Principles of Distributed Systems

inft-3507

Dr. J.Burns

ADA University

Autumn 2025

Section 6: Naming

Ref: https://www.distributed-systems.net/index.php/books/ds4/

Names, identifiers, and addresses

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 of 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 some specific properties

1. An identifier refers to at most one entity.
2. Each entity is referred to by at most one identifier.
3. An identifier always refers to the same entity (i.e., it is never reused).

Observation

An identifier need not necessarily be a pure name, i.e., it may have content.

Flat naming

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

Address Resolution Protocol (ARP)

To find out which MAC address is associated with an IP address, broadcast the query “who has this IP address”?

Forwarding pointers

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

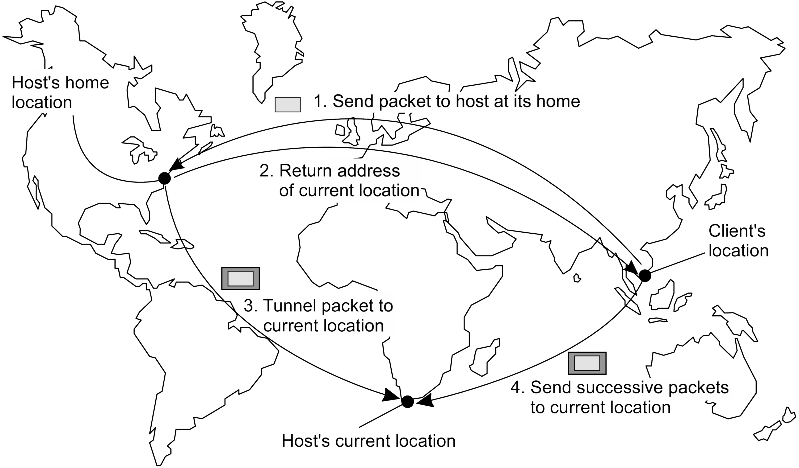
- Dereferencing can be made entirely transparent to clients by simply following the chain of pointers
- Update a client's reference when present location is found
- Geographical scalability problems (for which separate chain reduction mechanisms are needed):
    - Long chains are not fault tolerant
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

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

The principle of mobile IP



Home-based approaches

Problems with home-based approaches

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

Note

Permanent moves may be tackled with another level of naming (DNS)

Illustrative: 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 $succ(k)$).

Nonsolution

Let each node keep track of its neighbor and start linear search along the ring.

Notation

We will speak of node p as the node have identifier p

Chord 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: the i-th entry points to the first node succeeding p by at least 2^{i-1}.

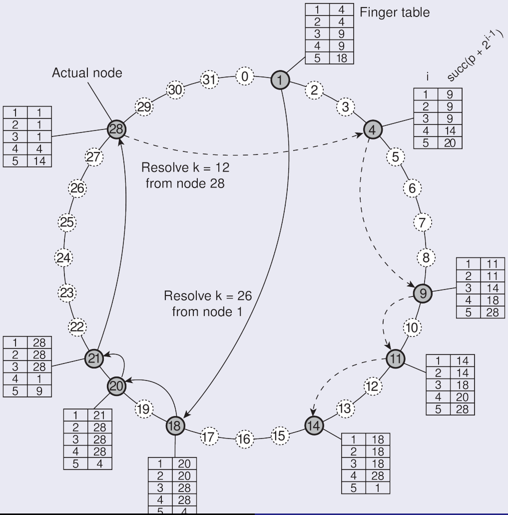
- To look up a key k, node p forwards the request to node with index j satisfying

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

- If $p < k < FT_p[1]$, the request is also forwarded to $FT_p[1]$

Chord lookup example

Resolving key 26 from node *1* and key *12* from node *28*



Exploiting network proximity

Problem

The logical organization of nodes in the overlay may lead to erratic message transfers in the underlying Internet: node p and node $succ(p+1)$ may be very far apart.

Solutions

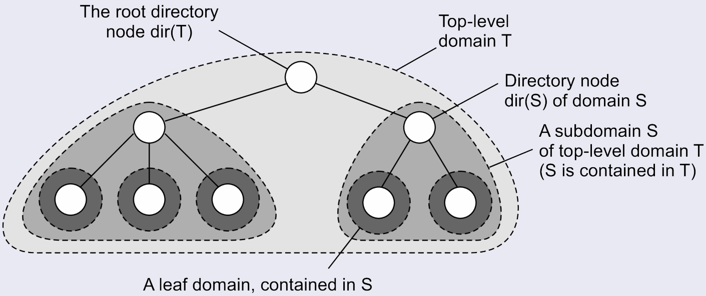
- Topology-aware node assignment: When assigning an ID to a node, make sure that nodes close in the ID space are also close in the network. Can be very difficult.
- Proximity routing: Maintain more than one possible successor, and forward to the closest. Example: in Chord $FT_p[i]$ points to first node 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.

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.

Principle

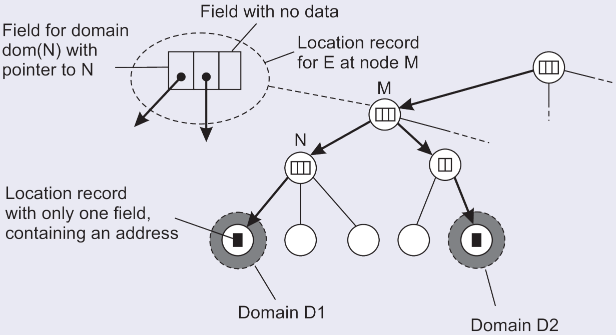


HLS: Tree organization

Invariants

- Address of entity E is stored in a leaf or 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

Storing information of an entity having two addresses in different leaf domains

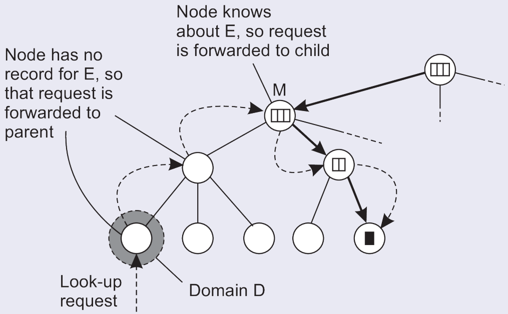


HLS: Lookup operation

Basic principles

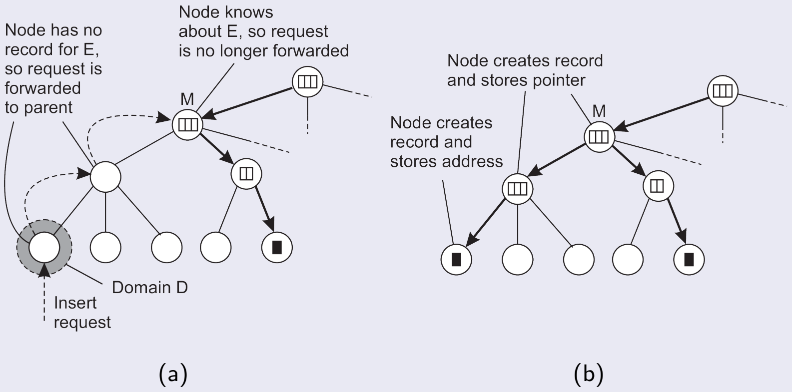
- Start lookup at local leaf node
- Node knows about $E \Rightarrow$ follow downward pointer, else go up
- Upward lookup always stops at root

Looking up a location



HLS: Insert operation

(a) An insert request is forwarded to the first node that knows about entity E. (b) A chain of forwarding pointers to the leaf node is created



Can an HLS scale?

Observation

A design flaw seems to be that the root node needs to keep track of all identifiers ⇒ make a distinction between a logical design and its physical implementation.

Notation

- Assume there are a total of N physical hosts $\{H_1, H_2, \ldots, H_N\}$. Each host is capable of running one or more location servers.
- $D_k(A)$ denotes the domain at level k that contains address A; $k = 0$ denotes the root domain.
- $LS_k(E, A)$ denotes the unique location server in $D_k(A)$ responsible for keeping track of entity E.

Can an HLS scale?

Basic idea for scaling

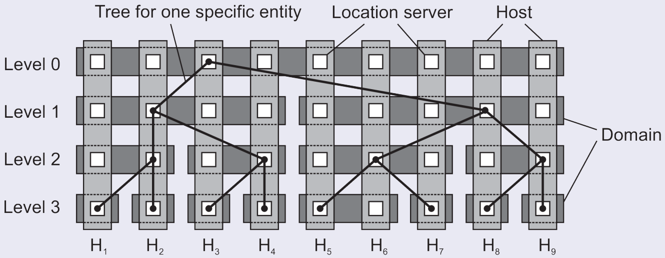
- Choose different physical servers for the logical name servers on a per-entity basis
  - (at root level, but also intermediate)
- Implement a mapping of entities to physical servers such that the load of storing records will be distributed

Can an HLS scale?

Solution

- $\mathbf{D}_k = \{D_{k,1}, D_{k,2}, \ldots, D_{k,N_k}\}$ denotes the N_k domains at level k
- Note: $N_0 = |\mathbf{D}_0| = 1$.
- For each level k, the set of hosts is partitioned into N_k subsets, with each host running a location server representing exactly one of the domains $D_{k,i}$ from $\mathbf{D}_k$.

Principle of distributing logical location servers



Security in flat naming

Basics

Without special measures, we need to trust that the name-resolution process to return what is associated with a flat name. Two approaches to follow:

- Secure the identifier-to-entity association
- Secure the name-resolution process

Self-certifying names

Use a value derived from the associated entity and make it (part of) the flat name:

- $id(entity) = hash(data\ associated\ with\ the\ entity)$

when dealing with read-only entities, otherwise

- $id(entity) = public\ key(entity)$

in which case additional data is returned, such as a verifiable digital signature.

Securing the name-resolution process

Much more involved: discussion deferred until discussing secure DNS.

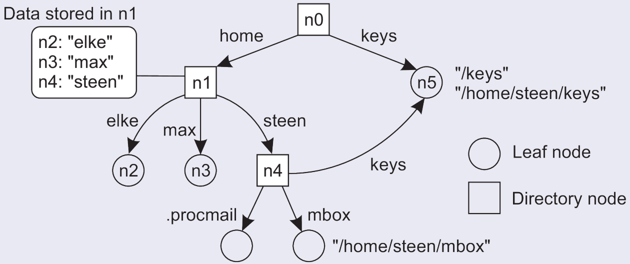
Structured naming

Name space

Naming graph

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

A general naming graph with a single root node



Note

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

Name space

We can easily store all kinds of attributes in a node

- 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 *(identifier, label)* pairs.

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.distributed-systems.net: start at a DNS name server
- /home/maarten/mbox: start at the local NFS file server (possible recursive search)
- 0031 20 598 7784: dial a phone number
- 77.167.55.6: route message to a specific IP address

Note

You cannot have an explicit closure mechanism – how would you start?

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.

Soft link: Allow a node N to contain a name of another node

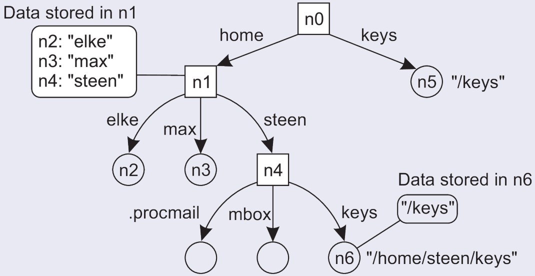
- First resolve N's name (leading to N)
- Read the content of N, yielding *name*
- Name resolution continues with *name*

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

The concept of a symbolic link explained in a naming graph



Observation

Node *n5* has only one name

DNS

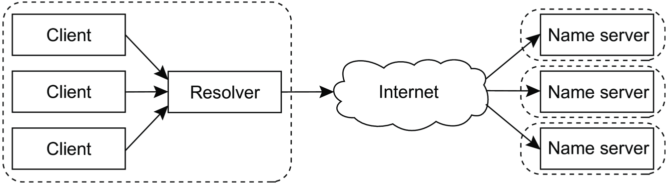
Essence

- Hierarchically organized name space with each node having exactly one incoming edge $\Rightarrow$ edge label $=$ node label.
- domain: a subtree
- domain name: a path name to a domain's root node.

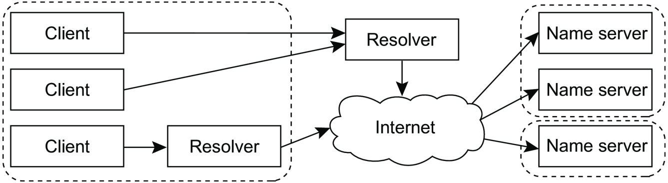
Information in a node

Type	**Refers to**	**Description**
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

Modern DNS



The traditional organization of the implementation of DNS



The modern organization of DNS

Secure DNS

Basic approach

Resource records of the same type are grouped into a signed set, per zone. Examples:

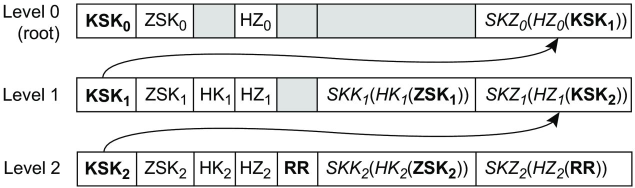
- A set with all the IPv4 addresses of a zone
- A set with all the IPv6 addresses of a zone
- A set with the name servers of a zone

The public key associated with the secret key used for signing a set of resource records is added to a zone, called a zone-signing key.

Trusting the signatures

- All zone-signing keys are grouped again into a separate set, which is signed using another secret key. The public key of the latter is the key-signing key.
- The hash of the key-signing key is stored at, and signed by, the parent zone

Secure DNS



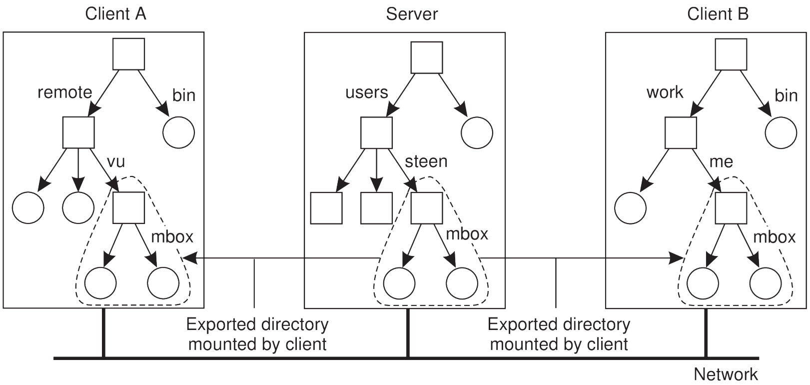
Building a trust chain

- Consider a single set of resource records RR, hashed with HZ_k and signed with SKZ_k
- SZK_k has associated public key ZSK_k
- (Set of) ZSK_k is hashed with HK_k and signed with SKK_k
- SKK_k has associated public key KSK_k

A client can verify signature $SKZ_2(HZ_2(RR))$ by checking

$$ZSK_2(SKZ_2(HZ_2(RR))) \stackrel{?}{=} HZ_2(RR)$$

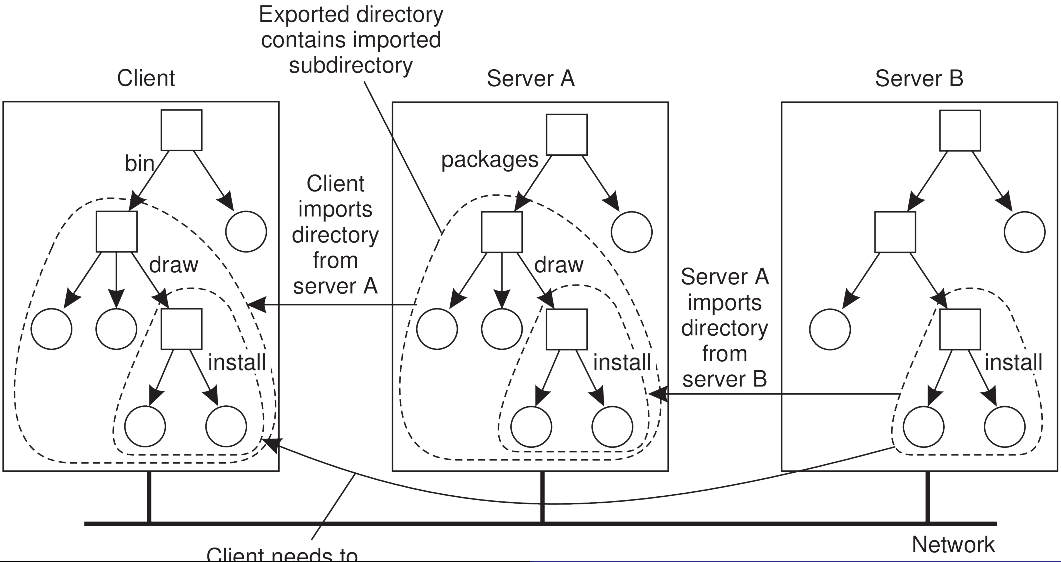
Naming in NFS



Observation

A server may export (a part of) its filesystem, which can then be imported by different clients by mounting. Note that different clients will have different (nonsharable) namespaces!

Mounting nested directories



Attribute-based naming

Attribute-based naming

Observation

In many cases, it is much more convenient to name, and look up entities through their attributes $\Rightarrow$ traditional directory services (aka yellow pages).

Problem

Lookup operations can be expensive, as they require matching requested attribute values, against actual attribute values $\Rightarrow$ inspect all entities (in principle).

Implementing directory services

Solution for scalable searching

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

Lightweight Directory Access Protocol (LDAP)

Each directory entry consists of (*attribute, value*) pairs, and is uniquely named to ease lookups.

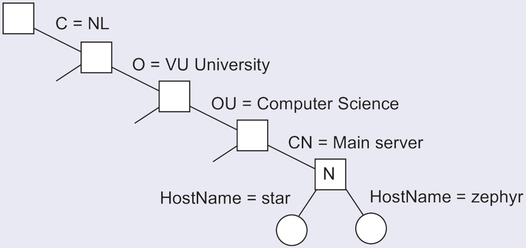
Attribute	**Abbr.**	**Value**
Country	*C*	NL
Locality	*L*	Amsterdam
Organization	*O*	VU University
OrganizationalUnit	*OU*	Computer Science
CommonName	*CN*	Main server
Mail_Servers	–	137.37.20.3, 130.37.24.6, 137.37.20.10
FTP_Server	–	130.37.20.20
WWW_Server	–	130.37.20.20

LDAP

Essence

- Directory Information Base: collection of all directory entries in an LDAP service.
- Each record is uniquely named as a sequence of naming attributes (called Relative Distinguished Name), so that it can be looked up.
- Directory Information Tree: the naming graph of an LDAP directory service; each node represents a directory entry.

Part of a directory information tree



LDAP

Two directory entries having *HostName* as RDN

Attribute	**Value**	**Attribute**	**Value**
Locality	*Amsterdam*	*Locality*	*Amsterdam*
Organization	*VU University*	*Organization*	*VU University*
OrganizationalUnit	*Computer Science*	*OrganizationalUnit*	*Computer Science*
CommonName	*Main server*	*CommonName*	*Main server*
HostName	*star*	*HostName*	*zephyr*
HostAddress	*192.31.231.42*	*HostAddress*	*137.37.20.10*

Result of `search("(C=NL)(O=VU University)(OU=*)(CN=Main server)")`

Distributed index

Basic idea

- Assume a set of attributes $\{a^1, \ldots, a^N\}$
- Each attribute a^k takes values from a set R^k
- For each attribute a^k associate a set $\mathbf{S^k} = \{S_1^k, \ldots, S_{n_k}^k\}$ of n_k servers
- Global mapping F: $F(a^k, v) = S_j^k$ with $S_j^k \in \mathbf{S^k}$ and $v \in R^k$

Observation

If $L(a^k, v)$ is set of keys returned by $F(a^k, v)$, then a query can be formulated as a logical expression, e.g.,

$$\left(F(a^1, v^1) \wedge F(a^2, v^2)\right) \vee F(a^3, v^3)$$

which can be processed by the client by constructing the set

$$\left(L(a^1, v^1) \cap L(a^2, v^2)\right) \cup L(a^3, v^3)$$

Drawbacks of distributed index

Quite a few

- A query involving k attributes requires contacting k servers
- Imagine looking up "$lastName = Smith \wedge firstName = Pheriby$": the client may need to process many files as there are so many people named "Smith."
- No (easy) support for range queries, such as "$price = [1000 - 2500]$."

Summary

Summary

In the *Naming* section of the lecture notes we have discussed the following topics

1. Names, identifiers, and addresses
2. Flat Naming
3. Structured Naming
4. Attribute-based Naming