CS 5523 Operating Systems: Naming Services

Instructor: Dr. Tongping Liu

Thank Dr. Dakai Zhu and Dr. Palden Lama for providing their slides.
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
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 an address (address of the entity, e.g., 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
A special name to \textit{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)

Addresses and identifiers are important and used for different purposes, but they are often represented in machine readable format (MAC, memory address)
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?

\[ \text{NS(.)} \rightarrow \text{NS(edu)} \rightarrow \text{NS(utsa.edu)} \rightarrow \text{NS(www.cs.utsa.edu)} \]
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)

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

- **Adv/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)
Distributed Hash Tables (DHT)

Recall Chord from Chapter 2, which organizes 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 \( id \geq k \) (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 \( k \leq pred(p) \) then
  - forward \( k \) to \( pred(p) \)
- If \( pred(p) < k \leq p \) then
  - return \( p \)'s address (\( p \) holds the entity)
DHT: Finger Table for Efficiency

- Each node $p$ maintains a **finger table** at most $m$ entries (short cuts) with exponentially increasing size

  \[ FT_p[i] = \text{succ}(p + 2^{i-1}) \]

- $FT_p[i]$ points to the first node succeeding $p$ by at least $2^{i-1}$

- 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 $\rightarrow 4 \rightarrow 6$)

- If $p < k < FT_p[1]$, 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
DHT: Example

Resolve k = 12 from node 28

Resolve k = 26 from node 1
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 \( FTp[i] \) points to first node in \( \text{INT} = [p+2i−1,p+2i −1] \). Node \( p \) can also store pointers to other nodes in \( \text{INT} \).

- **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
Structure Name

- **A name graph**
  - **Leaf node** represents a (named) entity.
  - **A directory node** is an entity that refers to other nodes: contains a (directory) table of *(edge label, node identifier)* pairs.
Structure Name (cont.)

- **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.
**Name Space Implementation**

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

- **Basic issue over a large geographical area:** Distribute the implementation (name resolution and management) across multiple name servers

---

**Hierarchical Design**
Name Space Implementation (cont.)

- **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 Resolution: where to start?

- **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 implicit context 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)→dir1`, returning the identification (address) of Server1, which stores dir1.
- Client sends `resolve(dir1,[name2,...,nameK])` to Server1, etc.

```
1. <nl,vu,cs,ftp>
2. #<nl>, <vu,cs,ftp>
3. <vu,cs,ftp>
4. #<vu>, <cs,ftp>
5. <cs,ftp>
6. #<cs>, <ftp>
7. <ftp>
8. #<ftp>

Nodes are managed by the same server
```
Name Resolution: **Recursive**

- : a higher performance demand on each name server, especially global layer
+ : caching results is more effective
+ : communication costs may be reduced
## 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>#&lt;ftp&gt;</td>
</tr>
</tbody>
</table>
| vu              | <cs, ftp>      | #<cs>    | <ftp>           | #<ftp>              | #<cs>  
|                 |                |          |                 |                     | #<cs, ftp>          |
| nl              | <vu, cs, ftp>  | #<vu>    | <cs, ftp>       | #<cs>  
|                 |                |          |                 | #<cs, ftp>          | #<vu>  
|                 |                |          |                 |                     | #<vu, cs>          
|                 |                |          |                 |                     | #<vu, cs, ftp>      |
| root            | <nl, vu, cs, ftp> | #<nl>   | <vu, cs, ftp>   | #<vu>  
|                 |                |          |                 | #<vu, cs>          
|                 |                |          |                 | #<vu, cs, ftp>      | #<nl>  
|                 |                |          |                 |                     | #<nl, vu>          
|                 |                |          |                 |                     | #<nl, vu, cs>      
|                 |                |          |                 |                     | #<nl, vu, cs, ftp>  |

+: 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 administrational 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
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.)
Name spaces in DNS

- Hierarchical structure - one or more components or labels *separated by periods* (.)
- A subtree is called as a domain; a path name to its root node is called a domain name;
  - [www.utsa.edu](http://www.utsa.edu) -- domain name
- Allows aliases such as [www.utsa.edu](http://www.utsa.edu) → pandora.cs.utsa.edu
## 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 <code>&lt;preference, host&gt;</code> pairs</td>
</tr>
<tr>
<td>TXT</td>
<td>Text string</td>
<td>Arbitrary text</td>
</tr>
</tbody>
</table>
**DNS resource records: Example**

Database for the domain `dcs.qmw.ac.uk`

<table>
<thead>
<tr>
<th>Domain name</th>
<th>Time to live</th>
<th>class</th>
<th>type</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1D</td>
<td>IN</td>
<td>NS</td>
<td>dns0</td>
</tr>
<tr>
<td></td>
<td>1D</td>
<td>IN</td>
<td>NS</td>
<td>dns1</td>
</tr>
<tr>
<td></td>
<td>1D</td>
<td>IN</td>
<td>NS</td>
<td>cancer.ucs.ed.ac.uk</td>
</tr>
<tr>
<td></td>
<td>1D</td>
<td>IN</td>
<td>MX</td>
<td>mail1.qmw.ac.uk</td>
</tr>
<tr>
<td></td>
<td>1D</td>
<td>IN</td>
<td>MX</td>
<td>mail2.qmw.ac.uk</td>
</tr>
<tr>
<td>www</td>
<td>1D</td>
<td>IN</td>
<td>CNAME</td>
<td>copper</td>
</tr>
<tr>
<td>copper</td>
<td>1D</td>
<td>IN</td>
<td>A</td>
<td>138.37.88.248</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
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 clients (resolvers)

- Resolvers are usually implemented as library routines (e.g., `gethostbyname`).
- The request is formatted into a DNS record.
- DNS servers use a well-known port.
- A request-reply protocol is used.
- The resolver times out and resends if it doesn’t receive a response in a specified time.
DNS name resolution

- Domain name $\rightarrow$ 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.

authoritative path to lookup:
jeans-pc.dcs.qmw.ac.uk
DNS name resolution with cache

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

IP: alpha.qmw.ac.uk

IP: dns0.dcs.qmw.ac.uk
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!
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 and 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>Abbr.</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, 130.37.24.6, 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>

*Multiple-valued attribute*
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.
- Collection of all directory entries is called Directory Information Base (DIB).

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Abbr.</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, 130.37.24.6, 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>

- Each record is **uniquely** named by using **naming attributes** in the record (e.g., first five in the above record).
- Each naming attribute is called **relative distinguished name** (RDN).
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>zephyr</td>
</tr>
<tr>
<td>Host_Address</td>
<td>137.37.20.10</td>
</tr>
</tbody>
</table>

```
List all main servers at Vrije search("&(C = NL) (O = Vrije Universiteit) (OU = *) (CN = Main server)")
```
Hierarchical implementation: LDAP (3)

- Clients called **Directory User Agent (DUA)**, similar to name resolver.
- LDAP server known as **Directory Service Agent (DSA)** maintains DIT and looks up entries based on attr.
- In case of a large scale directory, DIT is partitioned and distribute across several DSAs.
- Implementation of LDAP is similar to DNS, but LDAP provides more advanced lookup operations.
Hierarchical implementation: LDAP (4)

- Simple DUA interface to X.500 (see extra slides)
- LDAP runs over TCP/IP
- Uses textual encoding
- Provides secure access through authentication
- Other directory services have implemented it
- See RFC 2251 [Wahl et al. 1997]

LDAP different from DNS:
- Facilities for search through a DIB
LDAP Evolution

- University of Michigan added to LDAP servers the capability of accessing own database.
- Use of LDAP databases became widespread.
- Schemes were developed for registering changes and exchanging deltas between LDAP servers.
- In 1996 three engineers from U of Michigan joined Netscape. 40 companies (w/o Microsoft) announced support of LDAP as the standard for directory services.
- Core specifications for LDAPv3 was published as IETF RFCs 2251-2256.
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