A

Seminar report

On

Domain Name System

Submitted in partial fulfillment of the requirement for the award of degree of Computer Science

SUBMITTED TO:
www.studymafia.org

SUBMITTED BY:
www.studymafia.org

Acknowledgement

I would like to thank respected Mr…….. and Mr. ……..for giving me such a wonderful opportunity to expand my knowledge for my own branch and giving me guidelines to present a seminar report. It helped me a lot to realize of what we study for.

Secondly, I would like to thank my parents who patiently helped me as i went through my work and helped to modify and eliminate some of the irrelevant or un-necessary stuffs.

Thirdly, I would like to thank my friends who helped me to make my work more organized and well-stacked till the end.

Next, I would thank Microsoft for developing such a wonderful tool like MS Word. It helped my work a lot to remain error-free.

Last but clearly not the least, I would thank The Almighty for giving me strength to complete my report on time.

Preface

I have made this report file on the topic **Domain Name System**; I have tried my best to elucidate all the relevant detail to the topic to be included in the report. While in the beginning I have tried to give a general view about this topic.

My efforts and wholehearted co-corporation of each and everyone has ended on a successful note. I express my sincere gratitude to …………..who assisting me throughout the preparation of this topic. I thank him for providing me the reinforcement, confidence and most importantly the track for the topic whenever I needed it.

CONTENTS

INTRODUCTION

➢ ABSTRACT:

The **Domain Name System** (**DNS**) is a hierarchical naming system for computers, services, or any resource connected to the Internet or a private network. It associates various information with domain names assigned to each of the participants. Most importantly, it translates domain names meaningful to humans into the numerical (binary) identifiers associated with networking equipment for the purpose of locating and addressing these devices worldwide. An often-used analogy to explain the Domain Name System is that it serves as the "phone book" for the Internet by translating human-friendly computer hostnames into IP addresses. For example, www.example.com translates to 192.0.32.10.

The Domain Name System makes it possible to assign domain names to groups of Internet users in a meaningful way, independent of each user's physical location. Because of this, World Wide Web (WWW) hyperlinks and Internet contact information can remain consistent and constant even if the current Internet routing arrangements change or the participant uses a mobile device. Internet domain names are easier to remember than IP addresses such as 208.77.188.166 (IPv4) or 2001:db8:1f70:: 999:de8:7648:6e8 (IPv6). People take advantage of this when they recite meaningful URLs and e-mail addresses without having to know how the machine will actually locate them.

The Domain Name System distributes the responsibility of assigning domain names and mapping those names to IP addresses by designating authoritative name servers for each domain. Authoritative name servers are assigned to be responsible for their particular domains, and in turn can assign other authoritative name servers for their sub-domains. This mechanism has made the DNS distributed and fault tolerant and has helped avoid the need for a single central register to be continually consulted and updated.

In general, the Domain Name System also stores other types of information, such as the list of mail servers that accept email for a given Internet domain. By providing a worldwide, distributed keyword-based redirection service, the Domain Name System is an essential component of the functionality of the Internet.

- **<u>Names versus Addresses</u>**

  - An *address* is how you get to an endpoint
    - Often hierarchical, which helps with scaling
      - 950 Charter Street, Redwood City CA, 94063
      - +1.650.381.6003
      - 204.152.187.11
  - A *name* is how an endpoint is referenced
    - Often with no structurally significant hierarchy
      - "David", "Tokyo", "itu.int","google.com".
  - Names are more people-friendly.

- **<u>An Analogy</u>**

  - Devices on the telephone network all have a number
    - People have a hard time remembering numbers, but…
    - The network needs the numbers to connect endpoints
    - So a directory provides association of names people know with the numbers where they can be reached
  - Computers on the Internet all have a number
    - The DNS takes names people can relate to and converts them into the numbers computers need to interact.
  - This analogy has a crucial flaw: the DNS is not a directory service.
    - There is no way to search the data.

DNS HISTORY

The practice of using a name as a humanly more meaningful abstraction of a host's numerical address on the network dates back to the ARPANET era. Before the DNS was invented in 1983, each computer on the network retrieved a file called *HOSTS.TXT* from a computer at SRI (now SRI International).

The HOSTS.TXT file mapped names to numerical addresses. A hosts file still exists on most modern operating systems, either by default or through explicit configuration. Many operating systems use name resolution logic that allows the administrator to configure selection priorities for available DNS resolution methods.

The rapid growth of the network required a scalable system that recorded a change in a host's address in one place only. Other hosts would learn about the change dynamically through a notification system, thus completing a globally accessible network of all hosts' names and their associated IP addresses.

At the request of Jon Postel, Paul Mockapetris invented the Domain Name System in 1983 and wrote the first implementation. The original specifications appeared in RFC 882 and RFC 883 which were superseded in November 1987 by RFC 1034 and RFC 1035. Several additional Request for Comments have proposed various extensions to the core DNS protocols.

In 1984, four Berkeley students—Douglas Terry, Mark Painter, David Riggle and Songnian Zhou—wrote the first UNIX implementation, which was maintained by Ralph Campbell thereafter. In 1985, Kevin Dunlap of DEC significantly re-wrote the DNS implementation and renamed it BIND—Berkeley Internet Name Domain. Mike Karels, Phil Almquist and Paul Vixie have maintained BIND since then. BIND was ported to the Windows NT platform in the early 1990s.

BIND was widely distributed, especially on Unix systems, and is the dominant DNS software in use on the Internet. With the heavy use and resulting scrutiny of its open-source code, as well as increasingly more sophisticated attack methods, many security flaws were discovered in BIND.

This contributed to the development of a number of alternative nameserver and resolver programs. BIND itself was re-written from scratch in version 9, which has a security record comparable to other modern Internet software.

The DNS protocol was developed and defined in the early 1980s and published by the Internet Engineering Task Force.

NEED OF DNS:

The Internet Protocol address is a 32- bit integer. If somebody wants to send a message it is necessary to include the destination address, but people prefer to assign machines pronounceable, easily remembered names (host names). For this reason the Domain Name System is used. These logical names also allow independence from knowing the physical location of a host.

A host may be moved to a different network, while the users continue to use the same logical name. When Internet was small, mapping was done using a host file. Today, it is impossible to have one single host file relate every address to a name. The host file would be too large to store in every host. Also, it would be impossible to update all the host files in the world every time there is a change.

One solution would be to store the entire host file in a single computer and allow access to this centralized information to every computer that needs a mapping. But this would create a huge amount of traffic on the Internet.

Another solution, the one used today, is to divide this huge amount of information into smaller parts and store each part on a different computer. In this method, the host that needs mapping can contact the closest computer holding he needed information. This method is used by the Domain Name System.

NAME SPACE

To be unambiguous, the names assigned to machines must be carefully selected from a name space with complete control over the binding between the names and IP address. The names must be unique because the addresses are unique. A name space that maps each address to a unique name can be organized in two ways: flat or hierarchical.

FLAT NAME SPACE

In a flat name space, a name is assigned to an address. A name in this space is a sequence of characters without structure. The main disadvantage of a flat name space is that it cannot be used in a large system such as Internet because it must be centrally controlled to avoid ambiguity and duplication.

HIERARCHICAL NAME SPACE

In a hierarchical name space, each name is made of several parts. The first part can define the organization, the second part can define the name, the third part can define departments, and so on. In this case, the authority to assign and control the name spaces can be decentralized. A central authority can assign the part of the name that defines the nature of the organization and the name.

The responsibility for the rest of the name can be given to the organization itself. Suffixes can be added to the name to define host or resources. The management of the organization need not worry that the prefix chosen for a host is taken by another organization because even if part of an address is the same, the whole address is different. The names are unique without the need to be assigned by a central authority. The central authority controls only part of the name, not the whole name.

DOMAIN NAME SPACE

To have a hierarchical name space, a domain name space was designed. In this design, the names are defined in an inverted-tree structure with the root at the top. Each node in the tree has a label, which is a string with a maximum of 63 characters. The root label is a null string (empty string). DNS requires that children of a node(nodes that branch from the same node)have different labels, which guarantees the uniqueness of the domain names.

DOMAIN NAME

Each node in the tree has a domain name. A fully domain name is a sequence of labels separated by dots (.). The domain names are always read from the node up to the root. The last label is the label of the root (null). This means that a full domain name always ends in a null label, which means the last character is a dot because the null string is nothing.

FULLY QUALIFIED DOMAIN NAME (FQDN)

If a label is terminated by a null string, it is called a fully qualified domain name. An FQDN is a domain name that contains the full name of a host.

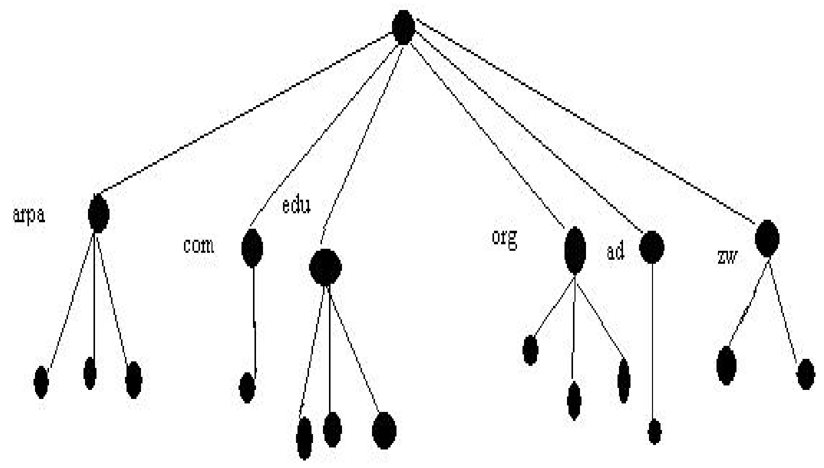
PARTIALLY QUALIFIED DOMAIN NAME (PDQN)

If a label is not terminated by a null string, it is called partially qualified domain name. A PQDN starts from a node, but it does not reach the root.

DOMAIN

A domain is a sub-tree of the domain name space. The name of the domain is the domain name of the node at the top of the sub-tree.

Fig.1



Domain Name Space

DISTRIBUTION OF NAME SPACE

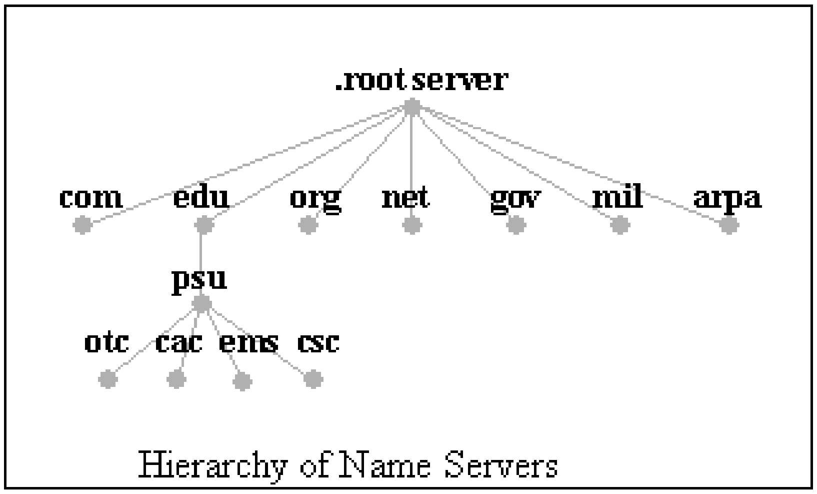
The information contained in the domain name space must be stored. But it is very inefficient and also not reliable to have just one computer store such a huge amount of information. It is inefficient because responding to requests from all over the world places a heavy load on the system. It is not reliable because any failure makes the data inaccessible.

The solution to these problems is to distribute the information among many computers called DNS servers.

HIERARCHY OF NAME SERVERS

The way to distribute information among DNS servers is to divide the whole space into many domains based on the first level. Let the root stand-alone and create as many domains as there are first level nodes. Because a domain created this way could be very large, DNS allows domains to be divided further into smaller domains. Thus we have a hierarchy of servers in the same way that we have a hierarchy of names.

Fig.2



ZONE

What a server is responsible for, or has authority over, is called a zone. The server makes a database called a zone file and keeps all the information for every node under that domain.

If a server accepts responsibility for a domain and does not divide the domains into smaller domains, the domain and zone refer to the same thing. But if a server divides its domain into sub domains and delegates parts of its authority to other servers, domain and zone refer to different things.

The information about the nodes in the sub domains is stored in the servers at the lower levels, with the original server keeping some sort of references to these lower level servers. Of course the original server does not free itself from responsibility totally: It still has a zone, but the detailed information is kept by the lower level servers.

RESOLUTION

Mapping a name to an address or an address to a name is called name address resolution.

RECURSIVE RESOLUTION

The client can ask for a recursive answer from a name server. This means that the resolver expects the server to supply the final answer. If the server is the authority for the domain name, it checks its database and responds. If the server is not the authority, it sends the request to another server and waits for the response. If the parent is the authority, it responds; otherwise, it sends the query to yet another server. When the query is finally resolved, the response travels back until it finally reaches the requesting client. This is recursive resolution.

ITERATIVE RESOLUTION

If the client does not ask for a recursive answer, the mapping can be done iteratively. If the server is an authority for the name, it sends the answer. If it is not, it returns the IP address of the server that it thinks can resolve the query. The client is responsible for repeating the query to this second server. If the newly addressed server can resolve the problem, it answers the query with the IP address; otherwise, it returns the IP address of a new server to the client. Now the client must repeat the query to the third server. This process is called Iterative Resolution because the client repeats the same query to multiple servers.

DNS FEATURES

I. DNS is a Database:

- Keys to the database are "domain names"
  - www.foo.com, 18.in-addr.arpa, 6.4.e164.arpa
- Over 100,000,000 domain names are now stored.
- Each domain name contains one or more attributes, known as *resource records.*
  - Each attribute is individually retrievable.

II. Global Distribution:

- Data is maintained locally, but retrievable globally
- No single computer has all DNS data
- DNS lookups can be performed by any Internet-connected device
- Remote DNS data is locally cacheable to improve performance

III. Loose Coherency:

- The database is always internally consistent
  - Each version of a subset of the database (a zone) has a serial number
  - The serial number is incremented on each database change
- Changes to the master copy of the database are replicated according to timing set by the zone administrator
- Cached data expires according to timeout set by zone administrator.

IV. Scalability:

- No intrinsic limit to the size of the database
- Some servers have over 20,000,000 names
- Not a particularly good idea

- No limit to the number of queries
- 80,000 queries per second handled regularly
- Queries distributed among many different servers

V. <u>Reliability:</u>

- Data is replicated
  - Data from master source is copied to multiple slave servers
  - Clients can query master server or slave servers
- DNS protocols can use either UDP or TCP
  - UDP is inherently unreliable, but the DNS protocol handles retransmission (perhaps with TCP), sequencing, *et cetera*.

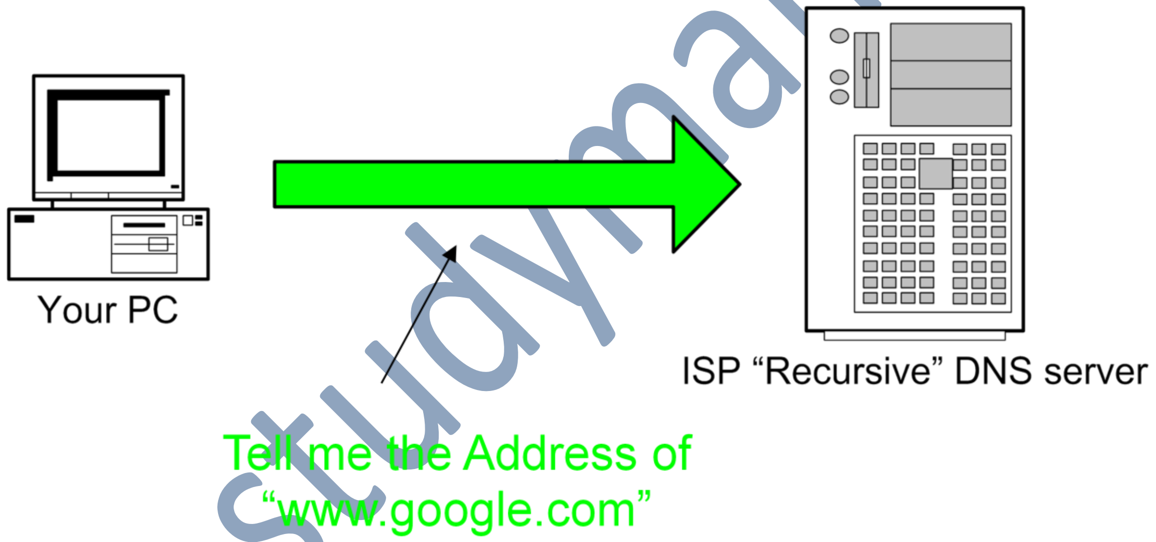
VI. <u>Dynamic Updates:</u>

- Database can be updated dynamically
  - Master server accepts update from over the network
  - Add/delete/modify any record
- Modification of the master database triggers replication
  - Only master can be dynamically updated
  - Dynamic updates create a single point of failure

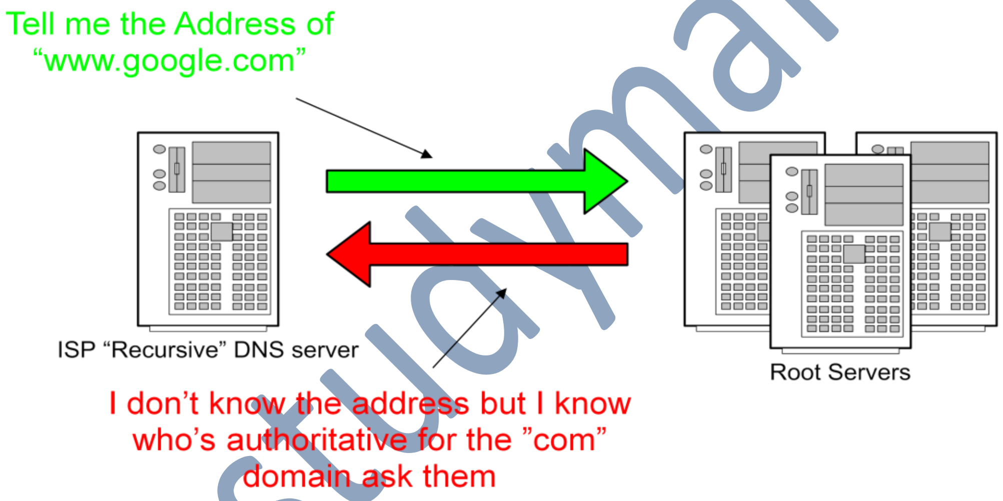
Accessing a web page

- When You type http://www.google.com into your web browser and hit enter.
- What happens now?

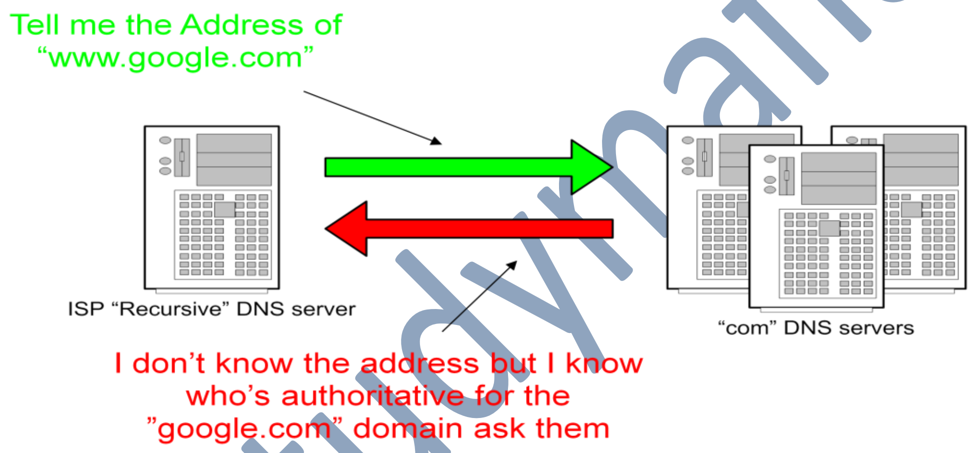
Step 1: Your PC sends a resolution request to its configured DNS Server, typically at your ISP.


Your PC
ISP “Recursive” DNS server
Tell me the Address of “www.google.com”

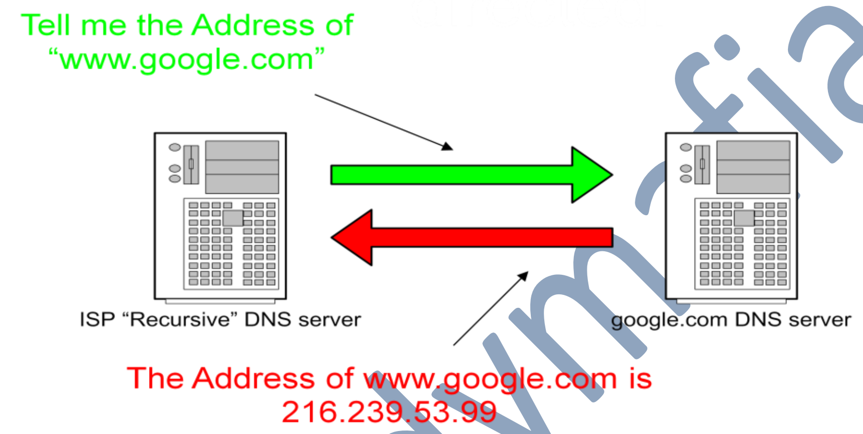
Step 2: Your ISPs recursive name server starts by asking one of the root servers predefined in its "hints" file.


Tell me the Address of "www.google.com"
ISP "Recursive" DNS server
Root Servers
I don't know the address but I know who's authoritative for the "com" domain ask them

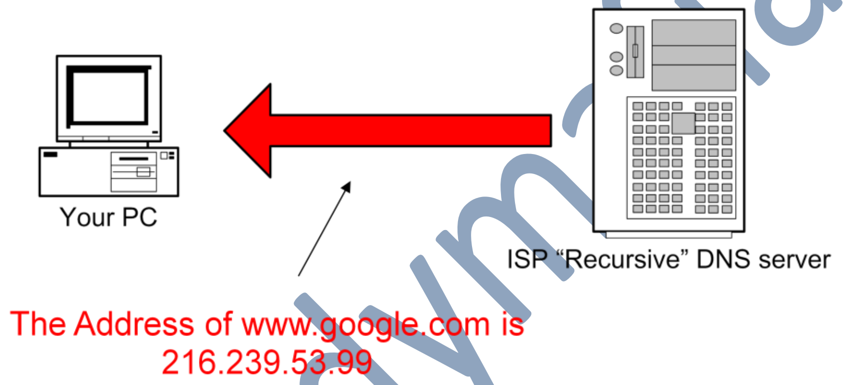
<u>Step 3: Your ISPs recursive name server then asks one of the “com” name servers as directed.</u>


Tell me the Address of “www.google.com”
ISP “Recursive” DNS server
“com” DNS servers
I don’t know the address but I know who’s authoritative for the ”google.com” domain ask them

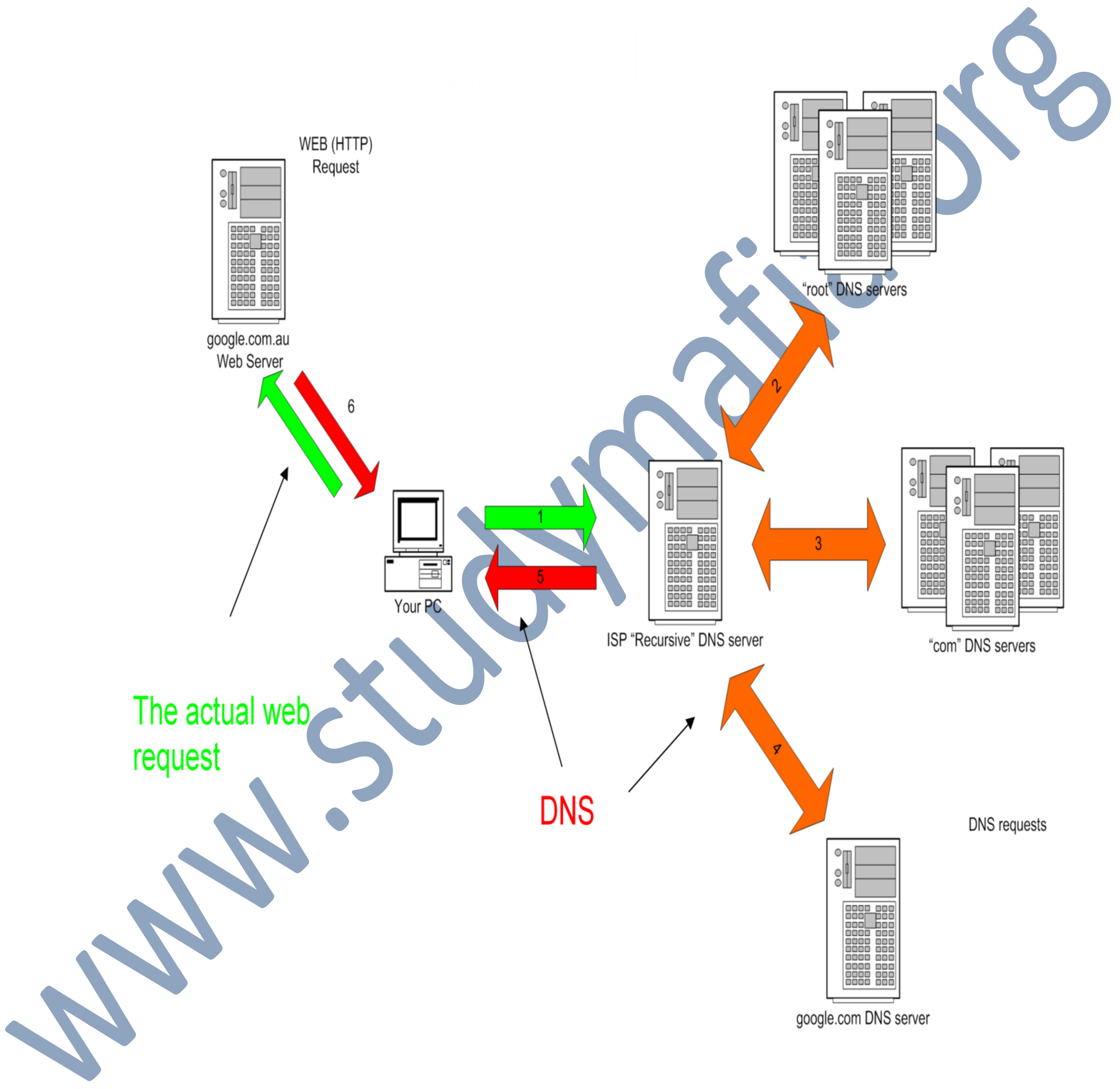
Step 4: Your ISPs recursive name server then asks one of the "google.com" name servers as directed.


Tell me the Address of "www.google.com"
ISP "Recursive" DNS server
google.com DNS server
The Address of www.google.com is 216.239.53.99

Step 5: ISP DNS server then send the answer back to your PC. The DNS server will "remember" the answer for a period of time.


Your PC
ISP "Recursive" DNS server
The Address of www.google.com is 216.239.53.99

ALL STEPS IN ONE:



CONCLUSION

This thesis presents the client redirection problem and the requirements that should be met by a good solution. These requirements include: transparency, scalability, maintainability, and efficiency. Among three popular redirection mechanisms (HTTP redirection, TCP handoff, and DNS redirection), we conclude that a DNS-based redirection mechanism meets the four above requirements best.

Our implementation, called NetAirt, proves that with little effort Apache can be made to service DNS packets, no matter which transport-layer protocol is used: UDP or TCP. Moreover, our module can easily be integrated with an external replication-managing component, designed either as a separate Apache module (like Globule, another project of ours), or as a completely independent program.

We have paid attention to the development of a reasonable set of redirection policies: static name-to-address mapping, round-robin replica selection, and the AS-path length policy. We have incorporated them into NetAirt, thus demonstrating its flexibility, and allowing users to switch between them.

We have described in detail how the most advanced one, the AS-path length policy, can be implemented. Firstly, BGP routing tables can be exploited to associate an ASN with any IP address, as well as to build a map of the ASes. This map allows to define the AS-path length metric, which is the base of the AS-path length redirection policy.

We have also conveyed two experiments. The first of them investigates the following three issues: the impact of the transport protocol on the Round Trip Time (RTT) of a DNS packet, the overhead introduced by running the round-robin address selection instead of the static name-to-address mapping, and the correspondence between the number of IP addresses returned in a single DNS response and the RTT. The results prove that sending DNS queries over TCP is almost 80% slower than in case of UDP. On the other hand, the overhead introduced by the round-robin policy, and the cost of responding with additional IP addresses turn out to be negligible.

The second experiment evaluates the overhead due to the AS-path length policy. The analysis shows that the majority of Autonomous Systems is located at distance from 3 to 5 from a typical client. We have also measured the time overhead generated by the AS-path length policy when it searches for a closest replica.

The results show that exploring the entire map of the Internet that we use for this purpose takes no longer than 3.1 milliseconds, and on average only 0.64 milliseconds. We find this time small compared to the overall RTT for a typical DNS request sent over a wide-area network. Thus, considering the beneficial impact of running the AS-path length policy, we believe that redirecting clients based on network distance calculations can be nearly as efficient as using other, less sophisticated policies. NetAirt will soon be released for public use. We hope that it can contribute to the development of worldwide-distributed services, and thus help sustaining the continuous growth of the Internet.

REFERENCES

- http://en.wikipedia.org/wiki/Category:Domain_name_system
- http://www.livinginternet.com/i/iw_dns.htm
- http://www.centr.org
- Domain Names - Concepts and Facilities, P. Mockapetris
- Role of the Domain Name System (DNS)-O'Reilly
- www.wikipedia.com
- www.studymafia.org