Distributed Systems
Principles and Paradigms

Maarten van Steen

VU Amsterdam, Dept. Computer Science
steen@cs.vu.nl

Chapter 05: Naming

Version: November 13, 2012

vrije Universiteit amsterdam

Naming Entities

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

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. Access points are entities that are named by means of an address.

Note

A location-independent name for an entity *E*, is independent from the addresses of the access points offered by *E*.

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.

Naming 5.1 Naming Entities

4 / 34

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

Naming 5.2 Flat Naming

5 / 34

Simple solutions

Broadcasting

Broadcast the ID, requesting the entity to return its current address.

- Can never scale beyond local-area networks
- Requires all processes to listen to incoming location requests

Forwarding pointers

When an entity moves, it leaves behind a pointer to its next location

- Dereferencing can be made entirely transparent to clients by simply following the chain of pointers
- Update a client's reference when present location is found
- Geographical scalability problems (for which separate chain reduction mechanisms are needed):
  - Long chains are not fault tolerant
  - Increased network latency at dereferencing

Naming 5.2 Flat Naming

6 / 34

Home-based approaches

Single-tiered scheme

Let a home keep track of where the entity is:

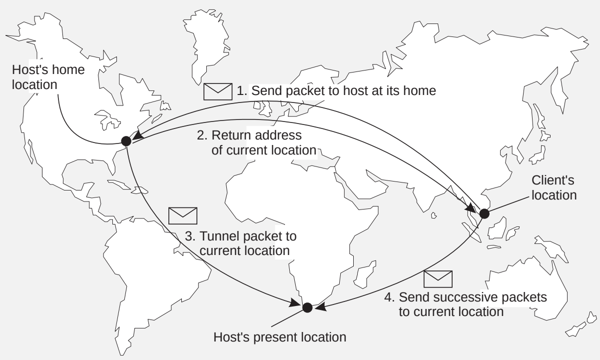
- Entity's home address registered at a naming service
- The home registers the foreign address of the entity
- Client contacts the home first, and then continues with foreign location

Naming 5.2 Flat Naming

7 / 34

Naming 5.2 Flat Naming

Home-based approaches



Naming 5.2 Flat Naming

8 / 34

Naming 5.2 Flat Naming

Home-based approaches

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

- Home address has to be supported for entity's lifetime
- Home address is fixed ⇒ unnecessary burden when the entity permanently moves
- Poor geographical scalability (entity may be next to client)

Question

How can we solve the "permanent move" problem?

Naming 5.2 Flat Naming

9 / 34

Distributed Hash Tables (DHT)

Chord

Consider the organization of 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).

Nonsolution

Let node id keep track of $succ(id)$ and start linear search along the ring.

DHTs: Finger tables

Principle

- Each node p maintains a finger table $FT_p[]$ with at most m entries:

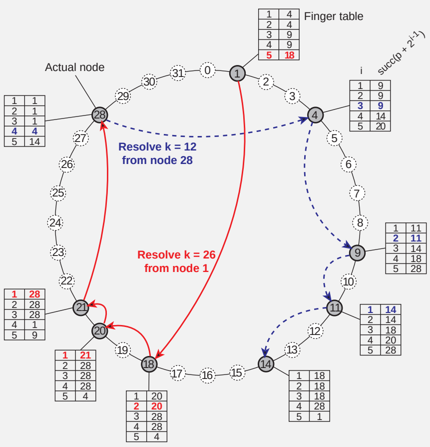
$$FT_p[i] = succ(p + 2^{i-1})$$

  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 $FT_p[1]$

DHTs: Finger tables



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 $succ(k+1)$ may be 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. Can be very difficult.

Proximity routing: Maintain more than one possible successor, and forward to the closest.
Example: in Chord $FT_p[i]$ points to first node in $INT = [p+2^{i-1}, p+2^i-1]$. 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.

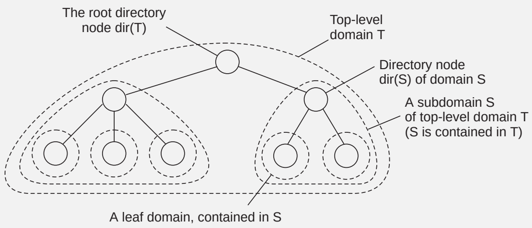
Naming 5.2 Flat Naming

13 / 34

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.



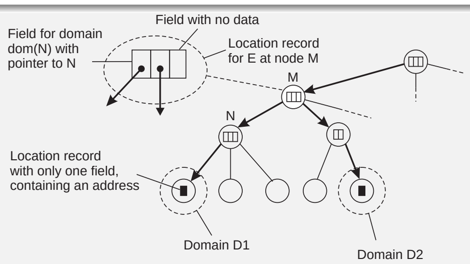
Naming 5.2 Flat Naming

14 / 34

HLS: Tree organization

Invariants

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



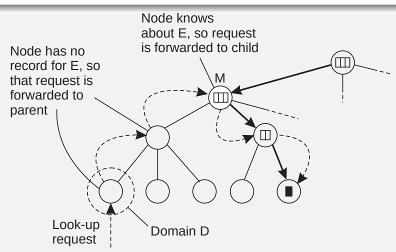
Naming 5.2 Flat Naming

15 / 34

HLS: Lookup operation

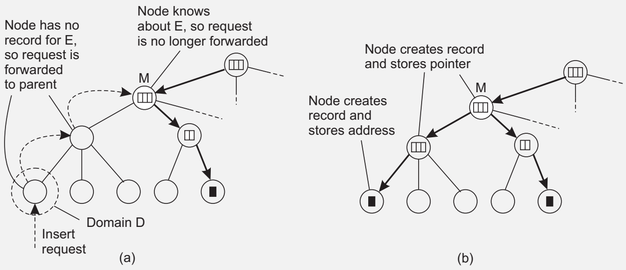
Basic principles

- Start lookup at local leaf node
- Node knows about $E \Rightarrow$ follow downward pointer, else go up
- Upward lookup always stops at root



Naming 5.2 Flat Naming

HLS: Insert operation

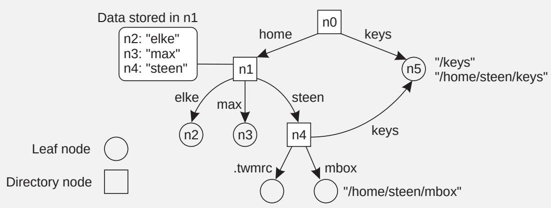


Naming 5.2 Flat Naming

Name space

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.

Naming 5.3 Structured Naming

Name space

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
- ...

Note

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

Naming 5.3 Structured Naming

19 / 34

Name resolution

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)
- *0031204447784*: dial a phone number
- *130.37.24.8*: route to the VU's Web server

Question

Why are closure mechanisms always implicit?

Naming 5.3 Structured Naming

20 / 34

Name linking

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.

Naming 5.3 Structured Naming

21 / 34

Name linking

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*

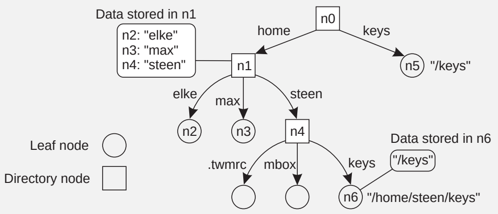
Observations

- 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*

Naming 5.3 Structured Naming

22 / 34

Name linking



Observation

Node *n5* has only one name

Naming 5.3 Structured Naming

23 / 34

Name-space implementation

Basic issue

Distribute the name resolution process as well as name space management across multiple machines, by distributing nodes of the naming graph.

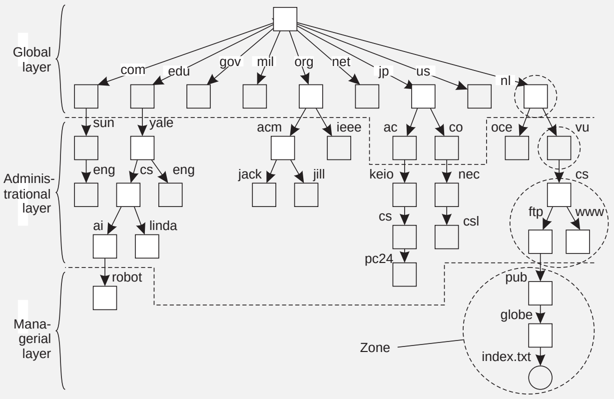
Distinguish three levels

- Global level: Consists of the high-level directory nodes. Main aspect is that these directory nodes have to be jointly managed by different administrations
- Administrational level: Contains mid-level directory nodes that can be grouped in such a way that each group can be assigned to a separate administration.
- Managerial level: Consists of low-level directory nodes within a single administration. Main issue is effectively mapping directory nodes to local name servers.

Naming 5.3 Structured Naming

24 / 34

Name-space implementation



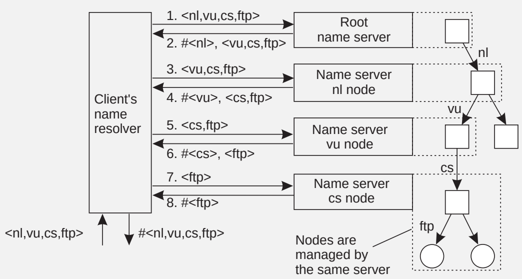
Name-space implementation

Item	Global	Administrational	Managerial
1	Worldwide	Organization	Department
2	Few	Many	Vast numbers
3	Seconds	Milliseconds	Immediate
4	Lazy	Immediate	Immediate
5	Many	None or few	None
6	Yes	Yes	Sometimes

1: Geographical scale	4: Update propagation
2: # Nodes	5: # Replicas
3: Responsiveness	6: Client-side caching?

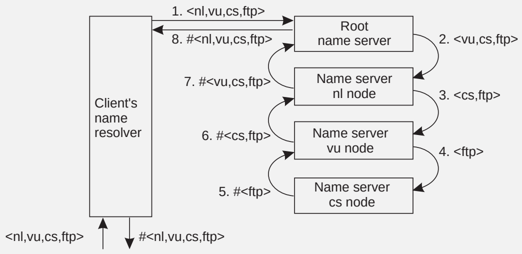
Iterative name resolution

1. *resolve(dir,[name1,...,nameK])* sent to *Server0* responsible for *dir*
2. *Server0* resolves *resolve(dir,name1)* → *dir1*, returning the identification (address) of *Server1*, which stores *dir1*.
3. Client sends *resolve(dir1,[name2,...,nameK])* to *Server1*, etc.



Recursive name resolution

1. *resolve(dir,[name1,...,nameK])* sent to *Server0* responsible for *dir*
2. *Server0* resolves *resolve(dir,name1)* → *dir1*, and sends *resolve(dir1,[name2,...,nameK])* to *Server1*, which stores *dir1*.
3. *Server0* waits for result from *Server1*, and returns it to client.



Naming 5.3 Structured Naming

Caching in recursive name resolution

Server for node	Should resolve	Looks up	Passes to child	Receives and caches	Returns to requester
cs	<ftp>	#<ftp>	—	—	#<ftp>
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>

Naming 5.3 Structured Naming

Scalability issues

Size scalability

We need to ensure that servers can 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. We can then apply extensive replication by mapping nodes to multiple servers, and start name resolution at the nearest server.

Observation

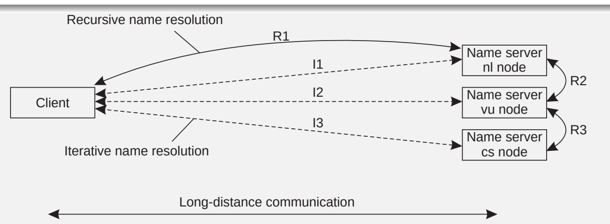
An important attribute of many nodes is the address where the represented entity can be contacted. Replicating nodes makes large-scale traditional name servers unsuitable for locating mobile entities.

Naming 5.3 Structured Naming

Scalability issues

Geographical scalability

We need to ensure that the name resolution process scales across large geographical distances.



Problem

By mapping nodes to servers that can be located anywhere, we introduce an implicit location dependency.

Naming 5.3 Structured Naming

31 / 34

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

SOA	Zone	Holds info on the represented zone
A	Host	IP addr. of host this node represents
MX	Domain	Mail server to handle mail for this node
SRV	Domain	Server handling a specific service
NS	Zone	Name server for the represented zone
CNAME	Node	Symbolic link
PTR	Host	Canonical name of a host
HINFO	Host	Info on this host
TXT	Any kind	Any info considered useful

Naming 5.3 Structured Naming

32 / 34

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 (aka 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).

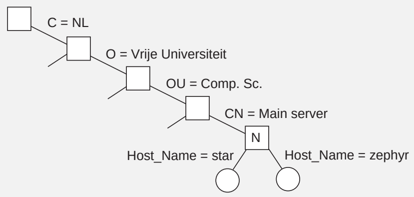
Solution

Implement basic directory service as database, and combine with traditional structured naming system.

Naming 5.4 Attribute-Based Naming

33 / 34

Example: LDAP



Attribute	Value
Country	NL
Locality	Amsterdam
Organization	Vrije Universiteit
OrganizationalUnit	Comp. Sc.
CommonName	Main server
Host_Name	star
Host_Address	192.31.231.42

Attribute	Value
Country	NL
Locality	Amsterdam
Organization	Vrije Universiteit
OrganizationalUnit	Comp. Sc.
CommonName	Main server
Host_Name	zephyr
Host_Address	137.37.20.10

```
answer =
search("&(C = NL) (O = Vrije Universiteit) (OU = *) (CN = Main server)")
```


Naming 5.4 Attribute-Based Naming