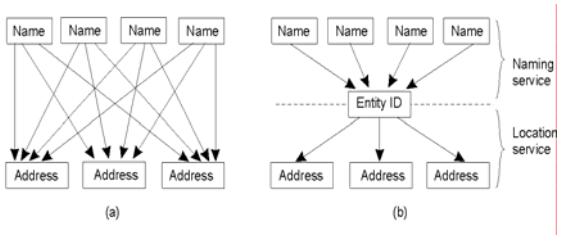
Naming

Tanenbaum Ch. 5
Distributed Software Systems
CS 707

Naming

- A name in a distributed system is a string of bits or characters that is used to refer to an entity
- Types of names:
  - Address: an access point of an entry
  - Identifiers: a name that uniquely identifies an entity
    - An identifer refers to at most one entity
    - Each entity is referred to by at most one identifier
    - An identifier always refers to the same entity
  - Human friendly names
  - Location-independent name: a name that is independent from its address
- Name-to-address binding
- Mobile vs. non-mobile

Naming versus Locating Entities



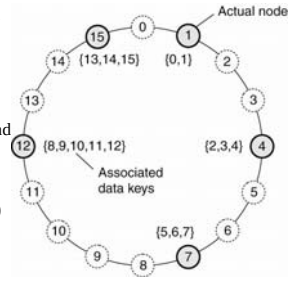
a) Direct, single level mapping between names and addresses.
b) T-level mapping using identities.

Flat Name-to-Address in a LAN

- Broadcast – a message containing identifier is sent to all machines; each machine checks to see if it has associated entity. Reply contains the address of entity.
- Multicast – message sent to a registered subset
- Simple but doesn't scale well

Distributed Hash Tables

- A Hash Table is can be used to do fast table lookups based on an identifier
- A Distributed Hash Table is a decentralized approach to lookups.
- Example: Chord
  - m-bit (typically m = 128 or 160) identifiers associated with nodes and with entities.
  - An entity with identifier k will be associated with node n with the smallest n >= k – known as succ(k)
  - Problem: How to locate succ(k) given k?



Distributed Hash Tables (2)

Each node p has a finger table of at most m entries.

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

At node 1:

$FT_1[1] = succ(1 + 2^0) = 4$

$FT_1[2] = succ(1 + 2^1) = 4$

$FT_1[3] = succ(1 + 2^2) = 9$

$FT_1[4] = succ(1 + 2^3) = 9$

$FT_1[5] = succ(1 + 2^4) = 18$

At node 20:

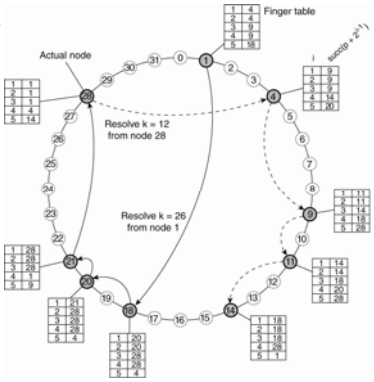
$FT_{20}[1] = succ(20 + 2^0) = 21$

$FT_{20}[2] = succ(20 + 2^1) = 28$

$FT_{20}[3] = succ(20 + 2^2) = 28$

$FT_{20}[4] = succ(20 + 2^3) = 28$

$FT_{20}[5] = succ(20 + 2^4) = 4$



Distributed Hash Tables (3)

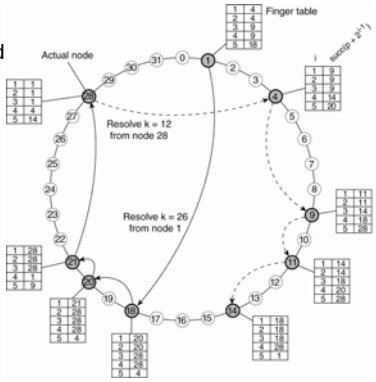
To lookup non-local key *k*, node *n* will forward request to node *q* with index *j* in *n*'s finger table where:

$q = FT_n[j] <= k < FT_n[j+1]$

Resolve k = 26 from node 1

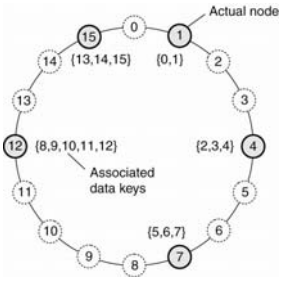
Resolve k = 12 from node 28

Search time is O(log n) for n = number of nodes in system



Distributed Hash Tables (4)

- Issues:
  - Nodes leaving and joining
    - Keeping the finger tables up to date
    - Keeping successor and predecessor information up to dates
  - Base algorithms make no assumptions about underlying topology (i.e. where the nodes are physically located).



Hierarchical Approaches (1)

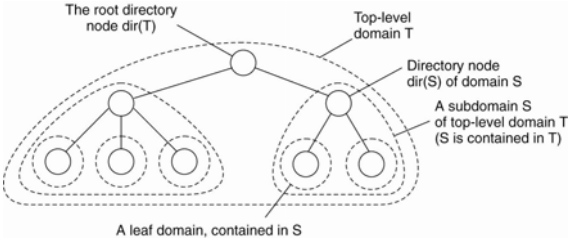


Figure 5-5.

Hierarchical organization of a location service into domains, each having an associated directory node. The top-level domain represents the entire network. Each domain is divided into multiple smaller domains until each leaf domain (typically small domain such as a LAN) is created.

Hierarchical Approaches (2)

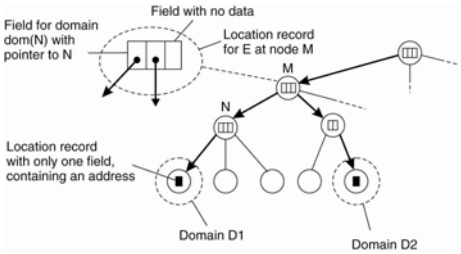


Figure 5-6.

Each entity has a record in each ancestor in the domain tree (all the way to the root). An entity having multiple addresses would typically have multiple pointers in different leaf domains.

Hierarchical Approaches (3)

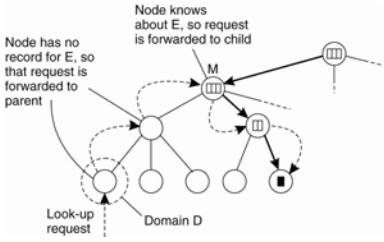


Figure 5-7.

Looking up a location in a hierarchically organized location service requires search upwards in the tree until a record of that entity is encountered. Once found, the pointers are followed down to the location of the entity.

Hierarchical Approaches (4)

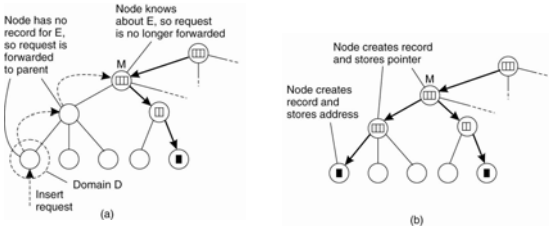


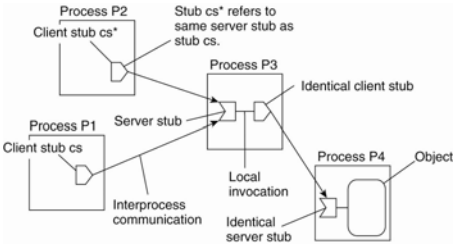
Figure 5-8

Inserting another copy of entry E results in an upward search to find the first referenced to E. Once found, this node can be updated, along with all of the nodes down the tree.

Deletions can be handled similarly, with removals from the leaf up to the root of the tree.

Mobile entities (1)

When an entity can be **mobile**, need a way to find current location given an old location. Forwarding pointers build a linked list using (client stub, server stub) information (fig 5-1 from text). When an entity moves from location A to location B, it leaves in A a reference to the new address.



Mobile entities (2)

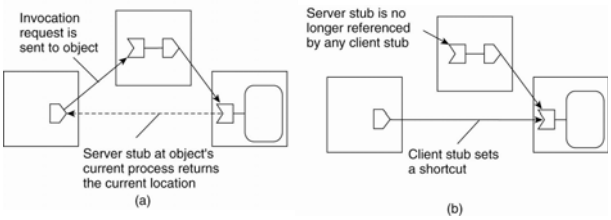


Figure 5-2.

Redirecting a forwarding pointer by storing a shortcut in a client stub. The goal is to keep the chains short to make communication faster and to reduce the chances of a break in the chain. Eventually, when the server stub is no longer referenced, it can be removed.

Mobile Entities (3)

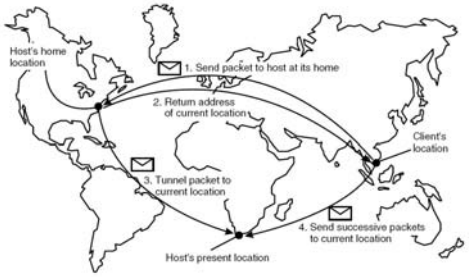


Figure 5-3.

A home location always keeps track of the current location of an entity. If the entity is not at the home location, the message is fowarded to the current location and the current location is sent to the client for further messages. [Mobile IP]

Name Spaces (1)

A Name Space is an organization mechanism for a group of names.

Terminology

- absolute vs. relative path names
- global name vs. local name
- name resolution – process of looking up a name

Name Spaces (2)

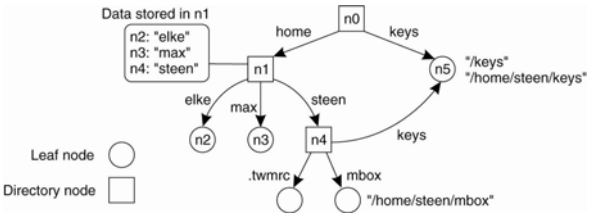


Figure 5-9. File systems are an example of a namespace that can be represented with a general naming graph with a single root node.

Linking and Mounting: Distributed Name Spaces

Information required to mount a foreign name space in a distributed system

- The name of an access protocol.
- The name of the server.
- The name of the mounting point in the foreign name space.

Ex: NFS

nfs://flits.cs.vu.nl//home/steen

protocol server mounting point

Linking and Mounting in NFS

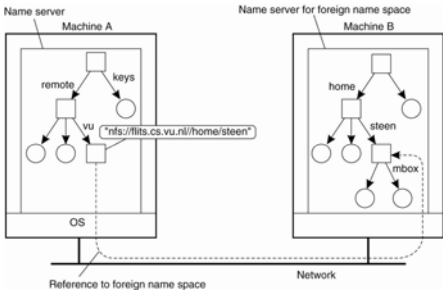


Figure 5-12. Mounting remote name spaces through a specific access protocol. Name */remove/vu/mbox* is resolved by starting at the root of the client and ending at the remote location.

Implementing Name Spaces

- Naming service: a service that allows users and processes to add, remove and lookup names
- Name spaces for large-scale widely distributed systems are typically organized hierarchically
- Three layers are used to implement such distributed name spaces
  - Global layer: root node and children
  - Administrational layer: directory nodes within a single organization
  - Managerial layer: typically a layer that is changed frequently

Name Space Distribution (1)

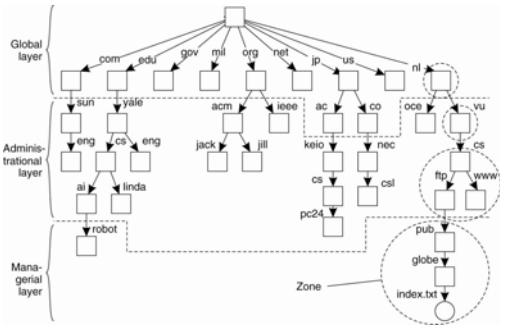


Figure 5-13. An example partitioning of the DNS name space, including Internet-accessible files, into three layers.

Name Space Distribution (2)

Item	Global	Administrational	Managerial
Geographical scale of network	Worldwide	Organization	Department
Total number of nodes	Few	Many	Vast numbers
Responsiveness to lookups	Seconds	Milliseconds	Immediate
Update propagation	Lazy	Immediate	Immediate
Number of replicas	Many	None or few	None
Is client-side caching applied?	Yes	Yes	Sometimes

Figure 5-14. A comparison between name servers for implementing nodes from a large-scale name space partitioned into a global layer, an administrational layer, and a managerial layer.

Name Resolution

- The process of looking up a name is called name resolution
- Two techniques: Iterative and Recursive
  - Iterative: repeated calls, each resulting in partial resolution, until address is resolved
  - Recursive: single call from client. Each server resolves path and makes requests to other servers to resolve.
- Recursive name resolution puts a higher performance demand on each name server
  - too high for global layer name servers
- Advantages of recusive name resolution
  - caching is more effective
  - communication costs may be reduced

Implementation of Name Resolution (1)

ftp://ftp.cs.vu.nl/pub/globe/index.html

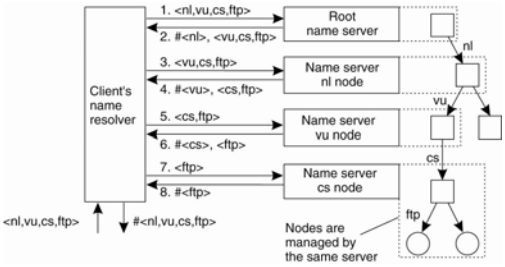


Figure 5-15. The principle of iterative name resolution.

Implementation of Name Resolution (2)

ftp://ftp.cs.vu.nl/pub/globe/index.html

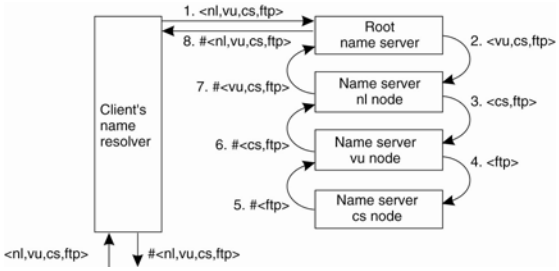


Figure 5-16. The principle of recursive name resolution.

Implementation of Name Resolution (3)

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>

Figure 5-17. Recursive name resolution of *<nl, vu, cs, ftp>*. Name servers cache intermediate results for subsequent lookups.

Example: Domain Name System (DNS)

- Host name to IP address translation
- Name space organized as a hierarchical rooted tree
  - Name space divided into non-overlapping zones
- Name servers implement the global and administrational layers
  - Managerial layer not part of DNS
  - Each zone has a name server which is typically replicated
  - Updates take place at the primary name server for a zone
    - Secondary name servers requires the primary name server to transfer its contents

DNS Implementation (1)

Name	Record type	Record value
cs.vu.nl.	SOA	star.cs.vu.nl. hostmaster.cs.vu.nl. 2005092900 7200 3600 2419200 3600
cs.vu.nl.	TXT	"Vrije Universiteit - Math. & Comp. Sc."
cs.vu.nl.	MX	1 mail.few.vu.nl.
cs.vu.nl.	NS	ns.vu.nl.
cs.vu.nl.	NS	top.cs.vu.nl.
cs.vu.nl.	NS	solo.cs.vu.nl.
cs.vu.nl.	NS	star.cs.vu.nl.
star.cs.vu.nl.	A	130.37.24.6
star.cs.vu.nl.	A	192.31.231.42
star.cs.vu.nl.	MX	1 star.cs.vu.nl.
star.cs.vu.nl.	MX	666 zephyr.cs.vu.nl.
star.cs.vu.nl.	HINFO	"Sun" "Unix"
zephyr.cs.vu.nl.	A	130.37.20.10
zephyr.cs.vu.nl.	MX	1 zephyr.cs.vu.nl.
zephyr.cs.vu.nl.	MX	2 tornado.cs.vu.nl.
zephyr.cs.vu.nl.	HINFO	"Sun" "Unix"

Figure 5-20. Each DNS node has a collection of resource records. An excerpt from the DNS database for the zone *cs.vu.nl*.

The DNS Name Space

Type of record	Associated entity	Description
SOA	Zone	Holds information on the represented zone
A	Host	Contains an IP address of the host this node represents
MX	Domain	Refers to a mail server to handle mail addressed to this node
SRV	Domain	Refers to a server handling a specific service
NS	Zone	Refers to a name server that implements the represented zone
CNAME	Node	Symbolic link with the primary name of the represented node
PTR	Host	Contains the canonical name of a host
HINFO	Host	Holds information on the host this node represents
TXT	Any kind	Contains any entity-specific information considered useful

Figure 5-19. The most important types of resource records forming the contents of nodes in the DNS name space.

DNS Implementation (2)

ftp.cs.vu.nl.	CNAME	soling.cs.vu.nl.
www.cs.vu.nl.	CNAME	soling.cs.vu.nl.
soling.cs.vu.nl.	A	130.37.20.20
soling.cs.vu.nl.	MX	1 soling.cs.vu.nl.
soling.cs.vu.nl.	MX	666 zephyr.cs.vu.nl.
soling.cs.vu.nl.	HINFO	"Sun" "Unix"
vucs-das1.cs.vu.nl.	PTR	0.198.37.130.in-addr.arpa.
vucs-das1.cs.vu.nl.	A	130.37.198.0
inkt.cs.vu.nl.	HINFO	"OCE" "Proprietary"
inkt.cs.vu.nl.	A	192.168.4.3
pen.cs.vu.nl.	HINFO	"OCE" "Proprietary"
pen.cs.vu.nl.	A	192.168.4.2
localhost.cs.vu.nl.	A	127.0.0.1

Figure 5-20. An excerpt from the DNS database for the zone *cs.vu.nl*.

Attribute Based Services

- In this approach, each entity is assumed to have an associated collection of named attributes.
- A *directory service* is a naming service in which a client can look for an entity based on a description of properties instead of a full name
- DNS-like approaches = white pages while a directory service = yellow pages.
- Ex: LDAP (derived from OSI's X.500 directory service), UDDI (Universal Directory and discovery integration)

Hierarchical Implementations: LDAP (1)

Attribute	Abbr.	Value
Country	C	NL
Locality	L	Amsterdam
Organization	O	Vrije Universiteit
OrganizationalUnit	OU	Comp. Sc.
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

Figure 5-22. An LDAP directory service consists of a collection of records (directory entries). Each record has *(attribute,value)* pairs, where each attribute has an associated type. The table above is a simple example of an LDAP directory entry using LDAP naming conventions.

Hierarchical Implementations: LDAP (2)

Attribute	Abbr.	Value
Country	C	NL
Locality	L	Amsterdam
Organization	O	Vrije Universiteit
OrganizationalUnit	OU	Comp. Sc.
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

The collection of all directory entries in an LDAP directory service is called a directory information base (DIB). Each record in a DIB is uniquely named by a series of *naming attributes* in each record (red box).

Hierarchical Implementations: LDAP (3)

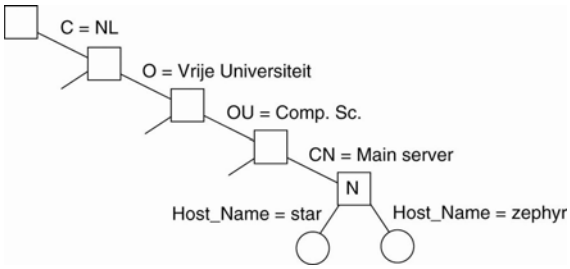


Figure 5-23. The globally unique names can be formed into a Directory Information Tree (DIT). The DIT forms the naming graph of a LDAP direcotry service. (a) Part of a directory information tree.

Hierarchical Implementations: LDAP (4)

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

(b)

Figure 5-23. (b) Two directory entries having *Host_Name* as RDN.

UDDI: Universal description, discovery, and integration

- Registry system with a XML/SOAP standards based framework for describing, discovering and managing web services
- Uses standard taxonomies to describe businesses, services, and service types
- "The UDDI Business Registry is intended to serve as a global, all-inclusive listing of businesses and their services. The UDDI Business Registry does not contain detailed specifications about business services. It points to other sources that contain the service specifications."
- private registries also possible

UDDI began as ad hoc consortium; now housed at OASIS (www.uddi.org).

WSDL: Web Services Definition Language

- Formal language that serves as the IDL to support RPC-based communication on the web.
- WSDL descriptions contain definitions of the interfaces provided by a service (data types, locaiont, …).
- This description can be translated into client and server side stubs.
- Simple Object Access Protocol (SOAP) is typically how communication in web services is done.

Web Services Fundamentals

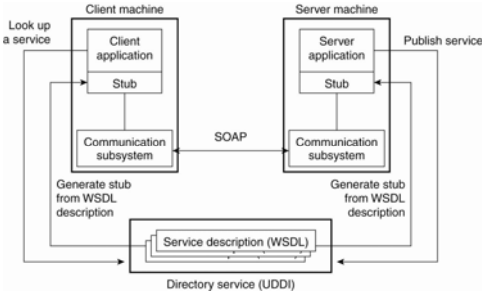


Figure 12-4. The principle of a Web service.

Summary

- Entities in distributed systems typically have names that are used to locate them
- Flat names
  - broadcast/multicast – easy but not scalable
  - distributed hash tables – more scalable, ex: Chord
  - hierarchical approaches – impose tree structure to reduce search time
  - mobile entities – additional mechanisms needed like pointers or home locations
- Structured names
  - ex: file systems
  - name resolution – tied to structure, iterative vs. recursive
  - hierarchical names spaces – ex: DNS
- Attribute based Name Resolution
  - look for entities based on properties
  - ex: LDAP, UDDI