Naming

Outline
- Names Entities
- Flat Naming
- Structured Naming
- Attribute-Based Naming

Naming

Essence: Names are used to denote entities in a distributed system.
- To operate on an entity, we need to access it at an access point (address).
- Note: A location-independent name for an entity E, is independent from the addresses of the access points offered by E.

Naming Entities

- Names, identifiers, and addresses
- Name resolution
- Name space implementation

Names are valuable!

Identifiers

- Pure name: A name that has no meaning at all; it is just a random string. Pure names can be used for comparison only.
- Identifier: A name having the following 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 (prohibits reusing an identifier)
- Observation: An identifier need not necessarily be a pure name, i.e., it may have content.
Outline

- Names Entities
- Flat Naming
- Structured Naming
- Attribute-Based Naming

Flat Naming

- **Problem:** Given an essentially unstructured name (e.g., an identifier), how can we locate its associated access point?
  - Simple solutions (broadcasting)
  - Home-based approaches
  - Distributed Hash Tables (structured P2P)
  - Hierarchical location service

Simple Solutions

- **Broadcasting:** Simply broadcast the ID, requesting the entity to return its current address.
  - Can never scale beyond local-area networks (think of ARP/RARP)
  - Requires all processes to listen to incoming location requests
- **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
  - Increased network latency at dereferencing
  - Essential to have separate chain reduction mechanisms

Home-Based Approaches (1/2)

- **Single-tiered scheme:** Let a home keep track of where the entity is:
  - An entity’s home address is registered at a naming service
  - The home registers the foreign address of the entity
  - Clients always contact the home first, and then continues with the foreign location

Home-Based Approaches (2/2)

- **Two-tiered scheme:** Keep track of visiting entities:
  - Check local visitor register first
  - Fall back to home location if local lookup fails
- **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
  - Poor geographical scalability (the entity may be next to the client)
- **Question:** How can we solve the “permanent move” problem?
- **Answer:** register the home at a traditional naming service and to let a client first look up the location of the home

Distributed Hash Tables

- **Example:** Consider the organization of many nodes into a logical ring (Chord)
  - 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).
- **Solution:** Let node $id$ keep track of $succ(id)$ and start linear search along the ring.
Each node \( p \) maintains a \textit{finger table} \( FT_p \) with at most \( m \) entries:

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

Note: \( FT_p[i] \) 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

\[
q = FT_p[j] \leq k < FT_p[j + 1]
\]

If \( p < k < FT_p[1] \), the request is also forwarded to \( FT_p[1] \).

Exploiting Network Proximity

- **Problem**: The logical organization of nodes in the overlay may lead to erratic message transfers in the underlying Internet: node \( k \) and node \( \text{succ}(k + 1) \) maybe very far apart.
- **Topology-aware node assignment**: When assigning an ID to a node, make sure that nodes close in the ID space are also close in the network. \textit{Can be very difficult}.
- **Proximity routing**: Maintain more than one possible successor, and forward to the closest.
  - Example: in Chord \( FT_p[0] \) points to first node in \( \text{INT} = [p + 2^0 - 1, p + 2^0 - 1] \). 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.

Hierarchical Location Services (HLS)

- **Basic idea**: Build a large-scale search tree for which the underlying network is divided into hierarchical domains. Each domain is represented by a separate directory node.

HLS: Tree Organization

- The address of an entity is stored in a leaf node, or in an intermediate node
- Intermediate nodes contain a pointer to a child if and only if the subtree rooted at the child stores an address of the entity
- The root knows about all entities

HLS: Lookup Operation

- **Basic principles**:
  - Start lookup at local leaf node
  - If node knows about the entity, follow downward pointer, otherwise go one level up
  - Upward lookup always stops at root
**Names Entities**

- Flat Naming
- Structured Naming
- Attribute-Based Naming

---

**Name Spaces**

- The name space is the way that names in a particular system are organized. This also defines the set of all possible names.
- Examples:
  - Phone numbers
  - Credit card numbers
  - DNS
  - Human names in the US
  - Robert Ludlum books
  - Files in UNIX, Windows
  - URLs
- Names are organized into what is commonly referred to as a name space. A name space can be represented as a labeled, directed graph with two types of nodes.

---

**Name Space (1/2)**

**Essence:** a graph in which a leaf node represents a (named) entity. A directory node is an entity that refers to other nodes.

**Note:** A directory node contains a (directory) table of (edge label, node identifier) pairs.

---

**Name Resolution**

- Looking up a name (finding the "value") is called name resolution.
- **Problem:** To resolve a name we need a directory node. How do we actually find that (initial) node?
  - **Closure mechanism** (or where to start)
    - The mechanism to select the implicit context from which to start name resolution. Examples: file systems, ZIP code, DNS
    - www.cs.vu.nl: start at a DNS name server
    - /home/steen/mbox: start at the local NFS file server (possible recursive search)
    - 0031204447784: dial a phone number
    - 130.37.24.8: route to the VU’s Web server
- **Observation:** A closure mechanism may also determine how name resolution should proceed
Name Linking (1/2)

- **Hard link**: What we have described so far as a path name: a name that is resolved by following a specific path in a naming graph from one node to another.
- **Soft link**: Allow a node \( O \) to contain a name of another node:
  - First resolve \( O \)'s name (leading to \( O \))
  - Read the content of \( O \), yielding name
  - Name resolution continues with name
- **Observations**:
  - The name resolution process determines that we read the content of a node, in particular, the name in the other node that we need to go to.
  - One way or the other, we know where and how to start name resolution given name.

Name Linking (2/2)

- **Observation**: Node \( n_5 \) has only one name

Name Space Implementation

- Name spaces always map names to something.
  - DNS maps what to what?
- Can be divided into three layers:
  - **Global layer**: Doesn't change very often.
  - **Administrational layer**: Single organization
  - **Managerial layer**: Change regularly
- **Observation**:
  - Node \( n_5 \) has only one name

Name Space Distribution

- An example partitioning of the DNS name space, including Internet-accessible files, into three layers.

Name Server Characteristics

<table>
<thead>
<tr>
<th>Item</th>
<th>Global</th>
<th>Administrational</th>
<th>Managerial</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geographical scale of network</td>
<td>World</td>
<td>Organization</td>
<td>Department</td>
</tr>
<tr>
<td>Total number of nodes</td>
<td>Few</td>
<td>Many</td>
<td>Used names</td>
</tr>
<tr>
<td>Responsiveness to lookups</td>
<td>Seconds</td>
<td>Miliseconds</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>

- A comparison between name servers for implementing nodes from a large-scale name space partitioned into a global layer, as an administrational layer, and a managerial layer.
Iterative Name Resolution

- \( \text{resolve}(\text{dir}, \{\text{name}_1, \ldots, \text{name}_K\}) \) is sent to Server0 responsible for \( \text{dir} \).
- Server0 resolves \( \text{resolve}(\text{dir}, \text{name}_1) \rightarrow \text{dir}_1 \), returning the identification (address) of Server1, which stores \( \text{dir}_1 \).
- Client sends \( \text{resolve}(\text{dir}_1, \{\text{name}_2, \ldots, \text{name}_K\}) \) to Server1, etc.

Recursive Name Resolution

- \( \text{resolve}(\text{dir}, \{\text{name}_1, \ldots, \text{name}_K\}) \) is sent to Server0 responsible for \( \text{dir} \).
- Server0 resolves \( \text{resolve}(\text{dir}, \text{name}_1) \rightarrow \text{dir}_1 \), and sends \( \text{resolve}(\text{dir}_1, \{\text{name}_2, \ldots, \text{name}_K\}) \) to Server1, which stores \( \text{dir}_1 \).
- Server0 waits for the result from Server1, and returns it to the client.

Caching in Recursive Name 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>( \text{nl} )</td>
<td>( \text{nl} )</td>
<td>( \text{nl} )</td>
<td>( \text{nl} )</td>
<td>( \text{nl} )</td>
<td>( \text{nl} )</td>
</tr>
<tr>
<td>( \text{vu} )</td>
<td>( \text{vu} )</td>
<td>( \text{vu} )</td>
<td>( \text{vu} )</td>
<td>( \text{vu} )</td>
<td>( \text{vu} )</td>
</tr>
<tr>
<td>( \text{cs} )</td>
<td>( \text{cs} )</td>
<td>( \text{cs} )</td>
<td>( \text{cs} )</td>
<td>( \text{cs} )</td>
<td>( \text{cs} )</td>
</tr>
<tr>
<td>( \text{ftp} )</td>
<td>( \text{ftp} )</td>
<td>( \text{ftp} )</td>
<td>( \text{ftp} )</td>
<td>( \text{ftp} )</td>
<td>( \text{ftp} )</td>
</tr>
</tbody>
</table>

- Recursive name resolution of \( \text{nl}, \text{vu}, \text{cs}, \text{ftp} \). Name servers cache intermediate results for subsequent lookups.

Scalability Issues (1/2)

- **Size scalability**: We need to ensure that servers can handle a large number of requests per time unit \( \Rightarrow \) 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.
- **Observation**: An important attribute of many nodes is the address where the represented entity can be contacted. Replicating nodes makes caching meaningless.

Scalability Issues (2/2)

- **Geographical scalability**: We need to ensure that the name resolution process scales across large geographical distances.
- **Problem**: By mapping nodes to servers that may, in principle, be located anywhere, we introduce an implicit location dependency in our naming scheme.

Example: Decentralized DNS

- **Basic idea**: Take a full DNS name, hash into a key \( k \), and use a DHT-based system to allow for key lookups.
- **Main drawback**: You can’t ask for all nodes in a subdomain (but very few people were doing this anyway).
- **Information in a node**: Typically what you find in a DNS record, of which there are different kinds:

<table>
<thead>
<tr>
<th>Type of record</th>
<th>Associated entity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Host</td>
<td>Contains an IP address of the host this node represents</td>
</tr>
<tr>
<td>MX</td>
<td>Domain</td>
<td>Points to a mail server to handle mail addressed to this node</td>
</tr>
<tr>
<td>SRV</td>
<td>Domain</td>
<td>Points to a server handling a specific service</td>
</tr>
<tr>
<td>NS</td>
<td>Zone</td>
<td>Points to a name server that implements the represented zone</td>
</tr>
<tr>
<td>CNAME</td>
<td>Node</td>
<td>Symbolic link with the primary name of the represented node</td>
</tr>
<tr>
<td>TXT</td>
<td>Host</td>
<td>Contains the numeric name of a host</td>
</tr>
<tr>
<td>MINFO</td>
<td>Domain</td>
<td>Contains information on the file this node represents</td>
</tr>
<tr>
<td>AT</td>
<td>Any file</td>
<td>Contains any entity-specific information considered useful</td>
</tr>
</tbody>
</table>
Outline

- Names Entities
- Flat Naming
- Structured Naming
- Attribute-Based Naming

Attribute-Based Naming

- **Observation:** In many cases, it is much more convenient to name, and look up entities by means of their attributes ⇒ traditional directory services (a.k.a., yellow pages).
- **Problem:** Lookup operations can be extremely expensive, as they require to match requested attribute values, against actual attribute values ⇒ inspect all entities (in principle).
- **Solution:** Implement basic directory service as database, and combine with traditional structured naming system.

### X.500 Directory Entry

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Abbrev.</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Country</td>
<td>C</td>
<td>CN</td>
</tr>
<tr>
<td>Locality</td>
<td>L</td>
<td>Beijing</td>
</tr>
<tr>
<td>Organization</td>
<td>O</td>
<td>Peking University</td>
</tr>
<tr>
<td>Organization/Unit</td>
<td>OU</td>
<td>School of EECS</td>
</tr>
<tr>
<td>CommonName</td>
<td>CN</td>
<td>Main server</td>
</tr>
<tr>
<td>Mail_Servers</td>
<td></td>
<td>162.105.203.25</td>
</tr>
<tr>
<td>FTP_Server</td>
<td></td>
<td>162.105.146.3</td>
</tr>
<tr>
<td>WWW_Server</td>
<td></td>
<td>162.105.203.25</td>
</tr>
</tbody>
</table>

- A simple example of a X.500 directory entry using X.500 naming conventions.

### X.500 Directory Service

- provides directory service based on a description of properties instead of a full name (e.g., yellow pages in telephone book)
- an X.500 directory entry is comparable to a resource record in DNS. Each record is made up of a collection of (attribute, value) pairs
- Collection of all entries is a Directory Information Base (DIB)
- Each naming attribute is a Relative Distinguished Name (RDN)
- RDNs, in sequence, can be used to form a Directory Information Tree (DIT)

Example: LDAP

```
<server>
    <cn=Main server> cn=Main server
    <cn=Peking University> cn=Peking University
    <cn=School of EECS> cn=School of EECS
    <cn=WWW Server> cn=WWW Server
    <cn=FTP Server> cn=FTP Server
    <cn=Mail Servers> cn=Mail Servers

    <country> CN
    <locality> Beijing
    <organization> Peking University
    <organization/ou> School of EECS

    <mailServers> 162.105.203.25
    <ftpServer> 162.105.146.3
    <wwwServer> 162.105.203.25

    <search>
        <baseDN> ou=WWW Server,ou=School of EECS
        <scope> sub
        <filter> (cn=Main server)
        <attributes> cn, sn
        <resultType> subtree
    </search>
```

*© NWDE-Peking University*
References

- Chapter 4 of [Tanenbaum, 2002]
  - book1, [Coulouri, 2005]
  - book2, [Birman, 2005]
  - book3, [Tanenbaum, 2006]
  - book4, [Tanenbaum, 2002]
  - Book5, [Xining Li, 2006]