# APPLYING DOMAIN NAME SYSTEM REAL-TIME
# REDUNDANCY TO THE CCSO
# PH PHONE DIRECTORY
# SYSTEM

By

JAMES M. JONES

Bachelor of Science

Florida International University

Miami, Florida

1976

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 1999

APPLYING DOMAIN NAME SYSTEM REAL-TIME
REDUNDANCY TO THE CCSO
PH PHONE DIRECTORY
SYSTEM

Thesis Approved:

Jacques E. LaFrance
Thesis Adviser

Jane Terry Nutter

Wayne B. Powell
Dean of the Graduate College

# ACKNOWLEDGEMENTS

I wish to express my sincere appreciation to my thesis advisor, Dr. Jacques LaFrance for his tolerance, patience, motivation, guidance, and advice. My appreciation also goes to my other committee advisors, Dr. Nutter and Dr. Peng, for their assistance.

Included in my thanks for their assistance in various forms are Ann Blakely, Charles Brooks, Sharon Corner, Reed Davis, Rick Kruse, Patti Kutzler or Whitaker (whichever she prefers), Robin Leech, Sandra Magee, Dr. Carolyn Quintero, Laura Daniel and Dana Saliba.

Of course, my family has been helpful in many ways and needs to be mentioned by individual name to be immortalized. Thanks for their constant prodding and encouragement to Nancy Pruett and Corky Searls, Barbara Jones, my kids: Ginger and Rhett, and my parents: Retired Professor Dr. James M. Jones, Sr., and the late Sheila B. Jones.

## TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# I. PURPOSE AND MOTIVATION

## General Purpose

The "PH" in "PH system" stands for PHone book and was initially designed as a relatively simple database system to maintain mostly information that one would find in a phone book. The "CCSO" in the title of the system refers to the "Computer and Communications Service Office", an organizational group at the University of Illinois, Urbana-Champaign that provided the public domain version of the software. The "Domain Name System (DNS)" refers to a specific convention, set of standards, technical requirements, and methods that describes identifying independent computing machines in a network. A "Domain Name" refers to a means of identifying a single member of the DNS network.

Since the inception of the PH system, among a number of functions, it has been given the capacity to forward E-mail. Electronic mail or "E-mail" refers to the electronic sending and receiving of messages to individuals in a computer network similar to the manner in which postal mail systems work. As a popular system for generic E-mail forwarding and other functions, the PH system lacks system reliability mechanisms. PH system failure can cause delays in E-mail delivery, and, in certain cases, "bounced" or lost E-mail. A typical PH system mail forwarding mechanism is set up in an Internet or Intranet type network that contains multiple machines. This multi-machine network structure provides an opportunity for multiple PH forwarding mechanisms without the need for any additional equipment. Thus the task of providing mechanisms to produce a higher reliability via multiple servers on existing machines seems a logical direction.

Reliability seems to be an interest for all Internet applications since often now organizations are placing a “front door” for customer usage and service via the Internet. Problems with access to the organization using Internet mechanisms can be at least embarrassing and at worst devastating to the activity of the organization. Commercial businesses that depend largely or wholly on Internet access can be eliminated quickly if they loose credibility or access to clients by having an unreliable or unproductive “front door” on the Internet.

The Domain Name System (DNS) with fault-tolerant features has been successfully in use on the Internet for approximately 15 years. This system has been amazingly reliable considering the structure and growth of the Internet. The PH system has been implemented for more than 5 years, but with only recent interest in the need for redundancy. The only efforts found so far regarding PH redundancy involve some recent work on downloading alternate PH servers on an overnight basis at University of Illinois, Urbana. This overnight-copying redundancy does not provide the level of dynamics found in the DNS.

As the Internet grows, it is likely that the dependency on E-mail will grow with it. As E-mail dependency grows, there is a need for more reliable E-mail delivery mechanisms. This project may be considered a contribution to the Internet community to provide some additional reliability to the community served.

## Overall Goals

The following are the overall goals of this project and paper:

- Show how simple, yet elegant, reliability mechanisms can be developed with less difficulty than may have been previously considered.
- Produce a fault-tolerant enhancement to the PH system.
- Make the product enhancements available to the Internet community.

# II. BACKGROUND AND HISTORY

## PH Description

### General Description

The CCSO Nameserver Directory System (known generally as the PH system or the PH Nameserver system) is a client/server database that provides information primarily on individuals. Initially, the PH system was used to store and retrieve personal information such as name, address, phone number, and E-mail address. As is common with many database applications, other features were added, making the package more attractive for widespread use. In addition to these other features, additional types of information such as special interest groups, administrative departments, telephone area codes, local weather, restaurants, and other information can be entered, maintained, and easily retrieved.

### Internal / Technical

The server software, know as the QI server (Query Interpreter), is written mostly in "C" for the Unix environment in general. The server software is coded so that it can be compiled and run on almost all Unix environments. Client software, known as PH (short for PHone directory) has been developed for most common computer systems. Since the source is readily available via the Internet, producing a client package is relatively simple, provided that the supporting systems are available. Supporting systems include mainly software supporting network connections using the "Transmission Control Protocol / Internet Protocol" (TCP/IP) network architecture to the QI server. The TCP/IP protocol is a well-documented and popular network architecture used throughout the Internet and on other private networks. The product is efficient, yet simply engineered to provide a limited set of functions.

## Features

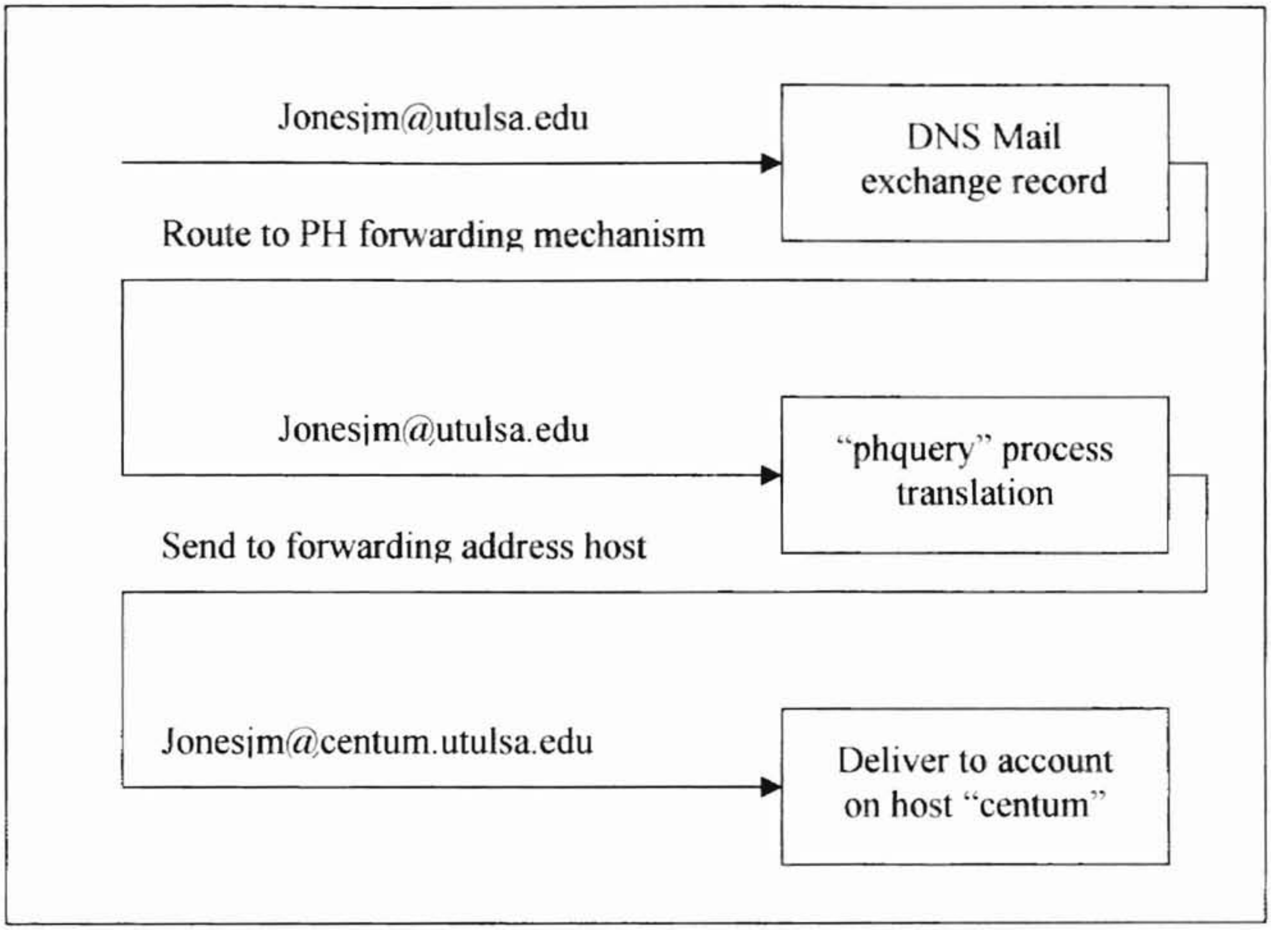
The PH Nameserver software system provides several additional popular features beyond those of a common "phone book" or "white pages". These additional user-oriented features that have popularized the package include:

- Automatic E-mail forwarding service
- Dial-In user validation
- User maintainable information

### E-mail Forwarding Feature

The E-mail forwarding feature seems to be the most popular feature in use at the University of Tulsa. The E-mail forwarding feature provides users with a "generic" E-mail address at a site. The PH system is then tied to the central E-mail processing system to forward each user's E-mail to the specific machine where the user maintains an account. What appears to be a rather cumbersome and complex technical process provides a service whereby a user may present a fairly simple E-mail address in correspondence, business cards, and the like. The simple address can be provided without concern about physical machines changing names or becoming obsolete so that the user's account might be moved. Should users need or want to move their accounts with PH forwarding in place, they can change the forwarding E-mail address on the PH database. Without the PH system, the user would normally have to notify all others that send E-mail to the user of a new address should the user change accounts to another machine.

The format of the address that the user would supply to the "public" typically appears "user@domain" where "user" is what is termed the PH "alias", and "domain" is the Fully Qualified Domain Name (FQDN) of the domain that the user will receive E-mail. The "FQDN" is an absolute form of referencing a Domain Name that prevents local interpretation of portions of the name and thus eliminates ambiguity. The "email" field (normally used for forwarding E-mail) is set to the user's mail account on a particular machine. The format of the forwarding address would be "user@machine.domain" where "user" would be the user's mail account and would not necessarily have to match the "user" in "user@domain". "Machine" would be a specifically named computer in "domain". "Domain" would be the domain name where the user's E-mail account would reside, and, as in "user", would not necessarily have to match the "domain" in "user@domain". For example, the generic address "jonesjm@utulsa.edu" would be routed to the PH system; the PH system would find the address "jonesjm@centum.utulsa.edu" in the email field of the record with "alias" "jonesjm", the E-mail would then be automatically forwarded to the forwarding address.


Jonesjm@utulsa.edu
DNS Mail exchange record
Route to PH forwarding mechanism
Jonesjm@utulsa.edu
"phquery" process translation
Send to forwarding address host
Jonesjm@centum.utulsa.edu
Deliver to account on host "centum"

PH Forwarding Diagram

Validation Feature

The dial-in user validation feature can be directly used by networking equipment such as dial-in modems. Cisco Systems has equipment that can take advantage of this feature of the PH system. Cisco provided the TACACS (Terminal Access Controller Access System) software package to the public domain to make this feature easily usable with certain networking Cisco equipment and a PH server.

The University of Tulsa uses this feature to validate users who dial-in to TU's modem pools of 16 14.4 KB and 16 28.8 KB modems to access the TU network and the subsequent TU Internet gateway if desired. The user provides their PH "alias" as a username then provides a password associated with the user (that can also be maintained by the user) to authenticate against the PH database prior to being permitted access to the TU network. The PH "alias" is set up on the TU PH database to be a field that functions as a unique key to the records in the database.

### User Maintenance Feature

The user maintenance feature of the PH system is probably used least, but may be one of the primary reasons that the whole PH system is so popular. The PH system provides the mechanism for "self-service". In this age of burgeoning bureaucracy and overwhelming administrative red tape, placing a little bit of self-reliance in the hands of users would appear to be a trend. The overwhelming popularity of the Internet seems to speak for this trend. There were a number of "on-line" services before the Internet, but due to their proprietary nature, they couldn't come close to the enormous growth of the Internet with its "open" architecture. Even though the Internet has been archaic to a great extent and initially "unfriendly", it held the promise of the self-service era. This era would provide users with features to manage and take control over their own information and methods.

The mechanics of the user maintenance feature of the PH system are similar to most other systems that require user validation. Typically, read access to most types of information is available to anyone, though there are certain configurable restrictions. Modification of information is generally provided only to the user associated with the information via a "login" sequence. The common "username" and "password" requirements are solicited for the login sequence. The same mechanism used to authenticate users for using dial-in facilities (described above) is used here to "login". For example, again, the TU PH "alias" field is entered as the "username". Once logged-in, a number of data fields may be altered, such as changing an E-mail forwarding address. This E-mail forwarding management by the user is one of the primary combined features that makes the PH package attractive. Users can identify which account and computer they want their E-mail sent to when directed to the "generic" PH address.

Other features of logging-in to PH enable the display of more "personal" information that might not be permitted with the generic querying of the PH database. Such personal information might be a Social Security number.

## PH History

Records indicate that the PH system was initial developed in 1989 by Steve Dorner at University of Illinois, Urbana-Champaign (UIUC)[1]. Dorner modified and overhauled software developed at CSNET for the same purpose. Paul Pomes took over software maintenance of PH in 1992 when Steve left UIUC. Through the efforts of a number of individuals, the PH system has been ported to most UNIX platforms, such as Ultrix, NeXT, Convex, Dynix, AIX, Sun, and HP.

Tim Lawless at the University of Southern Mississippi (USM) got a modified version of PH running on a Linux environment in 1996. As part of this project to enhance PH, I was able to get a Linux version operational with a minimum of changes to the original software. The changes that I made should not produce problems in other platforms, and, in fact, should reduce or eliminate certain problems with the configuration and running of the software on non-Linux platforms. I have posted the configuration file used, with comments on other changes, to the PH listserv to make the Linux version available to others in the Internet community.

Many others have been involved in the implementation of the software on other platforms and have also been gracious enough to enhance and debug the software and supply their work to the Internet community. Though many were involved, Alan Crosswell of Columbia University made one notable effort in 1993. Alan did a number of updates on the utilities that are used to build the PH database. Most changes were to speed up the processing of the generation of the database files. His efforts changed processing times from around 10 hours to about 15 minutes.

A software registration service with UIUC reveals that the PH software is being used at no less than 329 sites around the world at last count.[2] The registration service is only for those servers accessible through the Internet. Since the registration service is voluntary, and some sites may not desire promoting the use of their server since some consider the information to be reasonably personal, it is quite likely that a number of other sites are using the software on the Internet without registration and private and corporate networks connected via firewall or not connected to the Internet are running PH.

## Domain Name System Description

### General Description

The Domain Name System (DNS) primarily consists of what might be described as a loosely coupled cooperative hierarchical networked database system that enables

[1] Dorner

[2] Kubaitis-1

translation primarily between names and numbers that identify network hosts. The "names" typically consist of alphabetic strings that provide easy descriptions for human reference. The numbers, often called "IP addresses" are currently in the form of "nnn.nnn.nnn.nnn" where nnn is a number in the range of 0 to 255. Network devices use these IP numbers to direct packetized information to and from specific areas and computing equipment on the network

### Internal / Technical

The Domain Names, often called "IP names" (for "Internet Protocol names") or "Internet names", are arranged so that the portion of the name that designates the top of the hierarchy is to the right and each successive lower logical hierarchy branch name is to the left with "." (dot or period) as a separator. A "Fully Qualified Name" is the "node" or individual equipment name on the left with all the hierarchy names, from lowest to highest, to the right, separated by a dot and ending on the right with a dot. The rightmost dot itself designates what is termed the root of the hierarchy. A "domain" or "domain name" is normally a higher level grouping where the maintenance of designating IP numbers (with range restrictions) and node names is provided to an administrative entity.

IP numbers are organized in a hierarchy where the highest point in the hierarchy is to the left. Portions of the IP numbers can designate what is termed a "sub-net". Sub-nets are some portions of the numbers from the left, but not enough "binary digits" to designate the final computer device, but enough designation to determine what logical continuous segment of network that a machine or other sub-net is located on. A segment of a network is a logical grouping of machines and/or other sub-nets.

An example of such a name might be "a.cs.okstate.edu.". Though the exact particulars cannot be determined exactly without knowing the details of the hierarchy, the "a" probably means an individual machine on a "sub-net" of "cs" in the "domain" "okstate.edu.". Note the dot at the far right designating the root. The root is a "domain" in itself. The "edu." portion in the current scheme on the Internet is also defined as a domain. The logic in determining how the FQDN is evaluated is defined in the DNS database, and, though well defined, can be non-trivial. In the example, there is no reason that "cs.okstate.edu." cannot be defined as a "domain" also. At the time of this writing, the FQDN "a.cs.okstate.edu." represented:

- "a.cs" a machine
- The "cs" portion, though not officially designated a "sub-net" consistently represented machines on one "sub-net", in this case, the sub-net: 139.78.113.? or rather 139.78.113.0 since a zero ("0") in an IP number has special meaning.
- The domain "okstate.edu."
- The domain "edu."
- The domain "."
- The single IP number for the machine "a.cs.okstate.edu" is "139.78.113.1"
- The machine "a.cs.okstate.edu." serves as a "mail exchange" server for a list of other IP names.

Additional Features

The DNS provides for more services than an IP name to IP number translation. A less dramatic, but none-the-less useful function of the DNS is translating the IP number to an IP name. In addition, one of the more useful functions of the DNS is the designation of a "mail exchange" server.

### Mail Exchange Feature

A mail exchange record associates what would appear to be an IP name to a machine that may or may not match that name, but may direct E-mail designated to one IP name to be routed to a different IP name. This mail exchange feature provides a mechanism for routing E-mail to alternate E-mail servers. Though not designed into the DNS for the needs of the PH system specifically, the "alternate E-mail server" function of the DNS is a key mechanism in the additional functionality provided by the PH system.

### Fault-tolerant Function

The DNS has several key design considerations for various reasons. Our focus will be on the fault-tolerant features provided by the design of redundant servers.

Of all the contemporary systems that are relatively close in comparative functionality to the PH system, the DNS appears the closest. Certain DNS techniques appeared to be the most likely candidates for emulation by the PH system. The mechanisms applied to PH through this project are contained as key design features of the DNS, namely fault-tolerance via system redundancy. Previously lacking in the PH system, a fully functional form of the fault-tolerant design has been implemented. Details of the fault-tolerance are illustrated in chapter III, Reliability Mechanisms.

### DNS History

It became apparent in the early 1980s that copying the host tables from system to system and attempting to maintain a central host table file for the Internet would be unacceptable from an administrative standpoint due to the growth of the Internet. Paul Mockapetris, a principal engineer on the Internet Engineering Task Force, provided a schedule in 1983 detailed in "Domain Names - Concepts and Facilities" (RFC 882) to migrate to a new methodology and design whereby managing of sections of the net, termed "domains", could be implemented. The work to refine the scheme continued through the later 1980's and resulted in the refinement publication "Domain System Changes and Observations", RFC 973 (1986), and the current set of works "Domain Names - Concepts and Facilities", RFC 1034 (1987) and "Domain Names - Implementation and Specification", RFC 1035 (1987), are the foundation of the DNS as it stands today.

## DNS and PH Comparison

### PH and DNS like vs. unlike features

The following lists are summary information that illustrates some of the overall commonalties and differences. Details of the more significant comparisons are discussed following these lists.

Like Features

- relatively simple database
- Internet systems
- use a reserved TCP/IP port number
- reasonably easy to configure and set up
- very flexible in their configuration and characteristics and functions
- multi-tier architecture
- widely used
- seasoned software – many bugs removed
- features added – enhanced – extended
- simply engineered
- well defined functionality
- written primarily in "C"
- modularized
- use Unix "make" function
- similar coding styles
- similar copyright style

Unlike Features

DNS and PH Like vs. Unlike Features

| DNS | PH |
|---|---|
| The DNS is widely used throughout the Internet. | The PH system is widely used, but not nearly as widely as the DNS. |

| | |
|---|---|
| The DNS is a cornerstone foundation system for the current state of the Internet as a whole and failure of the DNS in part or whole would likely cause significant disruption on the Internet if not failure of the system as a whole due to the extent of the dependencies. | The PH system is not a basic function needed for the operation of the Internet. |
| The DNS database is completely distributed. | The PH database is completely standalone. |
| The DNS database has a hierarchical database structure. | There is no equivalent structure in the PH database. |
| The DNS database is fully and dynamically redundant. | The PH system did not have redundancy built-in prior to this project; dynamic redundancy is now a feature. |
| The DNS is designed for caching. | There is no caching mechanism in the PH system. |
| A team of software engineers (primarily Paul Mockapetris) designed the DNS. | The PH system appears to have been designed and written by one individual (Steven Dorner). |
| Documentation on the DNS is extensive and thorough. | PH documentation is sparse and limited. |
| The DNS has extensively designed-in interactive mechanisms and is distributed and dynamic. | A PH database is a completely separate entity from all other PH databases; there is no static or dynamic database interaction. |
| The DNS has "mail exchange" records that the PH system can and typically does use, particularly the project associated with this writing. | The DNS has no dependencies on the PH system. |
| The DNS system, though primarily a client – server model, can be classified a multi-tier system. | The PH system, prior to this project, was a basic client – server model; redundant servers add a 3rd tier. |
| Specifications for the DNS define records but do not define the structure of the database itself except to say that it is encouraged that the software be efficient and designed primarily for speed of look-ups. | The PH documents define exactly the internal structure of the database and the mechanics for database usage. |
| The DNS records have fields typically delimited by spaces. | The PH system uses 3 different field delimiters depending on the relative location of the data. |
| DNS has a fairly limited and well-defined record and field types. | The PH system has a dynamic method to define and alter field types and field and record characteristics. |

## Client/Server Model

Both DNS and PH follow the Client/Server Model in general. The DNS has features of a more elaborate structure with, among other components, Primary and Secondary servers. With the completion of this project, the PH system has a similar feature of Primary and Secondary servers. With the addition of the Secondary server, the structure of the PH system is more like the DNS.

The DNS Primary and Secondary servers replicate the server function from the client perspective. The client typically does not need to know which, if any of the servers specified are in fact the Primary server. The Primary and Secondary servers themselves are aware of the type of server that they are. The Primary knows about all the associated Secondaries and each Secondary knows of the Primary in the DNS. This "knowledge" in the DNS is modeled in the project changes to the PH system.

Upon examining the version of the PH client software that comes with the distribution, of note is the "enhancement" in the software to reference a "main" and an "alternate" server, though there is no direct provision for having a Secondary server. Paul Pomes of UIUC (at the time I spoke with him) indicated that UIUC copies the data base on an offline and overnight basis to provide a non-dynamic Secondary server. The offline copying mechanism is not described or provided for in the release from UIUC, the same release that contains the PH client that refers to an alternate server. While the offline copying is certainly functional, it does not provide the level of reliability that the dynamic zone transfers provides in the DNS. The "zone" portion of "zone transfer" refers to a logical collection or grouping of Domain Names. Details of these zone transfers is discussed elsewhere.

Among other dependencies, offline copying must rely on a separate set of mechanisms that must be operational for the copy and activation to work. The dynamic copying via the "zone transfer" mechanism modeled after the DNS provides for a less dependent system by relying on the same mechanism that the servers and clients uses. Also, the servers communicate directly with each other rather then depending on a separate offline process to copy the database and make it available.

## Database structure

The documents of the DNS specifically define the database record types that are presented in the Man-Machine Interfaces (MMI). A Man-Machine Interface is the point of communication between a person and a machine, typically a computer. The internal form of the database is intentionally not detailed in the DNS documents. The DNS documents indicate only that the internal database records need to be quickly retrievable, thus leaving the software engineer to determine the internal structure that might best fit the circumstance, given other variables, such as machine architecture and software language. The PH system on the other hand is very well defined in its internal database structure. Though defining the internal structures and methods produces a more limited framework, the internal workings give light to quirks and anomalies of processing where, as in the DNS, internal workings may vary and thus mask the reasoning for certain internal system behaviors such as unusual system delays and order of information returned.

Where the DNS is rigid in its record characteristics and types and dynamic in its internal structures, the PH system is just the opposite. The PH system though rigid in internal data structures, has a configuration file that defines record and field characteristics, not in structure, but in function. For example, one can customize the PH system to have organizational department records that are not necessarily associated with an individual. Where this feature might first appear to be difficult to work with from a "client" environment, as it turns out the generic "client" software, for the most part, simply reformats the information passed to it from the PH server and displays it to the user. The specialized clients in the PH environment are well defined and, typically, one would not want to customize the fields used by these specialized clients. Examples of the specialized clients are the process that forwards E-mail (phquery) from the generic address to the specific address and the authentication software that verifies users dialing in and logging into the PH database. In fact, there are warnings in the PH software and documentation to discourage modification of these specialized fields.

### Other characteristics

Though there is some concern for security and privacy in the DNS data, for the most part, access to any and all of the data throughout the entire DNS is relatively simple. The data in the PH system is typically more sensitive. Since the data is of a more personal nature, some individuals prefer not to make the information known to the outside world via electronic "publishing" of a sort. The train of thought is similar to why some people do not want their phone numbers listed in a phone book, although, in some cases, users don't mind paper publishing of the information in PH, but do not want electronic publishing of the information. The authors of the PH system were very much aware of the sensitivity of the data in the PH database. There is a configurable parameter in PH that prevents the return of any information should a request be made that would return 25 or more records. There is no equivalent in the DNS. Without going into great detail since the DNS is not the topic of this paper; an understanding of some of the general characteristics of the data contained in the PH system illustrates how the PH system works. A current very common problem today on the Internet is one of junk E-mail, most commonly referred to as "spam". Due to the usually very low costs of sending E-mail solicitations and mass mailings, and the current state of transition of the Internet ("self" regulation for the most part), "spam" has become widespread and a significant problem. The "fuel" for "spam" is getting as many E-mail addresses as possible. Thus any method of obtaining as many E-mail addresses as possible is very attractive to "spamers". Programs have been written to "crack" PH servers and obtain as many E-mail addresses as possible.

As can be noted from the above illustration, open access to a PH server or the possibility of another "back door" to access the PH server database is an important concern. For this reason, the zone transfer feature added to the PH system has a more secure method of access than that provided by the DNS, though the mechanics are the same otherwise.

# III. RELIABILITY MECHANISMS

## Background

### Traditional "Fault-Tolerance", i.e. Backup Copies

Discussion of the traditional "fault-tolerance" techniques will be followed by the techniques and reasons for the more contemporary approaches for on-line redundant data bases such as the DNS.

Backed-up data typically is used in what is generally referred to as an "off-line" state; that is, the data is not readily available for reference. When the original data is copied to an "off-line" backup, this often implies that the data is copied for backup either when the original data is placed in a state of unavailability, i.e. the original data is in an "off-line" or temporarily unusable state. The primary purpose of placing the original data in an off-line state helps to ensure that the copied data is a complete "snapshot", i.e. a completely uncorrupted state. The "off-line" state is attained when all update transactions are complete and any further updates are somehow prevented. Copying data while permitting updating the database can be complex, since methods used must take into account locking mechanisms and update order to preserve database integrity.

One problem in copying data from a database off-line is determining whether the data base is completely off-line, i.e., all updates have completed and no updates are in progress. Common mechanisms used to place databases off-line are sensitive enough to the database structure to know when any given transaction is complete, and place a moratorium of some sort on any further updates. Simpler techniques for obtaining the off-line state are eliminating access to the database either by "disconnecting" users, generally referred to as logging off users, or "disconnecting" the database. "Disconnecting" the data base may vary by operating system and can include "dismounting" a disk volume that the data base is contained on or terminating access by terminating the application program referencing the data base. The methods are many and varied and can include "gross" techniques. One such gross technique involves shutting down the machine, which can in itself can cause data base corruption on the original database; then restarting the machine in a mode that prevents users from obtaining access and then running physical disk to disk or disk to tape copies.

Commonly obtained with some mechanism for sensing the logical completion of a transaction, the popularity of the database snapshot copy for a backup is common due to the simplicity of the copy. The snapshot copy uses certain simple system information that defines the location and physical scope of the database for the information to be copied. The simplicity of this technique generally translates to significantly more rapid copies of the database than those techniques that use internal data base structures. With the snapshot technique, logic to follow internal data structures need not be followed and no record of replicating previously copied information need be kept as would be the case in data bases that provide for multiple pointers referencing identical information.

The other extreme in backups includes "fine" techniques. An example of such a technique provides for an application to be signaled that the database needs to be backed up and, in turn, completing updates, notifying the user of "temporary" inaccessibility, signaling the backup that it is "safe" to do the backup. The final step in such a process has the backup signaling the application that the application can continue. Another method even "finer" never indicates to the user that the database is unavailable, but goes through the database via the database internal structure and monitors active locking mechanisms and "journals" change activity. The "journal" is a record of all changes made to the database during the backup process. When the backup process completes, the process can go back and append changes that were made to the database. The most sophisticated of these finer processes issue "soft" locks on records, such that should the application/user request a record with a soft lock, the record is released so that the user never has any indication that a background process is running. The backup process then records the fact that the lock has been "taken" and processes the record with journaled information at a later time when the lock is released. As the backup routines go from "gross" to "finer", they generally develop a high level of complexity which translates to more overhead in both processing resources and code development and maintenance.

In any case, the off-line techniques typically will interrupt normal work activity, and are less desirable than on-line techniques from a "system usefulness" standpoint. Often, though, in average conventional production systems, activity during "off" hours can be used to provide a time frame where little to no required database updates are needed. Business systems of this type are disabled to the user after normal work hours for utility processing that includes a data base backup. These "gross" techniques are still very common. "Fine" techniques have been fairly rare. One of the most well known of the "fine" techniques, airline reservations systems, have long been a target of complex algorithms and methodologies to ensure data integrity and high availability. These complex techniques with their elaborate locking methods have had a high overhead with large numbers of technical programmers and support personnel regularly and carefully refining the complex software. With the popularity of the Internet, high available systems become the target of greater interest.

## Contemporary Fault-Tolerant Strategies

Conventional situations are, at best, unusual on the Internet. The norm on the Internet seems to be one of continuous availability particularly for the business world. Thus the methods for backups and other reliability mechanisms must be approached differently.

Providing Internet access to a database for the global community requires rethinking of redundancy and backup strategies. Certain types of applications need to use the "finer" backup methods yet continue to go beyond backup strategies moving on to fault-tolerance. Fortunately, the data in the DNS and the PH system do not require the absolute consistency and accuracy of an airline reservation system or a financial system. Databases, such as the DNS, PH system, and likely others, can be manipulated, distributed, and proliferated without concern for total and complete database consistency and integrity as other types of systems.

A classic example of a highly reliable and simply engineered application, and primary model for this project, is the DNS. The DNS has been in wide use on the Internet for around 10 years now with little to no significant network wide disruption. The primary designer and engineer, Paul Mockapetris, considered functionality more important that data integrity, a fairly revolutionary concept in the Computer Science world. "Access to information is more critical than instantaneous updates or guarantees of consistency." (Mockapetris, RFC 882, 1983) Though this assumption breaks with conventional database thought, the assumption is very workable in an environment where the absolute accuracy of the data is not critical.

This assertion is not meant to imply that data integrity was not important in the design. The engineers introduced methodologies and mechanisms to encourage data integrity, but did not guarantee accurate data. By design, the "bad" data does eventually migrate out of the system.

## Redundancy Level for Fault-Tolerance, Data and Function

### Duplicated data bases

Fully or partially duplicated database applications have been somewhat limited and often have been specialized. The two most prominent approaches to maintaining duplicate data are hierarchical and peer methods. The hierarchical method is organized such that there is one database that is considered a "master" copy and all other copies are generated from the master copy. The master copy provides a logical reference point for promoting accuracy. If there is a discrepancy between the master copy and duplicates, the master copy is presumed more accurate.

A peer method of maintaining duplicate data implies that all copies are theoretically equally updated and current, though, from a technical standpoint, this is all but impossible. Typically what occurs with peer systems is that individual updates are initially associated ("anchored") with one of the servers/databases and updates are performed from the point of anchoring to the other databases with signaling and "checking-off" as the updates are processed throughout the database system. The net effect is the equivalent function of the hierarchical structure, except any database copy may serve as the master copy for any update.

In either case of hierarchical or peer methods, typically extensive signaling, locking, and often transaction journaling mechanisms must be in put in place to ensure system reliability. These mechanisms can create significant overhead and thus have an impact on performance.

Other more complex methods provide for three or more copies of the data to exist. When discrepancies are discovered, the mechanism might determine via probabilities that the best chance for accurate data is based on the "majority", i.e. whichever copy exists identically in more that 50% of the locations is determined to be most correct and accepted as so. A system using more than three copies might maintain an odd number of copies to ensure that a "majority" could exist. These types of duplication are not nearly as common as the "master and one or more duplicates" method, primarily since the logic is fairly complex, therefore contributing again to high overhead and resulting reduced performance.

Fortunately, with the specialized database of the PH system and the DNS, a much simpler database update mechanism can be used.

### Raid arrays

One highly successful alternative to the "non-master" duplication method is Redundant Array of Inexpensive Disks (RAID) systems. Of the 7 RAID configurations, 2 have proven to be practical in providing useful redundant data. RAID 1, often know as "shadowing" or "mirroring", duplicates physical data on 2 or more physical disks.

Duplication of the data using RAID 1 can cause some performance problems due to the slowing in the writing mechanism using 2 or more disks. Performance of the read mechanism is generally increased, though, due to intelligence on the disk controller to read the most quickly accessible data.

Though performance is not a main topic here, from a practical standpoint, performance needs to be considered. While moderate performance degradation may be generated, if the degradation is kept in the moderate range, the overall benefit of the methodology may be of such value as to accept the degradation characteristic. Using RAID 1, with most common databases, the performance benefit gained through read activity far outweighs the moderate performance degradation for write activity.

Some databases that do not fare well are those with a large proportion of write to read activity where overall I/O activity is also high. Though these types of databases are becoming more and more common, they are still in the minority. A high write activity database often requires certain special treatment and structure due to performance needs. An example of such a database might be an application that collects a number of requests from a number of users, then redistributes these requests on to other organizations with a different sorting arrangement. While this might sound like an application that could be insignificant, consider a company that provides rapid driving record information to trucking and insurance companies, automated medical claims processing systems, or stock market companies that combine or group stock trades. All applications would require the rapid influx of large amounts of data followed by the rapid resorting and redistribution of the data in a timely manner. RAID 1 does not perform well in this environment.

RAID 5 provides the means for the other common and practical data redundancy capability. RAID 5 asserts that for the one or more disk volumes used for applications, there exists a disk volume that contains a specialized type of redundancy information based on the applications data. In other words, should any one of the disk volumes fail, the remaining volumes can be used to provide the original data through one of two methods. Should one of the application volumes fail, the redundancy volume can be used to regenerate the data on the failed volume by "reversing" the algorithm used to create the redundancy data. If the redundancy volume fails, the application data is not directly affected; the controller need only stop generating the redundancy information. There is a trade-off in RAID 5 between that of RAID 1 in that while RAID 5 requires more logic, translating to overhead, due to the calculation of redundancy volume, RAID 1 requires more physical disk volumes for application data that must be contained on more than one physical disk. RAID 1 requires each application volume be "shadowed" whereas RAID 5 only requires 1 redundancy volume for the entire set of application disk volumes. The net result makes no difference in redundancy, but in performance. RAID 1 normally performs better than RAID 5, but costs more since more hardware is required.

Most contemporary RAID systems use hardware or firmware level mechanisms to provide functionality. This method has been shown to provide less overhead and makes implementing a RAID system significantly simpler. Typically no other software changes need be made. The fact that this method has been easy to implement has been a boon to RAID system providers due to a natural progression of data base systems. The progression begins with the creation and use of a database by a user or user group, the growth of the database, followed by the dependency on the database and the realization of the dependency. The hardware or firmware localization of RAID systems permits fairly large and/or heavily used databases to be retrofitted with RAID systems. The segregation of the RAID systems from the application and system software in conjunction with hardware and firmware implementations has provided engineers with a situation that can produce exceptionally reliable systems. The RAID implementation, being at a hardware level, might have the feature of an isolated or partially isolated power source such that duplicated updates can be assured. The isolated power source and other engineering methods including redundant logic can provide a situation that eliminates a most significant problem of timing updates and single update failure. Since the problem can be eliminated from a practical standpoint at the hardware level, the system and application will be provided with a high degree of reliability.

### Multi-node arrays

There are numerous variations on the multi-node array topic. The 2 more popular methods are the multi-node / same database and multi-node / duplicate database configuration. The "same database" configuration typically requires some basic hardware component and operating system modifications or layered enhancements for a locking mechanism. The "duplicate database" configuration is an emerging technology of which the DNS is such an application. The "duplicate database" mechanism requires no special hardware components, making these systems more cost-effective in terms of hardware components.

Initially, the need for the "same database" type of configuration was partially driven by the sheer volume of data and the needs for equivalent bandwidth to be able to service requests of the data coupled with the relative high cost of the storage media and high bandwidth equipment. Though quite popular, the nature of the Internet is pressing the data out in a distributed fashion such that nodes do not necessarily contain the large volumes of data nor the needs for the high bandwidth, but now the network bandwidth itself has become the limiting factor.

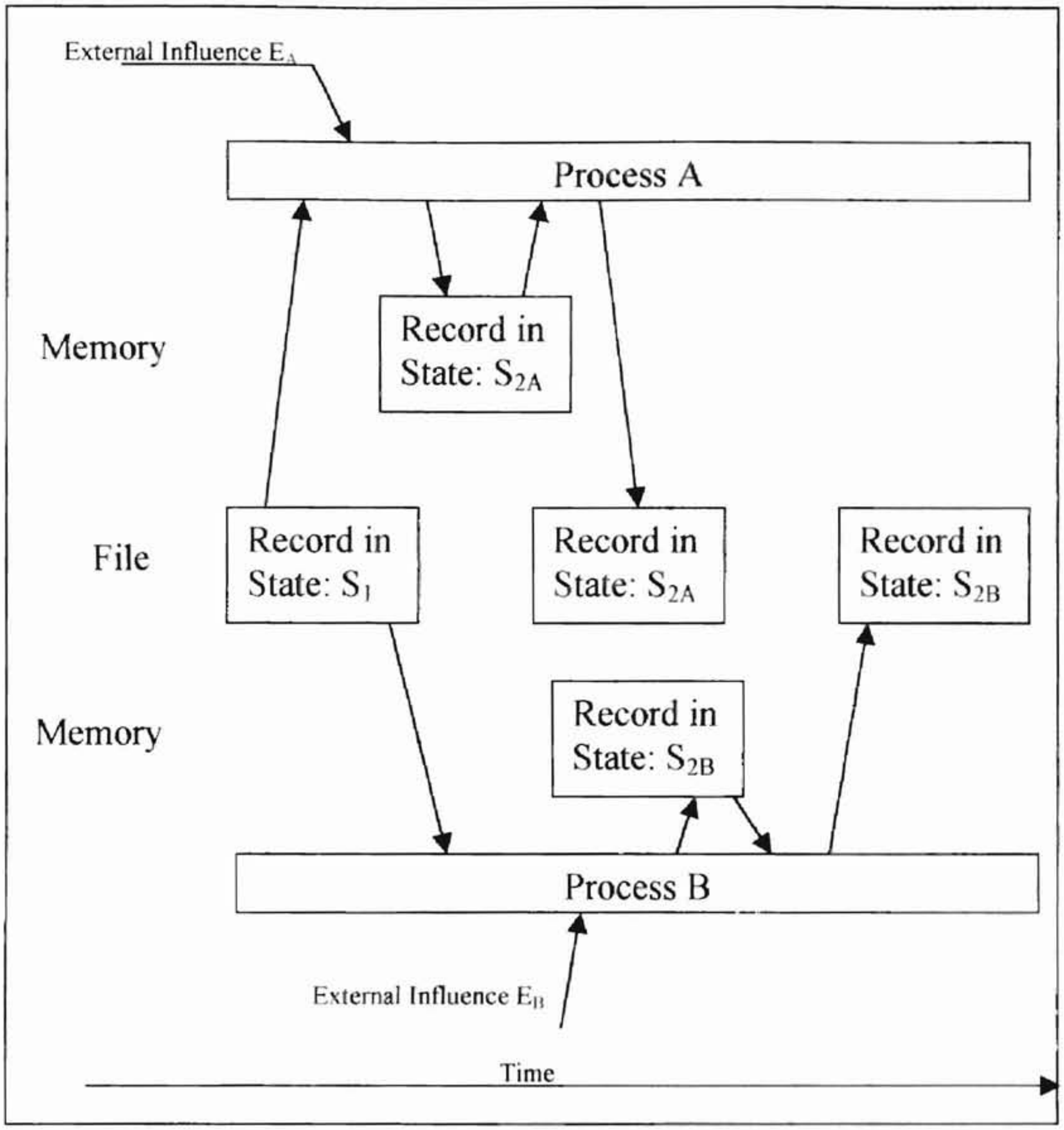
The primary software mechanism built into the "same database" configuration is a locking facility that must span nodes. Digital Equipment Corporation (DEC) successfully designed and built such a mechanism that is a part of their VMS operating system. DEC's terminology for their mechanism is "clustering", referring to a grouping of several nodes into one logical entity. The locking mechanism DEC uses incorporates a duplicate database of locks on at least 2 of the nodes in the cluster, so that if one node fails, there is still a complete lock database on another node. The duplicate lock database is reliable, but it consumes significant node communication bandwidth and CPU resources with most application database processing since common application database processing creates and releases locks information at a high volume as a normal result of record processing.

"Duplicate database" methods are put to use by the DNS and this PH project. The duplication is relatively reliable due to the lack of a high volume of changes and the lack of need of reliance by the system on the data always being totally accurate. The inaccuracies and incompleteness of the data is acceptable for the manner in which the data is used. Should other systems require more accuracy or absolute accuracy then these techniques cannot be applied directly, though some variation of the duplication might be produced to provide an acceptable mechanism. Fortunately, the nature of the PH data is such that the PH fault-tolerant functions can be modeled after the DNS.

### Timing and repetition

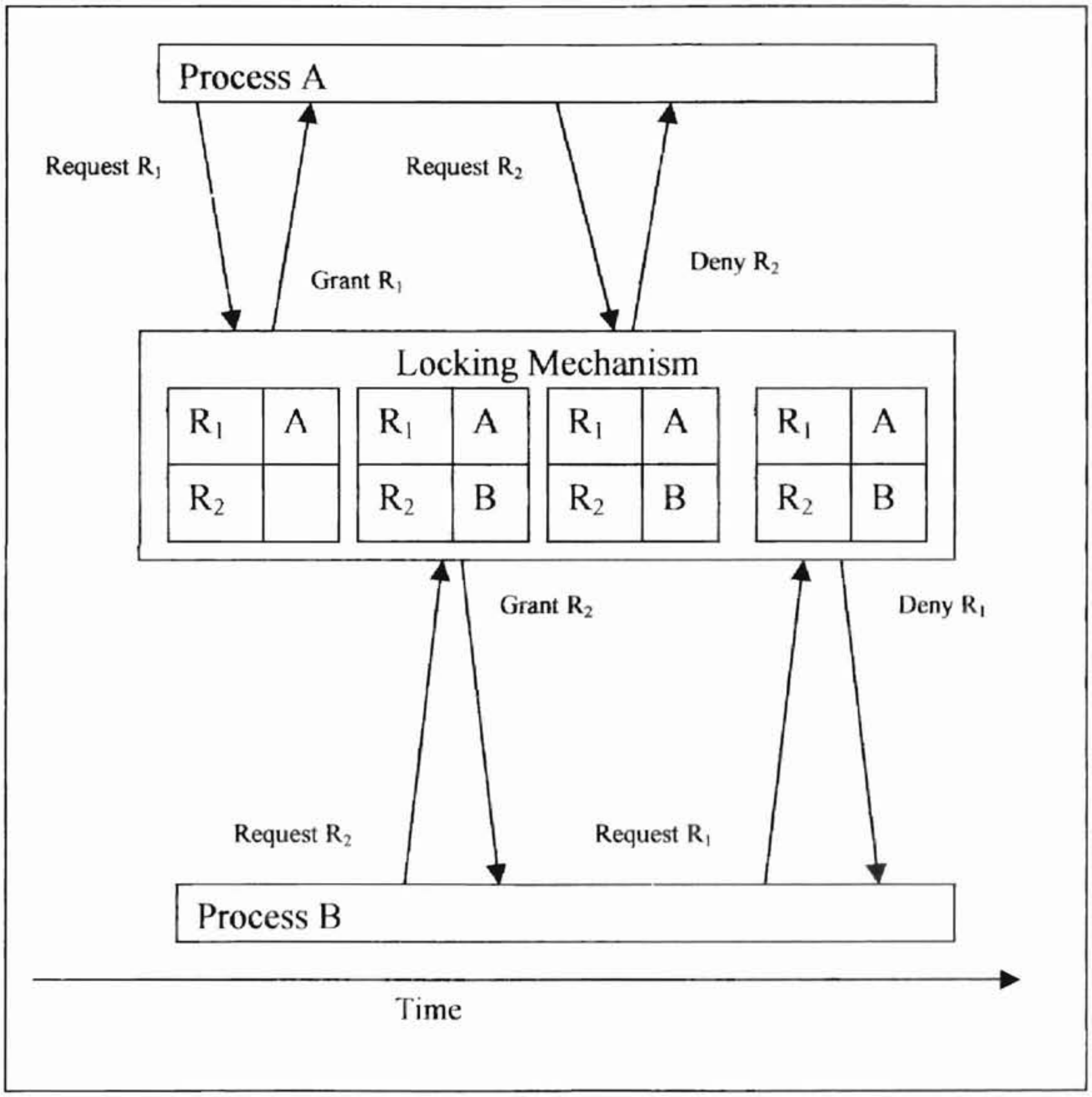
Timing and repetition play a critical role in all database schemes. There are many mechanisms that deal with timing and repetition. Here we review several of the mechanisms and the more common problems. We will not go into great detail, because, as it turns out, a rather simple mechanism is used due to the nature of the database.

The two most common problems with timing and repetition are the "race" and the "deadlock" problem. For our purpose here, we will include in the race problem, the failed system that does not complete or start a requested update.



Race Problem

The race problem is illustrated by the figure captioned "Race Problem". The problem occurs when a record is retrieved by 2 or more processes in the same state, $S_1$, are changed to 2 or more separate states, $S_{2a}$ and $S_{2b}$, by these processes where each change is different and is based on $S_1$ and some external influence, $E_a$ and $E_b$, but are recorded without knowledge or influence of each process on the other. Given the actions of recording the changes cannot occur simultaneously, one of the effects of the change will be lost with serial recording of the 2 transactions; the transaction recorded first, $S_{2f}$, will be discarded as a net effect. $S_{2f}$ is recorded, but the subsequent recording of the other or next record modification, $S_{2n}$, causes the evidence of $S_{2f}$ to be lost (unless there is some sort of journaling or other mechanism not described here) due to the recording of $S_{2n}$.



Deadlock Illustration

A deadlock situation occurs (see "Deadlock Illustration") when 2 or more requests are made for resources in such a manner that the requests can never be fulfilled.

The classic example follows: Requester A requests and obtains resource $R_1$. Requester B requests and obtains resource $R_2$. Requester A then requests resource $R_2$ and requester B requests $R_1$. Both requesters need to obtain the second resource, but are unwilling or unable to release the first resource they have obtained. The results are that the two requesters cannot proceed. This deadlock situation is well documented and discussed in many publications. Again, this is not covered in detail since a simple mechanism that is based on certain characteristics of the data is used. The point here is to emphasize that these problems can be and indeed are complex in nature. These points are only touched on to illustrate the number of complex problems solved by avoidance, or perhaps a better term is side-stepping.

## DNS Fault-Tolerant Features

### Authoritative and Primary/Secondary Facilities

The most common and known methods of maintaining duplicate databases generally fall into the category of either hierarchical or peer based structure. These two structures imply how and when the data on the corresponding server can be considered reliable. The hierarchical structure implies that the apex of the hierarchy is always considered to contain the reliable data and all other levels may have some doubt about their level of data reliability. Inconsistencies between a lower level and the top level database are resolved by assuming that the top level contains the correct data.

The peer structure typically does not associate level with authority, but uses other methods to resolve database differences. The most common methods to resolve differences use algorithms that imply integrity of one or more of the peer data over another. Typical algorithms are quorum based or change based. Quorum based algorithms normally require a majority of servers to agree on a data value if there is an inconsistency found. Change based algorithms normally assume that the first change made to data on a server is the most correct and propagates to other servers with often complex locking and timing mechanisms to prevent race and deadlock problems. Most of the peer methods can be thought of in hierarchical terms since there has to be some determination of which server(s) have the most reliable data. Thus the peer methods are often the same as hierarchical methods, but applied to each element of data instead of the servers themselves.

The Primary/Secondary method of the DNS falls into the hierarchical method of database duplication rather than the peer method. Though the hierarchical method is often considered less elegant than the peer method, it is often easier to implement and maintain since it is simpler model. Since all updates originate from the primary via zone transfer, there is no need for a locking mechanism. Thus locking determinations such as field vs. record locks need not be considered as well as "race" and "deadlock" issues.

Primary/Secondary servers have a drawback in that should the Primary server fail for any reason such as equipment failure, database corruption, application or operating system failure, the determination of reliable data on Secondary servers may be a problem. The method that practically eliminates the problem of the Primary server failure is a high enough "Time-To-Live" (TTL) value coupled with reasonably reliable recovery procedures. TTL is a parameter used in the DNS to designate how long a retriever of information may consider the information reasonably reliable. TTL is discussed in detail elsewhere.

The recovery procedures would normally include daily backups and the ability to use alternative hardware or a responsive hardware maintenance contract. The idea here is to have a TTL value in the range of days and the recovery of the Primary server to occur within less than a day. This enables the Secondary servers to carry the load while the Primary server is being fixed. The mechanism of the zone transfer provides for flexibility, even if the Primary is inoperative for a somewhat extended period of time. The system can be set up so that a zone database on a Secondary may be kept for a period of time and considered a reliable database, even though a zone transfer cannot be performed from the Primary for an extended period. With equipment and software becoming more and more reliable, the dependence on a Primary server that may not be available for extended periods of time (days) is a less significant risk factor in providing overall system reliability.

Primary and Secondary servers are both considered "authoritative", meaning the database maintained on the Secondary servers typically have a high degree of reliability. The "authoritative" distinction, by definition, indicates a "local", typically complete, database for the zone.[1] Non-authoritative servers are more commonly dependent on cached and remote information.

In the DNS, the two primary mechanisms of data base redundancy are the Time-To-Live (TTL) field and the Zone Transfer function. Secondary servers are loaded via the Zone Transfer function.

Zone Transfer

[1] Elz - 2

"Zone Transfer" is the mechanism used to copy a section or complete database from a Primary server to a Secondary server. A Primary Server contains all the information for a zone and/or domain plus a list of Secondary servers by IP number. A Secondary server is provided the name of the Primary Server(s) that the Secondary is to get a database transfer from. For the Secondary server to obtain the information that it must provide to client requests, the Secondary requests a zone transfer from a Primary when the Secondary initially starts up, then the Secondary will request an additional zone transfer each TTL interval (discussed below). When the database is transferred, the Primary includes the TTL information in the transfer of the zone information. Providing the TTL information enables the Primary indirectly to manage when the Secondary requests a zone transfer, though the transfer request originates from the Secondary. All requested records are copied from the Primary to a Secondary via the same means that the inquiry mechanism uses. The only differences between a zone transfer and an inquiry in the DNS are the protocol used and certain performance considerations.

The protocol used for an inquiry is the UDP protocol, characterized by less overhead. Zone transfers use a TCP protocol, characterized by reliability. The UDP protocol has less overhead due to the lack of verification mechanisms, thus the lowered reliability as compared with TCP. The verification mechanisms in TCP and intentionally lacking in UDP define that the sender must get an acknowledgement that the receiver has obtained data exactly as sent. Though this may initially seem backwards that inquiry activity uses UDP and Zone transfer requires TCP, some understanding of the how clients, Primary and Secondary servers work together will clarify the protocol issue.

Client software is typically set up with the assumption that if there was no response to a certain DNS server within a certain time period (usually seconds), there are one or more alternative DNS servers that can be queried. Rather than depending on reliable information from a single source via TCP, since multiple sources are known and each is generally as reliable as the other, the client can move on to an alternate server using UDP once again, generally resulting in better overall response time than TCP to a single server. The Zone transfer, on the other hand, requires reliable information to be assured so that the client inquiry assumption about reliable alternatives can be made. The Zone transfer also requires an extended period of time, relatively high volume of information, and lower priority / less intrusion. These requirements make UDP unworkable from the Zone transfer function.

The simple DNS query can typically be accommodated in a single packet, where a Zone transfer normally requires a large number of packets. Simply because there is more data for a Zone, the transfer will take longer. Transmission error rates increase in proportion to the volume of data as well as the amount of time it takes to make the transfer. Exacerbating the problem are the criteria implied by RFC 1034 [Mockapetris-4] that inquiries made upon a DNS server must not be made less responsive by Zone transfers in progress. Thus should a server find itself frequently queried, the server may find that Zone transfer activity might have to suffer intermittent delays to maintain responsiveness for queries, thus subjecting the transfer to additional transmission problems.

## Time-To-Live and Timing Issues

Though the TTL field is also used in caching data on caching servers, this project focuses on the TTL field only as it is associated with complete sets of redundant data on Secondary servers. Record layouts for DNS provide for a TTL field due to the DNS caching mechanisms; the TTL field of interest for zone transfers is associated with the authority information. The TTL field of the authority record is applied in practice to all of the records in the associated zone, though each record has the ability to have a separate TTL field.

The primary mechanism of the "Time-to-live" (TTL) field and function provides for the weeding out of outdated data. When zones are transferred, the TTL field indicates to the receiver that the data can only be referred to for a limited time. For performance considerations, the TTL field can be, and often is, several days. Thus the receiver is instructed to "use" the data for only a limited time, then request a new set of data, a new *zone transfer*. Some of the data might change during the TTL interval from the Primary server; thus the data on the receiver is updated when a new zone transfer occurs.

Instead of ensuring identical data in Secondary databases, the TTL field provides a functional opposite: a length of time that data (assuming eventual outdated data) can be permitted to remain on the Secondary system. This approach to duplication of data would appear to be rather unique. Most database algorithms and systems are concerned with keeping and maintaining data in a reliable and "good" data state; the approach with the DNS is nearly to assume that the data will be bad (at least eventually) and thus discard it. Applying this method or assumption to financial or inventory data would almost certainly be disastrous. This assumption can work where data is not absolutely required to be accurate for a system to function and yet the system reporting some data, not necessarily accurate data, is needed to produce a responsive system.

### Other Distributed Database Features

Other noteworthy features of the DNS distributed database are briefly described here though only the basic Primary / Secondary server facility is implemented in the PH project since only this feature applies to the primary fault-tolerant capability of DNS.

#### Caching Servers

The caching mechanism mentioned briefly in earlier discussions provides for more than performance benefits. The same TTL mechanism applies to caching data. If Primary and/or Secondary servers fail, there may be some caching information in a caching server for some period of time that provides limited use of the DNS. The function is slightly different on a caching server than a Secondary server.

A caching server has a primary purpose of providing better performance on the DNS. Performance is obtained by being a medium for the DNS mechanism and, in the process, "holding" information with the hope that the information will be requested soon again. The caching function works well with the nature of the DNS data involved. Despite the continual change in the structure of the Internet, from the general user's perspective the nature of the bulk of the DNS data is mostly static. Thus propagating static data to caching servers is acceptable for the boost in performance that can be obtained by using the caching servers. The caching server mechanism in fact provides a significant boost to the performance of the DNS by reducing the necessity of requesting information out on the Internet by 80% and more.

DNS Hierarchy Methods

The following notations are used in the explanation of hierarchy methods:

$D_s$ – the current domain level in the hierarchy – the "subject" level.

$D_{s+1}$ – the next level up from the subject level

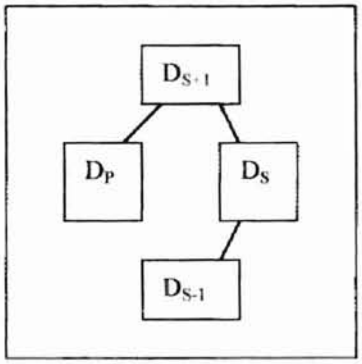
$D_{s-1}$ – the next level down from the subject level.

$D_{s+}$ – one or more levels up from the subject level

$D_{s-}$ – one or more levels down from the subject level

The mechanics of the DNS hierarchy are centered on the DNS servers. While the structure is useful from an administrative standpoint due to the distributed hierarchy, the structure itself is not inherently fault-tolerant. Since the structure is not inherently fault-tolerant, mechanisms have been built in to provide for fault-tolerance. The requirement of a Primary and at least one Secondary name server by the registering agent ($D_s$) gives the higher level domain($D_{s+1}$) the ability to provide fault tolerant features to requests made of domains that are for subordinate domains ($D_s$ and any peer of $D_s$). The encouragement of multiple "glue" pointer records placed in the subject DNS servers ($D_s$) to higher level domains ($D_{s+}$) provides the inverse redundancy features. "Glue" pointer records are "non-authoritative" references to additional DNS name servers (at $D_{s+}$) that are up one or more levels in the DNS hierarchy. These pointer records (at $D_s$) are "non-authoritative" because they are not "controlled" or "managed" in the subject domain ($D_s$).

Even though the "glue" pointer records are "non-authoritative", the frequency of change of Primary and Secondary name servers (in $D_{s+}$) is less frequent as the hierarchy is traversed toward the apex. In other words, the glue records (in $D_s$ pointing to $D_{s+}$) are generally as reliable as the addresses of the Primary and Secondary DNS name servers of the containing domain ($D_s$) since these server addresses are typically registered with the organization ($D_{s+1}$) that has the DNS servers that the pointer "glue" records are referencing (contained in $D_s$ referencing $D_{s+1}$).



DNS Hierarchy

# IV. PROJECT

## General Plan

The following steps were planned to evaluate applying DNS fault tolerance via a redundant server to the PH system.

- Complete an evaluation of current systems, software, engineering, and methods that might best be applied to the PH system with primary emphasis on the DNS.
- Apply system enhancements to the PH system that provide for similar or same redundancy features as in the DNS.
- Implement the new features at the University of Tulsa.

The project was to be considered complete either 1) when the redundancy features are successfully applied to a production PH system, or 2) when it is determined that the features are not useable or otherwise viable.

## Methods Used

### Zone Transfer and TTL

The most prominent redundant feature of the DNS is Primary and Secondary servers. Specifically, the main redundancy facility of DNS is provided by one or more Secondary servers operating on physically separate machines from the Primary server. The Secondary server periodically downloads the entire database from the Primary server. This provides the "client" system with the capability of having more than one server to query. Thus if one server machine, say the Primary machine, is inoperative for any reason, the client can automatically make requests of the Secondary after a preconfigured timeout period on the client is reached. The 2 key features of DNS that provide the bulk of the redundancy capabilities are 1) zone transfer and 2) the time-to-live data characteristic. These features and other related features have been applied successfully to the PH system.

The method of implementation requires the PH system to change from a 2 tier "client-server" system to a 3 tier system with at least one Primary and one Secondary. For the purposes of this project, only one Secondary is provided for and implemented, though it would be a relatively simple matter to enhance the software for multiple secondaries. In addition, the DNS provides for a "time-to-live" capability for each record and/or each zone. Since the TTL feature appears to be more for caching servers rather than zone servers, I have chosen to provide TTL for zone data only at this point. Again, it would appear that with relatively minor software engineering, TTL for each record could also be provided.

The "siteinfo" command of the QI server provides some general information about the characteristics of the PH server. This command was enhanced by this project to provide TTL, retry intervals, and the serial revision numbering method as is used by the DNS model. This same information is transferred by the PH zone transfer at the beginning and end of the zone transfer for the same purposes as the DNS system provides. The equivalent information in the DNS system is termed the "start-of-authority" or SOA record. The purpose of transmitting the SOA information at the beginning and end of a zone transfer in the DNS is to provide a reference point to identify the beginning and ending of the transfer, to provide certain general information about the zone, and to provide to the Secondary a method to check whether the zone information changed during the transmission. Changed information is signaled by the change of the serial number in the DNS. The PH enhancements model this behavior by providing the equivalent SOA information via the "siteinfo" command and at the beginning and end of the PH zone transfer.

### Data Modification

The nature of the PH data, and one of the most popular characteristics of the PH system, provides for a user to change much or all of the data that refers to that individual user. This characteristic is very much different than the DNS where typically a single or a few administrators manage the changes of all of the information on a set of DNS servers. Users do not generally assign themselves IP names and numbers, whereas in the PH system they themselves can change information relating to themselves since most of the information falls in the "personal" realm, i.e. address, phone number, etc. The popularity of this dynamic characteristic provides the advantage to the user to have complete control over their own information without interference from another administrator often requiring forms and signatures to change information plus the exposure of the change to other individuals. Placing the change dynamics in the hands of the user without interference or approval from another party provides the user with control over accuracy and responsibility of the data and eliminates the administrative overhead of maintaining the data.

This ability for users to change their own data imposes some additional dynamics that are not characteristic of the DNS. Even though the dynamics are different, the same mechanisms of the DNS were modeled. A description of the 2 situations where an impact of these characteristics might come to bear follows.

In the DNS, data changes can be grouped and changed at one time on the Primary server by using a serial number incrementing procedure, incrementing the serial number for all changes in a group, then loading the new database. This administrative convenience can not be provided for in the PH system since the data can be changed by the user and the user would not normally be concerned with administrative efficiencies of grouping applied changes. The net result is that for each PH data change by the user, there must be some mechanism provided that can identify to Secondary servers that the data has changed. Since there are significant dynamics of the DNS serial number, this serial number mechanism was modeled with only one change, the serial number changes with each and every change to the Primary database.

The only other characteristic of the DNS model that produces a potential of weakness in the PH system is centered around the same dynamic data change permitted by the users. If the user changes data, is this change made directly to the Primary server or to all servers? For simplicity of this project and due to more recent dynamic developments to the DNS, I have restricted changes to be made only to the Primary server. The DNS provides for the update mechanism to Secondary servers, and even though changes to PH data can be more frequent, the PH data is still mostly static, i.e. people don't change their "personal" information very often. Recent developments in the DNS [Vixie-2; Vixie-1, Ohta-1] provide for "incremental" zone transfers and signaling to Secondary servers to notify them that they need to check for a zone transfer rather than waiting for a TTL timeout. These methods provide quicker, more efficient, and more dynamic updates to Secondary servers while still in the original DNS framework. These update mechanisms, while useful and helpful from an efficiency perspective are not needed to produce basic fault-tolerance via redundancy. The focus on this project was on the modeling of the fault-tolerance features via server redundancy rather than the efficient update and proliferation of updated data. Provided that there is some mechanism to deal with managing the update of data, there is no pressing need to enhance the mechanism that updates the data. Modeling the DNS provides a minimal update mechanism.

### Serial Number System

The DNS serial number system provides a single point of control and management for database updates. Though a high update volume system would likely not use a single point update mechanism due to performance considerations, the single point update mechanism is adequate for those databases that inherently have low update volume such as the DNS and PH systems.

Specifically the serial number mechanism in the DNS establishes a data element for the database, the serial number, which is manually incremented when an administrator makes changes to the DNS database on the Primary server. When a Secondary DNS server prepares to request a zone transfer from the Primary server, the Secondary checks the serial number and, should the serial number from the Primary contain a "lesser" or equal number than the last serial number that the Secondary had from a previous zone transfer, the Secondary assumes that there have been no changes on the Primary and does not request a zone transfer. If the serial number from the Primary is "greater" than the serial number from the previous zone transfer, the Secondary assumes that the database has changed and requests a zone transfer. There are other details of the serial number including determining "lesser" and "greater" relative serial numbers due to "wrap around" (since machines are finite) as well as considerations for resetting the serial number when needed, but this material is not directly relevant to the fault-tolerance redundancy mechanism. The additional descriptions are provided in RFCs 1982 [Elz-1], 1034 [Mockapetris-4], and 1035 [Mockapetris-5].

## Enhancements

### General Enhancements

Prior to this project, primary and alternate servers had already been provided for in the client configuration, though not provided for in an "on-line" form on the server side. As an aside, based on the Primary/Alternate server characteristic of the client software, it would appear that at some time some thought was given to the PH system to enhance it along similar or the same lines as this project has done.

Project enhancements providing Primary/Secondary information via "siteinfo" provides the server with the ability to sense in what tier it should function while retaining the configuration and flexibility of the original code.

Scripts and separate utilities provided with the original distribution to build the database were incorporated into the Secondary server to rebuild downloaded data bases automatically. Early on in the project, a complication appeared to develop in this area involving the final "switching" between the old and new database once the zone transfer was complete and the new database was built. This was handled by maintaining two sets of directory structures and file sets for the database such that database "builds" would work with the currently unused structure and files. Once the new database was completely built, a link used to join the top level of the structure would be switched to point from the "old" database to the "new" database. This one step switch eliminates most uncertainty with the integrity of the database by the simple fact that only one step is involved. Any steps up to the "switch" would not render the current database unusable, thus the server can continue serving.

The "zone transfer" feature was a relatively simple enhancement to QI since all the server commands have been implemented via a parsing function written in Lex. This command enhancement simplicity was fortunate since the debugging and engineering tools for software written in Lex appear non-existent.

Another distinction between the DNS and the PH system requires the PH system to ensure a higher level of security for the database. The DNS for the most part does not contain private information. If anything, accesses to the DNS distributed databases and their data in its entirety are encouraged in the design and use of the DNS. While access to individual records is encouraged in the PH system, access to the bulk of the PH database en masse is not normally encouraged for several reasons. In fact, the PH system has several safeguards built in to prevent copying the database in total. Thus while a zone transfer request from an "abnormal" or "unrecognized" agent in the DNS is not common, it is none-the-less accepted and can prove to be a handy tool for, among other purposes, identifying and correcting DNS database problems. On the other hand, the PH system, due to the sensitive nature of the reasonably personal information contained in the database, must have tighter controls and must rely on some form of authentication mechanism to prevent the copying of the database en masse. To prevent unauthorized copying, a validation process was added to the PH Primary server that verifies that zone transfer requests originate from the recognized Secondary server.

The method chosen to reduce the risk of unauthorized copying of the database was to validate the requesting IP name/number on the Primary server against the configuration parameters specified when the Primary server was generated. Though this method is not foolproof, it significantly reduces the possibility of unauthorized copying by ensuring that the Secondary server is identified. Attempts to issue either a false name or IP number to the Primary are detected by the Primary. Even if the IP name and number are falsified, responses will be only sent to a valid Secondary. Thus, in a non-trivial network, which is most likely subjected to unauthorized attempts, routers will not return information requested to the "spoofing" requester, and the real Secondary server will ignore unsolicited responses from the Primary. For this process to work well, one must be reasonably confident that the Primary and Secondary servers are securely accessible, i.e. no unauthorized individuals (or programs) can access the servers with a privileged account.

All the fundamental data communications capability was fully featured and in no need of any change. The PH system was entirely written using TCP without any features using UDP so there was no need to make any changes in this area. The DNS client queries were specified to use UDP, and the zone transfers: TCP. The client software for DNS had the luxury from the beginning to use UDP since alternate servers were designed in from the beginning, thus a client could query a secondary when an initial UDP query failed, and the client engineers were encouraged in this direction by specification. For efficiency, the PH system might be further enhanced to use UDP for queries now that the PH system has a multiple server structure.

## Specific Enhancements

The following was used as an outline for the steps in the project:

- Update Linux PH install configuration file
- Data base of recognized zones – siteinfo – include data timeout and retry interval
- Add zone transfer command with age of data timeout and runtime range
- Incorporate "dump" to capture and rebuild the database on the Secondary
- Add zone request based on age of last zone
    - "On-line" copy to temp file for build
    - Use scripts to build new database in alternate database area
    - Change "link" pointer reference to new database

The "siteinfo" command of the QI server was modified to include the server IP names and numbers, the TTL information, and the database revision serial #.

The database field modification code was changed to update the revision serial # on the Primary server so that zone updates can be properly signaled. The Secondary server was modified to operate the database in a read-only mode to prevent inadvertent updates to the Secondary database that would be lost with a zone transfer. Ensuring updates are performed on the Primary server is a necessity and a difference of the DNS system. In the DNS system, typically only one administrator or administrative group is responsible for maintaining the IP names, numbers and other data contained in a particular domain, but in the PH system the data is typically maintainable by the user. In the DNS (hopefully) knowledgeable administrators maintain the system, where, in the PH system, an end-user that is often less knowledgeable of the inner workings of the system is typically maintaining their own data. Making the Secondary server read-only ensures that updates provided by the user go through the Primary server.

- update Linux PH install configuration file

At the time of this project, there was evidence that a configuration for Linux existed, but, upon examination of the configuration, there were many source changes that might prevent the Linux PH version from being used on other systems. Thus the task of providing a more flexible configuration file was completed with only several bug fixes to the original PH source.

- data base of recognized zones – siteinfo – include data timeout and retry interval

Prior to the zone transfer, a "siteinfo" command is sent to the Primary server to determine if the serial number is different. If the serial number has not changed from the last zone transfer request, no zone transfer is necessary. This step is performed to prevent a "blind" zone transfer from unnecessarily taking up network bandwidth.

- add zone transfer command with age of data timeout and runtime range

The zone transfer command was set up to take the data received from the transfer port and place the data in a "flat" file in "dump" format to be processed in the next step of creating and loading a database.

Another feature of the DNS that was incorporated into the PH zone transfer was the secondary check on serial number changes immediately after a zone transfer has completed. This second check for serial number changes is intended to detect possible changes in the Primary database that occurred while the zone transfer was going on. Should the serial number change be detected during the zone transfer, the zone information is effectively discarded and another request for a zone transfer is initiated by the Secondary server. There was no consideration as part of this project to put a limitation on this "changed serial" loop. As is the case with the DNS, the PH database is unlikely to change with sufficient frequency to warrant concern for the additional code needed to detect, prevent, and/or give notice of this type of potential looping problem. The PH database has a higher probability of frequency of change, but the frequency of change remains insignificant to create a looping problem with the zone transfer mechanism.

- incorporate "dump" to capture and rebuild the database on the Secondary

Portions of several scripts provided with the PH distribution were incorporated into the Secondary server code. Some of the scripts used contained the utilities for creating and loading the database from the temporary "dump" format flat file.

- add zone request based on age of last zone

Timing information provided by the siteinfo command determines the next zone transfer attempt that the Secondary server makes. The Secondary adds the delta time obtained by siteinfo to the current local time clock checked when the completion of the last successful zone transfer occurs and registers a time initiated process for the next zone transfer based on this computed absolute time.

- Copy to temp file for build

To attempt to reduce wait time for the zone transfer mechanism, the Secondary places the zone information in a flat file in a temporary area for processing once the zone transfer is successfully complete.

- Build new database in alternate area

Use the PH utilities to generate a new database from the flat file of the zone transfer, placing the new database, during the build, in an alternate area, so as not to interfere with production database activity on the server.

- Change "link" pointer reference to new database

The mechanism used is that known in the Unix world of a "soft link" as a directory. Once a valid new database is created, the "soft link" pointing to the previous database files is redirected to the new database files. Using this method also eliminates the need for any further coding changes, thus keeping the PH system as simple as possible.

With a zone transfer request, the Primary server functions nearly identically to that of the DNS from a procedural standpoint. The only significant exception to the procedure is due to the more sensitive nature of the PH information. The Secondary server is validated against identifying information generated when the Secondary server system was created (non-dynamic absolute information). While the DNS will process a zone transfer from any requester, designated Secondary server or otherwise, the PH will not process a zone transfer from any location other than the Secondary, as this defeats the purpose of the PH mechanisms to limit the volume of information provided.

```
Zone transfer request?
        Valid alternate?
                Send all info
```

Pseudo code for developing the zone transfer for the Primary Server

```
Start
        Zone processing

Zone transfer timeout interval met
        Zone processing

Zone processing
        Get info on valid data timeout and retry zone request interval – siteinfo
        Send zone request

        Server down or not responding or ?
                Resend zone request each "retry" minutes
        Write data to work area
        Spawn rebuild
        Redirect link to put production database in place
        Set up zone processing on timeout
```

Pseudo code for developing the zone transfer for the Secondary server

## Evaluation

While the project could not be implemented at TU due to circumstances beyond my control, a series of controlled tests were performed to emulate conditions and situations that would occur in practice. All code enhancements were verified to work using debugging tools provided in Linux. At least one case was tested for each additional logic branch added. While all existing features could not be tested in the test environment obtained, all new modules and features were tested thoroughly. To emulate rapid multiple updates on the primary server and thus to ensure that critical sections of the serial number update were being properly maintained, code was placed in the locking mechanism temporarily to cause the program to wait at key points.

While it is unlikely that all errors have been eliminated, using controlled and modular tests, most of the errors introduced by the new coding have been eliminated.

In addition to testing each new logic path, the following controlled tests were performed with both servers operating:

- Turned off primary server – client switched to secondary
- Turned off secondary server – no impact on client
- Disabled communications to primary server – client switched to secondary
    - Enabled communications to primary server – client switched back to primary at next request
- Disabled communications to secondary server - no impact on client
- Attempted an update on secondary server – client prevented
- Disabled communications to primary server
- Disabled primary server – secondary server requested and retried zone transfers at correct intervals
    - Enabled primary server – secondary server requested zone transfer
- Requested zone transfer from an unrecognized secondary server – obtained error message
- Made 2 updates to the primary server database (wait code placed in lock module to test critical section) – serial number incremented twice
- Made a change during a zone transfer (with lock code delay) – change accepted and zone transfer properly disposed by secondary server, secondary server requested another zone transfer

- Disabled communications during a zone transfer – secondary server properly timed out and requested another zone transfer
- Test coded errors in several places of zone transfer processing on secondary server to ensure that current secondary database did not get inadvertently corrupted, or replaced

## Environment

As a spin-off benefit, since the project budget was rather low, Linux was chosen for the operating system. Linux is a public-domain Unix OS that can run on PC compatible machines of the 386 and up variety. Though there was some previous work by several individuals to produce a PH server on a Linux system, no known previous attempts made the effort to maintain the software such that it could be still used in other Unix OS flavors. The custom configuration file containing detailed instructions for Linux while maintaining the flexibility of the original code was posted with to the PH users listserv and placed in the public domain. I have since gotten positive feedback on this posting.

### Test System

TU graciously loaned me a 486/33 PC for the project. I spent about 1 month configuring the system with SlackWare Linux and the original QI version 3.1B7. Though there was evidence of another individual, Tim Lawless, implementing QI on Linux, it appeared that he greatly modified his distribution that he downloaded from UIUC. The extent of his modifications may prevent his version from working on other platforms. I was careful in my configuration to only make changes to specialty configuration files unless absolutely necessary. I have completely documented the changes and submitted them to the PH listserv since others were inquiring about the same capability.

After attempting to configure other borrowed 386 and 486 machines with no success, I purchased a second older Pentium machine, network cards and a small 10BaseT hub and configured a 2-machine network, with both running the SlackWare Linux. The 486 is functioning as the Primary server and the Pentium functions as both the Secondary server and the client for testing the effectiveness of the redundancy. The Primary server can be literally turned off to emulate a failure. Using such a test environment eliminates "side effect" problems with the Primary server since the production PH Primary server performs other production functions as well.

### Production System

I had intended on implementing the production system at my employer, the University of Tulsa, but having left their employ, and their high workload for implementing summer computer projects, they have, as yet, been unable to accommodate the new package. I hope to at least be able to place the tared production code on an ftp server that is publicly accessible.

# V. NET RESULT, CONCLUSION, AND PRACTICAL APPLICATIONS

Though the details and complexities of the PH system and the DNS have been discussed in detail here, the ability to add system redundancies with minimal efforts has been shown to be attainable. While one generally must have a fairly thorough understanding of a computer application to modify the application for real-time redundancy, the actual efforts of putting in the redundancy for most modern (i.e. structured / modular / object-oriented) software can be a relatively simple task, particularly when pursuing the more simple and elegant methods like those used in the DNS. The demand for this type of redundancy may grow as more businesses are opening up doors on the Internet.

With the rapid pace of change of computers, networks, the Internet and technology in general today, redundancy methods may become standard with little fanfare as we become more and more dependent on applications based on the Internet. Redundancy methods are not as glamorous as some form of graphic dynamic web page, but for a business that places 80% or more of the activity of that business on the Internet, a broken or inaccessible (i.e. a portion of the network is inoperative) Internet server with no alternative will demand the less-flashy redundancy features be put into place. Have you ever cornered in a car with bad shocks on a rough and wet road? Car shocks are a safety device to help keep tires in continuous contact with the ground. If the tire is permitted to bounce, no lateral force can be applied while the tire is off the ground. The net effect can be loss of control or worse. The redundancy that is encouraged here is analogous to the car shock. Redundant features add a safety factor to applications being visited on the Internet. In fact, one of the reasons that the Internet works as well as it does currently is due to the DNS with these features designed and built in. Applying these redundancy methodologies to other applications on the Internet is a natural progression.

## Use of Fault-Tolerance Strategies

Fault-tolerant strategies continue to gain in acceptance as computer and communication equipment evolves to cost less and become more powerful both in CPU speed and software features. The success of RAID systems is a classic example of the redundancy trend. As the Internet grows and develops, the dependency on single node failure will also need to evolve. Though RAID configurations are effective, RAID does not provide for node or central CPU failure. Newer software and hardware techniques are likely to evolve to provide fault-tolerance with the multi-node characteristics of the Internet. Though it's hard to say in what direction the evolution will take since the computer and communications business is in such a fluctuating state, the DNS and now the PH system redundancy modifications may be considered a step in Internet fault-tolerance evolution.

## DNS as Model

The DNS has evolved since its inception. Each evolutionary step has provided more features and robustness. The modular architecture of the system has provided a foundation for the system to evolve gracefully. Considering the rapid development of the Internet with an amazing growth curve, the DNS represents a monument to system engineering as a significant part of the foundation of the Internet. With the most wide spread node-level fault-tolerant features of any system in the world, the DNS gives us a successful model from which to develop further fault-tolerant applications.

## PH Benefit

The PH fault-tolerant features that have been added as a result of this project are anticipated to provide more reliability to the PH systems worldwide. Due to the nature of freeware and the Internet, quantifying the number of users and thereby the direct effect of this benefit is difficult if not impossible. Having some feedback from putting the Linux PH configuration file on the Internet is encouraging in the sense that the PH base is still expanding, though there has been some interest in the specifications of Lightweight Directory Access Protocol (LDAP) which is a standard that, theoretically, is to take the place of the PH system. Because the PH system is so widely used and accepted due to the generic C code that adapts to many platforms, the newer LDAP standard has been slow to be accepted.

PH users can now shut down the Primary PH server with little or no direct effect on the user base. E-mail can be forwarded by the Secondary PH server with no interruption. Both "batch" and online updates of the PH database are prevented while the Primary PH server is down, but "batch" updates are done late at night and can be re-run. Online updates are relatively infrequent; thus they have little to no direct impact on the user base. The flexibility to bring down individual servers for software updates, hardware and firmware updates, while online systems are still available for the most part, is an invaluable advantage since it is becoming harder to schedule downtime on the Internet.

Recall the phrase "the Sun never sets on the British Empire". This expression is true of the Internet for the same reason. While it may be the middle of the night for local users, there are users and potential users on the other side of the globe that are in the middle of their workday. Being able to shut down part of a system for maintenance while keeping the system running becomes a valuable feature indeed. Usable redundant systems have thus been found to be helpful for reasons other than just disaster prevention, but for maintenance issues as well.

Finally, though it was found that a thorough knowledge of the application must be obtained to produce an effective redundant server capability, the modifications were nevertheless rather simple and more a matter of restructuring existing code rather than providing a significant re-write. Generally, very little additional programming was added, but sections of code were regrouped and recollected to provide the functionality desired. Thus the expectation is such that in a modular, structured, and/or object-oriented environment, little actual changes in the form of additional programming code would need to be performed to obtain a redundant server benefit. The expectation of fairly simple changes on other similar projects needs to be tempered, though, with the equivalent evaluation criteria: the changes here were relatively simple due to a relatively static database and the equivalent notion in the DNS of "obtaining the data is more important than the accuracy of the data". More dynamic databases might have similar features incorporated, though dealing with the locking mechanism and maintaining updates would probably need to be refined somewhat.

In conclusion, the DNS methods of redundant servers have been found to be applicable to the PH system and it would appear to apply to any similar database application. These redundant servers would appear to be attractive to Internet database applications in particular due to the nature of business on the Internet needing to keep automation doors open 7 days per week and 24 hours per day.

# BIBLIOGRAPHY

Abrams, Marshall (1980), Role Mailboxes, Network Working Group, Request for Comments: 763, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc763.html

Andleigh, Prabhat K.; Gretzinger, Michael R. (1992), Distributed Object-Oriented Data-Systems Design, Prabhat K. Andleigh, Michael R. Gretzinger, Englewood Cliffs, N.J., PTR Prentice Hall.

Anklesaria, F.; McCahill, M.; Lindner, P.; Johnson, D.; Torrey, D.; Alberti, B. (1993), The Internet Gopher Protocol (a distributed document search and retrieval protocol), University of Minnesota, Network Working Group, Request for Comments: 1436, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc1436.html

Atre, Shaku, (1992), Distributed databases, cooperative processing, and networking, 1st ed. New York, McGraw-Hill.

Baum, Richard L; Cumbow, Robert C, First Use: Key Test in Internet Domain Disputes, ProQuest Periodical Abstracts, Journal: National Law Journal [INLJ] ISSN: 0162-7325 Jrnl Group: Business Vol: 18 Iss: 24 Date: Feb 12, 1996 p: C17-C18

Bell, David A.; Grimson, Jane (1992), Distributed database systems, Wokingham, Eng.; Reading, Mass., Addison-Wesley Pub. Co.

Braden, R. - Editor (1989), Requirements for Internet Hosts -- Application and Support, Internet Engineering Task Force, Request for Comments: 1123, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc1123.html

Braden, R. - Editor (1989), Requirements for Internet Hosts -- Communication Layers, Network Working Group, Internet Engineering Task Force, Request for Comments: 1122, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc1122.html

Cerf, V. (ARPA) and Postel, J. (ISI) (1980), Mail Transition Plan, Network Working Group, Request for Comments: 771, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc771.html

Clark, David D. (1982), MIT Laboratory for Computer Science, Computer Systems and Communications Group, IP Datagram Reassembly Algorithms, Request for Comments: 815, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc815.html

Crispin, Mark (1979), A Universal Host Table, Request for Comments: 752, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc752.html

Crocker, David H., The Rand Corporation, Vittal, John J., Bolt Beranek and Newman Inc., Pogran, Kenneth T., Massachusets Institute of Technology, Henderson, D. Austin Jr., Bolt Beranek and Newman Inc. (1977), Standard for the Format of ARPA Network Text Messages, Request for Comments: 751, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc751.html

Crocker, David H. (1982), Dept. of Electrical Engineering, University of Delaware, Newark, DE 19711, Standard for the Format of ARPA Internet Text Messages, Request for Comments: 822, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc822.html

Davis, C. - Kapor Enterprises; Vixie P. - Vixie Enterprises; Goodwin, T. - FORE Systems, Dickinson, I. - University of Warwick (1996), A Means for Expressing Location Information in the Domain Name System, Network Working Group, Request for Comments: 1876, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc1876.html

Deutsch, Debra P. (1979), Bolt Beranek and Newman, 50 Moulton Street, Cambridge, Massachusetts 02138, A Suggested Solution to the Naming, Addressing, and Delivery Problem for ARPANET Message Systems, Request for Comments: 757, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc757.html

Dorner, Steve (1989), Computer and Communications Services Office, University of Illinois at Urbana, Why?, Original distribution of QI version: qi-3.1B7 documentation, Internet url: ftp://uiarchive.cso.uiuc.edu/local/packages/ph/qi-3.1b7.tar.qz

Elvy, Marc A. - Harvard University, Nedved, Rudy - Carnegie-Mellon University (1984), Network Mail Path Service, Network Working Group, Request for Comments: 915, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc915.html

[Elz-1] Elz, R.; Bush, R. (1996), Serial Number Arithmetic, Network Working Group, Request for Comments: 1982, Internet url: ftp://ftp.is.co.za/rfc/rfc1982.txt

[Elz-2] Elz, R.; Bush, R.; Bradner, S.; Patton, M. (1997), Selection and Operation of Secondary DNS Servers, Request for Comments: 2182, Internet url: http://www.es.net/pub/rfcs/rfc2181.txt

Elz, R.; Bush, R. (1997), Clarifications to the DNS Specification, Network Working Group, Request for Comments: 2181, Internet url: http://www.es.net/pub/rfcs/rfc2181.txt

Efrat, Lior; Berkovich, Gai; Huttner, Hadas (1996), TCP/IP Overview, Technion Israel Institute of Technology, Internet url: http://techst02.technion.ac.il/~s2382639/

Gilbert, H. (1995), Introduction to TCP/IP, PC Lube and Tune, Internet url: http://pclt.cis.yale.edu/pclt/comm/tcpip.htm

Gressley, Christine, Research Programmer, Computing and Communications Services, University of Illinois at Urbana-Champaign, Kline, Charley, Senior Research Programmer, Computing and Communications Services, University of Illinois at Urbana-Champaign (1993), University Dormitory Ethernets: Network Security and Management in a Fundamentally Hostile Environment, Internet url: http://tampico.cso.uiuc.edu/~gressley/projects/urhnet/ univ-dorm-ethers/paper.html

Hage, Carl (1995), Re: Re[2]: E-Mail : "White Pages" Directories, carl@chage.com, Internet url: http://nic.nca.or.kr/gov-net/0074.html

Hardy, Henry Edward (1993), The history of the net -- Thesis, Master's Thesis, School of Communications, Grand Valley State University, Allendale, MI 49401, Internet url: ftp://umcc.umich.edu/pub/users/seraphim/doc/nethist8.txt

Harrenstein, K. (SRI), Stahl, M., Feinler, E. (1985), Hostname Protocol, Network Working Group, Request for Comments: 953, Obsoletes: RFC 811, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc953.html

Hedrick, Charles L. (1987), Introduction to the Internet Protocols, RUTGERS, The State University of New Jersey, Internet url: http://www.engr.usask.ca/~kia127/misc/tcp-ip-intro.html

Hedrick, Charles L. (1988), Introduction to Administration of an Internet-based Local Network, Rutgers University, The State University of New Jersey, Center for Computers and Information Services, Laboratory for Computer Science Research, Internet url: http://www.engr.usask.ca/~kia127/misc/tcp-ip-admin.html

Hoffman, Paul E. (1997), Perl 5 for Dummies, IDG Books Worldwide, Inc., Foster City, CA.

Horton, Mark S. (1983), Standard for Interchange of USENET Messages, Request for Comments: 850, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc850.html

Horton, Mark - Bell Labratories (1986), UUCP Mail Interchange Format Standard, Network Working Group, Request for Comments: 976, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc976.html

Huizer, E. (1995), Multimedia E-mail (MIME) User Agent checklist, SURFnet bv, Network Working Group, Request for Comments: 1844, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc1844.html

Hunter, Noel (1996), University of Pretoria FAQ PH CSO, Subject: PH (cso nameserver) Frequently Asked Questions (FAQ), Internet url: http://www.up.ac.za/about/faq/faqph/phfaq.html

Hurson, A.R.; Bright, M.W.; Pakzad, S. (1994), Multidatabase systems, an advanced solution for global information sharing, Los Alamitos, CA, IEEE, Computer Society Press.

Information Sciences Institute, University of Southern California, 4676 Admiralty Way, Marina del Rey, California 90291 (1981), Internet Protocol (IP), Editor: Postel, Jon, Request for Comments: 791, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc791.html

International Organization For Standardization, ISO/TC 97/SC 6, final Text of DIS 8473, Protocol for Providing the Connectionless-mode Network Service, Network Working Group, Request for Comments: 994, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc994.html

Kantor, Brian (U.C. San Diego), Lapsley, Phil (U.C. Berkley) (1986), Network News Transfer Protocol, A Proposed Standard for the Stream-Based Transmission of News, Network Working Group, Request for Comments: 977, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc977.html

Kelly, Al; Pohl, Ira (1995), A Book on C, The Benjamin Cummings Publishing Company, Inc., Redwood City, CA.

Killen, Michael (1992), SAA, managing distributed data, New York, McGraw-Hill.

Kolstad, Rob (1995), Connecting Corporations to the Internet, Berkeley Software Design, Inc., Internet url: http://www.bsdi.com/white-papers/connecting-corporations-kolstad.html

Krol, E. (1989), The Hitchhikers Guide to the Internet, University of Illinois Urbana, Network Working Group, Request for Comments: 1118, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc1118.html

[Kubaitis-1] Kubaitis, Ed (1998), Phonebook – Server Lookup, Internet url: http://www.uiuc.edu/cgi-bin/ph/lookup

LaQuey, Tracy L.; Ryder, Jeanne C. (1993), The Internet companion, a beginner's guide to global networking, Addison-Wesley.

Lebling, P. David (1978), Survey of FTP Mail and MLFL, Network Working Group, Request for Comments: 751, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc751.html

Mills, D. L. (1981), Internet Name Domains, Network Working Group, Request for Comments: 799, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc799.html

Mockapetris, P. (1983), Domain Names - Concepts and Facilities, Network Working Group, Request for Comments: 882, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc882.html

Mockapetris, P. (1983), Domain Names - Implementation and Specification, Network Working Group, Request for Comments: 883, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc883.html

Mockapetris, Paul, (1986) Domain System Changes and Observations, Network Working Group, Request for Comments: 973, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc973.html

[Mockapetris-4] Mockapetris, Paul, (1987) Domain Names - Concepts and Facilities, Network Working Group, Request for Comments: 1034, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc1034.html

[Mockapetris-5] Mockapetris, Paul (1987) Domain Names - Implementation and Specification, Internet Network Working Group, Request for Comments: 1035, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc1035.html

Moore, Keith (1994), white paper on replication/caching/authentication for IAB workshop, University of Tennessee, Internet url: http://www.apocalypse.org/pub/u/romkey/ii-workshop/white-papers/msg00012.html

National Bureau of Standards (1983), Specification for Message Format for Computer Based Message Systems, Federal Information Processing Standard (FIPS) Publication 98, Request for Comments: 841, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc841.html

National Bureau of Standards (1981), Institute for Computer Sciences and Technology, Proposed Federal Information Processing Standard, Specification for Message Format for Computer Based Message Systems, Network Working Group, Request for Comments: 806, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc806.html

Noel, Rick (1996), Data Warehouses, A look at where we are going, Renselaer Polytechnic Institute, Internet url: http://www.cs.rpi.edu/~noel/warehouse/warehouse.html

Noel, Rick (1996), Scale Up in Distributed Databases, A Key Design Goal for Distributed Systems, Renselaer Polytechnic Institute, Internet url: http://www.cs.rpi.edu/~noel/distr_scaleup/distributed.html

[Ohta-1] Ohta, M.(1996), Incremental Zone Transfer in DNS, Internet Network Working Group, Request for Comments: 1995, Internet url: http://www.es.net/pub/rfcs/rfc1995.txt

Ozsu, Tamer M.; Valduriez, Patrick (1991), Principles of distributed database systems, Englewood Cliffs, N.J., Prentice Hall.

Parker, Timothy Ph.D., et al. (1997), Slackware Linux, Third Edition, Sams Publishing, Indianapolis, IN.

Partridge, Craig - CSNET CIC BBN Laboratories Inc (1986), Mail Routing and the Domain System, Network Working Group, Request for Comments: 974, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc974.html

Perry, Greg (1994), Turbo C++ Programming in 12 Easy Lessons, Sams Publishing, Indianapolis, Indiana.

Pickens, John R., Feinler, Elizabeth J., Mathis, James E., The NIC Name Server -- A Datagram-Based Information Utility, Network Working Group, Request for Comments: 756, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc756.html

Postel, Jonathan B. (1979), Information Sciences Institute, University of Southern California, 4676 Admiralty Way, Marina del Rey, California 90291, Internet Message Protocol, Request for Comments: 753, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc753.html

Postel, J. (ISI) (1979), Out-of-Net Host Addresses for Mail, Request for Comments: 754, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc754.html

Postel, J. (1980), User Datagram Protocol, Request for Comments: 768, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc768.html

Postel, Jonathan B. (1982), Information Sciences Institute, University of Southern California, 4676 Admiralty Way, Marina del Rey, California 90291, Simple Mail Transfer Protocol, Request for Comments: 821, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc821.html

Postel, J. (1983), The Domain Names Plan and Schedule, Network Working Group, Request for Comments: 881, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc881.html

Postel, Jon (1984), Domain Name System Implementation Schedule - Revised, Network Working Group, Request for Comments: 921, Updates: RFC 897, 881, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc921.html

Postel, J., Reynolds, J (1984), Domain Requirements, Network Working Group, Request for Comments: 920, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc920.html

Postel, J., Reynolds, J. (ISI 1985), The File Transfer Protocol, Network Working Group, Request for Comments: 959, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc959.html

Reynolds, J.; Postel, J. (1987), Official Internet Protocols, Network Working Group, Request for Comments: 1011, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc1011.html

Reynolds, J. and Postel, J. (1987), The Request For Comments Reference Guide, Network Working Group, Request for Comments: 1000, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc1000.html

Rose, Marshall T. (Delaware) - Einar A. Stefferud (NMA) (1985), Proposed Standard for Message Encapsulation Network Working Group, Request for Comments: 934, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc934.html

Rose, Marshall T. - Department of Information and Computer Science, University of California, Irvine, Irvine, CA 92717 (1983), Proposed Standard for Message Header Munging, Request for Comments:886, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc886.html

Sluizer, Suzanne; Postel, Jonathan B. (1981), Information Sciences Institute, University of Southern California, 4676 Admiralty Way, Marina del Rey, California 90291, Request for Comments: 780, Mail Transfer Protocol, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc780.html

Su, Zaw-Sing (1982), A Distributed System for Internet Name Service, Network Working Group, Request for Comments: 830, Internet url: http://www.cis.ohio-state.edu/htbin/rfc/rfc830.html

Su, Zaw-Sing; Postel, Jon (1982), The Domain Naming Convention for Internet User Applications, Network Working Group, Request for Comments: 819, Internet url:

Su, Zaw-Sing; Postel, Jon (1982), The Domain Naming Convention for Internet User Applications, Network Working Group, Request for Comments: 819, Internet url

Vixie, P. (1996), A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY), Network Working Group, Request for Comments: 1996, Internet url: http://www.es.net/pub/rfcs/rfc1996.txt

Vixie, P.; Thomson, S.; Rekhter, Y.; Bound, J. (1997), Dynamic Updates in the Domain Name System (DNS UPDATE), Network Working Group, Request for Comments: 2136, Internet url: http://www.es.net/pub/rfcs/rfc2136.txt

# Glossary

"C" – A popular 3rd generation programming language that typically requires compiling to machine code and linking with other routines prior to execution.

CCSO – Computer and Communications Service Office – The group at the University of Illinois, Urbana-Champaign that supplies the public domain version of the PH software.

DEC – Digital Equipment Corporation – A large manufacturer of computer systems primarily of the "mini" computer class.

Domain Name – Normally a reference to a logical grouping of IP Names such that some portion of the right side of the names (to a "dot") are the same. Example: "centum.utulsa.edu" and "www.utulsa.edu" are members of the same "domain", namely "utulsa.edu".

DNS - Domain Name System – The distributed data base system whose primary function is to map IP names to IP numbers.

E-mail – Electronic mail – An electronic messaging system similar in function to it's paper counterpart. For the purposes of this writing, only Internet style E-mail. There are a number of private E-mail packages and services that may or may not connect to the Internet or an Internet style network.

FQDN – Fully-Qualified Domain Name – A Domain Name that typically includes a rightmost dot ".", and does not normally include enough information to specify a specific machine (though a FQDN can be a FQN – Fully Qualified Name of a machine).

IP Name – Typically, a name of a computer or automated device on the Internet or Internet type network usually consisting of alphabetic characters, periods "." (dots), hyphens, and numbers. Example: "centum.utulsa.edu"

IP Number – A special numeric value assigned to a computer or automated device on the Internet or Internet type network in the form of "nnn.nnn.nnn.nnn" where each "nnn" is normally in the range of 1 to 255 (0 has special meaning). The numeric values are designed to make routing TCP/IP packets relatively easy for automated data communications equipment.

MMI – Man-Machine Interface – The point of communication between a person and a machine, typically a computer.

Nameserver – Usually referring to the PH Nameserver.

Node – Generally, in reference to Internet style terminology, a single machine or device attached to the network.

PH Nameserver – Usually referring to the entire PH system, not to be confused with the Domain Name Server of the Domain Name System.

PH – The CCSO Phone Directory system - Sometimes referring specifically to the PH client software. The "server" system itself runs software referred to as QI for Query Interpreter.

QI – Query Interpreter – see PH

RFC – Request For Comments – in general, a document that was published in some manner over the Internet that encourages and solicits comments of review and critique.

SOA – "Start of Authority" – general information about a server including to what domain it applies.

TCP/IP – Transmission Control Protocol / Internet Protocol – a data communications protocol used widely on the Internet.

Time-To-Live – TTL – A field in the DNS system that provides an "expiration time" that the receiver of the record may assume that the data is reasonably reliable.

TTL – See Time-To-Live

URL – Uniform Resource Locator – a reference point to locate "web" based information

Zone – A range of IP Names/Numbers in a specific group.

# Appendix - A Collection of Various Relevant Portions of Code Changes

From commands.c:

```
/* */
/* DoZonexfr - do zone transfer - jmj -6/8/98 */
/* */

/* static void
/* DoZonexfr(arg)
/*     ARG *arg;
/* {
/* int      Progress = 0;
/* static char *Me; */        /* the name of this program */
/* extern int Quiet; */ /* already defined */         /* qi/qi.c */

/* #define LENGTH 80 */

char * tnl __P((char *));

static void
DoZonexfr(arg)
    ARG *arg;
{
int   Progress = 0;
/* static char *Me; */        /* the name of this program */
/* extern int Quiet; */ /* already defined */         /* qi/qi.c */


/* take out main(argc, argv) */
/*    int   argc; */
/*    char   **argv; */
/*{ */
      INT32 entry;
      QDIR  dirp;
      int   count;
      int   selected;
      extern struct dirhead DirHead;
      struct dumptype *dt;

      char  *colon;
      int   f;
      FDESC *fd;

      char   **p;

      /* when you're strange, no one remembers your name */
/*    Me = *argv; */

/* validate secondary ~~~~~ */

/* WhoAreYou() */ /* jmj - 5/23/99 - lifted from */
{
      struct sockaddr From;
```

```
       int    i, FromLen = sizeof (From);
       struct passwd *pwd = NULL;
       char  errorstr[MAXSTR];
       int    s = fileno(stdin);
       char *hostname = NULL;

       if (isatty(s)) {
            /* we'll just let this happen from the same host */
         }
       else {

#ifndef QI_ALT
         (void) sprintf(errorstr, "QI_ALT not defined - cannot zone
xfer");
         DoReply(LR_NOALT, errorstr);
         IssueMessage(LOG_NOTICE, errorstr);
         return;
#endif
         if (getpeername(s, &From, &FromLen) == 0) { /* got name */
           struct sockaddr_in *sin = (struct sockaddr_in *) (&From);
           struct hostent *hp = NULL;
           int on = 1;

       /* get name of connected client */
           hp = gethostbyaddr((char *)&sin->sin_addr, sizeof (sin-
>sin_addr), AF_INET);

           if (hp) {
       /*
        * Attempt to verify that we haven't been fooled by
        * someone in a remote net; look up the name and check
        * that this address corresponds to the name.
        */

             int HostNameLen = strlen (hp->h_name);
             char remotehost[2 * MAXHOSTNAMELEN + 1];

             hostname = strdup(hp->h_name);
             strncpy(remotehost, hp->h_name, sizeof(remotehost) - 1);
             remotehost[sizeof(remotehost) - 1] = 0;
             hp = gethostbyname(remotehost);
             if (hp == NULL) { /* didn't get a name */
               (void) sprintf(errorstr
               , "Couldn't look up address for %s", remotehost);
               DoReply(LR_NOADDR, errorstr);
               IssueMessage(LOG_NOTICE, errorstr);
               return;
               }
             else {       /* got a name */
             for (; ; hp->h_addr_list++) { /* scan addresses */
                 if (hp->h_addr_list[0] == NULL) {
                 (void) sprintf(errorstr,
                 "Host addr %s not listed for host %s",
#if defined(sparc) && __GNUC__ == 1
                 inet_ntoa(&sin->sin_addr),
#else
                 inet_ntoa(sin->sin_addr),
#endif
                hp->h_name);
                DoReply(LR_MISMATCH, errorstr);
                IssueMessage(LOG_NOTICE, errorstr);
                return;
                } /* endif end of list - no address match */
              if (!memcmp(hp->h_addr_list[0], (caddr_t)&sin->sin_addr,
                  sizeof(sin->sin_addr))) {
                /* we be jammin... */
                hostname = strdup(hp->h_name);
                break;
                } /* endif match on address */
              } /* for */
            /* scan names for QI_ALT name ~~~~~~~*/
            if (strcmp(QI_ALT, hp->h_name)) { /* name match? (no) */
              for (; ; hp->h_aliases++) { /* scan aliases */
                  if (hp->h_aliases[0] == NULL) {
                  (void) sprintf(errorstr,
                  "Host name %s not authorized, expecting: %s",
                    hp->h_name, QI_ALT);
                  DoReply(LR_MISMATCH, errorstr);
                  IssueMessage(LOG_NOTICE, errorstr);
                  return;
                  } /* endif end of list - no address match */
                if (!(strcmp(QI_ALT, hp->h_aliases[0]))) {
                  /* we be jammin... */
                  hostname = strdup(hp->h_name);
                  break;
                  } /* endif match on address */
                } /* for */
              } /* official name match */
            } /* got a name */
            }
          else { /* couldn't get host by address */
#if defined(sparc) && __GNUC__ == 1
            hostname = strdup((char *) inet_ntoa(&sin->sin_addr));
#else
            hostname = strdup((char *) inet_ntoa(sin->sin_addr));
#endif
            IssueMessage(LOG_NOTICE, "Couldn't find hostname for
address (%s)",
              hostname);
            DoReply(LR_NONAME, "No hostname found for IP address");
            return;
            } /* end if get host by address */
          }
        else { /* getpeername failed */
          (void) sprintf(errorstr,
            "DoZonexfr: getpeername failed, errno: %s",
strerror(errno));
          IssueMessage(LOG_NOTICE, "DoReply: fprintf: %s",
strerror(errno));
          DoReply(LR_NONAME, "No hostname found from getpeername.");
          return;
          } /* end if getpeername */
        } /* !isatty */
}
```

```
/* validate secondary ~~~~~ */

      Database = DATABASE;
      selected = count = 0;

      if (!GonnaZone("zone")) /* lock thing */
            return;

/* provide header info here - db modification count */
      fprintf(Output, "%d:1:1:%d:2:%d:3:%d:4:%d:5:%d\n", -LR_OK
            , DirHead.nents
            , DirHead.serial
            , DirHead.refresh
            , DirHead.retry
            , DirHead.expire );

      for (entry = 1; entry < DirHead.nents; entry++)
      {
            if (Progress && !(entry % 100))
                  fprintf(stderr, "%d/%d/%d\r", selected, entry,
DirHead.nents);
            if (dnext_ent(entry) && !ent_dead())
            {
                  (void) getdata(&dirp);

/* here's where we plug in the stuff to send it out the port */
/* jmj 6/25/98 */
      fprintf(Output, "%d:2:%d:%s", -LR_OK, F_UNIVID,
tnl(FINDVALUE(dirp, F_UNIVID)));
      for (; *dirp; dirp++)
      {
            f = atoi(*dirp);
            colon = (char *)strchr(*dirp, ':');
            if (colon == *dirp)
                  fprintf(stderr, "\"%s\" lacks key =%s=\n",
                        FINDVALUE(dirp, F_NAME), colon);
            else if (f != F_UNIVID && colon && colon[1])
            {
                  if (fd = FindFDI(f))
                  {
/*                        putchar('\t');
                        fputs(tnl(*dirp), stdout);
*/
/* use Output instead of stdout */

                        fputc('\t', Output);
                        fputs(tnl(*dirp), Output);
                  } else
                        fprintf(stderr, "\"%s\" unknown field %d\n",
                              FINDVALUE(dirp, F_NAME), f);
            }
      }
      fputc('\n', Output);

/* } */
```

```
/*
 * Free a dir structure (modified for mdump) jmj - stolen from mdump
 */
      if (dirp)
      {
            for (p = dirp; *p; p++)
                  free(*p);
            /*free(dirp);
            dirp = 0;*/
      }

            }
#ifdef DEBUG
            if (entry % 500 == 0)
                  fprintf(stderr, " %d/%d\r", entry, DirHead.nents);
#endif
      }

/* provide header info here - db modification count */
      fprintf(Output, "%d:1:1:%d:2:%d:3:%d:4:%d:5:%d\n", LR_OK
            , DirHead.nents
            , DirHead.serial
            , DirHead.refresh
            , DirHead.retry
            , DirHead.expire );

/* secondary can verify modification count is the same to make sure */
/* database did not change during xfr */

      Unlock("zone"); /* now we unlock */

/*    exit(0); */

/*    fprintf(Output, "%d:Ok.", LR_OK); */

      return;
}

/*
 * replace tabs and newlines with escaped equivalents
 */
char  *
tnl(s)
      char  *s;
{
      register char *cp;
      static char value[8192];
      register char *vp;

      vp = value;
      for (cp = s; *cp; cp++)
            if (*cp == '\t')
            {
                  *vp++ = '\\';
                  *vp++ = 't';
            } else if (*cp == '\n')
```

```
                {
                        *vp++ = '\\';
                        *vp++ = 'n';
                } else if (*cp == '\\')
                {
                        *vp++ = '\\';
                        *vp++ = '\\';
                } else
                        *vp++ = *cp;
        *vp = '\0';
        return (value);
}
```

From dbm.c:

```
void
set_dir_head(serial, refresh, retry, expire)
  int serial, refresh, retry, expire;
{
  DirHead.serial = serial;
  DirHead.refresh = refresh;
  DirHead.retry = retry;
  DirHead.expire = expire;
}

void
incr_dir_head_serial()  /* to determine if a zone has been modified */
{
      if (DirHead.serial >= INT_MAX)
      {
            DirHead.serial = INT_MIN;
      }
      else
      {
            DirHead.serial++;
      }
}

void
put_dir_head()
{
      if (lseek(dir_fd, 0, 0) < 0)
      {
            IssueMessage(LOG_ERR, "put_dir_head: lseek(%d,0L,0): %s",
                  dir_fd, strerror(errno));
      }
      if (write(dir_fd, &DirHead, sizeof (DirHead)) < 0)
      {
            IssueMessage(LOG_ERR, "put_dir_head: write: %s",
strerror(errno));
      }
      Have_head = 0;
}

void
get_dir_head()
{
      if (lseek(dir_fd, 0, 0) < 0)
      {
            IssueMessage(LOG_ERR, "get_dir_head: lseek(%d,0L,0): %s",
                  dir_fd, strerror(errno));
            return;
      }
      if (read(dir_fd, &DirHead, sizeof (DirHead)) < 0)
      {
            IssueMessage(LOG_ERR, "get_dir_head: read: %s",
strerror(errno));
            return;
      }
```

```
        Have_head = 1;
}
```

From configs/defaults:

```
# jmj - 4/6/99 - primary and secondary servers
# be sure to set these to your primary and secondary server names
$QiHost = "node.domain.xxx";        # primary qi server
$QiAlt  = ".node1.domain.xxx";            # secondary qi server
                              # take out the leading dot in your
                              # version to have an active zone
                              # refresh version working
#
%DefineStrings=(
"DATABASE", "$SrcDir/../db/prod",   # Where the database lives, by
default
"MAILDOMAIN",     "uiuc.edu",       # Mail domain for phquery
"MAILFIELD",      "alias",          # Field returned by siteinfo for
email
# jmj - 8/17/97 - changed to single quote to prevent attempt to
translate
"ADMIN",    'p-pomes@uiuc.edu',     # Database administrator
"PASSW",    "CCSO Resource Center", # Mail here for passwords
(deprecated)
"MAILBOX",  "email",          # Use email field for forwarding
"RUNDIR",   "$SrcDir/qi",           # Where QI should chdir() to at
run-time
"HELPDIR",  "$SrcDir/help",         # Where the help lives
"NATIVESUBDIR","native",            # non-client-specific help
"NOHELP",   "nohelp",         # file printed when no help topic
"TEMPFILE", "/tmp/qiXXXXXX",  # temporary file template
"QI_HOST",  "$QiHost",        # FQDN of primary server for phquery
"QI_ALT",   "$QiAlt",         # FQDN of backup server for phquery
"QI_TIMEOUT",     '22*60*60',       # duration of valid database -
seconds
                              # timeout deprecated with secondary
"QI_RUNTIME",   '02:00-04:00',            # runtime "window"
"QI_INPUT", "$SrcDir/../qi.input",  # Where to put secondary rebuild
file
"REBUILD",  "$SrcDir/secondary/rebuild",  # Database rebuild script for
secondary
"QIEXECDIR",      "$QiExecDir",             # Secondary resubmits itself
"DEF_PORT", 105,              # Default port
"SERIAL",   1,                # Increments as database changes
"REFRESH",  3600,             # Check for new zone
"RETRY",    900,              # Check if network problem
"EXPIRE",   86400,                  # Thow away after...
);

# These will be used as normal defines.  The keys will be converted to
# all upper-case.
#
# QueryLimit      Limits the maximum number of entries of any type that
can
#           be returned in a single query.  (Like PersonLimit, but for
#           all types, not just type:p*).
%OtherDefines=(
  "Log_Qilog",    "LOG_LOCAL0",             # Syslog facility
  "Sig_Type",     "void",                   # signal() return type
  "Mem_Type",      "char",                      # *malloc() return type
  "INT32",  "long",                       # 32-bit integer type
  "Cpu_Limit",     "20",                  # limit for CPU time
  "Drecsize",      400,                   # size of .dir record
  "Dovrsize",      400,                   # size of .dov record
  "Max_Key_len",16,                       # max # chars to index
  "NIChars",       32,                    # size of .idx record
  "NOChars",       1024,                  # size of .iov record
  "PersonLimit","25",                     # max # of people to return
#  "Use_Uid",      "10",                  # setuid to this
#  "Use_Gid",      "15",                  # setgid to this
);
#
# jmj - 2/16/99 - secondary parameters
#
$DbaD = "$SrcDir/../dba";
$DbbD = "$SrcDir/../dbb";
$DbaN = "$SrcDir/../dba/prod";
$DblD = "$SrcDir/../db";
%SecDefines=(
  "DbaD",    "$DbaD",           # Database "A" directory
  "DbbD",    "$DbbD",           # Database "B" directory
  "DblD",    "$DblD",           # Database "link" directory
  "DbaN",    "$DbaN",           # Database "A" name
  "DbbN",    "$SrcDir/../dbb/prod",  # Database "B" name
  "QiInput",       "$SrcDir/../qi.input",  # input file
  "Util",    "$SrcDir/../bin", # utility commands
);
#
# flags that are used only in the ph client
#
%PhFlags=(
  "NsService",     "csnet-ns",        # name of nameserver service (RFC-
1700)
  "FallBackAddr","128.174.5.58",     # ns host ip address
  "FallBackPort",$DefineStrings{"DEF_PORT"},     # ns port number
  "MailDomain",    $DefineStrings{"MAILDOMAIN"}, # same as above
  "Host",    "$QiHost",         # default qi host
  "Alt",     "$QiAlt",          # default alternate qi host
  "Email_Auth",    "1",               # Use email authorization
);
```

From language.l:

```
%%
      BEGIN I;                      /* start state */
<I>query    {AddValue((char *)yytext,COMMAND); cmd = C_QUERY; BEGIN C;}
<I>ph       {AddValue((char *)yytext,COMMAND); cmd = C_QUERY; BEGIN C;}
<I>change   {AddValue((char *)yytext,COMMAND); cmd = C_CHANGE; BEGIN
C;}
<I>login    {AddValue((char *)yytext,COMMAND); cmd = C_LOGIN; BEGIN C;}
<I>xlogin   {AddValue((char *)yytext,COMMAND); cmd = C_XLOGIN; BEGIN
A;}
<I>klogin   {AddValue((char *)yytext,COMMAND); cmd = C_XLOGIN; BEGIN
A;}
<I>answer   {AddValue((char *)yytext,COMMAND); cmd = C_ANSWER; BEGIN
A;}
<I>email    {AddValue((char *)yytext,COMMAND); cmd = C_EMAIL; BEGIN C;}
<I>clear    {AddValue((char *)yytext,COMMAND); cmd = C_CLEAR; BEGIN C;}
<I>logout   {AddValue((char *)yytext,COMMAND); cmd = C_LOGOUT; BEGIN
C;}
<I>fields   {AddValue((char *)yytext,COMMAND); cmd = C_FIELDS; BEGIN
C;}
<I>add            {AddValue((char *)yytext,COMMAND); cmd = C_ADD; BEGIN
C;}
<I>delete   {AddValue((char *)yytext,COMMAND); cmd = C_DELETE; BEGIN
C;}
<I>set            {AddValue((char *)yytext,COMMAND); cmd = C_SET; BEGIN
C;}
<I>quit           {AddValue((char *)yytext,COMMAND); cmd = C_QUIT;
BEGIN C;}
<I>stop           {AddValue((char *)yytext,COMMAND); cmd = C_QUIT;
BEGIN C;}
<I>exit           {AddValue((char *)yytext,COMMAND); cmd = C_QUIT;
BEGIN C;}
<I>status   {AddValue((char *)yytext,COMMAND); cmd = C_STATUS; BEGIN
C;}
<I>help           {AddValue((char *)yytext,COMMAND); cmd = C_HELP;
BEGIN C;}
<I>id       {AddValue((char *)yytext,COMMAND); cmd = C_ID; BEGIN C;}
<I>siteinfo {AddValue((char *)yytext,COMMAND); cmd = C_INFO; BEGIN C;}

                  /* zone tranfer */
<I>axfr           {AddValue((char *)yytext,COMMAND); cmd = C_ZTRANSFER;
BEGIN C;}
```

configs/linux:

```
#
# configs/linux
#
# linux configuration file
#
# perl parameter file
# "extension" to perl configure of QI/PH system
#
# jonesjm - 8/16/97 - initial coding '(jonesjm@utulsa.edu)'
#
$CC = "gcc";            # use gnu c (although cc seems to point to gcc)
$Lex = "flex";          # use "fast lex"
$MoreLib = "-lfl";      # flex library
$Cflags .= " -I/usr/include/bsd"; # get sgtty.h ...
$DbmLib = "-lgdbm"      # to get "dbm" library
#
# Note:  The original distribution of qi-3.1B7 has an
include/sys/cdefs.h
# that wreaks havoc on the latest gcc - either move or remove the
cdefs.h
# to compile ok.  I'm not sure why this file is even in the include nor
# why the sys subdirectory exists.
#
# Note: Tim Lawless found a conflict with the variable name "dirfd" in
# dbd.c and dbm.c and certain routines in libc.  If you change the
variable
# name to dir_fd (as I did), I think you will be ok.
#
# Note:  The configs/default contains a line that has a literal in
double
# quotes: "p-pomes@uiuc.edu".  You will want to change this to single
quotes
# for Perl to not attempt to incorrectly translate the "@...".  If you
use
# the examples I have given below for the Configure command you will
have
# some type of corresponding statement in your own Configure Perl
script that
# you will want to use single quotes with also.
#
# I used the Slackware Linux distribution.  I see no reason for any
other
# Linux to not work, but I will express the standard disclaimer as
# usual.  This software has no warranty of any kind, but can be
distributed
# freely.  Although I do not require my name to be used, nor claim any
# copyright, it would be nice for you to acknowledge my efforts, i.e. a
# "Thank you" once in a while is nice to get. :-)
#
# The only errors I got from make after using this file with Configure
were
# warnings for incompatible pointer types in argument lists.  I'll
leave that
# problem, if there really is one, to the next person.
```

```
#
# To use this file, do not alter the configs/defaults, but create your
own
# perl parameter file overriding the values in configs/defaults then
enter
# the following (as an example) using ~mysystemconfig as your own Perl
# parameter file that you would use for your site.
#
# $ Configure configs/linux ~mysystemconfig
#
# Then do make:
#
# $ make install
#
# to generate the code.
#
# The rest of the setup is covered quite well in the docs and the faq.
#
# Here is also a list of dependencies (as best as I remember them) that
may
# help if you have a rather small disk (as I do) and need to install
with the
# minimum amount of supporting software to get the stuff running.
#
# Perl
# Gnu C
# Bc
# rcs
# ln
# Flex
# make
#
```

From lock.c:

```
/*
 * Set a zone "lock" - no lock needed - destination checks seq #
 */
int
GonnaZone(msg)
char *msg;
{
        bintree_init(Database); /* initialize the bintree code */
        read_index(Database);
        if (!dbi_init(Database) || !dbd_init(Database))
        {
                fprintf(Output, "%d:Couldn't open database.", LR_INTERNAL);
                exit(1);
        }
        get_dir_head();
        return (1);
}


/*
 * Release a lock
 */
void
Unlock(msg)
char *msg;
{
#ifdef FCNTL_LOCK
        struct flock arg;

        memset(&arg, (char)0, sizeof (arg));
        arg.l_type = F_UNLCK;
#endif /* FCNTL_LOCK */

        if (!strcmp(msg, "zone"))       /* zone lock? */
        {
                put_dir_head();
                put_tree_head();
                return;
        }

        LockInit();
        if (Type == WRITE)
        {
                incr_dir_head_serial(); /* for zone xfr */
                put_dir_head();
                put_tree_head();
        } else
                close_tree();
#ifdef NO_READ_LOCK
        if (Type == READ)
                return;
#endif
        IssueMessage(LOG_DEBUG, "%s %sunlock", msg, (Type == READ) ? "r"
: "w");
#ifdef FCNTL_LOCK
      if (fcntl(Lock, F_SETLKW, &arg) == -1)
#else /* !FCNTL_LOCK */
      if (flock(Lock, LOCK_UN) < 0)
#endif /* FCNTL_LOCK */
            IssueMessage(LOG_ERR, "Unlock: flock: %s",
strerror(errno));
}
```

**secondary/makefile.templ**:

```
# Copyright (c) 1998 James M. Jones
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions
# are met:
# 1. Redistributions of source code must retain the above copyright
#    notice, this list of conditions and the following disclaimer.
# 2. Redistributions in binary form must reproduce the above copyright
#    notice, this list of conditions and the following disclaimer in
the
#    documentation and/or other materials provided with the
distribution.
# 3. All advertising materials mentioning features or use of this
software
#    must display the following acknowledgement:
#     This portion of this product includes software developed by
#     James M. Jones, and other contributors.
# 4. Neither James M. Jones nor the names of other
#    contributors may be used to endorse or promote products derived
from
#    this software without specific prior written permission.
#
# THIS SOFTWARE IS PROVIDED BY THE CONTRIBUTORS ``AS IS'' AND
# ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
# IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE
# ARE DISCLAIMED.  IN NO EVENT SHALL THE CONTRIBUTORS BE LIABLE
# FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL
# DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE
GOODS
# OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
# HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT
# LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
WAY
# OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY
OF
# SUCH DAMAGE.
#

SHELL = /bin/sh

######################################################################
#
######################################################################
CC    = _CC_
CFLAGS      = _Cflags_
LFLAGS      = _Lflags_
######################################################################
# other variables
######################################################################
FLGS  =
SRCS  = secondary.c ../qi/conf.c
OBJS  = secondary.o ../qi/conf.o
BINS  = secondary
all:$(BINS)
#######################################################################
# specific targets
#######################################################################
secondary:  $(OBJS)
      $(CC) $(CFLAGS) -o secondary $(OBJS) \
        _Lflags_ _ApiLib_ _NsLib_ _MoreLib_

#######################################################################
# standard targets
#######################################################################
install:    $(BINS)
      -mkdir _QiExecDir_
      for b in $? ; do \
        _SrcDir_/bin/install.sh -m _Mode_ -o _Owner_ -g _Group_ $$b
_QiExecDir_; done
#
# jmj - 3/27/99 - for database rebuild
      -mkdir _DbaD_
      -mkdir _DbbD_
      -ln -snf _DbaD_ _DblD_
#
      @echo ""
      @echo "To run the secondary server zone refresh from this
machine,"
      @echo "Manually place a copy of the ?.cnf file from the primary"
      @echo "server in both database locations (the secondary cannot"
      @echo "rebuild without it), and"
      @echo "place the following line in your /etc/rc.d/rc.local"
      @echo "(or equivalent) file to get the server started at boot
time:"
      @echo "_QiExecDir_/secondary"
      @echo ""
      @touch install

rcs:  $(SRCS)
      _Ci_ $?
      touch rcs

clean:
      rm -f $(BINS) $(OBJS) tags install
depend:
      perl -i -e 'while(<>){print;if (/^# DO NOT DELETE/) {last;}}'
Makefile
      $(CC) _Depend_ $(CFLAGS) $(SRCS) >> Makefile
```

secondary/rebuild.templ:

```
#!/bin/sh
# rebuild.templ
#
# rebuild the qi database
#
# jmj - 1/19/99 - initial coding
# see Configure for translation
#
# parameters:
# 1 - serial:     -SERIAL=n
# 2 - refresh     -REFRESH=n
# 3 - retry -RETRY=n
# 4 - expire      -EXPIRE=n
#
# databases
DbaD
DbbD
DblD
DbaN
DbbN
# input file
QiInput
# utility commands
Util
#
if (!(file $DblD | grep "symbolic link" > /dev/null))
then
  echo "$DblD is not a symbolic link"
  exit
fi
#
# figure which database is currently being used
#
if (file $DblD | grep "$DbaD" > /dev/null)
then
  currdbd=$DbaD
  currdb=$DbaN
  otherdbd=$DbbD
  otherdb=$DbbN
else
  if (file $DblD | grep "$DbbD" > /dev/null)
  then
    currdbd=$DbbD
    currdb=$DbbN
    otherdbd=$DbaD
    otherdb=$DbaN
  else
    echo "$DblD != $DbaD or $DbbD"
    exit
  fi
fi
#
# get rid of old stuff
rm $otherdb.* 2> /dev/null
```

```
# put back configuration file
cp $currdb.cnf $otherdbd
#
# put together new db
#
if ($Util/credb `$Util/sizedb $currdb.cnf $QiInput` $otherdb)
then
  if ($Util/maked $1 $2 $3 $4 $otherdb < $QiInput)
  then
    if ($Util/makei $otherdb)
    then
      if ($Util/build -s $otherdb)
      then
#
# switch links
#
        ln -snf $otherdbd $DblD
#
      fi
    fi
  fi
fi
#
```

secondary.c:

```
/* */
/* secondary.c - set up a secondary data base for a ph server */
/* */
/* jmj - 9/1/98 - initial coding */
/* */
/* to do this, I basically scarfed up sections of PH and some QI.

   The QI code has been modified to respond to a zone transfer command
   "axfr" from the IP of what it thinks is the secondary server.

   There are 2 pieces to the secondary server: 1) a duplicate primary
   server to honor normal PH clients, and 2) a "secondary" (this code)
   which is really functioning as a "client" to the primary server.
   The secondary periodically checks, via siteinfo, for a difference
   in "serial" numbers between the secondary database and the
   primary database.  When a difference is found, the "secondary"
   request a zone transfer, rebuilds the database and fiddles with
   links to get the "server" portion of the secondary to look at
   the latest database.

      Notes:

   I took out all the VMS stuff since QI was never set up to run
   in a VMS environment.

   I am not overly proud of the code, but it does work.
*/

#include "protos.h"

#ifdef OSF1
# define INT32 int
#else /* !OSF1 */
# define INT32 long
#endif /* OSF1 */

#include <stdio.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/socket.h>
#include <sys/file.h>
#include <sys/stat.h>
#include <netinet/in.h>
#include <netdb.h>
#include <ctype.h>
#include <fcntl.h>
#include <arpa/inet.h>
#include <sys/param.h>
#include <errno.h>

#ifdef OS2
#include <process.h>
#endif   /* OS2 */
```

```
#include <qiapi.h>

/*
 * declarations for the functions in this file
 */
int  ContactQI();

int  GetGood __P((char *, int, FILE *));

void ComplainAboutHost __P((char *));

#ifdef __STDC__
# include <unistd.h>
# include <stdlib.h>
# include <string.h>
#else /* !__STDC__ */
# include <strings.h>
#endif /* __STDC__ */

/* for zone record type */
#define Other_Rec 0
#define Header_Rec 1
#define Zone_Rec 2

FILE *ToQI;             /* write to this to tell the nameserver stuff
*/
FILE *FromQI;                 /* read nameserver responses from here */
#define qprintf fprintf
#define qflush fflush

int  LocalPort = 0;           /* local port in use */
int Debug = 0;                /* print debug info if set */
int Dont_rerun = 0;           /* don't resubmit via "at" */

/*
 * switches
 */
char *UseHost = 0;            /*-s use server on what machine */
int  UsePort = 0;       /*-p use port # */

char  *Me;        /* the name of this program */
int retry_interval = 0; /* so trylater can get to it */
void trylater(void);    /* resubmit routine */

OPTION      OptionList[] =
{           /* only the nolog option is used */
      "echo", 0,  /* echo commands before execution */
      "limit", 0, /* limit the number of entries a command may affect
*/
      "verbose", 0,     /* I don't remember */
      "addonly", 0,     /* do not change fields that contain
information */
      "nolog", 0, /* do not issue syslog information */
      0, 0
};
```

```
/*
 * and the fun begins...
 */
int
main(argc, argv)
    int  argc;
    char **argv; {

    char scratch[MAXSTR];
    int Zone_Type(char *);
    int Serial_Part(char *);
    void All_Parts(char *, int *, int *, int *, int *);
    int local_serial = 0;
    int remote_serial = 0;
    int refresh_interval = 0;
    int retry_interval = 0;
    int expire_interval = 0;
    char * Zone_Data(char *);
    int qi_input_open = 0;
    FILE *qi_input;
    int last_rec;
    int not_this_time;
    int  count = 0;   /* for command line option processing */
    int new_qi;   /* switch for got database at least once */
    int got_first_serial;     /* switch */
    short code;
    char rebuild_line[255];

    Me = *argv;

/*
 * process a set of options
 */
    /* options processing */
    for (count = 0, argv++, argc--; argc && **argv == '-'; argc--,
argv++, count++) {
      for ((*argv)++; **argv; (*argv)++) {
          switch (**argv) {
            case 'O':
            case 'o':
            Dont_rerun = 1;
            break;
            case 'D':
            case 'd':
            Debug = 1;
            break;
            case 'p':
            if (isdigit(argv[0][1])) {
                UsePort = atoi(*argv + 1);
                goto whilebottom;
            }
            else if (argc > 1 && isdigit(*argv[1])) {
                UsePort = atoi(argv[1]);
                argc--, argv++, count++;
                goto whilebottom;
            }
            else
```

```
                fprintf(stderr, "-%c option given without port
number.\n", **argv);
            break;
            case 'H':
            case 'h':
            default:
            fprintf(stderr, "%s help:\n", Me);
            fprintf(stderr, "%s [-D|d] [-p port] [-H|h] [-O|o]\n", Me);
            fprintf(stderr, "  where:\n");
                fprintf(stderr, "D|d - debug\n");
                fprintf(stderr, "p - port number to use\n");
                fprintf(stderr, "H|h - help\n");
                fprintf(stderr, "O|o - run one time (don't
resubmit)\n");
            exit(0);
          }
      }
whilebottom:;
    }
    if (Debug) {
      printf("Debug on....\n");
      printf("Dont_rerun = %d\n", Dont_rerun);
      DoSysLog(0);      /* put messages to stderr */
      printf("(printf)If DoSysLog worked, you should see 2 of these
messages.\n");
      IssueMessage(LOG_ERR, "(logmsg)If DoSysLog worked, you should see
2 of these messages.");
      }

    not_this_time = 0;          /* optimism .... */
    if (!ContactQI()) {
        IssueMessage(LOG_ERR, "%s remote database unavailable.", Me);
        not_this_time = 1;
      }
    else if ((code = GetQISRRE(&remote_serial
          , &refresh_interval
          , &retry_interval
          , &expire_interval)) > 200) {
        IssueMessage(LOG_ERR, "%s remote information unavailable.",
Me);
        not_this_time = 1;
        if ((code = Get2ndSRRE(&local_serial
          , &refresh_interval
        , &retry_interval
        , &expire_interval)) != 200) {
          IssueMessage(LOG_ERR, "%s local information unavailable.",
Me);
                         /* can't get interval from anyone */
          retry_interval = atoi(RETRY);
          }
        } /* endif got retry from remote */

    if (!Dont_rerun) {
      atexit(trylater);             /* set up resubmit */
      }
    if (Debug) printf("got past atexit... \n");
    if (Debug) printf("not_this_time:|%d|\n", not_this_time);
    if (not_this_time) exit(0);   /* can't get to something - quit */

/* get serial #s */

    if ((code = GetQISRRE(&remote_serial
        , &refresh_interval
        , &retry_interval
        , &expire_interval)) > 200) {
        IssueMessage(LOG_ERR, "%s unable to get remote serial #", Me );
        exit(0);
        }
    if (Debug) printf("remote_serial:|%d|\n", remote_serial);
    if ((code = Get2ndSRRE(&local_serial
                    , &refresh_interval
                    , &retry_interval
                    , &expire_interval)) != 200 ) {
        IssueMessage(LOG_ERR, "%s unable to get local serial #", Me );
        exit(0);
        }
    if (Debug) printf("local_serial:|%d|\n", local_serial);

/* loop1 - zone transfer data if serial #s are different */

    new_qi = 0;              /* switch for new database gotten */
    while (local_serial != remote_serial) { /* change in serial #? */

/*    send axfr */

      qprintf(ToQI, "axfr\n");
      qflush(ToQI);
      if (Debug) {
      printf("sent=axfr\n");
      fprintf(stderr, "sent=axfr\n");
      }

/*    get records */

      got_first_serial = 0;  /* don't have first serial yet */
      last_rec = 0;

      while (GetGood(scratch, MAXSTR, FromQI)) {      /* read it */
      if (Debug) printf("secondary scratch:|%s|\n", scratch);
        code = atoi(scratch);
      if (Debug) printf("secondary code:|%d|\n", code);
        if (abs(code) == LR_OK) {

          if (Debug) printf("Zone_Type(scratch): %d\n",
Zone_Type(scratch));
        switch (Zone_Type(scratch)) {

/*    get serial records ~~~~*/
          case Header_Rec:
            if (!got_first_serial) {
              local_serial = Serial_Part(scratch);
            got_first_serial = 1; /* got it now */
```

```
              }
            else { /* got_first_serial - this must be last serial */
            All_Parts(scratch /* this must be latest info */
                   , &remote_serial
                   , &refresh_interval
                     , &retry_interval
                   , &expire_interval);
            last_rec = 1;
            new_qi = 1; /* also got at least 1 new database now */
            }
            break;

/*    get zone data */
          case Zone_Rec:
            /* write record to scratch area */
            if (!qi_input_open) {
              if (!(qi_input = fopen(QI_INPUT, "w"))) {
              IssueMessage(LOG_ERR, "%s could not open QI_INPUT for
write: %d", Me, qi_input);
              exit(0);
              }
            qi_input_open = 1;
              }
            fprintf(qi_input, Zone_Data(scratch)); /* write the zone
data */
            break;

          default:
              IssueMessage(LOG_ERR, "%s undefined record type for
record:%s", Me, scratch );
            exit(0);

          } /* end switch zone record type */
        } /* end if response "normal" */
        else { /* not a "normal" kind of record for axfr */
          IssueMessage(LOG_ERR, "%s unexpected record type: %d", Me,
code);
          IssueMessage(LOG_ERR, scratch);
          exit(0);
          }
        if (last_rec) break; /* if last good serial - done */

        } /* end while got good record */
        } /* end while serials different (serials match) */

    qprintf(ToQI, "quit\n");              /* let server know you are
done */
    qflush(ToQI);
    if (Debug) fprintf(stderr, "sent=quit\n");

    if (Debug) fprintf(stderr, "new_qi:|%d|\n", new_qi);
    if (new_qi) {    /* new qi database? */
      sprintf(rebuild_line
        , "%s -SERIAL=%d -REFRESH=%d -RETRY=%d -EXPIRE=%d"
        , REBUILD
      , remote_serial ? remote_serial : (local_serial ?
                                 local_serial :
                                  atoi(SERIAL))
      , refresh_interval ? refresh_interval : atoi(REFRESH)
      , retry_interval ? retry_interval : atoi(RETRY)
      , expire_interval ? expire_interval : atoi(EXPIRE)
      );
      system(rebuild_line);  /* run rebuild script */
    }

    exit(0); /* done (go again later - see "trylater") */
}  /* end of main routine */


/*
 * get the local info
*/
int Get2ndSRRE(int *serial
            , int *refresh
            , int *retry
            , int *expire) {
  char temp_name[100];
  int temp_fd;
  struct dirhead DirHead;

  (void) strcpy(temp_name, DATABASE);
  (void) strcat(temp_name, ".dir");
  if ((temp_fd = open(temp_name, 0)) < 0) { /* only need to read */
    IssueMessage(LOG_ERR, "%s unable to open local database %s", Me,
temp_name );
    return 0;
    }

  if (lseek(temp_fd, 0, 0) < 0) {
    IssueMessage(LOG_ERR, "%s lseek(%d,0,0): %s", Me,
                  temp_fd, strerror(errno));
    return 0;
    }
  if (read(temp_fd, &DirHead, sizeof (DirHead)) < 0) {
    IssueMessage(LOG_ERR, "%s read DirHead: %s", Me, strerror(errno));
    return 0;
    }
  if (close(temp_fd) < 0) {
    IssueMessage(LOG_ERR, "%s close: %s", Me, strerror(errno));
    return 0;
    }

  *serial  = DirHead.serial;
  *refresh = DirHead.refresh;
  *retry   = DirHead.retry;
  *expire  = DirHead.expire;
  return 200;
}


void trylater(void) {
/*
 * routine to resubmit zone transfer
*/
```

```
  char resubmit[MAXSTR];
  char secondary_executable[MAXSTR];

  (void) strcpy(secondary_executable, QIEXECDIR);
  (void) strcat(secondary_executable, Me);
  printf("echo %s | at now + %d minutes"
      , secondary_executable, retry_interval/60);
  sprintf(resubmit, "echo %s | at now + %d minutes"
      , secondary_executable, retry_interval/60);
  system(resubmit);
  }

/*
 * figure out which type of zone record this is
*/
int Zone_Type(char *rec) {
  char *ref;

  ref = strchr(rec, ':');
  if (strstr(ref,":1:")) return Header_Rec;
  else if (strstr(ref,":2:")) return Zone_Rec;
  else return Other_Rec;
}

/*
 * just get zone data - strip return code and record type
*/
char * Zone_Data(char *in) {
  char *temp;
  temp = (char *) strchr(in, ':');  /* get rid of return code */
  *temp++;
  temp = (char *) strchr(temp, ':');      /* get rid of record type */
  *temp++;
  return temp;
}

/*
 * pull serial number out of header rec on zone transfer
*/
int Serial_Part(char *rec) {
  char *ref;

  ref = strstr(rec, ":1:1:") + 5;
  ref = strstr(ref, ":2:") + 3;
  return atoi(ref);
}

/*
 * pull all numbers out of header rec on zone transfer
*/
void All_Parts(char *rec
            , int * serial
            , int * refresh
            , int * retry
            , int * expire) {
  char *ref;
  ref = strstr(rec, ":1:1:") + 5;
  ref = strstr(ref, ":2:") + 3;
  *serial = atoi(ref);
  ref = strstr(ref, ":3:") + 3;
  *refresh = atoi(ref);
  ref = strstr(ref, ":4:") + 3;
  *retry = atoi(ref);
  ref = strstr(ref, ":5:") + 3;
  *expire = atoi(ref);
}


    /* send siteinfo and get current information */

int GetQISRRE(int * serial, int * refresh, int * retry, int * expire) {

  char scratch [MAXSTR];
  char *lastc, *s2lastc;
  short code = 0;

      if (Debug) printf("GetQISRRE start\n");

      qprintf(ToQI, "siteinfo\n");
      qflush(ToQI);
      if (Debug) printf("GetQIRetry after print of siteinfo\n");

      while (GetGood(scratch, MAXSTR, FromQI)) {       /* read it */
        code = atoi(scratch);
        if (code == -200) {
          if ((lastc = (char *) strrchr(scratch, ':')) && lastc >
scratch) {
            *lastc++ = 0;
            if (s2lastc = (char *) strrchr(scratch, ':')) {
                s2lastc++;
                if (!strcmp("retry", s2lastc)) {
                  *retry = atoi(lastc);
                } /* endif retry */
                if (!strcmp("serial", s2lastc)) {
                  *serial = atoi(lastc);
                } /* endif  */
                if (!strcmp("refresh", s2lastc)) {
                  *refresh = atoi(lastc);
                } /* endif  */
                if (!strcmp("expire", s2lastc)) {
                  *expire = atoi(lastc);
                } /* endif  */
            } /* endif 2nd 1 : */
          } /* endif 1st 1 : */
        } /* endif -200 */
      else if (code >= LR_OK) break;
      } /* end while */
      return (code ? 200 : 500); /* only fail if the connection broke
*/
}

/*
 * contact the central nameserver
 */
int
ContactQI()
{
    int  sock;                /* our socket */
    static struct sockaddr_in QI;   /* the address of the nameserver */
    struct servent *sp;       /* nameserver service entry */
    static struct hostent *hp;      /* host entry for nameserver */
    char host[80];
    char *baseHost;
    int  backupNum = 0;
    int  mightBackup;
    int  result = 0;
    int  err;

    QI.sin_family = AF_INET;

    if (Debug) printf("in ContactQI start\n");
    /* give up privs if using anything other than default port and host
*/
    if (UsePort || (UseHost && *UseHost)) {
      if (Debug)
          fprintf(stderr, "giving up privs (UsePort || UseHost)\n");
      (void) setgid(getgid());
      (void) setuid(getuid());
    }
    if (Debug) printf("in ContactQI after userport check\n");

    /* find the proper port */
    if (UsePort)
      QI.sin_port = htons(UsePort);
    else {
      QI.sin_port = htons(atoi(DEF_PORT));
    }

    if (Debug) printf("in ContactQI after userport check 2\n");

    /* find the proper host */
    baseHost = UseHost ? UseHost : QI_HOST;
    strcpy(host, baseHost);

    if (Debug) printf("in ContactQI, host:|%s|, baseHost:|%s|\n", host,
baseHost);

    if (!geteuid())
      LocalPort = (IPPORT_RESERVED - 1);
    if (Debug) printf("in ContactQI, LocalPort:|%d|\n", LocalPort);
    for (;;) {
      /* create the socket */
#ifdef OS2
      sock = socket(PF_INET, SOCK_STREAM, 0);
#else
      sock = LocalPort ? rresvport(&LocalPort) : socket(PF_INET,
SOCK_STREAM, 0);
#endif /* OS2 */
        if (Debug) printf("in ContactQI, sock:|%d|\n", sock);
      if (sock < 0) {
          perror("socket");
          goto done;
      }
      QI.sin_family = AF_INET;
      if (hp = gethostbyname(host)) {
#ifdef _CRAY
          memmove((char *) &QI.sin_addr, hp -> h_addr, 4);
#else
          memmove((char *) &QI.sin_addr.s_addr, hp -> h_addr, 4);
#endif
      }
      else {
          ComplainAboutHost(host);
          goto done;
      }
        if (Debug) printf("in ContactQI, hp:|%d|\n", hp);

      /* connect to the nameserver */
      if (connect(sock, (struct sockaddr *) & QI, sizeof(QI)) < 0) {
          if (errno == EADDRINUSE) {
            if (LocalPort)
                LocalPort = LocalPort - 1;
            continue;
          }
          perror(host);
            if (Debug) printf("in ContactQI, connect worked.\n");
          goto done;
      }
      else
          break;
    }
    if (Debug) printf("in ContactQI, after for loop.\n");

    if (backupNum)
      fprintf(stderr, "WARNING--backup host %s; information may be out
of date.\n", host);
    /* open path to nameserver */
    if ((ToQI = fdopen(sock, "w")) == NULL) {
      perror("to qi");
      goto done;
    }
    if (Debug) printf("in ContactQI, after to open.\n");
    /* open path from nameserver */
    if ((FromQI = fdopen(sock, "r")) == NULL) {
      perror("from qi");
      goto done;
    }
    if (Debug) printf("in ContactQI, after from open.\n");
    if (UseHost)
      free(UseHost);
    UseHost = strdup(hp ? hp -> h_name : inet_ntoa(QI.sin_addr));
    UsePort = ntohs(QI.sin_port);
    result = 1;

done:
    if (Debug) printf("in ContactQI, after done.\n");
    setgid(getgid());
    setuid(getuid());
    if (Debug) printf("in ContactQI, before return, result:|%d|.\n",
result);
    return (result);
}

/*
 * complain that there isn't an entry for QI_HOST in /etc/hosts
 */
void
ComplainAboutHost(name)
    char *name;
{
    fprintf(stderr, "Warning--unable to find address for ``%s''.\n",
          name);
}

/*
 * Dot2Addr - turn a dotted decimal address into an inet address
 * ---Assumes 4 octets---
 */
unsigned INT32
Dot2Addr(dot)
    char *dot;
{
    unsigned INT32 addr = 0;

    do {
      addr <<= 8;
      addr |= atoi(dot);
      while (isdigit(*dot))
          dot++;
      if (*dot)
          dot++;
    }
    while (*dot);
    return ((unsigned INT32) htonl(addr));
}
```

# VITA

James M. Jones

Candidate for the Degree of

Master of Science

Thesis: APPLYING DOMAIN NAME SYSTEM REAL-TIME REDUNDANCY TO THE CCSO PH PHONE DIRECTORY SYSTEM

Major Field: Computer Science

Biographical:

Education: Graduated for Thomas Jefferson High School, Dallas, Texas in May 1971; received Bachelor of Science degree in Computer Science from Florida International University, Miami, Florida in December 1976. Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University in December 1999.

Experience: Seasoned computer professional with over 20 years experience in system engineering, system management, system analysis, programming, training, system development, and project management in scientific and business computing with inter-personal abilities and proven problem solving skills. Provide creative, elegant, yet simple, modular, thorough and low maintenance solutions.