A Fault-Tolerant Home-Based Naming Service For Mobile Agents

Camron Tolman and Carlos A. Varela
Rensselaer Polytechnic Institute
Troy, NY 12180, USA
{*tolmac,cvarela*}*@cs.rpi.edu*

Abstract

A *naming service* is in charge of locating a mobile agent in a distributed system given its name. Three critical characteristics of a naming service are: *fault tolerance*, *scalability*, and *efficient name resolution*. Most naming services provide support for efficient name resolution but do not address fault tolerance or scalability issues. Conversely, distributed hash-table approaches to naming provide fault tolerance and scalability albeit at a sacrificed name resolution performance. We introduce a Fault-Tolerant Home-Based Naming Service (FHNS) that is robust and scalable yet enabling efficient name resolution.

Keywords: Mobile agents, fault-tolerance, naming services, universal actors

1 INTRODUCTION

A *naming service* maps a name for an entity to a set of labelled properties. The Domain Name System (DNS) [11], for instance, is a network naming service that maps *easy-to-remember* names to *hard-to-remember* network addresses. Similarly, name services can be put to use in distributed computing architectures to map the identity of a mobile software agent to its current location on the network. The Fault-Tolerant Home-Based Naming Service (FHNS) we introduce, is one such naming service.

The motivation to develop a new naming service comes from considering the under-utilization of the Internet, which is predominately used for information retrieval, correspondence, commerce, and file sharing. The Internet, which can be thought of as a super-computer consisting of over a million processors, offers a vast amount of computing capability that has yet to be fully exercised. Substantial computations, such as data mining, financial forecasting, and complex simulations, can be divided and distributed across the Internet to be serviced by numerous machines concurrently. In order for these computations to be distributed, a framework first needs to be structured and deployed. The framework is required to be efficient, robust, scalable, and secure. The purpose of this research, however, is not to present a distributed computing framework in its entirety but to present a naming service that enables such a framework to meet a measure of the identified requirements.

The requirements for a naming service intended for a network like the Internet are equivalent to those placed on a distributed computing framework. The naming service must be:

- **Efficient:** The messages associated with resolving the location of an agent should be many times less than the number of nodes on the network. Not every node can be queried when an agent is being located.
- **Robust:** Any one failure cannot result in the entire service failing. The naming service must be resilient to node and link failures.
- **Scalable:** The service should allow nodes and agents to be located anywhere in the world and be able to support a huge number of these entities.
- **Secure:** The service must ensure data integrity and thus protect the names it is entrusted with from unauthorized modification or deletion.

And the names for the mobile agents must be:

- **Unique:** This is a requirement common to all naming services. In this case, the requirement for uniqueness is even more extensive in that the names for the mobile agents are to be globally unique.
- **Persistent:** The names must not change unless the naming service is notified.
- **Human Readable:** This is a requirement we place upon ourselves with the notion that human readable names are easier to remember and use. A human readable name can also indicate the purpose the agent serves, as would be the case of naming a personal-calendar agent: *JoeDoesCalendar*. Furthermore, people have the possibility to guess the agent name.

Mobile agents are best described as software processes that can migrate from one host to another while executing their assigned tasks. In between migrations, a mobile agent may need to interact with other agents to complete its jobs. Say for instance, a protein folding application decomposes the computations associated with protein folding into sub-computations to be processed by mobile software agents. These software agents are dispersed throughout the Internet to perform their distributed tasks. After performing a number of its computations, software agent GYPSY needs software agent NOMAD's results in order to continue with its computations. How does GYPSY locate the whereabouts of NOMAD given that it could be at any node on the Internet? There are different possible strategies to address the problem of locating an agent for the purpose of communicating directly with that agent [14, 17, 5]: for example, broadcast, search, tracking, centralized dedicated name services, and distributed dedicated name services.

Structure of the Paper The remainder of the paper is organized as follows. Section 2 presents the model of the Fault-Tolerant Home-Based Naming Service. Section 3 presents an FHNS implementation that was tested. The results of these tests are presented and evaluated in Section 4. And lastly, Section 5 presents some concluding remarks and contributions of this research.

2 FAULT-TOLERANT HOME-BASED NAMING SERVICE(FHNS)

2.1 Model

The Fault-Tolerant Home-Based Naming Service conforms to the home base naming service construct and as such each mobile agent has a home base that keeps track of that agents location. The mobile agents can either specify their home base directly or can opt to have a home base assigned. The naming service recognizes and is capable of resolving two types of names: location-dependent names or location-independent names. Due to this versatility, the service can effectively shift from a location-dependent service to a location-independent service in the event of a node failure (i.e. home base failure).

FHNS is succinctly comprised of the following:

1. **Unique Names:** Mobile agents have unique names that are premised upon Uniform Resource Identifiers (URI) [3]. The URI can take the form of a URL [2], which is location-dependent and is how the agent specifies its home base, or the URI can take the form of a URN [10], which is location-independent and requires a home base to be assigned.
2. **Home Bases:** Nodes on the network that provide the naming service for the mobile agents.
3. **Mapping Records:** A record kept for a mobile agent that is kept current with the agent's location on the network. The record is stored at the agent's respective home base.

Table 1 details the API for the Fault-Tolerant Home-Based Naming Service, which are the specific home base services.

2.2 Names

The location-dependent naming is attractive for the obvious reason that the location of the home base is given explicitly. Consequently, the agents themselves can be found quickly and efficiently simply by querying the respective home base. The location-dependent names of an agent take the form of a URL where the general syntax is:

<location-dependent name> ::= <scheme>:<scheme-specific-part>.

Function	Description
`put`(name, location)	Binds a name to an initial location.
`get`(name)	Returns the location for the given name.
`update`(name, location)	Updates the location of the agent with the specified name.
`delete`(name)	Remove the location record for the agent with the specified name.

Table 1: Fundamental API for FHNS

A URL contains the name of the *scheme* being used, which is an existing or experimental protocol. The scheme is followed by a colon and then a string (the <scheme-specific-part>) whose interpretation depends on the scheme. For Internet schemes the scheme specific part is written as follows:

//<user>:<password>@<host>:<port>/<url path>.

The optional elements are "<user>:<password>@", ":<password>" and ":<port>". The scheme specific data begins with a double slash "//" to indicate that it complies with the common Internet scheme syntax. The *host* field corresponds to the domain name of the node, or home base for the purposes of FHNS. The *url path* is not an optional element for FHNS and corresponds to the relative name of the agent. The names for an agent are referred to as either relative or absolute. The absolute name refers to the name in its entirety. The relative portion of the name refers to the name given to the agent that distinguishes it from other agents using the same host as a home base.

Location-independent naming is appealing because it allows the home base to change for a given agent. With location-independent naming, the names of an agent can take the form of a URN. The URN syntax is:

"urn:" <NID> ":" <NSS>.

The *NID* is the *Namespace Identifier*, and *NSS* is the *Namespace Specific String*. A ":" delineates the three elements. The "urn" descriptor string is specified to be a required element; however, for the purposes of FHNS this string will be replaced by another string corresponding to an existing or experimental protocol[1]. This modification allows for the two types of names to share a common element. The FHNS modified URN will be written as

<location-independent name> ::= <scheme> ":" <NID> ":" <NSS>.

A URN differs from a URL in that its primary purpose is persistent labelling of a resource with an identifier as opposed to serving as a locator. The location of agents are then resolved by using *consistent hashing* [7] and a lookup service such as the Chord [16, 4] peer-to-peer lookup service. By using a peer-to-peer lookup service, the home bases are required to keep a listing of other home bases so that when a *lookup* is required a home base will be able to query other home bases for the information it needs, in this case the location of an agent. A lookup is the process of a home base querying other home bases and could be performed when servicing any of the naming service requests. A lookup is not synonymous with the `get` service request.

2.3 Architecture

FHNS can be described as a layered architecture with DNS serving as an underlying naming service. DNS is utilized by FHNS when resolving the domain name of a home base to an IP address.

Names in the Domain Name System are a part of a naming hierarchy, with the most significant part on the right. The left-most portion of the name corresponds to the name given to the computer. Clients query local DNS servers for IP addresses. If a mapping has not been locally cached, the server starts with the root name server and recursively queries DNS servers until it finds a server that has the answer. In the case of a client request regarding *www.yahoo.com*, the local server queries a root DNS server then a *com* DNS server, and then finally the *yahoo.com* server for the web server IP address.

Fault tolerance is defined as the ability of a system to respond gracefully to an unexpected hardware or software failure. Fault tolerance for DNS is achieved with the use of a designated primary server and a

[1]The names adhere to the rules specified in [3, 2, 10] with this exception being made to the names following the URN format.

secondary or "back-up" server (or servers). If the primary server fails, requests are directed to the secondary server. With this approach, however, the notion of *failover* is always provided for the primary but not the secondary. If the secondary server fails or the last "backup" server fails, there exists a fault in the system. FHNS achieves fault tolerance with redundancy between home bases and provides for an iterative failover capability. Each home base functions as both a primary and secondary server. And each home base has a designated successor that is capable of servicing requests on behalf of that home base in the event of a failure. Essentially, FHNS is capable of recovering from all home bases, except any arbitrary one, failing.

Returning our attention back to DNS, when an organization wants to participate in the Domain Name System, the organization must apply for a name under one of the top-level domains (i.e. com, edu, gov, co, fr, etc.). If the name is already in use, the organization will not be allowed to use the name and thereby uniqueness is ensured among all the top-level domain names. If the name is not already in use and the organization is allowed to use the name. They then assume the responsibility of ensuring that names are unique with respect to their domain. Names are kept unique in FHNS in a similar manner. In the case of location-dependent naming the first part of the name, that being the host, must be unique since it is a name in the Domain Name System. The name of the agent must then be unique with respect to the home base (i.e. host). The home bases reject a put request if the name requesting to be used is already in use—thereby ensuring uniqueness. Ensuring unique naming with the location-independent names requires that the combination of namespace identifier and namespace specific string be unique. Again the home bases can reject a put request if the namespace specific string is not unique relative to the namespace identifier. A single global namespace identifier will be used, which is thus trivially unique.

The namespace identifier corresponds to a logical ring with distinct points along the ring called *identifiers*. A single home base is assigned to one of the identifiers by way of consistent hashing. Consistent hashing is a distribution scheme that pertains to using a hash function to distribute strings to a number (i.e. identifier) in the range $[0,\ldots, M]$, where M is determined by the selected hashing function. Home Bases are assigned to an identifier by hashing the domain name or the IP address. The URN of an agent is also hashed and mapped to an identifier. The agent is then assigned a home base by identifying the home base that has been mapped to a number greater than or equal to the number the agent has been mapped to (0 is the first number greater than M, when making assignments). All home bases belong to this single logical ring and each home base keeps a routing table that lists other identifiers and the home bases associated with those identifiers. The routing table contains $log\ M$ entries[2]. Each home base has its mapping records replicated at a number of home bases with identifiers greater than and nearest to the identifier of the respective home base. The logical ring is not static, new home bases can join and the mapping records will be transferred accordingly. Conversely, home bases can leave the logical ring and one of the replicating home bases will assume responsibility for the departed home base's mapping records. However, the hosts that will serve as home bases should be chosen judiciously since it would not be desirable to have a home base that is only powered on for short durations, as would be the case with mobile devices. Also, home bases are not restricted to being dedicated name servers only. Home bases can be like other hosts in providing a platform for mobile agents to execute their assigned tasks.

2.4 Protocol

As an example of how the naming service functions, we now present a description of agents using the service. We begin with a 3-bit namespace, called $drifters$, and we will use *SHA-1 % 8* [21] as our hashing function. We identify 3 hosts to be our home bases and assign them each an identifier.

NAME	IDENTIFIER
rome:3030	7
hongkong:4040	0
newyork:5050	5

Table 2: Example: Home Bases and the Assigned Identifiers

In practice we would use a hashing function bigger than 3-bits to avoid collisions and identifiers would not be assigned but determined by hashing the domain name. Next we introduce 4 mobile agents, half

[2]Unless stated otherwise, all logarithms in this paper are of base 2.

will use location-dependent names and the other half of agents will use location-independent names. Our experimental naming service scheme is called $fhns$.

RELATIVE NAME	ABSOLUTE NAME	IDENTIFIER
MIGRANT	fhns://rome:3030/MIGRANT	6
TUMBLEWEED	fhns://hongkong:4040/TUMBLEWEED	0
GYPSY	fhns:drifters:GYPSY	4
NOMAD	fhns:drifters:NOMAD	2

Table 3: Example: The Agents' Names and the Determined Identifiers

The agents would need to be aware of at least one home base so that they could issue a request to that home base to perform a put(<agent absolute name>, <agent location>). Agents TUMBLEWEED and MIGRANT could issue the request to the host specified in their names, which would then become their corresponding home base. GYPSY and NOMAD could issue their requests to any of the home bases. The home base servicing the request first hashes the names of these agents, GYPSY and NOMAD, to determine the proper home bases and then forwards the request to those identified home bases. Home bases are able to forward requests by determining from their routing tables the host nearest the identifier they are seeking.

If GYPSY wanted to communicate with MIGRANT it could request that the *rome* host machine perform a `get`(*fhns://rome:3030/MIGRANT*) operation. The location of MIGRANT would be provided with a single message loop (i.e. request and reply), which is why we say that the location-dependent naming aspect of this naming service is very efficient in resolving actor locations. But say GYPSY wanted to communicate with NOMAD. GYPSY would direct a request to any of the home bases to perform a `get`(*fhns:drifters:NOMAD*) operation. The home base, say *hongkong*, would first hash the string "NOMAD", which is the agent's relative name, according to the *SHA % 8* hashing function associated with the *drifters* namespace. The relative name of NOMAD hashes to identifier 2. *Hongkong* would look into its routing table and see that *newyork* maps to identifier 5, which is the node greater-than and closest to identifier 2. The `get`(*fhns:drifters:NOMAD*) request would be handled by *newyork* and the results would be forwarded to *hongkong* and then provided to GYPSY. Here it took more than a single message loop to find the location of NOMAD. Location-independent naming does not give as good results in resolving the location of agents since it requires that the home base for an agent first be discovered. Finding the location of a home base could take $O(log\ n)$ messages when using the Chord lookup strategy, where n is the number of nodes in the ring (i.e. home bases). When a home base is requested to determine the location of an agent, it first checks the mapping records in its possession. If a mapping record is found, it returns the location of the agent. If a mapping record is not found, then the home base forwards the request to the home base that is closest to the identifier to which the name of the agent is mapped. The request is forwarded until it is serviced by the authorized home base. To minimize the number of hops between nodes, the searching home base can cache the location of an agent's home base so as to avoid having to perform another multi-message lookup for subsequent locate requests for the same agent.

Some time has passed and NOMAD decides to move from a host machine called *host1* to *host2*. It first requests a home base to perform an `update`(*fhns:drifters:NOMAD*, *host2*) operation and then NOMAD moves to the new host. The agent framework is required to provide a mechanism to handle messages in transit that were sent to the old location of the agent. One way of handling this is for the agent to leave behind a forwarding agent at the old location storing messages while the agent is migrating and redirecting them to the new location upon the agent's request. The forwarding agent could be deleted by expiration or from a request made by the originating agent.

When the agents are done performing their tasks, they are able to remove themselves from the system by issuing a `delete`(<agent absolute name>) request.

When a home base crashes and is removed from the network, the agents using that home base still need to have some way of being located. The agents are still on the network but the locating mechanism has been disrupted, and not addressing this contingency would render these agents *unreachable*. Names that are location-independent lend themselves nicely to fault tolerance since they are not bound to any home base explicitly. Using Chord, we start with a logical ring and place home bases at distinct points along the ring based upon their corresponding identifiers. Moving in a clockwise fashion, the subsequent home bases keep copies of the mapping records belonging to the previous home bases. So in the event that a home base fails,

the subsequent home base will be able to service all the requests that would have been handled by the now departed home base. Agents therefore do not become unreachable when their home base fails. The agents using location-dependent names can still be found because the host portion of their names corresponds to a home base and thus maps to an identifier. The subsequent identifier can easily be determined and the replicating home base can thus be found. This does introduce a incongruity in that the location-dependent name does not accurately give the location of the agent's home base, but there is no way to solve this problem since we wish names to be persistent even in the event of a failure. This incongruity can be avoided by using location-independent names.

3 IMPLEMENTATION

An instance of the Fault-Tolerant Home-Based Naming Service has been prototyped using Chord as the lookup service. The prototype was then analyzed using the World-Wide Computer (WWC) [20] as a testbed. What follows are brief descriptions of the WWC and Chord. The architecture of the prototype is then described.

3.1 Context: World-Wide Computer (WWC)

This framework recognizes the Internet's vast computing capability with its very name: the World-Wide Computer. The WWC enables tasks to be distributed among participating computers. The World-Wide Computer has been introduced to make it possible to pool the computing ability of devices that can communicate over a wide-area network such as the Internet. The WWC consists of a set of virtual machines, or *theaters*, hosting universal actors. The universal actors are reachable throughout the Internet by their globally unique Universal Actor Names (UAN). The UAN identifiers persist over the lifetime of an actor and are used to obtain the actor's current theater. The Universal Actor Locator (UAL) represents the location of this theater. Universal actors extend actors [1] in that they are able:

- to bind to a unique global name and to an initial theater,
- to obtain references to their peers from their names,
- to communicate through message passing in a location-independent manner, and
- to migrate from one theater to another.

In summation, the WWC is comprised of universal actors, theaters, and dedicated name servers.

An example of a UAN is

uan://wwc.publisher.com:3030/poemAgent.

The relative name of this actor is *poemAgent*, *wwc.publisher.com* is the name server or home base, and *uan* is the naming service protocol. The UAN has an identical format to the location-dependent names of FHNS. The naming server keeps track of where the agent is located by receiving information in the form of UALs. In this example, say the actor's UAL is

rmsp://wwc.editor.com:4040/poemAgent.

The naming server knows from this UAL that the poemAgent is located at the *wwc.editor.com* theater. The *rmsp* refers to a Remote Message Sending Protocol that enables sending messages to remote actors. The format for UANs has been augmented so as to support location-independent names for FHNS.

The default naming scheme for this framework uses a home-based strategy and location-dependent names. The name servers do not include any fault tolerance features. So when a name server fails, the agents that are bound to that name server become unreachable to collaborating agents holding only their names. FHNS is used to eliminate this single point of failure.

3.2 Context: Chord

Chord is a distributed lookup protocol that addresses the problem of efficiently locating data in peer-to-peer networks. The Chord protocol supports just one operation: given a key, it maps the key onto a node. The key can be associated with a particular data item and together both the key and data are stored at the node to which the key maps. The data is then located by a *lookup(key)* algorithm that yields the value associated with that key, which could be the IP address for a given node on the Internet.

The consistent hash function assigns each node and key (such as the name of an agent) in the system an m-bit identifier. The node identifiers can be established by hashing the IP address or domain name. Likewise the identifiers for the keys, which are strings, are established by hashing the keys themselves. As with any hash function, collisions can occur. Collisions between keys are unimportant since keys are allowed to map to the same identifier and it is the keys themselves that must be unique. Collisions between nodes must be avoided in order for Chord to work correctly. Chord does not prevent collisions.

Once the keys and nodes have been mapped to identifiers, the keys are then assigned to nodes. Any given key is stored at the first node whose identifier is equal to or greater to the key's identifier, where the last identifier in the identifier space is less-than the first identifier to effectively give a logical ring or an *identifier circle*.

Each node maintains a routing table with m entries, where m is the number of bits in the binary representation of the key or node identifiers. The routing table is called the node's *finger table* and the i^{th} entry in this table is given by $n+2^{i-1}$, where n is the identifier of the node and $1 \leq i \leq m$. The node storing the identifier given by the i^{th} entry is listed in the finger table as the node to forward requests to when the $n + 2^{i-1}$ identifier is sought. Also listed in a node's finger table is the node previous to it in the identifier circle and is referred to as the *predecessor*.

If a node gets a lookup request for an identifier not listed in its finger table, it forwards the request to the entry in its finger table that lists an identifier closest to, but less than or equal to, the requested identifier.

In a dynamic network, nodes can join and leave at anytime. When a node joins the ring it has to request that an existing node provide the location of its nearest neighbors in the identifier space. Once the joining node receives its finger table information, it then requests that the keys it is responsible for be transferred. When a node leaves the network gracefully, it transfers its keys and has its neighbors update their finger tables accordingly. Nodes leave and join the Chord identifier circle at a cost, with high probability, of $O(log^2\ n)$ messages.

Each node's nearest neighbor or neighbors keeps copies of the keys belonging to that node. So when a node fails, the nearest neighbor of that node can service the *lookup* operation directed to the failed node. Chord does not specify the number of neighbors that should provide redundancy but this number does correspond to the amount of fault tolerance attainable. If ten neighbors provide redundancy, then the system can conceptually recover without losing any of the keys in the event of ten simultaneous node failures.

The nodes perform stabilizing procedures in which they determine if their predecessor and immediate successor information is correct. By performing stabilization, the immediate neighbors of a failed node will see that it is not responding and will take the appropriate measures to recover from the failure by updating their finger tables. Chord does not specify how often to perform stabilization; however, the Chord test-case implementation performed stabilization at an average rate of 0.5 invocations per minute.

3.3 FHNS Prototype Developed for the WWC

The WWC already has a naming service in place that uses location-dependent names. This naming service is first extended to use location-independent names. A UAN can take the traditional form, specifically

uan://<home base host>:<port>/<actor relative name>.

A UAN can also take the location-independent form,

uan:wwc:<actor relative name>.

The namespace identifier, for this prototype, is called *wwc* and the corresponding hashing function to be used will be *SHA-1* [21], which hashes the given inputs to a 160-bit identifier. The namespace will encompass all identifiers $0 <= i < 2^{160}$.

A unique type of theater has been developed to serve as the home bases for FHNS and are called *home base theaters*. The naming service has been incorporated into these theaters instead of using dedicated

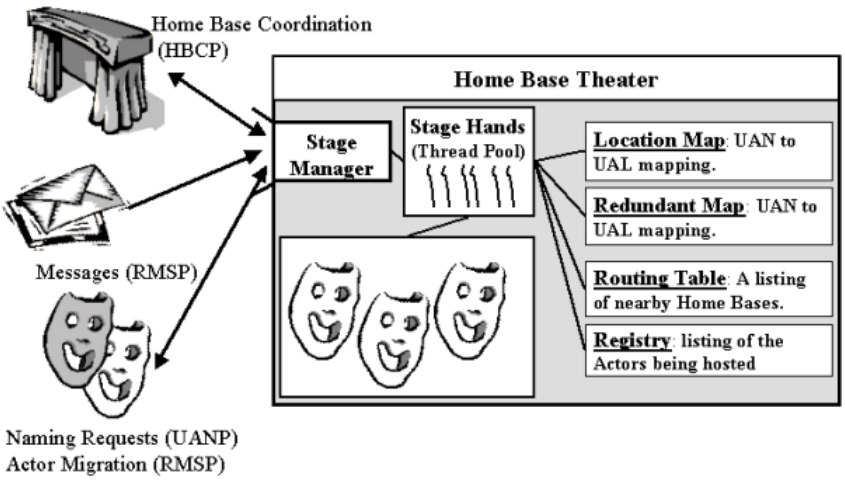

Figure 1: A Home Base Theater in the WWC.

naming servers. This is done to better ensure fault-tolerance. The alternative would be to have the actors keep a listing of home bases that they could query. A *Denial of Service* (DoS) could result if the home bases that an actor had a listing for had all failed. Having the actors direct all requests, with the exception of the `put` request, to the theater where they are presently "performing" ensures that the naming service is always available to the actors.

The test case consists of universal actors and home base theaters. These home base theaters host the universal actors and also serve as the name servers. Figure 1 depicts the structure of the home base theaters in the WWC. The figure also illustrates the other services provided by these theaters. In keeping with the "Broadway" manner of speaking, the home base theaters have a single *stage manager* that listens on a particular port for requests. Upon receiving a request, the stage manager delegates responsibility to servicing the request to a *stage hand* that is chosen from a thread pool. As illustrated in Figure 1 a home base theater can service a naming request using the Universal Actor Naming Protocol (UANP), a message directed to a hosted actor using the Remote Message Sending Protocol (RMSP), a hosting request also using RMSP, or a coordination request using the Home Base Coordination Protocol (HBCP).

The methods belonging to the Universal Actor Naming Protocol are identical to the methods in the API for the Fault-Tolerant Home-Based Naming Service. The Fault-Tolerant Home-Based Naming Service is thus capable of servicing all UANP requests. One modification to UANP that was needed was to change it from being a strictly text-based protocol to being able to use Java objects. This was done to facilitate communication in that all requests could come into the home base over the same port.

When a message is received via RMSP, the stage hand servicing the request obtains from the *registry* a reference to the universal actor to whom the message is intended and places the message in the actor's mailbox. A migrating actor comes into the home base theater through the same port as the other types of communications. The actor is registered with the home base and then is left to act upon the messages in its mailbox.

The Home Base Coordination Protocol provides the methods necessary for home bases to transfer needed information to participate in the namespace (logical ring). Table 3.3 details the different HBCP requests and corresponding services.

The system is initialized by a single home base theater being started. The domain name of the machine the home base theater is running on along with the port number is hashed according to the *wwc* namespace hashing algorithm and an identifier is assigned. Notice that the domain names are being hashed and not the IP addresses. This is not to say that the IP address cannot be used, they in fact can. However, for any given node either the domain name needs to be used in all cases or the IP address since the IP address and domain name will hash to different identifiers.

Other home base theaters can then join the system by contacting a home base theater that already belongs to the system and retrieving the necessary routing table information. A joining home base theater, or node

Request and Service	Description
NODEJOIN	Request made by a joining node where the servicing node returns with the requesting node's immediate successor
INITIALIZE	A joining node obtains its predecessor and routing entries from its successor
UPDATECOVERAGE	Joining or leaving node is inserted or removed and requests its successor and predecessor to update their Namespace coverage
GETKEYS	A joining node is provided the keys it is responsible for by its successor
REPLICATE	A node requests its successor to replicate the specified actor mapping
UPDATETABLE	A node has either joined or left (failed) and other nodes are informed to update their tables
GETNODE	Gets the node covering the region where the given Namespace identifier belongs
STABILIZE	A node periodically confirms the presence of its successor and predecessor

Table 4: Home Base Coordination Protocol

as it is referred in the proceeding explanations, first determines its identifier. It then issues a `nodejoin` request to an existing node. The servicing node returns with the name of the joining nodes immediate successor. The joining node then issues an `initialize` request to its immediate successor. The successor then provides the name of the joining node's predecessor and the nodes to list in its routing table and also the nodes that need to be informed of this node joining. The joining node then requests its successor provide the keys that it is responsible for with a `getkeys` request. Next the joining node issues `updatetable` requests to those nodes identified that need to be informed of the node's presence. Afterwards, the joining node tells its predecessor and immediate successor to update their coverage with an `updatecoverage` request. Upon completion of these requests, the joining node is now a part of the system and it begins its stabilization cycle. The stabilization cycle involves a node issuing a `stabilize` request to both its predecessor and successor. The predecessor returns with its predecessor and the successor returns with its successor. So in the event of a successor or predecessor failure, the node will already know of that particular node that will assume the role of the failed successor or predecessor. When the node, home base theater, receives a UANP `put`, `update`, or `delete` naming request, it sends a HBCP `replicate` request to its successor so as to keep the *location maps* and *redundant location maps* consistent.

Universal actors can enter the system at any point after the initial home base theater is up and listening for requests. The universal actors become reachable to other universal actors by issuing a `put`(*UAN*, *UAL*) request to one of the home base theaters.

4 PERFORMANCE ANALYSIS

The FHNS prototype has been put through different tests. The intent for these tests is to gather metrics that will support the requirements specified in Section 1 regarding FHNS being efficient, scalable, and robust. In the first test case, both location-independent and location-dependent names will be used. The results obtained from this first test case will pertain to resilience of FHNS in the event of a home base failure and is used to illustrate robustness. Messaging efficiency and scalability are evaluated in the next test case. The processing time to service requests for location-independent names versus location-dependent names is evaluated. Lastly, a scalability analysis pertaining to the load balancing capabilities is then detailed.

Before going further with the analysis, the reader should consider the specific implementation of the prototype. The tests that have been outlined are intended to measure the performance of the FHNS model; however, the tests predominately relate to the specific implementation of using Chord as the lookup service. Bearing the impact of Chord upon these results, the analysis of FHNS continues with the first test case regarding robustness.

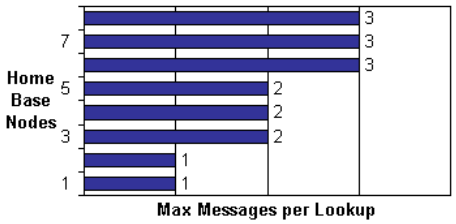

Figure 2: Single and Sequential Home Base Failures going from a total of 8 Home Bases to 1.

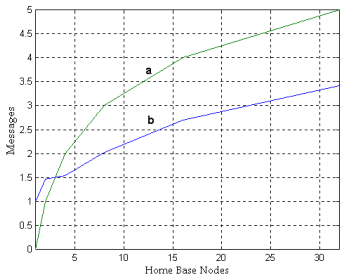

Figure 3: Message Count versus Number of Home Bases.

4.1 Robustness

The primary objective with using a lookup service such as Chord is to incorporate its fault tolerance features into FHNS. Since fault tolerance is the primary concern, this capability was tested to establish that the FHNS model is sound. We begin with 8 home bases and fail the home bases one by one, effectively degrading the distributed naming system to a centralized naming system. After each home base failure, `get` requests are made for all the actors in the system to confirm that they are all still reachable. The maximum number of messages it takes to have a request serviced is recorded. For this test, there are an equal number of actor location-dependent names as there are actors with location-independent names.

When the home base of an actor with a location-dependent name fails, FHNS hashes the domain name and port number for that failed home base and determines the immediate successor of the failed home base using a `getnode` request. Once the successor is determined, the request is then forwarded. It is known that the immediate successor of the failed home base has a redundant location map for the failed home base. The immediate successor is thus capable of servicing requests that were intended for its predecessor, which is the failed home base.

Figure 2 shows the number of messages it takes to find the back-up home base. The home bases refer to both their immediate location map and their redundant location map when servicing a request and this explains why only one message is needed when two home bases are present. The figure illustrates that FHNS can recover from all home bases failing except for one.

Performance regarding multiple home bases concurrently failing is not evaluated since it is known that this implementation has a single replication depth. Only a home base's immediate successor provides redundancy. If both the home base and its successor failed then there would be actors that would be rendered unreachable. It follows that if the replication depth was doubled, FHNS could recover from two consecutive and concurrent home bases failing. Increasing the replication depth, however, also has the disadvantage of increasing the number of HBCP `replication` requests.

The failed nodes are identified by the home bases periodically performing a stabilization routine. Stabilization involves a home base attempting to communicate with its predecessor and successor. If the home base is not able to communicate it takes the appropriate measures to replace the failed node and transfers and replicates the mapping records as needed.

4.2 Messaging Efficiency

Next we turn to the number of messages it takes for FHNS to service a request in a fault-free system. By messaging we count only the requests and forwarded requests and not the responses. When location-dependent names are used only a single message is needed. The messages needed for location-independent names depend on the lookup service employed. Stoica et al [16] state that with high probability, Chord resolves a lookup with $O(log\ n)$ messages. This assertion was confirmed by testing. Figure 3 shows the $log\ n$ trend with line a and line b is the average number of messages it took for FHNS, using Chord, to resolve a lookup pertaining to location-independent names. When only two home bases are present, it can take two messages to resolve a lookup; specifically the original request and the forwarded request. In the presence of n home bases, where $(n > 2)$, it takes on average less than $O(log\ n)$ messages.

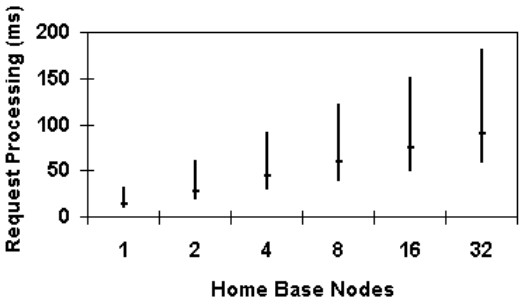

Figure 4: Prototype Request Processing Time when Using Location-Independent Names.

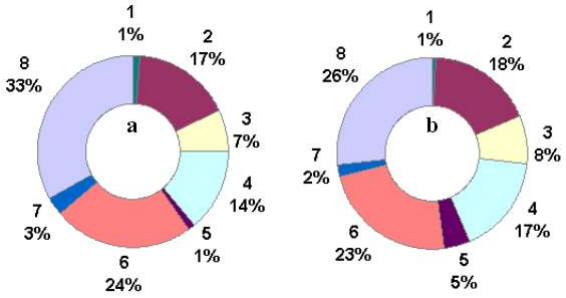

Figure 5: (a)The Mapping Records coverage and (b)Namespace coverage divided among 8 Home bases.

4.3 Time Efficiency

FHNS is indeed highly efficient when location-dependent names are used. Its efficiency is put into question with location-independent names. If we do not consider network communication, the *Request Processing Time* (RPT) when using location-independent names is over 3 times that of the processing for location-dependent names. FHNS takes on average 420μs to process a location-independent request and 130μs to process a location-dependent request. This time corresponds to the time elapsed from a home base receiving a request to the generation of a response.

If we consider communication over a local network, the total RPT is in the milliseconds range. It takes, on average, 14ms for FHNS to service a `put` request that requires no hops be taken. Approximately half this time is spent communicating with the requester and the other half communicating a `replicate` request. When hops are necessitated, the latency is increased by the number of hops. Figure 4 shows the minimum and maximum RPTs experienced by FHNS when servicing location-independent `put` requests. Due to these latencies, the location-dependent names are the preferred format for actor names, since they provide fault-tolerance without giving up on efficiency.

4.4 Load Balancing

To fully meet the efficiency and scalability requirements, the home bases can not be too overloaded, else there is the potential for a bottleneck. By using Chord as the lookup service, FHNS is able to attain load balancing. Ideally, load balancing entails that given K identifiers and H home bases, each home base is responsible for K/H identifiers[3]. In practice, however, the number of identifiers and mapping records per home base exhibits large variations. For instance, when the FHNS prototype consists of 2 home bases, one of the home bases covers 53% of the namespace identifiers and the other home base covers the remaining 47%. In contrast when two additional home bases are added giving a total of 4, 2 of these home bases each only cover 1% of the namespace identifiers, another covers 38%, and the last home base's coverage increases from 53% to 60%[4].

Figure 5 illustrates the namespace identifier coverage and the mapping record coverage. It can be seen from this figure that load balancing is not sufficiently achieved. It could be argued that load balancing is achieved by half the nodes but it certainly cannot be said for them all. Also depicted in the figure is the obvious correlation between namespace identifier coverage and mapping record coverage. The home bases with larger namespace coverage also store more of the mapping records.

The region for a node begins at the node's predecessor's identifier and spans all identifiers up to the node's identifier. Stocia et al [16] propose that each node (home base) divide the namespace into $(O(log\ n)\ +\ 1)$ regions of coverage. In order to do this, the nodes would each have $O(log\ n)$ virtual nodes. Applying this proposal has the effect of "balancing" the coverage amongst the nodes. Figure 6 depicts how using virtual nodes equalizes the coverage among the home bases. Although not necessarily ideal, a level of load balancing is thus achieved when using Chord as a lookup service and having each home base cover $(O(log\ n)\ +\ 1)$ regions. A node's total namespace coverage is the summation of the node's "non-virtual" coverage and its virtual coverage.

[3]Assumes all keys are equally "popular" and the nodes are homogeneous.
[4]The imbalance is due to two nodes producing very similar hashing values.

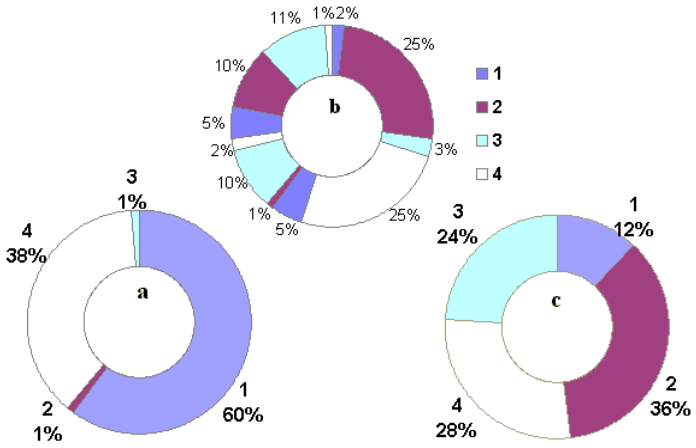

Figure 6: (a)Namespace coverage divided among 4 Home Bases, (b)Namespace Coverage divided among 4 Home Bases with 8 Virtual Nodes (c) Total Namespace Coverage with Virtual Nodes.

5 CONCLUSIONS

The Fault-Tolerant Home-Based Naming Service is a novel approach to locating mobile agents in a distributed computing framework. FHNS builds upon DNS and a peer-to-peer lookup service. And as a result of this meshing, this naming service is fault-tolerant as was shown by using a lookup service such as Chord. The Fault-Tolerant Home-Based Naming Service is also very efficient in resolving the location of a mobile agent.

The specific contributions made in this paper are as follows:

1. **A naming service specifically designed for use by mobile agents**. From the research that has been done, it was observed that efforts on the specific topic of naming services for mobile agents has not received much attention, at least when compared to mobile agent security, mobility, and messaging. Also from researching the naming services in current mobile agent platforms, it was observed that efforts do need to be focused on this very topic. Most Java-based agent systems use RMI or Corba which introduce single points of failure.

2. **A naming service that is highly efficient, fault tolerant, and scalable**. Location-dependent naming services, such as the one found in RMI for instance, are efficient in resolving a name to a location but introduce a single point of failure. FHNS is as efficient as these location-dependent naming services and it is also fault tolerant. Furthermore, the fault tolerance extension does not degrade the efficiency in resolving location-dependent names until a failure occurs.

3. **A naming service with more extensive fault tolerance than the conventional redundancy model**. Fault tolerance is a critical requirement of a naming service. Traditional redundancy schemes involve a designated primary server and an associated secondary server. The secondary server, however, does not necessarily have a back up. So when the primary fails, the system can no longer be described as being fault tolerant because the secondary, or the new primary, server can not fail. With FHNS, each server has a backup and each server is a backup for another server. The servers can fail down to a single remaining server and FHNS will still work as needed.

The FHNS implementation presented in this paper uses Chord, however other distributed hash table lookup services (e.g.CAN [13], Tapestry [22], Pastry [15], SPRR [9]) could be used to obtain different performance and reliability characteristics. [18] gives a detailed account of these alternatives.

There are numerous mobile agent systems currently in development (e.g, [6, 8, 12, 19]). Many of the systems are Java-based and have agents that communicate directly with other agents. Direct communication requires mobile agent naming and location mechanisms. The naming mechanisms currently employed by these mobile agent systems are not as efficient as they could be and/or they are not robust. FHNS could be incorporated by these mobile agent systems to realize an efficient and robust naming mechanism.

References

[1] G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems.* MIT Press, 1986.

[2] Tim Berners-Lee. Uniform Resource Locators (URL). Informational RFC 1738, 1994.

[3] Tim Berners-Lee. Uniform Resource Identifiers (URI): Generic Syntax. Informational RFC 2396, 1998.

[4] Frank Dabek, Emma Brunskill, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica, and Hari Balakrishnan. Building peer-to-peer systems with Chord, a distributed lookup service. In *Proc. 8 Wshop. Hot Topics in Operating Syst., (HOTOSVIII)*, pages 81–86, 2001.

[5] G. H. Forman and J. Zahorjan. The challenges of mobile computing. Technical Report TR-93-11-03, IEEE Computer, 1993.

[6] Mitsubishi Electric ITA. Concordia: An infrastructure for collaborating mobile agents.

[7] David Karger, Eric Lehman, Tom Leighton, Mathhew Levine, Daniel Lewin, and Rina Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *ACM Symposium on Theory of Computing*, pages 654–663, May 1997.

[8] D.B. Lange and M. Oshima. *Programming and Deploying Mobile Agents with Java Aglets*. Addison Wesley Longman INC, 1998.

[9] Xiaozhou Li and C. Greg Plaxton. On name resolution in peer-to-peer networks. In *Proceedings of the second ACM international workshop on Principles of mobile computing*, pages 82–89. ACM Press, 2002.

[10] R. Moats. URN Syntax. Informational RFC 2141, 1997.

[11] P. Mockapetris. Domain Names - Implementation and Specification. RFC 1035, 1987.

[12] Tomasz Muldner and Thian Tin Ter. Building Infrastructure for Mobile Software Agents. *World Conference on the WWW and Internet WEBNETC*, 2000:419–424, 2000.

[13] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 161–172. ACM Press, 2001.

[14] V. Roth. Scalable and secure global name services for mobile agents. In *6th ECOOP Workshop on Mobile Object Systems: Operating System Support, Security and Programming Languages*, 2000.

[15] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218:329+, 2001.

[16] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 149–160. ACM Press, 2001.

[17] Karim Taha and Thomi Pilioura. Agent Naming and Locating: Impact On Agent Design. In D. Tsichritzis, editor, *Trusted objects. - Geneve : Centre Universitaire d'Informatique*, pages 149–169. 1999.

[18] Camron Tolman. A fault-tolerant home-based naming service for mobile agents. Master's thesis, Rensselaer Polytechnic Institute, 2003.

[19] Maarten van Steen, Franz J. Hauck, and Andrew S. Tanenbaum. A model for worldwide tracking of distributed objects. In *Proceedings of the TINA'96 Conference*, 1996.

[20] Carlos Varela. *Worldwide Computing with Universal Actors: Linguistic Abstractions for Naming, Migration, and Coordination*. PhD thesis, University of Illinois at Urbana-Champaign, 2001.

[21] Fips 180-1. secure hashing standard. U.S. Department of Commerce/NTIS, National Technical Information Services, April 1995.

[22] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.