Zero Configuration Name Services for IP Networks

by

Paul E. Huck, Jr.

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

Massachusetts Institute of Technology

June 2001

Author____
Department of Electrical Engineering and Computer Science

Certified by____
Dr. Amar Gupta
Thesis Supervisor

Accepted by____
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Zero Configuration Name Services for IP Networks

by

Paul E. Huck, Jr.
Submitted to the
Department of Electrical Engineering and Computer Science

May 29, 2001

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The current implementation of the Domain Name System (DNS) requires extensive configuration and administration for proper operation. This thesis examines several solutions to reduce the need for human administration of DNS. A solution by Michael Feng that proposes adding an agent program to monitor and configure BIND dynamically is examined in detail. It is then compared to existing commercial products and research underway by the Internet Engineering Task Force (IETF) which have similar goals. Several improvements to Feng's design are suggested by the author that will increase the compatibility, security, and performance of the system.

Thesis Supervisor: Dr. Amar Gupta
Title: Co-Director of Productivity From Information Technology (PROFIT) Initiative

Acknowledgements

I would first like to thank Dr. Gupta for providing me with opportunity to work on this thesis as well as for the guidance and patience he has shown me.

In addition I would like to thank Ralph Preston, Kevin Grace and Sam Wiebenson for their technical support and overview of the project background. Also, without Mike Feng's original work much or this thesis would not be possible.

I would also like to thank my roommates Dan, Mark, John, Dave, and Chris for making this experience as enjoyable as possible and for all the distractions they offered me. Lea also deserves my thanks for making sure that I did not starve and die while alone at Athena and for the constant motivation to start writing.

Finally, I would like to give thanks to my parents, Paul and Patricia Huck, and my family for all the help and support they have provided me throughout my life and college career.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

Computer networks are no longer restricted to corporate intranets and the Internet. As households with multiple computers have become more common, smaller home networks have become increasingly more popular. Also many "non-PC" devices are available in network-ready forms. These include mobile phones, PDAs, and household appliances. Future plans call for every appliance in the house, from light-switches to the microwave, to connect to the home network [27]. The PC will act as the hub, and will be able to control every device on the network, as well as provide a link to the Internet. This will enable home users to enhance the functionality of everyday appliances such as downloading recipes from the Internet directly to their microwave oven [41].

The TCP/IP protocol forms the basis of communication between various devices on the network. It has become the standard protocol of the Internet, and consequently it has become the dominant protocol of Local Area Networks (LANs). The rapid growth of computer networks is accelerating innovations related to the TCP/IP protocol [40]. This innovation is necessary to keep pace with the dynamic requirements of managing a large TCP/IP network. The innovations cover many aspects, ranging from the physical network connections to the application level protocols. One aspect of IP networks that could benefit from innovation is the existing concept of the Domain Name System

(DNS). In a TCP/IP network, every resource is identified by its IP address, a unique 32-bit number assigned to the resource by the network administrator. Thus to communicate with a resource known to exist on the network, the IP address must be known to establish communication. Because human beings often find it easier to remember a name than a cryptic 12 digit number, most networks, including the Internet, provide a service that maps names to addresses. DNS is the standard naming service for the Internet. With it in place, all that is needed to communicate with another host is the hostname and the location of the DNS service.

For more than a decade, the Berkeley Internet Name Domain (BIND) has been the de facto standard for DNS implementations [40]. Nearly all modern networks use the public domain BIND implementation or commercial products derived from BIND. In fact, BIND has become virtually synonymous with the industry standards that define the DNS architecture. The DNS specification was written to allow product interoperability and thus permit and encourage flexibility in each DNS implementation. Thus, it is possible to develop new DNS implementations with added features, while remaining compliant with existing standards.

In particular, it would be desirable if a DNS service could be run with no lengthy configuration process and no need for user intervention after it has started. The current implementation of BIND requires a network administrator to write several configuration files in order to function correctly. These configuration files tell BIND the zone it is responsible for, the location of other DNS servers on the network, and the location of the root name server. Setting up this configuration can be a tedious process, as every time a new host is added or removed from the network, these files must be updated. In addition, if the network topology changes (such as by adding or removing name servers or by merging two networks together), significant changes are required to the configuration of BIND as the DNS zones and ownership of the zones may have changed too. This is certainly not ideal for home networks or networks consisting of mobile or wireless clients. In a home networked environment, hosts are frequently leaving and entering the network as appliances are turned on and off, or as new hosts are added. Mobile clients also will join and leave networks often as they are transported across network boundaries. In an ideal environment, hosts should be free to join and leave the network, and should immediately gain access to the services; this requires that the DNS be able to configure itself on a dynamic basis.

This thesis will examine current and proposed solutions to this problem. In Chapter 2 an overview of the current architecture and operation of DNS is given. Chapter 3 introduces a name a system proposed in [13] which is based on BIND and requires zero configuration and zero administration. Chapter 4 examines current industry solutions and other related research, and compares them to system in Chapter 3. In Chapter 5 the author proposes several design changes to the self-configuring and self-administering name system that allow for higher performance, security, and functionality. Finally, in Chapter 6 some conclusions are drawn.

Chapter 2 DNS

This section describes the hierarchical naming system and distributed architecture of the current DNS standard. It also examines the operations involved for performing updates to the network.

DNS was developed in 1984 by Paul Mockapetris as a distributed database that could resolve the address of any computer on the network [28]. It was created to replace the ASCII host files that associated host names with their respective IP addresses. These host files resided on every host in the network, and if a name was not listed in the file,

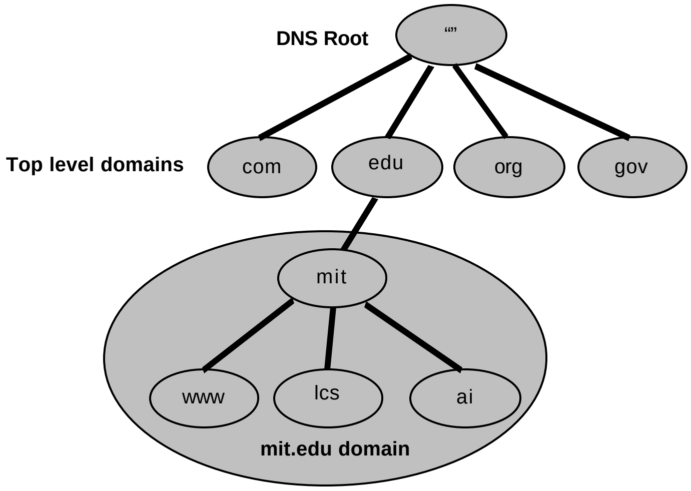


Figure 0 - DNS Hierarchy

that host could not be reached. As the Internet grew to become a worldwide network, the process of maintaining the host files became increasingly unwieldy, leading to a growing need for the DNS.

The DNS directory can be thought of as a tree, with each node on the tree representing a "domain" [1]. Each domain is identified and located in the tree by its domain name, which uses the familiar dotted notation (www.mit.edu, for example.). As we read the domain name from right to left, we can follow the path down the DNS tree to the correct node (See Figure 0). The "edu" domain is one of many top level domains; others include "com", "gov", "org", and "uk". The full and unique domain name of any node in the tree is the sequence of labels on the path from that node to the root.

The hierarchical tree of domain names can be referred to as the domain name space. Each domain in the space has host data associated with it. These host data are defined in resource records. There are many different types of resource records, the most common being the address-record (A-record), which associates the domain name with an IP address.

The other distinguishing characteristic of the DNS architecture is its distributed implementation. DNS servers may be operated by any organization which owns a domain. Each DNS server is given authority for one or more "zones". A zone is a collection of one or more DNS domains that share the same parent domain, along with the associated resource records. A DNS server receives authority over a zone when the network manager responsible for the domain that contains that zone delegates the authority to that particular DNS server. Therefore, the DNS infrastructure is distributed both geographically and administratively [40].

2.1 Name Servers

Each zone in the domain name space has at least two name servers authoritative for it, a primary and a secondary. Authoritative name servers are the only name servers guaranteed to contain the correct resource records for their zone. When querying a name on the Internet, a resolver can only be assured that the address is correct if the answer comes from an authoritative name server for the zone that is being queried [Figure 0]. To improve performance, other name servers may cache resource records, but each cached entry has time-to-live to prevent staleness of data. A name server stores all the resource records for the zone for which it is authoritative. The primary name server is an authoritative server for its zone(s) and reads its zone data from the zone files [1]. When changes are made to the zone's resource records, they must be made to the primary's zone file. This data is sent to secondary name servers upon request. A secondary name server is also authoritative for its zone(s); however, it obtains its zone data via data transfers from other name servers. A secondary name server interrogates the primary or other secondary servers periodically to determine if its zone's data have changed. The period of this interrogation can be set by a network administrator. A single name server can act as both the primary and secondary server for two or more different zones.

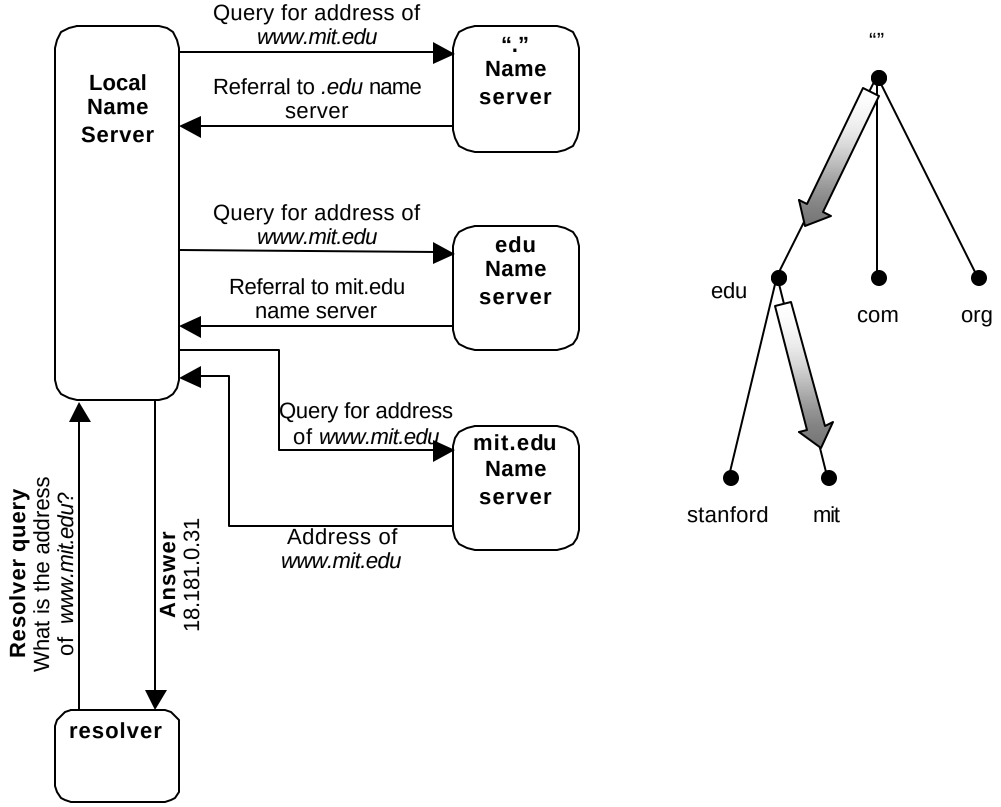

Query for address of www.mit.edu
"." Name server
Local Name Server
Referral to .edu name server
Query for address of www.mit.edu
edu Name server
Referral to mit.edu name server
Query for address of www.mit.edu
mit.edu Name server
Address of www.mit.edu
Resolver query What is the address of www.mit.edu?
Answer 18.181.0.31
resolver
""
edu
com
org
stanford
mit

Figure 0 - Using DNS to resolve www.mit.edu (from [13])

2.2 Dynamic DNS and Zone Transfers

The original DNS specification was written with static networks in mind. It was assumed that hosts would join and leave the network infrequently. However, in modern networks, computers are free to join and leave the network, and many new devices are being connected. In order to deal with dynamically changing networks, several extensions to DNS have been implemented. Specifically, RFC 2136 [48] defines the DNS Update protocol which allows for dynamic updates to the DNS. A dynamic update is a deletion or addition of resource records to a zone. It can be sent by either a DNS client, DNS server, or a DHCP server (see Section 3.1). The *update* signal is sent to the primary name server, which receives the signal and permanently updates its zone file. The signal may be sent to the primary server directly or passed through one or more secondary servers until it reaches the primary. When a primary server fulfills an *update* request, it can use the *notify* signal to inform its secondary servers that the zone information has changed.

In order for multiple name servers to maintain consistency of their records, zone transfers are performed on a periodic basis. A zone transfer is the transfer of resource records from a primary name server to a secondary name server. A full zone transfer occurs when all resource records are sent. Instead of sending all the resource records, it is possible for the primary name server to perform an incremental zone transfer. This will transfer only those records updated since the last zone transfer. A secondary name server requests an incremental zone transfer and a primary server chooses whether it will perform a full zone transfer or an incremental one. It is recommended that full zone transfers be performed no more than once every 15 minutes and at least once every 24 hours [3].

The following is a brief example of how the current version of BIND propagates changes to the resource records to all authoritative name servers. The process is illustrated in Figure 1.

1. An addition or deletion of a host is received by the primary server. This may come from administrator manually editing the zone file, or through an *update* message received. If the *update* message is received at a secondary server, it will pass it to the primary server if it knows its location, or to another secondary

server. This is continued until the primary server receives the *update* signal. Once the zone file is changed, the serial number of the file is incremented.

2. The primary name server reads the edited zone file. The frequency in which the server rereads its zone file and checks for zone changes is a configurable parameter of BIND.
3. The primary server will send a *notify* message to all known secondary servers. The primary server will wait some time between sending each *notify* to reduce the chance of multiple secondary servers requesting zone transfers at the same time.
4. If the secondary server(s) support(s) the *notify* signal, a zone transfer is immediately initiated. Otherwise, the secondary server will discard the *notify* and wait until the next scheduled zone transfer time.
5. The secondary server then notifies any other secondary servers which may be dependent on it for zone transfers. This multi-level transfer is continued until all secondary servers have received the changed records.

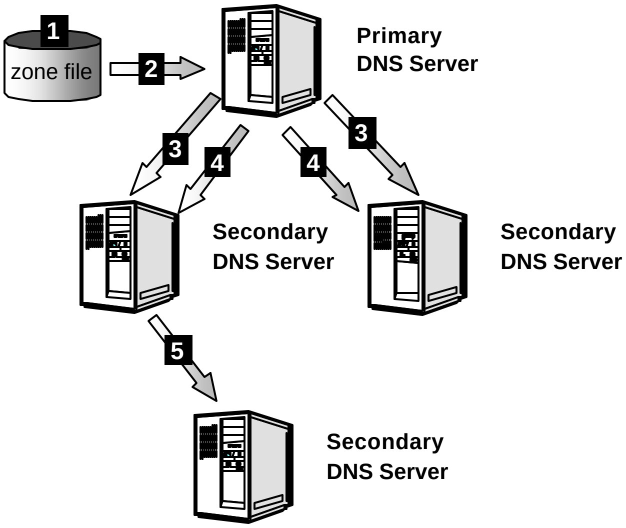


Figure 1 - BIND Updates (from [40])

While the dynamic update system may lessen the amount of administrative work for the name servers, it does not make them administrator free. Each name server in the network must be configured with the address of either the primary DNS server or other secondary DNS servers for its zone. Name servers are not free to join and leave the network. If a secondary name server is added to the network, either the primary DNS server or other secondary DNS must be configured to recognize the presence of the new secondary name server so that the primary DNS server can receive *update*s properly. Also, if the primary name server is removed and another added in its place, an administrator must manually change the configuration of every secondary server dependent on it for updates, so that it is informed of the new primary DNS server. In addition, the current DNS system requires extensive initialization effort to ensure proper operation. Zones must be properly allocated with specific primary and secondary servers and a clear domain hierarchy must be defined. These tasks all require extensive knowledge and effort from an administrator.

The above problems are addressed by the name system described in the next chapter. It uses a combination of an agent program monitoring BIND and dynamic DNS protocols to maintain a true self-configuring and self-administrating naming system.

Chapter 3 Self-Configuring and Self-Administering Name System

3.1 Network Overview

The naming system described in this section was conceived as part of a larger network system that is under development at a company codenamed as Research Corporation in this thesis. The proposed design provides a full array of network services to the hosts on the network. This is done with virtually no configuration and no administration. The network design allows for a group of computers to be physically connected together (through ethernet or other network media) and after a short time, they will all be properly configured in a working network. On this network there will be two types of nodes: managers and hosts [Figure 1 and Figure 1]. Managers form the backbone of the network and provide services, such as name to address resolution, to the hosts. Besides the name to address resolution service, the managers also provide dynamic IP configuration for new hosts, discovery mechanisms for new managers, packet routing, and a database of all hosts in the system. The services directly related to the naming service are automatic IP configuration and manager discovery, as these will be the services that the name system will directly interact with.

The goal of the network system is to allow any computer, either a host or a

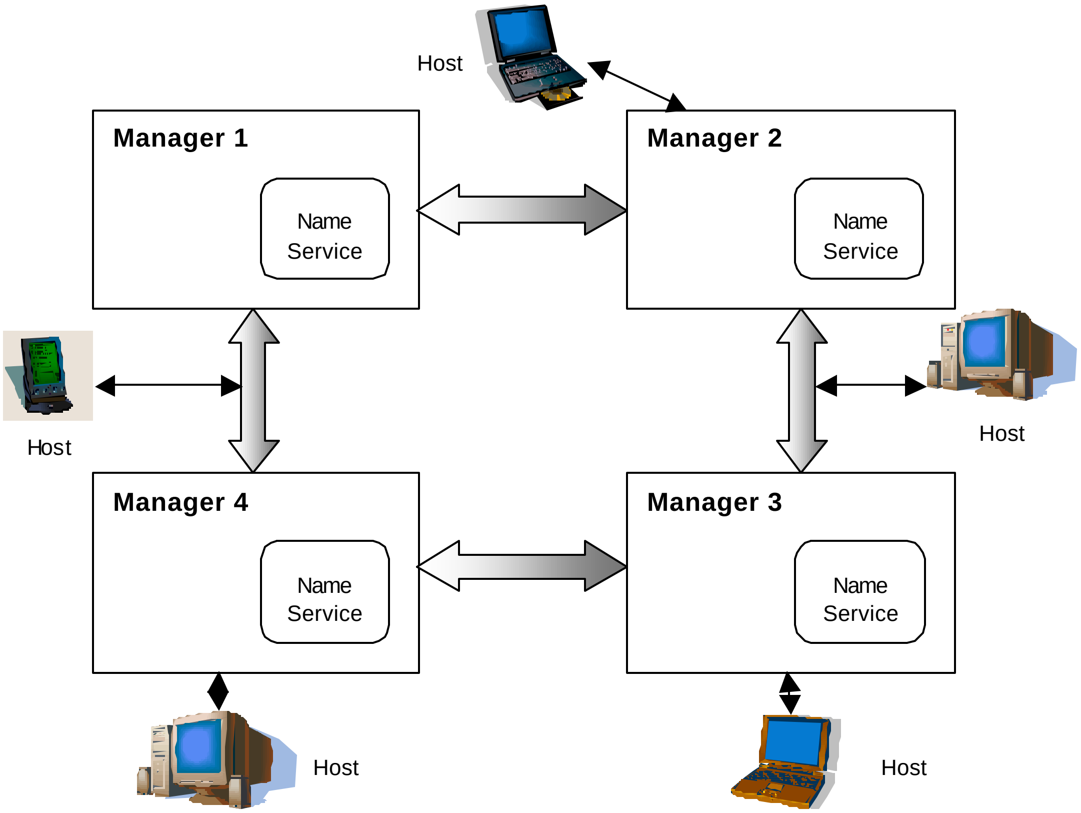


Figure 1 - Sample Network

manager, the freedom to join or leave the network, without the need for any external administration. The network should be able to detect the presence of a new node or the absence of an existing one, and deal with either type of change in an appropriate manner. In addition, the network system, and in particular the name system, should deal gracefully with the merging of two networks. Conflicts should be detected and resolved quickly.

In order to provide the self-configuring and self-administrating name service, each manager in the system is designed to run three types of services: the IP configuration service, the manager discovery service, and the agent based name service. The IP configuration service is handled by a standard implementation of the dynamic host configuration protocol (DHCP). DHCP is a network protocol specified by the IETF that provides hosts with configuration parameters for their IP interface [7]. This includes an IP address, a domain name, a subnet mask, a default gateway, and a location of DNS server. DHCP also allows a host to retain a previous configured domain name while receiving an IP address. After receiving the necessary information from a DHCP server, a previously unconfigured host's IP interface has all the necessary parameters in order to begin transmitting and receiving on the network. DHCP requires no prior configuration; as a host locates a DHCP server by broadcasting discover messages on the local network. All modern operating systems and even most embedded devices support DHCP [33]. Overall, DHCP meets the goals and design objectives of the desired self-configuring and self-administrating network.

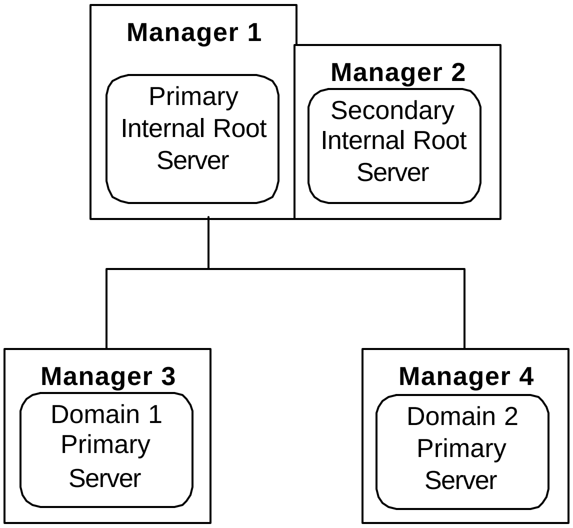


Figure 1 - Name Space View of Figure 1

Manager discovery is accomplished by having the manager discovery process periodically broadcast “discover packets” to each interface. These packets contain the source address and a unique network number that the manager resides on. All other manager discovery processes will receive the packet, and if the network number is known, the packet will be dropped. If, however, the network number is unknown, that manager will respond to the source of the “discover” with a “discover reply”. This reply includes the network number for the new manager. This allows the manager discovery process to inform the local name service of any new managers that appear on the network.

The design and the operation of the name service are discussed in the next subsection.

3.2 Name Service Interface

The name service runs on every manager in the system and consists of two concurrently running processes. The first is the BIND implementation of DNS. As stated before, BIND is currently used in the overwhelming majority of name servers on the Internet. It provides a scalable, stable, and fault-tolerant basis for the name service. The second process is an agent program which reacts to network conditions and configures BIND automatically. Figure 1 shows the relationship and manner of communication between BIND, the agent program, the other local manager processes, as well as other managers in the network. The agent program uses Berkeley UDP sockets to listen for two different formats of messages. On port 53, the standard DNS port, the agent program listens for DNS messages. It acts as a filter for the DNS messages, sending queries directly to the BIND process, and processing update messages from the IP configuration process. On port 54[1], the agent program listens for any agent messages coming from either the manager discovery service, or other agents in the system. The

[1] The IETF has designated port 54 to be used for XNS Clearinghouse [38]. We chose to use port 54 because XNS is uncommon on modern servers. If this service is needed, another port may be specified for agent communication.

agent messages allow an agent process to gain knowledge of other agents and offer a method of communication between agents. The details of both DNS and agent message processing are discussed in following sections.

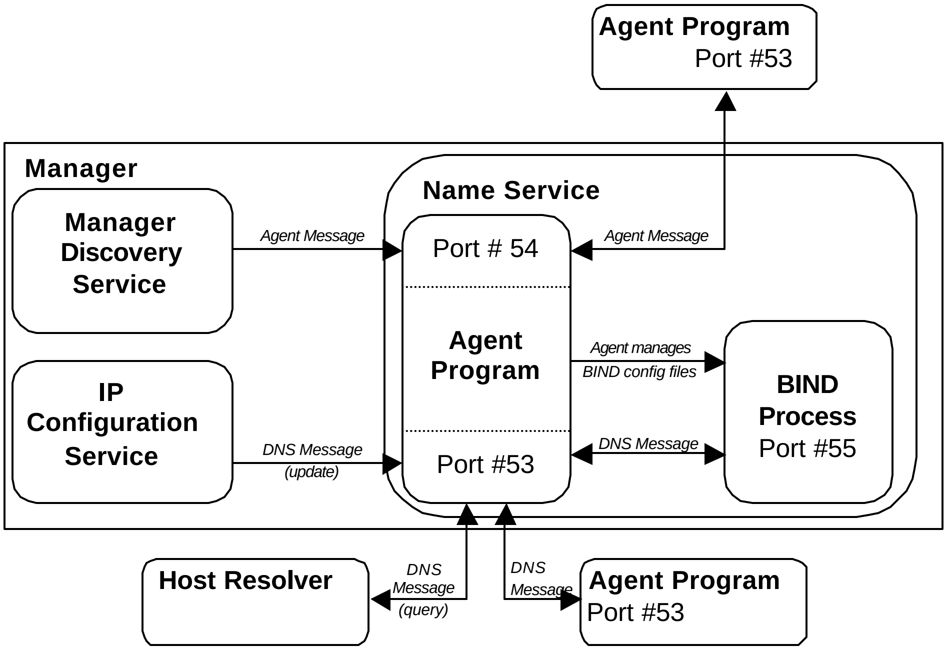


Figure 1 - Relationship between agent, BIND, and other processes (from [13])

3.3 Internal Root Servers

The name servers in the system are only authoritative for the IP addresses of the subset of hosts configured by the naming system. Presently there is no requirement for connectivity to outside networks such as the Internet. Therefore, hosts managed by the agent name system may communicate only with other hosts in the same system. Section 5.1 discusses extensions to the agent name system that will allow for communication with outside networks. With this in mind, the name system is implemented using the idea of internal root servers. Certain name services in the manager network act as the internal root servers and are authoritative for the entire domain that is serviced by the self-configuring naming system. So, as long as a name service knows the location of at least one internal root server, the name service will be able to provide name resolution for the network's name space. For example, if the entire mit.edu domain is using the self-configuring naming system, a host in the lcs.mit.edu sub domain may wish to know the IP address of a computer in another sub domain serviced by the naming system (such as media.mit.edu). This host will query any local name system, and since the manager is not authoritative for the media.mit.edu domain, it will query the internal root server of the mit.edu domain. This server may resolve any domain name in the mit.edu domain by directing the resolver to the proper media.mit.edu authoritative name server.

3.4 DNS Messages

The only DNS messages that the agent program processes directly are DNS updates. All other messages are simply directed to the local BIND process on port 55. These messages will simply be DNS queries and BIND will process the query, return an answer to the agent process, which in turn will forward the response to the inquirer. The agent process is the only entity with access to the BIND process. To all hosts on the network, it appears as if BIND is running on the normal port.

3.4.1 DNS Update

DNS updates, however, require special processing by the agent. DNS updates only occur when the IP configuration service or manager discovery service detects the addition or deletion of a host or manager on the network. The respective local manager service then assembles the appropriate DNS update message containing the resource record that must be added, modified, or deleted and sends it to the agent program. Although DNS updates may be sent to any DNS server that is authoritative for a zone, they will always end up being processed by the primary master for the zone, as that is the only server that has the definitive set of resource records for a particular zone. As stated in Section 2.2, standard DNS implementations state that any secondary server which receives a DNS update message must forward it directly to the primary master if its location is known, or alternatively through the secondary server chain until it reaches the primary master [48]. The secondary servers will only learn of the *update* through *notify* signals originating from the primary master.

To try to improve the efficiency of this mechanism, the agent program will examine the *update* message, and if it applies to RR in the zone that it is authoritative for, it will send it directly to the primary master for that zone. Each agent process has knowledge of its primary master as this is part of its state information described in Section 3.6. When the agent program is not authoritative for the zone receiving the *update*, then it must search for the primary master for that zone on the network. To do this, the agent program will send a Start of Authority (SOA) DNS query to the local BIND process. The SOA asks the DNS for the address of the primary master for a particular zone. Upon receipt of the SOA, BIND will perform standard DNS resolution to find the information on the zone name. If the zone exists, BIND will return to the agent process the IP address of the primary master for the requested zone, and the agent process will forward the *update* to that address. If, however, the DNS update message applies to a zone that does not exist on the network, BIND will return the non-existent domain error flag (NX_DOMAIN), and the agent program will configure the local BIND process to be authoritative for the new zone. The reconfigured BIND process will then process the update request. This allows for dynamic creation of zones and is especially useful at resolving all naming conflicts when two networks are merged.

3.5 Agent Messages

Agent messages are used for inter-agent communication and as a mechanism for agent discovery. Figure 2 shows the structure of an agent message. It consists of a three field header followed by a payload section. The header fields are explained in Table 1. The payload holds the data from the message. The data is specific to each Opcode. The data sent with each agent message is explained in more detail in the following sections.

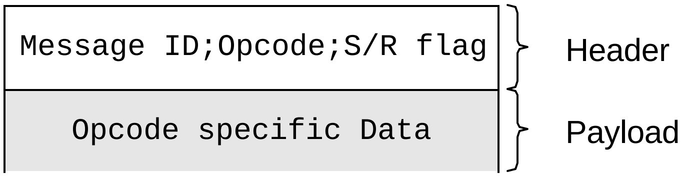


Figure 2- Agent Message Format

Table 1 - Header Fields

Header Field	**Description**
`Message ID`	The unique message ID for the message
`Opcode`	The operation code for message. Can have one of six values; see Table 2
`S/R flag`	Send/Response Flag. A flag indicating whether the message is a send request or response to an earlier message

The Opcode may take on one of six values. These values are explained in Table 2 and the following sections.

Currently the header is formatted as ASCII text separated by semicolons. A sample agent message header is shown here:

```
3;getroot;response
```

The total length of this header will be 18 bytes (one byte for each character). However, it is possible to represent each field with a more efficient binary representation. If the `message ID` were encoded as a 12 bit binary number (allowing for 2^{12} unique IDs), the `opcode` as 3 bit number, and the `S/R flag` as a one bit field, the header size could be reduced to two bytes. ASCII text was chosen to simplify the implementation and debugging of the system.

Table 2 - Opcode Values

Opcode	**Brief Description**
discover	Informs agent of new managers
leave	Informs agent of lost managers
getroot	Used to request the location of internal root server
becomeslave	Sent to an agent to make it a slave to the sender
negotiatemaster	Used to resolve primary master conflicts
forceslave	Sent by the winner of two conflicting servers, to force the loser into becoming a slave

3.5.1 Agent Message Behavior

The manager discovery process will send the local agent process *discover* and *leave* messages whenever it discovers that a manager has joined or left the system, respectively. These messages include the information about the new or lost server such as the zone it is authoritative for and also whether it was a master or slave for that zone. They allow an agent to gain a current view of the network topology and can also help to warn against conflicts and errors arising in the network. For example, if a manager that was a primary master for a zone disappeared, a slave for that zone would receive the appropriate *leave* message and negotiate with its peers to elect a new primary server for that zone.

The *negotiatemaster* and *forceslave* messages are designed to be used in such situations. The *negotiatemaster* is used when two managers discover that they are primary masters for the same zone. This may happen when two previously unconnected networks are physically joined. The two agents will exchange *negotiatemaster* messages and elect a new master. The *negotiatemaster* message includes metrics to help determine the optimal master, such as the number of slaves the server currently has. The winner of the election then sends a *forceslave* to the loser and requests the latter's zone data, so that such data may be merged with the existing data. The merge is accomplished via a zone transfer from the loser to the winner. If any conflicts are detected during the merge (such as two different hosts having the same name), then a new sub domain is automatically created for the loser, and every host listed in the loser's resource records is placed into that sub domain. See Figure 2 - 10 for an example.

The *becomeslave* message can also be used to make an agent a slave for a zone. However, while the *forceslave* message is only used between two competing primary servers, the *becomeslave* can be sent to any agent in the system. It is not used for elections, but only to notify agents of new primary servers. If a new primary server for a zone leaves the network and new one is elected, the *becomeslave* message is passed to all the old slaves of the former primary server and to the slaves of the loser in the election. The *becomeslave* message informs them of the new primary server and allows the agents to reconfigure themselves accordingly.

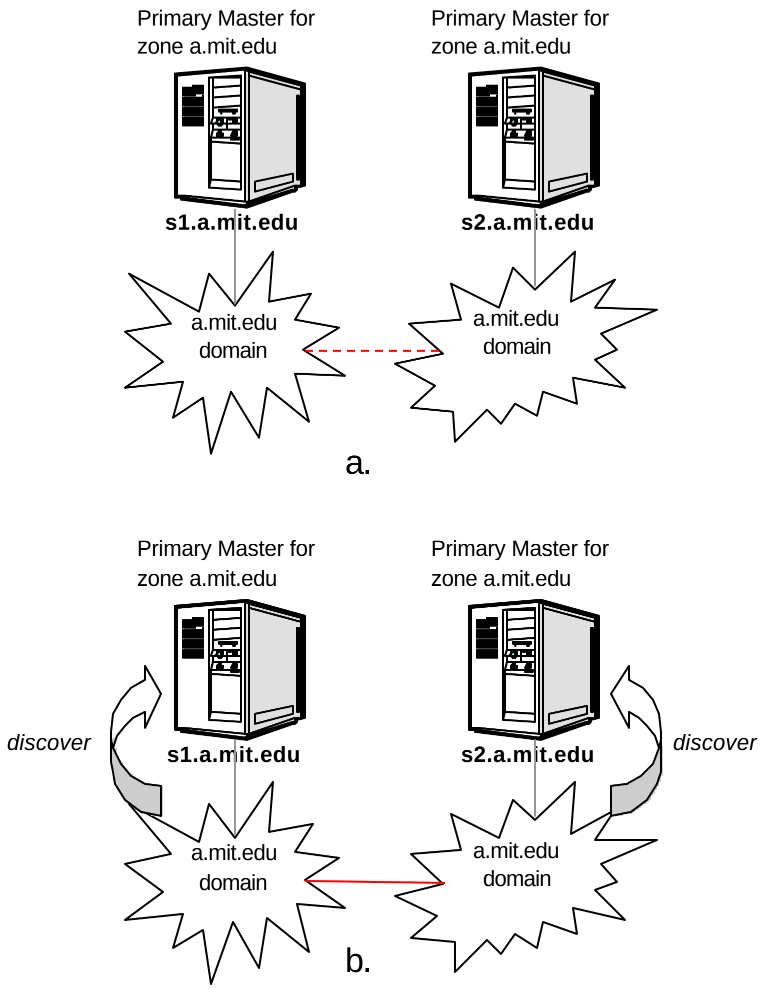


Figure 2 - Merging of two previously unconnected networks with name conflicts. a) A new physical connection joins the two (the dashed line). b) Agent programs in both networks receive *discover* messages informing them of the newly discovered managers.

The *getroot* message is used by agents to share the internal root server information. This is useful when a new unconfigured agent is introduced to the system and wishes to know the internal root server.

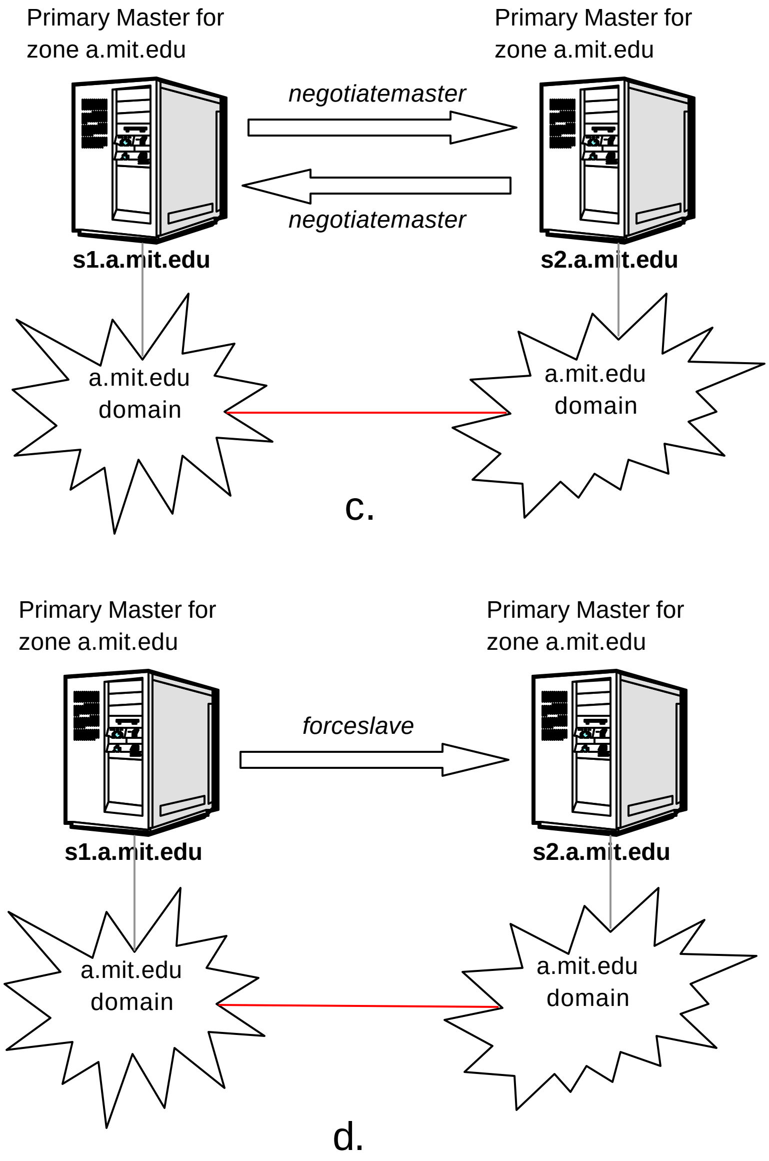


Figure 3 - c) The conflicting primary master servers send each other *negotiatemaster* messages. d) The winner of the election sends a *forceslave* message to the loser.

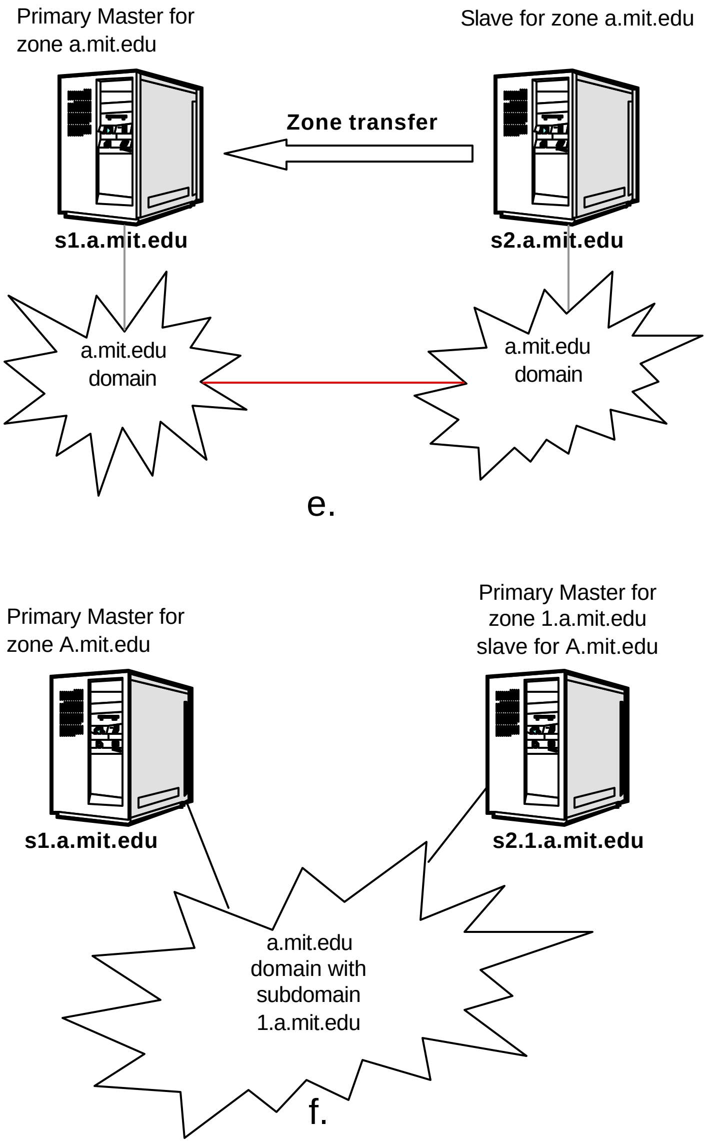


Figure 4 - e) The new slave performs a zone transfer with the master. f) Because there are name conflicts, a new sub domain is automatically created for every host in the loser's network. Every host in that network is renamed to be a part of the sub domain.

3.6 State Information

To assist with the agent tasks, each agent stores state information about itself and other managers in the network, as well as a log of all messages it has received. In particular, the agent will keep a record for all managers that it knows to exist on the network. This *server record* contains the name, IP address, a unique server ID, and a status flag that tells if the server is configured or just starting up [Table 3]. Each server record is placed into one or more of the following categories:

- Own Servers - contains the agent program's own server information.
- Known Servers - list of server records for every manager on the network.
- Root Servers - server records for every domain root server on the network
- Newly Discovered Servers - list of recently discovered managers. Once the agent processes the *discover* message, the server record will be moved to one of the above categories.

Table 3 - Server Record Fields

Server Record Field	**Description**
Name	Name of the manager the name service resides on.
IP Address	IP address of the manager the name service resides on
Server ID	Unique ID of the manager the name service resides on. The unique ID may be derived from the MAC address of the network interface in use by the manager.
Status Flag	*Configured:* name service is configured with the location of an internal root server. *Start-up:* name service is unaware of a location of an internal root server.

In addition to keeping records for every manager, each agent also keeps information on every zone it is authoritative for. If the server is a slave in that zone, the zone information includes the server record of the primary master for the zone. If the zone is the master for the zone, then the zone information includes the number of slaves for the zone, the *server records* of all the slaves, and the number of slaves required. If the number of slaves is less than the number required, the agent will attempt to send *becomeslave* messages to other agents.

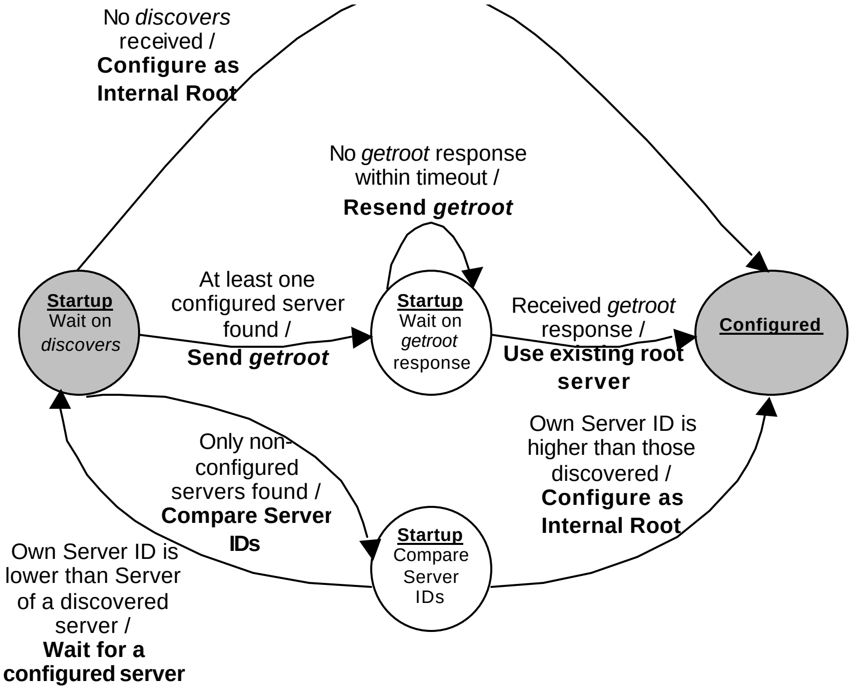
3.7 Operation

At any time, the agent program can be in two different states: the Configured state and the Start-up state. The state of the agent is stored in the in the server record information for the agent as described in Section 3.6.

3.7.1 Initial Configuration

The key piece of information any agent needs to operate is the location of an internal root server. On startup, the primary goal of any new agent is to find the location of an internal root server. Once an internal root is found, the agent is put into Configured state. An agent can only be in the Configured state if it knows the location of the internal root server, otherwise it is in the Unconfigured state.

Figure 4 depicts the startup scenario and the states of the transition between unconfigured and configured. When a new agent is started, it will wait for *discover* messages from the discovery process on the local manager. When at least one *discover* message is received, the new agent will send one *getroot* message to one of the discovered agents. If a specified timeout period expires and no other manager has been located, the agent will assume it is the only running manager on the system and configure itself to be the internal root. If the new agent hears only *discover* messages from Unconfigured managers, then they will compare *Server IDs* and elect the server with the highest ID to be the internal root server. This may happen when two or more managers join the network at the same time.



No *discovers* received /
Configure as Internal Root
No *getroot* response within timeout /
Resend *getroot*
Startup
Wait on *discovers*
At least one configured server found /
Send *getroot*
Startup
Wait on *getroot* response
Received *getroot* response /
Use existing root server
Configured
Only non-configured servers found /
Compare Server IDs
Own Server ID is higher than those discovered /
Configure as Internal Root
Startup
Compare Server IDs
Own Server ID is lower than Server of a discovered server /
Wait for a configured server

Figure 4 - Startup Scenario

3.7.2 Configured

Once an agent has entered the Configured state, it is ready to handle any name resolution query on the network. Even if it does not have the requested name in its database, it can query the internal root server and find an answer by working recursively down the DNS tree. When the agent is in the Configured state, it listens for queries, DNS updates, *discover* and *leave* messages, and (if it is a primary master for a zone) periodically runs the getSlave function. The getSlave function is used to find more slaves for a primary master. For every zone, the agent is a primary master for, the getSlave function will check the zone information to see if more slaves are needed. If more slaves are needed, the agent picks a random server out of the Known Servers list that is not already a slave for the zone. It then sends a *becomeslave* message to that server.

The behavior of the agent in response to both DNS queries and updates is described in section 3.4. The arrival of a *discover* or *leave* message signals a change in network topology, as managers have either joined or left the network. In the case of a *discover* message, there are two options: the discovery of a configured manager, or the discovery of an unconfigured manager. The case of the discovered unconfigured manager is rather simple as an agent need only record the *server record* of the new manager and respond to any *getroot* messages it may receive. When a configured manager is discovered, the process is slightly more complex.

Configured managers are discovered when two configured networks are joined. Network unions are revealed by the manager discovery process as two managers will have different network IDs that were previously unknown to the other manager. In this case it is possible to have a conflict where two managers are primary masters for the same zone. Therefore an agent needs to have a mechanism to detect and resolve this conflict.

Figure 4 illustrates the procedure when a configured agent receives a *discover* message. When an agent is a primary master for a zone and receives a *discover* message with information on new configured managers, the agent will send a *getroot* message to the newly discovered server with highest ID (the server ID is used so that only one server is contacted, any other metric could be used as well). Only primary masters need to participate in the conflict detection scheme, as any slave's master will detect the conflict and transfer the new network information in a future zone transfer.

Zone conflicts can only occur if the two networks have different internal root servers. If both networks shared the same internal root server, it would be impossible for zone conflicts to occur as a single DNS root tree will not allow the same zone to have two different primary masters. When the two internal root servers are different, an agent must make sure that it is the only primary master for its zone. Therefore, the agent will send a

SOA query to the differing root server for every zone that it serves as primary master for. The differing root server will then respond with the address of the manager that it considers to be the primary master for that zone, or an error message indicating that it is not aware of the zone. If the SOA response is an error message or if the address matches the querying agent, then no conflict exists[2]. If, however, the differing root server indicates that it believes another manager to be the primary master for the zone, a conflict exists and must be resolved. The conflict is resolved by the *negotiatemaster* master message described in section 3.5 and illustrated in Figure 2.

[2] If the SOA message is an error, this indicates the other root server is not aware of the zone on the network. However, it can still resolve names in that zone by recursing down the DNS tree from the top node.

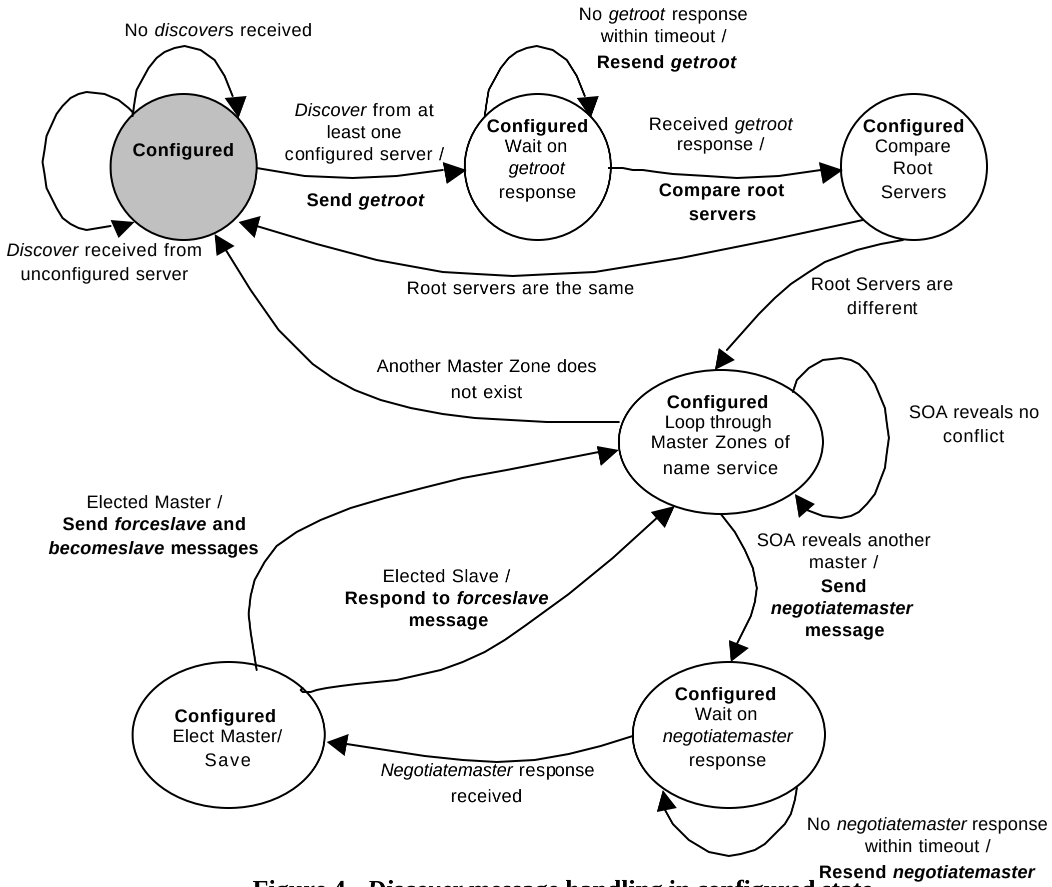

No *discovers* received
Configured
Discover received from unconfigured server
Discover from at least one configured server /
Send *getroot*
No *getroot* response within timeout /
Resend *getroot*
Configured Wait on *getroot* response
Received *getroot* response /
Compare root servers
Configured Compare Root Servers
Root servers are the same
Root Servers are different
Another Master Zone does not exist
Configured Loop through Master Zones of name service
SOA reveals no conflict
SOA reveals another master /
Send *negotiatemaster* message
Elected Master /
Send *forceslave* and *becomeslave* messages
Elected Slave /
Respond to *forceslave* message
Configured Elect Master/ Save
Negotiatemaster response received
Configured Wait on *negotiatemaster* response
No *negotiatemaster* response within timeout /
Resend *negotiatemaster*

Figure 4 - *Discover* message handling in configured state

An arrival of a leave message may also require an agent to reconfigure itself. If a master server loses a slave, it will delete the slave's server record from its state information and send an update to the BIND process informing it of the lost name server. If an agent is a slave, it only needs to be concerned with *leave* messages that inform it that its master has left the network. All other *leave* messages will be processed by its master. When a primary master has left the network, all its slaves will be notified and need to elect a new master for their zone. This is accomplished by the slave with the highest server ID asserting itself as master. The highest server ID can be calculated by looking at the Known Servers state information (see section 3.6). This new primary master will then send *becomeslave* messages to every other slave informing them of their new master.

Chapter 4 Related Work

4.1 ZEROCONF

The Internet Engineering Task Force's (IETF) ZEROCONF Working Group is currently proposing a protocol to enable networking in the absence of configuration and administration [52]. Although they have yet to propose a specific implementation or specification of the protocols, they have set a list of requirements for ZEROCONF protocols [17]. The goal is to allow hosts to communicate using IP without requiring any prior configuration or the presence of network services. Of particular relevance to this thesis is the name to address resolution problem. The ZEROCONF requirements specifically state that there should be no need for preconfigured DNS or DHCP servers. In fact, the ZEROCONF protocols are required to work in the absence of DNS servers. However, when these services are present, the ZEROCONF requirements direct that hosts should use them. Thus, the ZEROCONF requirements offer temporary and inferior solutions to the name resolution problem until a complete name resolver is located, such as a DNS server.

ZEROCONF protocols face two challenges when determining name to address bindings. The first is obtaining a unique address and/or hostname on the network. This is handled extremely well in modern networks by the use of a DHCP server; however, ZERCONF protocols must not rely on the presence of DHCP server. Therefore, the working group recommends using either IPv6 or IPv4 auto-configuration [15]. IPv6 auto configuration is vastly superior as it allows hosts to obtain a link local address (useful only on a single network) using address auto configuration [15] and a routable address by discovering routers using Neighbor Discovery [30]. IPv4 auto configuration is still in the research state by the IETF, but the initial specifications allow a host to only get a link-local address [5]. This will prevent the host from communicating with any device not directly on the same network as it. Also, while IPv6 provides nearly all necessary network parameters such an address, domain name, default router, and DNS server location (if present), IPv4 provides only an address. Thus, if a host configured with IPv4 auto configuration leaves a network and rejoins, it may have a new address, while a host configured by IPv6 will have a permanent method of contact, its domain name. While IPv6 clearly offers more advantages, it is expected that IPv4 will dominate for some time [5]. This is because IPv6 is relatively new standard and the lack of fully complaint IPv6 routers and hosts on most networks has prevented it from gaining widespread acceptance.

Once a ZEROCONF host has obtained an address on the network, it still has to discover other hosts and resolve domain names. The ZEROCONF requirements state that hosts should use multicast to resolve names in the absence of a DNS server. In order to support this requirement, an IP host will also need to listen for such requests and respond when the request corresponds to the host's own name. The IETF currently has two ongoing works in this area: multicast DNS [12] and "IPv6 Node Information Queries" [6]. In each case, all hosts will run a "stub" name service that only responds when it fields a request for its hostname. The stub service does not provide any name to address resolution for other hosts on the network.

While the naming service proposed in this paper may fit into the broad goals of the Zero Configuration Working Group, it has several key differences. The most obvious one is that the ZEROCONF requirements are designed to work with a network composed of entirely client devices, with no service providers or managers in the network. By design, the agent program is run on a network manager, and provides DNS services for the entire network. While the ZEROCONF requirements state that hosts need no previous configuration, they do rely on more complete solutions such as DNS and DHCP for long term operation and scalability. However, the ZEROCONF Working Group has set no requirements that these servers be self-configuring and self-administrating. This is precisely the problem that the agent program attempts to solve. In a network managed by the self-configuring naming service described in this paper, both hosts *and* managers are administrator free and may join and leave the network freely. Therefore it is possible to have a network that runs the agent program on the DNS servers and also meets the ZEROCONF requirements. Hosts could use the ZEROCONF protocols to obtain an address until the discovery of a manager. Once a manager is found, the host is free to resolve any name on the network using standard DNS calls.

4.2 Non-IP Networks

The requirements laid out by the ZEROCONF Working Group stress the importance of having computers networked together "just work" in the absence of service providers such as DNS and DHCP [17]. Two protocols in use today provide this level of functionality. The AppleTalk suite of protocols is simple to operate in small networks. Plugging a group of Macs into an Ethernet hub will get one a working AppleTalk network without the need to setup specialized servers like DNS [15]. As a consequence, AppleTalk networks can be used in homes, schoolrooms, and small offices -- environments where IP networks have been absent because they are too complicated and costly to administer. NetBIOS provides similar functionality and ease of use on Microsoft Windows machines.

However, because nearly all computers used today are connected to the Internet, they also require TCP/IP to be configured, as this is the protocol of the Internet. Therefore the benefits of AppleTalk and NetBIOS are overshadowed as application developers will need to support two protocols: TCP/IP to access the Internet, and either AppleTalk or NetBIOS to access the local network. This is the motivation behind our research as well as the ZEROCONF Working Group and other efforts to make the IP suite of protocols simple to configure. Allowing developers to concentrate on one protocol for all communications will make application development simpler and more efficient. Although the IETF has plans to eliminate the need for the AppleTalk and NetBIOS protocols, the design of each is relevant to this paper as it impacted the design of the agent naming system.

4.2.1 AppleTalk

AppleTalk follows a decentralized architecture with the goal of "plug and play" capability, wherein a user can plug in a compatible device into the network and have full access, without any associated configuration [2]. AppleTalk accomplishes name to address translation with the Name Binding Protocol (NBP). Each device on an AppleTalk network is assigned an entity name of the format: Entity:Type@Zone. The type specifier allows a host to determine what higher level protocol to use when communicating with the device, while the Zone field allows devices to be grouped together in logical clusters. Zones may include a number of nodes from different networks (i.e. nodes behind different routers). The NBP provides a mapping from these entity names to internet address[3]. Each node on the network will maintain a names table containing entity to internet address mappings. The names directory (ND) is a distributed database which stores all the entity to internet address mappings in the system. NBP does not require the use of name servers, however, it will allow the use of name servers if they are present [2].

When a host wishes to lookup the address of an entity, NBP is used to search the ND for the appropriate name. If the entity does not exist in the local ND, a request will be sent over the network to find the address. If the name in question resides in the same zone as the requester, a name lookup packet is broadcast over the network. To perform a lookup of an entity in a different zone the Zone Information Protocol (ZIP) is used to determine the zone of the requested entity, and then a directed broadcast is sent over that zone [43]. ZIP uses a zone information tables to determine which zone an entity belongs to. These tables must be stored on every router in the system.

Although AppleTalk provides decent performance on small to medium sized localized networks, it cannot scale to meet the needs of large networks such as the Internet. The use of broadcast packets to perform name lookup is inefficient when compared standard DNS lookups. While a node broadcasts the name lookup to every device on the network, it only accepts one response. Because the agent name system uses standard DNS queries, a name lookup consists of one request and one reply.

[3] Note that internet here is simply a collection of networks; internet addresses are not IP addresses, but a way of uniquely addressing a node across AppleTalk networks.

Another problem is the way AppleTalk stores the ND and zone information table. Because the ND is stored on all nodes, each node replicates data that is available on every other node in the network. Also, as the number of hosts in the network grows, this flat directory could become large, and every time a host joins or leaves the network, every host must be notified. Keeping the zone information table consistent across all the routers may also prove difficult when network topologies change.

The agent naming system presented in this paper exhibits none of the above weaknesses. Because the DNS is used as the name lookup scheme, scalability is not an issue, as DNS is proven to scale up to networks with millions of hosts. Also the managers handle the case of changing network topologies and can adapt the domain names and automatically update the DNS to prevent naming conflicts.

4.2.2 NetBIOS

NetBIOS allows several computers that share the same broadcast medium to locate resources, establish connections, send and receive data with an application peer, and terminate connections [37]. NetBIOS defines an interface for the above operations, but relies on lower level protocols for operation. The most common today is NetBIOS over TCP, which is what will be discussed below.

Devices using the NetBIOS service are identified by dynamically assigned names. The NetBIOS name space is flat and allows up to sixteen alphanumeric characters. For an application to acquire a name it must first perform the name registration process. During registration, a bid is placed for a name and broadcast over the network. The bidding process occurs in real time and is necessary for all applications that wish to use the NetBIOS service. Implicit permission for the name is given when no objections are received from other nodes on the network.

The NetBIOS service defines three types of nodes on a network. Broadcast nodes ("B"), point to point nodes ("P"), and mixed modes ("M"). Each node has a different method of name resolution. If the network consists of entirely of B nodes, then broadcast messages are used for name to address resolution. If the network contains P or M nodes, however, then a NetBIOS Name Server (NBNS) is required to be present on the network. The NBNS handles name lookups for a NetBIOS network. The NetBIOS specification allows for extreme flexibility in the implementation of the NBNS [37]. The NBNS can simply act as a "bulletin board" on which name and address information can be freely posted, or it may assume full control over the validation and management of the names. P nodes rely solely on the NBNS for address lookups, while M nodes will first broadcast queries and then contact the NBNS if no response is returned. At any time, either all the B nodes or the NBNS will have knowledge of the names on the network.

The most obvious problems with the NetBIOS service are issues of scalability. With a flat sixteen character name space, many name conflicts may occur in large networks, particularly when networks are merged together. Also because B and M nodes require the use of broadcasts to register and lookup names, the amount of bandwidth used for address resolution will be much higher than what is needed for a DNS query. Because the agent name service relies on standard DNS implementations, it allows for greater scalability and efficient bandwidth use.

The use of a properly configured NBNS in the NetBIOS environment with only P nodes may solve some of these problems. In particular, a DNS server may be used as a NBNS [37]. This would reduce the number of broadcasts in the network, as lookups would only require one query and response. However, the flat name space issue still remains.

4.3 Easing DNS Administration

The above solutions were all replacements for a full DNS system; either by using other methods of name to address mapping over IP or using different protocols altogether. However, because IP networks and DNS have been integrated into nearly all modern network systems, there have been other efforts to ease the configuration and administration of DNS.

4.3.1 Nssetup

Researchers in Japan have attempted to tackle DNS administration problems by simplifying configuration tasks and eliminating repetitive tasks through automation [14]. They developed a program with a graphical interface that reduced the work of a DNS administrator. Their tool, called *nssetup*, automates repetitive tasks such as generation a DNS database file from the machine's host file, keeping the root cache file up-to-date, and maintaining the reverse address lookup table. To check the correctness of the configuration, *nssetup* contained a feature that checked if the name server was up and running. In addition, *nssetup* provides a graphical user interface for configuring the resolver and adding new hosts into the database. The *nssetup* developers showed that it was considerably faster to configure a name server using *nssetup* than without it. They state they have reduced the configuration time from two hours to three minutes.

However, *nssetup* does not truly reach the goal of zero administration. It simply provides a nice user interface and a good set of default configuration values for BIND.

Every time new DNS servers are added, an administrator must configure them as well as every DNS server already running on the network so that they are properly integrated into the network. The agent program requires no prior configuration when DNS servers are added. Simply starting the agent program is all that is needed to be done; it will locate other managers on the network and they will configure themselves accordingly, all without user intervention. The time is takes for the agent program to configure a name service is less than the time using *nssetup*, and does not require a human administrator.

4.3.2 Commercial Solutions

There are several commercial products available today that aim to simplify the operation and administration of DNS servers. Most provide solutions that include both DHCP and dynamic DNS in a single software package. These products are aimed at corporations that run a large intranet with their own DNS servers. By including both dynamic DNS and DHCP, they allow hosts to freely join and leave network with no manual configuration. DHCP will assign the new host an IP address, and send DNS *update* signals to notify DNS server of the new host so that it may be located on the network. Figure 4 shows the integration between DNS and DHCP in the commercial solutions with the addition of remote administration. Some popular products are described in more detail below.

4.3.2.1 Lucent QIP Enterprise 5.0

Lucent's product provides Enterprise based IP network planning in addition to integrated DHCP and DNS services [25]. The DNS server is based upon BIND but adds several other features such as remote administration. QIP uses one centralized database to store all updates for recovery purposes. It also provides enterprise tools such as user profile management and directory services.

QIP is componentized set of services that can be turned on and off. For example, a third party implementation of DNS and DHCP could be used along with the QIP profile management features. Therefore it would be possible to use the agent name system to provide the DHCP and DNS services for the network managed by QIP.

QIP provides a fault tolerant design using replication and by eliminating any single source of failure. The QIP DHCP server can update both primary and secondary DNS server's resource records as DHCP leases are granted and deleted. This allows for *update* messages to be handled properly even when the primary server is unreachable. In the primary server's absence, a secondary server will receive the updates. When the primary again becomes reachable, they will be passed along.

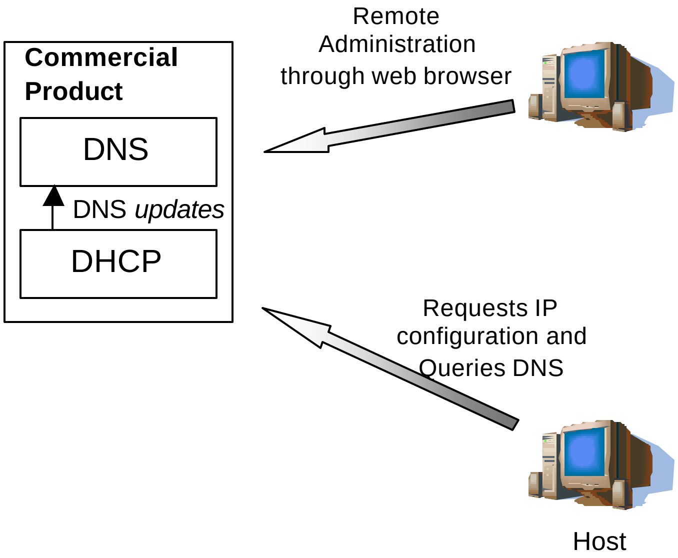

Commercial Product
DNS
DNS *updates*
DHCP
Remote Administration through web browser
Requests IP configuration and Queries DNS
Host

Figure 4 - Integration between DHCP and DNS

4.3.2.2 Check Point Meta IP 4.1

Like QIP, Meta IP provides an integrated service that includes DNS and DHCP [26]. BIND is again used as the basis for the DNS service. However, the Meta IP does include some changes to the standard DNS protocol. Specifically, the developers of Meta IP developed a proprietary way for primary and secondary servers to notify other slaves of zone updates. However, recent BIND implementations support the *notify* message which provides similar functionality [47]. Because the agent name service runs BIND, it also supports this feature.

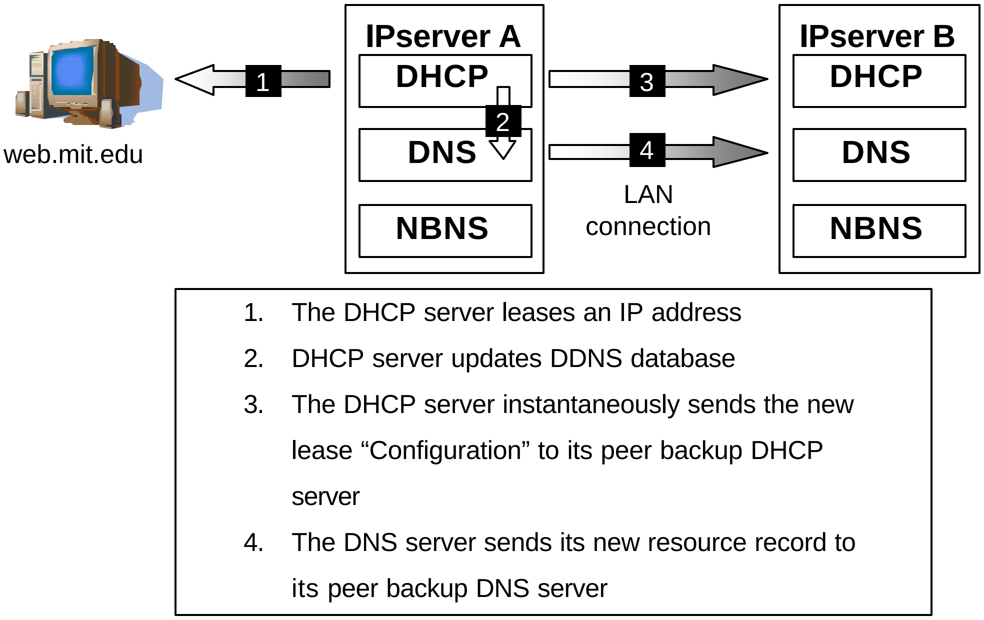
Another modification to the DNS system that Meta IP supports is that it allows for slaves to process any update messages received when the primary is unreachable. Thus, if the primary server is unavailable, updates to the zone will be maintained. While this is an improvement to standard DNS, this problem is also handled by the agent name system. If the primary goes down for any zone, its slaves will receive *leave* messages and they will negotiate amongst themselves to elect a new primary. Any updates to the zone will then be handled by the new primary.

Meta IP also includes several features that make it more enterprise friendly. It provides remote access services that track users to the addresses they login from, user login authentication services, and logon monitors. Meta IP, like QIP, uses a centralized data store to allow for centralized administration of all services.

4.3.2.3 NTS IP Server

In addition to providing integrating DNS and DCHP, IP Server also includes a NetBIOS NBNS. The goal of the IP Server is to provide a single solution that streamlines the administration of the above three services [40]. To reach this goal, NTS have added two extensions to the DNS protocol.

The first DNS extension developed by NTS allows for servers to have “peer backups”. These are essentially duplicates of the server, and unlike secondary servers, they are always consistent with the main server (Figure 4). Essentially, “peer backup” allows for more than one primary server. Thus if one fails, the other is available for updates. Peer backups share a consistent database of resource records. When one server receives an *update*, it is instantly transmitted to its peer. If one server should become unreachable, the other will continue to operate, and upon successful revival of its peer, it will transmit all changes it has received since communication terminated between them.



1. The DHCP server leases an IP address
2. DHCP server updates DDNS database
3. The DHCP server instantaneously sends the new lease “Configuration” to its peer backup DHCP server
4. The DNS server sends its new resource record to its peer backup DNS server

Figure 4 - Peer Backup IPservers (from [9])

The other DNS extension allows for zone “co-serving”. A zone can be split into a number of pieces, each served by a different server. However, any of the servers may be queried and appear to be primary masters for the zone. They will communicate amongst each other and return the correct answer to the resolver. For larger zones, this may lead to a performance improvement as the load can be balanced amongst a number of different servers. In addition co-servers may be deployed geographically near to clients, thereby partitioning and localizing DNS traffic [40]. Figure 4 illustrates the operation of DNS co-serving.

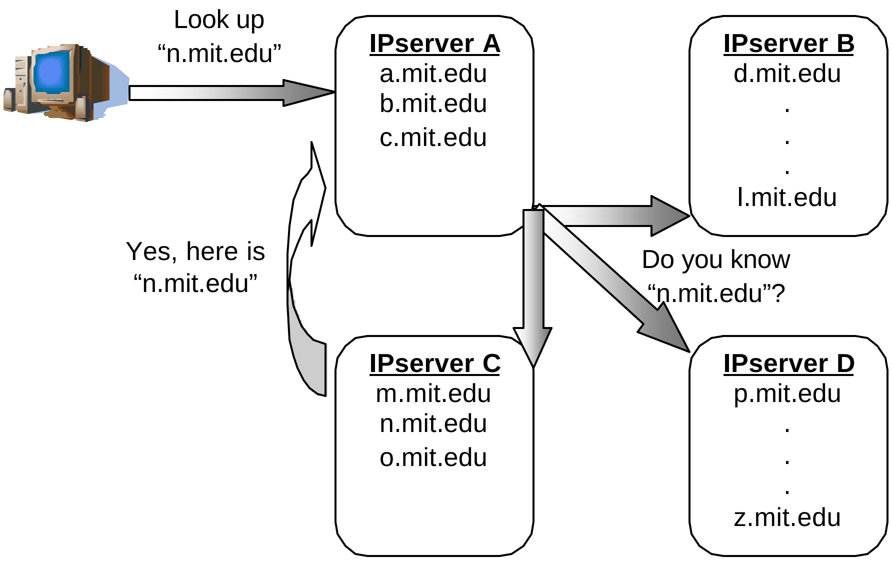


Figure 4- IPserver co-serving for the mit.edu zone (from [9])

IP Server is not based on any version of BIND and is instead built from the ground up to provide these services. The extensions created by NTS for DNS allow them to improve reliability and performance of their DNS system. However, these extensions are not Internet standards, and only those servers running the NTS IP Server will be able to use them. The agent name system can match many of these features while at the same time conforming to Internet standards. Primary backup is achieved in the agent name system by immediately promoting one of the slaves to the role of primary master. The new master may then handle any updates to the zone’s resource records. Neither the agent name system nor BIND provides any mechanism to “co-serve” a zone. The concept of “co-serving” is directly in conflict with DNS standards [29] as they state that every zone may have exactly one primary master. While “co-serving” may provide some performance increase, as updates may now be handled by many servers instead of just one, we believe it is more important to stay in compliance with existing standards.

4.3.2.4 Comparisons to the agent name system

In addition the products described above, there exist other similar solutions to ease the administration and operation of DNS [20][32]. Nearly all of these vendor products offer features that appeal to the corporate environment. These features include user profile support, directory services, and remote administration. While these features are certainly advantageous in the corporate environment, they do not pertain directly to the name system. In addition, the goal of these commercial solutions is to provide a central point of administration for the entire network.

In contrast, the agent name system takes a decentralized approach to the administration of the network. No one manager is in charge of the entire management and most network operations such as creating a new zone and electing a new master require communication amongst multiple managers. The decentralized approach allows for the relevant portions of the network to be involved in the decision making process. Table 3 illustrates the different features of each commercial solution as well as the features offered by the agent name system.

Table 3 - Comparison of Commercial Solutions to the Agent Name System

	Agent Name System	QIP 5.0	Meta IP 4.1	IP Server	Optivity Net ID
Configuration free					
Administrator free					
Allow hosts to freely join and leave network					
Allow name servers to freely join and leave network					
Based on BIND					
Properly handle *update*s while primary is unavailable					
Handle failure of slave server with no intervention					
Does not use propriety DNS extensions					
Decentralized administration					
Centralized administration					
Enterprise features (profile support, directory service)					
Remote Administration					
GUI for Administration					
Configuration error detection					
Built in DNS security					

Another key difference between the agent name system and those described in this section is the procedure for introducing new name servers into the system. Both the commercial systems and the agent name system require no administrative input when hosts join and leave the network. However, only the agent name system allows for servers to do the same. In all of the above products, extensive reconfiguration is required when a server exits or joins the network.

While most of the commercial products provide a method to handle *update* messages while the primary server is down, none of them provide a permanent solution like the agent name system. They simply store the *updates* until the primary comes back on line or allow the secondary server to process them. However, if the secondary also fails, the updates will be lost. In contrast, the agent name system quickly promotes a new server to primary allowing all *update*s to be recorded permanently. Networks managed by the commercial solutions must be carefully planned so that the master/slave server relationship exists for all zones. Servers must be configured so that they are aware of the locations of the other servers. In the best case, the commercial products provide a human administrator a nice graphical user interface for performing the necessary configuration. In the worst case, manual editing of the configuration files is needed. Whatever the case is, the agent name system provides a simpler solution. No human administration is necessary when name servers exit or join the network, as the managers will communicate amongst themselves and adapt dynamically to the changing network topology.

Chapter 5 Improvements to the Self-Configuring and Self-Administering Name System

Although Feng's design has several desirable attributes, it is not perfect. In particular, any network managed by the agent name system cannot communicate with an outside network. Also, security concerns seemed to be an afterthought, as no security measures were built into the system. This chapter outlines several improvements to the agent name system that will solve the above problems as well as increase overall system performance.

5.1 Compatibility with other networks

The current implementation of Feng's agent program is designed to only handle names and addresses in the domain managed by the agent name system. Presently, there is no requirement or method for hosts to communicate with other networks not under control of the agent system, such as the Internet.

In order to enable interaction with other networks, the internal root server must be configured to manage only a portion of the DNS hierarchy and not the entire tree. Each internal root will take a special domain name like *autonomous.mit.edu* and manage the namespace in that domain. The basic operation of the agent name system will remain the same, as agents in that domain would only need to be aware of *autonomous.mit.edu*'s address to enter the Configured state.

However, when the network managed by the agent name system is connected to the Internet, hosts will need the ability to resolve Internet names. Therefore, every agent name service in the system must be able receive Internet hostname queries and respond with the correct address. To provide this functionality, each agent name service needs to know the location of an Internet root DNS server. It can then direct the query to the root server and recurse down the DNS tree as necessary until the correct answer is found. Because the locations of the Internet root DNS servers are static, this can be pre-

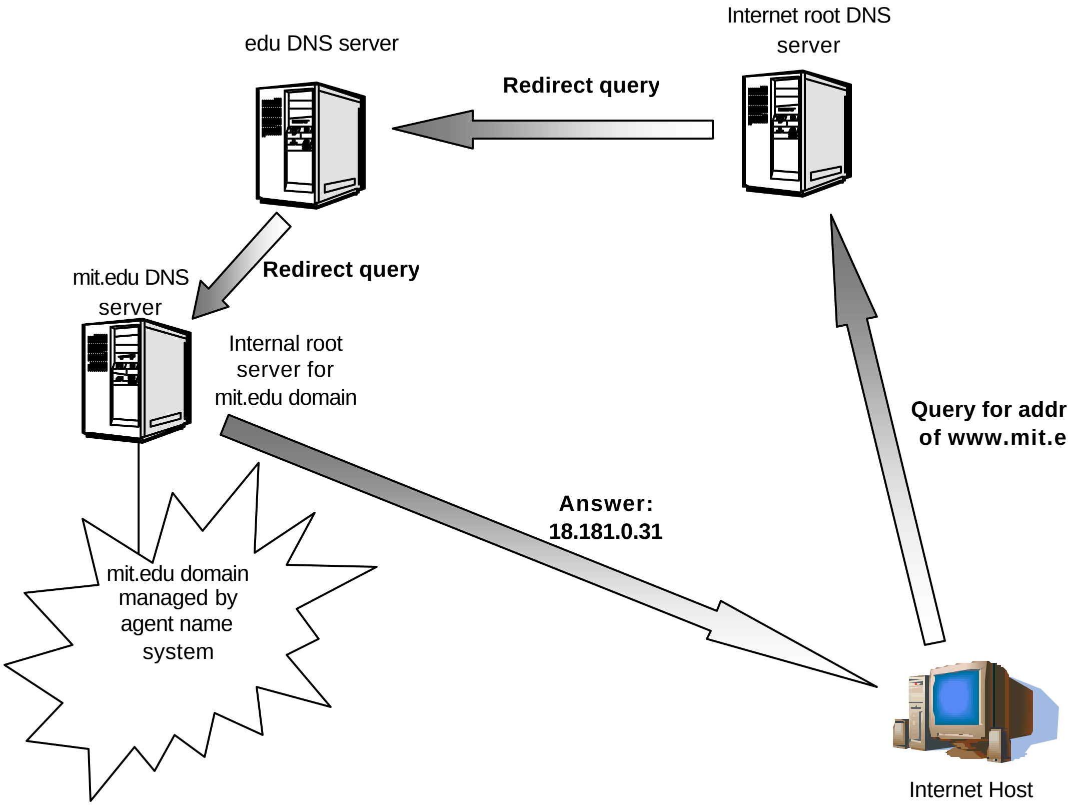


Figure 4 – An Internet host resolving the address of a host in the agent system

configured or built into the agent name system. With this information, any agent in the system may resolve any hostname on the Internet.

It also necessary for hosts on the Internet to be able to resolve the names of hosts managed by the agent name system. To do this, the agent's internal root server must be properly registered in the Internet's DNS hierarchy. This way any host on the Internet may query the Internet root server for an address managed by the agent name system. That resolver will be redirected down the DNS tree until it reaches the internal root server of the agent name system network, which will then return an answer or direct it to a manager in the system which is authoritative for that zone (see Figure 4).

5.2 DNS Messages

As described in Section 3.4 the agent program acts a filter for each DNS message sent to the BIND process. If the DNS message is an *update*, the agent program sends it to the appropriate manager or creates a new zone. For every other DNS message the agent program simply forwards the message to the local BIND process, waits for the result, and directs the result back to the original sending host.

This approach will suffer performance degradation when compared to normal BIND implementations. This is because the agent program must first examine every DNS message to see if its an *update* message. Then if the message is not an *update*, it has to log the sender of the message, forward the message to BIND, examine the answer, lookup the sender of the original message, reconstruct the message, and finally send it back to the correct host. This is an addition to the work that BIND will perform on the message it receives, such as looking up the address from its database, or querying other DNS servers. Comparatively, the number of *update*s versus the total number of DNS messages is very low, as any network will have much more queries than host updates. Under heavy load, this design of the agent name system will fail before standard implementations of BIND, due to the extra overhead of filtering every message.

Because the number of *update* messages is dwarfed by the total number of DNS messages, ł would be ideal if the agent name system could respond to these messages as fast as a standard BIND implementation. This can be accomplished by allowing the local BIND process to listen on the standard DNS port, port 53. Therefore, hosts in the network will query the local BIND process directly, bypassing the agent program altogether. Port 54 is still used for inter-agent communication.

This solution will provide equal performance to standard BIND, because the agent is not involved in any DNS messages. However, it is still necessary for the agent program to receive the *update* messages, as these need to be processed as described in Section 3.4.1. BIND may not directly receive *update* messages because if the zone does not exist, it will return an error to the *update* sender, instead of creating a new zone.

The DNS specification states that *update* messages may come from hosts, DHCP servers, or other DNS servers [48]. However, in a network managed by the agent name system, only other managers and the IP configuration process are allowed to send an *update* message to an agent. Hosts must first register with the IP configuration process, and that process will relay the information to the agent program. Other *update*s will be ignored. Therefore, we can require that all *update* messages be sent to a manager's port 54, where the agent program may examine the message before taking any action. Any action the agent program takes will be according to the procedure described in Section 3.4.1. The only change is that it will now listen on port 54 for *updates* and when necessary, it will forward the *update* to the local BIND process running on port 53. Figure 4 shows the new interaction between the agent, BIND, and other processes.

Secure DNS *updates* (see Section 5.4.2) may be used to ensure that only *updates* received from the trusted local IP configuration service or another manager are processed

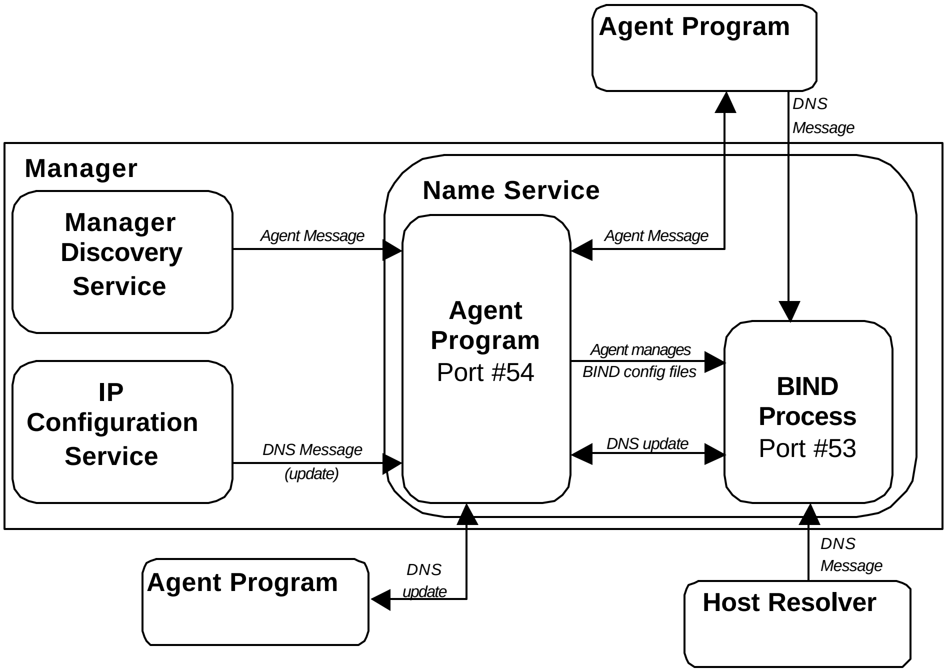


Figure 4 - Relationship between BIND, agent program, and other processes. *Update*s are now the only DNS message handled by the agent.

by the manager [50]. BIND will therefore refuse any *update* message received on port 53 coming from a host, or untrusted manager.

This change keeps all existing functionality of the agent name system, including dynamic zone creation, while increasing the performance to be equivalent to standard implementations of BIND. It also decreases the complexity of the agent program, as it no longer needs to deal with other DNS messages besides *update* messages or listen on two ports simultaneously.

5.3 Agent Messages

In order to fully optimize the performance of the agent program, the agent messages must also be reviewed. By changing the message payload or the agent's response to the receipt of the message, it may be possible to increase performance, while maintaining the current state of functionality.

5.3.1 *becomeslave*

When a primary master has less than the required number of slaves needed to serve its zone, it calls the getSlave function. getSlave adds new slaves by sending random known servers the *becomeslave* agent message. Managers which receive the *becomeslave* messages reconfigure their BIND process to act as a slave for the new zone, regardless of the current number of zones they already serve. This could lead to a single manager being authoritative for a large number of zones and therefore increasing its load, while other managers serve only a single zone with a light load. The optimal solution would be to distribute the load evenly across all managers.

If agents are permitted to decline *becomeslave* messages, the system could come closer to achieving the goal of an evenly distributed load. When an agent is already serving a number of zones and is at near capacity, it could respond to a *becomeslave* message with a decline message. Then the manager sending the *becomeslave* would pick a new agent to become its slave. However, if the requesting manager sends a *becomeslave* message to all other agents and they decline, then it may send a *forceslave* message to an agent which cannot be declined. This would ensure that a primary server would have a backup.

The *becomeslave* message, however, is not only used by the getSlave function. It is also used when a new primary server is elected. Every slave of the old primary master is notified by the new master with a *becomeslave* message. This message should not be declined, as it would result in a incorrectly configured agent. To prevent this, the agent can use *forceslave* messages instead of *becomeslave*. The *forceslave* message would be used in every case that *becomeslave* was used previously, except for the getSlave function. The getSlave function would be the only sender of *becomeslave* messages.

5.3.2 *negotiatemaster*

The *negotiatemaster* message is used when two different managers believe they are primary master for the same zone. The two conflicting servers exchange *negotiatemaster* messages and elect a new master based on message's payload. The payload includes the number of slaves the server currently has and the server ID of the sender. Currently, the manager with the most number of slaves will be the new master. If both managers have the same number of slaves, then the server with the higher ID will be elected.

When the servers have the same number of slaves, then the computer that is best equipped to handle responsibility for the zone should be elected. "Best equipped" can be defined by such statistics as CPU speed, available memory, etc. If these statistics are included in the payload of the *negotiatemaster* message, the conflicting servers can use them when making their decision. This ensures that the computer with the most available resources will be elected as the primary master for the zone, and most likely leading to an increase in performance.

5.3.3 *discover*

Discover messages are used when a new agent joins the system. Every agent in the system will receive two *discover* messages whenever an unconfigured agent joins the network. The first will come when the manager discover process discovers the new agent. A configured agent receiving this *discover* message will update its state information with the new manager's information. Then when the new agent is configured, a second *discover* message will be received by each agent on the network. This message informs them that the new agent is configured and will include the newly configured agent's *server record*. These discover messages generate a nontrivial amount of network traffic and could have negative impact on system performance.

In [13] Feng suggests a solution that will cut the number of *discover* messages received for each new manager from two to one. He proposes that each new agent enter the network configured as the internal root server. Only after the new agent hears of an existing internal root, will it change its configuration. To do this, it will negotiate with the internal root server using the previously discussed *negotiatemaster* message. Once the conflict is resolved, it may again begin servicing name requests.

Surely this is an improvement upon the existing implementation, as each new agent would require each configured agent to receive only one *discover* message. However, the negotiation between the conflicting roots also requires a nontrivial amount network traffic. This is especially true when one master is demoted to slave and all its previous slaves must be notified of their new master.

A better solution is to loosen the requirement that every manager receive a *discover* message whenever a new agent is introduced to the system. The *discover* information should be passed on a "need to know" basis. That is, only those managers that will directly interact with the new agent should be informed. This drastically reduces the number of *discover* messages, and also avoids the extra traffic involved when negotiating a new master.

To accomplish this, we must first define which servers "need to know" of an agent's arrival or departure. Clearly, the internal root server should be notified, as it keeps an accurate view of the current network topology. In addition, master servers should be aware of every slave for the same zone. Likewise, slaves should be aware of the primary master for their zone.

When a new manager is introduced into the network, its IP configuration service will notify it of any other managers it finds. If no other managers are found it will assume it is the only manager in the system and configure itself as the internal root server, as described in Section 3.7.1. If it locates other managers on the network it will also behave as described in Section 3.7.1. The only difference is that existing managers will not propagate the knowledge of the new agent. The new agent will send a *getroot* message to one existing manager and then configure itself with the location of the internal root server and send a *discover* to the root server. Upon receipt of the *discover* message, the internal root will record the new *server record* in its state information. The internal root server always has an accurate view of all the managers in the system.

Because individual managers are not aware of every agent in the system, the getSlave function may not operate correctly. This is because the agent may not know of any other servers that are not already slaves for its zone. Therefore, in order to find more slaves, the agent must contact the internal root server and request *server record*s for other managers.

The internal root server will now have the responsibility of detecting conflicts.

When a new manager joins the system and claims to be primary master for the same zone as another existing manager, the internal root will recognize this, and send a *discover* message to the both primary masters. The two conflicting managers will then resolve their conflict in the same manner as described in Section 3.5.

5.4 Security Concerns

The current implementation of the system has no security mechanisms built in. Securing a network designed to be configuration-free and administrator-free creates a conflict, as any meaningful security mechanism will require some sort of configuration [51]. The best that anyone can hope to accomplish is to minimize the configuration overhead necessary to keep the system secure. For a closed environment such as a corporate intranet, perhaps the easiest security model is to have none at all, simply relying on physical security mechanisms to control access to the machines [46]. This will work well for an environment that is well controlled and where one can physically control access to the network. If a corporation only allows approved machines and approved users to plug into the ethernet, it is impossible for an outside intruder to gain access to the network.

However, as wireless networks become increasingly popular, controlling physical access to networks becomes impossible [15]. Insecure wireless networks allow anyone in range to send and receive data. This allows for theft of services (such as an individual using his or her neighbor's internet connection), as well as the potential for malicious hosts to be introduced to the system.

Another reason for incorporative security capabilities is when the network managed by the agent-naming system is connected to another larger network, such as the

Internet. If the system is left insecure, any host in the world could attempt to violate the network's security.

When a network is managed by the agent naming system, it should be *at least* as secure as a standard IP network. That is, the agent naming system should not open up any new holes into the DNS system. Currently, most DNS implementations are completely insecure, instead relying on redundancy and physical security as the best weapon from attack [50]. DNS servers are kept in a trusted, safe location and distributed throughout the network. Normal DNS servers only allow administrators at the physical machine to change the configuration of the DNS server. However, because the agent naming system allows other agents to change its configuration, it is prone to attacks from outside. If a malicious user could introduce a "rogue agent" into the system, it could do significant damage.

The managers in the agent naming system have two main functions. First, they provide IP configuration information to new hosts that join the network, and second they provide the name to address resolution service for the network. Each of these services is vulnerable to attack, and for a fully secure network, both must be secured.

There are two possible methodologies for securing the managers. The first is to use IPSec to provide network layer security for all traffic [21]. The second is to secure the individual protocols that provide IP configuration and name to address resolution. Both are discussed in the following sections.

5.4.1 IPSec

RFC 2401 defines the protocol and motivation for IPSec. It states:

> “IPSec is designed to provide interoperable, high quality,

> cryptographically-based security for IPv4 and IPv6. The set of security services offered includes access control, connectionless integrity, data origin authentication, protection against replays (a form of partial sequence integrity), confidentiality (encryption), and limited traffic flow confidentiality. These services are offered at the IP layer, offering protection for IP and/or upper layer protocols."

IPSec uses two protocols to secure traffic: Authentication Header (AH) and Encapsulating Security Payload (ESP). For more information consult their respective RFCs [21][22]. The combination of these two protocols allows the user to select the granularity at which security service is offered. IPSec uses shared secret values (cryptographic keys) to encrypt and authenticate data. It relies on separate mechanisms to distribute these keys among hosts. This may be done manually or through an automatic key distribution method like IKE [16]. For environments where a vendor controls the hardware used in the network (such as corporate intranets, or even cellular networks), the key may be pre-installed by the hardware manufacturer. This would preclude the need for an automatic key distribution method. However, automatic systems, while being more complicated to implement, allow for more flexibility in the network. A multiple of different hardware devices could be added as long as they conform to the appropriate key distribution protocol. If the key is pre-installed in hardware, it may need to be updated manually if for any reason it becomes compromised. Key distribution is a significant problem in securing any network, particularly those designed for zero-configuration and administration. Other possible approaches include [51]:

- Using an "out-of-band" mechanism to transfer security, such as barcodes, smart cards, etc.

- Diffie-Hellman key agreement [23], with an additional form of external authentication to prevent man-in-the-middle attacks.
- A public key certification approach (such as the DNS extensions, described in the next section)

Steve Hanna discusses various secret sharing schemes in more depth in his internet draft [51]. For the remainder of this thesis, I will assume that there is a mechanism in place for distributing keys and concentrate on the security protocols that use these keys.

Using IPSec, all traffic between managers is authenticated and optionally encrypted [21]; this ensures that no malicious user can place a rogue manager into the network, for it will not be able to authenticate itself. Also hosts can authenticate managers, so that when they receive their IP configuration, they can make sure it is correct and that it came from a known manager. The alternative to using IPSec is to secure the existing IP configuration protocol and name to address resolution protocol.

5.4.2 Securing the individual protocols

The managers in the agent based naming system rely on DHCP to perform host IP configuration. Normal DHCP operation provides no security. Hosts discover DHCP servers through directed broadcasts on the local network and the DHCP server replies with an available IP address. This open model allows for several attacks. The inherent threat to any DHCP server is an insider threat. This is because when routers are configured correctly, DHCP packets only flow on the local medium. Thus, the DHCP server and host must be on the same local network. The main threat to a DHCP client (a host that is requesting information from a server) is a rogue DHCP server introduced to the network. This server can be configured to give false information, reeking havoc on the state of the network.

To prevent such attacks the IETF has proposed a standard for authenticating DHCP messages [8]. Their proposal defines a technique that can provide both entity authentication and message authentication. It uses a combination of the original Schiller-Huitema-Droms authentication mechanism along with the "delayed authentication" proposal developed by Bill Arbaugh [8]. Like IPSec, they assume a mechanism for distributing shared secrets. [18] also describes a DHCP authentication mechanism using Kerberos.

Using the proposed standards, a host can be assured that his configuration is coming from a reliable source and it can be trusted. Also, the DHCP can authenticate a client if necessary. In an open network environment, this is not necessary as any host may join. However, if the administrators choose to only allow access to certain hosts, this would allow the DHCP server to detect invalid clients masquerading as valid clients.

One threat that DHCP authentication does not address is denial of service attacks. These attacks can involve the exhaustion of valid addresses, or the consumption of all available network or CPU resources. This vulnerability is present anytime there is a shared resource available. The best current solution to such attacks is to provide redundancy of services.

Every manager in the network also provides naming services via DNS. DNS is also an insecure protocol, though there are configurations in use today that can protect themselves from many attacks [10]. In order to fully address the insecurities in DNS, the IETF has standardized the DNS Security Extensions (DNSSEC) [11]. DNSSEC provides three distinct services: key distribution, data origin authentication, and transaction and request authentication. DNSSEC was not designed to provide any sort of confidentiality for queries or replies. This is because the designers feel that DNS data is public and that DNS should give the same answers to all inquirers. Similarly, no attempt was made to implement any sort of access control lists for DNS. If confidentiality is needed for queries (such as DNS *updates*) this can be provided by the IPSec protocol.

To implement the key distribution system into DNS, DNSSEC defines a new resource record (RR) to associate keys with domain names. This allows the DNS to function as a public key distribution mechanism for either DNS security itself or other security protocols (such as IPSec). Thus to find the public the key of any host in the network, a client must simply query a known DNS server.

Data authentication is provided by associating cryptographically generated digital signatures with resource record sets. Usually there is a single private key that authenticates each zone [11]. If a resolver reliably learns the public key of a zone, it may authenticate any signed data read from that zone. Note that the keys are not associated with servers, but zones. So data received from the slave server or master server is signed by the same key. Resolvers may obtain public keys of zones from either DNS or by having it statically configured beforehand. To read a key from DNS reliably, the key itself must be signed with another key that the resolver trusts. To start, a resolver must be configured with at least one zone's public key. Then it can use that key as a starting point and reliably read keys for other zones if they are stored in the DNS.

DNS transaction and request authentication maintains an ability for a resolver to be assured that the authenticated response it got from the DNS was a reply to the correct query. This is accomplished by the DNS returning the concatenation of the query and response signed with the entity's digital signature. Request authentication also allows a host to be authenticated by the server. This is only useful in the case of secure DNS updates.

Thus, in order to provide a secure implementation of the agent naming system, DHCP authentication and DNS security extensions are needed. The minimal configuration needed for this system is every host needs the public key of its zone. Then with this public key, it can authenticate the DNS, and receive new reliable keys. When a new host is added to the system, it is authenticated with DNS and secure DNS updates are sent to the appropriate manager.

Additionally all non-DNS communication between managers (*discover, leave* messages, etc.) should be sent strictly over IPSec. This ensures every manager in the network may be authenticated. To accomplish this, all managers in the system must share a secret. It is not unreasonable to assume that this be done out-of-band, as all managers in the same network should be controlled by some central authority (either a person, or corporation). Thus, they will have methods in place to distribute the shared secrets reliably and securely.

These individual secure protocols may prove to be lighter weight than IPSec; however, they do not provide confidentiality of data. If confidentiality of data is desired, IPSec must be used to encrypt all data. IPSec may also be easier to implement as it is a broad solution covering all network traffic. The only requirement is that host’s and manager’s TCP/IP stack support IPSec. When IPSec is not used for security, any additional services added to the system must provide their own mechanism and protocol for authentication.

Chapter 6 Conclusion

DNS is an essential service on modern networks of any size. However, because of its complexity, administering DNS can prove quite difficult. Nearly all corporations today that run an internal intranet dedicate full-time personnel to this job. It is because of this complexity that much current research is devoted to simplifying DNS administration.

The implementation of the agent program successfully solves the problem of providing a scalable and fault-tolerant way to store name-to-address mappings and to handle name-to-address resolutions in the absence of human configuration or administration. The agent program handles the configuration and administration of a DNS name server implementation called BIND, and BIND stores name information and answers name queries with scalability and fault-tolerance in mind. In comparison to approaches proposed by other researchers, the agent name system offers superior functionality for large networks and can also be more easily integrated with existing Internet solutions. With the improvements of Chapter 5, the agent name system provides equal performance to standard BIND implementations, while at the same time providing superior functionality.

In particular, the improved agent name system requires no manual configuration or administration, a feature that other DNS implementations cannot match, commercial or otherwise. A procedure for securing the agent name system using existing standardized protocols is also presented in this thesis.

The improved agent name system presented in this thesis offers solutions for many of today's networks. It offers a simple solution for homes and small businesses that can't afford a permanent IT staff to administer their network, while being scalable enough to handle networks of much larger size and complexity.

Chapter 7 Bibliography

[1] P. Albitz, C. Liu, *DNS and Bind*, 3rd Edition, O'Reilly, 1998.

[2] *AppleTalk Network System Overview*, Addison-Wesley Publishing Company, Inc., 1990.

[3] *BIND Administrator Reference Manual*, Nominum BIND Development Team, January 2001.

[4] H. Chao, T.Y. Wu, S.W. Chang, R. Wang, "The Network Topology Based Domain Name Service", *Proc. of Int. Workshops on Parallel Processing*, pp. 214-219, 1999.

[5] S. Cheshire and B. Aboba, "Dynamic Configuration of IPv4 Link-Local Addresses", draft-ietf-zeroconf-ipv4-linklocal-02.txt, March 2001. (work in progress)

[6] M. Crawford, "IPv6 Node Information Queries", draft-ietf-ipngwg-icmp-name-lookups-07.txt, August 2000.

[7] R. Droms, "Dynamic Host Configuration Protocol", RFC 2131, March 1997.

[8] R. Droms, ed., "Authentication for DHCP Messages", draft-ietf-dhc-authentication-16.txt, January 2001. (work in progress)

[9] "Dynamic DNS, Infrastructure essential for today's intranets", http://www.nts.com/collateral/ddnstechpaper.pdf

[10] D. Eastlake, "DNS Security Operational Considerations", RFC 2541, March 1999.

[11] D. Eastlake, "Domain Name System Security Extensions", RFC 2535, March 1999.

[12] L Esibov, B. Aboba, and D. Thaler, "Multicast DNS", draft-ietf-dnsext-mdns-00.txt, November 2000.

[13] M. Feng, "A Self-Configuring and Self-Administering Name System", Master's Thesis, Massachusetts Institute of Technology, 2001.

[14] C.S. Giap, Y. Kadobayashi, S. Yamaguchi, "Zero Internet Administration Approach: the case of DNS", *Proc. of 12th Int. Conf. on Information Networking (ICOIN)*, pp. 350-355, 1998.

[15] E. Guttman, "Zero Configuration Networking", Proceedings *of INET 2000*, 2000.

[16] D. Harkins and D. Carrel, "The Internet Key Exchange (IKE)", RFC 2409, November 1998.

[17] M. Hattig, ed., *Zeroconf Requirements*, draft-ietf-zeroconf-reqts-07.txt, March 2001. (work in progress)

[18] K. Hornstein, et al, "DHCP Authentication Via Kerberos V", November 2000.

[19] D.B. Johnson, "Scalable Support for Transparent Mobile Host Internetworking", *Mobile Computing*, ch. 3, 1996.

[20] "JOIN DDNS", http://www.join.com/ddns.html

[21] S. Kent and R. Atkinson, "IP Authentication Header", RFC 2402, November 1998.

[22] S. Kent and R. Atkinson, "IP Encapsulating Security Payload (ESP)", RFC 2406, November 1998.

[23] S. Kent and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, November 1998.

[24] Y. Lin and M. Gerla, "Induction and Deduction for Autonomous Network", *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 9, pp. 1415-1425, December 1993.

[25] "Lucent QIP Enterprise 5.0: Automating IP Services Management", http://www.qip.lucent.com/products/qipent_6093.pdf

[26] "Meta IP Technical White Paper", http://www.checkpoint.com/products/metaip/whitepaper.pdf

[27] "Microsoft Vision for Home Networking", http://www.microsoft.com/homenet/Vision.htm, March 12, 2001.

[28] P.V. Mockapetris, "Domain Names – Concepts and Facilities", RFC 1034, November 1987.

[29] P.V. Mockapetris, "Domains Names – Implementation and Specification", RFC 1035 November 1987.

[30] T. Narton, E Nordmark, and W. Simpson, "Neighbor Discovery for IP Version 6 (IPv6)", RFC 2461, December 1998.

[31] N. Nuansri, T. Dillon, S. Singh, "An Application of Neural Network and Rule-Based System for Network Management: Application Level Problems", *Proc. of 30th Hawaii Int. Conf. on System Sciences*, vol 5, pp. 474-483, 1997.

[32] "Optivity NetID", http://www.nortelnetworks.com/products/01/unifiedmanagement/collateral/onetid_brief.pdf

[33] C. Park, et.al., "The Improvement for Integrity between DHCP and DNS", *High Performance Computing on the Information Superhighway*, pp. 511-516, 1997.

[34] C. Perkins, ed., "IP Mobility Support", RFC 2002, October 1996.

[35] C.E. Perkins, "Mobile networking in the Internet", *Mobile Networks and Applications,* vol. 3, pp. 319-334, 1998.

[36] C.E. Perkins, K. Luo, "Using DHCP with computers that move", *Wireless Networks*, vol. 1, pp. 341-353, 1995.

[37] "Protocol Standard for a NetBIOS Service on a TCP/UDP Transport: Concepts and Methods", RFC 1001, March 1987.

[38] J. Reynolds and J. Postel, "Assigned Numbers", RFC 1700, October 1994.

[39] D. Sabin, et al, "A Constraint-Based Approach to Diagnosing Configuration Problems", *Proceedings IJCAI-95 Workshop on AI in Distributed Information Networks*, 1995.

[40] "Shadow IPserver", http://www.nts.com/collateral/ipserverdatasheet.pdf.

[41] "Sharp Shows First 'Internet Ready' Convection Microwave Oven", Company Press Release, April 27, 2001.

[42] Y. Shim, H. Shim, M. Lee, O. Byeon, "Extension and Design of Secure Dynamic Updates in Domain Name Systems", *5th Asia-Pacific Conference on Communications (APCC),* vol. 2, pp. 1147-1150, 1998.

[43] G.S. Sidhu, R.F. Andrews, A.B. Oppenheimer, *Inside Appletalk*, Second Edition, Addison-Wesley Publishing Company, Inc., 1990.

[44] M. Stapp and Y. Rekhter, "The DHCP Client FQDN Option", draft-ietf-dhc-fqdn-option-01.txt, March 2001. (work in progress)

[45] R. Thayer, N. Doraswamy, and R. Glenn, "IP Security: Document Roadmap", RFC 2411, November 1998.

[46] H. Toivanen, “Secure Zero Configuration”, http://citeseer.nj.nec.com/401221.html.

[47] P. Vixie, “A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY)”, RFC 1996, August 1996.

[48] P. Vixie, et.al., “Dynamic Updates in the Domain Name System (DNS Update)”, RFC 2136, April 1997.

[49] P. Vixie, et al, “Secret Key Transaction Authentication for DNS (TSIG)”, RFC 2845, May 2000.

[50] B. Wellington, “Secure Domain Name System (DNS) Dynamic Update”, RFC 3007, November 2000.

[51] A. Williams, “Securing Zeroconf Networks”, draft-williams-zeroconf-security-00.txt, November 2000. (work in progress)

[52] “Zero Configuration Networking (zeroconf) Charter, http://www.ietf.org/html.charters/zeroconf-charter.html