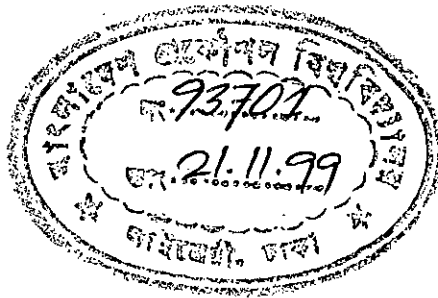


STUDY AND ADOPTION OF NETWORK STANDARDS FOR BANGLADESH

by

Khandaker Omar Farooq

A THESIS SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING, BUET, IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN
ENGINEERING



Thesis Supervisor : *Md. Abdus Sattar*

Thesis Co-Supervisor : *Dr. K. M. Waliuzzaman*



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY,
DHAKA, BANGLADESH.

OCTOBER, 1999.

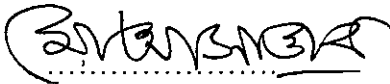
STUDY AND ADOPTION OF NETWORK STANDARDS FOR BANGLADESH

A Thesis

By

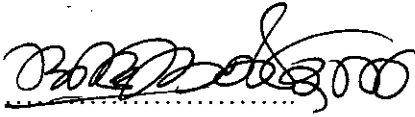
Khandaker Omar Farooq
Roll No. 901805P (1988-89)

Accepted as satisfactory as to style and contents for partial fulfilment of the requirements for the degree of M. Sc. Engineering in Computer Science and Engineering on October 7, 1999.



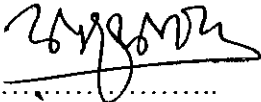
Md. Abdus Sattar
Assistant Professor
Department of Computer Science & Engineering
B.U.E.T., Dhaka, Bangladesh.

Chairman
and
Supervisor



Dr. K.M. Waliuzzaman
Professor & Head
Department of Computer and Information Technology
Islamic Institute of Technology, Gazipur, Bangladesh.

Co-Supervisor



Dr. M. Kaykobad
Professor & Head
Department of Computer Science & Engineering
B.U.E.T., Dhaka, Bangladesh.

Head of the Department
Ex-Officio Member



Dr. Chowdhury Mofizur Rahman
Associate Professor
Department of Computer Science & Engineering
B.U.E.T., Dhaka, Bangladesh.

Member



Prof. Dr. M. Abdus Sobhan
Executive Director
Bangladesh Computer Council, Dhaka, Bangladesh.

Member
(External)

ACKNOWLEDGEMENT

The work presented in this thesis is the result of the investigation carried out by the author under the supervision of Md. Abdus Sattar, Assistant Professor, Department of Computer Science and Engineering, BUET, Dhaka, and Dr. K.M. Waliuzzaman, Professor and Head, Department of Computer and Information Technology, IIT, Gazipur, Bangladesh.

The author would like to give his heartiest gratitude and thanks to his supervisors Dr. K.M. Waliuzzaman and Md. Abdus Sattar for their sincere guidance and co-operation, which lead this research work successful.

The author would like to thank Mr. M. A. Matin, IT Manager, British American Tobacco Bangladesh, and Mr. Mihir Kanti Majumder, Deputy Scientific Adviser, Ministry of Science & Technology for their active co-operation and invaluable contributions to this research work.

The author also likes to give his special thanks to his wife for her continuous support and co-operation during this research work.

Khandaker Omar Farooq

ABSTRACT

All the developed countries have their own standards in the field of Information Technology (IT), which guided them to achieve the significant growth in the IT industry. And these IT achievements worked as an initiator for them to acquire the top positions of the world. They have successfully changed their socio-economic status, and now controlling the whole world both economically and technologically with the help of IT. Unfortunately, we do not have any IT standards for our country, which could worked as an expediter to change our socio-economic structure.

In this circumstance, few standards have been developed to face the challenges of 21st Century. The standards have been developed based on the Directory Services – which is being considered as the technology for the next millennium. Traditional Domain Name System (DNS) and the Novell Directory Services (NDS) worked as a key enabler to develop these standards. The developed standards are Country top level domain structure i.e. **.bd** Domain model, NDS enabled DNS & Country NDS Tree structure, NDS objects naming conventions, NDS replication & DNS Zone transfer model, and IP address allocation standards.

Standards were developed considering the present technology and the trend of the technology for the future. Developed standards support existing technology and it will support upcoming future technology as well. These standards will work as a bridge between existing and the future technology.

Table of Contents

Chapter 1	6
Introduction	6
1.1 General Introduction to Network Standards in Bangladesh.	6
1.2 Literature Review.....	7
1.3 Recent Works on Directory Services Technologies	10
1.4 Scope of the Present Research	12
Chapter 2.....	14
TCP/IP Architecture.....	14
2.1 Architectural Model	14
2.1.1 Internetworking.....	14
2.1.2 Bridges, Routers and Gateways	14
2.2 Internet Protocol (IP)	15
2.2.1 IP Datagram	15
2.2.1.1 IP Datagram Format.....	16
2.2.1.2 IP Datagram Fragmentation.....	21
2.2.1.3 IP Datagram Routing Options.....	22
2.2.1.4 Internet Timestamp	24
2.3 Internet Control Message Protocol (ICMP).....	25
2.4 Ping	27
2.5 Traceroute	28
2.6 Address Resolution Protocol (ARP).....	28
2.6.1 ARP Detailed Concept.....	29
2.6.1.1 ARP Packet Generation	29
2.6.1.2 ARP Packet Reception.....	30
2.6.2 ARP and Subnets	31
2.6.3 Proxy-ARP or Transparent Subnetting.....	32
2.6.3.1 Proxy-ARP Concept.....	32
2.7 Reverse Address Resolution Protocol (RARP).....	33
2.7.1 RARP Concept.....	33
2.8 Ports and Sockets	34
2.8.1 Ports	34
2.8.2 Sockets	34
2.8.3 Basic Socket Calls.....	35
2.8.4 Socket Interfaces.....	37
2.9 User Datagram Protocol (UDP).....	37
2.9.1 Ports	38
2.9.2 UDP Datagram Format	39
2.9.3 UDP Application Programming Interface	40
2.10 Transmission Control Protocol (TCP)	40
2.10.1 Sockets	41
2.10.2 TCP Concept.....	42
2.10.2.1 Stream Data Transfer	42
2.10.2.2 Reliability.....	42
2.10.2.3 Flow Control	42
2.10.2.4 Multiplexing.....	43
2.10.2.5 Logical Connections	43

Chapter 3	44
Structure of the Domain Name Space	44
3.1 Introduction	44
3.2 Elements of the DNS	45
3.3 Domain Name Space and Resource Records	46
3.3.1 Name space specifications and terminology	46
3.3.2 Preferred name syntax	47
3.3.3 Resource Records	48
3.3.4 Textual expression of RRs	49
3.3.5 Aliases and canonical names	50
3.3.6 Queries	51
3.4 Name Servers	52
3.4.1 Introduction	52
3.4.2 How the database is divided into zones	52
3.4.3 Technical considerations for Name Server	53
3.4.4 Name server internals	54
3.4.4.1 Queries and responses	54
3.4.5 Zone maintenance and transfers	56
3.5 Resolvers	57
3.5.1 Introduction	57
3.5.2 Client-resolver interface	58
3.5.2.1 Typical functions	58
3.5.3 Aliases	59
3.5.4 Resolver internals	59
3.5.5 Stub resolvers	59
3.5.6 Resources	60
Chapter 4	62
Novell Directory Services	62
4.1 Introduction	62
4.2 How Objects Form the Directory Tree	63
4.3 Network-Wide Login	64
4.4 Analysis of NDS Schema and Objects	64
4.4.1 Schema Components	65
4.4.2 Schema Structure	65
4.4.3 Object Class Definitions	66
4.4.4 Object Class Structure Rules	67
4.4.5 Object Class Naming Structure Attributes	67
4.4.5.1 Naming Attribute Rules	67
4.4.5.2 Multi-valued Naming Attributes	68
4.4.5.3 Shareable Naming Attributes	68
4.4.5.4 Inheritance of Naming Attributes	68
4.4.6 Containment Classes	69
4.4.6.1 Containment Class Rules	69
4.4.6.2 Containment Classes in the Base Schema	70
4.4.6.3 Containment of Leaf Objects	71
4.4.6.4 Containment Classes and Inheritance	74
4.4.7 Super Classes	74
4.4.7.1 Root Schema Object	74
4.4.7.2 Super Class Rules	75
4.4.7.3 Class Hierarchy	75

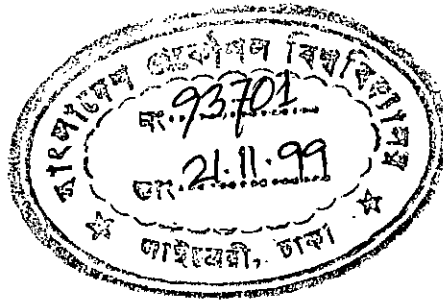
4.4.7.4 Object Class Inheritance Rules	76
4.4.8 Object Class Attributes	77
4.4.9 Object Class Flags.....	77
4.4.9.1 Container Flag.....	77
4.4.9.2 Effective Flag.....	78
4.4.9.3 Non-removable Flag	78
4.4.9.4 Ambiguous Naming Flag.....	79
4.4.9.5 Ambiguous Container Flag	79
4.4.10 Default ACL Templates.....	79
4.4.11 Construction Rules for Object Classes	80
4.4.12 Attribute Type Definitions.....	80
4.4.13 Attribute Syntaxes.....	81
4.4.14 Attribute Constraints.....	81
4.4.15 Attribute Syntax Definitions.....	82
4.5 NDS Partitioning.....	84
4.5.1 Partition Operations	84
4.6 NDS Replication	85
4.6.1 Replica Types.....	86
4.6.1.1 Master, Read-Write, and Read-Only Replicas.....	86
4.6.2 Subordinate References	87
4.6.3 Replica List	90
4.7 NDS Synchronization	90
4.7.1 Loose Consistency	91
4.7.2 Replica Synchronization Process.....	91
4.8 Distributed Relationship Management	92
4.8.1 External References	92
4.8.1.1 Creating External References	93
4.8.1.2 Deleting External References	93
4.8.1.3 Synchronizing External References.....	94
4.8.2 Back Links	94
4.8.2.1 Creating a Back Link	94
4.8.2.2 Deleting a Back Link	95
4.8.3 Obituaries.....	95
4.8.3.1 Primary and Secondary Obituaries	96
4.9 Network Time Synchronization.....	97
4.9.1 Secondary Time Server.....	97
4.9.2 Primary Time Server.....	97
4.9.3 Reference Time Server	98
4.9.4 Single Reference Time Server	98
Chapter 5.....	99
Directory Services Enabled .bd Domain.....	99
5.1 Introduction.....	99
5.2 Definition of Component Terms.....	99
5.3 Traditional Architectural Model of .bd Domain.....	100
5.4 Directory Services Enabled DNS.....	101
5.4.1 New NDS Objects for DNS	102
5.4.1.1 DNS Zone Object.....	102
5.4.1.2 DNS Resource Record Set Object	102
5.4.1.3 DNS Resource Records.....	102
5.4.1.4 DNS Server Object	103

5.4.2	New NDS objects for DHCP	103
5.5	NDS Enabled DNS Model of .bd Domain.....	104
5.6	Domain and Objects Naming Standards.....	107
5.6.1	Objectives	108
5.6.2	Use of Characters in Domain Object Naming	108
5.6.2.1	Acceptable characters	108
5.6.2.2	Use of underscore, space and hyphen	108
5.6.2.3	Country NDS Tree	109
5.6.2.4	Local NDS Tree	109
5.6.3	Second Level Domain Objects.....	110
5.6.4	The Third Level Domain Objects/Organisational Unit (OU) Objects...	110
5.6.5	User Accounts of Domains or Leaf Objects	111
5.6.6	Handling of Duplicate Names.....	111
5.7	IP Address Space Allocation Standard	112
5.8	Benefits of NDS enabled DNS Services.....	115
Chapter 6	116
Conclusion	116
6.1	Conclusion	116
6.2	Limitations.....	116
6.3	Future Development.....	117
6.3.1	Country legislation on information sharing	117
6.3.2	Contents of the Directory	118
6.3.3	Security Standards of the Network	118
6.3.4	Data Integrity	119
6.3.5	Country-wide Distributed National Database.....	119
References	120
Appendix A - ISO 3166 Country Codes	124



Table of Figures

- Figure 2.1: Internet Protocol (IP) 15
- Figure 2.2: Base IP Datagram 16
- Figure 2.3: IP Datagram Format 16
- Figure 2.4: Loose Source Routing Option 22
- Figure 2.5: Strict Source Routing Option 24
- Figure 2.6: Record Route Option 24
- Figure 2.7: Internet Timestamp Option 25
- Figure 2.8: Internet Control Message Protocol (ICMP) 26
- Figure 2.9: Packet InterNet Groper (PING) 27
- Figure 2.10: Traceroute 28
- Figure 2.11: ARP Request/Reply packet 29
- Figure 2.12: ARP Packet Reception Algorithm 31
- Figure 2.13: Hosts Interconnected by a Router 32
- Figure 2.14: Reverse Address Resolution Protocol (RARP) 33
- Figure 2.15: User Datagram Protocol (UDP) 37
- Figure 2.16: UDP, A Demultiplexer Based on Ports 38
- Figure 2.17: UDP Datagram Format 39
- Figure 2.18 Transmission Control Protocol (TCP) 40
- Figure 2.19: TCP Connection. 41
- Figure 3.1: Domain Name Space Tree 44
- Figure 4.1: Directory Tree 63
- Figure 4.2: Relationships between the Schema and the Directory Components. 66
- Figure 4.3: Containment Structure. 71
- Figure 4.4: Container Inheritance for Leaf Objects. 73
- Figure 4.5: Super Class Inheritance. 76
- Figure 4.6: A partitioned tree. 84
- Figure 4.7: Partitioning and Replication. 86
- Figure 4.8: Replica placement in a partitioned tree. 88
- Figure 4.9: Replica placement and subordinate references. 89
- Figure 4.10: Backlinks. 95
- Figure 4.11: Obituaries. 96
- Figure 5.1: The .bd Domain Name Space Tree. 100
- Figure 5.2: NDS Enabled DNS Model 101
- Figure 5.3: NDS based .bd model 104
- Figure 5.4: NDS Partition/Zone Model of .bd Domain 106
- Figure 5.5: NDS Replication/Zone Transfer Model of .bd Domain 107



Chapter 1

Introduction

1.1 General Introduction to Network Standards in Bangladesh.

Historically there has been no standardisation of the IT infrastructure in Bangladesh, which supports the deployment of information systems throughout the country to achieve a world-class standard. This is a major limitation to the expansion of IT infrastructure in Bangladesh. It is therefore required that a new network standard should be specified for the country to face the challenges of 21st century.

Over the past few years, Internet Technologies has played a vital role for the overall technological and economical changes of the world. Countries/Nations who had adopted the Internet Technologies have placed themselves at the top of the list of successful Countries/Nations. They have changed their fate by adopting this technology in every corner of their life. Now, the whole world is controlled by Internet Technologies, and at the same time it is also required to change the traditional architecture of Internet to adopt the fast growing demand of secure access to all network resources from anywhere in the world. To adopt these demands into Internet, a new concept has developed i.e. Directory Services Enabled Internet.

Directory Services Enabled Internet is basically an integration of traditional Domain Name System (DNS) and the X.500 Directory Services standard. Novell Directory Services (NDS) is the implementation of X.500 standard, and NDS has got the industry acceptance as the Directory Services standard. So, integration of DNS with NDS is being considered as one of the potential solutions for Directory Services Enabled Internet.

The Domain Name System (DNS) is a distributed database system that provides hostname to IP resource mapping (usually the IP address) and other information for computers on an inter-network. Computer connected on the Internet use a DNS server to locate other host/computer on the Internet.

DNS is made up of two distinct components, the hierarchy and the name service. The DNS hierarchy specifies the structure, naming conventions, and delegation of authority in the DNS service. The DNS name service provides the actual name-to-address mapping mechanism.

DNS uses a hierarchy to manage its distributed database system. The DNS hierarchy, also called the domain name space, is an inverted tree structure, much like NDS.

The DNS tree has a single domain at the top of the structure called the root domain. A period or dot (.) is the designation for the root domain. Below the root domain is the top-level domains that divide the DNS hierarchy into segments.

Novell Directory Services (NDS) is an implementation of a global networking directory. It stores network resources and services in a way that is accessible to all network users based on their access rights. NDS can be conveniently used to offer a service in a global network.

One major advantage of using Directory Services is to reduce network traffic. Before Directory Services was implemented, the service provider had to broadcast its service using SAP (Service Advertising Protocol). The client could then locate the service by querying for the nearest node that offered the service. This method generates much network traffic, primarily SAP advertising data. With Novell Directory Services, the service providers' object created in Directory Services is globally available for the client process to locate the service. Other advantages of using NDS as a service location mechanism, which provides fault tolerance, data store, global access, and security check.

Integrating DNS with NDS greatly simplifies the task of network administration by enabling all configuration information into one distributed database. Furthermore, the DNS configuration information is replicated just like any other data in NDS.

Integrating DNS with NDS also enables an update interaction between DNS and Dynamic Host Configuration Protocol (DHCP) through the Dynamic DNS (DDNS) feature. When a host is assigned an IP address by DHCP, the DNS information can be automatically updated to associate the hostname with the new address when using DDNS.

1.2 Literature Review

The Internet is growing at a phenomenal rate, with no deceleration in sight. Every month thousands of new users is added. New networks are added literally almost every day. In fact, it is entirely conceivable that in the future every human with access to a computer will be able to interact with every other over the Internet and her sister networks. However, the ability to interact with everyone is only useful if one can locate the people with whom they need to work. Thus, as the Internet grows, one of the limitations imposed on the effective use of the network will be determined by the quality and coverage of Directory Services available [1].

The existing Internet domain name space, however, has some self-imposed structure to it. Especially in the upper level domains, the domain names follow certain traditions (not rules, really, since they can and have been broken). This helps domain names from appearing totally chaotic. Understanding these tradition is an enormous asset if you're trying to decipher a domain name [2].

The original view of the Internet universe was a two-level hierarchy: the top level the Internet as a whole, and the level below it individual networks, each with its own

network number. The Internet does not have a hierarchical topology; rather the interpretation of addresses is hierarchical. In this two-level model, each host sees its network as a single entity; that is, the network may be treated as a "black box" to which a set of hosts is connected. [3]

While this view has proved simple and powerful, a number of organisations have found it inadequate, and have added a third level to the interpretation of Internet addresses. In this view, a given Internet network is divided into a collection of subnets [3].

The Domain Name Service, or DNS, contains information about the mapping of host and domain names, such as, "home.ans.net", to IP addresses. This is done so that human can use easily remembered names for machines rather than strings of numbers. It is maintained in a distributed fashion, with each DNS server providing nameservice for a limited number of domains. Also, secondary nameservers can be identified for each domain, so that one unreachable network will not necessarily cut off nameservice. However, even though the DNS is superlative at providing these services, there are some problems when we attempt to provide other Directory Services in the DNS. First, the DNS has very limited search capabilities. Second, the DNS supports only a small number of data types. Adding new data types, such as photographs, would involve very extensive implementation changes [1].

In order to make global communication over computer networks work efficiently, a global electronic White Pages service is indispensable. Such a directory service could also contain telephone and fax numbers, postal addresses as well as platform type to facilitate in translation of documents between users on different systems. An electronic White Pages may prove to be useful for specific local purposes; replacing paper directories or improving quality of personnel administration for example. An electronic directory is much easier to produce and more timely than paper directories which are often out of date as soon as they are printed [4].

The Internet White Pages Project provides many companies in the US with an opportunity to pilot X.500 in their organisations. Operating as a globally distributed directory service, this project allows organisations in a wide variety of industry type to make themselves known on the Internet and to provide access to their staff as desired [4].

NDS is a true directory service because it lists and provides access to every resource on your network. NDS provides administrators with a single, logical, and concise view of all network resources and services [5].

NDS adds value to applications because it simplifies access and management of information for network administrators and end-user customers. For example, the NDS name service maps network names to addresses using a hierarchical name space rather than a flat name space. This hierarchy allows the database to be mapped as a tree that can be partitioned by its subtrees. Because object names contain the hierarchy information, users can access network resources globally, and administrators can administer the entire tree and its objects from a single point [6].

NDS provides maximum flexibility and control in building and deploying applications. Simply, applications you build that are directory-enabled means they are aware of the network. Enabled applications can query and log in to network services, discover new services, and register application-specific information that can then be administered, managed and secured across the network, rather than on individual client or server machines [6].

Beyond global directory lookups, NDS not only provides the better-known benefits of a single sign-on and a single point of administration; NDS gives a powerful network database and the potential for new levels of automation and interactivity for application [7].

For example, with the extensibility of NDS, user can modify the directory schema to add application-specific attributes or information that automates how applications and users interact. Just as users and resources are managed across the network from the directory, application can also be benefited from the anytime, anywhere access control, management and security attributes of a global directory. So instead of requiring application's to perform its own login and password security access, it can leverage NDS for this information [7].

NDS provides the ability to automate how an application can log into the network and gain access to services, lowering application's cost of ownership by providing a single point of administration, management and distribution. It can serve as the universal link between disparate (and distant) workstations, servers, hubs, routers, databases, operating systems, network environments, individual users, workgroups, organisations, and applications [8].

With the evolution of multiple interface standards, NDS enables to develop a common interface and share its directory information across multiple name spaces. Among the interfaces, more and more organisations are already developing to LDAP (Lightweight Directory Access Protocol), and Novell is providing a scalable, secure, manageable LDAP directory via NDS. LDAP Services for NDS enables new uses of NDS in network environments, enabling developers, and administrators to access any NDS or NDS-compatible directory through any LDAP-compatible application [9].

NDS offers technology that acts as a repository of information for the network, and it makes it possible to keep tabs on network resources while enhancing the availability and security of business-critical resources. NDS is also distributed, fully replicated and fault-tolerant. Because of replication, users can log in once and access services and the most current information from any server on the network. In addition, in the event of a server failure, NDS automatically re-routes user requests to the closest replicated server without users taking any action [8].

Aside from the obvious benefits of single sign-on and single point of administration (and the enhanced security inherent in those features), NDS is the source for additional security, scalability, and user/administrator productivity. For example, Novell developer teams exploited the power and flexibility of NDS in ManageWise to enable administrators to monitor and manage mixed IntranetWare and Windows NT environments to reduce network failures more than ever before. Also through NDS,

ManageWise can provide an application inventory across the network to tell administrators which application versions are located on each workstation [8].

NDS also provides other kinds of directory information to all the clients in a network environment. LDAP Services for NDS supports most NDS security features and adds an LDAP access control layer that provides additional security features, which allow some types of directory information available to the public, few types available to specific organisation, and certain types available only to those groups or individuals that have a need to know. Working together, LDAP and NDS make the Internet/intranet environment directory-enabled [9].

Internet connections will also be an integrated part of the service, effectively eliminating the distinction between "the net" and local networks by seamlessly allowing users to access data and functionality resident anywhere in the networked world. For example, NDS will map frequently accessed sites on the World Wide Web, so that Internet Web sites can become indistinguishable from other network resources managed and accessed through NDS. NetWare Connect Services (NCS) will ultimately include worldwide dial-up and mobile access for individual users, completing the vision of a true global LAN accessible from any place, at anytime [10].

The bottom line is that NDS simplifies, automates and protects information and information technology [11].

1.3 Recent Works on Directory Services Technologies

X.500 is filling Directory Services needs in a large number of countries. As a directory to locate people, it is provided in the U.S. as the White Pages Pilot Project run by Performance Systems International (PSI), and in Europe under the PARADISE Project as a series of nation-wide pilots. It is also being used by the FOX Project in the United States to provide WHOIS services for people and networks, and to provide directories of objects as disparate as NIC (Network Information Center) Profiles and a pilot K-12 Educators directory. It is also being investigated for its ability to provide resource location facilities and to provide source location for WAIS servers. In fact, in almost every area where one could imagine needing a directory service (particularly for distributed directory services), X.500 is either providing those services or being expanded to provide those services [12].

X.500 is being used today to provide the backbone of a global White Pages service. There is almost 3 years of operational experience with X.500, and it is being used widely in Europe and Australia in addition to North America. In addition, the various X.500 implementations add some other features, such as photographs in G3-FAX format, and colour photos in JPEG format. However, as X.500 is standards based, there are very few incompatibilities between the various versions of X.500, and as the namespace is consistent, the information in the Directory can be accessed by any

implementation. Also, work is being done in providing Yellow Pages services and other information resource location tasks in the Directory. [1]

NDS is the implementation of X.500 standard and it is valuable for developers because of its support of emerging Internet standards, its stability, and its ever-increasing interoperability with platforms from a wide array of vendors. NDS is technology with a long track record that includes nearly a decade in development and more than four years on the market. NDS is a stable, mature, widely used directory service that lets you choose the development environment that's right for you (Java, ActiveX controls, Java Beans, Scripting, C/C++, etc.) [8].

NDS supports several open standards and emerging Internet protocols and languages. For example, LDAP (Lightweight Directory Access Protocol), JNDI (Java Naming and Directory Interface), CORBA (Common Object Request Broker) and RADIUS (Remote Authentication Dial-In User Service) let you approach any directory-enabling project with confidence and a wide range of choices. NDS have already been integrated into UNIX systems, Windows NT Server, Sun-Solaris, and is also being integrated into other operating systems [8].

No definition of NDS would be complete without mentioning its platform-independent capabilities [11]. NDS is the only directory service that supports the leading UNIX implementations, Windows NT Server, and Novell's own widely deployed networking software products, including IntranetWare, GroupWise, and ManageWise.

Major Novell OEM partners such as IBM, Hewlett-Packard, Sun, and SCO support NDS, and many third-party applications leverage NDS, including products from Cheyenne Software, Motorola, CallWare Technologies, and Oracle Corporation [8].

NDS has been licensed to Hewlett-Packard to be bundled in its HP-UX servers and is also bundled with SCO UnixWare and on Sun Solaris, and will soon run on IBM RS/6000 and S/390 systems; and Novell has already ported it to Windows NT Server, with more operating systems coming soon thereafter [11].

In addition, NDS already provides standard IP support that's accessible through a standard Web browser, Lightweight Directory Access Protocol (LDAP) or HTTP. And, it will soon provide access and management for Netscape SuiteSpot services [<http://www.novonyx.com>].

With the evolution of the NOS and the globalization of network computing, directories have risen in stature and importance. NDS is also bringing directory capabilities to the physical network layer like leading hardware vendors (Ascend, U.S. Robotics, Cisco, Bay Networks, 3Com, etc.) implementing directory services functionality with their dial-in products [8].

At the Internet/intranet level, NDS is the "carrier-grade" directory service. Novell is extending NDS's reach globally via telecommunications companies, which include AT&T, Deutsche Telekom and Nippon Telephone and Telegraph. AT&T WorldNet

Intranet Connect Services, for example, is an NDS-enabled network that brings the secure authentication and access of NDS to AT&T's customers and partners [8].

NDS offers single sign-on access through a secure login and organises network resources (users, printers, workgroups, applications, volumes, file servers, database servers, routers, objects, etc.) hierarchically in a directory tree. It also provides security by keeping the criminals and the curious from logging on [13].

The University of Michigan is using X.500 for electronic mail routing. Any mail coming to the university domain, umich.edu; gets expanded out to a local address that is stored in the rfc822Mailbox attribute. The University also operates a standard X.500 name server which provides name lookup service of over 200,000 names. They use the Lightweight Directory Access Protocol (LDAP) [14].

An implementation of the X.500 Standard directory service has been incorporated into the Open Software Foundation (OSF) Distributed Computing Environment (DCE). This component, known as the Global Directory Service (GDS), provides an area where distributed application clients can find their application servers. The GDS, in response to requests made by other clients, provides the unique network address for a particular DCE resource. Because it is based on an international standard, GDS can offer access to resources among users and organizations worldwide. This scalable service can be performed in DCE environments that range in size from the very small to the very large [4].

Lookup services can be implemented into a variety of applications. Cambridge University in Great Britain implemented the X.500 directory service into an employee locator application. Based on badge sensors at strategic locations, this application can determine the whereabouts of an employee on the campus. As the individual moves about, the sensors register their location in an X.500 Directory [4].

Digital Signature Service (DSS) and Privacy Enhanced Mail (PEM) work on the principal of a directory key server which generates and provide users with "public" codes that match previously registered "private" codes. Only the recipient can decipher messages sent in this fashion. The X.509 [15] standard for key certificates easily fits within the structure of the X.500 Directory Service.

1.4 Scope of the Present Research

Integration of DNS with NDS is one of the major milestones in the IT industry, and this integration is being considered as the future internet architecture for 21st Century. In Bangladesh we do not have any network standards, which could guide us to face the IT challenges of the next millennium.

The objective of the research was to develop the network standards for the Bangladesh by analyzing the Domain Name Space, Novell Directory Services,

Microsoft Windows NT, SCO UNIXWare and Novell NetWare. And test the developed standards into different platforms like NetWare, Windows NT and SCO UNIXWare.

In this research, a few standards have been developed for the IT infrastructure of the country. First of all a traditional domain structure was developed for the Bangladesh, which could work as a model for the .bd domain of the Bangladesh. After developing the traditional domain structure, a Country NDS Tree was developed where traditional domain structure was suited to achieve the NDS enabled DNS model for the country. Other standards like DNS Zone Transfer, NDS Replication, NDS Object Naming Convention, and IP address allocation was also developed for the country network standards. These standards have been tested in NetWare, Windows NT and SCO UNIX environment.

Chapter 2

TCP/IP Architecture

2.1 Architectural Model

The TCP/IP protocol suite is named for two of its most important protocols: Transmission Control Protocol (TCP) and Internet Protocol (IP). Another name for it is the Internet Protocol Suite, and this is the phrase used in official Internet standards documents.

2.1.1 Internetworking

The first design goal of TCP/IP was to build an interconnection of networks that provided universal communication services: an internetwork, or internet. Each physical network has its own technology-dependent communication interface, in the form of a programming interface that provides basic communication functions (primitives). Communication services are provided by software that runs between the physical network and the user applications and that provides a common interface for these applications, independent of the underlying physical network. The architecture of the physical networks is hidden from the user.

The second aim is to interconnect different physical networks to form what appears to the user to be one large network. Such a set of interconnected networks is called an internetwork or an internet.

2.1.2 Bridges, Routers and Gateways

Forming an internetwork by interconnecting multiple networks is done by bridges, routers and gateways.

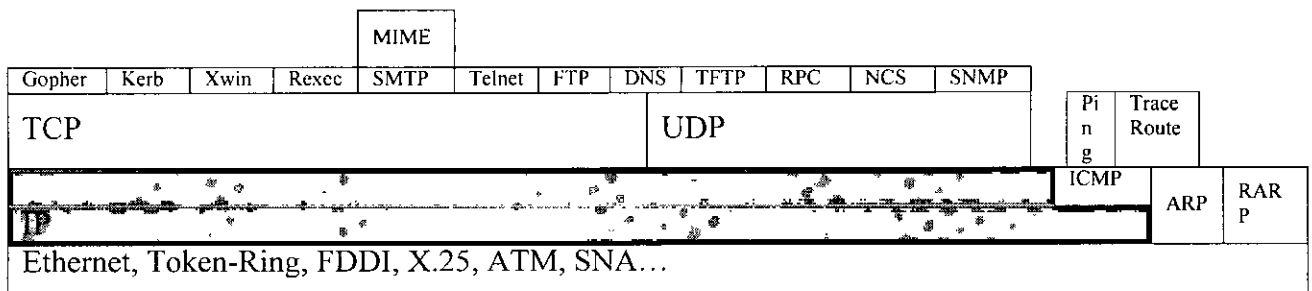
Bridge Interconnects LAN segments at the Data Link Interface layer level and forwards frames between them. A bridge performs the function of a MAC relay, and is independent of any higher layer protocol (including the Logical Link protocol). It provides MAC layer protocol conversion, if required. A bridge can be said to be transparent to IP. That is, when a host sends an IP datagram to another host on a network connected by a bridge, it sends the datagram directly to the host and the datagram “crosses” the bridge without the sending host being aware of it.

- Router** Interconnects networks at the Network layer level and routes packets between them. The router must understand the addressing structure associated with the networking protocols it supports and take decisions on whether, or how, to forward packets. Routers are able to select the best transmission paths and optimal packet sizes. The basic routing function is implemented in the IP layer of the TCP/IP protocol stack.
- Gateway** Interconnects networks at higher levels than bridges or routers. A gateway usually supports address mapping from one network to another, and may also provide transformation of the data between the environments to support end-to-end application connectivity.

2.2 Internet Protocol (IP)

IP is a standard protocol with STD number 5 which also includes ICMP (Internet Control Message Protocol (ICMP) and IGMP (Internet Group Management Protocol). Figure 2.1 shows the position of IP layer in TCP/IP protocol suite. IP is the protocol that hides the underlying physical network by creating a virtual network view. It is an unreliable, best-effort connectionless packet delivery protocol [16].

Figure 2.1: Internet Protocol (IP)



It adds no reliability, flow control or error recovery to the underlying network interface protocol. Packets (datagrams) sent by IP may be lost, out of order, or even duplicated, and IP will not handle these situations. It is up to higher layers to provide these facilities.

IP also assumes little from the underlying network mechanisms, only that the datagrams will “probably” (best-effort) be transported to the addressed host.

2.2.1 IP Datagram

The Internet datagram (IP datagram) is the base transfer packet in the Internet protocol suite. It has a header containing information for IP, and data that is relevant only to the higher level protocols. Figure 2.2 shows the base IP datagram mechanism.

Figure 2.2: Base IP Datagram

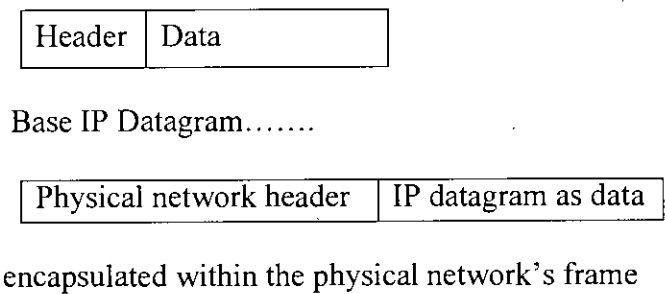


Figure 2.2 indicates that the IP datagram is encapsulated in the underlying network's frame, which usually has a maximum length or frame limitation, depending on the hardware used. For Ethernet, this will typically be 1500 bytes. Instead of limiting the IP datagram length to some maximum size, IP can deal with fragmentation and re-assembly of its datagrams. In particular, the IP standard does not impose a maximum size, but states that all subnetworks should be able to handle datagrams of at least 576 bytes.

Fragments of a datagram all have a header, basically copied from the original datagram, and data following it. They are treated as normal IP datagrams while being transported to their destination. However, if one of the fragments gets lost, the complete datagram is considered lost since IP does not provide any acknowledgement mechanism, so the remaining fragments will simply be discarded by the destination host.

2.2.1.1 IP Datagram Format

Figure 2.3 shows the detailed structure of the IP datagram format.

Figure 2.3: IP Datagram Format

0	4	8	16	19	31		
VERS	LEN	Type of Service	Total Length				
Identification				Flags	Fragment Offset		
TTL	Protocol		Header Checksum				
Source IP address							
Destination IP address							
Options				padding			
data							
.....							
.....							

The IP datagram header is a minimum of 20 bytes long:

Where:

VERS: 4 bits

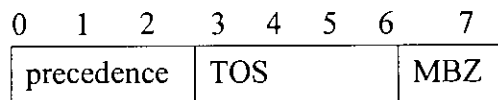
The version of the IP protocol. The current version is 4. 5 is experimental and 6 is "IP: The Next Generation (IPng)"

LEN: 4 bits

The length of the IP header counted in 32-bit quantities. This does not include the data field.

Type of Service: 8 bits

The type of service is an indication of the quality of service requested for this IP datagram.



Where:

Precedence is a measure of the nature and priority of this datagram:

000	Routine
001	Priority
010	Immediate
011	Flash
100	Flash override
101	Critical
110	Internetwork control
111	Network control

TOS Specifies the type of service value:

1000	Minimize delay
0100	Maximize throughput
0010	Maximize reliability
0001	Minimize monetary cost
0000	Normal service

MBZ Reserved for future use ("must be zero" unless participating in an Internet protocol experiment which makes use of this bit).

Total Length: 16 bits

The total length of the datagram, header and data, specified in bytes.

Identification: 16 bits



A unique number assigned by the sender to aid in reassembling a fragmented datagram. Fragments of a datagram will have the same identification number.

Flags: 3 bits

Various control flags:

	0	1	2
0	D	M	
	F	F	

Where

- 0 Reserved, must be zero
- DF Don't Fragment: 0 means allow fragmentation, 1 means do not allow fragmentation.
- MF More Fragments: 0 means that this is the last fragment of this datagram, 1 means that this is not the last fragment [16].

Fragment Offset: 13 bits

Used with fragmented datagrams, to aid in reassembly of the full datagram.

Time to Live: 8 bits

Specifies the time (in seconds) this datagram is allowed to travel. Each router where this datagram passes is supposed to subtract from this field its processing time for this datagram. Actually a router is able to process a datagram in less than 1 second; thus it will subtract one from this field, and the TTL becomes a hop-count metric rather than a time metric. When the value reaches zero, it is assumed that this datagram has been travelling in a closed loop and it is discarded. The initial value should be set by the higher-level protocol which creates the datagram [16].

Protocol : 8 bits

Indicates the higher-level protocol to which IP should deliver the data in this datagram.

Some important values are:

- 0 Reserved
- 1 Internet Control Message Protocol (ICMP)
- 2 Internet Group Management Protocol (IGMP)
- 3 Gateway-to-Gateway Protocol (GGP)
- 4 IP (IP encapsulation)
- 5 Stream
- 6 Transmission Control Protocol (TCP)
- 8 Exterior Gateway Protocol (EGP)
- 9 Private Interior Routing Protocol
- 17 User Datagram (UDP)
- 89 Open Shortest Path First

The full list can be found in STD 2 — Assigned Numbers [17].

Header Checksum: 16 bits

A checksum on the header only. It does not include the data. Since some header fields change (e.g., time to live), this is recomputed and verified at each point that the internet header is processed.

The checksum algorithm is:

The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header. For purpose of computing the checksum, the value of the checksum field is zero.

If the header checksum does not match the contents, the datagram is discarded because at least one bit in the header is corrupt, and the datagram may even have arrived at the wrong destination.

Source IP Address: 32 bits

The 32-bit IP address of the host sending this datagram.

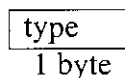
Destination IP Address: 32 bits

The 32-bit IP address of the destination host for this datagram.

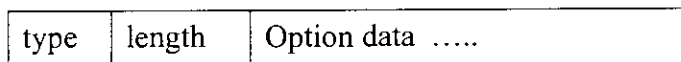
Options: Variable

An IP implementation is not required to be capable of generating options in the datagrams it creates, but all IP implementations are required to be able to process datagrams containing options. The Options field is variable in length. There may be zero or more options. There are two option formats. The format for each is dependent on the value of the option number found in the first byte.

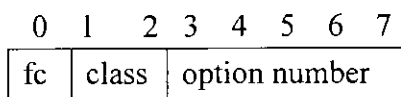
- A type byte alone.



- A type byte, a length byte and one or more option data bytes.



- The type byte has the same structure in both cases:



Where:

fc The copied flag indicates that this option is copied into all fragments on fragmentation.

0=not copied
1=copied

class

The option class is a 2-bit unsigned integer:

0= control
1 =reserved
2 =debugging and measurement
3= reserved

option number

The option number is a 5-bit unsigned integer.

The following internet options are defined:

CLASS NUMBER LENGTH DESCRIPTION

CLASS	NUMBER	LENGTH	DESCRIPTION
0	0	-	End of Option list. This option occupies only 1 octet; it has no length octet.
0	1	-	No Operation. This option occupies only 1 octet; it has no length octet.
0	2	11	Security. Used to carry Security, Compartmentation, User Group (TCC), and Handling Restriction Codes compatible with DOD requirements.
0	3	var.	Loose Source Routing. Used to route the internet datagram based on information supplied by the source.
0	9	var.	Strict Source Routing. Used to route the internet datagram based on information supplied by the source.
0	7	var.	Record Route. Used to trace the route an internet datagram takes.
0	8	4	Stream ID. Used to carry the stream identifier.
2	4	var.	Internet Timestamp.

length

counts the length (in bytes) of the option, including the type and length fields.

option data

contains data relevant to the option.

padding

If an option is used, the datagram is padded with all-zero bytes up to the next 32-bit boundary.

data

The data contained in the datagram is passed to a higher-level protocol, as specified in the protocol field.

2.2.1.2 IP Datagram Fragmentation

When an IP datagram travels from one host to another, it can cross different physical networks. Physical networks have a maximum frame size; called the Maximum Transmission Unit (MTU), which limits the length of a datagram that can be placed in one physical frame. Therefore, a scheme has been put in place to fragment long IP datagrams into smaller ones, and to reassemble them at the destination host. IP requires that each link has an MTU of at least 68 bytes, so if any network provides a lower value than this, fragmentation and re-assembly must be implemented in the network interface layer in a way that is transparent to IP. 68 is the sum of the maximum IP header length of 60 bytes and the minimum possible length of data in a non-final fragment (8 bytes). IP implementations are not required to handle unfragmented datagrams larger than 576 bytes, but most implementations will handle larger values, typically slightly more than 8192 bytes or higher, and rarely less than 1500.

An unfragmented datagram has all-zero fragmentation information. That is, the more fragments flag bit is zero and the fragment offset is zero. When fragmentation is to be done, the following steps are performed:

- The DF flag bit is checked to see if fragmentation is allowed. If the bit is set, the datagram will be discarded and an error will be returned to the originator using ICMP.
- Based on the MTU value, the data field is split into two or more parts. All newly created data portions must have a length which is a multiple of 8 bytes, with the exception of the last data portion.
- All data portions are placed in IP datagrams. The header of these datagrams are copies of the original one, with some modifications:
 - The more fragments flag bit is set in all fragments except the last.
 - The fragment-offset field in each is set to the location this data portion occupied in the original datagram, relative to the beginning of the original unfragmented datagram. The offset is measured in 8-byte units.
 - If options were included in the original datagram, the high order bit of the option type byte determines whether or not they will be copied to all fragment datagrams or just to the first one. For instance, source route options have to be copied in all fragments and therefore they have this bit set.
 - The header length field of the new datagram is set.
 - The total length field of the new datagram is set.

- The header checksum field is re-calculated.
- Each of these fragmented datagrams is now forwarded as a normal IP datagram. IP handles each fragment independently, that is, the fragments may traverse different routers to the intended destination, and they may be subject to further fragmentation if they pass through networks that have smaller MTUs.

At the destination host, the data has to be reassembled into one datagram. The identification field of the datagram was set by the sending host to a unique number (for the source host, within the limits imposed by the use of a 16-bit number). As fragmentation doesn't alter this field, incoming fragments at the receiving side can be identified, if this ID field is used together with the Source and Destination IP addresses in the datagram. The Protocol field is also be checked for this identification.

In order to reassemble the fragments, the receiving host allocates a buffer in storage as soon as the first fragment arrives. A timer routine is then started. When the timer timeouts and not all of the fragments have been received, the datagram is discarded. The initial value of this timer is called the IP datagram time-to-live (TTL) value.

When subsequent fragments of the datagram arrive, before the timer expires, the data is simply copied into the buffer storage, at the location indicated by the fragment-offset field. As soon as all fragments have arrived, the complete original unfragmented datagram is restored, and processing continues, just as for unfragmented datagrams.

IP does not provide the reassembly timer. It will treat each and every datagram, fragmented or not, the same way, that is, as individual messages. It is up to the higher layer to implement a timeout and to look after any missing fragments. The higher layer could be TCP for a connection-oriented transport network or the application for connectionless transport networks based upon UDP and IP.

2.2.1.3 IP Datagram Routing Options

The IP datagram Options field allows two methods for the originator of an IP datagram to explicitly provide routing information and one for an IP datagram to determine the route that it travels.

Loose Source Routing: Figure 2.4 shows the Loose Source Routing Option. The Loose Source Routing option, also called the Loose Source and Record Route (LSRR) option, provides a means for the source of an IP datagram to supply explicit routing information to be used by the routers in forwarding the datagram to the destination, and to record the route followed.

Figure 2.4: Loose Source Routing Option

10000011	length	pointer	route data
----------	--------	---------	------------

1000011 (decimal 131) is the value of the option type byte for loose source routing.

length contains the length of this option field, including the type and length fields.

pointer points to the option data at the next IP address to be processed. It is counted relative to the beginning of the option, so its minimum value is four. If the pointer is greater than the length of the option, the end of the source route is reached and further routing is to be based on the destination IP address.

route data is a series of 32-bit IP addresses.

Whenever a datagram arrives at its destination and the source route is not empty ($\text{pointer} < \text{length}$) the receiving host will:

- Take the next IP address in the route data field (the one indicated by the pointer field) and put it in the Destination IP address field of the datagram.
- Put the local IP address in the source list at the location pointed to by the pointer field. The IP address for this is the local IP address corresponding to the network on which the datagram will be forwarded (routers are attached to multiple physical networks and thus have multiple IP addresses).
- Increment pointer by 4.
- Transmit the datagram to the new destination IP address.

This procedure ensures that the return route is recorded in the route data (in reverse order) so that the final recipient uses this data to construct a loose source route in the reverse direction. This is a loose source route because the forwarding router is allowed to use any route and any number of intermediate routers to reach the next address in the route.

The originating host puts the IP address of the first intermediate router in the destination address field and the IP addresses of the remaining routers in the path, including the target destination are placed in the source route option. The recorded route in the datagram when it arrives at the target contains the IP addresses of each of the routers that forwarded the datagram. Each router has moved one place in the source route, and normally a different IP address will be used, since the routers record the IP address of the outbound interface but the source route originally contained the IP address of the inbound interface.

Strict Source Routing: Figure 2.5 shows the Strict Source Routing Option. The Strict Source Routing option, also called the Strict Source and Record Route (SSRR) option, uses the same principle as loose source routing except that the intermediate router must send the datagram to the next IP address in the source route via a directly connected network and not via an intermediate router. If it cannot do so it reports an error with an ICMP Destination Unreachable message.

Figure 2.5: Strict Source Routing Option

10001001	length	pointer	route data
----------	--------	---------	------------

10001001 (decimal 137) is the value of the option type byte for strict source routing

length has the same meaning as for loose source routing

pointer has the same meaning as for loose source routing

route data is a series of 32-bit IP addresses

Record Route: Figure 2.6 shows the Record Route Option. This option provides a means to record the route of an IP datagram. It functions similarly to the source routing discussed above, but this time the source host has provided an empty routing data field, which will be filled in as the datagram traverses routers. Note that sufficient space for this routing information must be provided by the source host: if the data field is filled before the datagram reaches its destination, the datagram is forwarded with no further recording of the route.

Figure 2.6: Record Route Option

00000111	length	pointer	route data
----------	--------	---------	------------

00000111 (decimal 7) is the value of the option type byte for record route

length has the same meaning as for loose source routing

pointer has the same meaning as for loose source routing

route data is a multiple of four bytes in length chosen by the originator of the datagram

2.2.1.4 Internet Timestamp

A timestamp is an option forcing some (or all) of the routers on the route to the destination to put a timestamp in the option data. The timestamps are measured in seconds and can be used for debugging purposes. Figure 2.7 shows the value and option field of the Internet Timestamp. They cannot be used for performance measurement for two reasons:

- They are insufficiently precise because most IP datagrams will be forwarded in less than one second.
- They are insufficiently accurate because IP routers are not required to have synchronized clocks.

Figure 2.7: Internet Timestamp Option

0	8	16	24	28	31
01000100	length	pointer	oflw	flag	
IP address					
timestamp					

Where

01000100 (Decimal 68) is the value of the option type for the internet time stamp option.

length Contains the total length of this option, including the type and length-fields.

pointer Points to the next timestamp to be processed (first free timestamp).

oflw (overflow) Is a 4 bit unsigned integer of the number of IP modules that cannot register timestamps due to a lack of space in the data field.

flag Is a 4-bit value which indicates how timestamps are to be registered.
Values are:

0 Timestamps only, stored in consecutive 32-bit words.

1 Each timestamp is preceded by the IP address of the registering module.

2 The IP address fields are pre-specified, and an IP module only registers when it finds its own address in the list.

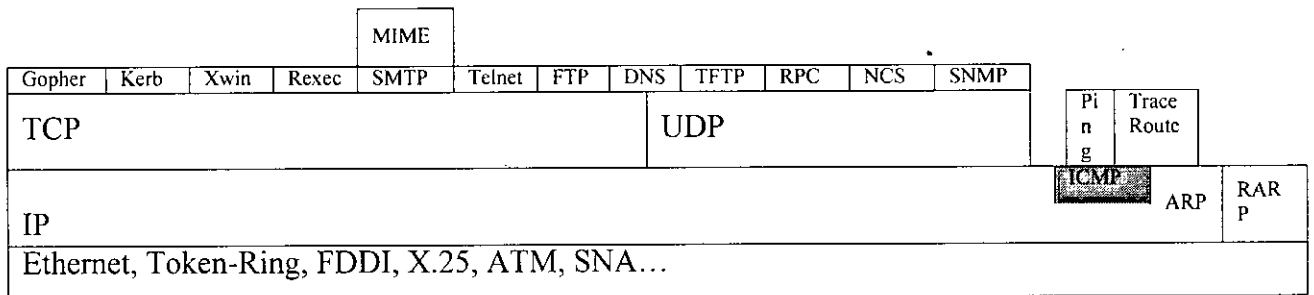
timestamp A 32-bit timestamp recorded in milliseconds since midnight UT (GMT).

The originating host must compose this option with a large enough data area to hold all the timestamps. If the timestamp area becomes full, no further timestamps are added.

2.3 Internet Control Message Protocol (ICMP)

ICMP is a standard protocol with STD number 5 which also includes IP. Its status is required. It is described in RFC 792 with updates in RFC 950 [19, 9]. ICMP position in TCP/IP suite is shown in Figure 2.8.

Figure 2.8: Internet Control Message Protocol (ICMP)



ICMP Router Discovery is a proposed standard protocol with a status of elective. It is described in RFC 1256.

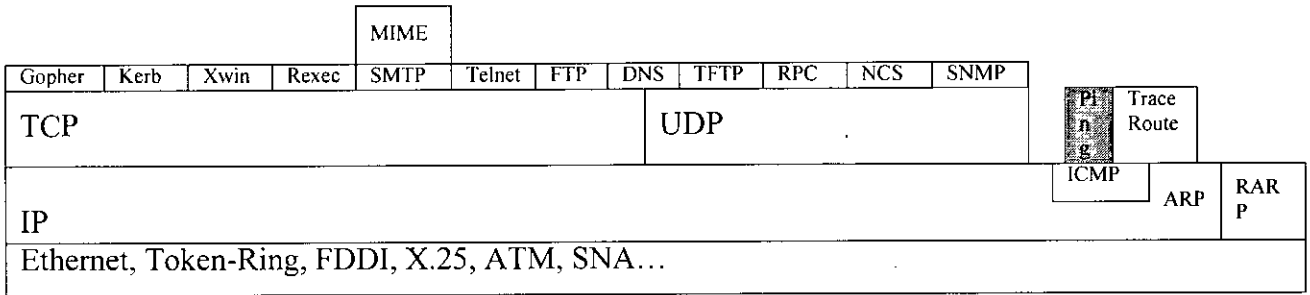
When a router or a destination host must inform the source host about errors in datagram processing, it uses the Internet Control Message Protocol (ICMP). ICMP can be characterized as follows:

- ICMP uses IP as if ICMP were a higher-level protocol (that is, ICMP messages are encapsulated in IP datagrams). However, ICMP is an integral part of IP and must be implemented by every IP module.
- ICMP is used to report some errors, not to make IP reliable. Datagrams may still be undelivered without any report on their loss. Reliability must be implemented by the higher-level protocols that use IP.
- ICMP can report errors on any IP datagram with the exception of ICMP messages, to avoid infinite repetitions.
- For fragmented IP datagrams, ICMP messages are only sent about errors on fragment zero. That is, ICMP messages never refer to an IP datagram with a non-zero fragment offset field.
- ICMP messages are never sent in response to datagrams with a destination IP address that is a broadcast or a multicast address.
- ICMP messages are never sent in response to a datagram which does not have a source IP address which represents a unique host. That is, the source address cannot be zero, a loopback address, a broadcast address or a multicast address.
- ICMP messages are never sent in response to ICMP error messages. They may be sent in response to ICMP query messages (ICMP types 0, 8, 9, 10 and 13 through 18).
- RFC 792 states that ICMP messages “may” be generated to report IP datagram processing errors, not “must.” In practice, routers will almost always generate ICMP messages for errors, but for destination hosts, the number of ICMP messages generated is implementation dependent [18].

2.4 Ping

Ping is the simplest of all TCP/IP applications. It sends one or more IP datagrams to a specified destination host requesting a reply and measures the round trip time. The word ping, the abbreviation for Packet InterNet Groper. Figure 2.8 shows PING as an TCP/IP utility application.

Figure 2.9: Packet InterNet Groper (PING)



Traditionally, if you could ping a host other applications like Telnet or FTP could reach that host. Ping uses the ICMP Echo and Echo Reply messages. Since ICMP is required in every TCP/IP implementation, hosts do not require a separate server to respond to pings.

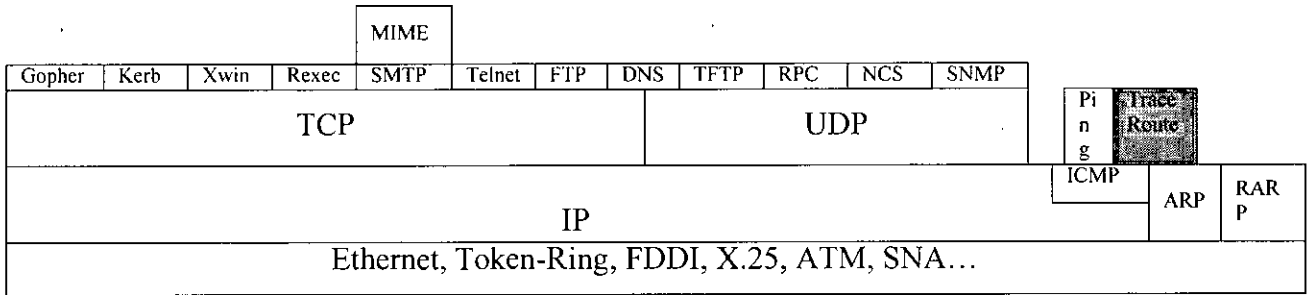
Ping is useful for verifying a TCP/IP installation. Consider the following four forms of the command; each requires the operation of an additional part of the TCP/IP installation.

- ping loopback* Verifies the operation of the base TCP/IP software.
- ping my-IP-address* Verifies whether the physical network device can be addressed.
- ping a-remote-IP-address* Verifies whether the network can be accessed.
- ping a-remote-host-name* Verifies the operation of the name server (or the flat namespace resolver, depending on the installation).

2.5 Traceroute

The Traceroute program can be useful when used for debugging purposes. Traceroute enables us to determine the route that IP datagrams follow from host to host. Figure 2.10 shows TraceRoute as a utility application of TCP/IP protocol suite.

Figure 2.10: Traceroute



Traceroute is based upon ICMP. It sends an IP datagram with a TTL of 1 to the destination host. The first router to see the datagram will decrement the TTL to 0 and return an ICMP Time Exceeded message as well as discarding the datagram. In this way, the first router in the path is identified. This process can be repeated with successively larger TTL values in order to identify the series of routers in the path to the destination host. Traceroute actually sends UDP datagrams to the destination host which reference a port number that is outside the normally used range

2.6 Address Resolution Protocol (ARP)

The ARP protocol is a *network-specific* standard protocol. Its status is *elective*, and the position of ARP in TCP/IP protocol suite is shown in Figure 2.10.

On a single physical network, individual hosts are known on the network by their physical hardware address. Higher-level protocols address destination hosts in the form of a symbolic address (IP address in this case). When such a protocol wants to send a datagram to destination IP address w.x.y.z, the device driver does not understand this address.

Therefore, a module (ARP) is provided that will translate the IP address to the physical address of the destination host. It uses a lookup table (sometimes referred to as the ARP cache) to perform this translation.

When the address is not found in the ARP cache, a broadcast is sent out on the network, with a special format called the ARP request. If one of the machines on the network recognizes its own IP address in the request, it will send an ARP reply back to the requesting host. The reply will contain the physical hardware address of the host and source route information (if the packet has crossed bridges on its path). Both

this address and the source route information are stored in the ARP cache of the requesting host. All subsequent datagrams to this destination IP address can now be translated to a physical address, which is used by the device driver to send out the datagram on the network.

ARP was designed to be used on networks that support hardware broadcast. This means, for example, that ARP will not work on an X.25 network.

2.6.1 ARP Detailed Concept

ARP is used on IEEE 802 networks as well as on the older DIX Ethernet networks to map IP addresses to physical hardware addresses. To do this, it is closely related to the device driver for that network. In fact, the ARP specifications only describe its functionality, not its implementation [19]. The implementation depends to a large extent on the device driver for a network type and they are usually coded together in the adapter microcode.

2.6.1.1 ARP Packet Generation

If an application wishes to send data to a certain IP destination address, the IP routing mechanism first determines the IP address of the “next hop” of the packet (it can be the destination host itself, or a router) and the hardware device on which it should be sent. If it is an IEEE 802.3/4/5 network, the ARP module must be consulted to map the <protocol type, target protocol address> to a physical address. Figure 2.11 shows the ARP request/reply packet structure.

Figure 2.11: ARP Request/Reply packet

physical layer header		x bytes
hardware address space		2 bytes
protocol address space		2 bytes
hardware address byte length (n)	protocol address byte length (m)	2 bytes
operation code		2 bytes
hardware address of sender		n bytes
protocol address of sender		m bytes
hardware address of target		n bytes
protocol address of target		m bytes

The ARP module tries to find the address in this ARP cache. If it finds the matching pair, it gives the corresponding 48-bit physical address back to the caller (the device driver) which then transmits the packet. If it doesn't find the pair in its table, it discards the packet (assumption is that a higher-level protocol will retransmit) and generates a network broadcast of an ARP request.

Where:

<i>hardware address space</i>	Specifies the type of hardware; examples are Ethernet
<i>protocol address space</i>	Specifies the type of protocol, same as EtherType field in the IEEE 802 header (IP or ARP).
<i>hardware address length</i>	Specifies the length (in bytes) of the hardware addresses in this packet. For IEEE 802.3 and IEEE 802.5 this will be 6.
<i>Protocol address length</i>	Specifies the length (in bytes) of the protocol addresses in this packet. For IP this will be 4.
<i>Operation code</i>	Specifies whether this is an ARP request (1) or reply (2).
<i>Source/target hardware address</i>	Contains the physical network hardware addresses. For IEEE 802.3 these are 48-bit addresses.
<i>Source/target protocol address</i>	Contains the protocol addresses. For TCP/IP these are the 32-bit IP addresses.

For the ARP request packet, the target hardware address is the only undefined field in the packet.

2.6.1.2 ARP Packet Reception

When a host receives an ARP packet (either a broadcast request or a point-to-point reply), the receiving device driver passes the packet to the ARP module which treats it as shown in Figure 2.12 algorithm.

Figure 2.12: ARP Packet Reception Algorithm

?Do I have the specified hardware type?

Yes: (almost definitely)

[optionally check the hardware length ar\$hln]

? Do I speak the specified protocol?

Yes:

[optionally check the protocol length]

Merge_flag := false

If the pair <protocol type, sender protocol address> is already in my translation table, update the sender hardware address field of the entry with the new information in the packet and set Merge_flag to true.

? Am I the target protocol address?

Yes:

If Merge_flag is false, add the triplet <protocol type, sender protocol address, sender hardware address> to the translation table.

? Is the opcode a REQUEST? (NOW look at the opcode!!)

Yes:

Swap source and target addresses in the ARP Packet

Put my local addresses in the source address fields

Send back ARP packet as an ARP

Reply to the requesting host

END

The requesting host will receive this ARP reply, and will follow the same algorithm to treat it. As a result of this, the triplet <protocol type, protocol address, hardware address> for the desired host will be added to its lookup table (ARP cache). The next time a higher-level protocol wants to send a packet to that host, the ARP module will find the target hardware address and the packet will be sent to that host.

Note that because the original ARP request was a broadcast on the network, all hosts on that network will have updated the sender's hardware address in their table [RFC 826]

2.6.2 ARP and Subnets

The ARP protocol remains unchanged in the presence of subnets. Remember that each IP datagram first goes through the IP routing algorithm. This algorithm selects the hardware device driver which should send out the packet. Only then, the ARP module associated with that device driver is consulted. [19]

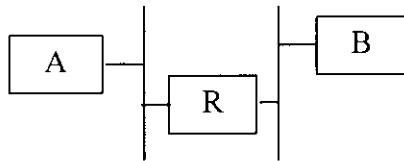
2.6.3 Proxy-ARP or Transparent Subnetting

Proxy-ARP is described in RFC 1027 — Using ARP to Implement Transparent Subnet Gateways, which is in fact a subset of the method proposed in RFC 925 — Multi-LAN Address Resolution [20]. It is another method to construct local subnets, without the need for a modification to the IP routing algorithm, but with modifications to the routers, which interconnect the subnets.

2.6.3.1 Proxy-ARP Concept

Consider one IP network, which is divided into subnets, interconnected by routers. We use the “old” IP routing algorithm, which means that no host knows about the existence of multiple physical networks. Consider in Figure 2.13, hosts A and B which are on different physical networks within the same IP network, and a router R between the two subnetworks:

Figure 2.13: Hosts Interconnected by a Router



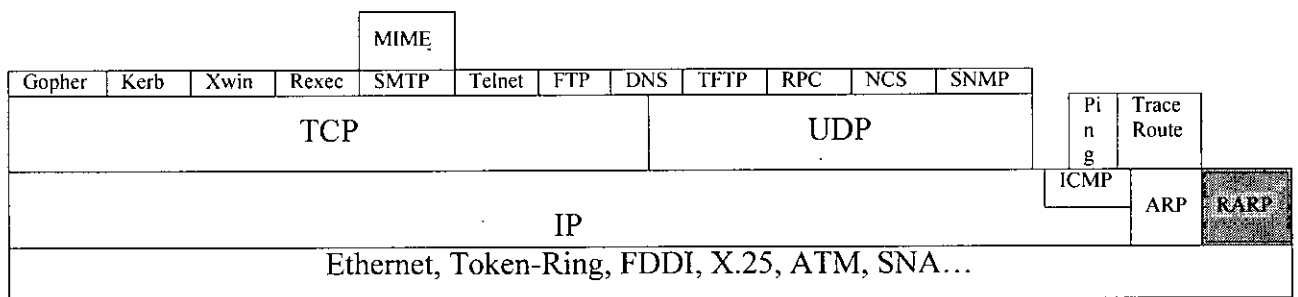
When host A wants to send an IP datagram to host B, it first has to determine the physical network address of host B through the use of the ARP protocol. As host A cannot differentiate between the physical networks, his IP routing algorithm thinks that host B is on the local physical network and sends out a broadcast ARP request. Host B does not receive this broadcast, but router R does. Router R understands subnets, that is, it runs the “subnet” version of the IP routing algorithm and it will be able to see that the destination of the ARP request (from the target protocol address field) is on another physical network. If router R’s routing tables specify that the next hop to that other network is through a different physical device, it will reply to the ARP as if it were host B, saying that the network address of host B is that of the router R itself.

Host A receives this ARP reply, puts it in his cache and will send future IP packets for host B to the router R. The router will forward such packets to the correct sub-net.

2.7 Reverse Address Resolution Protocol (RARP)

The RARP protocol is a network-specific standard protocol. Its status is elective. Some network hosts, such as diskless workstations, do not know their own IP address when they are booted. To determine their own IP address, they use a mechanism similar to ARP (Address Resolution Protocol), but now the hardware address of the host is the known parameter, and the IP address the queried parameter. It differs more fundamentally from ARP in the fact that a “RARP server” must exist on the network which maintains a database of mappings from hardware address to protocol address. Figure 2.14 shows the RARP in TCP/IP suite.

Figure 2.14: Reverse Address Resolution Protocol (RARP)



2.7.1 RARP Concept

The reverse address resolution is performed the same way as the ARP addresses resolution. The same packet format (see Figure 2.11) is used as for ARP.

An exception is the “operation code” field which now takes the following values:

- 3 for the RARP request
- 4 for the RARP reply

And of course, the “physical” header of the frame will now indicate RARP as the higher-level protocol (8035 hex) instead of ARP (0806 hex) or IP (0800 hex) in the EtherType field. Some differences arise from the concept of RARP itself:

- ARP only assumes that every host knows the mapping between its own hardware address and protocol address. RARP requires one or more server hosts on the network to maintain a database of mappings between hardware addresses and protocol addresses so that they will be able to reply to requests from client hosts.
- Due to the size this database can take, part of the server function is usually implemented outside the adapter’s microcode, with optionally a small cache in the microcode. The microcode part is then only responsible for reception and transmission of the RARP frames, the RARP mapping itself being taken care of by server software running as a normal process in the host machine.

- The nature of this database also requires some software to create and update the database manually.
- In case there are multiple RARP servers on the network, the RARP requester only uses the first RARP reply received on its broadcast RARP request, and discards the others.

2.8 Ports and Sockets

In this section we will introduce the concepts of port and socket.

2.8.1 Ports

Each process that wants to communicate with another process identifies itself to the TCP/IP protocol suite by one or more ports. A port is a 16-bit number, used by the host-to-host protocol to identify to which higher-level protocol or application program (process) it must deliver incoming messages.

As some higher-level programs are themselves protocols, standardized in the TCP/IP protocol suite, such as TELNET and FTP, they use the same port number in all TCP/IP implementations. Those “assigned” port numbers are called well-known ports and the standard applications well-known services.

The “well-known” ports are controlled and assigned by the Internet Assigned Numbers Authority (IANA) and on most systems can only be used by system processes or by programs executed by privileged users. The assigned “well-known” ports occupy port numbers in the range 0 to 1023. The ports with numbers in the range 1024-65535 are not controlled by the IANA and on most systems can be used by ordinary user-developed programs.

Confusion due to two different applications trying to use the same port numbers on one host is avoided by writing those applications to request an available port from TCP/IP. Because this port number is dynamically assigned, it may differ from one invocation of an application to the next.

2.8.2 Sockets

Let us first consider the following terminologies:

- A socket is a special type of file handle which is used by a process to request network services from the operating system.
- A socket address is the triple:
 - { protocol, local-address, local-process }
 - In the TCP/IP suite, for example:
 - { tcp, 193.44.234.3, 12345 }

- A conversation is the communication link between two processes.
- An association is the 5-tuple that completely specifies the two processes that comprise a connection:
 - { protocol, local-address, local-process, foreign-address, foreign-process}
 - In the TCP/IP suite, for example:
 - { tcp, 193.44.234.3, 1500, 193.44.234.5, 21 }
 could be a valid association.
- A half-association is either:
 - { protocol, local-address, local-process}
 - or
 - { protocol, foreign-address, foreign-process}

which specify each half of a connection.
- The half-association is also called a socket or a transport address. That is, a socket is an end point for communication that can be named and addressed in a network.

The socket interface is one of several application programming interfaces (APIs) to the communication protocols. Designed to be a generic communication programming interface, it was first introduced by the 4.2BSD UNIX system. Although it has not been standardized, it has become a de facto industry standard.

2.8.3 Basic Socket Calls

The following lists some basic socket interface calls.

- Initialize a socket

FORMAT: int sockfd = **socket** (int family, int type, int protocol)

where:

- family stands for addressing family. It can take on values such as AF_UNIX, AF_INET, AF_NS and AF_IUCV. Its purpose is to specify the method of addressing used by the socket.
- type stands for the type of socket interface to be used. It can take on values such as SOCK_STREAM, SOCK_DGRAM, SOCK_RAW, and SOCK_SEQPACKET.
- protocol can be UDP, TCP, IP or ICMP.
- sockfd is an integer (similar to a file descriptor) returned by the **socket** call.

- Bind (Register) a socket to a port address

FORMAT: int **bind** (int sockfd, struct sockaddr *localaddr, int addrlen)

where:

- sockfd is the same integer returned by the **socket** call.
- localaddr is the local address returned by the **bind** call.

Note that after the **bind** call, we now have values for the first three parameters inside our 5-tuple association:
{ protocol, local-address, local-process, foreign-address, foreign-process}

- Indicate readiness to receive connections

FORMAT: int **listen**(int sockfd, int queue-size)

where:

- sockfd is the same integer returned by the **socket** call.

- queue-size indicates the number of connection requests which can be queued by the system while the local process has not yet issued the **accept** call.

- Accept a connection

FORMAT: int **accept**(int sockfd, struct sockaddr *foreign-address, int addrlen)

where:

- sockfd is the same integer returned by the **socket** call.

- foreign-address is the address of the foreign (client) process returned by the **accept** call.

Note that this **accept** call is issued by a server process rather than a client process. If there is a connection request waiting on the queue for this socket connection, **accept** takes the first request on the queue and creates another socket with the same properties as sockfd; otherwise, **accept** will block the Callers process until a connection request arrives.

- Request connection to the server

FORMAT: int **connect** (int sockfd, struct sockaddr *foreign-address, int addrlen)

where:

- sockfd is the same integer returned by the **socket** call.

- foreign-address is the address of the foreign (server) process returned by the **connect** call.

Note that this call is issued by a client process rather than a server process.

- Send and/or receive data

The **read()**, **readv**(sockfd, char *buffer, int addrlen), **recv()**, **readfrom()**, **send**(sockfd, msg, len, flags), **write()** calls can be used to receive and send data in an established socket association (or connection).

Note that these calls are similar to the standard **read** and **write** file I/O System calls.

- Close a socket

FORMAT: int **close** (int sockfd)

where:

- sockfd is the same integer returned by the **socket** call.

2.8.4 Socket Interfaces

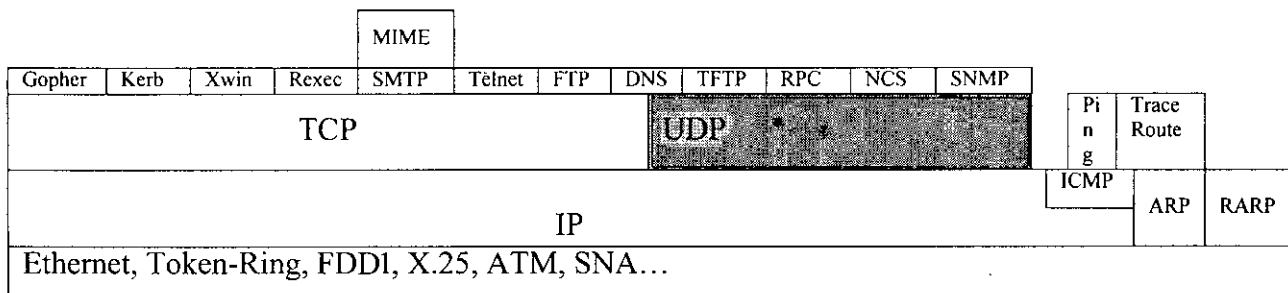
The socket interface is differentiated by the different services that are provided. Stream, datagram, and raw sockets each define a different service available to applications.

- **Stream socket interface (SOCK_STREAM):** It defines a reliable connection-oriented service (over TCP for example). Data is sent without errors or duplication and is received in the same order as it is sent. Flow control is built-in to avoid data overruns. No boundaries are imposed on the exchanged data, which is considered to be a stream of bytes. An example of an application that uses stream sockets is the File Transfer Program (FTP).
- **Datagram socket interface (SOCK_DGRAM):** It defines a connectionless service (over UDP for example). Datagrams are sent as independent packets. The service provides no guarantees; data can be lost or duplicated, and datagrams can arrive out of order. No disassembly and reassembly of packets is performed. An example of an application that uses datagram sockets is the Network File System (NFS).
- **Raw socket interface (SOCK_RAW):** It allows direct access to lower-layer protocols such as IP and ICMP. This interface is often used for testing new protocol implementations. An example of an application that uses raw sockets is the Ping command.

2.9 User Datagram Protocol (UDP)

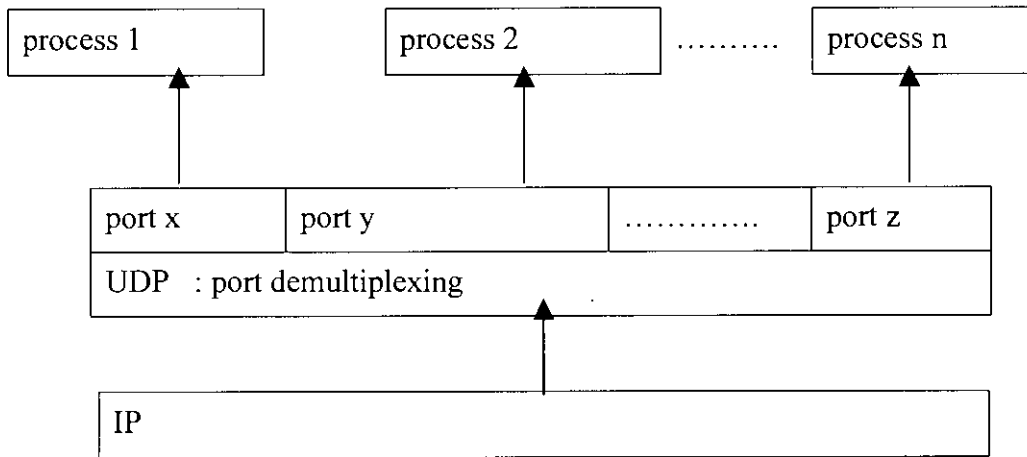
UDP is a standard protocol with STD number 6 is shown in Figure 2.15. UDP is described by RFC 768 — User Datagram Protocol [21]. Its status is recommended, but in practice every TCP/IP implementation which is not used exclusively for routing will include UDP.

Figure 2.15: User Datagram Protocol (UDP)



UDP is basically an application interface to IP. It adds no reliability, flow-control or error recovery to IP. It simply serves as a “multiplexer/demultiplexer” for sending and receiving datagrams, using ports to direct the datagrams as shown in Figure 2.16.

Figure 2.16: UDP, A Demultiplexer Based on Ports



UDP provides a mechanism for one application to send a datagram to another. The UDP layer can be regarded as being extremely thin and consequently has low overheads, but it requires the application to take responsibility for error recovery and so on.

2.9.1 Ports

The port concept was discussed earlier in 2.8, “Ports and Sockets”. Applications sending datagrams to a host need to identify a target which is more specific than the IP address, since datagrams are normally directed to certain processes and not to the system as a whole. UDP provides this by using ports. A port is a 16-bit number, which identifies, which process on a host is associated with a datagram. There are two types of port:

well-known Well-known ports belong to standard servers; for example TELNET uses port 23. Well-known port numbers range between 1 and 1023 (prior to 1992, the range between 256 and 1023 was used for UNIX-specific servers). Well-known port numbers are typically odd, because early systems using the port concept required an odd/even pair of ports for duplex operations. Most servers require only a single port. An exception is the BOOTP server, which uses two: 67 and 68. The reason for well-known ports is to allow clients to be able to find servers without configuration information. The well-known port numbers are defined in STD 2 — Assigned Numbers [17].

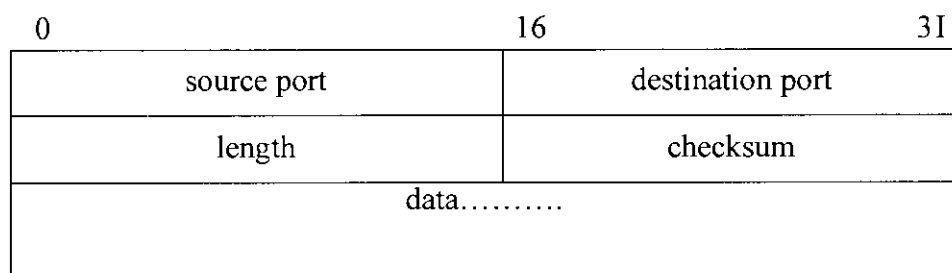
ephemeral Clients do not need well-known port numbers because they initiate communication with servers and the port number they are using is contained in the UDP datagrams sent to the server. Each client process is allocated a port number as long as it needs it by the host it is running on. Ephemeral port numbers have values greater than 1023, normally in the range 1024 to 5000. A client can use any number allocated to it, as long as the combination of <transport protocol, IP address, port number> is unique.

TCP also uses port numbers with the same values. These ports are quite independent. Normally, a server will use either TCP or UDP, but there are exceptions. For example, Domain Name Servers use both UDP port 53 and TCP port 53.

2.9.2 UDP Datagram Format

Each UDP datagram is sent within a single IP datagram. Although, the IP datagram may be fragmented during transmission, the receiving IP implementation will re-assemble it before presenting it to the UDP layer. All IP implementations are required to accept datagrams of 576 bytes, which, allowing for maximum-size IP header of 60 bytes means that a UDP datagram of 516 bytes is acceptable to all implementations. Many implementations will accept larger datagrams, but this is not guaranteed. The UDP datagram has a 16-byte header which is described in Figure 2.17.

Figure 2.17: UDP Datagram Format



Where:

source Port Indicates the port of the sending process. It is the port to which replies should be addressed.

destination Port Specifies the port of the destination process on the destination host.

length Is the length (in bytes) of this user datagram including the header.

checksum Is an optional 16-bit one's complement of the one's complement sum of a pseudo-IP header, the UDP header and the UDP data. The pseudo-IP header contains the source and destination IP addresses, the protocol and the UDP length:

2.9.3 UDP Application Programming Interface

The application interface offered by UDP is described in RFC 768. It provides for:

- The creation of new receive ports.
- Receive operation that returns the data bytes and an indication of source port and source IP address.
- Send operation that has as parameters the data, source and destination ports and addresses.

Be aware that UDP and IP do not provide guaranteed delivery, flow-control or error recovery, so these must be provided by the application.

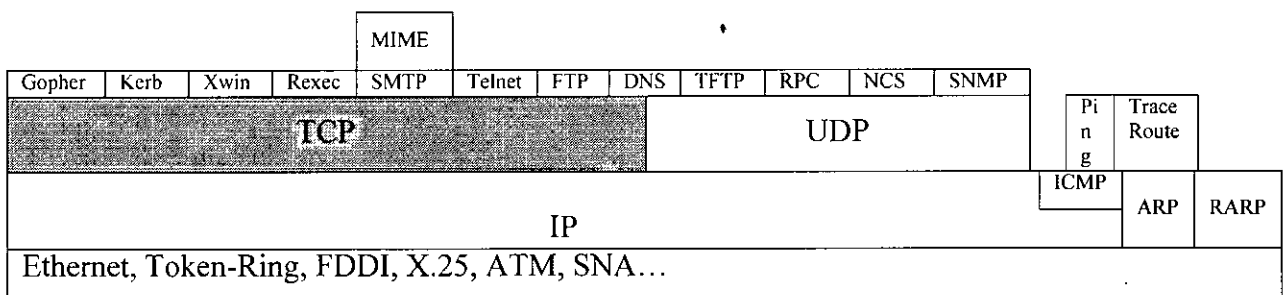
Standard applications using UDP include:

- Trivial File Transfer Protocol (TFTP)
- Domain Name System (DNS) name server
- Remote Procedure Call (RPC), used by the Network File System (NFS)
- Network Computing System (NCS)
- Simple Network Management Protocol (SNMP)

2.10 Transmission Control Protocol (TCP)

TCP is a standard protocol with STD number 7 [22]. TCP is described by RFC 793 — Transmission Control Protocol and as shown in Figure 2.18. Its status is recommended, but in practice every TCP/IP implementation which is not used exclusively for routing will include TCP.

Figure 2.18 Transmission Control Protocol (TCP)



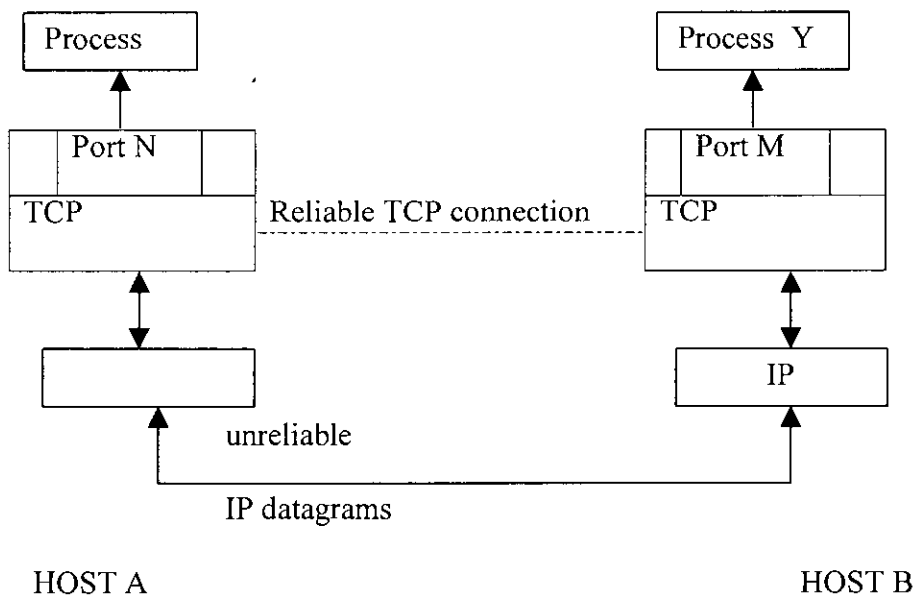
TCP provides considerably more facilities for applications than UDP, notably error recovery, flow control and reliability. TCP is a connection-oriented protocol unlike UDP which is connectionless. Most of the user application protocols, such as TELNET and FTP, use TCP.

2.10.1 Sockets

The socket concept was discussed earlier in 2.8, "Ports and Sockets". Two processes communicate via TCP sockets. The socket model provides a process with a full-duplex byte stream connection to another process. The application need not concern itself with the management of this stream; these facilities are provided by TCP.

TCP uses the same port principle as UDP (see 2.9.1, "Ports") to provide multiplexing. Like UDP, TCP uses well-known and ephemeral ports. Each side of a TCP connection has a socket, which can be identified by the triple <TCP, IP address, port number>. This is also called a half-association. If two processes are communicating over TCP, they have a logical connection that is uniquely identifiable by the two sockets involved, that is by the combination <TCP, local IP address, local port, remote IP address, remote port>. See Figure 2.19. Server processes are able to manage multiple conversations through a single port.

Figure 2.19: TCP Connection.



Processes X and Y communicate over a TCP connection carried by IP datagrams.

2.10.2 TCP Concept

As noted above, the primary purpose of TCP is to provide reliable logical circuit or connection service between pairs of processes. It does not assume reliability from the lower-level protocols such as IP (see Figure 2.18). Therefore, TCP must guarantee this itself. TCP can be characterized by the following facilities it provides for the applications using it:

- Stream Data Transfer
- Reliability
- Flow Control
- Multiplexing
- Logical Connections

2.10.2.1 Stream Data Transfer

From the application's viewpoint, TCP transfers a contiguous stream of bytes through the internet. The application does not have to bother with chopping the data into basic blocks or datagrams. TCP does this by grouping the bytes in TCP segments, which are passed to IP for transmission to the destination. Also, TCP itself decides how to segment the data and it may forward the data at its own convenience. Sometimes, an application needs to be sure that all the data passed to TCP has actually been transmitted to the destination. For that reason, a push function is defined. It will push all remaining TCP segments still in storage to the destination host. The normal close connection function also pushes the data to the destination.

2.10.2.2 Reliability

TCP assigns a sequence number to each byte transmitted, and expect a positive acknowledgment (ACK) from the receiving TCP. If the ACK is not received within a timeout interval, the data is retransmitted. As the data is transmitted in blocks (TCP segments) only the sequence number of the first data byte in the segment is sent to the destination host. The receiving TCP uses the sequence numbers to rearrange the segments when they arrive out of order, and to eliminate duplicate segments.

2.10.2.3 Flow Control

The receiving TCP, when sending an ACK back to the sender, also indicates to the sender the number of bytes it can receive beyond the last received TCP segment, without causing overrun and overflow in its internal buffers. This is sent in the ACK in the form of the highest sequence number it can receive without problems. This mechanism is also referred to as a window-mechanism.

2.10.2.4 Multiplexing

To allow for many processes within a single Host to use TCP communication facilities simultaneously, the TCP provides a set of addresses or ports within each host. Concatenated with the network and host addresses from the internet communication layer, this forms a socket. A pair of sockets uniquely identifies each connection. That is, a socket may be simultaneously used in multiple connections [22].

The binding of ports to processes is handled independently by each Host. However, it proves useful to attach frequently used processes (e.g., a "logger" or timesharing service) to fixed sockets which are made known to the public. These services can then be accessed through the known addresses. Establishing and learning the port addresses of other processes may involve more dynamic mechanisms.

2.10.2.5 Logical Connections

The reliability and flow control mechanisms described above require that TCP initializes and maintains certain status information for each "data stream." The combination of this status, including sockets, sequence numbers and window sizes, is called a logical connection. Each connection is uniquely identified by the pair of sockets used by the sending and receiving processes.

Chapter 3

Structure of the Domain Name Space

3.1 Introduction

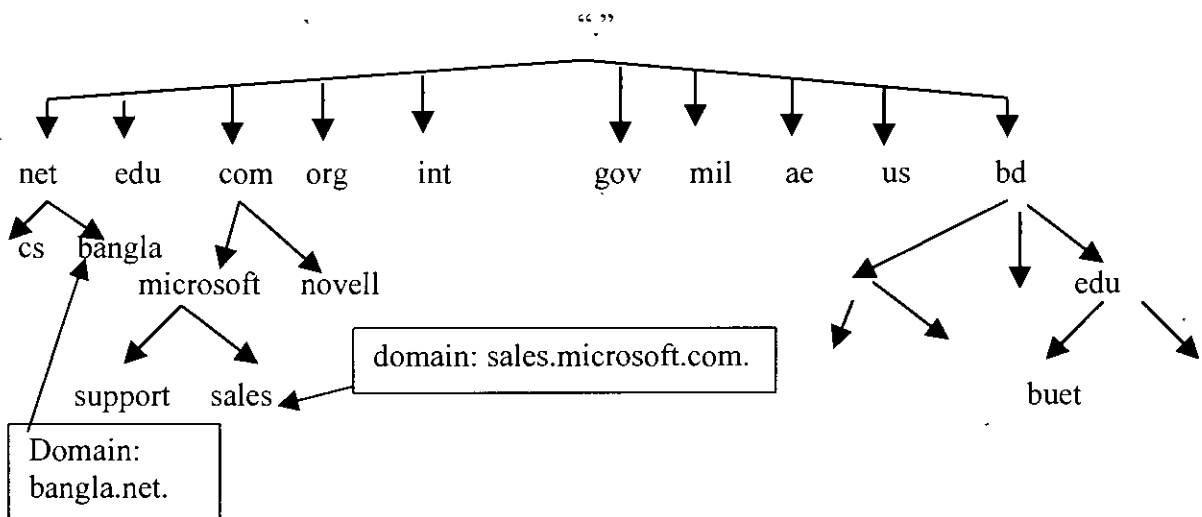
The domain system is a tree-structured global name space that has a few top-level domains. The top-level domains are subdivided into second level domains. The second level domains may be subdivided into third level domains, and so on.

The administration of a domain requires controlling the assignment of names within that domain and providing access to the names and name related information (such as addresses) to users both inside and outside the domain.

Technically the Domain Name System (DNS) is a distributed database. This allows local control of the segments of the overall database, yet data in each segment are available across the entire network through a client-server scheme. Robustness and adequate performance are achieved through replication and caching [23].

Programs called *Name Servers* constitute the server half of DNS's client-server mechanism. Name servers contain information about some segment of the database and make it available to clients, called *Resolvers*. Resolvers are often just library routines that create queries and send them across a network to a Name Server [24].

Figure 3.1: Domain Name Space Tree



The structure of the DNS database, shown in Figure 3.1, is very similar to the structure of the UNIX file system. The whole database is pictured as an inverted tree,

with the root at the top. In DNS, the root's name is the null label (''), but is written as a single dot (".") in text.

Each node in the tree represents a partition of the overall database—a domain in the Domain Name System. Each domain can be further divided into partitions, called sub-domains in DNS. A domain also has a domain name, which identifies its position in the database. In DNS, the full domain name is the sequence of labels from the domain to the root, with "." separating the labels.

3.2 Elements of the DNS

The DNS has three major components [25]:

- The DOMAIN NAME SPACE and RESOURCE RECORDS, which are specifications for a tree structured name space and data associated with the names. Conceptually, each node and leaf of the domain name space tree names a set of information, and query operations are attempts to extract specific types of information from a particular set. A query names the domain name of interest and describes the type of resource information that is desired. For example, the Internet uses some of its domain names to identify hosts; queries for address resources return Internet host addresses.

- NAME SERVERS are server programs, which hold information about the domain tree's structure and set information. A name server may cache structure or set information about any part of the domain tree, but in general a particular name server has complete information about a subset of the domain space, and pointers to other name servers that can be used to lead to information from any part of the domain tree. Name servers know the parts of the domain tree for which they have complete information; a name server is said to be an AUTHORITY for these parts of the name space. Authoritative information is organized into units called ZONES, and these zones can be automatically distributed to the name servers, which provide redundant service for the data in a zone.

- RESOLVERS are programs that extract information from name servers in response to client requests. Resolvers must be able to access at least one name server and use that name server's information to answer a query directly, or pursue the query using referrals to other name servers. A resolver will typically be a system routine that is directly accessible to user programs; hence no protocol is necessary between the resolver and the user program.

3.3 Domain Name Space and Resource Records

3.3.1 Name space specifications and terminology

The domain name space is a tree structure. Each node and leaf on the tree corresponds to a resource set (which may be empty). The domain system makes no distinctions between the uses of the interior nodes and leaves, and this research uses the term "node" to refer to both.

Each node has a label, which is zero to 63 octets in length. Brother nodes may not have the same label, although the same label can be used for nodes which are not brothers. One label is reserved, and that is the null (i.e., zero length) label used for the root.

The domain name of a node is the list of the labels on the path from the node to the root of the tree. By convention, the labels that compose a domain name are printed or read left to right, from the most specific (lowest, farthest from the root) to the least specific (highest, closest to the root).

Internally, programs that manipulate domain names should represent them as sequences of labels, where each label is a length octet followed by an octet string. Because all domain names end at the root, which has a null string for a label, these internal representations can use a length byte of zero to terminate a domain name.

By convention, domain names can be stored with arbitrary case, but domain name comparisons for all present domain functions are done in a case-insensitive manner, assuming an ASCII character set, and a high order zero bit. This means that you are free to create a node with label "A" or a node with label "a", but not both as brothers; you could refer to either using "a" or "A". When you receive a domain name or label, you should preserve its case. The rationale for this choice is that we may someday need to add full binary domain names for new services; existing services would not be changed.

When a user needs to type a domain name, the length of each label is omitted and the labels are separated by dots ("."). Since a complete domain name ends with the root label, this leads to a printed form which ends in a dot. We use this property to distinguish between:

- a character string which represents a complete domain name (often called "absolute"). For example, "buet.edu.bd."
- a character string that represents the starting labels of a domain name which is incomplete, and should be completed by local software using knowledge of the local domain (often called "relative"). For example, "buet" used in the edu.bd. domain.

Relative names are either taken relative to a well known origin, or to a list of domains used as a search list. Relative names appear mostly at the user interface, where their

interpretation varies from implementation to implementation, and in master files, where they are relative to a single origin domain name. The most common interpretation uses the root "." as either the single origin or as one of the members of the search list, so a multi-label relative name is often one where the trailing dot has been omitted to save typing.

To simplify implementations, the total number of octets that represent a domain name (i.e., the sum of all label octets and label lengths) is limited to 255 [24].

3.3.2 Preferred name syntax

The DNS specifications attempt to be as general as possible in the rules for constructing domain names. The idea is that the name of any existing object can be expressed as a domain name with minimal changes. However, when assigning a domain name for an object, the prudent user will select a name, which satisfies both the rules of the domain system and any existing rules for the object, whether these rules are published or implied by existing programs.

For example, when naming a mail domain, the user should satisfy both the rules of this thesis and those in RFC-822. When creating a new host name, the old rules for HOSTS.TXT should be followed. This avoids problems when old software is converted to use domain names.

The following syntax will result in fewer problems with many applications that use domain names (e.g., mail, TELNET).

`<domain> ::= <subdomain> | " "`

`<subdomain> ::= <label> | <subdomain> "." <label>`

`<label> ::= <letter> [[<ldh-str>] <let-dig>]`

`<ldh-str> ::= <let-dig-hyp> | <let-dig-hyp> <ldh-str>`

`<let-dig-hyp> ::= <let-dig> | "-"`

`<let-dig> ::= <letter> | <digit>`

`<letter> ::= any one of the 52 alphabetic characters "A" through "Z" in upper case and "a" through "z" in lower case`

`<digit> ::= any one of the ten digits 0 through 9`

Note that while upper and lower case letters are allowed in domain names, no significance is attached to the case. That is, two names with the same spelling but different case are to be treated as if identical.

The labels must follow the rules for ARPANET host names. They must start with a letter, end with a letter or digit, and have as interior characters only letters, digits, and hyphen. There are also some restrictions on the length. Labels must be 63 characters or less.

3.3.3 Resource Records

A domain name identifies a node. Each node has a set of resource information, which may be empty. The set of resource information associated with a particular name is composed of separate resource records (RRs). The order of RRs in a set is not significant, and need not be preserved by name servers, resolvers, or other parts of the DNS [38].

When we talk about a specific RR, we assume it has the following:

- owner* which is the domain name where the RR is found.
- type* which is an encoded 16 bit value that specifies the type of the resource in this resource record. Types refer to abstract resources.

This research uses the following types:

- A a host address
- CNAME identifies the canonical name of an alias
- HINFO identifies the CPU and OS used by a host
- MX identifies a mail exchange for the domain.
 See [RFC-974 for details].
- NS the authoritative name server for the domain
- PTR a pointer to another part of the domain name space
- SOA identifies the start of a zone of authority

- class* which is an encoded 16 bit value which identifies a protocol family or instance of a protocol.

This research uses the following classes:

- IN the Internet system
- CH the Chaos system

- TTL which is the time to live of the RR. This field is a 32 bit integer in units of seconds, and is primarily used by resolvers when they cache RRs. The TTL describes how long a RR can be cached before it should be discarded.

RDATA which is the type and sometimes class dependent data which describes the resource:

A For the IN class, a 32 bit IP address

For the CH class, a domain name followed by a 16 bit octal Chaos address.

CNAME a domain name.

MX a 16 bit preference value (lower is better) followed by a host name willing to act as a mail exchange for the owner domain.

NS a host name.

PTR a domain name.

SOA several fields.

The owner name is often implicit, rather than forming an integral part of the RR. For example, many name servers internally form tree or hash structures for the name space, and chain RRs off nodes. The remaining RR parts are the fixed header (type, class, TTL) which is consistent for all RRs, and a variable part (RDATA) that fits the needs of the resource being described.

The meaning of the TTL field is a time limit on how long an RR can be kept in a cache. This limit does not apply to authoritative data in zones; it is also timed out, but by the refreshing policies for the zone. The TTL is assigned by the administrator for the zone where the data originates. While short TTLs can be used to minimize caching, and a zero TTL prohibits caching, the realities of Internet performance suggest that these times should be on the order of days for the typical host. If a change can be anticipated, the TTL can be reduced prior to the change to minimize inconsistency during the change, and then increased back to its former value following the change.

The data in the RDATA section of RRs is carried as a combination of binary strings and domain names. The domain names are frequently used as "pointers" to other data in the DNS.

3.3.4 Textual expression of RRs

RRs are represented in binary form in the packets of the DNS protocol, and are usually represented in highly encoded form when stored in a name server or resolver. In this research, we adopt a style similar to that used in master files in order to show the contents of RRs. In this format, most RRs are shown on a single line, although continuation lines are possible using parenthesis [25].

The start of the line gives the owner of the RR. If a line begins with a blank, then the owner is assumed to be the same as that of the previous RR. Blank lines are often included for readability.

Following the owner, we list the TTL, type, and class of the RR. Class and type use the mnemonics defined above, and TTL is an integer before the type field. In order to avoid ambiguity in parsing, type and class mnemonics are disjoint, TTLs are integers, and the type mnemonic is always last. The IN class and TTL values are often omitted from examples in the interests of clarity.

The resource data or RDATA section of the RR is given using knowledge of the typical representation for the data.

For example, we might show the RRs carried in a message as:

```
EDU.BD.           MX  10 BUET.EDU.BD.
                  MX  10 IIT.EDU.BD.
BUET.EDU.BD.     A   w.x.y.z1
                  A   w.x.y.z2
IIT.EDU.BD.      A   w.x.y.z3
                  A   w.x.y.z4
```

The MX RRs have an RDATA section which consists of a 16 bit number followed by a domain name. The address RRs use a standard IP address format to contain a 32 bit internet addresses.

This example shows six RRs, with two RRs at each of three domain names.

3.3.5 Aliases and canonical names

In existing systems, hosts and other resources often have several names that identify the same resource. For example, the names C.ISI.EDU and USC-ISIC.ARPA both identify the same host. Similarly, in the case of mailboxes, many organizations provide many names that actually go to the same mailbox; for example ofarooq@C.ISI.EDU.BD, wzaman@B.ISI.EDU.BD, and PVM@ISI.EDU.BD all go to the same mailbox (although the mechanism behind this is somewhat complicated).

Most of these systems have a notion that one of the equivalent sets of names is the canonical or primary name and all others are aliases.

The domain system provides such a feature using the canonical name (CNAME) RR. A CNAME RR identifies its owner name as an alias, and specifies the corresponding canonical name in the RDATA section of the RR. If a CNAME RR is present at a node, no other data should be present; this ensures that the data for a canonical name and its aliases cannot be different. This rule also insures that a cached CNAME can be used without checking with an authoritative server for other RR types [26].

CNAME RRs causes special action in DNS software. When a name server fails to find a desired RR in the resource set associated with the domain name, it checks to

see if the resource set consists of a CNAME record with a matching class. If so, the name server includes the CNAME record in the response and restarts the query at the domain name specified in the data field of the CNAME record. The one exception to this rule is that queries which match the CNAME type are not restarted.

For example, suppose a name server was processing a query with for USC-ISIC.ARPA, asking for type A information, and had the following resource records:

```
USC-ISIC.ARPA IN  CNAME  C.ISI.EDU
C.ISI.EDU      IN  A      10.0.0.52
```

Both of these RRs would be returned in the response to the type A query, while a type CNAME or * query should return just the CNAME.

Domain names in RRs which point at another name should always point at the primary name and not the alias. This avoids extra instructions in accessing information. For example, the address to name RR for the above host should be:

```
52.0.0.10.IN-ADDR.ARPA IN  PTR  C.ISI.EDU
```

rather than pointing at USC-ISIC.ARPA. Of course, by the robustness principle, domain software should not fail when presented with CNAME chains or loops; CNAME chains should be followed and CNAME loops signalled as an error.

3.3.6 Queries

Queries are messages, which may be sent to a name server to provoke a response. In the Internet, queries are carried in UDP datagrams or over TCP connections. The response by the name server either answers the question posed in the query, refers the requester to another set of name servers, or signals some error condition.

In general, the user does not generate queries directly, but instead makes a request to a resolver which in turn sends one or more queries to name servers and deals with the error conditions and referrals that may result. Of course, the possible questions which can be asked in a query does shape the kind of service a resolver could provide.

DNS queries and responses are carried in a standard message format. The message format has a header containing a number of fixed fields which are always present, and four sections which carry query parameters and RRs.

The most important field in the header is a four bit field called an *opcode* which separates different queries. Of the possible 16 values, one (standard query) is part of the official protocol, two (inverse query and status query) are options, one (completion) is obsolete, and the rest are unassigned.

The four sections are:

Question	Carries the query name and other query parameters.
----------	--

Answer	Carries RRs which directly answer the query.
Authority	Carries RRs which describe other authoritative servers. May optionally carry the SOA RR for the authoritative data in the answer section.
Additional	Carries RRs which may be helpful in using the RRs in the other sections.

Note that the content, but not the format, of these sections varies with header opcode.

3.4 Name Servers

3.4.1 Introduction

Name servers are the repositories of information that make up the domain database. The database is divided up into sections called zones, which are distributed among the name servers. While name servers can have several optional functions and sources of data, the essential task of a name server is to answer queries using data in its zones. By design, name servers can answer queries in a simple manner; the response can always be generated using only local data, and either contains the answer to the question or a referral to other name servers "closer" to the desired information [27].

A given zone will be available from several name servers to insure its availability in spite of host or communication link failure. By administrative fiat, we require every zone to be available on at least two servers, and many zones have more redundancy than that.

A given name server will typically support one or more zones, but this gives it authoritative information about only a small section of the domain tree. It may also have some cached non-authoritative data about other parts of the tree. The name server marks its responses to queries so that the requester can tell whether the response comes from authoritative data or not.

3.4.2 How the database is divided into zones

The domain database is partitioned in two ways: by class, and by "cuts" made in the name space between nodes.

The class partition is simple. The database for any class is organized, delegated, and maintained separately from all other classes. Since, by convention, the name spaces are the same for all classes, the separate classes can be thought of as an array of parallel namespace trees. Note that the data attached to nodes will be different for

these different parallel classes. The most common reasons for creating a new class are the necessity for a new data format for existing types or a desire for a separately managed version of the existing name space [28].

Within a class, "cuts" in the name space can be made between any two adjacent nodes. After all cuts are made, each group of connected name space is a separate zone. The zone is said to be authoritative for all names in the connected region. Note that the "cuts" in the name space may be in different places for different classes, the name servers may be different, etc.

These rules mean that every zone has at least one node, and hence domain name, for which it is authoritative, and all of the nodes in a particular zone are connected. Given, the tree structure, every zone has a highest node which is closer to the root than any other node in the zone. The name of this node is often used to identify the zone.

It would be possible, though not particularly useful, to partition the name space so that each domain name was in a separate zone or so that all nodes were in a single zone. Instead, the database is partitioned at points where a particular organization wants to take over control of a subtree. Once an organization controls its own zone it can unilaterally change the data in the zone, grow new tree sections connected to the zone, delete existing nodes, or delegate new subzones under its zone.

3.4.3 Technical considerations for Name Server

The data that describes a zone has four major parts:

- Authoritative data for all nodes within the zone.
- Data that defines the top node of the zone (can be thought of as part of the authoritative data).
- Data that describes delegated subzones, i.e., cuts around the bottom of the zone.
- Data that allows access to name servers for subzones (sometimes called "glue" data).

All of this data is expressed in the form of RRs, so a zone can be completely described in terms of a set of RRs. Whole zones can be transferred between name servers by transferring the RRs, either carried in a series of messages or by FTPing a master file which is a textual representation.

The authoritative data for a zone is simply all of the RRs attached to all of the nodes from the top node of the zone down to leaf nodes or nodes above cuts around the bottom edge of the zone.

Though logically part of the authoritative data, the RRs that describe the top node of the zone are especially important to the zone's management. These RRs are of two types: name server RRs that list, one per RR, all of the servers for the zone, and a single SOA RR that describes zone management parameters.

The RRs that describe cuts around the bottom of the zone are NS RRs that name the servers for the subzones. Since the cuts are between nodes, these RRs are NOT part of the authoritative data of the zone, and should be exactly the same as the corresponding RRs in the top node of the subzone. Since name servers are always associated with zone boundaries, NS RRs are only found at nodes which are the top node of some zone. In the data that makes up a zone, NS RRs are found at the top node of the zone (and are authoritative) and at cuts around the bottom of the zone (where they are not authoritative), but never in between.

One of the goals of the zone structure is that any zones have all the data required to set up communications with the name servers for any subzones. That is, parent zones have all the information needed to access servers for their children zones. The NS RRs that name the servers for subzones are often not enough for this task since they name the servers, but do not give their addresses. In particular, if the name of the name server is itself in the subzone, we could be faced with the situation where the NS RRs tell us that in order to learn a name server's address, we should contact the server using the address we wish to learn. To fix this problem, a zone contains "glue" RRs which are not part of the authoritative data, and are address RRs for the servers. These RRs are only necessary if the name server's name is "below" the cut, and are only used as part of a referral response.

3.4.4 Name server internals

3.4.4.1 Queries and responses

The principal activity of name servers is to answer standard queries. Both the query and its response are carried in a standard message format, which is described in [RFC-1035]. The query contains a QTYPE, QCLASS, and QNAME, which describe the types and classes of desired information and the name of interest.

The way that the name server answers the query depends upon whether it is operating in recursive mode or not:

- The simplest mode for the server is non-recursive, since it can answer queries using only local information: the response contains an error, the answer, or a referral to some other server "closer" to the answer. All name servers must implement non-recursive queries.
- The simplest mode for the client is recursive, since in this mode the name server acts in the role of a resolver and returns either an error or the answer, but never referrals. This service is optional in a name server, and the name server may also choose to restrict the clients, which can use recursive mode.

Recursive service is helpful in several situations:

- a relatively simple requester that lacks the ability to use anything other than a direct answer to the question.
- a request that needs to cross protocol or other boundaries and can be sent to a server which can act as intermediary.
- a network where we want to concentrate the cache rather than having a separate cache for each client.

Non-recursive service is appropriate if the requester is capable of pursuing referrals and interested in information which will aid future requests.

The use of recursive mode is limited to cases where both the client and the name server agree to its use. The agreement is negotiated through the use of two bits in query and response messages:

- The recursion available, or RA bit, is set or cleared by a name server in all responses. The bit is true if the name server is willing to provide recursive service for the client, regardless of whether the client requested recursive service. That is, RA signals availability rather than use.
- Queries contain a bit called recursion desired or RD. This bit specifies whether the requester wants recursive service for this query. Clients may request recursive service from any name server, though they should depend upon receiving it only from servers which have previously sent an RA, or servers which have agreed to provide service through private agreement or some other means outside of the DNS protocol.

The recursive mode occurs when a query with RD set arrives at a server which is willing to provide recursive service; the client can verify that recursive mode was used by checking that both RA and RD are set in the reply. Note that the name server should never perform recursive service unless asked via RD, since this interferes with trouble shooting of name servers and their databases.

If recursive service is requested and available, the recursive response to a query will be one of the following:

- The answer to the query possibly prefaces by one or more CNAME RRs that specify aliases encountered on the way to an answer.
- A name error indicating that the name does not exist. This may include CNAME RRs that indicates that the original query name was an alias for a name which does not exist.
- A temporary error indication.

If recursive service is not requested or is not available, the non-recursive response will be one of the following:

- An authoritative name error indicating that the name does not exist.
- A temporary error indication.
- Some combination of:

RRs that answer the question, together with an indication whether the data comes from a zone or is cached.

A referral to name servers which have zones which are closer ancestors to the name than the server sending the reply.

RRs that the name server thinks will prove useful to the requester.

3.4.5 Zone maintenance and transfers

The general model of automatic zone transfer or refreshing is that one of the name servers is the master or primary for the zone. Changes are co-ordinated at the primary, typically by editing a master file for the zone. After editing, the administrator signals the master server to load the new zone. The other non-master or secondary servers for the zone periodically check for changes (at a selectable interval) and obtain new zone copies when changes have been made [28].

To detect changes, secondaries just check the SERIAL field of the SOA for the zone. In addition to whatever other changes are made, the SERIAL field in the SOA of the zone is always advanced whenever any change is made to the zone. The advancing can be a simple increment, or could be based on the write date and time of the master file, etc. The purpose is to make it possible to determine which of two copies of a zone is more recent by comparing serial numbers. Serial number advances and comparisons use sequence space arithmetic, so there is a theoretic limit on how fast a zone can be updated, basically that old copies must die out before the serial number covers half of its 32 bit range. In practice, the only concern is that the compare operation deals properly with comparisons around the boundary between the most positive and most negative 32 bit numbers [29].

The periodic polling of the secondary servers is controlled by parameters in the SOA RR for the zone, which set the minimum acceptable polling intervals. The parameters are called REFRESH, RETRY, and EXPIRE. Whenever a new zone is loaded in a secondary, the secondary waits REFRESH seconds before checking with the primary for a new serial. If this check cannot be completed, new checks are started every RETRY seconds. The check is a simple query to the primary for the SOA RR of the zone. If the serial field in the secondary's zone copy is equal to the serial returned by the primary, then no changes have occurred, and the REFRESH interval wait is

restarted. If the secondary finds it impossible to perform a serial check for the EXPIRE interval, it must assume that its copy of the zone is obsolete and discard it.

When the poll shows that the zone has changed, then the secondary server must request a zone transfer via an AXFR request for the zone. The AXFR may cause an error, such as refused, but normally is answered by a sequence of response messages. The first and last messages must contain the data for the top authoritative node of the zone. Intermediate messages carry all of the other RRs from the zone, including both authoritative and non-authoritative RRs. The stream of messages allows the secondary to construct a copy of the zone. Because accuracy is essential, TCP or some other reliable protocol must be used for AXFR requests.

Each secondary server is required to perform the following operations against the master, but may also optionally perform these operations against other secondary servers. This strategy can improve the transfer process when the primary is unavailable due to host downtime or network problems, or when a secondary server has better network access to an "intermediate" secondary than to the primary.

3.5 Resolvers

3.5.1 Introduction

Resolvers are programs that interface user programs to domain name servers. In the simplest case, a resolver receives a request from a user program (e.g., mail programs, TELNET, FTP) in the form of a subroutine call, system call etc., and returns the desired information in a form compatible with the local host's data formats.

The resolver is located on the same machine as the program that requests the resolver's services, but it may need to consult name servers on other hosts. Because a resolver may need to consult several name servers, or may have the requested information in a local cache, the amount of time that a resolver will take to complete can vary quite a bit, from milliseconds to several seconds.

A very important goal of the resolver is to eliminate network delay and name server load from most requests by answering them from its cache of prior results. It follows that caches which are shared by multiple processes, users, machines, etc., are more efficient than non-shared caches.

3.5.2 Client-resolver interface

3.5.2.1 Typical functions

The client interface to the resolver is influenced by the local host's conventions, but the typical resolver-client interface has three functions:

Host name to host address translation.

This function is often defined to mimic a previous HOSTS.TXT based function. Given a character string, the caller wants one or more 32 bit IP addresses. Under the DNS, it translates into a request for type A RRs. Since the DNS does not preserve the order of RRs, this function may choose to sort the returned addresses or select the "best" address if the service returns only one choice to the client. Note that a multiple address return is recommended, but a single address may be the only way to emulate prior HOSTS.TXT services [39].

Host address to host name translation

This function will often follow the form of previous functions. Given a 32 bit IP address, the caller wants a character string. The octets of the IP address are reversed, used as name components, and suffixed with "IN-ADDR.ARPA". A type PTR query is used to get the RR with the primary name of the host. For example, a request for the host name corresponding to IP address 1.2.3.4 looks for PTR RRs for domain name "4.3.2.1.IN-ADDR.ARPA" [39].

General lookup function

This function retrieves arbitrary information from the DNS, and has no counterpart in previous systems. The caller supplies a QNAME, QTYPE, and QCLASS, and wants all of the matching RRs. This function will often use the DNS format for all RR data instead of the local host's, and returns all RR content (e.g., TTL) instead of a processed form with local quoting conventions.

When the resolver performs the indicated function, it usually has one of the following results to pass back to the client:

- One or more RRs giving the requested data.

In this case the resolver returns the answer in the appropriate format.

- A name error (NE).

This happens when the referenced name does not exist. For example, a user may have mistyped a host name.

- A data not found error.

This happens when the referenced name exists, but data of the appropriate type does not. For example, a host address function applied to a mailbox name would return this error since the name exists, but no address RR is present.

It is important to note that the functions for translating between host names and addresses may combine the "name error" and "data not found" error conditions into a single type of error return, but the general function should not. One reason for this is that applications may ask first for one type of information about a name followed by a second request to the same name for some other type of information; if the two errors are combined, then useless queries may slow the application.

3.5.3 Aliases

While attempting to resolve a particular request, the resolver may find that the name in question is an alias. For example, the resolver might find that the name given for host name to address translation is an alias when it finds the CNAME RR. If possible, the alias condition should be signalled back from the resolver to the client.

In most cases a resolver simply restarts the query at the new name when it encounters a CNAME. However, when performing the general function, the resolver should not pursue aliases when the CNAME RR matches the query type. This allows queries which ask whether an alias is present. For example, if the query type is CNAME, the user is interested in the CNAME RR itself, and not the RRs at the name it points to.

Several special conditions can occur with aliases. Multiple levels of aliases should be avoided due to their lack of efficiency, but should not be signalled as an error. Alias loops and aliases which point to non-existent names should be caught and an error condition passed back to the client.

3.5.4 Resolver internals

Every resolver implementation uses slightly different algorithms, and typically spends much more logic dealing with errors of various sorts than typical occurrences. This section outlines a recommended basic strategy for resolver operation, but leaves details to [29].

3.5.5 Stub resolvers

One option for implementing a resolver is to move the resolution function out of the local machine and into a name server which supports recursive queries. This can provide an easy method of providing domain service in a PC, which lacks the

resources to perform the resolver function, or can centralize the cache for a whole local network or organization.

All that the remaining stub needs is a list of name server addresses that will perform the recursive requests. This type of resolver presumably needs the information in a configuration file, since it probably lacks the sophistication to locate it in the domain database. The user also needs to verify that the listed servers will perform the recursive service; a name server is free to refuse to perform recursive services for any or all clients. The user should consult the local system administrator to find name servers willing to perform the service.

This type of service suffers from some drawbacks. Since the recursive requests may take an arbitrary amount of time to perform, the stub may have difficulty optimizing retransmission intervals to deal with both lost UDP packets and dead servers; the name server can be easily overloaded by too zealous a stub if it interprets retransmissions as new requests. Use of TCP may be an answer, but TCP may well place burdens on the host's capabilities which are similar to those of a real resolver.

3.5.6 Resources

In addition to its own resources, the resolver may also have shared access to zones maintained by a local name server. This gives the resolver the advantage of more rapid access, but the resolver must be careful to never let cached information override zone data. In this research the term "local information" is meant to mean the union of the cache and such shared zones, with the understanding that authoritative data is always used in preference to cached data when both are present.

The following data structures represent the state of a request when all are functions converted to a general lookup function in the resolver:

- SNAME the domain name we are searching for.
- STYPE the QTYPE of the search request.
- SCLASS the QCLASS of the search request.
- SLIST a structure which describes the name servers and the zone which the resolver is currently trying to query. This structure keeps track of the resolver's current best guess about which name servers hold the desired information; it is updated when arriving-information changes the guess. This structure includes the equivalent of a zone name, the known name servers for the zone, the known addresses for the name servers, and history information which can be used to suggest which server is likely to be the best one to try next. The zone name equivalent is a match count of the number of labels from the root down which SNAME has in common with the zone being queried; this is used as a measure of how "close" the resolver is to SNAME.

SBELT a "safety belt" structure of the same form as SLIST, which is initialized from a configuration file, and lists servers which should be used when the resolver doesn't have any local information to guide name server selection. The match count will be -1 to indicate that no labels are known to match.

CACHE A structure which stores the results from previous responses. Since resolvers are responsible for discarding old RRs whose TTL has expired, most implementations convert the interval specified in arriving RRs to some sort of absolute time when the RR is stored in the cache. Instead of counting the TTLs down individually, the resolver just ignores or discards old RRs when it runs across them in the course of a search, or discards them during periodic sweeps to reclaim the memory consumed by old RRs.

Chapter 4

Novell Directory Services

4.1 Introduction

Novell Directory Services (NDS) is a global, distributed, and replicated network database; it also provides an easily managed and more secure network environment. NDS is based on the CCITT X.500 standard. NDS maintains information about all network resources (users, groups, servers, file-systems, printers, and so on) in a hierarchical tree structure. Network resources can be organized in the tree independent of their physical location. Thus network users can access any network resource they have rights to, without having to know the exact location of that resource. The NDS Directory is different from the file system directory and its structure. With NDS, it is now possible to integrate a diverse network of resources into a single, easy-to-use environment [30].

Rather than supporting a single server, NDS supports an entire network of servers. Distributing the network database allows all servers to easily access all network information. It also allows the database to be replicated, thus minimizing the risk of a single point of failure.

Traditional DNS/DHCP solutions have some drawbacks, which could be solved using NDS enabled DNS/DHCP services [31]. Following are the few drawbacks of traditional DNS/DHCP solutions:

- Tracking IP address usage is difficult
- Configuration of TCP/IP Nodes is laborious
- Name and Address Services are not Integrated
- Fault-Tolerance system is not transparent

To solve the above problems of traditional DNS/DHCP services and to get the NDS enabled DNS/DHCP services for the Country Network, this chapter introduces the basic concepts behind NDS, discussing NDS Schema, Objects and properties and telling how Root, Container, and Leaf objects form the Directory tree. It also explains about NDS partitions and replicas, and time synchronization [30].

4.2 How Objects Form the Directory Tree

NDS operates in a logical organization called the Directory tree. This is so named because objects are stored in a hierarchical tree structure with the [Root] object at the top of the tree and branching downward [53].

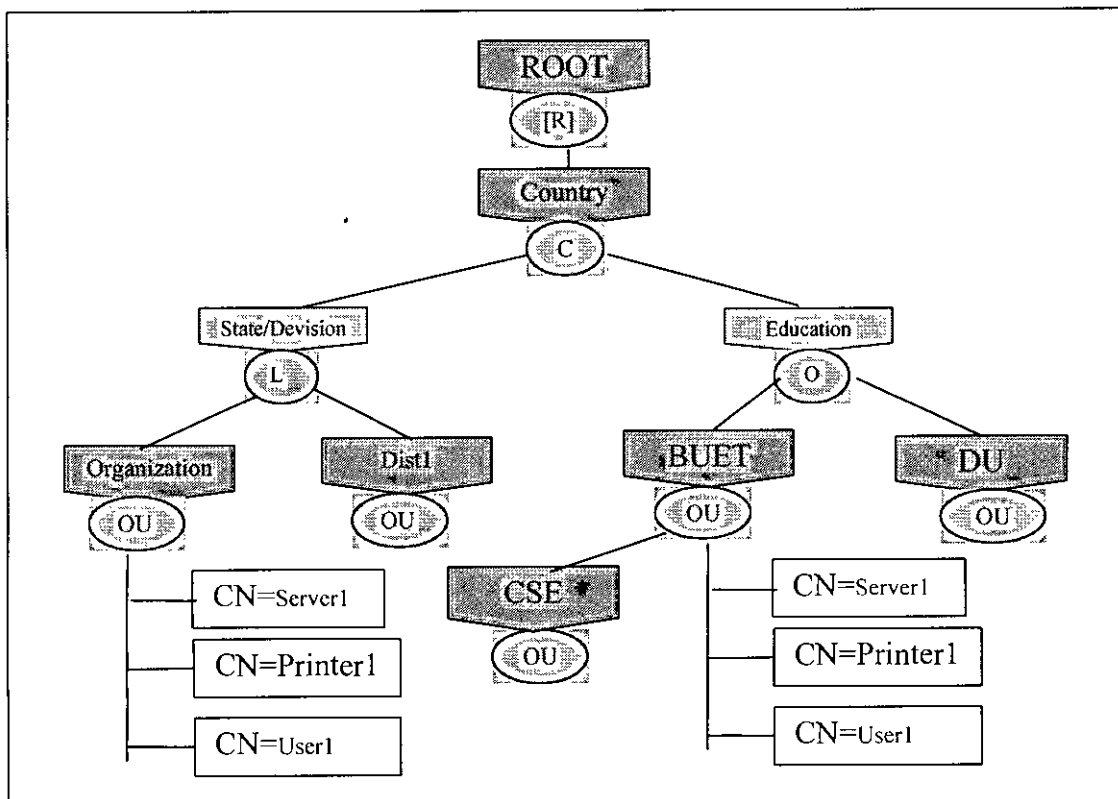
The Directory tree is made up of three types of objects:

- The [Root] object
- Container objects
- Leaf objects

[Root] Object

The [Root] object is placed at the top of the NDS tree. Branches of the Directory tree consist of container objects and all of the objects they hold. These container objects can also contain other container objects. Leaf objects are at the ends of the branches and do not contain any other objects. Figure 4.1: Illustrates how objects can be laid out to form the Directory tree.

Figure 4.1: Directory Tree



Country (C=) object.

The Country object provides another layer of identification for Country based NDS Tree and is needed for participation in a global NDS network [32]. The Country

object is directly below the [Root] object in the NDS tree, and is needed as entry point for a Country to merge with the Global NDS Tree.

The [Root] object of a Directory tree should not be confused with the root directory in the file system. In the file system, the root directory is the first directory and it bears no relation to the [Root] object of a Directory tree.

4.3 Network-Wide Login

With NDS, users no longer need to login or attach to specific servers. Instead, they can login to the network. Once logged in to the network, users can access any service or resource they have rights to, without having to explicitly login or attach to other servers. The users will be transparently attached to the server on which the specified service resides. NDS handles all of the address resolution issues in the background, so users are shielded from the complexity of having to understand the network topology, protocols, media, and communication links [33].

Because the NDS database is replicated, multiple copies of users' required login information are spread throughout the network. This replication allows users to login to the network whether or not their home server is on-line. As long as the servers which provide the necessary data or services are operational, the user can access them. In this sense, when a user is logged in to the network, servers become transparent to the process of actually using the network.

4.4 Analysis of NDS Schema and Objects

The NDS schema consists of the set of rules governing the structure of the Directory tree. It defines the objects that can exist in the tree, including how entries may be constructed, which attribute values are permitted, how Distinguished Names may be built, and other characteristics of use to the Directory itself. These object and attribute rules are specified through a data dictionary that provides a standard set of data types from which objects can be created [34], [35].

Every object in the Directory belongs to an object class that specifies the attributes that can be associated with the object. All attributes are based on a set of standard attribute types, which in turn are based on standard attribute syntaxes.

The Directory schema controls the structure of individual objects and the relationship among objects in the tree. In controlling this relationship the schema specifies subordination among object classes.

The schema is extensible & dynamic, and the administrators can define new objects classes and attributes of objects in addition to those provided by the base schema. Extending the schema is accomplished through modifying or creating new object definitions (attribute or object class) and then adding these new definitions to the base schema.

Because the schema has been moved into the regular hierarchical object space in the Directory Information Tree (DIT), schema objects generally are named and behave as normal NDS objects in instantiation, synchronization, and modification. Additionally, the normal NDS functions in the DSAPIs (Directory Services Application Program Interface) can be used to manipulate schema definitions (create, browse, modify, etc.).

4.4.1 Schema Components

The NDS schema consists of three basic components:

- Object Classes
- Attribute Types
- Attribute Syntaxes

The set of rules that controls the creation of a particular object is called an object class. Each object class is defined in terms of attributes. An attribute is a specific piece of information that can exist for an object.

For example, NDS contains an object class for users, called User object. This User object class defines many attributes (over 80), including attributes for such items as the user's name, telephone number, address, and group memberships.

Attributes are defined in terms of a base set of data types called attribute syntaxes, and the attribute syntaxes define the primary data types for values stored in the NDS database.

For example, some attributes such as Password Minimum Length or Minimum Account Balance take integer values while other attributes such as a user's Full Name or Given Name take string values.

NDS has a set of built-in classes and attribute types that accommodate general categories of network objects such as organizations, users, and devices. This set is called the base schema. NDS developers can build on the base schema to create new classes and new attributes for objects. However, these new classes and attributes must be defined in terms of the existing syntaxes. Defining new syntaxes is not allowed.

4.4.2 Schema Structure

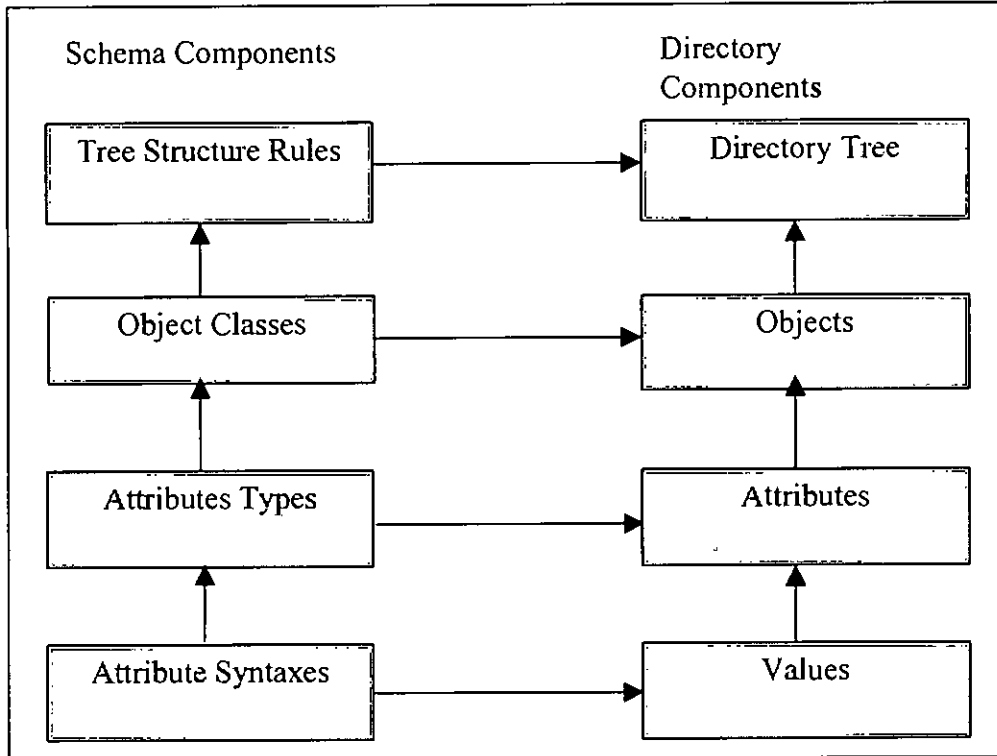
The schema defines the set of rules that govern the types of objects that can exist in an NDS tree. Each object belongs to an object class that specifies what attributes can be associated with the object. All attributes are based on a set of attribute types that are, in turn, based on a standard set of attribute syntaxes.

The NDS schema not only controls the structure of individual objects, but it also controls the relationship among objects in the NDS tree. The schema rules allow some objects to contain other subordinate objects. Thus the schema gives structure to the NDS tree.

Figure 4.2 shows how the schema components and the Directory components are interrelated. The vertical arrows indicate the structure dependencies from the basic building blocks up to the schema and the Directory, respectively. The horizontal arrows denote the schema rules that apply to the respective Directory components.

The attribute syntaxes define the primary data types for values stored in the NDS tree. Attribute types are defined from the attribute syntaxes and define the possible attributes an object can have. Object classes are defined using a subset of the possible attributes and determine the types of objects that can be in the tree. The tree structure rules define how the object classes can be organized and nested in the tree, and therefore determine the tree's structure [34].

Figure 4.2: Relationships between the Schema and the Directory Components.



Objects that can contain other objects are called *container objects*. Container objects are the building blocks of the hierarchical structure of the Directory tree. Objects that cannot contain other objects are known as non-container or *leaf objects*. Leaf objects comprise the actual network resources such as a user, a server, or a printer.

4.4.3 Object Class Definitions

Object classes define the types of objects that can exist in the NDS database. Database entries are created by selecting an object class and then supplying the required attribute information for the entry. For example, to create an entry for a user, you must select the User object class and then supply a name for the user.

In the base schema, all object classes are nonremovable; that is, they cannot be deleted or in any other way removed from the schema. Object classes that extended the schema are removable [36].

An object class is defined by its characteristics, which consists of the following kinds

of information:

- Structure rules for naming and containment
- Super classes
- Object class attributes
- ACL templates
- Object class flags

An object class does not have to specify definitions for all characteristics because it can inherit characteristics from super classes. See “Object Class Inheritance Rules” for more information.

4.4.4 Object Class Structure Rules

All object classes possess two types of structure rules:

- Naming attributes which determine how objects of the class are named
- Containment classes which determine where in the NDS tree hierarchy objects can be placed

The structure rules for an object class define the possible structural relationships of objects in the NDS tree. The structure rules are either explicitly defined by the class or inherited from a super class. If the class defines them, the class definitions take precedence over inherited definitions.

4.4.5 Object Class Naming Structure Attributes

Objects are identified by their own name and the name of their parent objects. An object’s name is called its partial name or relative distinguished name (RDN). An object’s RDN is determined by its naming attribute.

The object’s full name (with all its parent names included) is called the complete name or distinguished name (DN). An object’s DN is determined by all the objects it is subordinate to. Hence, containment rules, which control subordination in the tree, effectively control the formation of distinguished names.

The sections below describe the following characteristics of naming attributes:

- Naming attribute rules
- Multi-valued naming attributes
- Shareable naming attributes
- Inheritance of naming attributes
-

4.4.5.1 Naming Attribute Rules

Each class has one or more attributes designated as naming attributes. These attributes can be either mandatory or optional attributes, but at least one must be given a value

when creating an object of that class. If the only naming attribute is declared as optional, it is, in effect, mandatory.

Naming attributes specify the rules for the partial name of the object. For example, Organization objects are named by the O (Organization Name) attribute. This attribute is the only attribute value that can appear in an organizational entry's partial name.

4.4.5.2 Multi-valued Naming Attributes

Naming attributes can be multi-valued; in other words, more than one name (value) can be added to the naming attribute. For example, an organization can have both "Testing" and "Engineering" as values for the O (Organization Name) attribute. However, only one value will be flagged as the naming value, and that value is used in search operations.

Some object class definitions specify multiple naming attributes. For example, the *Locality* object class is named by the L (Locality Name) and S (State or Province Name) attributes. Thus, an RDN for locality can include just an L (Locality Name) attribute, just an S (State or Province Name) attribute, or both attributes.

For example, the name for the Provo, Utah locality could be

L=Provo
S=Utah
L=Provo + S=Utah

The last example uses both attributes with a plus sign (+) to indicate where the second attribute's value begins. When the type specifiers (in this case L and S) are used as shown, the name is referred to as a typed name. A typeless name has the following format: "Provo+Utah".

4.4.5.3 Shareable Naming Attributes

A naming attribute does not necessarily reflect the class an object belongs to. Many classes, such as Computer, User, and Server, are named by their CN (Common Name) naming attribute. In fact, CN is the recommended naming attribute for leaf objects. In such cases, the naming attribute itself does not indicate which class the object belongs to, but the value of the naming attribute may suggest the nature of the object.

However, some naming attributes are closely tied to specific classes. For example, the C (Country Name) naming attribute is used to name only Country objects in the base schema.

4.4.5.4 Inheritance of Naming Attributes

Naming attributes for effective classes must follow the inheritance rules. Effective classes can inherit naming attributes only if the naming attributes of the super classes

are identical and do not conflict. If they are different and therefore ambiguous, the effective class must define its own naming attributes. Non-effective classes may have ambiguous naming attributes, but often define the naming attributes so subordinate objects can inherit them. For example, the Server class defines naming attributes that are inherited by the AFP Server, NCP Server, CommExec, Messaging Server, and Print Server classes.

4.4.6 Containment Classes

Objects that can contain other objects are called container objects or parent objects. Container objects are the branches of the NDS tree and provide a structure that is similar to a directory in a file system. Objects that cannot contain other objects are called non-container or leaf objects. Leaf objects represent the actual network resources that perform some function in the NDS tree, such as users, printers, modems, servers, or volumes.

The sections below describe the following characteristics of containment classes:

- Containment class rules
- Containment classes in the base schema
- Containment of leaf objects
- Containment classes and inheritance

4.4.6.1 Containment Class Rules

For each object class, a list of containment classes specifies where an object of that class may appear in the hierarchical structure of the NDS tree. An object can be immediately subordinate to only those objects whose classes appear in the containment list of the object's expanded class definition. An expanded class definition includes all the characteristics defined for the class plus all the characteristics that the class can inherit from super classes.

Effective classes can inherit containment classes from super classes only if the inheritance does not make containment ambiguous. If the inherited containment is ambiguous, the class must define containment. Class-defined containment overrides containment defined for super classes.

Effective classes are those object classes that can be used to create entries in the NDS database. Non-effective classes cannot be used to create entries and are used by the schema so that multiple object classes can inherit a common set of schema characteristics. Non-effective classes can have ambiguous containment.

Containment classes limit the possible location of an object in the Directory tree, thus restricting the order and types of partial names that appear in the object's complete name. Containment helps to ensure that the NDS tree expands in a consistent and logical fashion. For example, an Organization object can be the topmost object of the NDS tree or subordinate to the Tree Root object. A User object can be subordinate to an Organization object but not to a Tree Root object. Before users can be added to an NDS tree, the tree must contain either an Organization object or an Organizational

Unit object, which are the containment classes for the User object.

While helping to control the structure of the Directory, containment classes must also be flexible enough to accommodate a variety of organizational situations. An example is the relationship between the Organization and Locality classes. Each class specifies the other as a containment class. This allows an administrator to decide which hierarchical order best represents the company's organization.

4.4.6.2 Containment Classes in the Base Schema

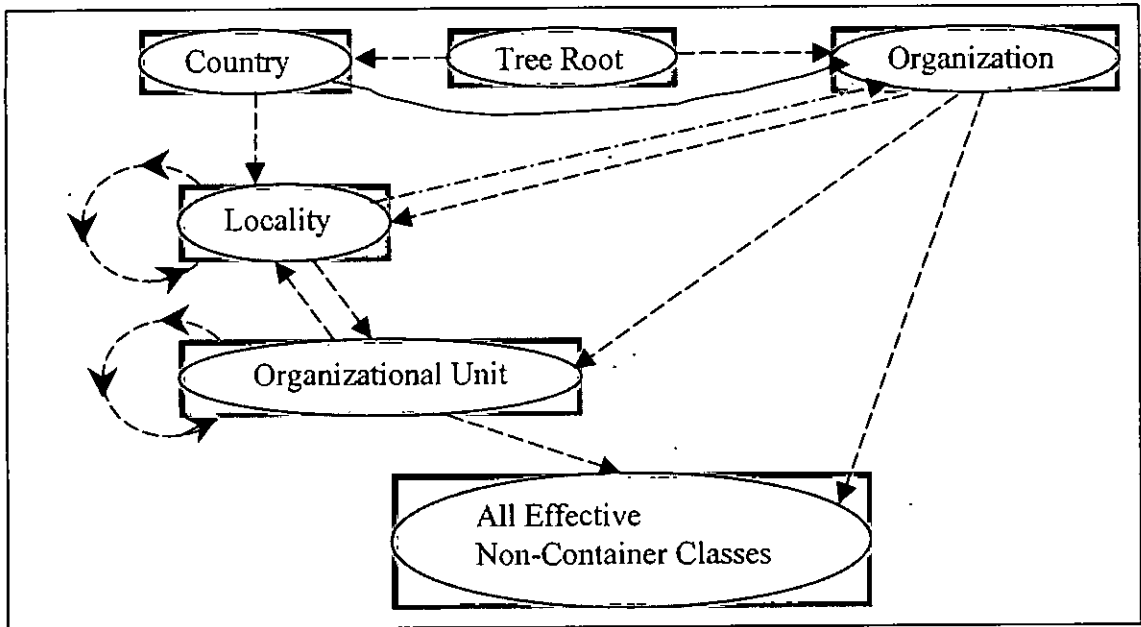
The table below lists the classes in the base schema that can contain other objects and the object types that they can contain.

Table 4.1: List of Object Class and Contained Class

Object Class	Contained Classes
Tree Root	Country Organization
Country	Locality Organization
Locality	Locality Organization Organizational Unit
Organization	Locality Organizational Unit Leaf Objects
Organizational Unit	Locality Organizational Unit Leaf Objects

Figure 4.3 presents a graphical view of the NDS containment structure. This view shows the containment classes and the object classes that they can contain and that can be contained by them. Object classes that cannot contain other objects (the leaf objects) are collectively shown as non-container classes. The object class Top is not shown in this graphical view because Top is used for schema hierarchy and inheritance but not for the NDS tree hierarchy.

Figure 4.3: Containment Structure.



Tree Root, Organization, and Country are shown on the same level because they all can be the topmost objects in the tree. Tree Root has arrows pointing to both Country and Organization because Country and Organization can be, but are not required to be, subordinate to Tree Root.

The base schema defines all effective leaf objects as subordinate to either Organizational Unit or Organization. Applications which extend the schema can define leaf objects that are subordinate to any container in the tree. They can also define new container objects. Such applications cannot, however, define new container objects for the root of the tree.

4.4.6.3 Containment of Leaf Objects

The following table lists the leaf objects (effective and non-effective), the object classes they can be their parent container, and the object that defined the containment.

Table 4.2: List of Leaf Object Class.

Object Class	Contained By	Class Defined For
	AFP Server	Organization Organizational Unit
Alias	Special case-inherits containment from the referenced object	Alias
Bindery Object	Organization Organizational Unit	Bindery Object

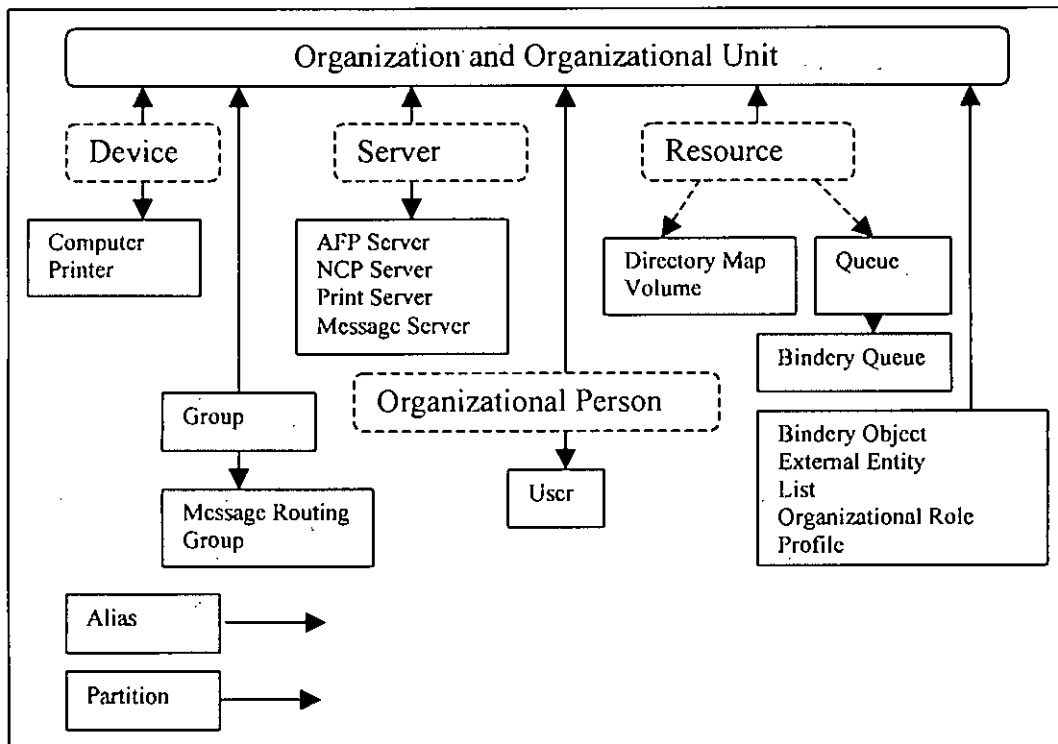
Bindery Queue	Organization Organizational Unit	Queue
Comm Exec	Organization Organizational Unit	Server
Computer	Organization Organizational Unit	Device
Device	Organization Organizational Unit	Device
Directory Map	Organization Organizational Unit	Resource
External Entity	Organization Organizational Unit	External Entity
Group	Organization Organizational Unit	Group
List	Organization Organizational Unit	List
Message Routing Group	Organization Organizational Unit	Group
Messaging Server	Organization Organizational Unit	Server
NCP Server	Organization Organizational Unit	Server
Organizational Person	Organization Organizational Unit	Organizational Person
Organizational Role	Organization Organizational Unit	Organizational Role
Partition	Special case	Partition
Person	None	Top
Print Server	Organization Organizational Unit	Server
Printer	Organization Organizational Unit	Device

Profile	Organization Organizational Unit	Profile
Queue	Organization Organizational Unit	Resource
Resource	Organization Organizational Unit	Resource
Server	Organization Organizational Unit	Server
Unknown	Special case	Any
User	Organization Organizational Unit	Organizational Person

Non-effective classes cannot be used to create object in the NDS tree, but they are often used to define containment classes for other object classes to inherit.

Figure 4.4 presents a graphical view of how the leaf objects inherited their containment classes. The arrows pointing up to container objects indicate which object class declared the containment classes. Arrows pointing down to a leaf object indicate the objects that inherit the containment classes. Effective classes use shapes with solid lines and non-effective classes use shapes with dotted lines.

Figure 4.4: Container Inheritance for Leaf Objects.



A couple of effective object classes are unique: Alias and Partition. They are shown at the bottom of Figure 4. Alias inherits its containment classes from the object that it references. Since all leaf objects have Organizations and Organizational Units as their containment classes, an Alias will usually inherit these containment classes. However, an Alias can reference a container and when it does, the Alias inherits the container's containment classes.

Partition inherits its containment class from its root container object. Since a partition can be defined for any container object in the tree, a partition can have the containment rules of Tree Root, Country, Organization, Locality, and Organizational Unit in the base schema.

The Person class is not shown because it is a non-effective class, defines no containment classes, and inherits no containment class from its super class, Top. Thus, Person is like Top in that they both do not affect containment classes of any objects in the NDS tree [36].

4.4.6.4 Containment Classes and Inheritance

Containment classes create the hierarchy of the NDS tree and determine where an instance of an object can be created in the NDS tree. A special flag, [Nothing], allows the three objects, Tree Root, Country, and Organization, to have no superior object.

Once an instance of an object (or an entry) is created in the NDS tree, the entry inherits rights from its container objects and the container objects are part of the entry's distinguished name. However, the object classes in the schema do not inherit anything from their containment classes.

Object classes can inherit containment definitions, but such inheritances come from the schema's super class structure.

4.4.7 Super Classes

Super classes create the hierarchy of the schema and determine the characteristics that an object class can inherit from another object class. Inheritance simplifies the rules of the schema because it allows some characteristics to be defined once, while multiple objects classes can use and enforce these common characteristics.

The sections below describe the following characteristics of super classes:

- Root schema object
- Super class rules
- Class hierarchy
- Object class inheritance rules

4.4.7.1 Root Schema Object

The Top object class is the root of the schema. Since all other object classes inherit

characteristics from the Top class, the Top class specifies information that pertains to all other classes. For example, the Top class defines the following optional attributes:

- ACL
- Back Link
- Last Referenced Time
- Obituary
- Used By

NDS uses these attributes to maintain information. Since these attributes are defined for Top, all object classes inherit these attributes. Entries in the NDS tree have them available whenever NDS needs to assign a value to one.

4.4.7.2 Super Class Rules

Each object class must define an object class as its super class. Super classes cannot be recursive; therefore an object class cannot list itself as a super class. The complete definition of each object class is derived from the characteristics of the object class itself plus the characteristics of all classes in its super class lineage. Hierarchies of classes develop through class inheritance in this manner. The classes at the top of the hierarchy provide general characteristics, while those at the bottom become more and more specialized. The complete set of rules for an object class is called the expanded class definition.

The object class from which an entry is created is called the entry's base class. The expanded class definition for an object class includes the base class and the sum of the information specified by all its super classes. For the purpose of searching the NDS tree, an entry is considered a member of all of its super classes. For example, the base class for creating a user is the User class. The User class inherits from the following super classes: Organizational Person, Person, and Top [43].

Although the schema is stored with the rest of the NDS database, schema data is logically separated from the NDS tree and must be accessed through different functions. Also, the schema's class hierarchy does not necessarily form a simple tree graph because a class can list more than one class as a super class. Listing multiple classes as super classes is called multiple inheritance (None of the objects in the NDS base schema uses multiple inheritance).

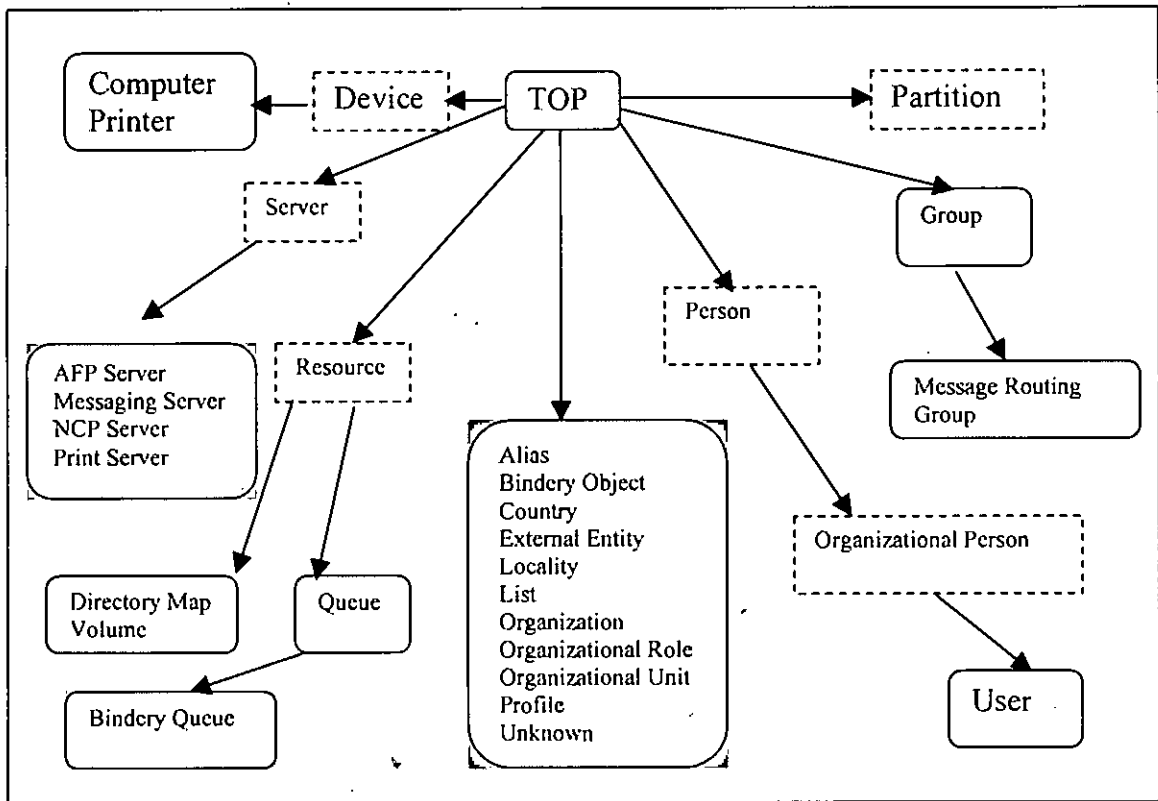
4.4.7.3 Class Hierarchy

Figure 4.5 provides a single graphic view of the base schema, showing the object classes in the structure of the class hierarchy. This provides a visual view of the object classes, super classes, and inheritance. In this view, the arrows show the direction of flow for inheritance. An object class inherits the rules and attributes defined by all its super classes, but does not inherit from its subordinates.

Effective object classes are represented as shapes with solid lines, and non-effective

object classes are represented as shapes with broken lines.

Figure 4.5: Super Class Inheritance.



The class *Top* is an effective class, but it is a special super class because it cannot be used to define an instance of an object.

Figure 4.5 illustrates the effective use of non-effective classes. For example, the *Server* class (non-effective) defines those characteristics shared by all servers; the effective classes (AFP server, NCP Server, Print Server, etc.) define only those characteristics that are particular to that type of server.

4.4.7.4 Object Class Inheritance Rules

While a class automatically inherits some characteristics in the schema, a class can select to inherit or block the inheritance of other characteristics. The schema follows the following inheritance rules.

- A class must declare another class as its super class. The class then automatically inherits any super classes of its defined super class. (*Top* is the only class that has no super class.)
- A class may, but is not required to, define mandatory or optional attributes. The class, however, always inherits all the attributes, both mandatory and optional, of its super classes.
- A class may, but is not required to, define a default ACL template. The class

always inherits all the default ACL templates of its super classes. Classes that extend the schema cannot define new default ACL templates.

- A class can inherit containment classes and naming attributes, but if the class defines them, any definitions made in super classes are not applied to the class.

4.4.8 Object Class Attributes

An attribute is a single piece of information that is stored in the database about an object. The attributes assigned to an object class can be mandatory or optional:

- If an attribute is mandatory, a value must be assigned to the attribute before an instance of the object can be created.
- If an attribute is optional, a value does not need to be assigned to create an instance of the object. The only exception is an optional naming attribute. If the optional naming attribute is the only attribute used for naming the object, this optional attribute becomes a mandatory attribute.

Both mandatory and optional attributes are always inherited from super classes. There is no way to block the inheritance. Also, mandatory definitions take precedence over optional designations. For example, a subordinate class can define an attribute as mandatory that is optional in a super class. For that class it is now mandatory. However, if a subordinate class tries to define an attribute as optional that a super class defines as mandatory, the attribute is still mandatory for the subordinate class.

A client cannot associate an attribute with an object unless the attribute is listed among the mandatory or optional attributes of the object's expanded class definition. If a client must associate an attribute with a particular object and the attribute is not specified by the object class, the client must extend the schema by:

- Adding the new attribute to the class or a super class as an attribute
- Defining a new class that inherits from the original class and adds the new attribute as an attribute

If the attribute is added to a non-removable class, the attribute becomes non-removable. Attributes are only removable when they are not assigned to any class.

4.4.9 Object Class Flags

There are five object class flags that can be "set" (turned On) or "not set" (turned Off). Applications extending the schema can set two: Container and Effective. NDS sets three: Non-removable, Ambiguous Naming, and Ambiguous Container.

4.4.9.1 Container Flag

The Container flag indicates whether the object can contain other objects. The flag is turned On for those object classes that are designated as container classes. The flag is turned Off for all leaf object classes.

4.4.9.2 Effective Flag

The Effective flag indicates whether an object class is effective or non-effective. The Effective flag is turned On for those classes which can be used both to provide definition and to create objects. The Effective flag is turned Off for those classes which provide definition but cannot be used to create objects.

Only effective classes are:

- Used to create entries in the NDS database
- Assigned as base classes to the entries they create in the NDS database

Most of the object classes in the base schema are effective classes. Since effective classes are the active building blocks from which an NDS tree is created, their structure rules must be complete. This means that the naming attributes and containment classes cannot be ambiguous.

For example, if naming attributes or containment classes are not specified for a new effective class, they are inherited from the new class's super classes. If the new effective class inherits from multiple super classes, the naming attribute and containment classes must be identical. If they aren't identical, the structure rules conflict and are ambiguous. In this case, an effective class must define its naming attributes and containment classes.

If the structure rules are incomplete or ambiguous, NDS automatically flags the class as non-effective. The effective or non-effective flag is assigned to a class when it is originally defined. The value cannot be modified after the class is created.

The non-effective classes are not active and thus cannot be used to create objects in an NDS tree. They are typically used as super classes to define class information that is shared by multiple effective classes. The effective classes can then inherit the class information from the non-effective super class rather than repetitively defining it.

The base schema defines the following non-effective classes:

- Device
- Organizational Person
- Partition
- Person
- Resource
- Server

Top is the one special case for the Effective flag. Although Top is flagged as an effective class, no object can be created from the Top class.

4.4.9.3 Non-removable Flag

The Non-removable flag indicates whether the object class can be removed from the schema. The flag is turned On for objects that cannot be removed. The flag is turned Off for object classes that can be removed. All base schema object classes are flagged non-removable. Object classes added to extend the schema are the only classes that may have the non-removable flag turned off.

4.4.9.4 Ambiguous Naming Flag

The Ambiguous Naming flag indicates whether the object class has clearly defined naming attributes. As a general rule, non-effective classes can be created with ambiguous naming, but effective classes must have non-ambiguous naming attributes. Only in special cases can effective classes be created with ambiguous naming. The Alias class object is one of these special cases since it needs to inherit the naming attributes of its reference object class.

For most object classes in the base schema, the Ambiguous Naming flag is turned Off. The only object classes where this flag is turned On are Top, Alias, Person, and Partition.

4.4.9.5 Ambiguous Container Flag

The Ambiguous Container flag indicates whether the object class has clearly defined containment classes. As a general rule, non-effective classes can be created with ambiguous containment, but effective classes must have non-ambiguous containment. Only in special cases can effective classes be created with ambiguous containment. The Alias class object is one of these special cases since it needs to inherit the containment classes of its reference object class.

For most object classes in the base schema, the Ambiguous Container flag is turned Off. It is turned On for object classes Top, Alias, Person, and Partition.

4.4.10 Default ACL Templates

Every object in the NDS tree has an Access Control List (ACL) attribute. This attribute holds information about which trustees have access to the object itself (entry rights), and which trustees have access to the attributes for the object. This information is stored in sets of information containing:

- The trustee name
- The affected attribute-[Entry Rights], [All Attributes Rights], or a specific attribute
- The privileges

Default ACL templates are defined for specific classes in the base schema and provide a minimum amount of access security for newly created objects. Only base schema objects can have default ACL templates. Developers extending the schema cannot create default ACL templates for new objects.

Since the Top object class defines a default ACL template, all object classes inherit a default ACL template. The ACL defined for Top allows the object that creates another object the right to supervise the created object. This ACL ensures that every object added to an NDS tree has a supervisor.

An object inherits the default ACL templates that are defined for any of the object's super classes. For example, the NCP Server object inherits default ACL templates from Top and Server, and then defines one for itself.

Developers extending the schema cannot create templates that overwrite or add to the

templates in the base schema. However, when an object is created in an NDS tree, the creation process can set the object's ACLs to any value, including one that changes a value that comes from a default ACL template.

4.4.11 Construction Rules for Object Classes

The following rules regulate the construction of new object classes. Developers that need to define the new object classes should pay close attention to these rules.

- Object class definitions cannot be recursive. That is to say, an object cannot have itself as a super class.
- Only classes with complete structure rules can be flagged as effective, and thus used to create objects. This means the super classes, containment, and naming attributes must be complete.
- An effective class can be constructed in three ways:
 - The class defines its own structure rules
 - The class inherits structure rules from its super classes
 - The class defines part of the structure rules (such as naming) and inherits the other part of the structure rules (such as containment) from a super class
- Structure rules that might be inherited from its super classes are ignored for a class that defines its own structure rules.
- If structure rules of an effective class are inherited, they must be non-ambiguous.

4.4.12 Attribute Type Definitions

All attributes found in an NDS tree consist of an attribute type and an attribute value, which can be multi-valued. The attribute type identifies the nature of information the attribute stores, and the value is the stored information.

The attribute type definition:

- Identifies the attribute syntax used for the value
- Specifies the constraints that are imposed on the syntax

These constraints are also known as attribute flags. Attributes are assigned to objects according to the object's class definition.

An example of an attribute type is *CN (Common Name)* which uses the "Case Ignore String" syntax. *CN (Common Name)* constrains this syntax to a range from 1 to 64 Unicode characters. This attribute is used by many object classes, including Server, Person, Group, and Bindery Object.

Attribute types can be added to the NDS schema. However, once an attribute type has

been created, it can not be modified.

Attribute types can be removed from the NDS schema, but only if the attribute is not part of the base schema and only if the attribute type isn't assigned to a class. All attribute types in the base schema are always flagged non-removable.

4.4.13 Attribute Syntaxes

The attribute syntax controls the type of information that can be stored in the value (for example, integer, string, or stream data). The syntax must be selected from the set of predefined attribute syntaxes. The syntax also controls the type of compare operations that can be performed on the value. See "4.4.15 Attribute Syntax Definitions" for more information.

4.4.14 Attribute Constraints

The attribute constraints restrict the information that can be stored in the data type and constrain the operations of NDS and NDS clients. The constraints specify whether the attribute:

- Allows only a single value or multiple values
- Has a range or size limit to the value
- Is synchronized immediately, at the next scheduled interval, or never
- Is hidden or viewable
- Is write-able or read-only

The table below lists all of the attribute constraints.

Table 4.3: List of Attribute Constraints.

Constraint	Description
DS_SINGLE_VALUED_ATTR	Indicates that the attribute has a single value, with no order implied.
DS_SIZED_ATTR	Indicates that the attribute has an upper and lower boundary. This can be the length for strings or the value for integers. The first number indicates the lower boundary and the second, the upper boundary.
DS_NONREMOVABLE_ATTR	Prevents the attribute from being removed from an object class definition. The client cannot set or modify this constraint flag and thus cannot modify the attribute. All base schema attribute type definitions have the non-removable flag set.
DS_READ_ONLY_ATTR	Prevents clients from modifying the attribute. The NDS

	server creates and maintains the attribute.
DS_HIDDEN_ATTR	Marks the attribute as usable only by the NDS server. The client cannot set or modify this flag and thus cannot see or modify the attribute.
DS_STRING_ATTR	Labels the attribute as a string type. You can use attributes of this type as naming attributes
DS_OPERATIONAL	Indicates that NDS uses the attribute internally and requires the attribute to function correctly.
DS_PUBLIC_READ	Indicates that anyone can read the attribute without read privileges being assigned. You cannot use inheritance masks to prevent an object from reading attributes with this constraint.
DS_PER_REPLICA	Marks the attribute so that the information in the attribute is not synchronized with other replicas. The client cannot set or modify this constraint flag and thus cannot modify the attribute
DS_SCHEDULE_SYNC_NEVER	Allows the attribute's value to change without such a change triggering synchronization. The attribute can wait to propagate the change until the next regularly scheduled synchronization cycle or some other event triggers synchronization.
DS_WRITE_MANAGED	Forces users to have managed rights on the object that contains this attribute before they can change the attribute's value.
DS_SERVER_READ	Indicates that Server class objects can read the attribute even though the privilege to read has not been inherited or explicitly granted. You cannot use inheritance masks to restrict servers from reading attributes with this constraint.
DS_SYNC_IMMEDIATE	Forces immediate synchronization with other replicas when the value of the attribute changes. Attributes without this constraint are synchronized at the next synchronization interval.

4.4.15 Attribute Syntax Definitions

An attribute syntax defines a standard data type which an attribute uses to store its values in the NDS tree. The syntax definitions are static definitions represented in basic C-code format. For example, the schema includes the following attribute syntaxes:

- SYN_CI_STRING-The Case Ignore String syntax is used by attributes whose values are strings and the case (upper or lower) is not significant.
- SYN_INTEGER-The Integer syntax is used by attributes whose values are signed integers.

An attribute syntax consists of a single data type for which syntax matching rules and qualifiers have been specified. Matching rules indicate the characteristics that are significant when comparing two values of the same syntax. There are three primary matching rules:

- Equality-To match for equality, two values must be identical, use the same attribute syntax, and conform to the data type of the attribute syntax. Most syntaxes specify a match for equality. NDS checks that the values being matched conform to the data type of the syntax. NDS will not attempt to match two values if the syntax does not specify a match for equality.
- Ordering-To match for ordering, a syntax must be open to comparisons of “less than,” “equal to,” and “greater than.” For example, 50 are less than 100, and N is greater than B.
- Substrings-To match substrings, a syntax must be open to search and comparison patterns that include the asterisk (*) wildcard. For example in a syntax using substring matching, “n*v*1” would match “naval,” “navel,” and “novel.”

An approximate comparison rule can be used in searches and comparisons on syntaxes with lists of strings and syntaxes with distinguished names:

- Strings -The approximate rule determines whether a string is present in a syntax with a string list.
- Distinguished Names-The approximate rule determines whether a distinguished name matches the distinguished name in a corresponding field while ignoring the other fields in the syntax. To increase performance, NDS replaces distinguished names with IDs in the comparison and search operations.

A syntax can specify one or more of these matching rules. For example, the Case Ignore String syntax specifies matching rules of equality and substrings.

A syntax can also specify qualifiers for comparison which ignore characters such as dashes, leading spaces, trailing spaces, and multiple consecutive internal spaces. All string syntaxes use comparison operations that ignore extra spaces. Other qualifiers allow only digits or only printable characters.

Attribute type definitions are built on attribute syntaxes. Developers extending the schema can create new attribute types using these syntaxes, but they cannot create any new syntax definitions.

4.5 NDS Partitioning

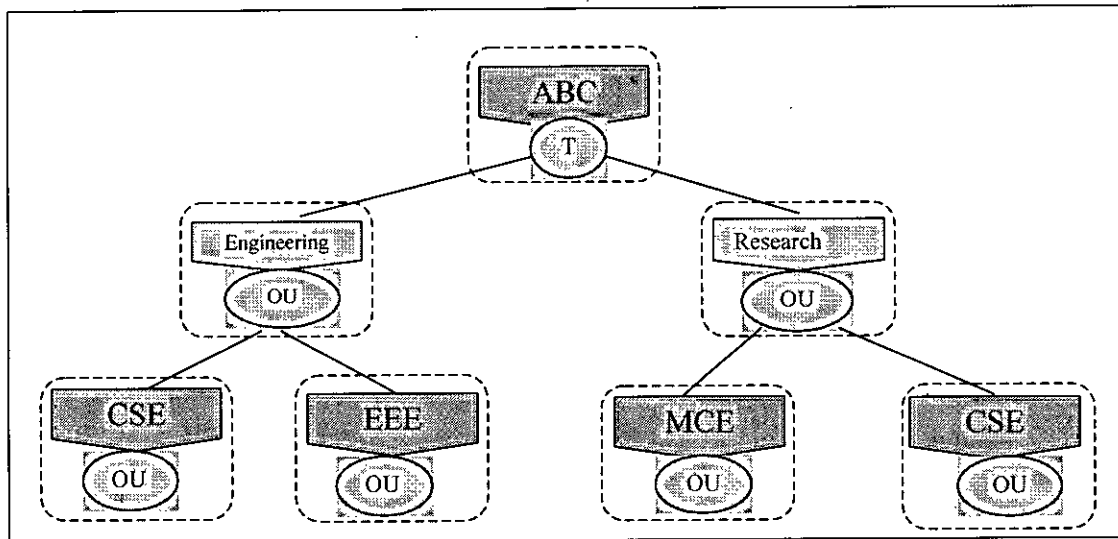
NDS divides the Directory tree into logical sub-trees called *partitions*. Although any part of the Directory can be considered a sub-tree, a partition forms a distinct unit of data for storing and replicating Directory information. Partition boundaries cannot overlap, so each entry in the Directory appears in only one partition.

A partition subordinate to another partition is called a *child* partition, while its immediate superior is called a *parent* partition. Partitions must obey the following rules:

- They must contain a connected sub-tree.
- They must contain only one container object as the root of the sub-tree.
- They cannot overlap with any other partition.
- They take their name from the root-most container object (the container at the root of the sub-tree), which is called the *partition root*.

Figure 4.6 shows a partitioned tree with Engineering.ABC as the parent and CSE.Engineering.ABC as a child partition. ABC is the Directory tree's root partition.

Figure 4.6: A partitioned tree.



4.5.1 Partition Operations

NDS allows administrators to create and manage partitions and their replicas. These operations, called *partition operations*, allow great flexibility in maintaining and modifying the Directory tree. Partition operations include the following:

- Adding a replica of a partition. This operation involves placing a replica of a given partition on a specific server.
- Changing a replica's type. This operation changes a replica's type,

including creating a new master replica. For example, an administrator may want to change a read-write replica to a read-only replica to restrict changes to that partition's data.

- Removing a replica from a set of replicas. This operation removes one or more replicas of a given partition.
- Splitting a partition. This operation creates a new partition from a container in an existing partition.
- Joining two partitions. This operation joins a parent and child partition, making one partition from the two.
- Moving a partition. This allows administrators to move an entire partition and its contents to another part of the Directory tree without affecting connectivity or access control privileges.

All these operations involve two major stages: the initial operation involving the client and the master replica, and a second stage during which the partition changes are sent to each replica of the partition.

4.6 NDS Replication

A single instance of a partition is called a *replica*. Partitions can have multiple replicas, but only one replica of a particular partition can exist on each server. (Servers can hold more than one replica, as long as each replica is of a different partition.) One of the replicas (usually the first created) of a given partition must be designated the master replica. Each partition can have only one master replica; the other replicas are designated as either read-write or read-only replicas. (You can use the read-only replica only to read the information in the partition replica. You cannot write to a read-only replica.)

Replication adds fault tolerance to the NDS because the database has more than one copy of its information.

4.6.1 Replica Types

Replicas must be designated as one of four types:

- Master
- Read-write
- Read-only
- Subordinate reference

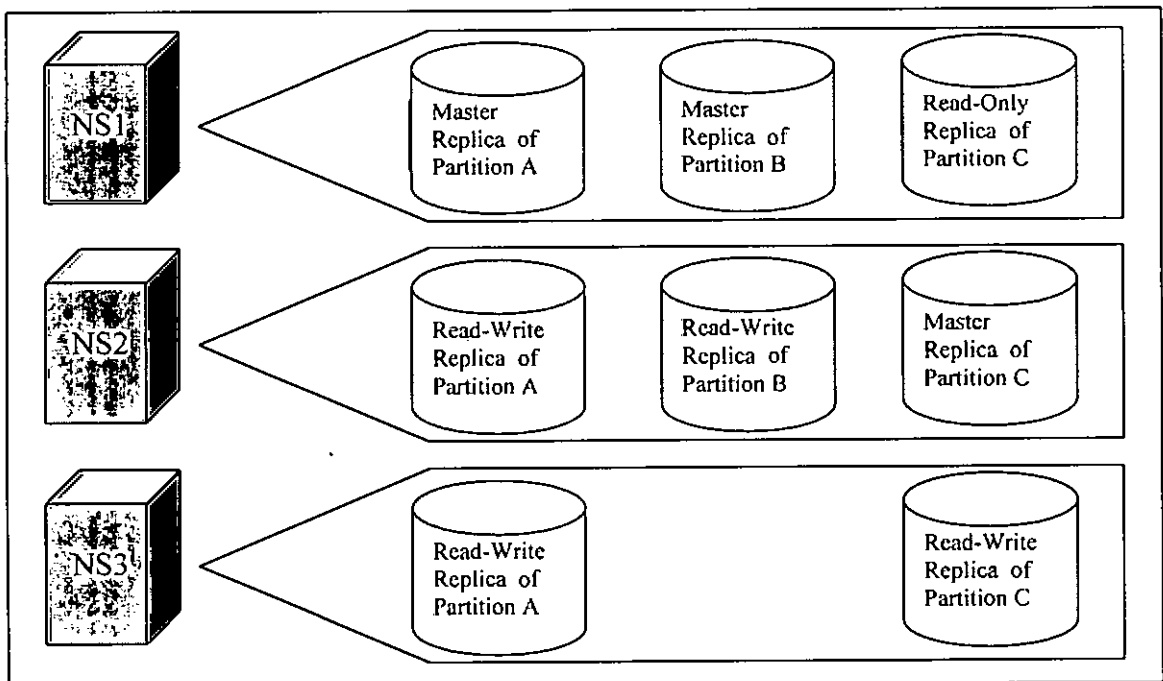
One (and only one) replica of a partition must be designated as the master replica; the other replicas must be designated as either a read-write or read-only replica, or a subordinate reference. The replicas are invisible to the end user; that is, the user does not know which replica contains the entries being accessed.

4.6.1.1 Master, Read-Write, and Read-Only Replicas

Clients can create, modify, and delete entries on either master or read-write replicas. However, clients can perform operations that deal with partitions only on the master replica. Clients cannot make any changes to read-only replicas.

Figure 4.7 shows three partitions (A, B, and C) replicated across three name servers (NS1, NS2, and NS3).

Figure 4.7: Partitioning and Replication.



- NS1 stores the master replicas of partitions A and B and a read-only replica of partition C.
- NS2 stores the master replica of partition C and read-write replicas of Partition A and B.
- NS3 stores Read-Write replicas of Partition A and C.

Given this arrangement, any of the servers could handle a request to add an entry to partition A. Only NS1 and NS2 could handle a similar request for partition B, and only NS2 and NS3 could handle such a request for partition C.

Only NS1 can create a new partition that is subordinate to partition A or B, and only NS2 can create a new partition that is subordinate to partition C.

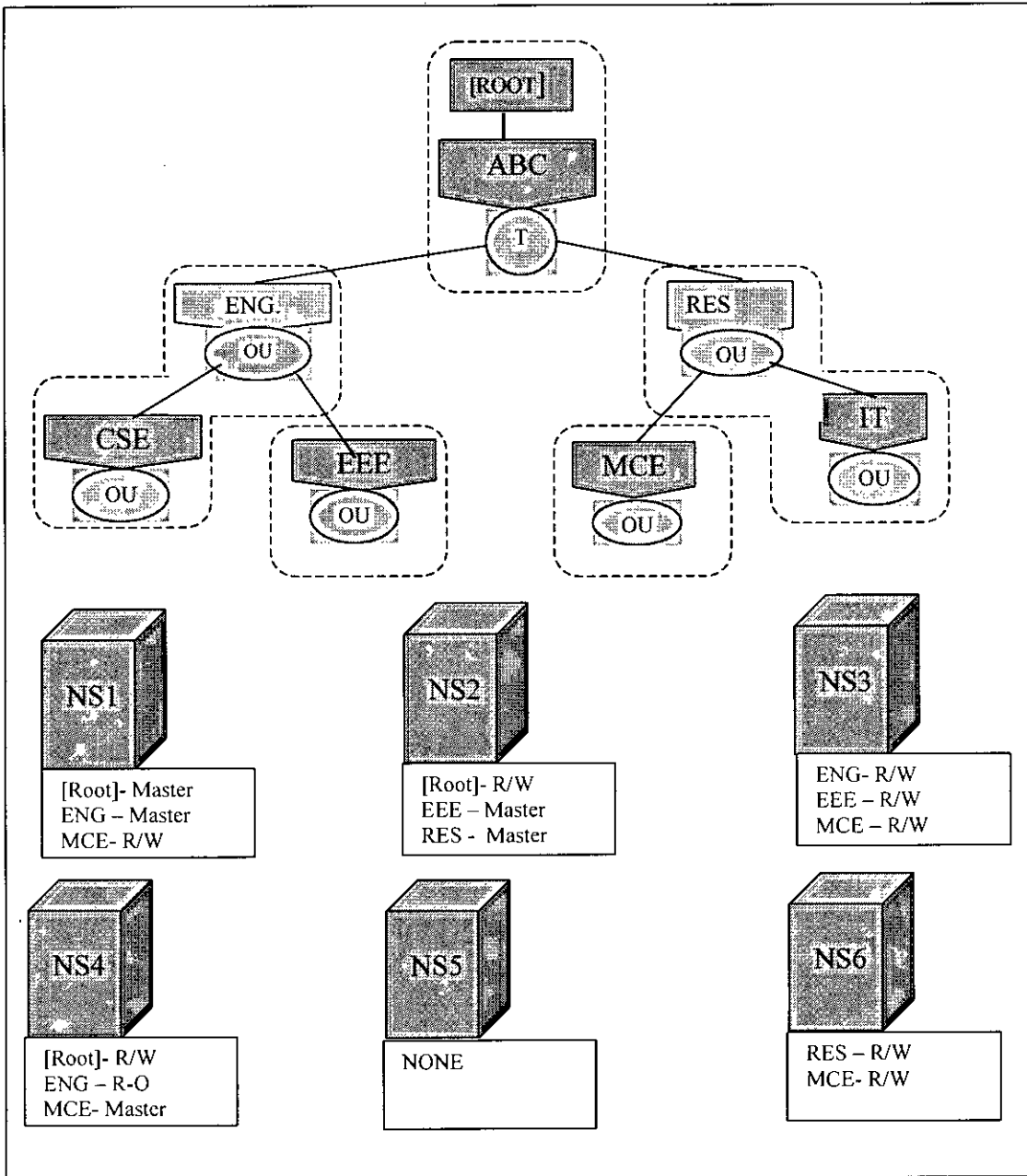
4.6.2 Subordinate References

Subordinate references, which are not visible to users, provide tree connectivity. Each subordinate reference is a complete copy of a given partition's root object but is not a copy of the whole partition. As a general rule, subordinate references are placed on servers that contain a replica of a parent partition but not the relevant child partitions. In other words, a subordinate reference points to an absent subordinate partition. In this case, the server contains a subordinate reference for each child partition it does not store.

Subordinate references provide tree connectivity by referring to replicas the server may need to find. Because the subordinate reference is a copy of a partition root object, it holds the *Replica* attribute, which lists all the servers on which replicas of the child partition can be found. NDS can then use this list to locate replicas of the subordinate partition.

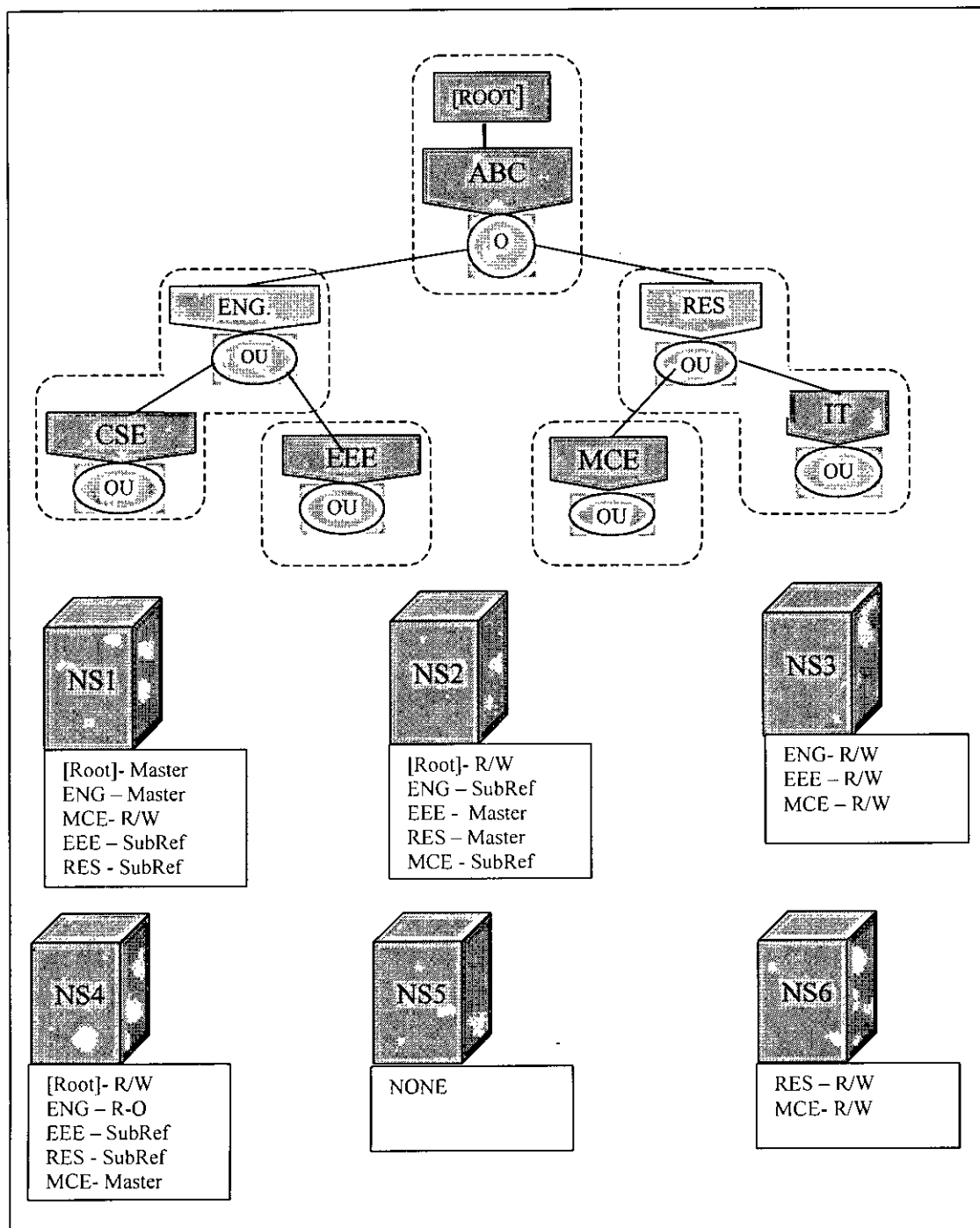
Figure 4.8 shows a partitioned tree and its subsequent replica placement on the servers that are holding the tree.

Figure 4.8: Replica placement in a partitioned tree.



Some of the servers in Figure 4.8 hold replicas of parent but not replicas of the corresponding child. These servers must also hold subordinate references to the child partitions they do not hold, as shown in Figure 4.9. For example, because server NS4 holds a replica of the ENG partition but not of EEE, it must hold a subordinate reference to EEE.

Figure 4.9: Replica placement and subordinate references.



On server NS1 in Figure 4.9, the subordinate reference of RES.ABC is a complete copy of the root object, RES.ABC, but not of its subordinate objects; the subordinate reference of EEE.ENG.ABC is a complete copy of the entire partition, since EEE.ENG.ABC is the only object in the partition. Users cannot change a subordinate reference's replica type.

Besides providing tree connectivity, subordinate references also help determine rights. Because a subordinate reference holds a copy of the partition root object, it holds that

object's *Inherited ACL* attribute, which summarizes the Access Control Lists up to that point in the tree.

4.6.3 Replica List

Each replica contains a list of servers that support the partition it represents. The replica list is stored in each replica as a *Replica* attribute of the partition's root-most container entry. This list provides information needed for navigating the NDS tree and synchronizing the replicas. The replica list contains the following elements for each replica:

- *Server Name*. The name of the server where the replica is located.
- *Replica Type*. The type of the replica stored on the server designated in the *Server Name* field. (The type is either *Master*, *RW*, *RO*, or *SR*.)
- *Replica State*. The status of the replica. (The statuses include *On*, *New*, *Replica dying*, among others.)
- *Replica Number*. The number that the master assigned to this replica at the time the replica was created.
- *Network Address*. The server's address.
- *Remote ID*. The Entry ID of the replica's partition root entry.

4.7 NDS Synchronization

Synchronization is the process of ensuring that all changes to a particular partition are made to every replica of that partition. The X.500 standard defines two synchronization mechanisms: master slave synchronization and peer-to-peer synchronization.

The master-slave mechanism requires that all changes be made on the master replica. That replica is then responsible to update all the other replicas (slave replicas).

In a peer-to-peer synchronization system, updates can be made to any read-write or master replica. At a predetermined interval, all servers holding copies of the same partition communicate with each other to determine who holds the latest information for each object. The servers update their replicas with the latest information for each replica. In NetWare, the synchronization time interval ranges from between 10 seconds to 30 minutes depending upon the type of information updated.

NDS uses both the master-slave and peer-to-peer synchronization processes, depending upon the type of change being made. The master-slave mechanism synchronizes operations such as partition operations that require a single point of control. The peer-to-peer mechanism synchronizes all other system changes. Most operations use peer-to-peer synchronization.



4.7.1 Loose Consistency

Because the NDS database must synchronize replicas, not all replicas hold the latest changes at any given time. This concept is referred to as *loose consistency* (called *transient consistency* in the X.500 standard), which simply means that the partition replicas are not instantaneously updated. In other words, as long as the database is being updated, the network Directory is not guaranteed to be completely synchronized at any instance in time. However, during periods in which the database is not updated, it will completely synchronize [48].

Loose consistency has the advantage of allowing Directory servers to be connected to the network with different types of media. For example, you could connect one portion of your company's network to another by using a satellite link. Data travelling over a satellite link experiences transmission delays, so any update to the database on one side of the satellite link is delayed in reaching the database on the other side of the satellite link. However, because the database is loosely consistent, these transmission delays do not interfere with the normal operation of the network. The new information arrives over the satellite link and is propagated through the network at the next synchronization interval.

Another advantage to loose consistency is that if part of the network is down, the changes will synchronize to available servers. When the problem is resolved, the replicas on the affected servers will receive updates.

4.7.2 Replica Synchronization Process

Because an NDS partition can be replicated and distributed across a network, any changes made to one replica must be sent to, or synchronized with, the other replicas. The synchronization process keeps data consistent across the network. *Time Stamps* and *Partition Information* components of NDS play a vital role in the synchronization process.

Time Stamps: One critical component in synchronization is the time stamp, which records information about when and where a given value in a given attribute was modified. When NDS updates a replica, it sends a modification time stamp with the data to be updated. The replica compares time stamps and replaces the old information with the new. A replica is considered synchronized when it has received the latest updates from all other replicas of its partition.

Partition Information: For normal operations, including synchronization, to be successful, the partition root object on each server must store several important attributes and their values:

- Replica Pointer(s)
- Partition Control attribute
- *SynchronizedUpTo* vector
- Synchronization cache

- Object Control attribute

In the source code synchronization is known as *Skulking*. The purpose of the synchronization operation is to check the synchronization status of every server that has a replica of a given partition. Factors that determine whether synchronization is necessary include the replica's convergence attribute, its replica type, and the time that has elapsed since the replica was last synchronized or updated. The system scans the partition records locally to decide which partitions need to be synchronized.

NDS provides a trigger, or heartbeat, every thirty minutes to schedule synchronization. The network administrator can adjust the trigger's time interval with the use of the DTrace console SET command.

The synchronization process involves updating all replicas with all the changes made to a partition since the last synchronization cycle. The synchronization process takes the replica list and synchronizes the replicas one at a time to the replica that has changed.

Since NDS is a loosely synchronized database, an update made at one replica propagates to other replicas of the partition over time. Any modification to the NDS database activates the replica synchronization process. When a change is made locally to an NDS entry on one server, the synchronization process wakes up to propagate the change to other replicas of the partition. There is a ten-second hold-down time to allow several updates to be propagated in one update session. Replica synchronization proceeds one replica at a time throughout the replica ring of a partition.

After a server successfully sends all pending updates to one replica, it goes on to the next replica until all replicas have been updated. If the operation fails for one or more replicas and they are not updated in one round of the synchronization process, it reschedules them for a later synchronization cycle.

4.8 Distributed Relationship Management

Distributed relationship management consists of three components that help keep NDS trees connected:

- External references
- Back links
- Obituaries

4.8.1 External References

A server usually stores replicas of only some of an NDS Directory's partitions. Sometimes a server must hold information about entries in partitions that the server does not store. Often, the server requires information about an entry in a parent partition. Having this information helps the local server maintain connectivity with the upper part of the tree. At other times, the server requires information about entries

in partitions that are not parents or children of partitions it stores; for example, the file system may need to refer to these entries, or an entry stored on the local server may need to refer to them.

NDS stores these types of information in *external references*, which are placeholders containing information about entries that the server does not hold. External references are not “real” entries because they do not contain complete entry information.

Besides providing connectivity, external references improve system performance by caching frequently accessed information. Currently, NDS caches only an entry’s public key. The *Modify Entry* routine is used to store the public key as an attribute on the external reference [48].

4.8.1.1 Creating External References

NDS creates external references when an entry is not stored on the local server:

- Authenticates and attaches to the server.
- Is added as a trustee to a locally stored file system or entry.
- Becomes a member of a locally stored group.

In addition, NDS creates external references when a replica is removed from the server. NDS calls change all of the entries in the removed replica into external references and mark them as expired.

Keep in mind the following two rules about creating external references:

- NDS never creates an external reference below a real entry in the tree.
- NDS never creates a subordinate reference below an external reference in the tree. Any subordinate references below an external reference will be removed during synchronization.

4.8.1.2 Deleting External References

On each server, NDS deletes expired external references if they have not been used within a specified time period. The system administrator can use a SET parameter to set a number of days after which NDS deletes external references that have not used, are not needed for another entry’s context, or do not contain information that the operating system needs.

To remove expired external references, NDS builds a list of unused external references by checking the life-span interval of each external reference. This interval defaults to eight days and thirty minutes.

NDS checks to see if the file system must access any of the external references. This process then deletes any external references not accessed by the file system within the life-span interval. The *Janitor* process then purges the deleted external references.

4.8.1.3 Synchronizing External References

When NDS updates entries and partitions, it also must update external references created for those entries.

After successfully synchronizing all the replicas of a partition, NDS checks any entry that has been renamed, moved, or deleted. All of these processes involve the *Remove Entry* process, which adds an obituary on the object being removed. If NDS finds any back link obituaries, it notifies the server that contains the entry's external reference to update that external reference.

4.8.2 Back Links

When NDS creates a new external reference, it also attempts to create a pointer to the server holding the external reference as one of the attributes of the non-local entry. This pointer is called a back link and is stored as a *Back Link* attribute. If NDS is unable to create the back link, it continues trying to create the link 9 times. The default retry interval is currently 3 minutes. If NDS cannot create the back link after 9 times, the *Back Link* process creates the back link [50].

NDS uses back links to update external references in the cases where the real object has been renamed or deleted.

The *Back Link* process executes on a time interval set by the network administrator. Currently, the default interval is 13 hours. The *Back Link* process has two basic functions:

- Remove any expired and unneeded external references from the system.
- Create and maintain any back links not created at the same time as the external reference.

4.8.2.1 Creating a Back Link

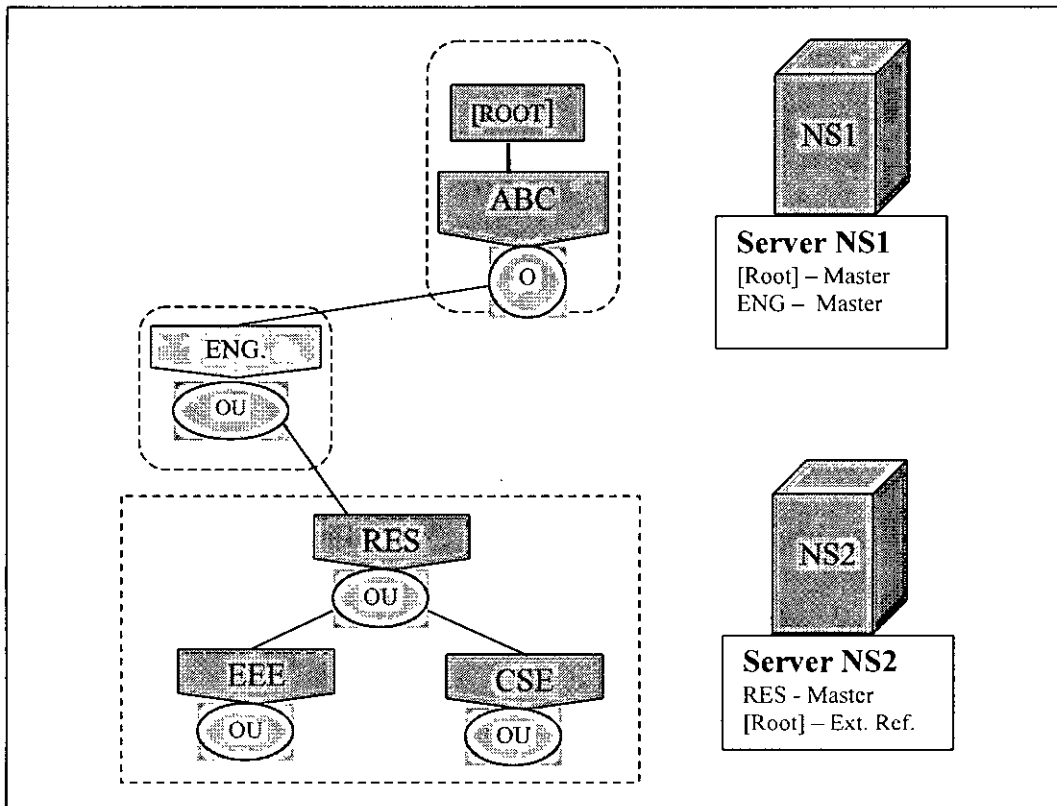
When NDS creates a new external reference for an entry not stored on the local server, NDS attempts to place a back link on the real entry. The back link points to the server that holds the external reference. For example, in the tree in Figure 4.10, partition ENG is stored on server NS1. Because partition RES is stored on server NS2, which does not store a copy of ENG, server NS2 needs an external reference for partition ENG to connect partition RES with [Root]. When NDS creates the external reference, NDS places a back link on server NS1's copy of entry ENG. This back link points to NS2.

In this case, server NS1 sends a Create Back Link request to NS2, which places the back link as an attribute value for the entry ENG.

4.8.2.2 Deleting a Back Link

When NDS removes an external reference, the back link to that external reference must be deleted. The server holding the external reference requests that the server holding the real entry deletes the back link, and the server holding the back link then deletes the reference.

Figure 4.10: Backlinks.



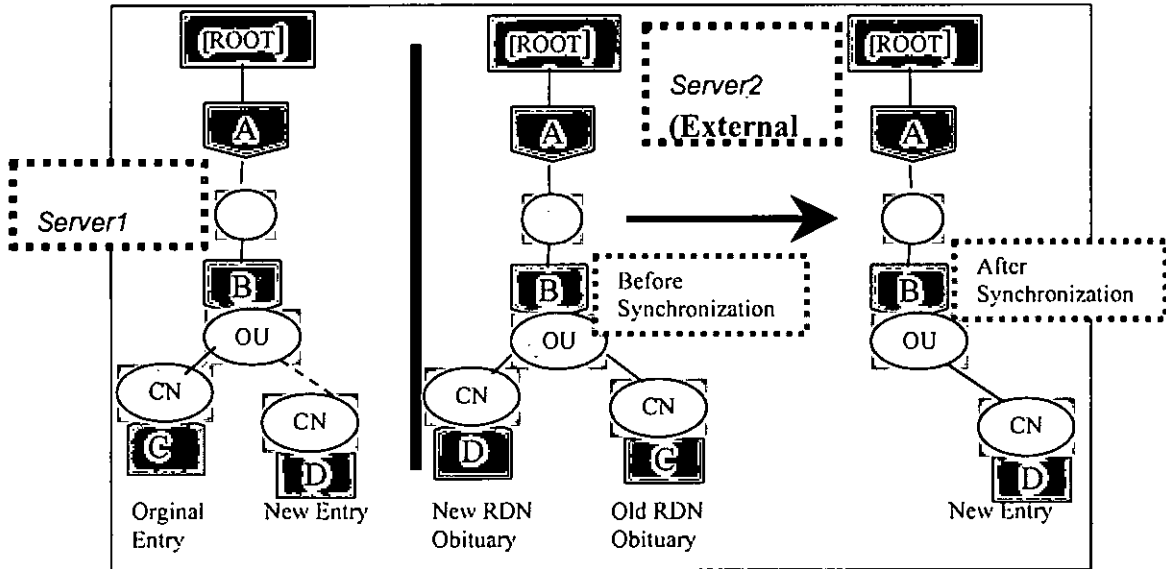
4.8.3 Obituaries

In a distributed database, each server receives updated information through synchronization. Because the servers do not receive updates simultaneously, the servers may not hold the same information at a given time. For this reason, each server holds on to the old information until all the other servers receive updates. NDS uses *obituaries* to keep track of such information.

For example, Figure 4.11 shows how obituaries are used when an entry is renamed. On server 1, the entry C is renamed to D. When server 2, which holds a replica of C, receives the update during synchronization, it keeps the copy of C and attaches a New RDN obituary to it. This obituary ensures that all servers can access C, even if they

have not been notified of the name change. When server 2 creates entry D, it attaches an Old RDN obituary pointing back to the original object. After all replicas have been synchronized, server 2 can delete its copy of C and remove the obituary from entry ID.

Figure 4.11: Obituaries.



Obituaries are attribute values that are not visible to clients and are used in server-to-server exchanges. Because obituaries are attribute values, NDS synchronizes them the same way it synchronizes other values. NDS synchronizes all obituaries across partition replicas.

4.8.3.1 Primary and Secondary Obituaries

The Back Link obituary is considered a *secondary obituary*. It keeps external references synchronized with the real entries. All other obituaries are *primary obituaries*, which keep track of entry-level modifications, including:

- Renaming an entry
- Deleting an entry
- Moving an entry
- Moving a subtree

Generally, when data is changed, primary obituaries convey the change(s) to servers holding the affected entry. Secondary obituaries convey the change to servers holding external references to the changed entry.

4.9 Network Time Synchronization

Time synchronization is important to the operation of NDS, as it establishes the order of Directory events that take place. Whenever an event occurs in the Directory, such as when an object is renamed or a password changed, NDS requests a time stamp. A time stamp is a unique code that includes the time an event took place and identification of the replica that initiated the event. Every NDS event is assigned a time stamp so that replicas can be updated in the proper order [37].

NDS uses four types of Time-Servers to synchronize the time over the network, which are:

- Secondary Time Server
- Primary Time Server
- Reference Time Server
- Single Reference Time Server

4.9.1 Secondary Time Server

A Secondary Time Server (STS) obtains the time from a Single Reference, Primary, or Reference time server and provides the time to clients (such as workstations or applications). A secondary time server does not vote to determine the correct network time.

Secondary time servers contact Primary or Reference time servers that are physically close in order to keep network traffic between the time servers to a minimum.

For optimal time synchronization, require each Secondary time server to contact a Primary, Reference, or Single Reference time server with as few intervening routers and slow LAN segments as possible.

4.9.2 Primary Time Server

A Primary Time Server (PTS) synchronizes network time with at least one other Primary or Reference time server, and provides the time to Secondary time servers and clients. A PTS polls other Primary or Reference time servers, and then votes with them to synchronize the time.

Primary time servers adjust their internal clocks to synchronize with the decided on common network time. Because all Primary time servers adjust their clocks, network time may drift slightly during this process.

A PTS is used primarily on larger networks to increase fault tolerance by providing redundant paths for Secondary time servers. If a Primary time server goes down, the Secondary time server can get the time from an alternate Primary time server. On a

large network, use at least one Primary time server for every 125 to 150 Secondary time servers.

4.9.3 Reference Time Server

A Reference Time Server (RTS) provides a time to which all other servers and clients synchronize. Reference time servers should be synchronized with an external time source, such as from a radio clock that receives time signals from the Naval Observatory or some other accurate time source.

An RTS polls other Primary or Reference time servers, then votes with them to synchronize the time. Because the Reference time server does not change its clock, the Primary time servers must reach consensus with the time provided by the Reference server.

An RTS is used when it is important to have a central point to control network time.

Reference and Single Reference time servers are two distinctly different types of time servers. Do not confuse the two.

4.9.4 Single Reference Time Server

The Single Reference Time Server (SRTS) provides time to Secondary time servers and clients. When a Single Reference time server is used, it is the sole source of time for the entire network. The network supervisor sets the time on the Single Reference time server.

The SRTS can be used for all networks, regardless of size, but is primarily recommended for small networks.

Important: Because the Single Reference Time Server is the sole source of time on the network it is installed on, all other servers on the network must be able to contact it. When the Single Reference Time Server is used, you cannot have any Primary or Reference time servers on the network.

Single Reference, Reference, and Primary time servers are all time source servers. Time source servers provide time to the network. Secondary time servers do not provide time to the network; they only receive the time from a time source server and pass the received time along to clients.

Chapter 5

Directory Services Enabled .bd Domain

5.1 Introduction

This section of the research works discusses about the possible standards for .bd domain of Bangladesh. In this section, we have introduced Directory Services enabled Domain Name System for .bd domain. We have also discussed about Naming Standards, IP address allocation guidelines, NDS partition and Zone Transfer of .bd domain.

We have also introduced traditional model of DNS for .bd domain, and discussed how NDS enabled DNS gives us the countrywide fully managed fault-tolerant Domain Name Space.

5.2 Definition of Component Terms

To develop the standard for the country domain we have classified the information of this chapter into different classes, which are:

- Mandatory
- Recommended
- Optional

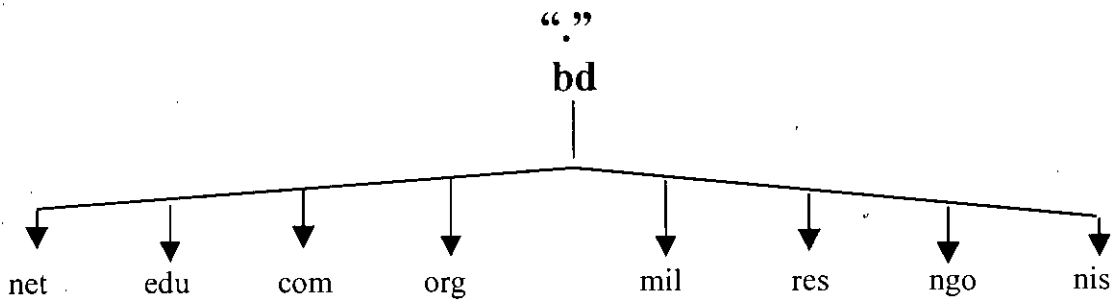
Classifications definition are given below:

Mandatory	Information which must be adhered to as specified.
Recommended	Information which provide a best practice suggestion which should be followed where ever possible. Failure to adopt these recommendations could cause significant problems, therefore, organizations who feel they need to use an alternative should check with the .bd management committee before proceeding.
Optional	Information which provides standards, and a range of choices from which the organization can choose a solution to suit their local requirements.

5.3 Traditional Architectural Model of .bd Domain

Figure 5.1 shows the proposed traditional DNS model of .bd Domain of Bangladesh. In this model, we have considered present architecture of Internet as a standard for .bd Domain. As Directory Services enabled Internet is going to be a 21st Century Architecture of Internet, we have also developed directory services enabled .bd Domain Standard for the Country network standard, which is discussed in section 5.4.

Figure 5.1: The .bd Domain Name Space Tree.



We have divided .bd Country Domain into eight second-level domains/Zones, which are the child of .bd domain. Adherence to this structure and standard is **Mandatory**

The .bd Domain Name Space (DNS) Tree in Fig. 5.2 shows the following domains:

.bd. – Country Top Level Domain of Bangladesh

nis.bd. – Second Level Domain for all government bodies including ministries, who are providing services to nation but not involved in business.

ngo.bd. – Second Level Domain for all non-government organizations of Bangladesh

res.bd. - Second Level Domain for all research organization (including government, non-government, private etc.) of Bangladesh

mil.bd. – Second Level Domain for Department of Defense of Bangladesh.

org.bd. – Second Level Domain for all autonomous and international organizations in Bangladesh, who are not involved in business.

com.bd. – Second Level Domain for all commercial organization (government, non-government, private and public) of Bangladesh

edu.bd. – Second Level Domain for all educational institutions (government & private school, college, institute, and university) of Bangladesh.

net.bd. – Second Level Domain for network service providers of Bangladesh.

5.4 Directory Services Enabled DNS

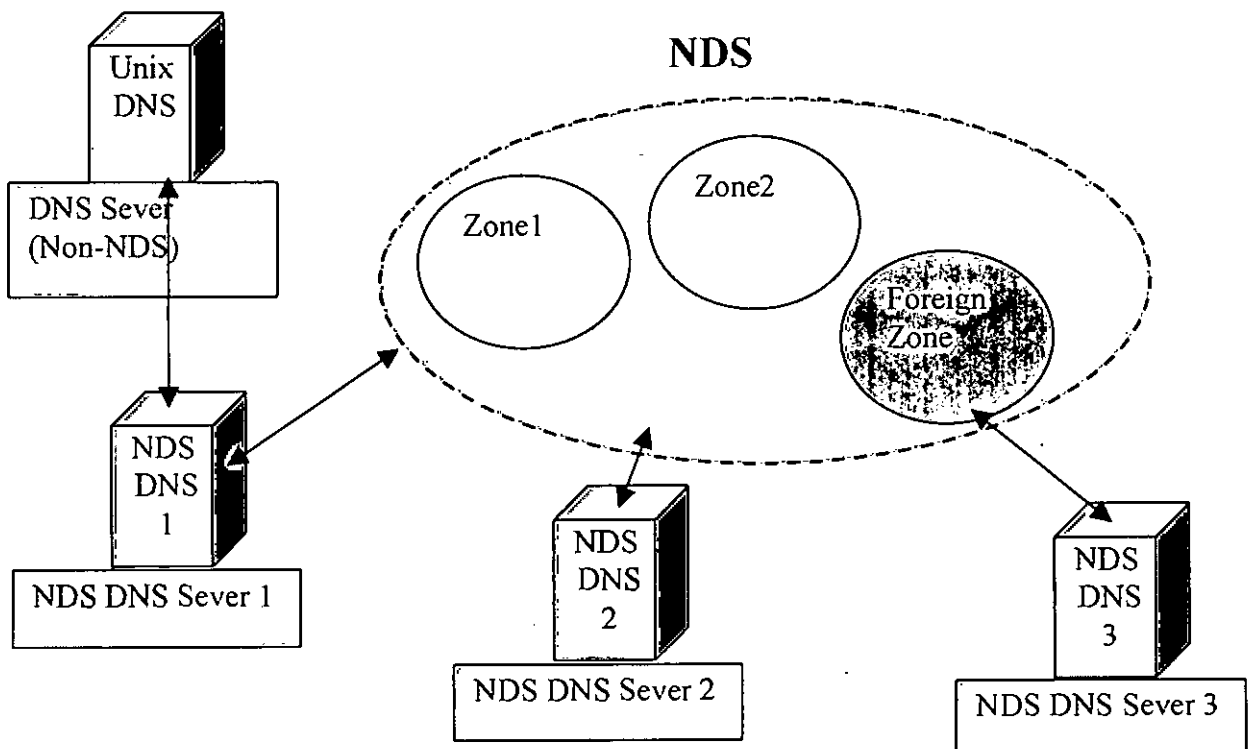
NDS enabled DNS greatly simplifies the task of network administration by enabling all configuration information into one distributed database. Furthermore, the DNS configuration information is replicated just like any other data in NDS.

NDS enabled DNS has shifted the concept of a primary or secondary zone away from the server to the zone itself. Once the zone is configured the data is available to any of the authoritative NDS-DNS servers for the zone. The server assigned to this function is called the **designated server**. The NDS-DNS server takes advantage of the peer-to-peer nature of NDS by replicating the DNS data.

The NDS enabled DNS servers inter-operate with other non-DNS servers. The NDS DNS server can act as either a master DNS server or a secondary DNS server in relation to non-NDS DNS servers. The NDS-DNS server can act as the master DNS server and transfer data to non-NDS secondary servers. Alternatively, one NDS-DNS server can act as a secondary DNS server and transfer data in from a non-NDS master server. All NDS-DNS servers can then access the data through NDS replication.

Figure 5.2 illustrates NDS-DNS as the Primary DNS name server with primary and secondary zones within NDS. In this model there are two primary zones. Any of the NDS-DNS servers assigned to the zone are able to respond to queries for the zones. For each zone, one server is designated to act as the designated server. In this model Server1 is the designated server for Zone 1, Server2 is the designated server for Zone 2, and Server3 is the designated server for the secondary zone called Foreign Zone. Server 3 will occasionally request zone transfers from the foreign server and place the modified zone data into NDS where any of the NDS servers can respond to queries for it.

Figure 5.2: NDS Enabled DNS Model



5.4.1 New NDS Objects for DNS

The following new NDS objects support DNS:

- DNS Zone object
- DNS Resource Record Set object
- DNS Name Server object

5.4.1.1 DNS Zone Object

The DNS Zone object is a container object that contains all the data for a single DNS zone. A Zone object is the first level of the DNS zone description.

Multiple DNS domains can be represented within NDS by using separate, independent DNS Zone objects. Multiple DNS domains could be supported on a single NDS server by creating multiple DNS Zone objects and assigning the server to serve those zones.

The DNS Zone object contains data that correlates to a DNS Start of Authority (SOA) Resource Record (RR) - a member list of all NDS-based DNS servers that support the zone, and designated server information.

The DNS name space hierarchy is not represented within the NDS hierarchy. A zone and its child zone might appear as peers within the NDS hierarchy, even though they have a parent-child relationship within the DNS hierarchy.

5.4.1.2 DNS Resource Record Set Object

The DNS Resource Record Set (RRSet) object is an NDS leaf object contained within a DNS Zone object. An RRSet object represents an individual domain name within a DNS zone. Its required attributes are a DNS domain name, a DNS address class, and a Time-to-Live (TTL) record.

Each domain name within a DNS zone object has an RRSet object. Each RRSet object has one or more resource records beneath it containing additional information about the domain, including a description of the object and version information.

5.4.1.3 DNS Resource Records

A DNS Resource Record (RR) is an attribute of an RRSet that contains the RR type and data of a single RR. RRs are configured beneath their respective RRSet objects. Resource records describe their associated RRset object.

The most common RRs are A records, which map a domain name to an IP address, and PTR records, which map an IP address to a domain name within an IN-ADDR.ARPA zone.

5.4.1.4 DNS Server Object

The DNS Server object (or Service object) is a separate entity from the NDS Server object. A DNS Server object can be contained in an O, OU, C, or L. The DNS Server object contains DNS server configuration parameters, including the following:

- Zone List
- DNS Server IP Address
- DNS Server Options
- Forwarding List
- No Forwarding List

5.4.2 New NDS objects for DHCP

The following new NDS objects support DHCP:

- Subnet object
- Address Range object
- IP Address object
- DHCP Server object
- Subnet Pool object

The Subnet object represents a subnet and is the most fundamental DHCP object. The Subnet object can be contained by an Organization (O), Organizational Unit (OU), a Country (C), or a Locality (L). The Subnet object acts as a container object for the IP Address and Address Range objects. A Subnet object's specific DHCP options and configuration parameters apply to the entire subnet and override global options.

The Address Range object is used primarily to create a pool of addresses for dynamic address assignment or to identify a range of addresses to be excluded from address assignment. Optionally, the Address Range object also stores the start of a host name that can be assigned to clients when addresses are assigned.

Multiple address range objects could be used under a subnet object. Different range types such as, a range for dynamic address assignment, a range for BOOTP clients, or a range to be excluded from the subnet could be specified under a single sub-net object.

The IP Address object represents a single IP address. The IP Address object includes an address number and an assignment type. The assignment type is "dynamic" for address objects created by the server to represent addresses assigned dynamically from an address range.

For dynamically or automatically assigned client addresses, DHCP creates an IP Address object under the subnet where the address is assigned. An IP address can be assigned to a client based on the client's MAC (Media Access Control) address i.e. hardware address. These IP Address objects can also receive specific DHCP or

BOOTP options. Address objects contain the lease expiration time for both dynamically and statically assigned addresses.

The DHCP Server Object represents the DHCP server and contains a multi-valued attribute listing of the subnet ranges this DHCP server is servicing. The DHCP server also contains all server-specific configuration and policy information.

The Subnet Pool object provides support for multiple subnets through a DHCP or BOOTP forwarder by identifying a pool of subnets.

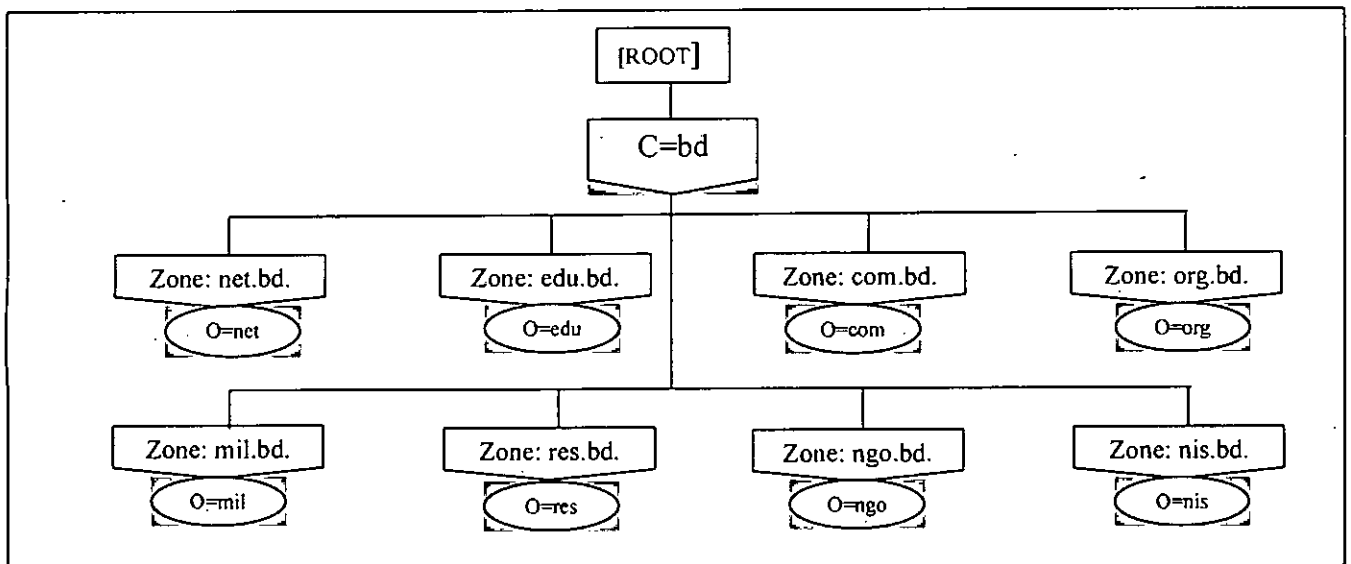
DHCP servers are not required to be on the local subnet to which they assign addresses. If desired, they can be deployed centrally and service remote subnets. Initial DHCP/BOOTP DISCOVER requests, however, will not be sent to a DHCP server unless a DHCP/BOOTP forwarder local to the client has been configured to forward the addresses.

The Subnet Pool object contains a list of subnet object references (a list of subnet objects of the same type within the pool) and comments.

5.5 NDS Enabled DNS Model of .bd Domain

Figure 5.3 shows the countrywide single NDS based DNS Tree of .bd domain, which will enable to manage the entire .bd domain from a single point of the domain. In future this DNS tree will be merged with the Global Internet NDS Tree, which is being considered as the backbone of future Internet architecture.

Figure 5.3: NDS based .bd model



From the architectural point of view, we have divided .bd Domain into eight Zones and each Zone will be managed by it's own dedicated DNS server. All Zones' DNS

server will be linked with .bd DNS server, which will act as the Master DNS Server for the country.

Figure 5.3 NDS based .bd model, shows the Zones/Organization objects of .bd NDS DNS Tree:

bd – Country Top Level Domain/Country NDS Object of Bangladesh

Zone: nis.bd. – Second Level Domain/Zone for all government bodies including ministries, who are providing services to nation but not involved in business.

Zone: ngo.bd. – Second Level Domain/Zone for all non-government organizations of Bangladesh

Zone: res.bd. - Second Level Domain/Zone for all research organization (including government, non-government, private etc.) of Bangladesh

Zone: mil.bd. – Second Level Domain/Zone for Departments of Defense of Bangladesh.

Zone: org.bd. – Second Level Domain/Zone for all autonomous and international organizations in Bangladesh, who are not involved in business.

Zone: com.bd. – Second Level Domain/Zone for all commercial organization (government, non-government, private and public) of Bangladesh

Zone: edu.bd. – Second Level Domain/Zone for all educational institutions (government & private school, college, institute, and university) of Bangladesh.

Zone: net.bd. – Second Level Domain/Zone for network service providers of Bangladesh.

Figure 5.4 shows the Zone/NDS Partition model of .bd Domain. Country Domain NDS Tree is partitioned as per the Zones depicted in Figure 5.3 model, and master partition of NDS Tree is .bd domain itself. Each Zone and Master Domain server will hold the replica of other zone as per the replication model depicted in Figure 5.5.

Figure 5.4: NDS Partition/Zone Model of .bd Domain

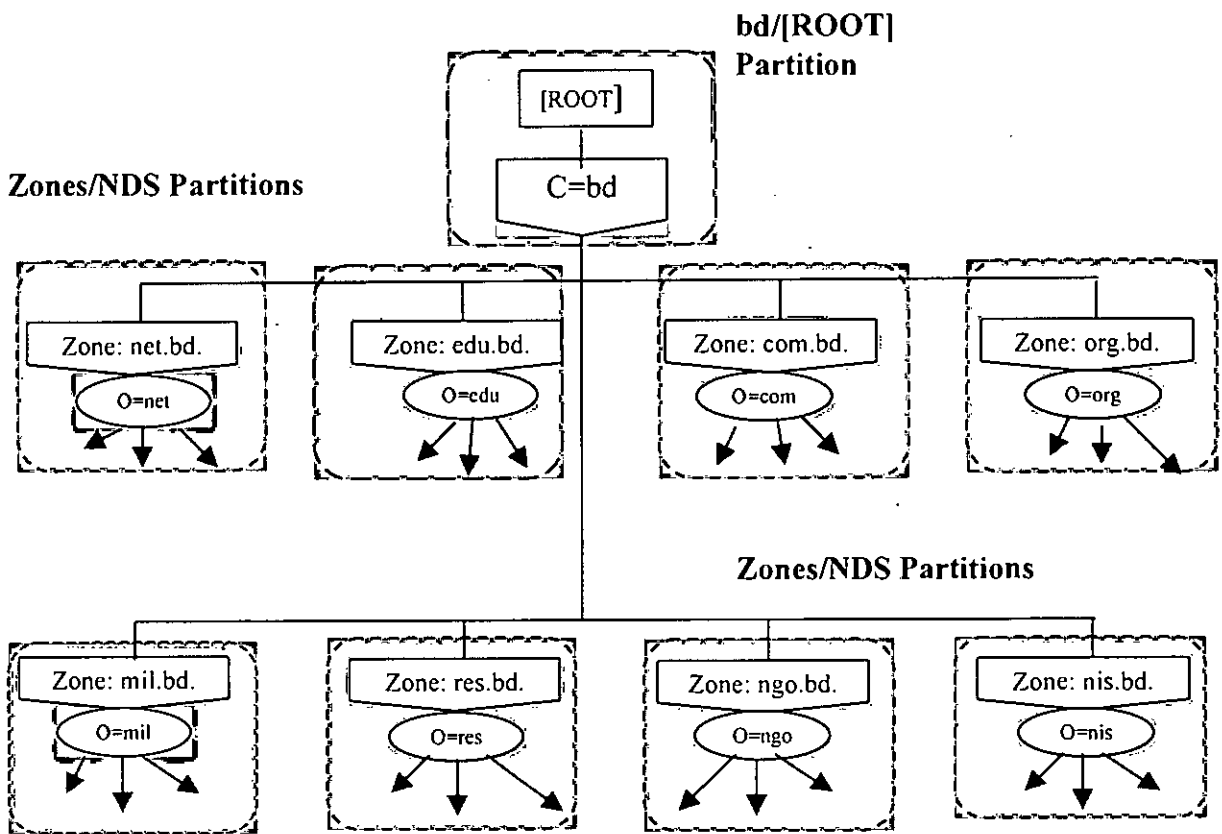
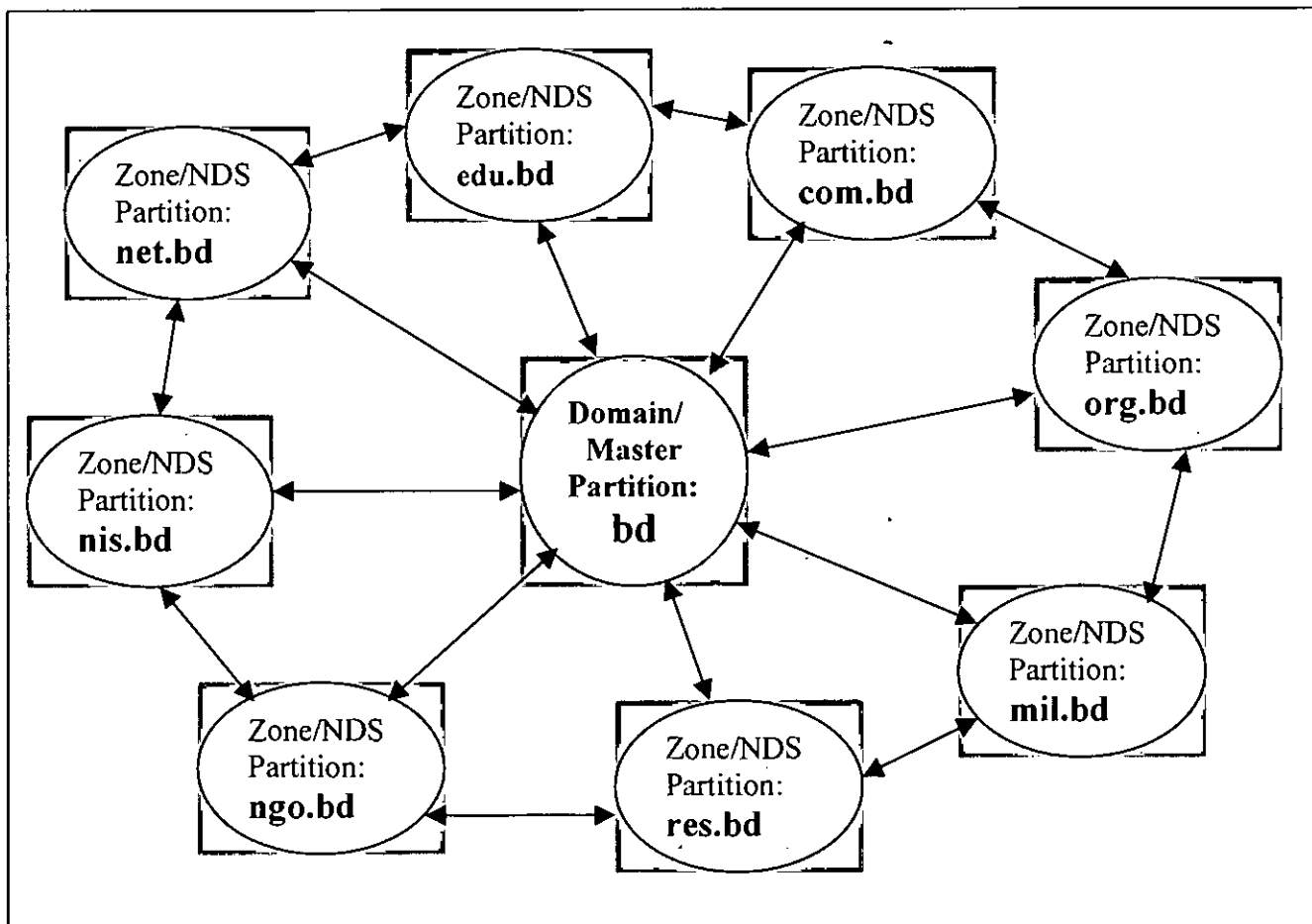


Figure 5.5 shows the Zone Transfer/NDS Replication model of .bd Domain Tree, which ensure the fully fault-tolerant country domain systems for the Bangladesh. Replication model ensures the availability of each Zone replica into Master DNS server and the Master Partition into each Zone Server.

Figure 5.5: NDS Replication/Zone Transfer Model of .bd Domain



In this model if one Zone server goes down other nearest server will handle the network information of the downed server.

5.6 Domain and Objects Naming Standards

The domain and objects defined in the NDS based .bd domain tree must adhere to standards to ensure a consistent and predictable method of identifying each domain and object. As a result, a domain and objects naming standard have been developed.

Adherence to this standard is **Mandatory**.

5.6.1 Objectives

This section defines the standards for the domain and objects in the .bd Tree. The standards are described in the following terms:

- General naming standards
- Sub-domains/Container objects
- User Accounts/Leaf objects

5.6.2 Use of Characters in Domain Object Naming

This section identifies which characters are acceptable for use in the .bd Domain Tree.

5.6.2.1 Acceptable characters

Mandatory The following characters are the only characters to be used in the NDS Tree when assigning names to objects:

Table 5.1: Characters that should only be used within the Domain TREE

	Acceptable characters
Numeric	1 2 3 4 5 6 7 8 9 0
Alpha	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z a b c d e f g h i j k l m n o p q r s t u v w x y z
Non Alpha/Numeric	_ underscore - hyphen [square bracket - open] square bracket - close

5.6.2.2 Use of underscore, space and hyphen

Mandatory NDS will read a space and an underscore as exactly the same character. Therefore the following standards must be used:

- A space character should **not** be used in an object name unless specifically required. An underscore should be used to represent a space between two words in an object name.

Example: Khaled_Mahmood

- The hyphen is used where multiple words have been abbreviated or as a delimiter in a composite name such as:

Example: Bangladesh Computer Council, Dhaka could be abbreviated as BCC-DA, and Bangladesh Telegraph & Telephone Board, Kushtia could be abbreviated as BTTB-KS.

5.6.2.3 Country NDS Tree

Our objective is to use only one live Country NDS Tree.

The name of the Country NDS Tree is: **BD_TREE**

IMPORTANT
To ensure no confusion arises between description of services The name "BD_TREE" must not be used by any other:
<ul style="list-style-type: none"> ▪ Tree set up on the .bd country network ▪ Live network device or service

5.6.2.4 Local NDS Tree

Local NDS Trees may be required as part of the preparation for the solitary Country NDS Tree (BD_TREE).

Mandatory Local NDS trees must be designed and implemented according to the *Novell Directory Services Overview* and *Novell Directory Services Naming standards* available from the Novell Document set. The NDS trees must have their own unique name in order to prevent coexistence of multiple trees with identical names, which will lead to potential corruption of all trees involved in the collision.

Mandatory The following naming convention must be applied to all Local NDS Trees.

Table 5.2: Naming convention for production NDS Tree

Local Production Tree	BD prefix	Delimiter	Second Level Domain Code	Delimiter	Third Level Domain Code	Delimiter	Suffix
Number of characters	2	1	3	1	5	1	4
Alpha	A	Underscore	A	Underscore	A	Underscore	A
Naming Convention	BD std.	NOS std	BD std.	NOS std	BD std.	NOS std	BD std.
Example	BD	_	EDU	_	BUET	_	TREE

The example in the above table creates the following name for a Local NDS tree created at BUET: **BD_EDU_BUET_TREE**

5.6.3 Second Level Domain Objects

This section detail the standards associated with second level domain objects in the Tree.

The Second Level Domain objects represent the Organisation (O) objects of the NDS Tree.

Table 5.3: Characteristics of the Second Level Domain objects /Organisation objects name

Use of Object	Represents the second level domain of the country
Number of characters	3
Alpha	A
Classification	Mandatory

These objects are unique within the Country Domain Tree.

5.6.4 The Third Level Domain Objects/Organisational Unit (OU) Objects

Table 5.4: Characteristics of the Third Level Domain Objects/Organisation Unit (OU) objects

Use of Object	Represents Third Level Domain Objects
Number of characters	5
Alpha or /and Numeric	A+AN
Classification	Recommended

Fourth and Lower Level domain objects should follow Third Level Domain Objects standard.

5.6.5 User Accounts of Domains or Leaf Objects

Recommended User names created within a Domain must be unique.

To ensure uniqueness of user naming, names should be checked against the existing list of user accounts of the particular Domain.

Table 5.6: Characteristics of a User objects/User accounts

Use of Object	Represents a user ID
Number of characters	8
Alpha or /and Numeric	A+AN
Classification	Recommended

The user object name or User ID should be constructed with the first eight characters of a user's last name. Use the entire last name if a user has less than eight characters in their last name.

Table 5.7: Implementation of a User object

Surname	First Name	Middle Name	User ID
Rahman	Khandaker	Mirazur	Rahman
Islam	Mohammad	Aminul	Islam
Shamsuddin	Mohammad		Shamsudd

Do not use hyphens or spaces in hyphenated or double barrelled last names like Bart-Williams, Clare-Whitecker, Rob-Tyre. Treat the two last names as one for the purpose of naming a user object/User ID.

Table 5.7: Non-use of hyphens within a User Name object

Surname	First Name	Middle Names	User ID
Graham-Hyde	Andrew	Roger	GRAHAMHY
Bart-Williams	Clare	Lisa	BARTWILL

5.6.6 Handling of Duplicate Names

Names should be kept unique within a Domain.

If a user login ID has been created for a user and the same login name is then required by users with the same last name, there must be a method of distinguishing between the users. When same Last Name is found use First and Middle Names initial along

with the Last Name to create a unique User ID. Table 5.8 shows the method of creating unique User ID whose last name is same.

UserID = LastName/+ First & Middle Names Initial.

Table 5.8: Sole identity of each User Name object

Request	Surname	First Name	Middle Names	User ID	Action
1 st Request	Islam	Mohammad	Rabiul	Islam	use last name
2 nd Request	Islam	Mohammad	Rafiqul	IslamM	add 1 st initial from First Name
3 rd Request	Islam	Mohammad	Raqibul	IslamMR	add 2 nd initial from Middle Name
4 th Request	Islam	Mohammad	Reazul	IslamMR1	add '1' at end of name
5 th Request	Islam	Mohammad	Rezaul	IslamMR2	use next number in sequence

If IslamMR9 is reached and another IslamMR is required, the middle initial is dropped and the next available user name in the IslamMR sequence is used. The IslamMR sequence can go to 99 on qualifiers.

In the above example, the last name is less than eight characters. The table below shows the same principle can be assigned to a user ID, which is based on a name that is greater than eight characters long. The last characters of the name are replaced as appropriate.

Table 5.9: Non standard User Objects

Request	Surname	First Name	Middle Names	User ID	Action
1 st Request	Chowdhury	Mizanur	Rahman	Chowdhur	use last name
2 nd Request	Chowdhury	Mirazur	Rahman	ChowdhuM	input 1 st initial at end
3 rd Request	Chowdhury	Mahmud	Hussain	ChowdhMH	input 2 nd at end
4 th Request	Chowdhury	Mobarrak	Hussain	ChowdMH1	input '1' at end
5 th Request	Chowdhury	Momenul	Hussain	ChowdMH2	use next number in sequence

5.7 IP Address Space Allocation Standard

To bring the Bangladesh under a single IP network, we need a registered Class A address from InterNIC which could cover all hosts in the Country. To develop the IP Address Allocation Standard for the .bd Domain, we are assuming that our Class A address is x.0.0.0 and net-mask 255.0.0.0, where x could be any value within the Class A address range.

Adherence to this standard is **Recommended**.

The Class A address X.0.0.0 covers IP addresses range from X.0.0.0 to X.255.255.255, which could be equally divided into eight blocks like:

X.0.0.0 – X.31.255.255

X.32.0.0 – X.63.255.255

X.64.0.0 – X.95.255.255

X.96.0.0 – X.127.255.255

X.128.0.0 – X.159.255.255

X.160.0.0 – X.191.255.255

X.192.0.0 – X.223.255.255

X.224.0.0 – X.255.255.255

Each block represents 2,097,152 addresses, which will be managed by each second level domain/Zone of .bd domain. The allocation of IP Address Block to each Zone/Second Level Domain (alphabetically) are given below:

com.bd.	X.0.0.0 – X.31.255.255
edu.bd.	X.32.0.0 – X.63.255.255
mil.bd.	X.64.0.0 – X.95.255.255
net.bd.	X.96.0.0 – X.127.255.255
ngo.bd.	X.128.0.0 – X.159.255.255
nis.bd.	X.160.0.0 – X.191.255.255
org.bd.	X.192.0.0 – X.223.255.255
res.bd.	X.224.0.0 – X.255.255.255

The assigned IP address range for each Zone/ Second Level Domain will cover all the hosts connected under the particular Zone/ Second Level Domain. To manage each Zone/ Second Level Domain IP addresses assignment efficiently, each block could be divided into sub-blocks, where each sub-block will contain 256 addresses with sub-net mask 255.255.255.0. Each sub-block could be further divided into partial sub-block containing 128 addresses with sub-net mask 255.255.255.128. Allocation of sub-block and partial sub-blocks will depend on the requirement of the organizations.

There will be a Central Domain Management Committee (CDMC), who will be responsible for overall country DNS Tree. For each zone there will be a Zone Management Committee (ZMC), who will be responsible for individual Zone/ Second Level Domain. Each ZMC will assign the IP addresses to the organizations related with its' own Zone. The range of IP addresses assignment to an organisation will be based on their number of LANs, hosts and the future expandability in terms of LAN and hosts of the organisation. In that case, ZMC should follow the following guidelines to assign the IP address spaces to the organizations:

Organization's Requirement	Assignment
1) requires fewer than 256 addresses	1 sub-block address space
2) requires fewer than 512 addresses	2 contiguous sub-block address space
3) requires fewer than 1024 addresses	4 contiguous sub-block address space
4) requires fewer than 2048 addresses	8 contiguous sub-block address space
5) requires fewer than 4096 addresses	16 contiguous sub-block address space
6) requires fewer than 8192 addresses	32 contiguous sub-block address space
7) requires fewer than 16384 addresses	64 contiguous sub-block address space

If the subscriber or the organization's network is divided into logically distinct LANs across which it would be difficult to use the given number of sub-block network numbers, the above criteria may apply on a per-LAN basis. For example, if a subscriber has 600 hosts equally divided across ten Ethernets, the allocation to that subscriber could be ten partial sub-block network numbers; one for each Ethernet. The subscriber would have to support the request of ramification from the stated criteria with an engineering plan.

These criteria are not intended to cause a subscriber to subnet the sub-block networks unnecessarily. Although, if a subscriber has a small number of hosts per subnet, the subscriber should investigate the feasibility of subnetting the sub-block network numbers rather than requesting one Sub-block network number for every subnet. In cases where the lack of Sub-block subnetting would result in an extravagant waste of address space, the ZMC may request a detailed engineering plan of subnetting from the organization to evaluate its requirement.

If a subscriber has a requirement for almost 4096 unique IP addresses it could conceivably receive 16 contiguous sub-blocks. However, there are cases where a subscriber may request a larger block of Sub-block network numbers. For instance, if an organization requires fewer than 8192 addresses and requests 32 Sub-block network addresses, the ZMC may honour this request. The maximal block of Sub-block network numbers that should be assigned to a subscriber consists of 64 contiguous Sub-block networks. This would correspond to a single IP prefix of 18 bits. Exceptions from the above stated criteria need to be determined on a case-by-case basis.

5.8 Benefits of NDS enabled DNS Services

The NDS enabled DNS Services provides the following DNS features:

- All DNS configuration is done in NDS, facilitating enterprise-wide management.
- A NDS-DNS server can be a secondary name server to another zone (DNS data loaded into NDS through a zone transfer), or it can be a primary name server
- DNS data can be read in from a BIND Master file to populate NDS for convenient upgrades from BIND implementations of DNS.
- DNS data can also be exported out of NDS into BIND Master file format. Root server information is stored in NDS and shared by all NDS-based DNS servers.
- Zone transfers are made to and from NDS through NDS servers and include interoperability with non-NDS-based DNS.
- A NDS-DNS server can be authoritative for multiple domains.
- NDS-DNS servers maintain a cache of data out of NDS so they can respond to queries very quickly.
- A NDS-DNS Server can act as a caching or forwarding server instead of being authoritative server for zones.
- The NDS DNS Services support multi-homing.
- The NDS DNS Services support round-robinning of responses to queries with multiple A records for a domain name.

The NDS DNS Services conforms to RFCs 1035, 1036, and 1183, and is based on BIND 4.9.5.

Chapter 6

Conclusion

6.1 Conclusion

In this thesis, a traditional .bd Domain, and Directory Services Enabled .bd Domain structures have been proposed. A detailed study have been made on TCP/IP protocol suite, traditional DNS, X.500 Directory Services, and Novell Directory Services to develop the Country Network standards.

TCP/IP protocols have been chosen as the standard backbone protocol for the country network. TCP/IP is being worked as a standard protocol for the IT industry and the future Information Technology is being developed based on TCP/IP. Considering the present trend of IT, TCP/IP address allocation standards have been developed, which will guide to manage the country IP-Net more effectively.

Domain Name System (DNS) is the core structure of the Internet, and Internet has worked as a key enabler to develop the Today's IT industry. Present and future technological development is being attained based on Internet technology. As DNS is the basic structure of the Internet, a traditional DNS standard has been developed for .bd Domain to adopt with the present Internet technology with the country network.

X.500 a standard, a concept, a direction for future Network Computing Architecture of Information Technology. Global Directory Services is the leading concept in the IT industry, which is being worked as a key enabler to develop the future global networking. Based on the strategic direction of Directory Services (DS) in the IT industry, a DS enabled .bd Domain standards have been developed to face challenges of next millennium.

6.2 Limitations

Basic standards have been developed and based on these standards other objects standard like printers, print queue, print server, computer, directory map object, group, application object, etc. could be developed for the real life implementations of directory services in the .bd domain.

Security of the information is a vital factor in today's networking, different operating Systems or platforms use different security techniques/technology for their systems. Standardisation and adoption of Information Technology security standards for the country network are not discussed in this thesis.

In this research work, main concentration has been given on Domain Name System and Directory Services standard, but we have not discussed about physical layer standards for the country network.

Practical experience on Information Technology industry, and knowledge base on different Network Operating Systems (NOS) are a pre-requisite for a researcher, which could be treated as a major limitation for this type of thesis work.

6.3 Future Development

Successful implementation of directory services depends on the country network standard, and the systematic approach of offerings of the services. The complexity of implementation could be reduced if the offerings of the services become more transparent to its users. Several factors must be considered as these services are the co-operative effort among the technical, administrative, organizational, and legal disciplines. Standards have been developed but, procedures must be defined and agreed at the initial phase of implementation of the Directory service.

Following are the issues that should be addressed before implementation of the Directory Services country-wide, and those could be resolved by further research work on this thesis:

- Country legislation on information sharing
- Contents of the Directory
- Security Standards of the Network
- Data Integrity
- Country-wide Distributed National Database

Establishing a Directory service within a country/organization will involve a great deal of co-operative effort. It is essential to get commitment from the integral parties of a country/organization at the onset. This includes the technical, legal, and data management components of the country. Executive level commitment will make it much easier to get the co-operation necessary.

6.3.1 Country legislation on information sharing

The information provided by the Directory Services will be in electronic format to its users. The users of the Directory Services may be from inside the country or may be from outside the country. The information provided by the Directory Services may be considered as Private by one country/organisation and not by another, or there may be a classification of information access.

Operational procedures must be clearly defined, as the inclusions in globally distributed services have wide visibility. Adherence to these procedures must be maintained at all level of information access, as misinformation may result in unintentional legal violations and unreliable access of data can adversely affect on organisations' reputation.

Therefore, it is necessary to be aware of the legalities and restrictions on electronic information sharing of the countries. Some countries have published a Code of Conduct with the Internet Engineering Task Force (IETF), explicitly stating the legal restrictions on directory and list data [4]. A Code of Conduct on information sharing needs to be developed for the country network, and that needs to be approved by IETF.

6.3.2 Contents of the Directory

The contents of the Directory Services information are totally dependent on the requirements of a particular country or organisation. The information provided in the directory is tightly coupled with the purpose. If the purpose is to provide addressing information for individuals, then customary information would include: Name, address, phone, e-mail address, facsimile number, pager, etc. If the use of the directory is to facilitate electronic mail routing, then the destination mail address needs to be included for each user. No other information should be presented in the directory if it is not directly related to the purpose.

If the directory is internal only, it may be desirable to include the registrants' title as well. Remember that information available on the Internet is generally open to anyone who wants to access it. Individuals wishing to target a specific market may access directories to create customer mailing lists.

The structure or schema of the X.500 Directory must be an initial consideration. Will the hierarchy follow the company structure or is a different approach more practical? How many entries will there be in the directory; 5 or 500,000? A complex hierarchy for thousands of users may affect the efficiency of queries. A study could be done on the contents standard of BD (Bangladesh) Directory Services Tree.

6.3.3 Security Standards of the Network

Securing networked information resources is inherently complex. Attempts must be made to preserve the security of the data. These may include access control lists (ACLs), limiting the number or responses allowed to queries, or internal/external access to the directory.

The subjects included in the directory shall have well defined rights. Company policy, legal restrictions, and the ultimate use of the directory may mandate these. For a basic Internet White Pages Service these rights may include:

1. the option of inclusion in the directory
2. the right of access to the information
3. the right to have inaccurate entries corrected

The terms and conditions for employees of an organization may affect these rights. On becoming an employee of any organization, an individual inevitably agrees to forego certain personal privacies and to accept restrictions.

The decentralized nature of the X.500 Directory service means that each organization has complete control over the data. As part of a global service however, it is important that the operation of the DSA (Directory Services Agent) be monitored and maintained in a consistent manner. Authorisation must be given to the local committee/administrator to manage the local part of the Directory Services more effectively.

Security needs to be implemented at workstation, server, and network level. When information passes from one network to another network that needs to be encrypted to protect sensitive information from hacking. Depending on the confidentiality different encryption technology needs to be implemented into different network segments. A detailed study could be done on the security standards of the .bd domain tree.

6.3.4 Data Integrity

Information that needs to be included in the directory may come from various sources. Demographic information may originate from the human resources department. Electronic mail addresses may be provided by the computer network department. To guarantee data integrity, it is advised that the data-need to be identified and maintained as corporate information.

The required timeliness of the data-need is unique for each DSA (Directory Services Agent). Updates to the data must be provided on a regular basis, it may be once a day or once a month. In cases where data is time sensitive, an attribute should be included to display the most recent maintenance date.

A regular check for data accuracy should be included in the directory administration. Faulty information may put an organization in breach of any data protection laws and possibly render the company as unreliable. A detailed study could be done on the data integrity of Directory Services, and this could be implemented in .bd Domain Tree services tree.

6.3.5 Country-wide Distributed National Database

NDS can be extremely useful for different services of a country if it operates as designed. It may serve as the "hub" of the information routing and the basis for several everyday activities. A NDS based national database could be developed, which could be used for Social Security ID, Health Care, Insurance, Banking, National Voting, etc. services. Presently, we do not have a national distributed database, and if we want to provide the above services to the nation, further research work needs to be done. For people to make use of these services consistent and accurate information needs to be provided. Communication infrastructure of the country is one of the most important factors to provide these services to our people.

References

- [1] Weider, C.; Reynolds, J., "Executive Introduction to Directory Services Using the X.500 Protocol", RFC: 1308, 1992 March, 4p.
- [2] Albitz. Paul, Liu. Cricket, "DNS and BIND", Second Edition, p-17, O'Reilly, January 1997.
- [3] Mogul, J.; Postel, J., "Internet Standard Subnetting Procedure", RFC: 950, August 1985.
- [4] Jennings, B., "Building an X.500 Directory Service in the US", RFC:1943, May 1996.
- [5] Novell Inc., "Novell Delivers Directory Enabled IP Management and Proposes Standard to DMTF, IETF", Novell® Research, July 1998.
- [6] A Novell White Paper, "Global Network Services: Novell's Strategy for Enabling a Smart Global Network", Novell Research, 1995.
- [7] Novell Developer Net, NDS enabled application development www.developer.novell.com.
- [8] Shropshire, ED, "Developing NDS-Enabled Applications", Novell Research, Developer Alliance Division, 1997 December.
- [9] Novell Inc., "NetVision: Using NDS as a Metadirectory to Synchronize Different Directories", Novell® Research, June 1998 www.netvisn.com.
- [10] Laube, Sheldon, "Novell's Smart Global Network and the Future of Computing", Novell Research, December 1995.
- [11] Jennings, B., "Building an X.500 Directory Service in the US", RFC: 1943, May 1996.
- [12] Weider, C.; Reynolds, J.; Heker, S., "Technical Overview of Directory Services Using the X.500 Protocol", RFC: 1309, March 1992.
- [13] Novell Inc., "Mission Data Systems: Using an NDS "Smart Door" for Physical Security", Novell® Research, May 1998 www.missiondata.co.uk.
- [14] Yeong, W.; Howes, T.; Kille, S., "Lightweight Directory Access Protocol", RFC: 1777; March 1995.
- [15] CCITT Blue Book, Volume VIII - Fascicle VIII - Rec. X.509, November 1988.

- [16] Marina del Rey, "Internet Protocol Darpa Internet Program Protocol Specification", Information Sciences Institute, University of Southern California, RFC: 791, September 1981.
- [17] Postel, J., "Assigned Numbers", Information Sciences Institute, University of Southern California, RFC: 790, September 1981.
- [18] Postel, J., "Internet Control Message Protocol -DARPA Internet Program Protocol Specification", RFC: 792, September 1981.
- [19] Plummer ,David C., "An Ethernet Address Resolution Protocol", RFC: 826, November 1982.
- [20] Postel, J., "Multi-LAN Address Resolution", USC/Information Sciences Institute, RFC: 925, October 1984.
- [21] Postel, J., "User Datagram Protocol", USC/Information Sciences Institute, RFC: 768, August 1980.
- [22] Postel, J., "Transmission Control Protocol", USC/Information Sciences Institute, RFC: 793, September 1981.
- [23] Smoot Carl-Mitchell, and John S. Quarterman " Practical Internetworking with TCP/IP and UNIX", Addison-Wesley Publishing Company, 1993, pp 51-78, 109-128.
- [24] Drew Heywood " Novell's Guide to Integrating NetWare and TCP/IP",Novell Press,San Jose, CA95131, June 1996, 1st ed., pp 107-120, 173-184.
- [25] Mockapetris, P.,"Domain Names - Concepts and Facilities", USC/Information Sciences Institute, RFC: 1034, November 1987.
- [26] Jeffrey F. Hughes, "Understanding and Using NDS Objects ", Novell Consulting Services, Novell® Research, January 1995.
- [27] Lottor, M. K., "Domain Administrators Operations Guide", USC/Information Sciences Institute, RFC: 1033, November 1987.
- [28] Stahl, M. K., "Establishing a Domain - Guidelines for Administrators", USC/Information Sciences Institute, RFC:1032, November 1987.
- [29] Mockapetris, P., "Domain Names - Implementation and Specification", USC/Information Sciences Institute, RFC:1035, November 1987.
- [30] Steve Winn, "Controlling The Server Clock under Netware 4", Novell® Research, November/December 1994.
- [31] Partridge, C., "Mail routing and the domain system", RFC: 974, CSNET CIC BBN Labs, January 1986.

- [32] Kunze, Bernd, "Applying X.500 Naming Conventions to NDS", Novell Research, January 1996.
- [33] A Novell White Paper "Global Network Services: Novell's Strategy for Enabling a Smart Global Network", Novell Research, December 1995.
- [34] Wilson, Judy; Williams, John, "NDS Technical Overview", Novell Products Group, Novell® Research, April 1998.
- [35] Crossen, Nancy; Williams, John; Herrin, Selby, "Overview Of NDS Scale", Novell® Research, April 1997.
- [36] Williams, John; Crossen, Nancy; Herrin, Selby, "NDS Technical Overview: Novell Layered Services", Novell® Research, August 1997.
- [37] Omew, Paul Barthol; Neff, Ken, "Easing TCP/IP Network Management with Novell's DNS/DHCP Services", Novell Research, Novell Inc., April 1998.
- [38] Harrenstien, K.; Stahl, M.; Feinler, E., "DoD Internet Host Table Specification", RFC: 952, SRI, October 1985.
- [39] Mockapetris, P., "Domain System Changes and Observations", RFC: 973, USC/Information Sciences Institute, January 1986.
- [40] Shropshire, Ed, "Developing with NDS Using Industry Standards: Java, ActiveX, ODBC, Scripting, and C", Novell® Research, December 1997.
- [41] Novell Inc., "Novell Announces New Web-to-Host Connectivity Solution for Mainframe Access via the Internet". Novell® Research, July 1997.
- [42] Davin, J.; Case, J.; Fedor, M.; Schoffstall, M., "A Simple Gateway Monitoring Protocol", RFC: 1028, November 1987.
- [43] Judy Wilson, "NDS Schema Overview", Novell Products Group, Novell® Research, October 1998.
- [44] Rosen, Eric C., "Exterior Gateway Protocol (EGP)", RFC: 827, October 1982.
- [45] E. Gerich, Merit, "Guidelines for Management of IP Address Space", RFC:1466, May 1993.
- [46] Quarterman, J., and J. Hoskins, "Notable Computer Networks", Communications of the ACM, October 1986, volume 29, number 10.
- [47] Butler Group, "Intranet Technologies", Management Guide Strategies and Technologies, United Kingdom, September 1996, pp 12-23.
- [48] Doug Woodward, "The State of the Infrastructure for Distributed Computing", Novell Research, January 1993.

- [49] Postel, J.; Reynolds, J., "Domain Requirements", RFC: 920 USC/Information Sciences Institute October 1984.
- [50] Burnett, Kevin; Christopher Jenkins, "Extending the NDS Schema: A Beginner's Approach", Novell® Research, June 1997.
- [51] Nancy Crossen; John Williams, Selby Herrin, "Overview of Novell Directory Services", Novell® Research, March 1997.

Appendix A - ISO 3166 Country Codes

The following table lists the ISO 3166 country code list used in NDS object naming.

Country Name	Code
Afghanistan	AF
Albania	AL
Algeria	DZ
Andorra	AD
Angola	AO
Anguilla	AI
Antarctica	AQ
Antigua and Barbuda	AG
Argentina	AR
Armenia	AM
Aruba	AW
Australia	AU
Austria	AT
Azerbaijan	AZ
Bahamas	BS
Bahrain	BH
Bangladesh	BD
Barbados	BB
Belarus	BY
Belgium	BE
Belize	BZ
Benin	BJ
Bermuda	BM
Bhutan	BT
Bolivia	BO
Botswana	BW
Bouvet Island	BV
Brazil	BR
British Indian Ocean Territory	IO
Brunei	BN
Bulgaria	BG
Burkina Faso	BF
Burundi	BI
Cambodia (Kampuchea)	KH
Cameroun	CM
Canada	CA
Cape Verde	CV
Cayman Islands	KY
Central African Republic	CF
Chad	TD

Country Name	Code
Chile	CL
China	CN
Christmas Island	CX
Cocos (Keeling) Islands	CC
Colombia	CO
Comoro Islands	KM
Congo	CG
Cook Islands	CK
Costa Rica	CR
Croatia	HR
Cuba	CU
Cyprus	CY
Czech Republic	CZ
Denmark	DK
Djibouti	DJ
Dominica	DM
Dominican Republic	DO
Ecuador	EC
Egypt	EG
El Salvador	SV
Equatorial Guinea	GQ
Estonia	EE
Ethiopia	ET
Falkland Islands (Malvinas)	FK
Faroe Islands	FO
Fiji	FJ
Finland	FI
France	FR
Gabon	GA
Gambia	GM
Georgia	GE
Germany	DE
Ghana	GH
Gibraltar	GI
Greece	GR
Greenland	GL
Grenada	GD
Guadeloupe	GP
Guam	GU
Guatemala	GT
Guiana (French)	GF
Guinea	GN
Guinea Bissau	GW
Guyana	GY
Haiti	HT
Honduras	HN

Country Name	Code
Hong Kong	HK
Hungary	HU
Iceland	IS
India	IN
Indonesia	ID
Iran	IR
Iraq	IQ
Ireland	IE
Israel	IL
Italy	IT
Ivory Coast	CI
Jamaica	JM
Japan	JP
Johnston Island	JT
Jordan	JO
Kazakhstan	KZ
Kenya	KE
Kiribati	KI
Korea (North)	KP
Korea (South)	KR
Kuwait	KW
Kyrgyzstan	KG
Laos	LA
Latvia	LV
Lebanon	LB
Lesotho	LS
Liberia	LR
Libya	LY
Liechtenstein	LI
Lithuania	LT
Luxembourg	LU
Macao	MO
Madagascar	MG
Malawi	MW
Malaysia	MY
Maldives	MV
Mali	ML
Malta	MT
Marshall Islands	MH
Martinique	MQ
Mauritania	MR
Mauritius	MU
Mexico	MX
Micronesia	FM
Midway Islands	MI
Moldavia	MD

Country Name	Code
Monaco	MC
Mongolia	MN
Montserrat	MS
Morocco	MA
Mozambique	MZ
Myanmar	MM
Namibia	NA
Nauru	NR
Nepal	NP
Netherlands	NL
Netherlands Antilles	AN
New Caledonia	NC
New Zealand	NZ
Nicaragua	NI
Niger	NE
Nigeria	NG
Niue	NU
Norfolk Island	NF
Norway	NO
Oman	OM
Pacific Islands (US)	PC
Pakistan	PK
Panama	PA
Papua New Guinea	PG
Paraguay	PY
Peru	PE
Philippines	PH
Pitcairn Islands	PN
Poland	PL
Polynesia (French)	PF
Portugal	PT
Puerto Rico	PR
Qatar	QA
Reunion	RE
Romania	RO
Russia	RU
Rwanda	RW
Sahara (Western)	EH
Saint Helena	SH
Saint Kitts and Nevis	KN
Saint Lucia	LC
Saint Pierre and Miquelon	PM
Saint Vincent and Grenadines	VC
Samoa (American)	AS
Samoa (Western)	WS
San Marino	SM

Country Name	Code
Sao Tome and Principe	ST
Saudi Arabia	SA
Senegal	SN
Seychelles	SC
Sierra Leone	SL
Singapore	SG
Slovakia	SK
Slovenia	SI
Solomon Islands	SB
Somalia	SO
South Africa	ZA
Spain	ES
Sri Lanka	LK
Sudan	SD
Surinam	SR
Swaziland	SZ
Sweden	SE
Switzerland	CH
Syria	SY
Tadzhikistan	TJ
Taiwan	TW
Tanzania	TZ
Thailand	TH
Timor (East)	TP
Togo	TG
Tokelau	TK
Tonga	TO
Trinidad and Tobago	TT
Tunisia	TN
Turkey	TR
Turkmenistan	TM
Turks and Caicos Islands	TC
Tuvalu	TV
Uganda	UG
Ukraine	UA
United Arab Emirates	AE
United Kingdom	GB
United States of America	US
Uruguay	UY
Uzbekistan	UZ
Vanuatu	VU
Vatican	VA
Venezuela	VE
Vietnam	VN
Virgin Islands (British)	VG
Virgin Islands (US)	VI

Country Name	Code
Wake Island	WK
Wallis and Futuna Islands	WF
Yemen	YE
Yugoslavia	YU
Zaire	ZR
Zambia	ZM
Zimbabwe	ZW

