Today
- What’s in a name
- Flat naming
- Structured naming
- Attribute-based naming
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 1+ access points (think of phone #s)
Names, identifiers and addresses

- Access points are entities as well; the name of an access point is an *address*
  - Address of an access point of an entity = address of an entity
    (person – telephone – telephone #)

Chile 371, San Telmo
Capital Federal
Argentina

- 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
Names, identifiers and addresses

- Pure names – un-interpreted bit patterns
  - Have no meaning, they are just random strings
  - Can be used for comparison only
  - No-pure names contain information about the object that they name (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
Names, identifiers and addresses

- Name space – the collection of all valid names recognized by a particular service
- Binding – association between a name and an object
  - In general names are bound to attributes of an objects rather than the object itself; e.g., address
- Name resolution – a name is resolved when it is translated into data about the named object
- Alias – a name defined to denote the same info as another name
  - Simple allow multiple absolute paths names refer to the same node – hard link
  - Allow a node S to contain a name of another node – soft link
Names in distributed systems

- **Flat names**
  - Chosen from a flat set of numeric or symbolic identifiers
  - Must be globally unique

- **Hierarchical 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 services

- Names service – stores a collection of textual names, in the form of bindings between names and attributes
  - Collection often subdivided in 1+ contexts managed as a unit
- Key operation supported – resolution
- 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 (and so must have common naming)
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 – needed to select the implicit context from which to start name resolution
  - www.eecs.northwestern.edu: start at a DNS name server
- Name resolver can follow
  - Interactive – client drives the resolution
    - Caching by clients
    - Potentially costly communication
  - Recursive – the server does
    - Higher performance demand on servers
    - More effective caching
    - Reduced communication costs
Name resolution and flat names

- Given an essentially unstructured name, how to locate its associated access point?
- Key point – name does not contain any information on how to locate the access point
  - Simple solutions
    - Broadcasting
    - Forwarding pointers
  - Home-based approaches
  - Distributed Hash Tables
  - Hierarchical location service
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?

- And what happens if the entity moves?
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
- 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

The root directory node $\text{dir}(T)$

Top-level domain $T$

Directory node $\text{dir}(S)$ of domain $S$

A subdomain $S$ of top-level domain $T$ (S is contained in T)

A leaf domain, contained in $S$
Lookup and caching

- **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 local leaf node
  - If node knows it, follow downward pointer, otherwise go one up …
  - Upward lookup stops at root
Hierarchical naming – DNS

- Naming for the Internet before DNS (1983) each computer retrieved HOST.TXT from a computer at SRI
  - Legacy – a host file still exist in most modern OS
- DNS by Paul Mockapetris (1983)

- Internet names are structured, no flat
  - eleuthera.cs.northwestern.edu

- Resolution works the same way
  - If a name is cached in local server, try to use it
  - Otherwise go up the hierarchy one level …
  - Difference is that each level knows only its level
    - edu knows of northwestern but not about eleuthera
DNS names and name spaces

- Names organized into name spaces
- Domain name space is partitioned organizationally and geographically (.edu and .ar)
- Name space consist of a tree of domain names; each node has 0+ resource records
  - SOA – Holds info on the represented zone
  - 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
  - HINFO – Info on this host
  - ...
DNS scalability

- Scalability through 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
  - The information contained in each zone is implemented in 2+ name servers (2 for redundancy)
- Any server can cache data from other servers
  - If a non-authoritative server caches data, it notes the TTL
DNS resolution and other uses

- DNS client, a.k.a. resolver, normally a library
- Accepts queries, formats them and contacts name servers to answer them
- Follow simple request/reply over UDP
  - Possible with multiple queries/responses per request/reply to save on network traffic
- Either recursive (optional) or iterative resolution
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 a exhaustive search!
Scaling directory services

- Hierarchies again – combine with traditional structured naming system – LDAP, 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 (DIT)
  - 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 are translated into attribute-value trees which are encoded into a set of unique hash ids for a DHT
  - INS/Twine, SWORD, Mercury

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

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)
```

Query: Books written by Tolkien?
hash
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?