# Names

<table>
<thead>
<tr>
<th>Type of Name</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>E-mail</td>
<td><a href="mailto:jrex@cs.princeton.edu">jrex@cs.princeton.edu</a></td>
</tr>
<tr>
<td>Hostname</td>
<td><a href="http://www.cs.princeton.edu">www.cs.princeton.edu</a></td>
</tr>
<tr>
<td>Internet Protocol</td>
<td>128.112.7.156</td>
</tr>
<tr>
<td>Media Access Control</td>
<td>00:15:C5:49:04:A9</td>
</tr>
</tbody>
</table>

Today’s lecture focuses on the last three!
Internet Protocol Layers

**Application**
- Applications
- Hostname

**Transport**
- Reliable streams
- Messages
- IP address

**Network**
- Best-effort *global* packet delivery
- MAC address

**Link**
- Best-effort *local* packet delivery
What’s in a Name?

• Human readable?
  – If end users interact with the names

• Fixed length?
  – If names must be processed at high speed

• Large name space?
  – If many nodes need unique names

• Hierarchical names?
  – If the system is very large and/or federated

• Self-certifying?
  – If preventing “spoofing” is important
Different Layers, Different Names

• **Host name** (e.g., www.cs.princeton.edu)
  – Mnemonic, variable-length, appreciated *by humans*
  – Hierarchical, based on organizations

• **IP address** (e.g., 128.112.7.156)
  – Numerical 32-bit address appreciated *by routers*
  – Hierarchical, based on organizations and topology

• **MAC address** (e.g., 00:15:C5:49:04:A9)
  – Numerical 48-bit address appreciated *by adapters*
  – Non-hierarchical, unrelated to network topology
Hierarchical Allocation Processes

• **Host name**: www.cs.princeton.edu
  – **Domain**: registrar for each top-level domain (e.g., .edu)
  – **Host name**: local administrator assigns to each host

• **IP addresses**: 128.112.7.156
  – **Prefixes**: ICANN, regional Internet registries, and ISPs
  – **Hosts**: static configuration, or dynamic using DHCP

• **MAC addresses**: 00:15:C5:49:04:A9
  – **Blocks**: assigned to equipment vendors by the IEEE
  – **Adapters**: assigned by the vendor from its block
Host Names vs. IP Addresses

• Names are easier (for us!) to remember
  – www.cnn.com vs. 64.236.16.20
• IP addresses can change underneath
  – E.g., renumbering when changing providers
• Name could map to multiple IP addresses
  – www.cnn.com to multiple replicas of the Web site
• Map to different addresses in different places
  – E.g., to reduce latency, or return different content
• Multiple names for the same address
  – E.g., aliases like ee.mit.edu and cs.mit.edu
IP vs. MAC Addresses

- LANs designed for arbitrary network protocols
  - Not just for IP (e.g., IPX, Appletalk, X.25, ...)
  - Different LANs may have different address schemes
- A host may move to a new location
  - So, cannot simply assign a static IP address
  - Instead, must reconfigure the adapter
- Must identify the adapter during bootstrap
  - Need to talk to the adapter to assign it an IP address
## Hostname, IP, and MAC

<table>
<thead>
<tr>
<th></th>
<th>Hostname</th>
<th>IP Address</th>
<th>MAC Address</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Example</strong></td>
<td><strong><a href="http://www.cs.princeton.edu">www.cs.princeton.edu</a></strong></td>
<td><strong>128.112.7.156</strong></td>
<td><strong>00:15:C5:49:04:A9</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Size</strong></th>
<th>Hierarchical, human readable, variable length</th>
<th>Hierarchical, machine readable, 32 bits (in IPv4)</th>
<th>Flat, machine readable, 48 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Read by</strong></td>
<td>Humans, hosts</td>
<td>Internet routers</td>
<td>LAN switches</td>
</tr>
<tr>
<td><strong>Allocation, top-level</strong></td>
<td>Domain name assigned by registrar (e.g., for .edu)</td>
<td>Variable-length prefixes, assigned by ICANN, RIR, or ISP</td>
<td>Fixed-sized blocks, assigned by IEEE to vendors (e.g., Dell)</td>
</tr>
<tr>
<td><strong>Allocation, low-level</strong></td>
<td>Host name assigned by local administrator</td>
<td>Interface, by DHCP or local administrator</td>
<td>Interface, by equipment vendor</td>
</tr>
</tbody>
</table>
Directory: Translate **Name** to **Address**

<table>
<thead>
<tr>
<th>Link</th>
<th>Session</th>
<th>Name</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Path</th>
<th>Address</th>
</tr>
</thead>
</table>
Directory

- A key-value store
  - Key: name, value: address(es)
  - Answer queries: given name, return address(es)
- Caching the response
  - Reuse the response, for a period of time
  - Better performance and lower overhead
- Allow entries to change
  - Updating the address(es) associated with a name
  - Invalidating or expiring cached responses
Directory Design: Three Extremes

• Flood the query (e.g., ARP)
  – The named node responds with its own address
  – But, high overhead in large networks
Address Resolution Protocol (ARP)

• Every host in a LAN maintains an ARP table
  – (IP address, MAC address) pair
• Consult the table when sending a packet
  – Map destination IP address to dest MAC address
  – Transmit the IP packet within an Ethernet frame
Address Resolution Protocol (ARP)

- But, what if the key is not in the table?
  - Sender broadcast: “Who has IP address 1.2.3.19?”
  - Sender caches the result in its local ARP cache
Address Resolution Protocol (ARP)

- Managing the ARP cache
  - Storing all key-value pairs introduces overhead
  - Entries become stale (e.g., IP assigned to new host)
  - Remove an entry if not used for some period of time
Directory Design: Three Extremes

• Flood the query (e.g., ARP)
  – The named node responds with its address
  – But, high overhead in large networks

• Push data to all nodes (e.g., /etc/hosts)
  – All nodes store a full copy of the directory
  – But, high overhead for many names and updates

• Central directory server
  – All data and queries handled by one node
  – But, poor performance, scalability, and reliability
Distributed Directory Design

• Hierarchical directory (e.g., DNS)
  – Follow the hierarchy of the name space
  – Distribute the directory, distribute the queries
  – Enable decentralized updates to the directory

• Distributed Hash Table (e.g., P2P applications)
  – Directory as a hash table with flat names
  – Each directory node handles range of hash outputs
  – Use hash to direct query to the directory node
Domain Name System (DNS)

• Properties of DNS
  – Hierarchical name space divided into zones
  – Distributed over a collection of DNS servers

• Hierarchy of DNS servers
  – Root servers
  – Top-level domain (TLD) servers
  – Authoritative DNS servers

• Performing the translations
  – Local DNS servers and client resolvers
Distributed Hierarchical Database

- **Generic Domains:**
  - com
  - edu
  - org
  - bar
  - west
  - east
  - foo
  - my
  - my.east.bar.edu

- **Country Domains:**
  - ac
  - uk
  - zw
  - arpa
  - in-addr
  - 12
  - 34
  - 56
  - 12.34.56.0/24
  - usr.cam.ac.uk

**Example:**
- my.east.bar.edu
- usr.cam.ac.uk
- 12.34.56.0/24
DNS Root Servers

- 13 root servers (see http://www.root-servers.org/)
- Labeled A through M

A Verisign, Dulles, VA
C Cogent, Herndon, VA (also Los Angeles)
D U Maryland College Park, MD

G US DoD Vienna, VA
H ARL Aberdeen, MD
J Verisign, (11 locations)

K RIPE London (also Amsterdam, Frankfurt)

I Autonomica, Stockholm (plus 3 other locations)

m WIDE Tokyo

E NASA Mt View, CA
F Internet Software C. Palo Alto, CA (and 17 other locations)

B USC-ISI Marina del Rey, CA
L ICANN Los Angeles, CA
TLD and Authoritative DNS Servers

• Global Top-level domain (gTLD) servers
  – Generic domains (e.g., .com, .org, .edu)
  – Country domains (e.g., .uk, .fr, .ca, .jp)
  – Managed professionally (e.g., Verisign for .com .net)

• Authoritative DNS servers
  – Provide public records for hosts at an organization
  – For the organization’s servers (e.g., Web and mail)
  – Can be maintained locally or by a service provider
Using DNS

• Local DNS server ("default name server")
  – Usually near the end hosts who use it
  – Local hosts configured with local server (e.g.,
    /etc/resolv.conf) or learn the server via DHCP

• Client application
  – Extract server name (e.g., from the URL)
  – Do gethostbyname() or getaddrinfo() to get address

• Server application
  – Extract client IP address from socket
  – Optional gethostbyaddr() to translate into name
Host at cis.poly.edu wants IP address for \texttt{gaia.cs.umass.edu}

Recursive query: \#1
Iterative queries: \#2, 4, 6
Recursive vs. Iterative Queries

• Recursive query
  – Ask server to get answer for you
  – E.g., request 1 and response 8

• Iterative query
  – Ask server who to ask next
  – E.g., all other request-response pairs
DNS Caching

- DNS query latency
  - E.g., 1 sec latency before starting a download

- Caching to reduce overhead and delay
  - Small # of top-level servers, that change rarely
  - Popular sites visited often

- Where to cache?
  - Local DNS server
  - Browser
DNS Cache Consistency

• Cache consistency
  – Ensuring cached data is up to date

• DNS design considerations
  – Cached data is “read only”
  – Explicit invalidation would be expensive

• Avoiding stale information
  – Responses include a “time to live” (TTL) field
  – Delete the cached entry after TTL expires
Setting the Time To Live (TTL)

• TTL trade-offs
  – Small TTL: fast response to change
  – Large TTL: higher cache hit rate

• Following the hierarchy
  – Top of the hierarchy: days or weeks
  – Bottom of the hierarchy: seconds to hours

• Tension in practice
  – Set low TTLs for load balancing and failover
  – Browsers cache for 15-60 seconds
Negative Caching

• Broken domain names are slow to resolve
  – Misspellings like www.cnn.comm and www.cnnn.com
  – These can take a long time to fail the first time
• Remember things that *don’t* work
  – Good to remember that they don’t work
  – ... so the failure takes less time in the future
• But don’t remember for *too* long
  – Use a time-to-live to expire
DNS Reliability

• DNS servers are replicated
  – Name service available if at least one replica is up
  – Queries can be load balanced between replicas

• Retransmission of lost queries
  – No response to a query? Try again!

• Try alternate servers on timeout
  – Exponential back-off when retrying same server
Conclusions

• Network names
  – To identify remote end-points
  – Hostnames, IP addresses, and MAC addresses

• Network directories
  – Key-value stores to map name to address
  – Flooding (ARP), local copy, central server
  – Hierarchical (DNS) or non-hierarchical (DHT)

• More on protocol layers in a few weeks!