Naming

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 one or more access points (think 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 #)

- 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

*Chile 371, San Telmo Capital Federal Argentina*
Other names

- 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

- Alias – a name defined to denote the same info as another name
Name services

- 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

- Binding – Association between a name and an object
  - In general, names are bound to data about an objects rather than the object itself; e.g., address
  - A name is resolved when it is translated into data about the named object

- Name space – The collection of all valid names recognized by a particular service

- 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 more convenient to look up entities by attributes
    - \{(attribute, value)\}
  - Entities have a collection of (attribute, value) pairs
Name resolution and flat names

- Given an essentially 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
  - 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
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)

1. Send packet to host at its home
2. Return address of current location
3. Tunnel packet to current location
4. Send successive packets to current location
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

Diagram:
- 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
Structured naming

- **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
  - Knowing how to and where to start name resolution
  - 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
Recursive name resolution

- Recursive – the server does
  - Higher performance demand on servers
  - More effective caching
  - Reduced communication costs
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 by Paul Mockapetris

- 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 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
  - 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
  - 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 scalability

- Top of DNS hierarchy – root nameservers keep track of authoritative nameservers for the 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
DNS scalability and resolution

- DNS resolvers operate on behalf of clients to map queries to matching resource records
  - Accepts queries, formats them and contacts nameservers to answer them

- Resolver follows simple request/reply over UDP
  - Multiple queries/responses 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

- 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

*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 institution contribute peers
- Same API as DNS, compatible with legacy DNS
  - Use legacy to resolve queries not explicitly inserted by nameowners

*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

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

Queries translated into AVTree for matching

Query: Books written by Tolkien?

- 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?