Naming Services

Instructor: Dr. Tongping Liu

Outline

- Overview: name and name services
  - Naming space and implementation
- Flat name and simple resolutions
- Structure name
  - Name space and resolution
  - Case study: DNS
- Attributed-based naming
  - Directory service
  - Hierarchical Implementations: LDAP
  - Decentralized Implementations

Objectives

- To understand naming and related issues in DS
- To learn naming space and implementation
- To learn flat and structured names and how they are resolved
- To learn Attributed-based naming

Midterm II

- Average: 132.58
- Median: 137.5
- Minimum: 85
- Maximum: 154
- 80~100: 19 (73%)
What a Name is in DS?

- A name is a string of bits or characters that is used to refer to an entity (an entity could be anything such as host, printer, file, process, mailbox, user etc.).
- To operate on an entity, we need to access it, for which we need an access point.
- Access point is a special kind of entity and its name is called as an address (address of the entity, e.g., hardware address, IP, port #, phone #)
  - An entity may have more than one access point/address
  - An entity may change its access points/addresses
  - So using an address as a reference is inflexible and human unfriendly
  - A better approach is to use a name that is location independent, much easier, and flexible to use: web service for different web servers

Identifier

- A special name to uniquely identify an entity (SSN, MAC)
- A true identifier has the following three properties:
  - P1: Each identifier refers to at most one entity
  - P2: Each entity is referred to by at most one identifier
  - P3: An identifier always refers to the same entity (no reuse)

Human-friendly names

- File names, www.cs.utsa.edu, variable names etc. are human-friendly names given to each entity
- Question: how to map/resolve these names to addresses so that we can access the entities on which we want to operate?
- Solution: have a naming system that maintains name-to-address binding!
- A centralized table: Why or why not this will work?

```
NS() ➔ NS(edu) ➔ NS(utsa.edu) ➔ NS(www.cs.utsa.edu)
```

Naming Systems and Their Goals

- Naming Systems
  - Flat names
  - Structured names
  - Attributed-based names
- Goals
  - Scalable to arbitrary size
  - Have a long lifetime
  - Be highly available
  - Have fault isolation
  - Tolerate mistrust
Name Service Basics

- **Name space**
  - Collection of *valid* names recognized by a name service

- **Naming context**
  - A set of textual names and attributes

- A naming **domain** is a name space which has a single administrative authority

- **Name service: serve requests to**
  - Create and delete bindings
  - List bindings by name or attribute
  - Add or delete contexts

Name Space Requirements

- Allow simple but meaningful names to be used
- Potentially infinite number of names
- Structured
  - to group related names
  - to allow similar sub-names without clashes
- Allow re-structuring of name trees
  - Easy compatible with old programs
- Management of trust

Outline

- **Overview: name and name services**
  - Naming space and implementation

- **Flat name and simple resolutions**

- **Structure name**
  - Name space and resolution
  - Case study: DNS

- **Attributed-based naming**
  - Directory service
  - Hierarchical Implementations: LDAP
  - Decentralized Implementations

Flat Naming

- **Flat name** random bits of string, no structure
  - E.g., SSN, MAC address

- **Resolution problem:**
  Given a flat (unstructured) name, how can we find/locate its associated access point and its address?

- **Solutions:**
  - Simple solutions (broadcasting)
  - Home-based approaches
  - Distributed Hash Tables (structured P2P)
  - Hierarchical location service
Simple Solution: Broadcasting

- Simply **broadcast** the target ID to every entity
- Each entity compares the requested ID with its own ID
- The target entity returns its current address

**Example:**
- Recall ARP in LAN (Address Resolution Protocol)

**Advantages/Disadvantages**
- + simple
- - not scale beyond LANs
- - it requires all entities to listen to all incoming requests

---

Home-based Approach: Forwarding Pointers

- When an entity moves from A to B, leaves a pointer to A that it is at B now...

**Dereferencing:** simply follow the chain of pointers and make this entirely transparent to clients

**Advantages/Disadvantages**
- + support for mobile nodes
- - geographical scalability problems
- - long chains are not fault tolerant
- - increased network latency

Short-cuts can be introduced

---

Introduce a short-cut

- Sending response back for the server:
  - Sending directly to the initiating client: faster
  - Along the reverse path of forwarding pointers: allows adjustment of all stubs

---

Home-Based Approaches

How to deal with scalability problem when locating mobile entities?

- **Let a home** keep track of where the entity is!

**How will the clients continue to communicate?**
- Home agent gives the new location to the client so it can directly communicate
  - efficient but not transparent

Home agent forwards the messages to new location
- Transparent but may not be efficient
Problems with home-based approaches

- The home address has to be supported as long as the entity lives.
- The home address is fixed, which means an unnecessary burden when the entity **permanently moves** to another location
  - How can we solve the “permanent move” problem?
    - Register the home at a naming service and let the client to lookup at first
- Poor geographical scalability (the entity may be next to the client)

DHT: Overview

- **Abstraction**: a distributed “hash-table” (DHT) data structure supports two operations:
  - put(id, item);
  - item = get(id);
- **Implementation**: nodes in system form a distributed data structure
  - Can be Ring, Tree, Hypercube, Skip List, Butterfly Network, ...

Distributed Hash Tables

- Academic answer to p2p
- **Goals**
  - Guaranteed lookup success
  - Provable bounds on search time
  - Provable scalability
- **Challenges**
  - Decentralized: no central authority
  - Scalable: low network traffic overhead
  - Efficient: find items quickly (latency)
  - Dynamic: nodes fail, new nodes join
  - General-purpose: flexible naming

What Is a DHT?

- A building block used to locate key-based objects over millions of hosts on the internet
- Inspired from traditional hash table:
  - key = Hash(name)
  - put(key, value)
  - get(key) -> value

Borrowed from Adriana Iamnitchi at Usf

Borrowed from Adriana Iamnitchi at Usf
The Lookup Problem

Distributed Hash Tables (DHT)

- Organizing many nodes into a logical ring
  - Each node is assigned a random m-bit identifier.
  - Every entity is assigned a unique m-bit key.
  - Entity with key k falls under jurisdiction of node with smallest id >= k (called its successor).

- Linearly resolve a key k to the address of succ(k)
  - Each node p keeps two neighbors:
    - succ(p+1) and pred(p)
  - If k > p then forward to succ(p+1)
  - If k <= pred(p) then forward k to pred(p)
  - If pred(p) < k <= p then return p’s address (p holds the entity)

- Answer question quickly: where is key 10?
  - in this case, S12 should respond that “S12 has K10?”
  - However, it involves in multiple message transfers over the network, depending on which node you start with.

- Need at most O(log N) steps, where N is the number of nodes in the systems

DHT: Finger Table for Efficiency

- Each node p maintains a finger table
  - at most i entries (short cuts) with exponentially increasing size
    \[ FT[p][i] = succ(p + 2^{−i}) \]
  - \( FT[p][i] \) points to the first node succeeding p by at least \( 2^{−i} \)

- To look up a key k, node p forwards the request to node with index j satisfying
  \[ q = FT[p][j] \leq k < FT[p][j + 1] \]
  (e.g., node 0 sends req \( \rightarrow 4 \rightarrow 6 \))
  \( p < k < FT[p][1] \), the request is also forwarded to \( FT[p][1] \)

- Need at most O(log N) steps, where N is the number of nodes in the systems

Borrowed from Adriana Iamnitchi at Usf
DHT: Example

DHT: Finger Table (cont’d)

- How to handle
  - Join: Lookup for $\text{succ}(p+1)$.
  - Leave: Node id informs its departure to its predecessor and successor, and transfers its data item to successor.

- The complexity comes from keeping the finger tables up to date.

- By-and-large Chord tries to keep them consistent
  - But a simple mechanism may lead to performance problems.
  - To fix this we need to exploit network proximity when assigning node ID.

Exploiting network proximity

- **Problem:** Lead to erratic message transfers in the underlying Internet: node $k$ and node $\text{succ}(k+1)$ may be very far apart (node (1,20) SA, node (18, 21) is Germany).

- **Topology-aware node assignment:**
  - Nodes close in the ID space are also close in the topology. Can be very difficult.

- **Proximity routing:**
  - Maintain more than one possible successor, and forward to the closest.
  - Example: in Chord $\text{FT}_{p}[i]$ points to first node in INT = $[p+2i−1, p+2i−1]$. Node $p$ can also store pointers to other nodes in INT.

- **Proximity neighbor selection:**
  - When there is a choice of selecting who your neighbor will be (not in Chord), pick the closest one.

Outline

- **Overview:** Name and name services
  - Naming space and implementation
- **Flat name** and simple resolutions
- **Structure name**
  - Name space and resolution
  - Case study: DNS
- **Attributed-based naming**
  - Directory service
  - Hierarchical Implementations: LDAP
  - Decentralized Implementations
A name graph represents a (named) entity.

A directory node is an entity that refers to other nodes: contains a (directory) table of (node identifier, edge label) pairs.

Observation: We can easily store all kinds of attributes in a node, describing aspects of the entity the node represents:

- Type of the entity
- An identifier for that entity
- Address of the entity’s location
- Nicknames
  ...

Directory nodes can also have attributes, besides just storing a directory table with (edge label, node identifier) pairs.

Global level: high-level directory nodes. Stable

Administrational level: directory nodes that are managed within a single organization. Relatively stable.

Managerial level: within a single administration. Nodes may change regularly.

<table>
<thead>
<tr>
<th>Item</th>
<th>Global</th>
<th>Administrative</th>
<th>Managerial</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geographical scale of network</td>
<td>Worldwide</td>
<td>Organization</td>
<td>Department</td>
</tr>
<tr>
<td>Total number of nodes</td>
<td>Few</td>
<td>Many</td>
<td>Vast numbers</td>
</tr>
<tr>
<td>Responsiveness to lookups</td>
<td>Seconds</td>
<td>Milliseconds</td>
<td>Immediate</td>
</tr>
<tr>
<td>Update propagation</td>
<td>Lazy</td>
<td>Immediate</td>
<td>Immediate</td>
</tr>
<tr>
<td>Number of replicas</td>
<td>Many</td>
<td>None or few</td>
<td>None</td>
</tr>
<tr>
<td>Is client-side caching applied?</td>
<td>Yes</td>
<td>Yes</td>
<td>Sometimes</td>
</tr>
</tbody>
</table>

Caching and replicas are introduced for performance, but with consistency problems.

Name Space Implementation
Name service: allows users to add, remove and lookup name

Hierarchical Design

Name Space Implementation (cont.)
Name Resolution: where to start?

Definition: the process of looking up a name
- DNS resolution: the process of resolving a domain name to an IP address

Problem: To resolve a name we need a directory node. How do we actually find that (initial) node?

Closure mechanism: The mechanism to select the initial node from which to start name resolution
- www.cs.vu.nl: start at a DNS name server
- /home/steen/mbox: start at the local NFS file server (possible recursive search)

Name Resolution: Iterative

- `resolve(dir,[name1, ..., nameK])` is sent to Server0 responsible for `dir`
- Server0 resolves `resolve(dir,name1)` to `dir1`, returning the identification (address) of Server1, which stores `dir1`.
- Client sends `resolve(dir1,[name2, ..., nameK])` to Server1, etc.

Name Resolution: Recursive

- `resolve(dir,[name1,...,nameK])` is sent to Server0 responsible for `dir`
- Server0 resolves `resolve(dir,name1)` to `dir1`, returning the identification (address) of Server1, which stores `dir1`.
- Client sends `resolve(dir1,[name2,...,nameK])` to Server1, etc.

Cache in Recursive Naming Resolution

<table>
<thead>
<tr>
<th>Server for node</th>
<th>Should resolve</th>
<th>Looks up</th>
<th>Passes to child</th>
<th>Receives and caches</th>
<th>Returns to requester</th>
</tr>
</thead>
<tbody>
<tr>
<td>cs</td>
<td>&lt;ftp&gt;</td>
<td>&lt;ftp&gt;</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>vu</td>
<td>&lt;cs,ftp&gt;</td>
<td>&lt;cs,ftp&gt;</td>
<td>&lt;ftp&gt;</td>
<td>&lt;ftp&gt;</td>
<td>—</td>
</tr>
<tr>
<td>nl</td>
<td>&lt;cs,ftp&gt;</td>
<td>&lt;cs,ftp&gt;</td>
<td>&lt;ftp&gt;</td>
<td>&lt;ftp&gt;</td>
<td>—</td>
</tr>
<tr>
<td>root</td>
<td>&lt;nl,cs,ftp&gt;</td>
<td>&lt;cs&gt;</td>
<td>&lt;cs,ftp&gt;</td>
<td>&lt;cs,ftp&gt;</td>
<td>&lt;cs,ftp&gt;</td>
</tr>
</tbody>
</table>

*: another one ask for `<nl,vu,cs,flits>`, forward it to `<nl,vu,cs>` server
Recursive vs. Iterative Name Resolution

Scalability Issue
- **Size scalability**: handle a large number of requests per time unit → high-level servers are in big trouble.
- **Solution**: Assume (at least at global and administration level) that content of nodes *hardly ever changes*. In that case, we can apply extensive replication by mapping nodes to multiple servers, and start name resolution at the nearest server.
- **Geographical scalability**: We need to ensure that the name resolution process scales across large geographical distances.

Outline
- **Overview**: name and name services
  - Naming space and implementation
- **Flat name** and simple resolutions
- **Structure name**
  - Name space and resolution
  - Case study: DNS
- **Attributed-based naming**
  - Directory service
  - Hierarchical Implementations: LDAP
  - Decentralized Implementations

Case Study: Domain Name System (DNS)
- A distributed naming database
- Name structure reflects administrative structure of the Internet
- Rapidly resolves domain names to IP addresses
  - exploits caching heavily
  - typical query time ~100 milliseconds
- Scales to millions of computers
  - partitioned database
  - caching
- Resilient to failure of a server
  - replication
Specific Design

- A label has a maximum of 63 characters; the length of a path is limited to 255 characters.
- Labels are separated by ".".
  - Path name: "root: <nl, vu, cs, flits>"
  - String: flits.cs.vu.nl.
- Each node has exactly one incoming edge, the label of its incoming edge is used as the name for the node.
- A subtree is called a domain. A path name to its root node is called a domain name.

Name spaces in DNS

- A subtree is called a domain; a path name to its root node is called a domain name.
  - www.utsa.edu -- domain name
- Allows aliases such as www.utsa.edu → pandora.cs.utsa.edu

Domain names (last element of name)

- com - commercial organizations
- edu - universities and educational institutions
- gov - US government agencies
- mil - US military organizations
- net - major network support centers
- org - organizations not included in first five
- int - international organization
- country codes - (e.g., cn, us, uk, fr, etc.)

DNS resource records: contents of a node

<table>
<thead>
<tr>
<th>Record type</th>
<th>Meaning</th>
<th>Main contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A computer address</td>
<td>IP number</td>
</tr>
<tr>
<td>NS</td>
<td>An authoritative name server</td>
<td>Domain name for server</td>
</tr>
<tr>
<td>CNAME</td>
<td>The canonical name for an alias</td>
<td>Domain name for alias</td>
</tr>
<tr>
<td>SOA</td>
<td>Marks the start of data for a zone</td>
<td>Parameters governing the zone</td>
</tr>
<tr>
<td>WKS</td>
<td>A well-known service description</td>
<td>List of service names and protocols</td>
</tr>
<tr>
<td>PTR</td>
<td>Domain name pointer (reverse lookups)</td>
<td>Domain name</td>
</tr>
<tr>
<td>HINFO</td>
<td>Host information</td>
<td>Machine architecture and operating system</td>
</tr>
<tr>
<td>MX</td>
<td>Mail exchange</td>
<td>List of &lt;preference, host&gt; pairs</td>
</tr>
<tr>
<td>TXT</td>
<td>Text string</td>
<td>Arbitrary text</td>
</tr>
</tbody>
</table>
Zone partitioning of DNS name space

- Zone:
  - Each zone is implemented by a name server and is always replicated for availability
  - Zone updates are handled by the primary name server
  - Example: UTSA has a name server for utsa.edu, but cs.utsa.edu names are resolved by the CS Dept. server
- Names of the servers for the sub-domains
- At least two name servers that provide authoritative data for the zone
- Zone management parameters: cache, replication

---

DNS Name Resolution

- Domain name → IP address ???
- Look for the name in the local cache
- Try a superior DNS server, which responds with:
  - the IP address
  - another recommended DNS server

---

DNS name servers

Note: Name server names are in italics, and the corresponding domains are in parentheses. Arrows denote name server entries.

authentative path to lookup: jeans-pc.dcs.qmw.ac.uk

---

The zone partitioning diagram illustrates the structure and management of DNS zones, highlighting the role of name servers and the replication strategy for availability. The DNS name resolution process is summarized, showing a step-by-step approach from domain name to IP address lookup, utilizing both local cache and superior DNS servers when necessary.
DNS Resolution

Three classes of DNS servers:
- Root servers - 13 root name servers worldwide. These servers maintain the IP address of all of the top-level domain servers.
- Top-Level Domain (TLD) servers: each organization responsible for a particular domain (e.g., com, edu), e.g., Educase maintains EDU TLD.
- Authoritative servers - Each organization (e.g., UTSA or Amazon) maintains the DNS server that can provide authoritative mappings from hostname to IP.

DNS name resolution with cache

Lookup:
jeans-po.dcs.qmw.ac.uk

IP: alpha.qmw.ac.uk

IP: dns0.dcs.qmw.ac.uk

IP: ns0.ja.net

IP: ns1.purdue.edu

DNS Resolution

The computer asks its local DNS server for a mapping.
Local DNS server contacts a root DNS server for the mapping.
The root DNS server responds with the IP address of the TLD DNS server for relevant domain.
Your local DNS server contacts the TLD server and the TLD server responds with the address of the authoritative server for the domain in question.
Your local DNS server contacts the authoritative server and gets the correct IP address.
Your local DNS server returns the address to your computer.
DNS Name Resolution

- The combination of both recursive and iterative name resolution
  - Computer to local DNS – recursive
  - Local DNS to domain server and authoritative servers are iterative

- It is able to configure DNS to use recursive queries, but the result would then propagate back through the chain and the root will be the bottleneck
  
  How to reduce the messages over the network?

Caching in DNS

- Any server can cache any name
- Non-authoritative servers note time-to-live when they cache data
- Non-authoritative servers indicate that they are such when responding to clients with cached names

DNS issues

- Name tables change infrequently, but when they do, caching can result in the delivery of stale data.
  - Clients are responsible for detecting this and recovering
- Its design makes changes to the structure of the name space difficult. For example:
  - merging previously separate domain trees under a new root
  - moving sub-trees to a different part of the structure

  Overall: it still runs well after 30 years, no need to be replaced!

Basic Principles for Scalable Design

- Partitioning
  - Split systems into parts that can operate independently to a large extent
- Replication: provide several copies of components
  - that are kept consistent eventually
  - that can be used in case of failure of copies
- Locality (caching):
  - maintain a copy of information that is nearer, cheaper/faster to access than the original
How DNS Achieves the Scalability?

- Hierarchical Design
- Partitioning: Zones
  - Resolves all names within a zone recursively
- Replication: currently 13 root name servers
  - Each zone has at least one primary and one secondary name server
- Locality (caching): each name server caches resource records
  - time to live attribute
  - authoritative versus non-authoritative answers

Outline

- Overview: name and name services
  - Naming space and implementation
- Flat name and simple resolutions
- Structure name
  - Name space and resolution
  - Case study: DNS
- Attributed-based naming
  - Directory service
  - Hierarchical Implementations: LDAP
  - Decentralized Implementations

Flat vs Structure Names

- +: Provides a unique and location-independent way of referring to entities
- +: Structured names can provide a human-friendly way to name entities
- -: Location independence and human friendliness is not enough. More information is helpful too.

Directory Services

- Entities have a set of attributes (e.g., email, send, recv, subject, ...)
- In most cases, attributes are determined manually
- Setting values consistently is a crucial problem ...
- Often organized in a hierarchy
  - Examples of directory services: X.500, Microsoft’s Active Directory Services,
- Then, look up entities by means of their attributes
- Problem: Lookup operations can be extremely expensive, as they require to match requested attribute values, against actual attribute values
  - In the simplest form, inspect all entities.
Directory Services (cont’d)

Solutions:

- **Lightweight Directory Access Protocol (LDAP):**
  - Implement basic directory service as database, and
  - Combine it with traditional structured naming system.
  - Derived from OSI’s X.500 directory service, which maps a person’s name to attributes (email address, etc.)

Hierarchical implementation: LDAP (1)

- LDAP directory service consists of a set of records
- Each directory entry (record) is made up of a set of (Attribute, Value(s)) pairs

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Addr.</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Country</td>
<td>C</td>
<td>NL</td>
</tr>
<tr>
<td>Locality</td>
<td>L</td>
<td>Amsterdam</td>
</tr>
<tr>
<td>Organization</td>
<td>O</td>
<td>Vrije Universiteit</td>
</tr>
<tr>
<td>OrganizationalUnit</td>
<td>OU</td>
<td>Comp. Sc.</td>
</tr>
<tr>
<td>CommonName</td>
<td>CN</td>
<td>Main server</td>
</tr>
<tr>
<td>Mail Servers</td>
<td>—</td>
<td>137.37.20.3, 130.37.24.8, 137.37.20.10</td>
</tr>
<tr>
<td>FTP Server</td>
<td>—</td>
<td>130.37.20.20</td>
</tr>
<tr>
<td>WWW Server</td>
<td>—</td>
<td>130.37.20.20</td>
</tr>
</tbody>
</table>

Hierarchical implementation: LDAP (2)

- We can create a directory information tree (DIT) by listing RDNs in sequence

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Country</td>
<td>NL</td>
</tr>
<tr>
<td>Locality</td>
<td>Amsterdam</td>
</tr>
<tr>
<td>Organization</td>
<td>Vrije Universiteit</td>
</tr>
<tr>
<td>OrganizationalUnit</td>
<td>Comp. Sc.</td>
</tr>
<tr>
<td>CommonName</td>
<td>Main server</td>
</tr>
<tr>
<td>Host Name</td>
<td>zaphyr</td>
</tr>
<tr>
<td>Host Address</td>
<td>142.31.290.42</td>
</tr>
</tbody>
</table>

List all main servers at Vrije search("(&(C = NL) (O = Vrije Universiteit) (OU = *) (CN = Main server))")
Summary

- Overview: name and name services
  - Naming space and implementation
- Flat name and resolutions
- Structure name
  - Name resolution: iterative vs. recursive
  - Case study: DNS
- Attributed-based naming
  - Directory service
  - Case study: X.500