Applications: DNS, HTTP and the WWW

EECS 122: Lecture 6

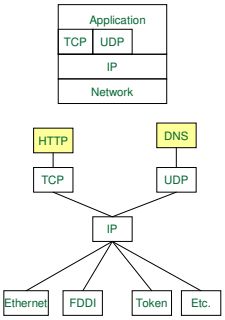
Department of Electrical Engineering and Computer Sciences
University of California
Berkeley

What we've covered so far..

- Basic Background
  - General Overview of different kinds of networks
  - General Design Principles
    - Architecture
    - Performance
  - How to write a network application
- Now let's get into how things really work!

Applications

- Must separate the application processing from the application protocols
- Example:
  - WWW Browser & Server
  - HTTP
- Also, applications can be run on the end hosts or inside the network cloud
  - WWW on end hosts
  - DNS in the network cloud



Today

- DNS
- WWW
  - HTTP
- Both
  - are client – server applications
  - have decentralized management
  - enable access to vast amounts of distributed information
  - are based on open protocols
  - are distributed databases

Domain Name Service

- Resolves a host name names into an IP address
- Why host names?
  - To organize machines
    - Eg. robotics.eecs.berkeley.edu
    - This conveys more information to humans than 128.32.48.234
- Why IP addresss?
  - The network needs an address to route
- Host Names yield information to people and IP addresses yield information to routers

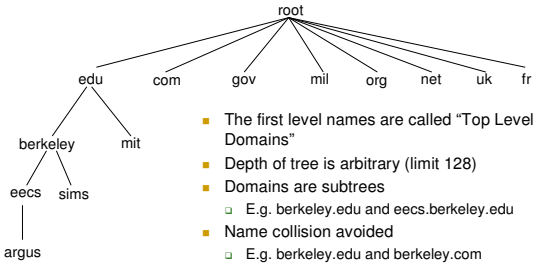
DNS: History

- Initially all host-addess mappings were in a file called hosts.txt (in /etc/hosts)
  - Changes were submitted to SRI by email
  - New versions of hosts.txt were ftp'd periodically from SRI
  - An administrator could pick names at their discretion
- As the internet grew this system broke down because
  - SRI couldn't handled the load
  - The system was unreliable since there was a single point of contact
  - Names were not unique
  - Many hosts had inaccurate copies of hosts.txt
- Internet growth was threatened!

DNS Features

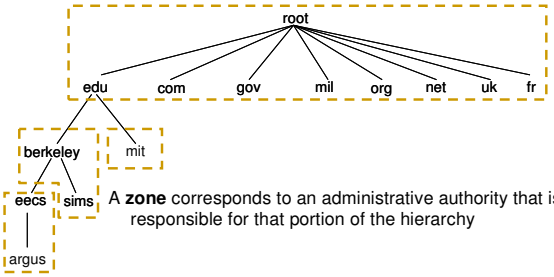
- Hierarchical Namespace
- Distributed architecture for storing names
  - Nameservers assigned zones of the hierarchical namespace
  - Backup servers available for redundancy
- Administration divided along the same hierarchy
  - DNS client is simple: Resolver
- Client server interaction on UDP Port 53 (but can use TCP if desired)

Host names are organized hierarchically



- The first level names are called "Top Level Domains"
- Depth of tree is arbitrary (limit 128)
- Domains are subtrees
  - E.g. berkeley.edu and eecs.berkeley.edu
- Name collision avoided
  - E.g. berkeley.edu and berkeley.com

Host names are administered hierarchically



A **zone** corresponds to an administrative authority that is responsible for that portion of the hierarchy

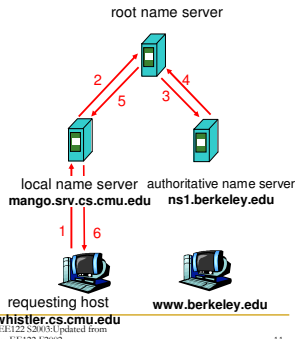
Server Hierarchy

- Servers are organized in hierarchies
- Each server has authority over a portion of the hierarchy
  - A server maintains only a subset of all names
- Each server contains all the records for the hosts in its zone
- Each server needs to know other servers that are responsible for the other portions of the hierarchy
  - Every server knows the root
  - Root server knows about all top-level domains

DNS Example: Recursive Query

Host `whistler.cs.cmu.edu` wants IP address of `www.berkeley.edu`

1. Contacts its local DNS server, `mango.srv.cs.cmu.edu`
2. `mango.srv.cs.cmu.edu` contacts root name server, if necessary
3. Root name server contacts authoritative name server, `ns1.berkeley.edu`, if necessary



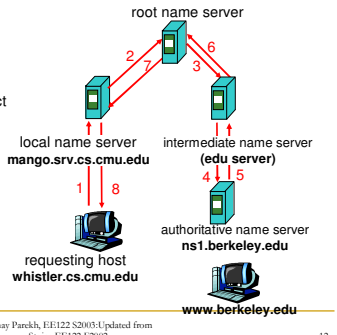
DNS Example: : Recursive Query

Root name server:

- May not know authoritative name server
- May know intermediate name server: who to contact to find authoritative name server?

Recursive query:

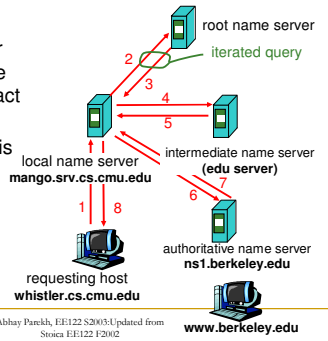
- Puts burden of name resolution on contacted name server
- Heavy load?



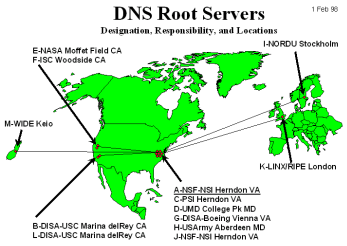
DNS: Iterated Queries

Iterated query:

- Contacted server replies with name of server to contact
- "I don't know this name, but ask this server"



Robustness and Security



- For non-root severs multiple servers are common as well
- Caching provides another form of redundancy and quicker response time
- DOS attack in October 2002
- Secure DNS

{A,..,M}.Root-Servers.Net

Examples

DNS and Mail

- Mail Exchange Point: A host that either processes or forwards mail
- Why should the DNS just resolve IP addresses?
  - MX records map a name to the name of the mail exchange point for that name
  - Example:

```
www.tecknowbasic.com IN 10 formidible.cnchost.com
www.tecknowbasic.com IN 20 zealous.cnchost.com
www.tecknowbasic.com IN 30 inflexible.cnchost.com
```

  - Lower numbers imply higher preference

DNS and Virtual IP addresses

- DNS records don't have to store the real IP address of the host
- All hosts in the acme.com may have the same IP address
  - A firewall at this IP address decides whether to "admit" a transport level connection (firewall) to the host x.acme.com
  - A load balancer decides to forward the connection to one of several identical servers
  - In both cases, the gateway must use a local lookup to decide which end host to direct the connection
- Redirection to anywhere! Even another country.
- Allows for distributed caching architectures
- Makes tracking the geographic location of a name very difficult

Example: www.akamai.com

- From Berkeley

```
C:\>ping www.akamai.com
Pinging a1440.g.akamai.net [64.164.108.148] with 32 bytes of data:
Reply from 64.164.108.148: bytes=32 time=10ms TTL=249
Reply from 64.164.108.148: bytes=32 time=10ms TTL=249
Reply from 64.164.108.148: bytes=32 time=10ms TTL=249
Reply from 64.164.108.148: bytes=32 time=20ms TTL=249

Ping statistics for 64.164.108.148:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 10ms, Maximum =  20ms, Average =  12ms
```

- From the NY Area
  - 63.240.15.146
- From the UK
  - *194.82.174.224*

DNS Summary

- DNS is a crucial part of the internet
- Namespace is hierarchical
- Administration is distributed
- It is vulnerable in various ways but no more than other parts of the internet infrastructure
- Its performance is enhanced by caching
- DNS "Hacks" can enable many interesting things

The WWW

- A distributed database of URLs
- Core components:
  - Servers which store files and execute remote commands
  - Browsers retrieve and display "pages" of content linked by hypertext
    - Each link is a URL
- Can build arbitrarily complex applications, all of which share a uniform client!
- Need a protocol to transfer information between clients and servers
  - HTTP

Uniform Record Locator

- `protocol://host-name:port/directory-path/resource`
- Extend the idea of hierarchical namespaces to include anything in a file system
  - ftp://www.eecs.berkeley.edu/122/Lecture6/presentation.ppt
- Extend to program executions as well…
  - http://us.f413.mail.yahoo.com/ym/ShowLetter?box=%40B%40Bulk&MsgId=2604_1744106_29699_1123_1261_0_28917_3552_1289957100&Search=&Nhead=f&YY=31454&order=down&sort=date&pos=0&view=a&head=b
  - Server side processing can be incorporated in the name

Hyper Text Transfer Protocol

- Client-server architecture
- Synchronous request/reply protocol
  - Runs over TCP, Port 80
- Stateless
- Uses unicast
- (FTP must maintain state)

Hyper Text Transfer Protocol Commands

- GET – transfer resource from given URL
- HEAD – GET resource metadata (headers) only
- PUT – store/modify resource under the given URL
- DELETE – remove resource
- POST – provide input for a process identified by the given URL (usually used to post CGI parameters)

Client Request

- Steps to get the resource: http://www.eecs.berkeley.edu/index.html
  1. Use DNS to obtain the IP address of www.eecs.berkeley.edu
  2. Send to an HTTP request:

```
GET /index.html HTTP/1.0
```

Server Response

```
HTTP/1.0 200 OK
Content-Type: text/html
Content-Length: 1234
Last-Modified: Mon, 19 Nov 2001 15:31:20 GMT
<HTML>
<HEAD>
<TITLE>EECS Home Page</TITLE>
</HEAD>
…
</BODY>
</HTML>
```


Response Codes

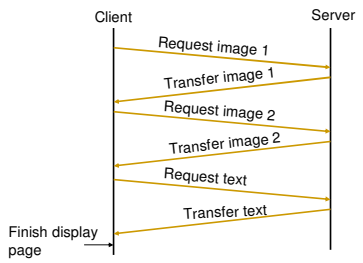
- 1x informational
- 2x success
- 3x redirection
- 4x client error in request
- 5x server error; can't satisfy the request

Example (from Kurose and Ross)

- http://www.mylife.org/mypictures.htm
- After finding out the IP address of the host…

1. http client initiates a TCP connection on :80
2. Client sends the get request via socket established in 1
3. Server sends the html file, which is encapsulated in its response
4. http server tells tcp to terminate connection
5. http client receives the file and the browser parses it…contains ten jpeg images
6. Client repeats steps 1-4

HTTP/1.0 Example



HHTP/1.0 Performance

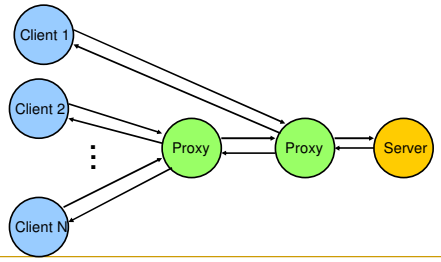
- Create a new TCP connection for each resource
  - Large number of embedded objects in a web page
  - Many short lived connections
- TCP transfer
  - Too slow for small object
  - May never exit slow-start phase
- Connections may be set up in parallel (5 is default in most browsers)

HTTP/1.0 Caching

- Exploit locality of reference
- A modifier to the GET request:
  - If-modified-since – return a "not modified" response if resource was not modified since specified time
- A response header:
  - Expires – specify to the client for how long it is safe to cache the resource
- A request directive:
  - No-cache – ignore all caches and get resource directly from server
- These features can be best taken advantage of with HTTP proxies
  - Locality of reference increases if many clients share a proxy

Web Proxies

- Intermediaries between client and server



Web Proxies (cont'd)

- Location: close to the server, client, or in the network
- Functions:
  - Caching
  - Filter requests/responses
  - Modify requests/responses
    - Change http requests to ftp requests
    - Change response content, e.g., transcoding to display data efficiently on a Palm Pilot
  - Provide better privacy

HTTP/1.1 (1996)

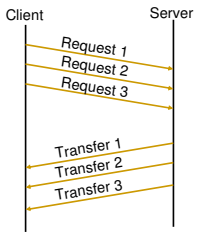
- Performance:
  - Persistent connections
  - Pipelined requests/responses
  - …
- Support for virtual hosting
- Efficient caching support
  - Network Cache assumed more explicitly in the design
  - Gives more control to the server on how it wants data cached

Persistent Connections

- Allow multiple transfers over one connection
- Avoid multiple TCP connection setups
- Avoid multiple TCP slow starts

Pipelined Requests/Responses

- Buffer requests and responses to reduce the number of packets
- Multiple requests can be contained in one TCP segment
- Note: order of responses has to be maintained



Support for Virtual Hosting

- Problem: recall that a request to get http://www.foo.com/index.html has in its header only:
  - GET /index.html HTTP/1.0
- It is not possible to run two web servers at the same IP address, because GET is ambiguous
  - This is useful when outsourcing web content, i.e., company "foo" asks company "outsource" to manage its content
- HTTP/1.1 addresses this problem by mandating "Host" header line, e.g.,

  GET /index.html HTTP/1.1
  Host: www.foo.com

HTTP/1.1 - Caching

- Four new headers
  - Age Header – the amount of time that is known to have passed since the response message was retrieved by the cache
  - Entity tags – unique tags to differentiate between different cached versions of the same resource

HTTP/1.1 - Caching (cont'd)

- Cache-Control
  - no-cache: get resource only from server
  - only-if-cached: obtain resource only from cache
  - no-store: don't allow caches to store request/response
  - max-age: response's should be no greater than this value
  - max-stale: expired response OK but not older than staled value
  - min-fresh: response should remain fresh for at least stated value
  - no-transform: proxy should not change media type

HTTP/1.1 – Caching (cont'd)

- Vary
  - Accommodate multiple representations of the same resource
  - Used to list a set of request headers to be used to select the appropriate representation
- Example:
  - Server sends the following response

```
HTTP/1.1 200 OK
…
Vary: Accept-Language
```

  - Request will contain:

```
Accept-Language: en-us
```

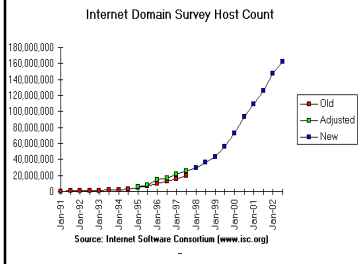
  - Cache return the response that has:

```
Accept-Language: en-us
```


Summary

- HTTP the backbone of WWW'
- Evolution of HTTP has concentrated on increasing the performance
- Next generations (HTTP/NG) concentrate on increasing extensibility

Conclusion



- The applications we discussed today are not complex but they have had huge global impact
- Simplicity, trust in distributed control and open standards helped make this happen.