.

• **Efficiency.** The communication and processing overhead of joining and leaving the NEVRLATE network must be minimized. In particular, the number of nodes involved in any join/leave operation and the amount of information exchanged between those nodes should be kept as small as possible.

Joining NEVRLATE The basic join protocol begins with the joining node sending a *join request* to an arbitrary NEVRLATE node (in the same way that resource lookup requests are made). This node may further forward the join request until a node willing to receive the new node is found. The receiving node uses its current estimates of the sizes of each set in the network to determine where (to which set) the joining node should be assigned to best maintain network balance. In the process, it may determine that the number of sets needs to be increased, possibly resulting in a *set splitting* operation (described later). After determining the set to which the joining node will be assigned, the receiving node forwards the join request to its publish peer for that set (which may be itself), referred to here as the *target node*. The target node then sends to the joining node (1) the IDs of all the nodes in its set and (2) the IDs of all its publish peers (one node in every other set). The joining node then joins the target node's set by notifying each node in set (1). Next, the joining node forms its own set of publish peers by requesting from each node in set (2) the ID of a random node in that node's set. In the process, each node in set (2) replaces the ID of its publish peer for the joining node's set with the ID of the joining node. This ensures both that the joining node becomes a publish peer of another node (and thus can receive publish requests that populate it with resource directory entries) and that there is diversity among the publish peer sets of the nodes in any given set. The latter property is important in maintaining viable publish peer sets when nodes leave the network. Initially, the joining node's portion of the resource directory is empty, and it will be populated with resource directory entries only by receiving publish requests.

Leaving NEVRLATE Unlike joining, which always happens as the result of an explicit request, a node may leave the NEVRLATE network either explicitly (by notification) or implicitly (by failure). In either case, two main problems must be addressed: (1) the loss to the leaving node's set of the leaving node's portion of the resource directory, and (2) the potential loss of a publish peer for nodes in other sets.

The first problem poses a significant challenge. The leaving node's portion of the directory may be quite large and, hence, costly to recover. Even if the node leaves explicitly, it would need to communicate (distribute) its portion of the directory to the remaining members of its set. But if the node fails, this sort of full recovery is not even an option. One solution is to ignore the full recovery issue, and instead enhance the lookup protocol to perform incremental, on-demand recovery, in which lookup requests unanswerable by an incomplete node set are forwarded to another set (or sets) and the answers added to the first set. Another solution is to rely on other approaches to fault tolerance, such as replication, as described in Section 3.5.

The solution to the second problem is essentially the same whether the leaving node leaves explicitly or implicitly. Only the nodes that have the leaving node as a publish peer know it, and thus they cannot be explicitly notified when the leaving node leaves the network. Instead, the nodes that have the leaving node as a publish peer must infer the loss of the leaving node from the failure of the publish protocol. Upon realizing the failure, each such node can determine a replacement publish peer for the leaving node's set by querying the members of its own set — the join protocol ensures that some member of the set will have a publish peer different from the leaving node, and thus, all nodes will continue to have a complete set of publish peers after the leaving node has left the network.

Set Splitting To maintain acceptable dimensions in the NEVRLATE network, it is sometimes necessary to increase the number of sets. To increase the number of sets while maintaining an acceptable range of set sizes, NEVRLATE uses a *set splitting* operation to divide one set into two sets of roughly equal size. Of course, the two sets produced in this way are incomplete — neither maintains the complete directory. To address this issue, each node in each division of the split records the ID of a node in the other division.

Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02)
0-7695-1582-7/02 $17.00 © 2002 IEEE

IEEE COMPUTER SOCIETY

When a lookup request to a node in one division cannot be answered by that division, it is forwarded to the other division, and the results are then stored by a node in the first division in addition to being returned to the querying node. If a set resulting from an earlier split is subsequently split, only information pertaining to the latest such split is maintained directly, and any remaining incompleteness in the directory in those sets after possible multiple splits is handled in a manner similar to that of lost nodes.

Set Absorption Just as the number of sets may need to be increased as the network grows, the number of sets may also need to be decreased as nodes leave the network. To reduce the number of sets in the network, NEVRLATE uses *set absorption*, whereby an existing set (perhaps an incomplete set or one of smaller cardinality) is simply recycled by eliminating the entire set and rejoining each node from that set into the network as a fresh node.

3.2. Ordered Search

In the basic NEVRLATE system, searching for an available resource is performed by querying every server in the querying node's set, yielding an $O(\sqrt{n})$ lookup cost. This cost is necessary, as although we know that if the resource has been published, *some* node in the set will know about it, we do not know *which* node. By introducing some basic knowledge of 'what belongs where' and depending on very low flux of nodes into and out of the network, we can reduce the $O(\sqrt{n})$ cost to $O(\log n)$ or $O(1)$.

Many resources have a natural ordering relation; for example, telephone numbers can be ordered by $\leq$ over the natural numbers and song titles by lexicographic ordering. For any collection of resources that have a defined ordering (or can be hashed into a sequential linear finite order) we can overlay that ordering on the members of the set in the NEVRLATE system, creating a sequence of server nodes in each set. Enforcement of the ordering is performed at publication time, by inserting the new resource on any server that maintains the ordering within the sequence. Lookup is then performed by binary search, yielding worst-case $O(\log n)$ lookup.

Optimizing this approach, we can define subsets of the ordering range for which each server in the sequence is responsible in a manner similar to bucket sort. The servers can be assigned ranges within the ordering, and requests are published directly to the node responsible for that region of the ordering. Lookup requests can subsequently be directed immediately to the responsible server, giving a constant cost of lookup in the usual case. When the resource descriptions are hashes of information, approaches similar to LH* for distributed linear hashing can be used to manage distributed growing hash tables [15].

In declaring each server in the set to be responsible for a particular part of the ordering, we made no assumptions on the distribution of resources within the ordering. A reasonable approach is to make the subsets of the ordering equal in size, but if more is known about the distribution of resources within the ordering, the subsets could be of differing sizes — optimized so that the number of resources known to each server is approximately equal. However, such distribution of the resources themselves may not be interesting — what is often more interesting is the queries for the available resources. If a certain pattern of queries is expected, the resources may be balanced across servers to make the number of queries to each server approximately equal.

Ordered search optimizations of NEVRLATE are particularly effective when the membership is static. In a NEVRLATE system with a dynamic membership, the join/leave protocols must be enhanced to maintain the orderings imposed. In a system with a small join/leave rate, simple variations on the protocols described earlier will suffice, but in highly dynamic systems, the overhead required to hand off responsibility may well overwhelm the publish/query traffic.

To cope with highly dynamic systems, further optimizations of NEVRLATE utilize a ramped hashing approach. In the first optimization, we considered balancing the resource index size or balancing the query load, with implicit assumptions of equal reliability and stability of the servers in the set. This could be considered equivalent to a hashing function with a flat distribution. If these assumptions are flawed, we need an allocation of resources to servers that favors *stable* servers — those that have good network connectivity, and have been in the NEVRLATE network for longer. Introducing this notion, we obtain a ramped (or curved) distribution from the hashing function. A non-flat assignment of hashing function results to servers can be advantageous in general for distributed linear hashing. Further, the curvature of the hashing function can be dynamic itself — responsive to the current join/leave rates, the profile of resources published, and the queries being performed.

3.3. Lookups with Higher-Order Semantics

The discussion so far has assumed that the resource index is an association of key/value pairs and that lookup corresponds to equality of the given key with some stored key. NEVRLATE is, however, not restricted to such a simple approach to resource discovery. A more general approach is to allow the query to contain both a key and an equivalence function. Commonly used equivalences could be predefined in the system (e.g., soundex matching, substrings) and others could be specified by regular expressions, or using a general-purpose programming/scripting language. These equivalences might be used implicitly, while other equiva-

Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02)
0-7695-1582-7/02 $17.00 © 2002 IEEE
IEEE COMPUTER SOCIETY

lences might be sent explicitly with the query.

An application under development (see Section 4) involves the use of NEVRLATE to provide distributed lookup services for software agents. An agent can use NEVRLATE to publish a description of its capabilities, similar to the Jini lookup service for local area networks, a description of its API, or a formal description of the input-output relation. Lookup of agents can be very simple and direct (exact match for name or exact match on API), or can be done in a *delegation-based* computing style.

Delegation-based computing presumes the availability of a large set of agents that have published a machine-readable description of the service they can perform in some logic. Agents are invoked by some process delegating some computation, described as a goal in that same logic. A facilitating service then attempts to prove the goal from the published axioms representing capabilities of agents using logical inference. A successful proof can then be read as a plan for the agents to be invoked. The logic involved can be as simple as propositional classical logic (which might reflect the types of arguments, like Integer and String), first-order classical logic (where input-output relation might be defined), or more exotic logics like higher-order logic (where one can reason about programs that take programs as arguments) and linear logic (or other resource-sensitive logics).

NEVRLATE provides a scalable means to publish such logical descriptions of agent capabilities, and then to perform distributed facilitation of agent-based computing. For example, one can delegate a mathematical expression "38 + sqrt (16)" and an agent that knows how to evaluate sqrt may be invoked to reduce "sqrt 16" to "4", and an agent that knows how to evaluate + can be invoked on the result to produce the final answer of "42" which is returned as the result. A more substantial example might involve large bioinformatic calculations involving publicly available databases of proteomic strings, certain Boyer-Moore fast string-matching (BLAST) agents, and certain other evaluation agents to choose among many matches. The key novel features of NEVRLATE include the global reach (if there is an agent able to compute something somewhere in the network, it will be found) and efficiency. Note that agents may or may not be mobile. An agent might provide an interface to a large immobile database, or it might be a stateless bit of code that can be copied arbitrarily, or it might be mobile and contain state that should not be copied, but that can be moved.

3.4. Resource Partitioning

One approach to scalable resource discovery is to partition the global resource set into categories, and provide a resource discovery mechanism for each category or some collection of categories. A meta-process directs queries to the relevant resource discovery mechanism, minimizing the amount of resource space that must be searched to locate items of interest. Any type of categorization may be used — for example, semantic, geographical, or perhaps network-connectivity oriented.

To enable collections of NEVRLATE systems to cooperate in such a manner, a meta-protocol is used to locate resources on *peer*-grids. Resource publication is still maintained locally within the grid, but if a resource cannot be found on the current grid, the request may be forwarded to any number of peer-grids. This is placed under user control, so that the query may be forwarded to specified alternate grids. The forwarding protocol has an effect similar to the TTL on queries in Gnutella, where queries are forwarded for a specified number of hops.

3.5. Extensions to NEVRLATE

Here, we briefly sketch extensions to the basic NEVRLATE system that provide security and fault-tolerance. We are continuing to experiment with additions to the basic system, and will report on them further in future work.

A simple strategy that protects against random server faults is based on redundancy while publishing content. Nodes register their content with more than one server in each set, and in the case of ordered sets more than one server shares the responsibility for any subset of the ordering. The departure of a node in a fault-tolerant NEVRLATE system might not affect the span of the resource directory as the lost information may already be present with different members of its set. However, to be resilient against multiple failures in a set, a node can share the intersection of its directory entries with the entries of the departing node, with a randomly chosen small subset of members in its set. Moreover, as in the basic system, the lookup protocol may be enhanced to provide incremental, on-demand recovery in case some of the lost information had not been duplicated within the departing node's set.

The NEVRLATE system may potentially be abused by byzantine users who publish non-existent resources, misleading resource descriptions, or make casual resource requests and provide byzantine responses to queries. The NEVRLATE system, like any other general directory service is susceptible to such behavior. General content tracking and auditing mechanisms including watermarking and quality-of-service ratings for the directory nodes can enforce "good-citizen" behavior from participating peers. Bandwidth-based queuing algorithms may be used to assure fairness among competing requests at directory nodes, and thwart resource starvation attacks. With additional expense, redundant queries could be performed, and hybrid Byzantine-agreement algorithms could be employed [12]. Further, in a PKI-based world, authentication of publishers,

Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02)
0-7695-1582-7/02 $17.00 © 2002 IEEE

IEEE COMPUTER SOCIETY

servers, and subscribers could be employed to reduce the possibility of a distributed denial of service attack.

4. Implementation

The NEVRLATE protocol has been formally specified and a Java prototype implemented [4]. A plugin [4] for the Gnutella client LimeWire was used to populate the NEVRLATE directories with resources existing on the Gnutella network. The specification was done in MuCAPSL [9], a general purpose specification language for group protocols. The rigor of the specification clarified and improved the initial protocol design, and will provide a systematic basis for a formal analysis. The NEVRLATE system has recently been used [3] to provide resource lookups for agents in the Open Agent Architecture [2], which is a general framework for delegation-based computing. This work uses the DARPA Agent Markup Language (DAML) to express NEVRLATE queries, and demonstrates the framework's support for lookups with higher-order semantics.

5. Conclusions and Work in Progress

We have described NEVRLATE, a scalable resource discovery service that occupies a middle ground between a central directory and full replication of directory services. The basic system allows for $O(\sqrt{n})$ publication and lookup cost, although with information about orderings over resources or hash functions, we can obtain constant time lookups at the expense of slightly greater publication cost. The protocol has been formally specified and a prototype implemented; we are continuing to further analyze the evolution of the grid (as a stochastic process) and experiment with extensions for security. This effort will provide the foundations for distributed agent services, transforming the web from its current status as an information resource into a computing resource, capable of independently solving complex problems.

Acknowledgments Latifa Bouabdillah, Grit Denker and David Martin formally specified and implemented the NEVRLATE protocol, and this process greatly benefited the initial design. Imen Atallah implemented the LimeWire plugin. David Martin provided a great usage study by implementing a plugin for the OAA which used NEVRLATE to discover agents within a distributed delegation-based computing architecture. We also thank Mary Baker, Drew Dean, Steven Eker, TJ Guili, Raymonde Guindon, Andrea Lincoln, Petros Maniatis, John Mitchell, Natarajan Shankar, Vitaly Shmatikov and Steven Weiner for interesting related discussions.

References

[1] Clip2, *The Gnutella Protocol Specification*, `http://www.clip2.com`, 2000.

[2] D. L. Martin, A. J. Cheyer, and D. B. Moran, *The Open Agent Architecture: A framework for building distributed software systems*, Applied AI, January-March 1999.

[3] I. Atallah, G. Denker and D. Martin, *Bridging Between the Semantic Web and a Delegated Computing Framework: The DAML-OAA Bridge Agent*, Technical Report SRI-CSL-2002-04, Computer Science Laboratory, SRI International, March 2002.

[4] I. Atallah, L. Bouabdillah, G. Denker, and D. Martin, *Detailed Design and Implementation of NEVRLATE: A Scalable Peer-to-Peer Framework for Resource Discovery*, Technical Report SRI-CSL-2002-03, March 2002.

[5] I. Clarke, O. Sandberg, B. Wiley, and T.W. Hong, *Freenet: A Distributed Anonymous Information Storage and Retrieval System*, International Workshop on Design Issues in Anonymity and Unobservability, ed. by H. Federrath, Springer, NY, 2001.

[6] SUN Microsystems, *Jini Architecture Specification*, `http://www.sun.com/jini/specs/`.

[7] I. Stoica, R. Morris, D. Karger, M. Frans Kaashoek, and H. Balakrishnan, *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications*, ACM SIGCOMM 2001, San Diego, CA, August 2001.

[8] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, *A Scalable content-addressable network*, ACM SIGCOMM 2001, San Diego, CA, August 2001.

[9] J. Millen, *Common Authentication Protocol Specification Language*, Available at `http://www.csl.sri.com/users/millen/capsl`.

[10] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, *OceanStore: An Architecture for global-scale persistent storage*, In Proc. ASPLOS 2002, pp. 190–201.

[11] A. Barker, E. Amade, G. Ballintijn, I. Kuz, P. Verkaik, I. van der Wijk, M. van Steen, and A. Tanenbaum, *The Globe distribution network*, In Proc. 2000 USENIX Annual Conference, pp. 141–152.

[12] P. Lincoln and J. Rushby, *A Formally Verified Algorithm for Interactive Consistency Under a Hybrid Fault Model*, Fault-Tolerant Computing Symposium, FTCS 23, 1993.

[13] SUN Microsystems, *Project JXTA*, `http://www.jxta.org/`, 2001.

[14] T. Berners-Lee, *Universal Resource Identifiers in WWW, A Unifying Syntax for the Expression of Names and Addresses of Objects on the Network as Used in the World-Wide Web*, RFC 1630, CERN, June 1994.

[15] W. Litwin, M.-A. Neimat and D. Schneider, *LH* — A scalable, distributed data structure*, ACM Transactions on Database Systems, 1996.

Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02)
0-7695-1582-7/02 $17.00 © 2002 **IEEE**

IEEE COMPUTER SOCIETY