In Proceedings of IEEE CollaborateCom 2005, San Jose, California, U.S., December 2005

On-Demand Overlay Networking of Collaborative Applications

Cheng-Jia Lai †
Fuji Xerox Palo Alto Laboratory, Inc.
cjlai@fxpal.com

Richard R. Muntz
Computer Science Department, UCLA
muntz@cs.ucla.edu

Abstract

We propose a new overlay network, called Generic Identifier Network (GIN), for collaborative nodes to share objects with transactions across affiliated organizations by merging the organizational local namespaces upon mutual agreement. Using local namespaces instead of a global namespace can avoid excessive dissemination of organizational information, reduce maintenance costs, and improve robustness against external security attacks. GIN can forward a query with an O(1) latency stretch with high probability and achieve high performance. In the absence of a complete distance map, its heuristic algorithms for self configuration are scalable and efficient. Routing tables are maintained using soft-state mechanisms for fault tolerance and adapting to performance updates of network distances. Thus, GIN has significant new advantages for building an efficient and scalable Distributed Hash Table for modern collaborative applications across organizations.

1. Introduction

Modern middleware, such as Web Services [4], CORBA [15] and DCOM [12], facilitates remote object access, where collaborative nodes can make a remote procedure call for accessing an object (also known as web resource or service) installed at each other as easily as accessing a locally installed object. Those nodes are typically installed in clusters each of which represents an organization, e.g. business corporations, governments, special-purpose/interest communities, regions of online multiplayer games, and peer-to-peer (P2P) file-sharing systems [6][7][14][24]. An organization is autonomous for managing its node and object installation including location and replication, for high performance, fault tolerance, security, authorization, etc. Despite the autonomy, a node collaborating with other nodes in a decentralized transaction—e.g. by Web Service Choreography [5], or Business Process Execution Language (BPEL) [1]—may need to access objects outside its organization. Thus, upon demands from the current applications, an organization is affiliated with some other organizations such that a node in it can remotely access objects in its affiliates. However, as those application demands are transient, e.g. regulated by mutually agreed policies or contracts [25], the on-demand affiliating between their organizations will not be persistent. In addition, objects can have mobile locations due to dynamic authoring, caching, or replicating. Thus, we are focused on how to facilitate remote access to mobile and transient objects in dynamically affiliated organizations, with a scalable and efficient query-forwarding algorithm.

We assume that each object is assigned a character string as its "name" that is the key for a querying node to find the object on the network. Thus, a namespace for all available objects in the affiliated organizations should be maintained, where the object names are only significant locally in those affiliates. This is different from using a global name service that assigns each object, for example, a Uniform Resource Identifier (URI) [3], which is either a Uniform Resource Locator (URL) or Name (URN). Using a URL tightly binds the object access point to the hostname written in the URL, while using a URN avoids such binding but requires a namespace identifier that must be globally unique. Both the URL and URN schemes need Domain Name System (DNS) to retrieve the IP addresses of the URL hostnames, and the dedicated URN name resolvers for a namespace identifier, respectively. Unfortunately, as recently reported in [16], the currently deployed DNS servers have severe load-imbalance, resulting in poor performance and scalability. Caching among the DNS servers also has low coherency upon updates [17]. To address these problems, it has recently been advocated that a global name service should be implemented using the emergent Distributed Hash Table (DHT) approaches [8][10][20][22][23], which initially aimed at supporting P2P file-sharing systems on large scales. For example, CoDoNS [21] was proposed as a DNS implementation using Beehive [20], which is a DHT with proactive caching. A four-layer Internet naming architecture [2] was proposed using DHT technologies to allow objects to have a globally unique and *flat* name. These global name services can provide high

† The work was previously finished in Computer Science Department at University of California, Los Angeles (UCLA).

performance and scalability in answering queries based on DHTs.

However, deploying a global name service for local objects in a set of affiliated organizations has several drawbacks. First, it excessively disseminates the name registrations across orthogonal namespaces each of which belongs to a set of affiliated organizations where nodes never access external objects. This excessive dissemination results in high overhead for maintaining the namespace, and violates the privacy policies that, typical in practice, disallow unnecessary information leakage. Second, subscribers to a global name service are often required to pay fees for service maintenance and management, while self-installing a local name service makes economy. Third, it is more difficult to generate an object name with global uniqueness, and is unnecessary when local uniqueness suffices. Finally, relying on a global name service poses security threats to those organizations when the global name service is vulnerable to attacks.

Thus, we argue that an organization should instead employ a local name service for addressing its internal objects for its own applications, and that if multiple organizations are affiliated, their local name services should accordingly become merged to reflect a mutual agreement for sharing objects, while the autonomy for managing internal objects is preserved at all times. To provide scalability and efficiency, we also advocate that a local name service be implemented in a DHT approach. However, in addition to those functions for supporting P2P systems, our approach—called Generic Identifier Network (GIN)—supports dynamic creation and removal of *admin links* for affiliation-relationships. In general, an admin link connecting two nodes shows that the two nodes participate in the same namespace. Thus, admin links are used to connect nodes inside an organization for clustering as well as nodes in different organizations for affiliating.

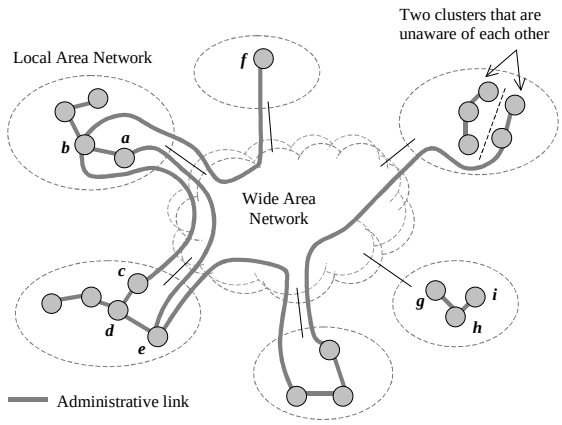

Figure 1. Admin links between nodes

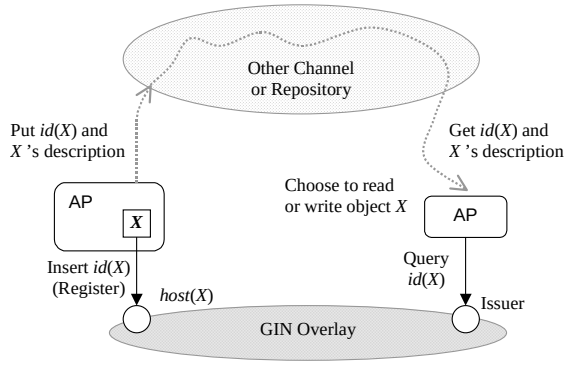

Figure 2. Object registration and query

For example, referring to Figure 1, there are 3 sets of affiliated organizations. Although it is typical for an organization to reside within one local area network (LAN), two or more organizations may coexist in one LAN, either affiliated or not, while one organization may have its nodes distributed in multiple LANs with admin links connecting those nodes. Therefore, the topology of the admin links in a GIN does not reveal which node belongs to which organization, though an admin link can be annotated with a high link-cost if affiliating two organizations. Note that the admin links in GIN are allowed to form a loop, e.g. *a*-*b*-*c*-*d*-*e*-*a*, for fault tolerance and ease of maintenance.

Based on the created admin links, GIN dynamically create some other overlay links to facilitate locating an object by its object name. Like other DHT approaches, a predefined hash function is deployed to generate a fixed-length Object Identifier (OID) from each object name such that an OID has a pseudo-random numeric value. Then, an object registers its OID at its hosting node so that an application process (AP) can use any node in the same namespace to send a query for it by specifying its OID. Objects can be mobile since object names and OIDs need not contain any information for the hosting nodes' network addresses.

For example, referring to Figure 2, an object X is registered at a node y by an Insert command so that y acquires explicit access to X, which can be exercised on behalf of a local or remote AP—y can update or retrieve the contents of X, trigger some actions on X, or simply deliver an event or a message to X. Then, node y is called the *host* of object X, and is denoted by *host*(X). The OID of X is denoted by *id*(X). In GIN, an object X is allowed to have multiple replicas that share the same OID, i.e. *id*(X), registered at different hosts. A query for *id*(X) from the issuer, i.e. the node issuing the query on behalf of a querying AP, is sent to one of the hosts of X, and the query is likely to be received by the host nearest to the issuer.

Our approach shares many desirable properties with other DHT proposals. Suppose a namespace involves n nodes and $O(n)$ objects where a node has $O(1)$ admin

links. Then, the average routing table size is $O(\log n)$ per node with high probability, which is scalable. Since the overlay links in GIN are built with the awareness of node locations, like Tapestry [8] and Pastry [23], it can provide a latency stretch as low as $O(1)$ in forwarding a query with high probability. Its overlay network is self-configurable, and resilient with soft-state techniques.

Nonetheless, unlike other DHT proposals, GIN can dynamically adapt to the performance updates of those link-costs that are typically dependent on the end-to-end transmission delays between nodes. In addition, two GINs with n_1 and n_2 nodes ($n_1 \le n_2$), and $O(n_1)$ and $O(n_2)$ objects, respectively, will merge by $O(n_1 \log n_2)$ messages if $n_1 << n_2$, or $O(n_1)$ otherwise, on average with high probability. This implies that inserting a new node into a GIN with n nodes takes $O(\log n)$ messages only. Partitioning a GIN and removing a node require similar numbers of messages to those for merging two GINs and inserting a node, respectively.

2. GIN Infrastructure

Each node, say u, is assigned a random but unique Node Identifier (NID), denoted by $id(u)$. The OID and the NID share the same format with a fixed number of bits, e.g. 128 bits. NID's and OID's are all considered pseudo random, typically generated by a hash function that takes as input the text string of the node or object name, properties, or contents, and outputs a hash code equally likely to be any in the range of possible values.

A node can send and receive messages to and from any other node through an underlying network. The underlying network defines a set of network addresses, assigns each node a unique network address, and is capable of delivering a message from any node to any other node that is identified by its underlying network address. (We write "address," "network address," and "underlying network address," interchangeably.) The underlying network is typically a transport-layer data transmission service, e.g., TCP. Thus, the node address in GIN can be a combination of its IP address and TCP port number for the TCP socket. GIN identifies a node by its node address and uses a node address to refer to any remote node. When the context is clear, we simply write the address of a node x as x.

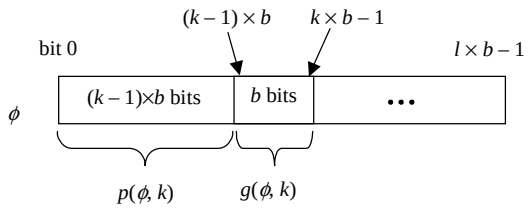

Figure 3. The level-*k* prefix and digit

The routing tables at those GIN nodes contain two data structures: links, and *object pointers*. An object pointer [OID, host] means that an object with that OID has been registered at that host. In this section, we will describe how to use the data structures to find an object with a specified OID, and how to maintain these data structures dynamically.

2.1. Basics

Both the OID and the NID have $l \times b$ bits, where b is a small integer and l is the number of *levels*. Assume that $l = \Omega(\log n)$ where n is the total number of nodes, e.g., $b = 4$ and $l = 32$. Both the OID and the NID have $l \times b$ bits.

Refer to Figure 3. For each level $k = 1, 2, \ldots l$, we define the *digit* of an ID (either an OID or a NID) to be the number represented in binary by those b bits from bit $(k - 1) \times b$ to bit $k \times b - 1$, where the bits are indexed in order from left to right, starting with 0. This digit is the "routing goal" at level k, and is denoted by $g(\phi, k)$ for an identifier ϕ. Note that $g(id(x), k)$ is briefly written as $g(x, k)$ for a node x, and $g(id(X), k)$ as $g(X, k)$ for an object X.

We define the *prefix* of an identifier ϕ at a level k to be the ordered sequence of the first $k - 1$ digits of ϕ, which is denoted by $p(\phi, k)$. Similarly, $p(x, k) = p(id(x), k)$ for a node x, and $p(X, k) = p(id(X), k)$ for an object X. Note that $p(\phi, 1)$ is always a null prefix, denoted by ε, that contains no digit, and $p(\phi, l + 1) = \phi$, for convenience. For an identifier θ, when $p(\theta, k) = p(\phi, k)$, we say that θ matches $p(\phi, k)$. Clearly, all identifiers match ε at level 1, while a node x is the only node with an NID matching $p(x, l + 1)$. In addition, for a prefix α at level k and for a digit δ, we write $\alpha\delta$ to refer to the prefix at level $k + 1$ such that the first $k - 1$ digits are α and the k-th digit is δ.

For a level-k prefix α, we define the *class* of α (also called the α-class) as the set of all those nodes with an NID matching α. The α-class is denoted by $C(\alpha)$. For convenience, the class of a prefix $p(\phi, k)$ for an identifier ϕ is denoted by $C(\phi, k)$. Similarly, $C(x, k) = C(id(x), k)$ for a node x, and $C(X, k) = C(id(X), k)$ for an object X. Clearly, two nodes in the same class of a level-k prefix are contained in the same class at any level $j < k$. In addition, since NID's are random, the average number of nodes in a class at level $k + 1$ is expected $n \times 2^{-b \times k}$, only $1/2^b$ of the number at level k, for $k \ge 1$, while the class of the null prefix ε contains all nodes. For a node x, $C(x, l + 1)$ only contains x since $id(x)$ is unique.

However, since the expected cardinality of $C(\phi, k+1)$ is $n \times 2^{-b \times k}$, when k is small, recording $C(\phi, k+1)$ in the routing table at a node is not scalable—a scalable algorithm should only take into account a subset of $C(\phi, k+1)$ at a node.

Thus, respect to an OID ϕ and a node y, we define the *scope* $S(k, \delta)$ locally at y for each level $k = 1, 2, \ldots l$, where $\delta = g(\phi, k)$ and $S(k, \delta)$ is a subset of $C(\phi, k+1)$ such that $S(k, \delta)$ contains those both in $C(\phi, k+1)$ and somehow known to x. As will be described in 2.3, $S(k, \delta)$ is updated such that the nearest node in $C(\phi, k+1)$ to x can be contained in $S(k, \delta)$ while the average cardinality of $S(k, \delta)$ is $O(2^b)$, with high probability.

In the absence of a complete distance map between nodes, a node will periodically estimate its network distances to those nodes recorded in its $S(k, \delta)$ for any k and δ. The network distance (i.e. the link cost) between two nodes x and y is denoted by $d(x, y)$, and is typically measured half as the round-trip delay for a probing packet that bounces between x and y. A node caches the measured distances in a partial distance map at its local memory, and periodically updates these distances. We assume that $d(x, y) = d(y, x) \geq 0$, and $d(x, y) = 0$ iff x is y.[1] In addition, if y is unreachable from x for any reason, x resets $d(x, y)$ to infinity. Periodically updating $d(x, y)$ at x incurs little overhead since nodes in distributed systems typically need to exchange keep-alive messages periodically.

Each link belongs to a level; there are two types of links: I (informative) and F (forwarding). An I-link at a level k indicates that the nodes at the link endpoints consider each other *neighbors* at level k. All neighbors at a level k must share the same level-k NID prefix, and will exchange the routing information for the k-th NID digits via the corresponding I-links. A node uses a variable $I(k)$ in its routing table to record all of its level-k neighbors. Note that there are only l different $I(k)$ variables, i.e. for $k = 1, 2, \ldots l$, while each of these variables could contain $O(2^b)$ node addresses with high probability, assuming that each node has $O(1)$ admin links.

On the other hand, an F-link is directional, and is associated with a digit δ. The F-links actually provide some chains to forward queries and OID registrations. For each digit δ at a level k, an F-link from a node x to a node y indicates that x considers y the *nearest* node in $S(k, \delta)$. To represent this F-link, a variable $F(k, \delta)$ in the routing table at x is assigned y, while in the routing table at y, a variable $V(k)$ is maintained for all level-k F-links ending at y and thus contains the address of x. Note that a node only has $l \times 2^b$ different $F(k, \delta)$ variables, and it also has l different $V(k)$ variables each of which could contain $O(2^b)$ node addresses.

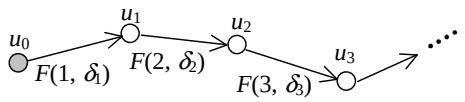

Figure 4. The forwarding chain of F-links

Finally, for any level-k prefix α, we define the *fabric* of α as the subgraph where the vertices are $C(\alpha)$ and the arcs are all those level-k I-links between nodes in $C(\alpha)$. The fabric of α is also called the α-fabric. If the α-fabric contains only one node, forwarding a query in this fabric always reaches the destination host directly (if it exists). Thus, the fabric of a longer prefix with the first $k-1$ digits equal to α is never needed in routing and thus not maintained. Therefore, with n nodes in GIN, each node would likely be contained in only $O(\log n)$ fabrics.

2.2. Handling Objects

An object X registers its OID $id(X)$ at $host(X)$ that will distribute an *object pointer* $[id(X), host(X)]$ to a limited number of nodes such that it is efficient and scalable to reply to a query for $id(X)$. If $id(X)$ has a set of replicas $\{X_i \mid i = 1, 2, \ldots\}$, then $[id(X), host(X_i)]$ is so distributed for each X_i.

Referring to Figure 4, the object pointer of an object X is forwarded along F-links. For convenience, we define $\delta_k \equiv g(X, k)$ for each $k = 1, 2 \ldots l$. Then, $[id(X), host(X)]$ is forwarded along a chain of nodes $u_0u_1u_2\ldots u_l$ where $u_0 \equiv host(X)$, and for $k = 1, 2 \ldots l$, $u_k \equiv F(k, \delta_k)$ recorded at u_{k-1}. Each node in the chain stores a copy of $[id(X), host(X)]$. If a node u_m finds that $F(m+1, \delta_{m+1})$ is null, the forwarding pauses at this node, but will resume if later $F(m+1, \delta_{m+1})$ becomes non-null. In addition, for each $k \in \{0, 1, \ldots m-1\}$, u_k multicasts $[id(X), host(X)]$ to each node $s \in S(k+1, \delta_{k+1})$. However, node s only stores a copy of $[id(X), host(X)]$ but does not forward it. Finally, u_m sends $[id(X), host(X)]$ to the node $r = F(m+1, \delta')$, where δ' is the smallest digit such that $F(m+1, \delta')$ is non-null; r is called the root node of $id(X)$, denoted by $root(X)$. Each node will discard duplicate copies of an object pointer.

To withdraw an object X, $host(X)$ reuses the same forwarding chain (via F-links) that has been used for registering X as in Figure 4, to send an "object-

[1] Since $d(x, y)$ is typically half the round-trip time between node x and y, and since an AP that sends data on top of the transport layer, e.g. by using TCP/IP, it is reasonable (and typical in DHT approaches) that the network distances are assumed symmetric in opposite directions. In addition, to obtain analytic results, it is typically assumed by DHT approaches that for any three nodes x, y, and z, $d(x, y) + d(y, z) \geq d(x, z)$.

withdrawal" message that will remove all existing copies of the object pointer [$id(X)$, $host(X)$]. Similarly, the object withdrawal message is multicast to nodes in $S(k + 1, \delta_{k+1})$ by u_k after the k-th hop, and finally sent to $root(X)$.

A query message for an OID ϕ is also forwarded via the F-links. Refer to Figure 4 again, with u_0 indicating the query issuer, and $\delta_k \equiv g(\phi, k)$ for each $k = 1, 2... l$. Then, starting from $k = 0$, u_k searches in its memory to find an object pointer [ϕ, h] that matches π, where h is the host with minimal $d(u_k, h)$ if more than one [ϕ, h] is found. If a match is found, u_k forwards the query to h that will access an object X with $id(X) = \phi$ and reply to u_0. Otherwise, u_k checks the current value of $F(k + 1, \delta_{k+1})$ at it. If the current value is non-null, u_k forwards the query to the next node $u_{k+1} = F(k + 1, \delta_{k+1})$. If the current value is null, u_k forwards the query to the root node $r = F(m + 1, \delta')$ where δ' is the smallest digit such that $F(m + 1, \delta')$ is non-null. However, if r is u_k itself, u_k alternatively notifies u_0 of an access failure.

2.3. Handling Links

Without a complete distance map between nodes, we design a heuristic algorithm that uses only a small and limited number of distance estimates to construct F-links. Based on the assumption that the level-1 fabric is a connected graph, the heuristic algorithm can ensure that all fabrics be connected graphs, and near nodes in a fabric be connected, with high probability, by I-links while the F-links be created properly.

Referring to Figure 5, the heuristic algorithm works recursively—given any level-k I-links, the F-links on a level k will be created via those I-links while the I-links on level $k + 1$ will be created via the F-links on level k. It starts with the level-1 I-links, i.e. admin links.

Specifically, based on the fabric of a level-k prefix α, we show how (and by what messages) those nodes can create proper F-links respect to a digit δ. Then, as those nodes with the k-th NID digit equal to δ are called the δ-nodes, we show how those δ-nodes in adjacent regions can create new I-links on the next

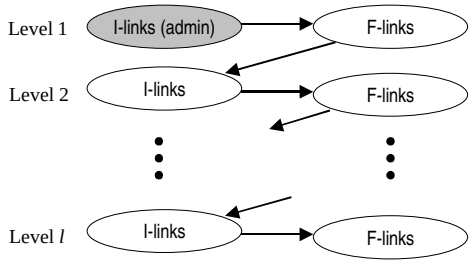

Figure 5. The recursive manner in creating the F-links and I-links

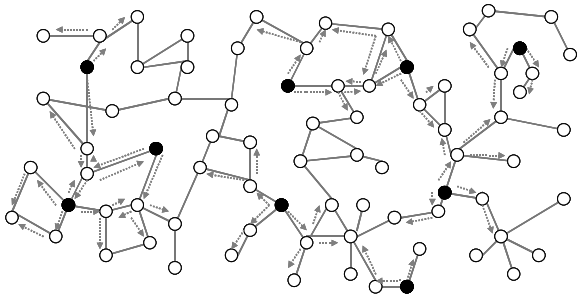

Figure 6. The advertisement for the nearest δ-node is forwarded via I-links.

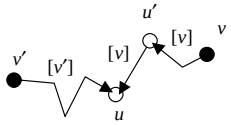

Figure 7. Choosing the nearest δ-node

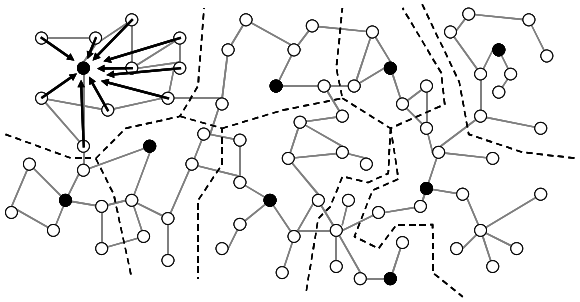

Figure 8. Creating proper F-links

level $k + 1$. The dynamics of the algorithm is shown chronologically, starting from Figure 6 to Figure 11, with explanations in the text.

Figure 6 shows a fabric at level k, of an arbitrary prefix α, where only nodes in class $C(\alpha)$ and I-links between these nodes are shown. Now, we consider an arbitrary digit δ. Then, any δ-node v, i.e. with $g(v, k) = \delta$, is drawn in black, while all the remaining nodes are in white. Following the algorithm, each δ-node v is required to advertise the fact that $g(v, k) = \delta$, via all I-links by which v is connected. The advertisements are denoted by dashed arrows. A receiver u of an advertisement from v now knows that at level k, v is a δ-node, and u also knows that very likely, v is near u, relative to other δ-nodes. Then, u obtains the distance estimate of $d(u, v)$ if that has not been measured or the last measurement has expired.

Referring to Figure 7, if u has received two (or more) advertisements for different δ-nodes, e.g. v and v', u selects the nearest one. Whenever u selects (either for the first time or as an update) the nearest δ-node, say v, then u will forward the advertisement for x (denoted by dashed arrows in Figure 6) via all u's I-links in the α-

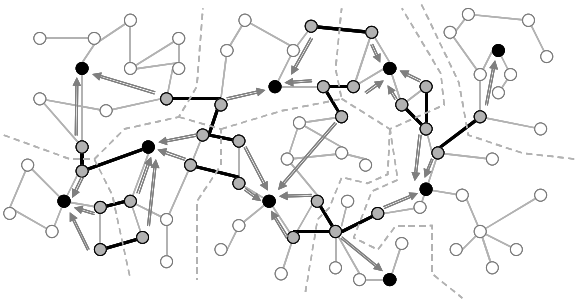

Figure 9. The Acquaint messages from boundary nodes to δ-nodes

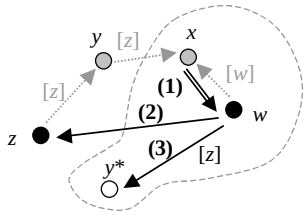

Figure 10. The Acquaint, Hello, and Advise messages in creating an I-link

fabric but avoid loops. In fact, u maintains variable $S(k, \delta)$ to record the set of nodes s which u is aware of with $g(s, k) = \delta$. In addition, u constantly updates variable $F(k, \delta) = v \in S(k, \delta)$ such that $d(u, v)$ is minimal, and if $v \neq u$, u sends a Vote message to v so that v adds u into v's variable $V(k)$ that records all those nodes selecting v as the nearest δ-node. Note that $g(v, k) = \delta$. Moreover, the two entries, v in u's $F(k, \delta)$ and u in v's $V(k)$, in pair represents an F-link from u to v on level k for digit δ. In the mean time, u utilizes its I-links to send all its neighbors an Advertise message so that the receiver will know v with $g(v, k) = \delta$. On the other hand, as u receives another Advertise message for a node $v' \neq F(k, \delta)$ with $g(v', k) = \delta$, u checks if $d(u, v') < d(u, F(k, \delta))$. If so, u sends a Veto message to the node $v = F(k, \delta)$ so that v will remove u from $V(k)$; then, u updates $F(k, \delta) = v'$ and sends a Vote message to v' so that v' will add u into its $V(k)$. Otherwise, since $d(u, v') \geq d(u, F(k, \delta))$, u alternatively sends an Acquaint message to v, which will soon be described.

Then, referring to Figure 8, every node including any δ-node eventually selects its nearest δ-node (a δ-node always selects itself), and creates an F-link to that δ-node (drawn in an arrowed thick line). Note that to keep Figure 8 from becoming too cluttered, only those F-links pointing to the δ-node on the upper left corner are shown, while the F-links to other δ-nodes are not, but Figure 8 does show certain boundaries which form a partition of the nodes: In each region defined by these boundaries (drawn in dashed lines), the F-links of all nodes point to the same δ-node.

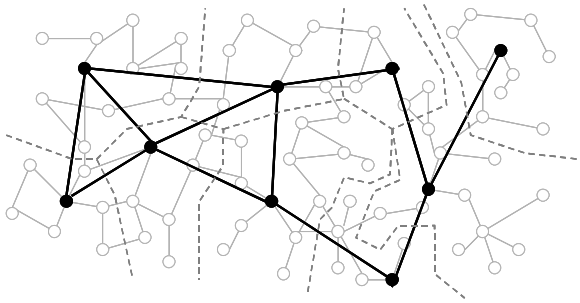

Figure 11. I-links created at level $k + 1$ by messaging in a level-k fabric

Referring to Figure 9, consider those I-links (in thick lines) that are crossed by boundary lines. The nodes (always in pairs) connected by such I-links are called the *boundary nodes* that play the key role in creating new I-links at the next level. Referring to Figure 10, we take a closer look at two boundary nodes x and y in the fabric of a level-k prefix α, where w and z are the δ-nodes for a digit δ. Clearly, a boundary node x receives two (at least) Advertise messages for w and z, respectively; x can infer that both w and z are near x in this α-fabric, and thus, w and z are near each other in the $\alpha\delta$-fabric where they are both contained. Suppose x sets $F(k, \delta) = w$, x sends an Acquaint message (marked by 1) to w so that w will create a next-level I-link with z in the $\alpha\delta$-fabric if $z \notin I(k + 1)$ at w, i.e. add z into $I(k + 1)$ and send a Hello message to z (marked by 2) so that z will also add w into z's $I(k + 1)$. The newly created I-link will then be exploited by w and z to exchange Advertise messages for NID digits such as $g(w, k + 1)$ and $g(z, k + 1)$ for F-links in the $\alpha\delta$-fabric. Therefore, referring to Figure 11, each pair of acquainted δ-nodes will create an I-link at the next level $k + 1$. Note that duplicate I-links (with the same node addresses on both ends) will not be created.

Each node x periodically updates $d(x, y)$ iff $y \in S(k, \delta)$ for any k and any δ at x. Updating $d(x, y)$ could possibly update x's $F(k, \delta)$, and generate Advertise messages to alter some F-links and I-links elsewhere. If a Detach command removes a level-1 I-link between x and y, both x and y delete the entries for each other from $S(k, \delta)$ for each k and each δ, and set $d(x, y) = \infty$.

Since the heuristic algorithm is based on the admin links that are created by AP's, if the admin I-links of a node x connect x to distant nodes but no node nearby, other links might also be created improperly. Thus, an optional remedial action can be taken to optimize the link topology by sending an Advise message (marked by 3 in Figure 10) from w to each node $y^* \in V(k)$, with a probability, to check if $d(y^*, z) < d(y^*, w)$.

2.4. Soft States and Fault Tolerance

A node constantly probes other nodes via I-links and F-links, to detect node failures. All entries in variable $I(k)$, $S(k, \delta)$, $F(k, \delta)$, and $V(k)$, as well as the object pointers, are maintained in soft states. An entry expires if it has not been renewed by a timeout. If an entry $x \in S(k, \delta)$ expires where $x = F(k, \delta)$ for some k and δ, then $F(k, \delta)$ must be updated by the new nearest node in $S(k, \delta) \setminus \{x\}$, which may change other I-links and F-links. The timeout should be selected carefully to achieve high performance in applications. The ideal soft-state timeout, regarding the overhead of renewal and the timeliness of failure detection, may be twice as long as the renewal period.

Using soft states in the routing tables improves fault tolerance, and significantly reduces the overhead when a GIN overlay is to be partitioned if two organizations terminate their affiliating-relationship, e.g. for contract expiration. Then, an object pointer to an object at a no-longer-affiliated organization can easily be removed by timeout since it will not be renewed any more.

3. Analysis Results

Theorem 1 (*Resilience*) Given that the admin links are created such that the level-1 fabric is a connected graph, all fabrics become connected graphs.□

Theorem 2 (*Fabrics*) With n nodes, there are up to $2n - 1$ fabrics maintained in GIN.□

Theorem 3 (*Memory Usage*) Assume that there are n nodes with $O(n)$ objects and each node has $O(1)$ admin links. Then, the average routing table size per node is $O(\log n)$ with high probability.□

Theorem 4 (*Merging Complexity*) A merging of two GINs with n_1 and n_2 nodes, and $O(n_1)$ and $O(n_2)$ objects, respectively, requires $O(n_1)$ total messages on average with high probability, as $n_1 \le n_2$ and $n_2 = O(n_1)$. However, if instead $n_1 << n_2$, a merging requires $O(n_1 \log n_2)$ messages.□

Theorem 5 (*Routing Efficiency*) The forwarding path of a query has a $O(1)$ latency stretch with high probability.□

Consider that a node y issues a query for an OID ϕ, and the host of an object X with $id(X) = \phi$ replies to y, with a latency L. Then, the *stretch* of the access path from y to X is defined as

$$\frac{L}{2 \times d(y, host(X))},$$

where L is the roundtrip time, and the processing time for $host(X)$ to access X is ignored. Thus, the stretch is a ratio that indicates the efficiency in forwarding a query—a smaller stretch indicates higher efficiency.

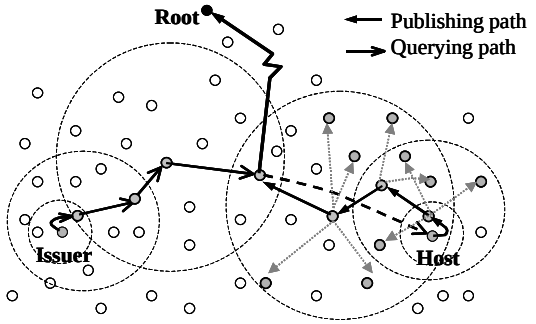

Figure 12. Overlapping of the querying and publishing paths in GIN

Referring to Figure 12, the publishing path and the querying path in GIN would overlap with high probability at a node at a network distance that is comparable to the distance between the query issuer and the host of the queried object. Since the overlapping node has a copy of the object pointer, the query is immediately sent to the host address so that the overall latency in forwarding this query is also comparable to the network distance between the query issuer and the host, that is, it has a $O(1)$ stretch, with high probability. This routing scheme is similar to the PRR scheme of which the achieved $O(1)$ stretch has been formally proven in [19].

4. Simulation Results

Refer to Figure 13 for three underlying networks in our simulations where the participating nodes are initially clustered in the local area networks (LAN's), and extra admin links will be created between LAN's. The number annotating an arc between two LAN's is the network distance (i.e. the propagation delay of any message) between those two LAN's. The network distance between any two nodes in different LAN's is

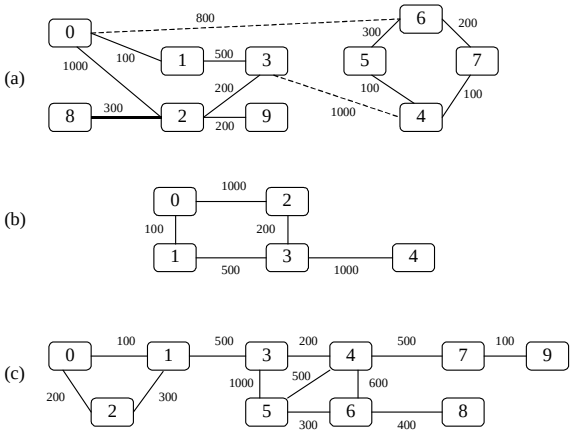

Figure 13. Underlying network topologies

4 ms plus the shortest path delay between their LAN's. In addition, the network distance between any two nodes in the same LAN is 2 ms. We assume that each node spends 1 ms at processing each GIN message, or accessing a locally registered object (as a host handling an incoming query for the object). In all simulations, b = 1 or 4, l = 32, and the Advise probability = 1/3.

4.1. Merging Complexity

We design two separate simulations on the same underlying network topology in Figure 13(a) but with different chronological orders in creating the admin links such that the merging conditions are different. In the first simulation, two GINs in similar sizes merge into one. In the second simulation, a smaller GIN (with fewer nodes) merges with a larger GIN (with more nodes). Note that the total numbers of all nodes (in the two merging GINs) in both simulations will be equal. We set b = 1 in both simulations.

Here we describe the first simulation. Initially at 0 sec, the simulation generates 300 nodes each of which is placed in a randomly selected LAN (i.e. the local cluster). Then, each node randomly chooses another node in the same LAN to create an admin link with that node. Then, at 1 sec, for each pair of LAN's connected in a solid line, we randomly choose one node from each LAN, and create an admin link between those two nodes. As such, two GINs are initially created, while all generated messages are recorded. In the simulation, no messages are generated after 20 seconds—i.e. the routing table contents at all nodes have converged and become stable by 20 sec.

Then, we create two extra admin links: One is to make the two GIN overlays merge, while the other is to later create a loop in the merged GIN overlay. That is, at 50 second, we first create an admin link between two nodes that are randomly selected from LAN 0 and 6, respectively. Then, at 100 second, we create the second admin link between two nodes that are randomly selected from LAN 3 and 4, respectively.

Figure 14 shows the simulation result in a three-dimensional diagram where the x-axis is the time, the y-axis is the level with which a message is associated, and the z-axis is the number of generated messages in each 1-second interval on a level. We categorize the generated messages by different levels to show the recursive behavior of our algorithm. Clearly, creating an admin link (i.e. an I-link on level 1) results in a sequence of messages that climb up the levels—as the F-links are updated at a level, some I-links are created at the higher level and trigger more updates of the F-links there.

Now, we describe the second simulation where two GINs in different sizes merge. On the same underlying

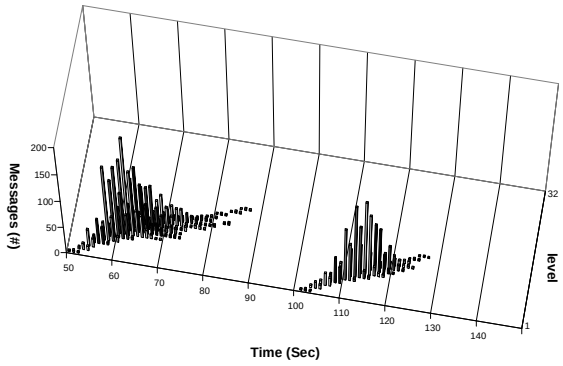

Figure 14. The simulation result of merging two GINs and creating a loop

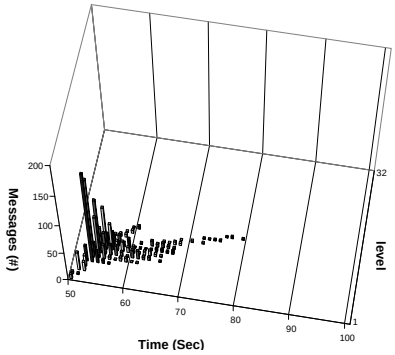

Figure 15. The simulation result of merging two GINs in different sizes

network topology and with the same number of nodes as in the previous simulation, the admin links in this simulation are created in a different chronological order. A larger GIN is initialized with the nodes in all LAN's excluding LAN 8, while a smaller GIN is initialized with only those nodes in LAN 8. In addition, the admin links between LAN's are created at 1 sec except that the admin link between LAN 2 and 8 will be created at 50 sec. It is observed that the routing table contents converge by 35 sec (compared to 20 sec in the previous simulation) since the larger GIN has a longer network diameter; then, the admin link created last at 50 sec makes the two different-sized GINs merge. Note that we do not create a loop in this simulation.

Figure 15 shows the second simulation result in a 3-dimensional diagram similar to Figure 14. Compared to the previous simulation, the total number of messages generated for the created admin link at 50 sec is significantly lower. We conclude that the number of messages generated for merging two GINs depends on the number of nodes in the smaller GIN. However, the convergence of the merging still takes approximately 38 seconds in Figure 14 and 30 seconds in Figure 15. Thus, the convergence time depends on the underlying

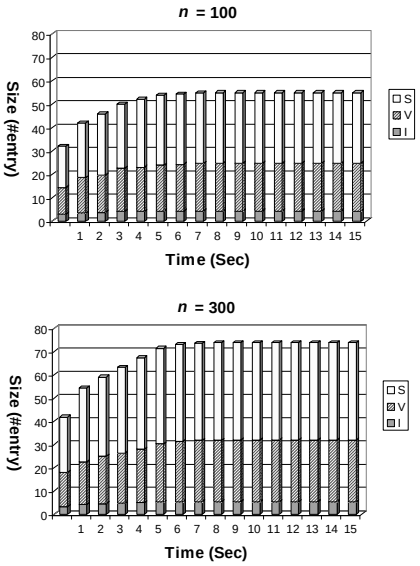

Figure 16. Simulation results of the runtime convergence of average routing table size

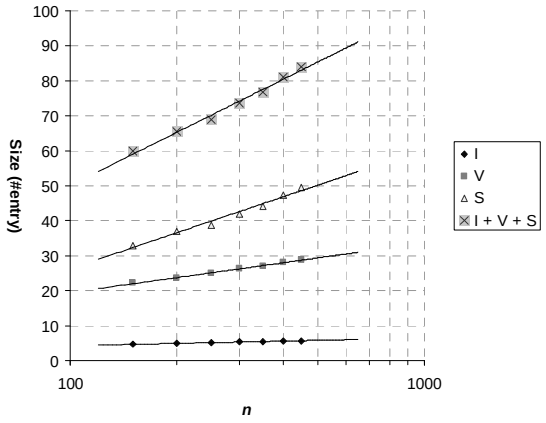

Figure 17. Simulation results of the average routing table size for variable *n*

network diameter (similar in these two simulations) for message propagation delays.

In addition, we categorize the messages by type, and observe that the majority of the generated messages are Advertise, approximately 60%.

4.2. Routing Table Size

Here, we show the simulation results of the average routing table size that is dynamically tracked at run time, with $b = 4$ and $l = 32$, for two cases $n = 100$ and $n = 300$, where n is the total number of nodes. We use the underlying network topology in Figure 13(a) for $n = 300$, and Figure 13(b) for $n = 100$. In addition to these two simulations, we design a set of simulations

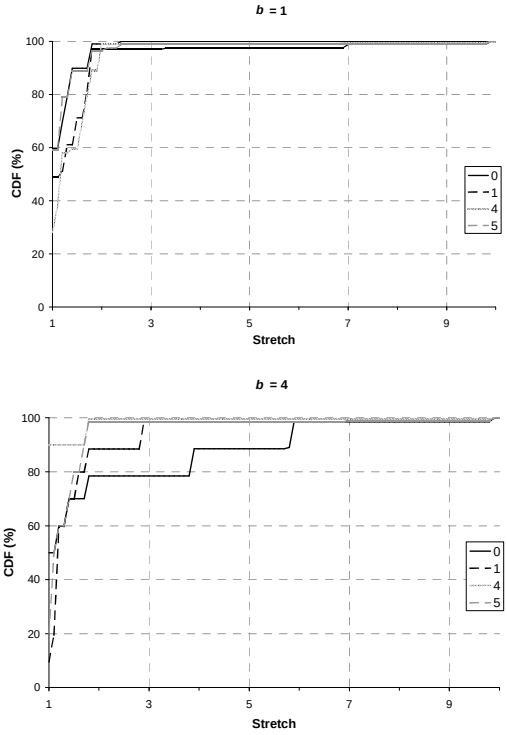

Figure 18. Simulation results of query stretches with *b* = 1 and *b* = 4

where n is a variable chosen in a range from 150 to 450, with $b = 4$, using the underlying network topology shown in Figure 13(c). In those simulations, we initialize the nodes and the admin links at 0 sec, and then we record the cardinalities (or sizes) of all $I(k)$, $V(k)$, and $S(k, \delta)$ variables at run time.

The results of the first 2 simulations are shown in Figure 16 and the last in Figure 17. Note that the *I*-series is for the average number of entries in all $I(k)$'s variables per node, recorded at the end of each 1-second interval. Similarly, the *V*-series is for all $V(k)$'s per node, and the *S*-series is for all $S(k, \delta)$'s per node. In Figure 16, since no more messages are generated after 8 sec in both cases, the routing table contents converge and the sizes become stable. It can be seen that as n increases 200% (from 100 to 300), the average number of entries in all these variables increases only 36% (approximately from 53 to 72). It is easier to see in Figure 17 (the x-axis on logarithmic scale) that the average routing table size increases only logarithmically with n.

4.3. Stretch

Then, we use the underlying network topology in Figure 13(c) and design two simulations (with $b = 1$ and $b = 4$, respectively), both with $n = 300$, to observe

the stretch of the routing latency in processing queries for objects. Each simulation has all nodes and admin links initially generated at 0 sec. Then, four objects are registered, each with unique OID's, and the host of each object is randomly chosen from the nodes in LAN 0, 1, 4, and 5, respectively. For each object X, 200 nodes issue a query for $id(X)$, where 20 nodes per LAN are randomly selected to do so.

The simulation results are shown in Figure 18. The cumulative distribution functions (CDFs) of the latency stretches for those four objects are plotted separately. Approximately 90% stretches are less than 2.0 as $b = 1$, and 80% as $b = 4$. We conclude that the access latency to an object has a $O(1)$ stretch with high probability.

5. Conclusion

We have proposed GIN for collaborative nodes to share objects across affiliated organizations, allowing their local namespaces to merge upon affiliating. It can forward a query with an $O(1)$ latency stretch with high probability and achieve high performance. Its routing tables are maintained in soft-states for fault tolerance, with a scalable and efficient algorithm, and adapting to performance updates of network distances. Thus, it has significant new advantages for building a DHT-based namespace for applications that will collaborate across organizations with dynamic affiliating-relationship.

References

[1] A. Arkin, S. Askary, et al., "Web Services Business Process Execution Language", *OASIS Specification Draft*, online at http://www.oasis-open.org/, Feb 2005.

[2] H. Balakrishnan, K. Lakshminarayanan, S. Ratnasamy, S. Shenker, I. Stoica, M. Walfish, "A layered naming architecture for the Internet", *Proc. of the ACM SIGCOMM '04 Conf.*, Aug 2004.

[3] T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", *IETF RFC 3986* (STD 1), Jan 2005.

[4] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, D. Orchard, "Web Services Architecture", *W3C Web Services Architecture Working Group Note*, documents online at http://www.w3.org/TR/ws-arch/, Feb 2004.

[5] D. Burdett, N. Kavantzas, "WS Choreography model overview", *W3C Working Draft*, March 2004. Online at http://www.w3.org/TR/ws-chor-model/.

[6] Gnutella.com, online at http://www.gnutella.com/.

[7] KaZaA.com, online at http://www.kazaa.com/.

[8] K. Hildrum, J. D. Kubiatowicz, S. Rao, B. Y. Zhao, "Distributed object location in a dynamic network", *Proc. of the ACM Symposium of Parallel Algorithms and Architectures (SPAA) '02*, Aug 2002.

[9] B. Karp, S. Ratnasamy, S. Rhea, S. Shenker, "Spurring adoption of DHTs with OpenHash, a public DHT service", *Proc. of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS '04)*, Feb 2004.

[10] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balakrishnan, "Chord: a scalable peer-to-peer lookup service for Internet applications", *Proc. of the ACM SIGCOMM '01 Conf.*, Aug 2001.

[11] M. Mealling, "Dynamic Delegation Discovery System (DDDS) - Part 4: The URI resolution application", *IETF RFC 3404* (STD 1), Sep 2000.

[12] Microsoft, Inc., *Distributed Component Object Model* (DCOM), documents online at http://msdn.microsoft.com/.

[13] R. Moats, "URN syntax", *IETF RFC 2141* (STD 1), May 1997.

[14] Napster.com, online at http://www.napster.com/.

[15] Object Management Group, Inc., "Common Object Request Broker Architecture (CORBA)", documents online at http://www.omg.org/.

[16] J. Pang, J. Hendricks, A. Akella, R. de Prisoco, B. Maggs, S. Seshan, "Availability, usage, and deployment characteristics of the Domain Name System", *Proc. of the ACM Internet Measurement Conf. (IMC) '04*, Oct 2004.

[17] V. Pappas, Z. Xu, S. Lu, D. Massey, A. Terzis, L. Zhang, "Impact of configuration errors on DNS robustness", *Proc. of the ACM SIGCOMM '04 Conf.*, Aug 2004.

[18] K. Park, V. Pai, L. Peterson, "CoDNS: Improving DNS performance and reliability via cooperative lookups", *Proc. of the 6th USENIX Symposium on Operating Systems Design & Implementation (OSDI '04)*, Dec 2004.

[19] G. Plaxton, R. Rajaraman, A. W. Richa, "Accessing nearby copies of replicated objects in a distributed environment", *Proc. of the ACM Symposium of Parallel Algorithms and Architectures (SPAA) '97*, Jun 1997.

[20] V. Ramasubramanian, E. G. Sirer, "Beehive: Exploiting power law query distributions for O(1) lookup performance in peer to peer overlays", *Proc. of the 1st USENIX Symposium on Networked Systems Design and Implementation (NSDI '04)*, Mar 2004.

[21] V. Ramasubramanian, E. G. Sirer, "The design and implementation of a next generation name service for the Internet", *Proc. of the ACM SIGCOMM '04 Conf.*, Sep 2004.

[22] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, "A scalable content-addressable network", *Proc. of the ACM SIGCOMM '01 Conf.*, Aug 2001.

[23] A. Rowstron, P. Druschel, "Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems", *Proc. of the IFIP/ACM Int'l Conf. On Distributed Systems Platforms*, Heidelberg, pp. 329-350, Nov 12-16, 2001.

[24] StreamCast Networks, Inc., online at http://www.streamcastnetworks.com/.

[25] V. Ungureanu, "Using certified policies to regulate E-commerce Transactions", *ACM Trans. on Internet Technology, Vol. 5, No. 1*, pp 129-153, Feb 2005.