6. Naming Systems

Sistemes Distribuïts en Xarxa (SDX)
Facultat d’Informàtica de Barcelona (FIB)
Universitat Politècnica de Catalunya (UPC)
2024/2025 Q2

Contents

- **Introduction to naming**
- Flat naming
- Structured naming
- Attribute-based naming

Introduction to naming

- Names play a very important role: identify entities, refer and share resources, …
- **Name**: String of bits or characters that is used to refer to an entity (e.g. your name)
  - Human-friendly names: Character string that is understandable by a human
- **Entity**: Anything in a distributed system
  - e.g. hosts, printers, files, processes, users, etc.
- **Address**: Special kind of name that refers to an access point for operating on an entity
  - e.g. phone number, server endpoint (IP+port)

Introduction to naming

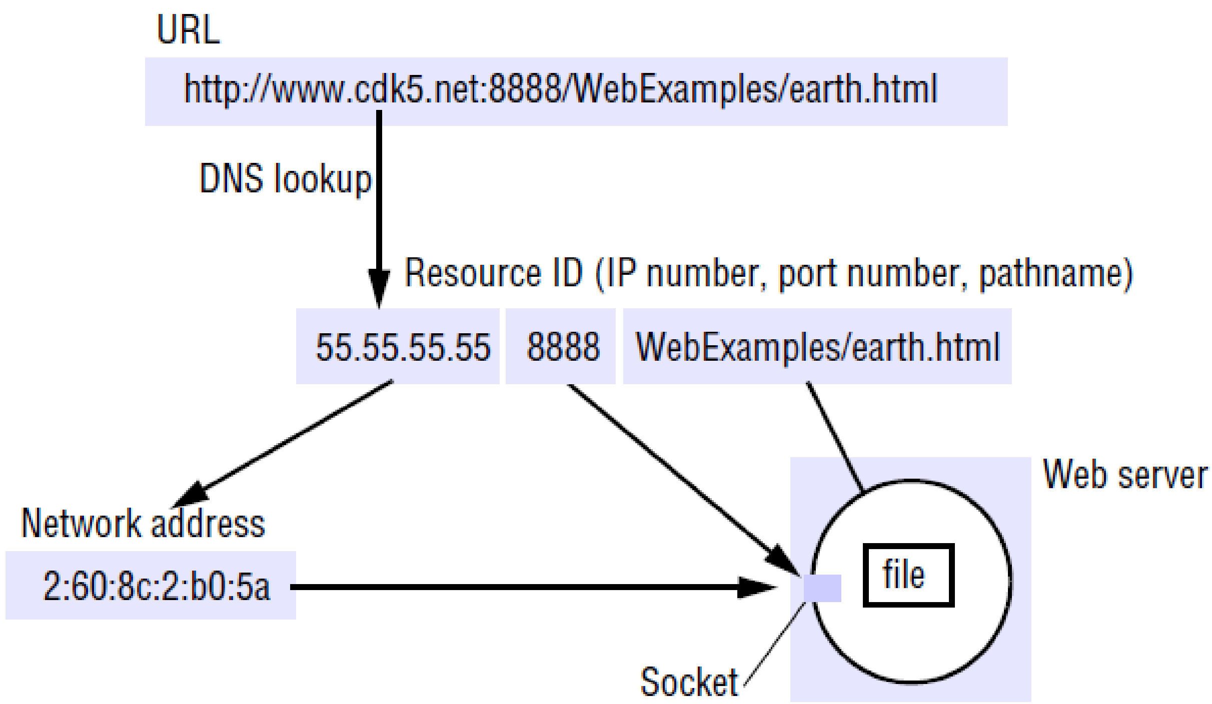
- Easier to refer an entity through a **location** (i.e. address) **independent** name
  - Access points can change over time (mobility)
  - An entity can offer more than one access point
    - e.g. several phone numbers
- **Identifier**: Name that uniquely identifies an entity (e.g. your DNI). Has these properties:
  1. Each 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)

Introduction to naming

- Names are organized into **name spaces**
  - Collection of all valid names recognized by a particular name service
- **Naming systems** allow <u>resolving names to addresses</u> so that we can operate on entities
  - Sometimes, resolution involves routing requests to a specific entity by using its name/identifier
    - Boundary between resolution and routing can be blurry
  - Can be: A) Flat
    B) Structured
    C) Attribute-based

Introduction to naming

- An address in one level could be a name (that must resolved) in an underlying one
  - e.g. Domain name - IP address - MAC address



Contents

- Introduction to naming
- **Flat naming**
- Structured naming
- Attribute-based naming

Flat naming

- Identifiers are just random bit strings
  - a.k.a. unstructured or flat names
  - Name does not contain any information on how to locate the address of its associated entity
- Problem: Given a flat name, how can we locate its associated address?
  1. Simple solutions for LANs
  2. Home-based approaches
  3. Distributed Hash Tables

Simple solutions

(*) Alternatively, only to a restricted group of nodes through multicast

A. Broadcasting

– Broadcast the identifier to all* the nodes; the one hosting the entity replies with its current address

– e.g. ARP protocol (IP to MAC)

↑ Simple and performs well in local-area networks

↓ Cannot scale to wide-area networks

- Network bandwidth wasted by broadcast messages
- Many nodes are disrupted by requests they cannot reply
- All nodes must be listening to incoming requests

Simple solutions

B. Forwarding pointers

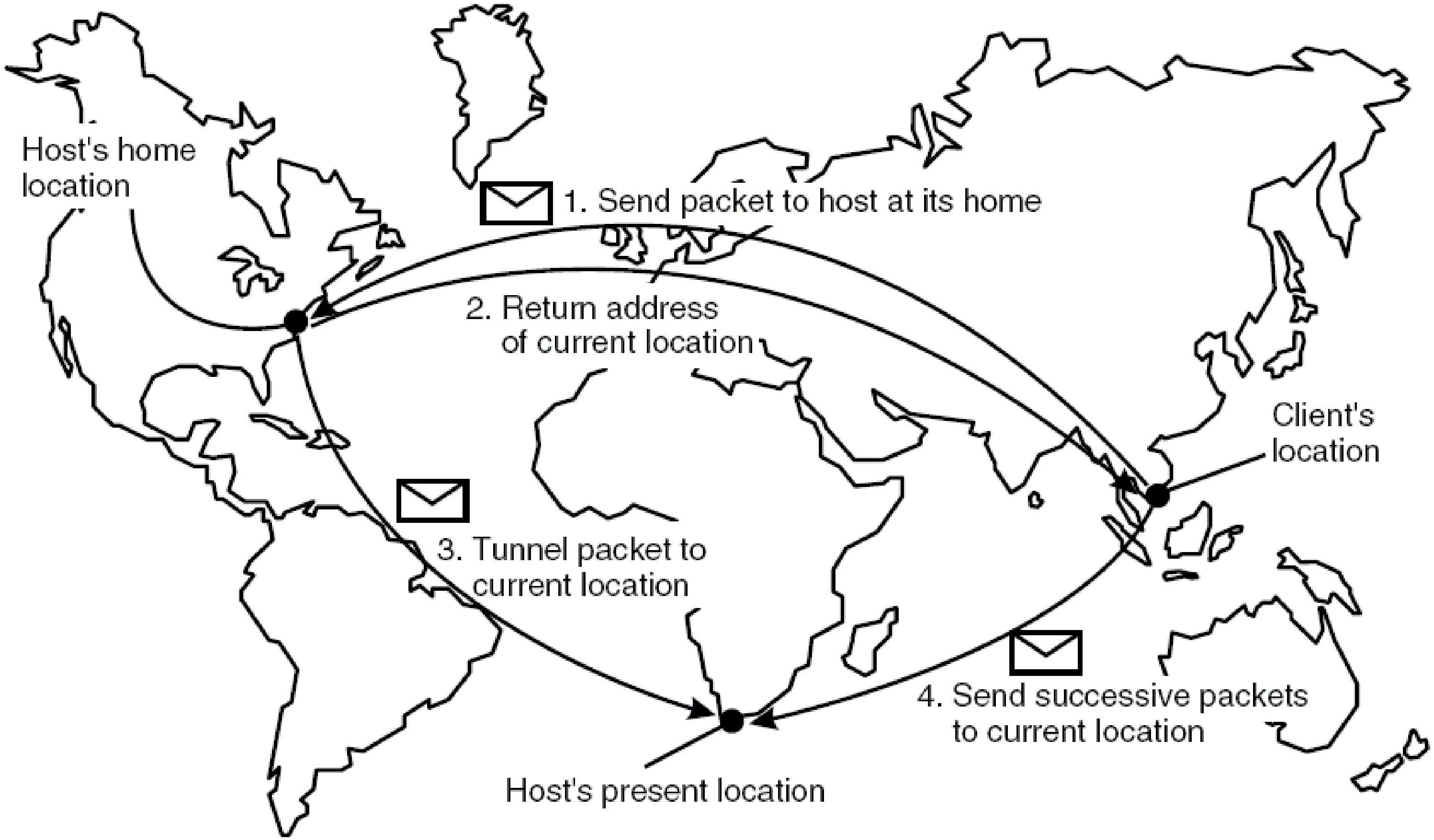
- – Intended for locating **mobile** entities
- – Each time an entity moves, it leaves behind a reference to its new location
- ↑ Mobility transparency
  - Dereferencing by simply following the chain of pointers
- ↓ Long chains make dereferencing expensive (increased latency)
- ↓ Intermediary nodes have to maintain pointers as long as needed
- ↓ Vulnerability to broken links (not fault tolerant)

Home-based approaches

- Simple solutions are not intended for large-scale networks
- Introduce a **home location** which keeps track of the current location of an entity (i.e. foreign location)
  - Home location registered at a naming service
  - It is often the place where an entity was created
- Clients always contacts the home first, which replies with the foreign location

Home-based approaches

- e.g. Mobile IP

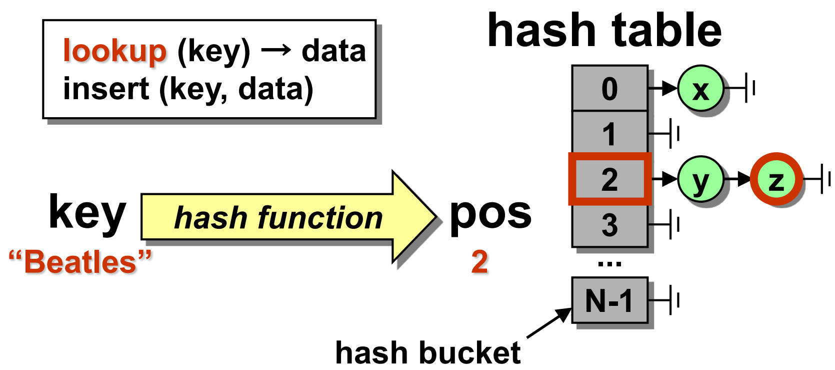


Home-based approaches

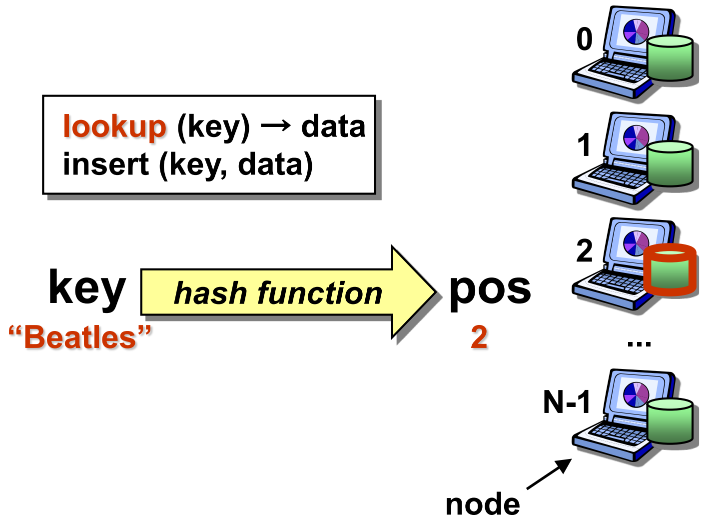
- ↑ Support **mobile** entities in large-scale networks
- ↓ Home location has to be supported as long as the entity exists
- ↓ Home address is fixed → unneeded burden when the entity permanently moves to another location
  - Alternative: move home along with the entity and update the home's location at the naming service
- ↓ Increased latency (for the first packet)

Distributed Hash Tables (DHT)

- A hash table associates data with keys
  - Key is hashed to find bucket in hash table



- In a DHT, the nodes are the hash buckets
  - Key is hashed to find responsible peer node



DHT: Components

1. Keyspace definition
   – Assign a m-bit identifier to each node
   – Assign a m-bit key to each data item
2. Keyspace partitioning
   – Split keyspace among participating nodes
   a) Simple hashing
      • Having N nodes, we could calculate the node for a given key as “key % N”
      ↓ When a node fails ($N \rightarrow N-1$) or joins ($N \rightarrow N+1$), nearly the entire keyspace must be remapped

DHT: Components

2. Keyspace partitioning
   b) Consistent hashing
      - Map keys to nodes with ‘closest’ IDs
        – We have to define an abstract notion of the <u>distance</u> between keys
      - Removal or addition of one node changes only the set of keys owned by the nodes with adjacent IDs
        – Only K/N keys need to be remapped on average
3. Overlay network
   – To connect the nodes, allowing them to find the owner of any given key in the keyspace

DHT: Usage

A. Index a file with given filename and data
   1. Hash the filename to obtain the key K
   2. Send message **put(K, data)** to any node in the DHT
   3. Message is routed through the overlay network until it reaches the single node <u>responsible</u> for key K
   4. That node then stores the key and the data

B. Retrieve the contents of a file
   1. Hash the filename to obtain the key K
   2. Send message **get(K)** to any node in the DHT
   3. Message is routed through the overlay network until it reaches the single node <u>responsible</u> for key K
   4. That node then replies with the stored data

DHT: Example implementation

- e.g. Chord (further details in Lesson 9)
  1. m bit identifiers ($0 \ldots 2^m-1$) organized in a ring
     - Use hash function to map nodes and data to identifiers
       - Key ID = SHA-1(data)
       - Node ID = SHA-1(IP address)
  2. A key K is stored in the **successor** node of K (node with next higher ID)
  3. Resolve a key K to the successor node of K (i.e. lookup) in O(log N) time using finger tables
     - Every node knows m other nodes
     - Forward the request to the largest node in finger table not exceeding the key ID

Contents

- Introduction to naming
- Flat naming
- **Structured naming**
- Attribute-based naming

Structured naming

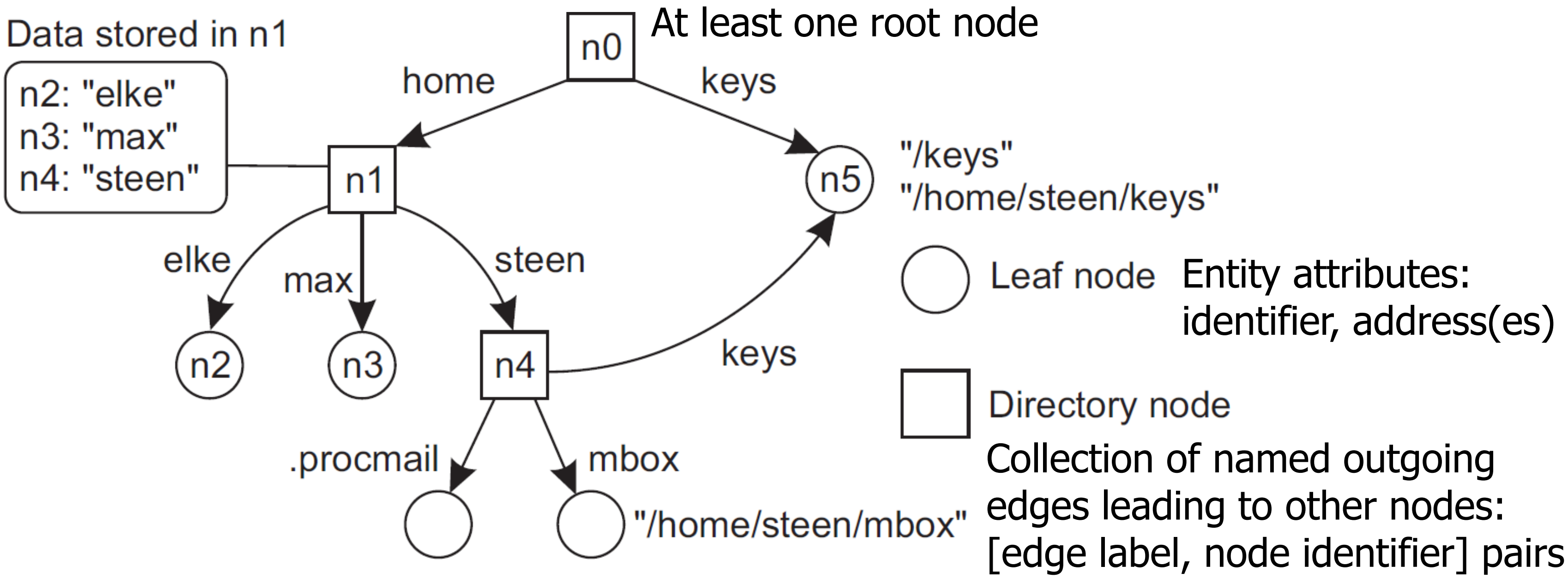
- Flat names are easy for machines (e.g. to compare two names), but not so for humans
- Instead, use structured names composed of simple, human-readable names
  - e.g. file naming: /home/steen/mbox
  - e.g. naming on the Internet: www.cs.vu.nl
- Structured name space is potentially infinite
  - Size of flat name spaces is determined by the number of bits of the identifiers

Contents

- Introduction to naming
- Flat naming
- **Structured naming**
  - **Name spaces**
  - Name resolution
  - Example: DNS
- Attribute-based naming

Name spaces

- Represented by a labeled, directed graph



- Nodes are referred to by path names
- Given a path name, a name resolution retrieves the data in the node referenced by that name

Name space distribution

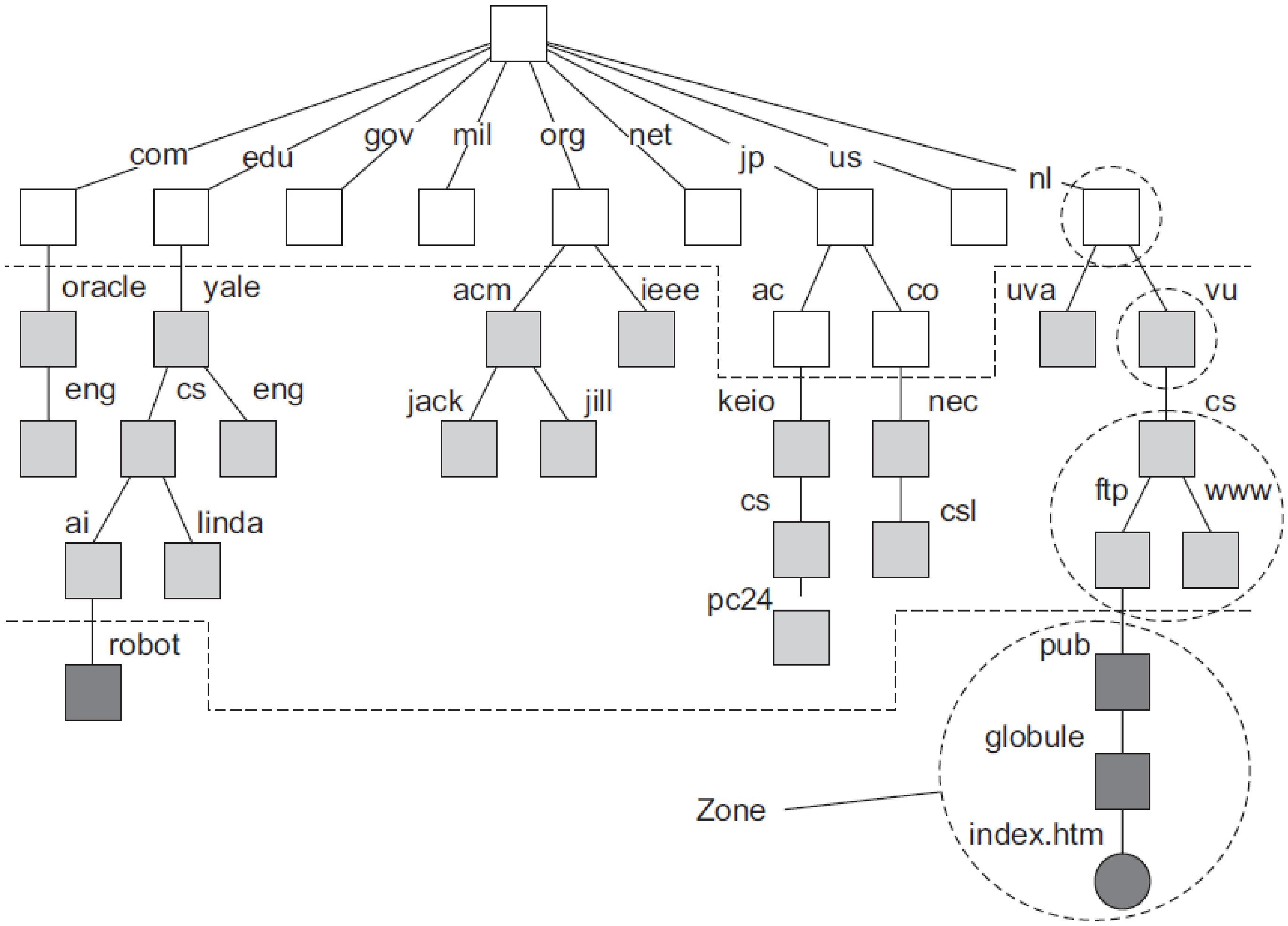
- In large-scale systems, name resolution and name space management are distributed across multiple name servers
  - ↑ Better performance and scalability
  - ↑ Different contexts can be managed by different people or organizations
- Three logical layers are distinguished by their performance, reliability, security, and administrative requirements
  - A) Global, B) Administrational, C) Managerial

Name space distribution

<u>Global layer</u>
- Highest-level nodes
- Jointly managed by different administrations
- Stable, directory tables are rarely changed

<u>Administrational layer</u>
- Mid-level directory nodes managed by a single organization
- Relatively stable

<u>Managerial layer</u>
- Low-level nodes within a single organization (e.g. hosts, files, …)
- They change frequently
- Maintained also by users



Name space distribution

- Availability and performance are met by replication
  - Map nodes to multiple servers and start name resolution at the nearest server
  - Works well when the data of node (i.e. address) hardly ever changes, as normally occurs in high-level nodes (not suitable for low-level servers and mobile entities)

Feature/Characteristic	**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

Contents

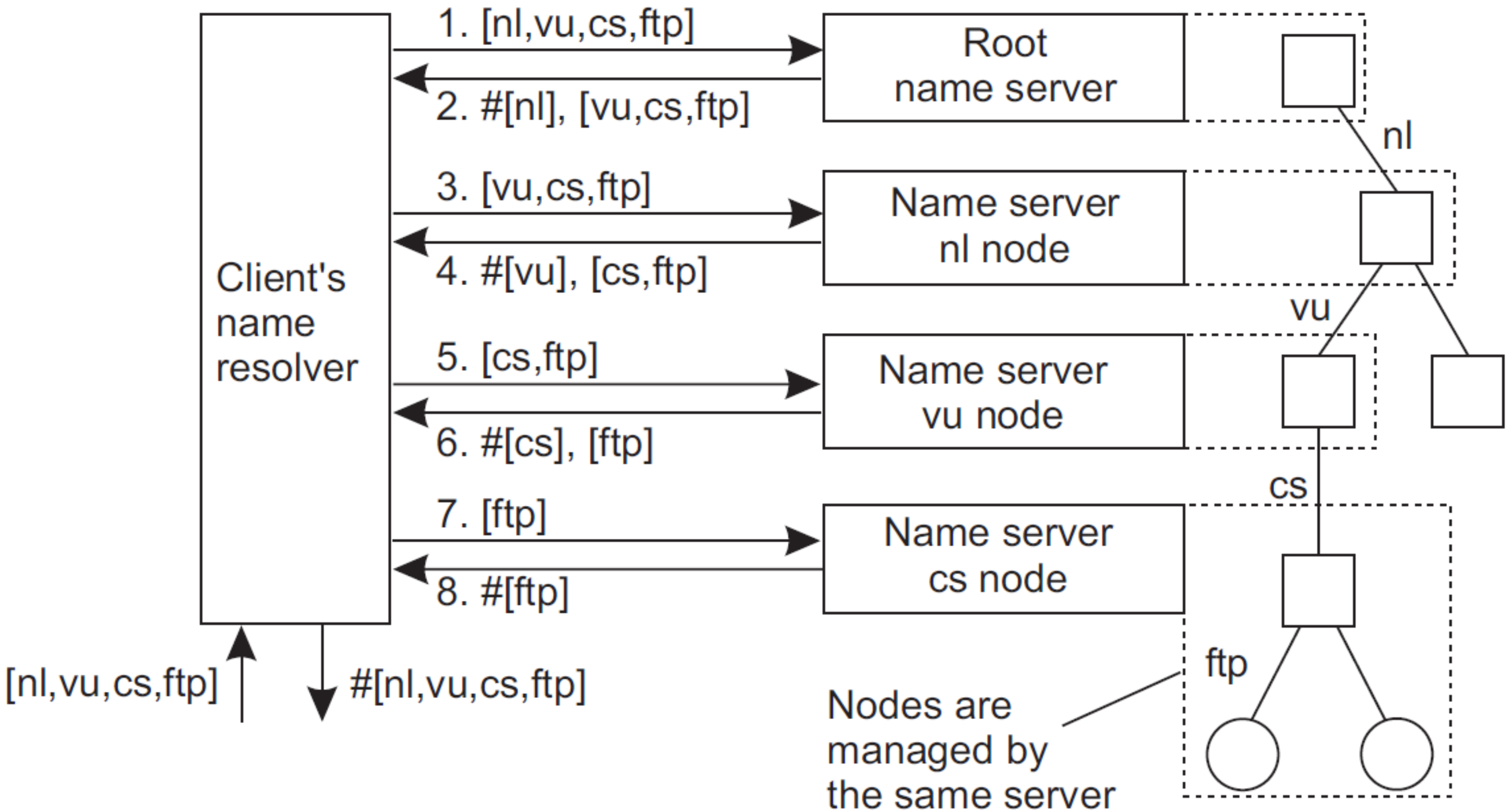
- Introduction to naming
- Flat naming
- **Structured naming**
  - Name spaces
  - **Name resolution**
  - Example: DNS
- Attribute-based naming

Name resolution

- Each client accesses a local **name resolver**
  - It drives the name resolution on behalf of the client by interacting with the name servers
- **Closure mechanism** is needed: Know <u>how and where</u> to start the resolution
  - <u>Implicit</u> way to know the initial node in a name space from which name resolution has to start
  - e.g. UNIX file system: know root inode location
  - e.g. DNS: know root name server

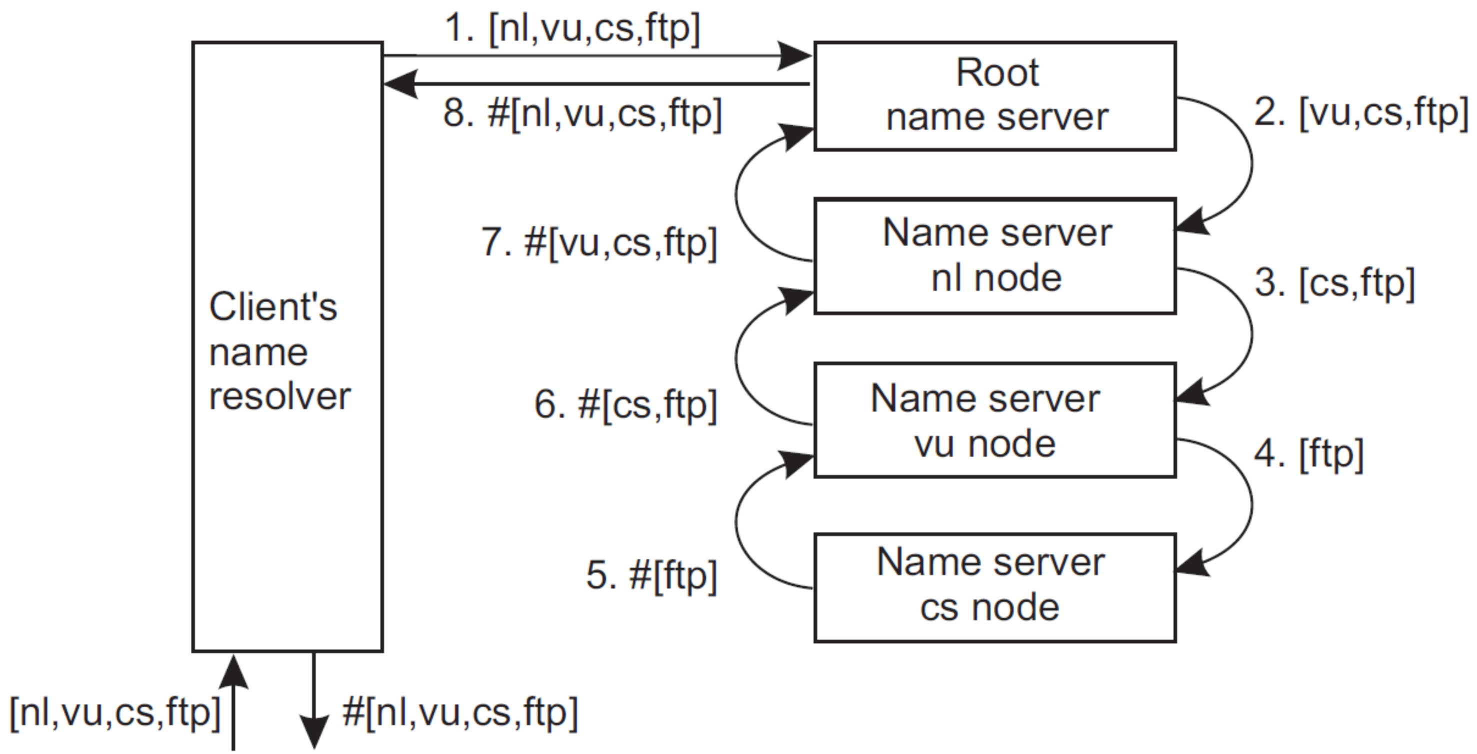
Iterative name resolution

- Name resolver queries each name server (at each layer) in an iterative fashion (it does all the work)



Recursive name resolution

- Name resolver starts the process and each server queries the next one it finds until the resolution is satisfied; results are then returned to the client



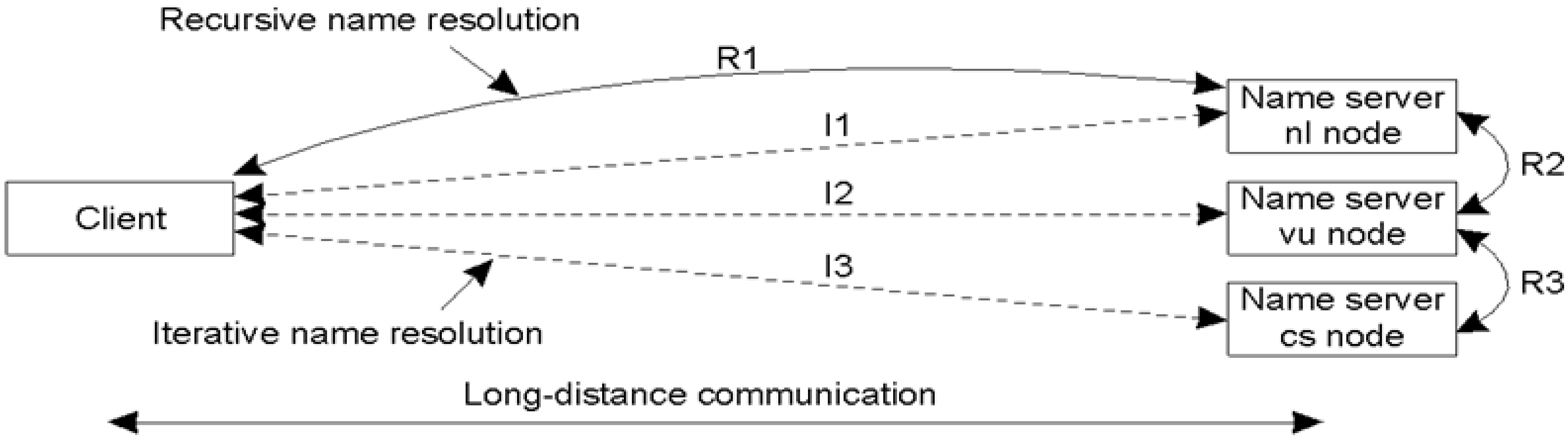
Caching in name resolution

- Caching is more effective with recursive resolution
  - Servers cache partial results for subsequent lookups
  - With iterative resolution, caching is restricted to resolver

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]

Iterative vs. recursive name resolution

- Recursive name resolution puts higher performance demand on each name server
  - Hence, name servers in the global layer of a name space support only iterative name resolution
- But provides better geographical scalability
  - Especially over long WAN links



Contents

Example: Domain Name Service (DNS)

- Primarily used for looking up IP addresses of host and mail servers in the Internet
- DNS name space
  - Hierarchically organized as a rooted tree
  - A subtree is referred to as 'naming domain'
    - A single overall administrative authority is responsible for assigning names within it (but can delegate)
  - A path name (called 'domain name') consists of 'labels' separated by '.' (highest-level on the right)
    - e.g. root:[nl, vu, cs, ftp] $\Rightarrow$ ftp.cs.vu.nl
    - Note that even geographic-sounding domain names are completely independent of their physical locations

Example: DNS

- Nodes contain a collection of 'resource records'
  - e.g. domain attributes, name & address of servers in the zone and servers managing delegated subdomains

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

Example: DNS implementation

- Divide name space in non-overlapping **zones**
- Each zone implemented by two authoritative name servers (at least) in a replicated fashion
- Works through a **primary-based** protocol
  - Updates are applied on the primary server by modifying its local master file
  - Secondary servers contact periodically with the primary to check/download zone data (pull-based)
    - This zone transfer is done typically once or twice a day
- Queries from resolver can go to any server

Example: DNS implementation

- Both clients' resolvers and name servers can **cache** data from other servers
  - ↑ Reduces network bandwidth and response time
- Cached entries have a **time-to-live** (TTL) value (decided by the authoritative server)
  - – Cached entries are provided to clients for up to this time only
  - – For queries after TTL expiration, the authoritative server is contacted to check the data
  - ↑ Gives flexibility to system administrators
    - Set large TTL when data is expected to change rarely

READING REPORT

[Vlajic12] Vlajic, N., Andrade, M., Nguyen, U.T., *The Role of DNS TTL Values in Potential DDoS Attacks: What Do the Major Banks Know About It?*, Procedia Computer Science, Vol. 10, pp. 466-473, August 2012

SEMINAR PREPARATION – Namy

[Albitz06] Albitz, P., Liu, C., *How Does DNS Work?*, DNS and BIND, 5th Edition, Chapter 2, O'Reilly Media, May 2006

Contents

Attribute-based naming

- Sometimes, it is convenient to name, and look up entities by means of their **attributes**
  - {attribute, value} pairs
- Use a **directory service**
  - Special kind of naming service in which a client can look for an entity based on a description of its properties instead of its full name
  - e.g. Lightweight Directory Access Protocol (LDAP)
    - Based on OSI's X.500 and commonly used in the Internet
  - e.g. Universal Description, Discovery & Integration (UDDI)

Contents

Directory entries

- Directory entries in LDAP are comparable to resource records in DNS
- Each entry is made up of a series of {attribute, value} pairs
  - Each attribute has an associated type
  - Single-valued and multiple-valued attributes can be distinguished
- The collection of all directory entries is called Directory Information Base (DIB)

Directory entries

- Example of a LDAP directory entry

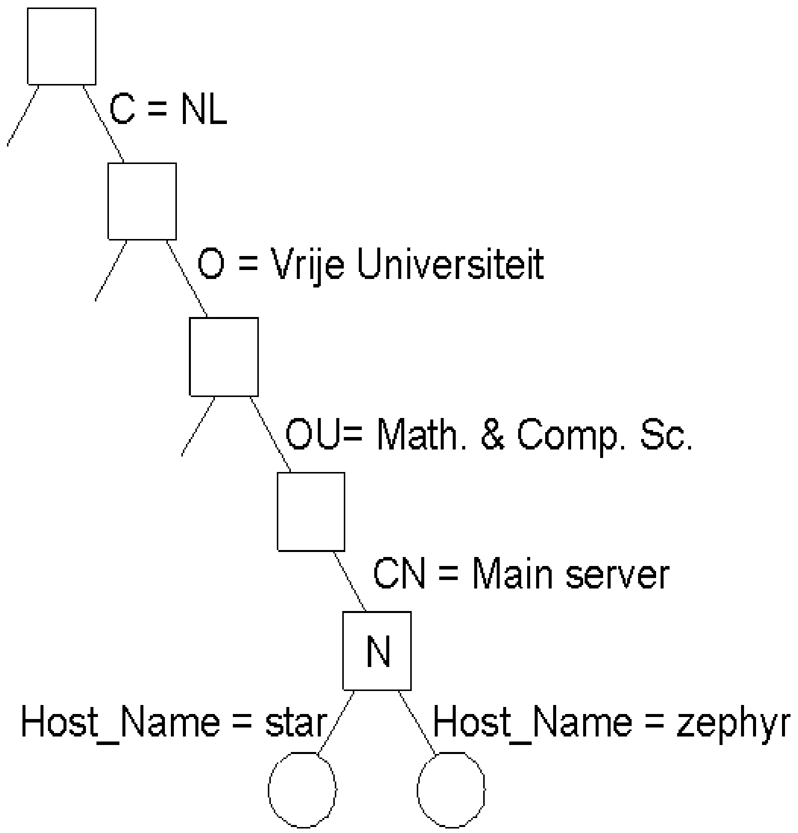
Attribute	Abbr.	Value
Country	C	NL
Locality	L	Amsterdam
Organization	O	Vrije Universiteit
OrganizationalUnit	OU	Computer Science
CommonName	CN	Main server
Mail_Servers	--	130.37.24.6, 137.37.20.3
FTP_Server	--	130.37.20.20
WWW_Server	--	130.37.20.20

Directory entries

- Each directory entry is **uniquely** named using a sequence of naming attributes
  - Naming attributes are a <u>subset</u> of all the attributes of the entry (i.e. distinguished attributes)
  - Each naming attribute is called a **Relative Distinguished Name (RDN)**
  - e.g. /C=NL/O=Vrije Universiteit/OU=Math. & Comp. Sc. $\Rightarrow$ cs.vu.nl (DNS)
- RDNs can be arranged in sequence into a hierarchy $\Rightarrow$ Directory Information Tree (DIT)
  - This is the LDAP name space

Directory Information Tree

- DIT is usually partitioned and distributed across several servers
  - Known as Directory Service Agents (DSA)
  - Similar to zones in DNS
- Clients are known as Directory User Agents (DUA)
  - Similar to name resolvers in DNS



Examples: LDAP

- LDAP provides facilities for reading entries and <u>searching</u> them through a DIB
  1. Read the attributes of an entry given its name
  2. Search for directory entries given a set of criteria that their attributes should meet
     - We can restrict the search within subtrees of the DIT
  – e.g. search("&(C=NL)(O=Vrije Universiteit)(OU=*)(CN=Main server)")
  ↓ Searching operations can be <u>expensive</u>
     - In the worst case, it requires inspecting <u>all entities</u> to match requested attribute values

Summary

- Names are used to refer to entities
  - Address: name of an entity access point
  - Identifier: one-to-one mapping to an entity
- Naming system: resolve names and identifiers to addresses
  1. Flat: Identifiers are just random bit strings
     - Name to address resolution
       - Simple solutions for LANs
       - Home-based approaches
       - DHT

Summary

2. Structured: Names organized into name spaces, which can be represented by a naming graph
   - Name resolution: looking up information stored in the node given the path name
     - Iterative vs. Recursive name resolution
   - e.g. DNS
3. Attribute-based: look up entities by means of their attributes using a directory service
   - e.g. LDAP

- Further details:
  - [Tanenbaum]: chapter 6
  - [Coulouris]: chapter 13