3: Application Protocols: HTTP and DNS

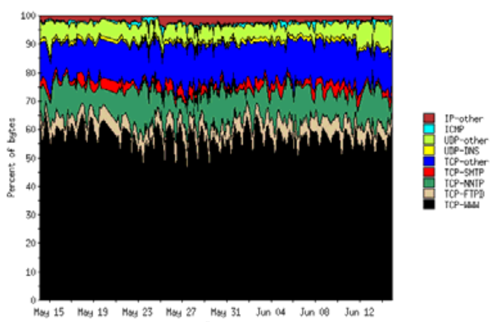
Last Modified:
2/3/2003 8:13:18 PM

Network Applications Drive Network Design

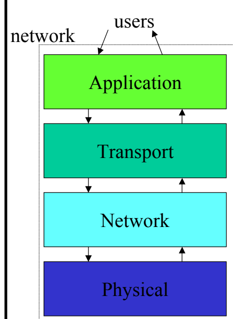
- Important to remember that network applications are the reason we care about building a network infrastructure
- Applications range from text based command line ones popular in the 1980s (like telnet, ftp, news, chat, etc) to multimedia applications (Web browsers, audio and video streaming, real-time video conferencing, etc.)

What is the Internet used for?

Credit: CAIDA (1999)



Top-down: Internet protocol stack



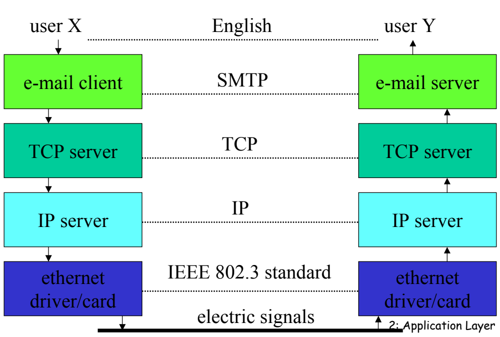
HTTP, SMTP, FTP, TELNET, DNS, …

TCP, UDP.

IP

Point-to-point links, LANs, radios, ...

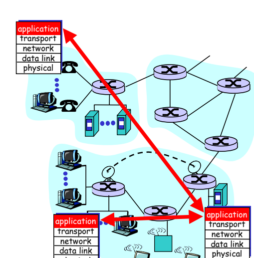
Protocol stack



Applications and application-layer protocols

Application: communicating, distributed processes
- running in network hosts in "user space"
- exchange messages to implement app
- e.g., email, file transfer, the Web

Application-layer protocols
- one "piece" of an app (web browser do more than speak HTTP)
- define messages exchanged by apps and actions taken
- user services provided by lower layer protocols



Client-server paradigm

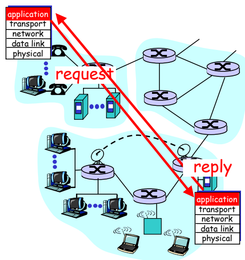
Typical network app has two pieces: *client* and *server*

Client:

- initiates contact with server ("speaks first")
- typically requests service from server,
- for Web, client is implemented in browser; for e-mail, in mail reader

Server:

- Running first (always?)
- provides requested service to client e.g., Web server sends requested Web page, mail server delivers e-mail

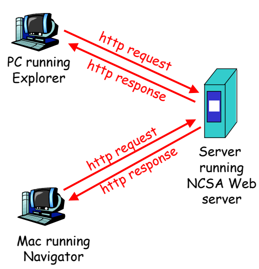


HTTP

The Web: the http protocol

http: hypertext transfer protocol

- Web's application layer protocol
- client/server model
  - *client:* browser that requests, receives, "displays" Web objects
  - *server:* Web server has access to storage containing a set of Web documents; sends copies in response to requests
- http1.0: RFC 1945
- http1.1: RFC 2616
- r (e.g. Java applet)



The http protocol: more

http: TCP transport service:

- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- http messages (application-layer protocol messages) exchanged between browser (http client) and Web server (http server)
- TCP connection closed

http is "stateless"

- server maintains no information about past client requests

aside

Protocols that maintain "state" are complex!

- past history (state) must be maintained
- if server/client crashes, their views of "state" may be inconsistent, must be reconciled

Uniform Resource Locator (URL)

protocol://authority:port/p/a/th/item_name?query

- protocol = http
- authority = server machine
- port = 80 by default
- /p/a/th/item_name = specifies a file to be returned or possibly a program to be executed to produce the file to be returned
- query = data to be interpreted by server

Note: Static vs Dynamic vs Active Web Pages

- Static: Stored in a file and unchanging
- Dynamic: Formed by server on demand in response to a request
  - Output from a program (e.g. Common Gateway Interface (CGI))
  - Often use query data sent with URL
- Active: Executed at the client!
  - Computer program (not just output) that can interact with user (e.g. Java applet)

http example

Suppose user enters URL
www.someSchool.edu/someDepartment/home.index

(contains text, references to 10 jpeg images)

1a. http client initiates TCP connection to http server (process) at www.someSchool.edu. Port 80 is default for http server.

1b. http server at host www.someSchool.edu waiting for TCP connection at port 80. "accepts" connection, notifying client

2. http client sends http *request message* (containing URL) into TCP connection socket

3. http server receives request message, forms *response message* containing requested object (someDepartment/home.index), sends message into socket

time

http example (cont.)

4. http server closes TCP connection.

5. http client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

6. Steps 1-5 repeated for each of 10 jpeg objects

time

http message format: request

- Two types of http messages: *request, response*
- Http request message:
  - ASCII (human-readable format)

request line (GET, POST, HEAD commands)

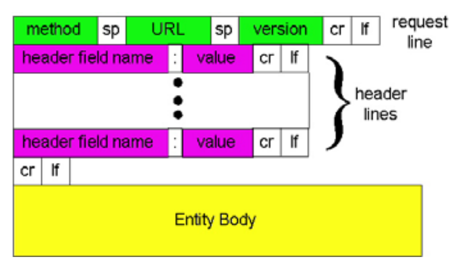
```
GET /somedir/page.html HTTP/1.0
User-agent: Mozilla/4.0
Accept: text/html, image/gif,image/jpeg
Accept-language:fr
```

header lines

(extra carriage return, line feed)

Carriage return, line feed indicates end of message

http request message: general format



http message format: response

status line (protocol status code status phrase)

```
HTTP/1.0 200 OK
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 …...
Content-Length: 6821
Content-Type: text/html

data data data data data ...
```

header lines

data, e.g., requested html file

http response status codes

In first line in server->client response message.
A few sample codes:

200 OK
- request succeeded, requested object later in this message

301 Moved Permanently
- requested object moved, new location specified later in this message (Location:)

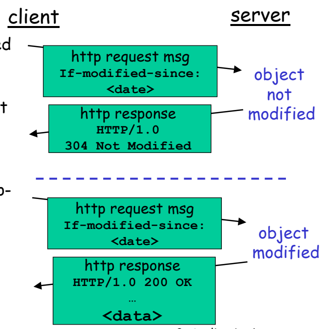
400 Bad Request
- request message not understood by server

404 Not Found
- requested document not found on this server

505 HTTP Version Not Supported

Conditional GET

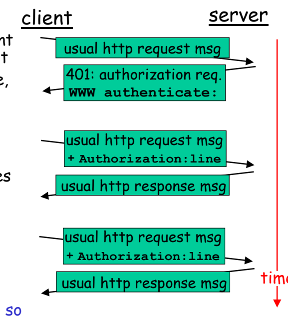
- Goal: don't send object if client has up-to-date stored (cached) version
- client: specify date of cached copy in http request
  `If-modified-since: <date>`
- server: response contains no object if cached copy up-to-date:
  `HTTP/1.0 304 Not Modified`



Authentication (and statelessness)

Authentication goal: control access to server documents

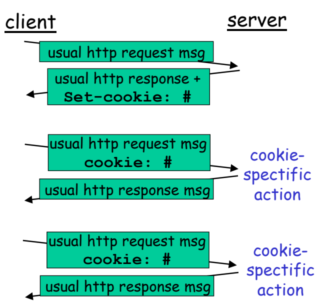
- stateless: client must present authorization in each request
- authorization: typically name, password
  - `authorization:` header line in request
  - if no authorization presented, server refuses access, sends
    `WWW authenticate:`
    header line in response
- Authorization will go with each request to server



Browser caches name & password so that user does not have to repeatedly enter it.

Cookies (and statelessness ?)

- server sends "cookie" to client in response mst
  `Set-cookie:`
- client presents cookie in later requests
  `cookie:`
- server matches presented-cookie with server-stored info
  - authentication
  - remembering user preferences, previous choices
- Get client to remember "state" so server can be stateless!



HTTP 1.1 : Persistent connections

Non-persistent

- HTTP/1.0
- server parses request, responds, and closes TCP connection
- Each object transfer suffers from TCP connection setup overhead
- 2 RTTs to fetch each object

Persistent

- default for HTTP/1.1
- on same TCP connection: server, parses request, responds, parses new request,..
- Client sends requests for all referenced objects as soon as it receives base HTML.
- Fewer RTTs

But most 1.0 browsers use parallel TCP connections. Do 1.1 browsers do this? ☺

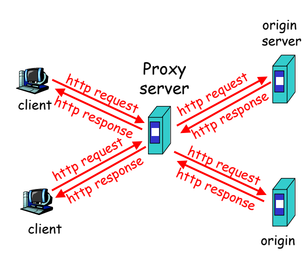
Other Features in HTTP 1.1

- Hostname Identification
  - Allows one physical web server to serve content for multiple logical servers
- Content Negotiation
  - Allows client to request a specific version of a resource
- Chunked Transfers
  - For dynamic content, server needn't specify all characteristics like size ahead of time
- Byte Ranges
  - Clients can ask for small pieces of documents
- Support for Proxies and Caches

Web Caches (proxy server)

Goal: satisfy client request without involving origin server

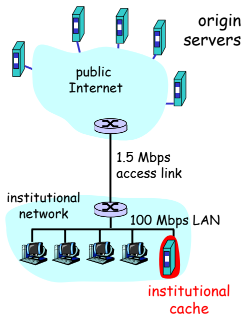
- user sets browser: Web accesses via web cache
- client sends all http requests to web cache
  - if object at web cache, web cache immediately returns object in http response
  - else requests object from origin server, then returns http response to client



Why Web Caching?

Assume: cache is “close” to client (e.g., in same network)

- smaller response time: cache “closer” to client
  - decrease traffic to distant servers
  - link out of institutional/local ISP network often bottleneck
- Other reasons? Anonymity? Translation for low feature clients (ex. PDAs)



Why not web caching?

- It adds time to a requests that miss in the cache
- Servers don’t see accurate number of hits to their content
  - To collect information on who is requesting what, extract fees, etc.

Trying out http (client side) for yourself

1. Telnet to your favorite Web server:

```
telnet www.google.com 80
```

Opens TCP connection to port 80 (default http server port) at www.eurecom.fr. Anything typed in sent to port 80 at www.eurecom.fr

2. Type in a GET http request:

```
GET / HTTP/1.0
```

By typing this in (hit carriage return twice), you send this minimal (but complete) GET request to http server

3. Look at response message sent by http server!

HTTP 1.0 vs 1.1

1. HTTP 1.0

```
telnet www.google.com 80
GET / HTTP/1.0

<send data >
Connection closed by foreign host.
```

2. HTTP 1.1

```
telnet www.google.com 80
GET / HTTP/1.1

<send data>
GET / HTTP/1.1

<send data>
GET / HTTP/1.0

<send data >
Connection closed by foreign host.
```


Experiment yourself

1. Try some headers

```
telnet www.google.com 80
GET / HTTP/1.1
Host: www.google.com
```

2. Try a real query (look at syntax of URL when you use google)
3. Try a chunked transfer
4. ….

For the record: HTTP vs HTML

- HTML format is highly specified but is just considered the data or body of an HTTP message
- HTML is not part of the HTTP protocol
- Example of layering: each layer speaks to a peer layer in an agreed upon language or protocol
- In this case, both are processed by the web browser. The web browser is both an HTTP client and an HTML parser.

DNS

Names and IP addresses

People: many identifiers:
- SSN, name, Passport #

Internet hosts, routers: many identifiers too
- IP address (32 bit) - used for addressing datagrams
- "name", e.g., www.google.org - used by humans

Q: map between IP addresses and name ?
DNS does

..but before we talk about DNS lets talk more about names and addresses!

Names and addresses: why both?

- Name: www.google.com
- IP address: 216.239.57.101
  - (Also Ethernet or other link-layer addresses.)
- IP addresses are fixed-size numbers.
  - 32 bits. 216.239.57.101 = 11011000.11101111.111001.1100101
- Names are memorizable, flexible:
  - Variable-length
  - Many names for a single IP address.
  - Change address doesn't imply change name.
  - iPv6 addresses are 128 bit – even harder to memorize!

Mapping Not 1 to 1

- One name may map to more than one IP address
  - IP addresses are per network interface
  - Multihomed machines have more than one network interface - each with its own IP address
  - Example: routers must be like this
- One IP address may map to more than one name
  - One server machine may be the web server (www.foo,com), mail server (mail.foo.com)etc.

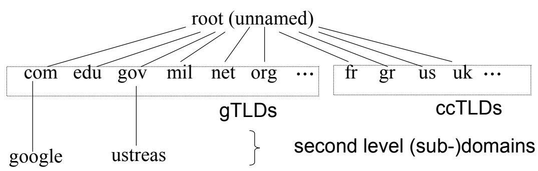
How to get names and numbers?

- Acquistion of Names and numbers are both regulated
  - Why?

How to get a machine name?

- First, get a domain name then you are free to assign sub names in that domain
  - How to get a domain name coming up
- Before you ask for a domain name though
  - Should understand domain name structure…
  - Should also know that you are responsible for providing authoritative DNS server (actually a primary and one or more secondary DNS servers) for that domain and registration information through "whois"

Domain name structure



gTLDs= Generic Top Level Domains
ccTLDs = Country Code Top Level Domains

Top-level Domains (TLDs)

- Generic Top Level Domains (gTLDs)
  - .com - commercial organizations
  - .org - not-for-profit organizations
  - .edu - educational organizations
  - .mil - military organizations
  - .gov - governmental organizations
  - .net - network service providers
  - New: .biz, .info, .name, …
- Country code Top Level Domains (ccTLDs)
  - One for each country

How to get a domain name?

- In 1998, non-profit corporation, Internet Corporation for Assigned Names and Numbers (ICANN), was formed to assume responsibility from the US Government
- ICANN authorizes other companies to register domains in com, org and net and new gTLDs
  - Network Solutions is one of the largest and in transitional period between US Govt and ICANN had sole authority to register domains in com, org and net

Want to be a registrar?

- http://www.icann.org/registrars/accreditation.htm
- Application + $2500 application fee
- Sign agreement
- Demonstrate $70,000 in working capital
- Yearly fee - $4000 for first TLD + $500 for each additional

How to get an IP Address?

- Answer 1: Normally, answer is get an IP address from your upstream provider
  - This is essential to maintain efficient routing!
- Answer 2: If you need lots of IP addresses then you can acquire your own block of them.
  - Get them from a regional Internet registry

Internet Registries

If you want a block of IP addresses, go to an Internet Registry

- RIPE NCC (Riseaux IP Europiens Network Coordination Centre) for Europe, Middle-East, Africa
- APNIC (Asia Pacific Network Information Centre)for Asia and Pacific
- ARIN (American Registry for Internet Numbers) for North America, the Caribbean, sub-equatorial Africa
- LACNIC – Latin American and Caribbean Registry (new 10/2002)
- **Note: Once again regional distribution is important for efficient routing!**

Can also get Autonomous System Numbers (ASNs from these registries

Obtaining a Block of IP addresses

- Price (ARIN,Jan 2003)
  - http://www.arin.net/registration/fee_schedule.html
  - $2500/year for /20 ; $20000/year for a /14
  - /20 = 20 of the 32 bits in IP address are specified, 12 bits free, ~2^{12}= 4096 possible hosts
  - See why a /14 would be more expensive than a /20?
- Can't just pay and not use them
  - IP address space is a scarce resource
  - You must prove you have fully utilized a small block before can ask for a larger one!

Checkpoint

- Now you know both how to get a machine name and how to get an IP address
- Now back to DNS – how to map from one to the other!

Mapping from name to IP Address?

How could we provide this service?

- In the beginning, file containing mapping for all hosts copied to each new host
  - Size of file?
  - Propagation of changes?
- Centralized DNS server?
  - single point of failure
  - traffic volume
  - distant centralized database
  - maintenance

doesn't *scale!*

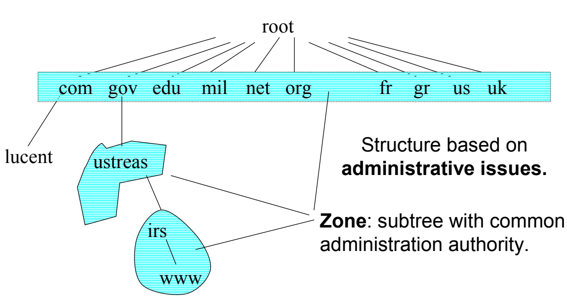
- no server has all name-to-IP address mappings

DNS: Domain Name System

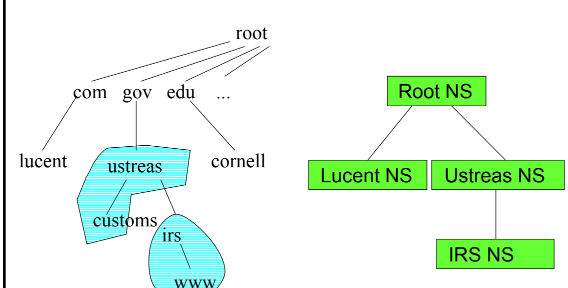
Domain Name System:

- *distributed database* implemented in hierarchy of many *name servers*
- *application-layer protocol* host, routers, name servers to communicate to *resolve* names (address/name translation)
  - note: core Internet function implemented as application-layer protocol
  - complexity at network's "edge"

Name Server Zone Structure



Mapping Name Servers to "Zones"



Kinds of Name Servers

Name server: process running on a host that processes DNS requests

- local name servers:
  - each ISP, company has *local (default) name server*
  - host DNS query first goes to local name server
- authoritative name server:
  - can perform name/address translation for a specific domain or zone
- root name server:
  - Knows the authoritative server for each domain
- intermediate name server:
  - Authoritative servers for a large domain may hand off queries to lower level name servers that are responsible for a portion of the domain

Local Name Servers

- Each host knows the IP address of a local NS.
- Each local NS knows the IP addresses of all root NSs.

Authoritative Name Servers

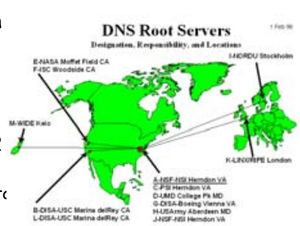
- Authoritative name servers for a given domain do not "cache" the translation instead they are the official source for translating all machine names in that domain
- For each domain, there must be an authoritative name server
  - In fact, must be at least two- a primary and secondary

Root Name Servers

- How do local name servers find the authoritative NS for a given domain?
- Local name servers contact root name servers for the address of the authoritative name server for a domain

Root name servers

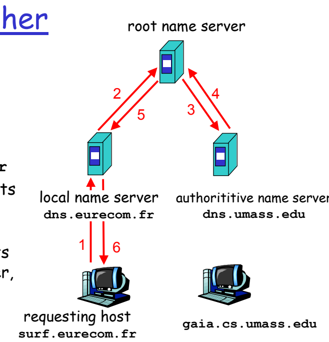
- ~10 root name servers in the Internet
  - A. ROOT-SERVERS.NET
  - B.ROOT-SERVERS.NET
  - …
- Most in US, 1 in Japan, 2 in Europe
  - http://netmon.grnet.gr/stc thost/rootns/
  - ftp://rs,internic.net/domai n/named.cache
- RFC 2870: Root Name Server Operational Requirements



Putting it together

host **surf.eurecom.fr** wants IP address of **gaia.cs.umass.edu**

1. Contacts its local DNS server, **dns.eurecom.fr**
2. **dns.eurecom.fr** contacts root name server, if necessary
3. root name server contacts authoritative name server, **dns.umass.edu**, if necessary



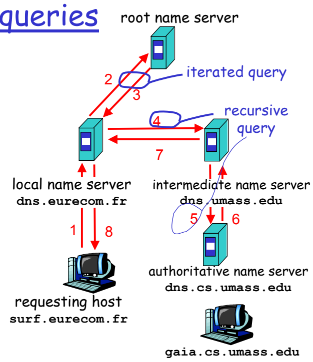
What is wrong with this picture?

DNS: iterated queries

recursive query:
- Contacted server completes translation itself
- Puts burden on contacted server

iterated query:
- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"
- Takes burden off contacted servers

Root servers disable recursive queries!

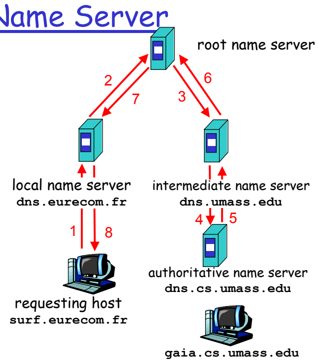


Intermediate Name Servers

- What about big domains? Couldn't the authoritative name servers for a big domain get overloaded like the root? Or maybe it is inconvenient administratively for two sub domains to share the same DNS server?
- We don't want the root to have to remember different servers for sub domains.
- Give the root the name of an "intermediate name server"
  - They aren't really the authority for each sub domain but they can point you to the authority!

Intermediate Name Server

- Root name server may not know the real authoritative name server
- may know *intermediate name server:* who to contact to find authoritative name server



DNS - Point of Failure

- How often are failures a result of DNS failure?
  - Make notes of IP addresses of common machines you use
  - If can't access, try instead accessing by IP address
  - If you can -> DNS failure somewhere

DNS UPDATE

- DNS designed for fairly slow/infrequent change to these mappings
  - Changes made via external edits to a zone's Master File
  - Faster more automatic update/notify mechanisms under design by IETF
  - Proposed Standard: RFC 2136
- Example: home machines that get a new IP address all the time - can update the translation of human readable name to that new IP address; DHCP in general
- Once a non-authoritative name server learns a mapping, it *caches* the mapping
  - cache entries timeout (disappear) after some time
  - What it change faster than cache entries time out?

DNS records: More than Name to IP Address

DNS: distributed db storing resource records (RR)

RR format: `(name, value, type,ttl)`

- Type=A
  - One we've been discussing
  - Maps name to IP address
  - `name` is hostname
  - `value` is IP address
- Other common ones? NS, MX, CNAME, PTR
- Lots more: SOA, HINFO, MB, MR, MG, WKS, RB

DNS records: More than Name to IP Address

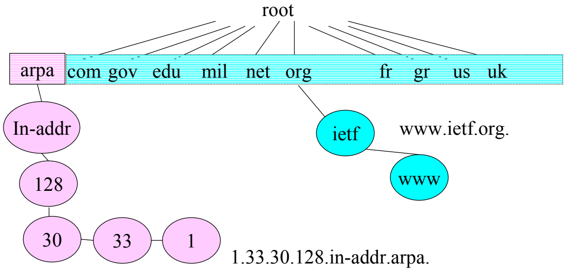
- Type=NS
  - **name** is domain (e.g. foo.com)
  - **value** is IP address of authoritative name server for this domain (why not name?)
- Type=CNAME
  - **name** is an alias name for some "cannonical" (the real) name
  - **value** is cannonical name
- Type=MX
  - **value** is hostname of mailserver associated with **name**
- Type=PTR
  - **name** is IP address (in special format)
  - **value** is name
  - Reverse of type A

PTR Records

- Do reverse mapping from IP address to name
- Why is that hard? Which name server is responsible for that mapping? How do you find them?
- Answer: special root domain, arpa, for reverse lookups

Arpa top level domain

Want to know machine name for 128.30.33.1?
Issue a PTR request for 1.33.30.128.in-addr.arpa



Why is it backwards?

- Notice that 1.33.30.128.in-addr.arpa is written in order of increasing scope of authority just like www.irs.gov
- From largest scope of authority, gov, up to single machine www.irs.gov
- From largest scope of activity, arpa, up to single machine 1.33.30.128.in-addr.arpa (or 128.30.33.1)
- nslookup –query=any 1.33.30.128.in-addr.arpa ??

In-addr.arpa domain

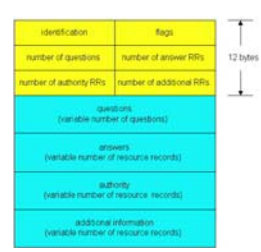
- When an organization acquires a domain name, they receive authority over the corresponding part of the domain name space.
- When an organization acquires a block of IP address space, they receive authority over the corresponding part of the in-addr.arpa space.
- Example: Acquire domain berkeley.edu and acquire a class B IP Network ID 128.143

DNS protocol, messages

DNS protocol : *query* and *repy* messages, both with same *message format*

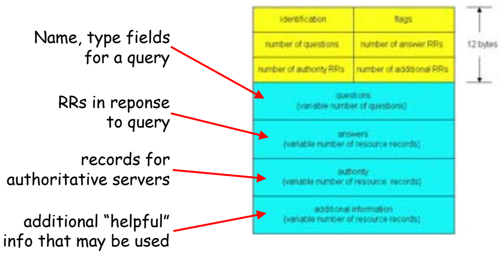
msg header

- identification: 16 bit # for query, repy to query uses same #
- flags:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative
  - reply was truncated



Sample query and response?

DNS protocol, messages



UDP or TCP

- DNS usually uses UDP
- Doesn't DNS need error control? Why is UDP usually ok?
  - Each object small enough to go in one datagram – no need for reorder
  - Retransmission? Just instrument client to resend request if doesn't get a response
- When does DNS use TCP?
  - Truncation bit; if reply too long, set truncate bit as signal to request using TCP
  - Also for zone transfers from primary to secondary servers (RFC still says try UDP first)
- BIND can be configured to only respond to a TCP request if a corresponding UDP request was made first

Why not always TCP?

- TCP has higher overhead
  - 2 Round Trips per query rather than 1
  - Many apps that use UDP implement only the subset of TCP functionality they really need
- Also UDP requires less state on server
  - With TCP, each connection requires significant state
  - More prone to overload (denial of service attacks?)

HTTP vs DNS

- Why is HTTP human readable and DNS not?
  - Saves space is the limited size of the query/response packet
  - HTTP used by an application focused on end users; DNS used by an application focused on network management?
  - Better answer??

nslookup

- Use to query DNS servers (not telnet like with http – why?)
- Interactive and Non-interactive modes
- Examples:
  - nslookup www.yahoo.com
    - Many IP addresses why?
  - nslookup –query=mx gnu.org
  - nslookup
    - Enter interactive shell
    - Type a host name; get its IP address info
    - ls –d <domain.name> (rarely supported)
    - set debug, set recurse, set norecurse,…
    - exit

Summary

- We looked at two application level protocols: HTTP and DNS
- HTTP runs on TCP
- DNS usually runs on UDP (sometimes on TCP)
- HTTP is human readable; DNS not

Outtakes

Other

- DNS forwarding
  - Way to say if don't find it here look here instead
  - Examples
    - I used to be authoritative for this – now I'm not look here
    - Also useful for reverse lookups when organizations don't have a full class A/B/C address – say where else to look for possible reverse name lookup
    - Internal DNS server behind firewall and has full translations within domain; External has publicly visible like web and mail servers; Internal is firewalled off so forwards request for outside world to external that queries the root servers etc

Other

- Need to use TCP for DNS through firewalls?
- Common DDOS attack on DNS is to send TCP requests to a large array of servers around the world for some zone that they are not authoritative for. In turn,all those servers then go and make a large number of TCP requests to that zone's authoritative server at once.

DNS Notify

- Used by a master server to inform the slave servers that they should ask for an update. Zone Transfers are typically limited to only allow the slave servers to receive that zone. For that reason, using the "ls" feature in nslookup almost never works.

HTML overview

- Markup language give general layout guidelines - not exact placement or format- so browsers may display the same document differently
- Free form (i.e. Spaces don't matter)
- Embedded tags give guidelines
- Tags often appear in pairs
  - beginning <TAGNAME>
  - ending </TAGNAME>

How do clients and servers communicate?

API: application programming interface
- defines interface between application and transport layer
- socket: Internet API
  - two processes communicate by sending data into socket, reading data out of socket

Q: how does a process "identify" the other process with which it wants to communicate?
- IP address of host running other process
- "port number" - allows receiving host to determine to which local process the message should be delivered

… more on this later.

Sockets Specify Transport Services

- Sockets define the interfaces between an application and the transport layer
- Applications choose the type of transport layer by choosing the type of socket
  - UDP Sockets – called DatagramSocket in Java, SOCK_DGRAM in C
  - TCP Sockets – called Socket/ServerSocket in Java, SOCK_STREAM in C
- Client and server agree on the type of socket, the server port number and the protocol

QUICK LOOK AHEAD: TCP vs UDP

TCP service:

- *connection-oriented:* setup required between client, server
- *reliable transport* between sending and receiving process
- *flow control:* sender won't overwhelm receiver
- *congestion control:* throttle sender when nework overloaded
- *does not providing:* timing, minimum bandwidth guarantees

UDP service:

- unreliable data transfer between sending and receiving process
- does not provide: connection setup, reliability, flow control, congestion control, timing, or bandwidth guarantee