Internet Names & Addresses

- **Names**: *e.g., ariachne.berkeley.edu*
  - human-usuable labels for machines
  - conforms to "organizational" structure

- **Addresses**: *e.g., 169.229.131.109*
  - router-usuable labels for machines
  - conforms to "network" structure

- How do you map from one to another?  
  - Domain Name System (DNS)
DNS: History

- Initially all host-address mappings were in a file called hosts.txt (in /etc/hosts)
  - Changes were submitted to SRI by email
  - New versions of hosts.txt 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
  - Names were not unique
  - Many hosts had inaccurate copies of hosts.txt

- Internet growth was threatened!
  - Domain Name System (DNS) was born

Basic DNS Features

- Hierarchical namespace
  - As opposed to original flat namespace

- Distributed storage architecture
  - As opposed to centralized storage (plus replication)

- Client–server interaction on UDP Port 53
  - But can use TCP if desired
**Naming Hierarchy**

- "Top Level Domains" are at the top
- Depth of tree is arbitrary (limit 128)
- Domains are subtrees
  - E.g.: .edu, berkeley.edu, eecs.berkeley.edu
- Name collisions avoided
  - E.g. berkeley.edu and berkeley.com can coexist, but uniqueness is job of domain root

**Host names are administered hierarchically**

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

- eecs controls names: x.eecs.berkeley.edu
- berkeley controls names: x.berkeley.edu and y.sims.berkeley.edu
Server Hierarchy

- 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
  - Might be replicated for robustness

- 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 Name Servers

- Local name servers:
  - Each ISP (company) has local default name server
  - Host DNS query first goes to local name server

- Authoritative name servers:
  - For a host: stores that host’s (name, IP address)
  - Can perform name/address translation for that host’s name

- Can also do IP to name translation, but won’t discuss
DNS: Root Name Servers

- Contacted by local name server that cannot resolve name
- Root name server:
  - Contacts authoritative name server if name mapping not known
  - Gets mapping
  - Returns mapping to local name server
- ~ Dozen root name servers worldwide

Simple DNS Example

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
4. `ns1.berkeley.edu` contacts local name server, `mango.srv.cs.cmu.edu`
5. `mango.srv.cs.cmu.edu` returns mapping to local name server
6. Local name server returns IP address to requesting host `whistler.cs.cmu.edu`
Example of Recursive DNS 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?

Example of Iterated DNS Query

Iterated query:
- Contacted server replies with name of server to contact
- “I don’t know this name, but ask this server”
DNS Records

- Four fields: (name, value, type, TTL)

- Type = A:
  - name = hostname
  - value = IP address

- Type = NS:
  - name = domain
  - value = name of dns server for domain

DNS Records (cont’d)

- Type = CNAME:
  - name = hostname
  - value = canonical name

- Type = MX:
  - name = domain in email address
  - value = canonical name of mail server
DNS as Indirection Service

- Can refer to machines by name, not address
  - Not only easier for humans
  - Also allows machines to change IP addresses without having to change way you refer to machine

- Can refer to machines by alias
  - www.berkeley.edu can be generic web server
  - but DNS can point this to particular machine that can change over time

- But, this flexibility applies only within domain!

Special Topics

- DNS caching
  - Improve performance by saving results of previous lookups

- DNS “hacks”
  - Return records based on requesting IP address

- Dynamic DNS
  - Allows remote updating of IP address for mobile hosts

- DNS politics (ICANN) and branding battles
**Important Properties of DNS**

Administrative delegation and distributed server architecture results in:

- Easy unique naming
- Fate sharing for network failures
- Reasonable trust model

---

**The Web – History (I)**

- 1945: Vannevar Bush, Memex:

  "a device in which an individual stores all his books, records, and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility"

(See [http://www.iath.virginia.edu/elab/hf0051.html](http://www.iath.virginia.edu/elab/hf0051.html))
The Web – History (II)

- 1967, Ted Nelson, Xanadu:
  - A world-wide publishing network that would allow information to be stored not as separate files but as connected literature
  - Owners of documents would be automatically paid via electronic means for the virtual copying of their documents
  - Coined the term “Hypertext”

Ted Nelson

The Web – History (III)

- World Wide Web (WWW): a distributed database of “pages” linked through Hypertext Transport Protocol (HTTP)
  - First HTTP implementation - 1990
    - Tim Berners-Lee at CERN
  - HTTP/0.9 – 1991
    - Simple GET command for the Web
  - HTTP/1.0 –1992
    - Client/Server information, simple caching
  - HTTP/1.1 - 1996

Tim Berners-Lee
The Web

- Core components:
  - Servers: store files and execute remote commands
  - Browsers: retrieve and display “pages”
  - Uniform Resource Locators (URLs): way to refer to pages

- A protocol to transfer information between clients and servers
  - HTTP

Uniform Record Locator (URL)

protocol://host-name:port/directory-path/resource

- Extend the idea of hierarchical namespaces to include anything in a file system
  - ftp://www.cs.berkeley.edu/~istoica/cs194/05/lecture.ppt

- Extend to program executions as well...
  - http://us.f413.mail.yahoo.com/ym/Showl etter?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
Web and DNS

- URLs use hostnames
- Thus, content names are tied to specific hosts
- This is bad!
- Uniform Resource Names (URNs) are one proposal to achieve persistence
  - Not discussed in this lecture

Hyper Text Transfer Protocol (HTTP)

- Client-server architecture
- Synchronous request/reply protocol
  - Runs over TCP, Port 80
- Stateless
Big Picture

Hyper Text Transfer Protocol Commands

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

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

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>

HTTP/1.0 Example

![Diagram showing client-server interaction with request and transfer of images and text](image-url)
**HTTP/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 Support**

- 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
HTTP/1.1 (1996)

- Performance:
  - Persistent connections
  - Pipelined requests/responses
  - ...
- Efficient caching support
  - Network Cache assumed more explicitly in the design
  - Gives more control to the server on how it wants data cached
- Support for virtual hosting
  - Allows to run multiple web servers on the same machine

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

### Caching and Replication

- Problem: You are a web content provider
  - How do you handle millions of web clients?
  - How do you ensure that all clients experience good performance?
  - How do you maintain availability in the presence of server and network failures?

- Solutions:
  - Add more servers at different locations → If you are CNN this might work!
  - Caching
  - Content Distribution Networks (Replication)
“Base-line”

- Many clients transfer same information
  - Generate unnecessary server and network load
  - Clients experience unnecessary latency

Reverse Caches

- Cache documents close to server → decrease server load
- Typically done by content providers
Forward Proxies

- Cache documents close to clients → reduce network traffic and decrease latency
- Typically done by ISPs or corporate LANs

![Diagram showing Forward Proxies](image)

Content Distribution Networks (CDNs)

- Integrate forward and reverse caching functionalities into one overlay network (usually) administrated by one entity
  - Example: Akamai
- Documents are cached both
  - As a result of clients’ requests (pull)
  - Pushed in the expectation of a high access rate
- Beside caching do processing, e.g.,
  - Handle dynamic web pages
  - Transcoding
Example: Akamai

- Akamai creates new domain names for each client content provider.
  - e.g., a128.g.akamai.net
- The CDN’s DNS servers are authoritative for the new domains
- The client content provider modifies its content so that embedded URLs reference the new domains.
  - “Akamaize” content, e.g.: http://www.cnn.com/image-of-the-day.gif becomes http://a128.g.akamai.net/image-of-the-day.gif.
Example: Akamai

DNS servers store/cache secondary content for “Akamaized” services.

“Akamaizes” its content.

“Akamaized” response object has inline URLs for secondary content at a128.g.akamai.net and other Akamai-managed DNS names.

What You Need to Know

- DNS: record types, and how they are used
- HTTP basics (and essential differences between 1.0 and 1.1)