

UNIT 5

Naming

Distributed Systems (CAC352) — BCA III/VI

⚠ Please note

*These notes were prepared with AI assistance and **may contain errors or omissions**. Please cross-check definitions, algorithms and numerical examples against your prescribed textbook (e.g., Tanenbaum & van Steen, Distributed Systems), your class notes and your instructor before relying on them for exams.*

5.1 Names, Identifiers and Addresses

A **name** is a string of bits or characters used to refer to an **entity** (host, printer, file, user, Web page, process...). To operate on an entity we need an **access point**, itself a special entity, whose name is an **address**. An entity may have several access points and they can change (a laptop's IP address changes when it moves), so it is better not to use an address as the entity's name. A **location-independent name** is one that does not depend on where the entity is.

Identifiers

An **identifier** is a name with three properties: (1) it refers to at most one entity, (2) each entity is referred to by at most one identifier, and (3) an identifier always refers to the same entity (never reused). Identifiers are usually random bit strings that humans do not read. In contrast, **human-friendly names** (e.g., example.com) are meaningful to people.

Kind	Properties	Example
Address	Locates an access point; changes when the entity moves	IP address, MAC address
Identifier	Unique, permanent, location-independent	UUID, ISBN, ORCID
Human-friendly name	Readable; may be reused or changed	File name, domain name

Flat naming

Identifiers are unstructured (flat), so they tell nothing about the entity's location. To find the address of an entity given only its identifier we need special techniques:

- **Broadcasting and multicasting** — the id is broadcast on the LAN and the owner answers (e.g., ARP resolves an IP address to a MAC address). Does not scale beyond a LAN.

- **Forwarding pointers** — when an entity moves, it leaves behind a pointer to its next location; clients follow the chain. Chains can grow long and break, so they must be shortened.
- **Home-based approaches** — a home agent keeps track of the current location; clients contact home, which forwards (Mobile IP). Drawbacks: home must be reachable and the triangular route is inefficient.
- **Distributed hash tables (DHTs)** — e.g., Chord (below).
- **Hierarchical approaches** — a tree of domains where each directory node stores location records or pointers to sub-domains for entities in its domain; lookups go up until an entry is found, then down.

Chord lookup with finger tables

Each node n in an m -bit ring keeps a **finger table** with m entries: entry i points to $\text{succ}(n + 2^{i-1}) \bmod 2^m$. To look up key k , node n forwards the request to the finger with the largest id that still precedes k ; the search space halves at each hop, so lookup needs $O(\log N)$ hops.

Example 5.1 — Chord finger table and lookup

Ring with $m = 5$ (ids 0–31) and nodes 1, 8, 14, 21, 28. (a) Build the finger table of node 8. (b) Find the node responsible for key 26 starting the lookup at node 8.

Solution

(a) Entry $i = \text{successor}((8 + 2^{i-1}) \bmod 32)$: $i=1 \rightarrow 9 \rightarrow \text{node 14}$; $i=2 \rightarrow 10 \rightarrow \text{node 14}$; $i=3 \rightarrow 12 \rightarrow \text{node 14}$; $i=4 \rightarrow 16 \rightarrow \text{node 21}$; $i=5 \rightarrow 24 \rightarrow \text{node 28}$.

So the finger table of node 8 = [14, 14, 14, 21, 28].

(b) Key 26 is not in (8, 14], so node 8 forwards to its largest finger that precedes 26 → **node 21**.

Node 21: its successor is 28 and $26 \in (21, 28]$, so the key belongs to **node 28**. Path: $8 \rightarrow 21 \rightarrow 28$ (2 hops).

5.2 Structured Naming

Structured (human-friendly) names are organised in a **name space** that is represented as a **naming graph**: a directed graph with **leaf nodes** (entities) and **directory nodes** that hold a *directory table* of (edge label, node identifier) pairs. A **path name** such as `/home/steen/mbx` names the path from the **root** node. Names may be absolute or relative.

Name resolution and closure

Name resolution is the process of looking up a name in the naming graph: start at a node, read the directory table to follow the first label, and repeat for each remaining label until the target node is reached. Resolution needs a starting point — the **closure mechanism** (e.g., the file system's root is known via the superblock, the DNS resolver knows the root servers).

Aliases, linking and mounting

- **Hard link** — a second name (edge) pointing to the same node.
- **Symbolic link (soft link)** — a leaf node that stores the path name of another node; resolution continues with that path.
- **Mounting** — a directory node (mount point) in one name space refers to the root of another (possibly remote) name space. This is how NFS attaches remote file systems and how foreign name spaces are merged.

Implementation of a name space

In large systems the name space is divided into three **layers** according to how stable the names are and who manages them:

Layer	Scale	Change rate	Managed by	Example
Global	Worldwide	Very slow	Organisations	Root, TLDs (com, org, np)
Administrational	Organisation	Slow	Administrators	tu.edu.np
Managerial	Department / host	Frequent	Users / admins	Files, hosts inside a LAN

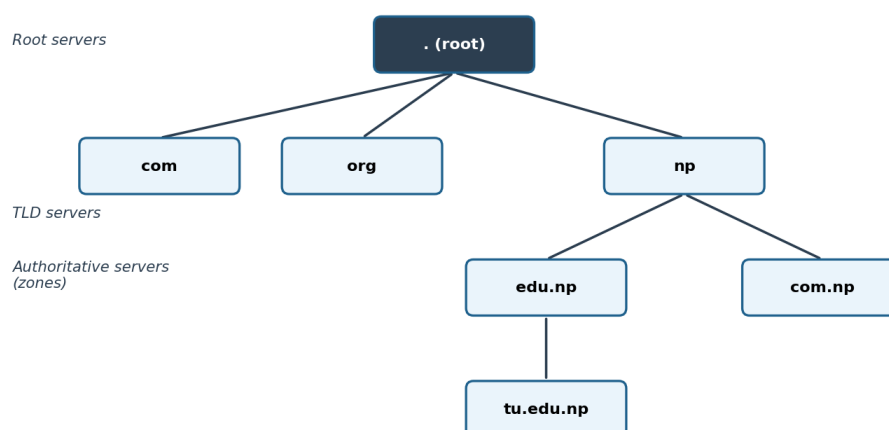


Figure 5.1 — DNS name space partitioned into zones.

Domain Name System (DNS)

DNS is the largest distributed naming system. Its name space is a labelled tree; each node's label is at most 63 characters; a full name is the labels joined with dots, read right to left (root at the right). The tree is divided into **zones**, each served by an **authoritative name server**; zones are replicated for reliability.

Record	Meaning
A / AAAA	Maps host name to IPv4 / IPv6 address
NS	Name server of a zone
CNAME	Canonical name (alias)
MX	Mail exchanger for the domain
PTR	Reverse mapping: address to name
SOA	Start of authority: zone parameters
TXT	Free text (SPF, verification)

Iterative and recursive name resolution

- **Iterative** — the resolver contacts each server in turn; each replies with the next server to ask (referral). Less load on servers; more work for the client.
- **Recursive** — the contacted server resolves the whole name on the client's behalf, calling the next server itself. Allows effective caching at servers but puts more load on them.

Example 5.2 — Resolving `www.tu.edu.np`

Describe iterative resolution by a local resolver whose cache is empty.

Solution

1. Resolver asks a **root server** for `www.tu.edu.np` → root replies with the NS of the **np** TLD.
 2. Asks the np server → it refers to the server for `edu.np`.
 3. Asks the `edu.np` server → it refers to the authoritative server for `tu.edu.np`.
 4. Asks the `tu.edu.np` authoritative server → gets the **A record** (IP address) of `www`.
- The resolver caches every answer for its **TTL**, so later queries skip steps 1–3.

5.3 Attribute-Based Naming

With structured names you must already know the name. In **attribute-based naming** (also *directory services*) an entity is described by a set of (attribute, value) pairs, and users search for entities by giving some attribute values (like a phone book or 'yellow pages'). Example query: *country = NP AND type = printer AND floor = 2*.

Directory service — LDAP

The Lightweight Directory Access Protocol (LDAP) is the practical descendant of the ISO X.500 directory service. A directory is a collection of **directory entries**, each with a set of attribute–value pairs. Each entry has a unique **Relative Distinguished Name (RDN)** within its parent; the concatenation gives the globally unique **Distinguished Name (DN)**, and the entries form a **Directory Information Tree (DIT)**.

```
DN: CN=Ram Sharma, OU=Computer Science, O=TU, C=NP
    C = country          O = organisation
    OU = organisational unit  CN = common name
Query: (&(objectClass=printer)(location=Floor2))
```

Implementation: the DIT is partitioned across **directory servers** (DSAs) and can be replicated. Because searches inspect attribute values, not just names, they are more costly than name look-ups, so many designs use hierarchical implementations plus indexes. **Decentralised implementations** map attributes to DHT keys or use gossip-based semantic overlays for large, dynamic collections.

Aspect	Structured naming	Attribute-based naming
Lookup key	Exact name / path	Any subset of attributes
Result	One entity	Zero or more entities
Example	DNS, file paths	LDAP, Active Directory, UDDI

5.4 Case Study: The Global Name Service (GNS)

The **Global Name Service** (Lampson, DEC Systems Research Center) is an early design for naming in a worldwide distributed system, intended to remain usable for a very long time while the system grows and is reorganised. Its main features are as follows.

- **Hierarchical name space** — the database is a tree of **directories**. Every directory has a unique **directory identifier (DI)** that never changes. A full name is essentially a DI (or a starting directory) plus a path within it.

- **Values** — leaf entries hold a tree of attribute values (e.g., the network address, authentication key or mailbox of a principal), so GNS acts as a general directory.
- **Replication for availability** — each directory is replicated on several servers; the update goes to one replica and is spread to the others lazily. Concurrent updates are resolved using **timestamps** (the most recent wins), so replicas converge eventually (weak consistency).
- **Caching** — clients cache results (hints) with time limits so that most look-ups are local.
- **Restructuring** — since directories are named by DIs, subtrees can be moved or merged (e.g., when two organisations merge) without renaming everything.
- **Security** — access control lists on directories decide who may read or update entries; entries can store authentication information.

Note: GNS is described differently in different textbooks (e.g., Coulouris et al. discuss it as a case study in the naming chapter). Use the exact points from your prescribed text for exam answers.

Key Takeaways

- ✓ Names refer to entities; addresses are names of access points; identifiers are unique, permanent and location-independent.
- ✓ Flat names are located through broadcasting, forwarding pointers, home-based approaches, DHTs (Chord, $O(\log N)$) or hierarchical location services.
- ✓ Structured names live in a naming graph; resolution needs a closure mechanism; links and mounting merge name spaces.
- ✓ DNS is a three-layer, zone-partitioned, cached, replicated system; resolution can be iterative or recursive.
- ✓ Attribute-based naming (LDAP) locates entities by their properties, not by name.
- ✓ GNS: hierarchical directories with permanent DIs, replication, timestamp-based reconciliation and caching.

— End of Unit 5 —