Naming

To do ...

- What’s in a name
- Flat naming
- Structured naming
- Attribute-based naming
- Next: Content distribution networks
“What's in a name? That which we call a rose
By any other name would smell as sweet.”

W. Shakespeare, “Romeo and Juliet” (II, ii, 1-2)

Names are artificial conventions
Names, identifiers and addresses

- **Names** are used to denote **entities** in a system
  - Hosts, printers, files, processes, users ….

- To operate on an entity, e.g. print a file, we need to access it at an **access point**
  - An entity can offer one or more access points (think phone #s)
Names, identifiers and addresses

- Access points are entities too
- An **address** is the name of an access point
  - Address of an access point of an entity = address of an entity
    (telephone # - telephone - person)

```
742 Evergreen Terrace
Springfield
```

- Location-independent name for entity $E$ – A name **independent** from the addresses of $E$’s access points
  - Easier and more flexible to use
  - $E$ can have multiple addresses and change them
Other names

- Pure names – un-interpreted bit patterns
  - Have no meaning, random strings (only good for comparison)
  - No-pure names contain info about the object (e.g., location)

- (True) Identifiers – names interpreted by programs
  - Each identifier refers to at most one entity (no reusing)
  - Each entity is referred to by at most one identifier
  - E.g. phone is not, passport number maybe

- An identifier need not necessarily be a pure name, i.e., it may have content
- Alias – a name defined to denote the same info as another name
Name services

- How do we resolve (map) names and identifiers to addresses?

- Name service
  - Stores a collection of textual names, as bindings between names and attributes

- A name is resolved when it is translated into data about the named object
Name services

- Name space – Collection of all valid names recognized by a particular service
  - *Name the two principal name spaces in the Internet*

- Binding – Association between name and object
  - In general, names are bound to data about an objects rather than the object itself; e.g., address

- Why is name management a separate service?
  - Unification – resources managed by different services using the same naming scheme
  - Integration - resources created in different administrative domain may eventually be shared
Names in distributed systems

- **Flat naming**
  - Names chosen from a flat set of numeric or symbolic ids
  - Must be globally unique

- **Structured naming**
  - Have an internal structure that represent their position in a hierarchic name space
  - Unique only within immediately containing level
  - Each level resolved within the context of the next higher one

- **Attribute-based naming**
  - Maybe easier to look up entities by attributes \{(attribute, value)\}
  - Entities have a collection of (attribute, value) pairs
Given a unstructured name, how to locate its associated access point?

Key point – name has no information to locate the access point

- Simple solutions
  - Broadcasting
  - Forwarding pointers
- Home-based approaches
- Hierarchical location service
- Distributed Hash Tables
Broadcasting

- Broadcasting – simply broadcast the ID, requesting the entity to return its current address
  - Think Address Resolution Protocol (ARP)
  - Can never scale beyond local-area networks
  - Requires all processes to listen to incoming location requests

- Multicasting can help
  - Join a multicast address and wait for your call
  - Can (sort-of) help when selecting nearest replica
  - What happens if the name is unbound?

- What happens if the entity can move?
Forwarding pointers

- Forwarding pointers – each time an entity moves, it leaves behind a pointer telling where it has gone to
  - Dereferencing can be made entirely transparent to clients by simply following the chain of pointers
  - Update a client’s reference as soon as present location has been found

- Geographical scalability problems
  - Long chains are not fault tolerant – need some chain reduction mechanism
  - Increased network latency at dereferencing
Home-based approaches

- A home keeps track of an entity location (Mobile IP)
  - An entity’s home address is registered at a naming service
  - The home registers the foreign address of the entity
  - Clients contact home first, and continues with foreign location
Home-based approaches – issues

- Home address has to be supported as long as the entity lives
- Home address is fixed, even if entity moves permanently
- Poor geographical scalability (entity may be next to the client)
Hierarchical location system

- Scalability through hierarchies
  - Flat name space with hierarchical administration
- Network divided into a collection of domains and subdomains
- Each domain has an associated directory node
  - Keeps track of entities in that domain
• Tree organization
  – Address of an entity kept in leaf node, or an intermediate node
  – Intermediate nodes contain a pointer to a child iff the sub-tree rooted at the child stores an address of the entity
  – The root knows about all entities

• Lookup
  – Start lookup at the directory node of the local leaf node
  – If node knows it, follow downward pointer, otherwise go one up …
  – Upward lookup stops at root
Back in 5’

• … structured naming
Structured naming

- Flat names are good for machines but hard for humans

- Structured names
  - Have an internal structure that represent their position in a hierarchic name space
  - Unique only within immediately containing level
  - Each level resolved within the context of the next higher one

- Advantages of hierarchies
  - Makes large name spaces more manageable
  - They are potentially infinite (rather than limited by the length/base of the id space)
Name resolution

- On the client’s behalf, a local name resolver
- To resolve a name we need a directory node; how do we actually find that (initial) node?
- Closure mechanism
  - Closure mechanism deal with selecting the initial node in a name space from which to start
  - Many times implicit, e.g., in the Unix FS the i-node of the root directory is the first i-node in the FS
Iterative name resolution

- Interactive – client drives the resolution
  - Caching by clients only (a second client’s resolution of the same name has to go through the same sequence …)
  - Potentially costly communication

At a well-known location
Recursive name resolution

- Recursive – a name server passes result to next server
  - Higher performance demand on servers
  - More effective caching
  - Reduced communication costs

Diagram:
- Client's name resolver
- Root name server
- Name server
- Name server
- Name server
- (www.eecs.northwestern.edu)
- #edu, (www.eecs, northwestern)
- #northwestern (www.eecs)
- #eecs, (www)
- #www
- To edu (www.eecs.northwestern)
- To northwestern (www.eecs)
- To eecs (www)
- (www.eecs, northwestern,edu)
Hierarchical naming – DNS

- Naming for the Internet before DNS (1983)
  - Each computer retrieved HOST.TXT from a computer at SRI
  - A legacy – a *fossil* host file still exist in most modern OS

- DNS (Paul Mockapetris, then at UC Irvine)
  - Names organized into name spaces
  - Name space hierarchically organized as a rooted tree
  - A path name, a listing of labels separated by ‘.’ (root is represented by a dot, generally omitted)
    - eleuthera.cs.northwestern.edu
  - A subtree is called a domain, the path name to its root node is the domain name
  - Content of a node – a collection of resource records
DNS names and name spaces

- Domain name space is partitioned organizationally and geographically (\.edu and \.ar)
- Name space consist of a tree of domains; each node has 0+ resource records
  - SOA – Holds info on the represented zone (email address of sys admin, host from where data on zone can be fetched, …)
  - A – IP address of the host this node represents
  - MX – Mail server to handle mail address to this node
  - NS – Name server that implement the represented zone
  - CNAME – Canonical name of the host (alias implemented by a node storing a CNAME record)
  - HINFO – Info on this host
  - …
DNS scalability

- Scalability though partitioning, replication and caching
- Tree sub-divides into zones beginning at the root
  - Each zone could be 1+ domains and sub-domains
- Zone files – the txt file that describes a zone
  - Includes name and address for 2+ authoritative servers and for delegated subdomains
  - Management parameters (e.g. caching) and RR
  - Information in a zone is kept in 2+ name servers (redundancy)
- Any server can cache data from other servers
  - If a non-authoritative server caches data, it notes the TTL
DNS scalability

- Top of DNS hierarchy – root nameservers keep track of authoritative nameservers for top-level domains (TLDs)
  - Generic TLDs (gTLDs) - .com, .edu, .net
  - Country-code TLDs - .ar, .uk, .in, .cn, ...
- Nameservers are statically configured with 13 IP addresses for the root servers (today)
DNS scalability and resolution

- DNS resolvers operate on behalf of clients to map queries to matching resource records
  - Accepts queries, formats them, contacts servers to answer them
- Resolver follows simple request/reply over UDP
  - Multiple queries/resp per request/reply to save on network traffic
- Either recursive (optional) or iterative resolution
  - Client ask local resolver, resolver contact a root nameserver to find the TLD nameserver
  - … issues the query to TLD, obtains the authoritative nameserver of the next subdomain
    - …
- Aggressive caching to scale
  - Weak consistency through time-to-live (TTL)
Stress on current DNS*

- Increases in malicious behavior
  - Delegation bottleneck – number of NS in delegation chain that need to be compromised to control the domain
  - Worst at the network level – minimum number of gateways …
    - 33% at a single gateway (Microsoft DDOS attack 2001)
  - Buggy implementations with known vulnerabilities
    - 2% with a known buffer overflow bug (in 2004)

- Explosion in client population,
  - Zipf-like query distribution – low performance
  - In 2000, 29% of queries took >2"

- Hierarchy implies higher load at the higher levels
  - 2002 DDOS left 9/13 root servers unresponsive

---

<table>
<thead>
<tr>
<th>Bottlenecks</th>
<th>All Domains</th>
<th>Top 500</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.82%</td>
<td>0.8%</td>
</tr>
<tr>
<td>2</td>
<td>78.44%</td>
<td>62.80%</td>
</tr>
<tr>
<td>3</td>
<td>9.96%</td>
<td>13.20%</td>
</tr>
<tr>
<td>4</td>
<td>4.64%</td>
<td>13%</td>
</tr>
</tbody>
</table>

*V. Ramasubramanian et al, SIGCOMM 2004
Decentralized DNS – back to DHTs

- Map names to responsible nodes through a hash
- CoDoNS based on Beehive
  - Proactive replication on prefix-matching DHTs for $O(1)$ lookup performance (instead of $O(\log N)$)
  - Clearly replicating every entry at every node gives $O(1)$
    - Trick is in picking what to replicate
- Every organization contribute peers
- Same API as DNS, compatible with legacy DNS
  - Use legacy to resolve queries not explicitly inserted by name owners

*V. Ramasubramanian et al, SIGCOMM 2004*
Attributed-based and directory services

- Maybe more convenient to name, and look up entities by attributes \{(attribute, value)\}
  - Simply agreeing on attributes and their values is a challenge

- Directory service
  - Stores collection of bindings between names and attributes
  - Allows attribute-based queries (a.k.a. attribute-based name services or yellow-age services)
  - Expensive lookup – basically an exhaustive search!
Scaling directory services

- Hierarchies again – combine with traditional structured naming system – LDAP, Microsoft’s Active Directory
  - A directory entry – a collection of (attr,value) pairs
  - Set of directory entries form a directory information base
  - Listing attributes in sequence, leads to a tree – the directory information tree
  - And you can partition and distribute the tree among servers

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Abbr</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Country</td>
<td>C</td>
<td>USA</td>
</tr>
<tr>
<td>Locality</td>
<td>L</td>
<td>Evanston</td>
</tr>
<tr>
<td>Organization</td>
<td>O</td>
<td>Northwestern U.</td>
</tr>
<tr>
<td>Organization Unit</td>
<td>OU</td>
<td>EECS</td>
</tr>
<tr>
<td>Common name</td>
<td>CN</td>
<td>Main server</td>
</tr>
<tr>
<td>Mail_Server</td>
<td>-</td>
<td>129.105.8.4</td>
</tr>
<tr>
<td>WWW_Server</td>
<td>-</td>
<td>129.105.8.227</td>
</tr>
</tbody>
</table>
Scaling directory services

- Entities’ descriptions translated into attribute-value trees (AVTree) then encoded into a set of unique hash ids for a … (yes) … DHT
  - INS/Twine, SWORD, Mercury

Entity description

```plaintext
description {
  type = book
  description {
    author = Tolkien
    title = LOTR
  }
  genre = fantasy
}
```

Corresponding AVTree

```
  type: book
  author: Tolkien
  title: LOTR

genus: fantasy
```

A hash for every path

- h1: hash(type-book)
- h2: hash(type-book-author)
- h3: hash(type-book-author-Tolkien)
- h4: hash(type-book-title)
- h5: hash(type-book-title-LOTR)
- h6: hash(genre-fantasy)

Queries translated into AVTree for matching

- Single-value queries (ranges would be harder)
Summary

- Naming is central to computer systems in general and distributed systems in particular
- How do you name things?
- How do you find what you are looking for?
- What if that’s a moving target?
- How do you implement name/directory services in an scalable manner?