Distributed Systems

Naming in Distributed Systems

WEEK 9: Identifiers, Naming Schemes, Flat Naming, Structured Naming, Attribute-Based Naming.

Online Lecture Series - 9

Felix Edesa

Addis Ababa Science and Technology University

AASTU

Naming in Distributed Systems

Topics We Will Cover

- **Names, Identifiers, and Addresses** — Core concepts and distinctions
- **Flat Naming** — Simple, home-based, and DHT-based solutions
- **Structured Naming** — Hierarchical namespaces and resolution
- **Attribute-Based Naming** — Directory services and LDAP
- **Named Data Networking (NDN)** — Routing and security in data-centric models
- **Summary** — Design trade-offs and performance implications

AASTU

Naming

Essence of Naming

- Entities in a distributed system are identified by **names**.
- To access an entity, we use an **access point**.
- Each access point has an **address**.
- A **location-independent name** does not depend on addresses.

Concept Summary

- **Name** → Identifier of an entity.
- **Address** → Location for communication.
- **Goal** → Access entities without needing their physical location.

AASTU

Purpose of Naming

Why We Need Naming

- Naming provides a **uniform way to reference entities** across distributed systems.
- Simplifies locating and communicating with resources.
- Enables users and applications to interact without knowing the entity's location.
- Allows **abstraction** — separating identity from physical presence.

Example

- Accessing `printer@lab` without knowing its IP.
- Calling `auth-service` instead of a server address.
- Human-readable names simplify system configuration.

AASTU

Entities and Access Points

Key Concepts

- An **entity** represents any resource (e.g., file, user, process, or service).
- An **access point** is an interface through which an entity can be reached.
- Each access point has one or more **addresses**.
- Entities may offer multiple access points for fault tolerance.

Practical Example

- **Entity:** A file server process.
- **Access Points:**
  `TCP:10.0.0.1:8080`,
  `TCP:10.0.0.2:8080`.
- Clients use the name
  `fileserver@aastu.edu.et`.

AASTU

Location Independence in Naming

Core Idea

- A **location-independent name** remains valid even when the entity moves.
- Promotes flexibility and supports mobile systems.
- Enables transparent resource migration and replication.
- Requires dynamic binding between names and addresses.

Example — Real Systems

- **DNS:** Same domain name → new IP after migration.
- **Cloud microservices:** Same service name, changing container IPs.
- **Mobile clients:** Same username regardless of location.

AASTU

Challenges in Naming

Design Challenges

- Maintaining **uniqueness** across distributed systems.
- Handling **name collisions** and duplicates.
- Achieving **scalability** as entities grow.
- Managing **name-to-address mappings** dynamically.
- Ensuring **security and trust** in name resolution.

Case Example

- Two services register as "`printer@lab`" — conflict!
- Network renumbering changes IPs $\rightarrow$ outdated mappings.
- Attackers spoof DNS to redirect users (DNS poisoning).

AASTU

Identifiers

Core Concepts

- **Pure Name:** Random string with no inherent meaning.
- Used only for **comparison**.
- **Identifier:** Name with specific properties.
- Refers to **at most one entity**.
- Each entity has **at most one identifier**.
- Always refers to the **same entity** (never reused).

Practical Example

- **Student ID:** uniquely identifies a student.
- Two students may share the same name, but IDs differ.
- Ensures **consistent and reliable referencing**.

AASTU

Identifiers — Properties

Key Properties

- **Uniqueness:** No two entities share the same identifier.
- **Persistence:** Remains valid over the entity's lifetime.
- **Location-independence:** Identifier does not reveal physical location.
- **Non-reusability:** Once an entity is gone, its identifier is retired.

Practical Example

- **Bank Account Number:** Unique for each customer.
- Accounts can move between branches (location-independence).
- Closed account numbers are never reassigned.

AASTU

Identifiers — Types

Types of Identifiers

- **System-generated:** IDs assigned automatically (e.g., UUIDs).
- **Human-readable:** Names with semantic meaning (e.g., usernames).
- **Composite identifiers:** Combination of attributes (e.g., Country-Code + Serial Number).

Example

- **UUID:** Globally unique, system-generated.
- **Email:** Human-readable identifier for login.
- **License Plate:** Composite of state + number ensures uniqueness.

AASTU

Identifiers — Challenges

Challenges in Using Identifiers

- Avoiding **duplicate identifiers** in distributed systems.
- Handling **identifier collisions** across multiple domains.
- Ensuring **persistence and non-reusability**.
- Maintaining **privacy and security**.

Example Scenarios

- Two students accidentally assigned the same university ID.
- Networked devices generate overlapping UUIDs due to misconfiguration.
- Public exposure of identifiers may lead to privacy risks.

AASTU

Broadcasting

Core Concepts

- **Broadcasting:** Send a message to all entities in a network.
- Broadcast the **ID** requesting the entity's current address.
- **Limitation:** Cannot scale beyond local-area networks (LANs).
- **Requirement:** All processes must listen for incoming location requests.

Practical Example — ARP

- Address Resolution Protocol (ARP) queries: "Who has this IP address?"
- MAC address of the requested IP is returned to the sender.
- Operates efficiently in LAN, but **does not scale to WAN**.

AASTU

Broadcasting — Industrial Example

Example: Data Center Management

- Server management tools broadcast health check requests.
- Each server replies with its status (CPU, memory, disk usage).
- Quick detection of failures in LAN-based clusters.
- Limitation: Broadcast storms can occur if the cluster is large.

Notes

- Used in **high-availability clusters** for rapid state updates.
- Broadcast works best in **confined networks** (LANs or VLANs).
- In WAN or cloud-scale, broadcasting is replaced by **multicast or directory services**.

AASTU

Broadcasting — Limitations

Challenges in Broadcasting

- **Scalability:** Flooding large networks causes congestion.
- **Collision Risk:** Simultaneous replies lead to packet loss.
- **Security:** Susceptible to spoofing and denial-of-service attacks.
- **Inefficient for WANs:** Network-wide broadcasts generate heavy traffic.

Industrial Insight

- Large-scale systems replace broadcast with **multicast or DNS-like services**.
- Examples: ARP is effective only in local network segments.
- For cloud infrastructure, **service registries** maintain mapping instead of broadcasting.

AASTU

Broadcasting vs Unicast Multicast

Comparison of Routing Types

- **Broadcast:** Send to all nodes in network.
- **Unicast:** Send to a single node.
- **Multicast:** Send to a selected group of nodes.
- Broadcast is simple but **high traffic in large networks**.

Industrial Example

- Streaming updates in LAN: Broadcast sufficient for all nodes.
- Video conferencing: Multicast avoids sending multiple copies.
- DNS or ARP: Broadcast only within subnet; beyond that, hierarchy is used.

AASTU

Broadcasting — Practical Exercise / Case Study

Exercise: ARP Broadcast Simulation

- Scenario: 5-node LAN, each node has an IP and MAC address.
- Task: Node A wants to find MAC for IP 192.168.1.3.
- Step 1: Node A broadcasts “Who has 192.168.1.3?”.
- Step 2: All nodes listen and only 192.168.1.3 replies.
- Step 3: Node A caches the MAC for future communication.

Industrial Insight

- Broadcast works efficiently only in small LANs.
- Large networks use **ARP proxies** or **subnetting** to limit traffic.
- Exercise demonstrates trade-offs between **simplicity vs scalability**.

AASTU

Forwarding Pointers

Core Concepts

- When an entity moves, it leaves a **pointer** to its new location.
- Clients can **dereference pointers** transparently by following the chain.
- Update the client's reference once the current location is found.
- **Scalability challenge:** Long chains increase network latency.
- **Fault tolerance issue:** Longer chains are more vulnerable to failures.

Example Industrial Insight

- Mobile IP: Home agent forwards packets to current foreign agent using forwarding pointers.
- Location services in distributed file systems maintain chains.
- **Optimization:** Shorten chains periodically to reduce latency.

AASTU

Forwarding Pointers — Concept

Core Idea

- When an entity moves, it leaves a **pointer** to its next location.
- Clients can dereference pointers **transparently** by following the chain.
- Client references can be **updated dynamically** when the entity's current location is found.
- Helps in **decoupling entity identity from physical location**.
- Supports **mobility** in distributed systems (e.g., mobile devices, drones).

Practical Example

- Mobile app locates a moving delivery drone by following a pointer chain.
- Distributed game servers track players moving across regions.
- Cloud storage: objects migrated between nodes, client follows pointers.

AASTU

Forwarding Pointers — Challenges

Key Challenges

- **Long pointer chains** are not fault tolerant.
- Increased **network latency** during dereferencing.
- Maintaining **consistency** across distributed clients is difficult.
- **Geographical scalability** is limited without chain reduction.
- Overhead in updating pointers when entities move frequently.

Industrial Examples

- Tracking a user moving across multiple regional servers.
- Delivery services: repeated lookup slows down real-time tracking.
- Peer-to-peer systems: long chains increase message hops.

AASTU

Forwarding Pointers — Optimization

Chain Reduction Techniques

- Update client references periodically to **shorten chains**.
- Use **shortcut pointers** to recent.
- Collapse chains after entity stabilizes in a region.
- Implement **distributed caching** for frequently accessed entities.
- Combine with **location directories** for faster access.

Practical Examples

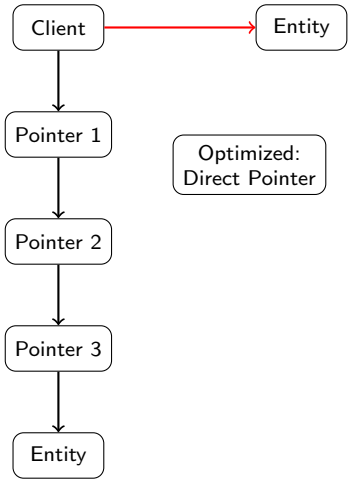
- IoT fleet management: optimized pointers reduce lookup time.
- Multiplayer gaming: reduces latency when players move between servers.

AASTU

Forwarding Pointers — Visualization (Vertical Chain)

Pointer Chains

- Original chain: client follows multiple pointers to reach entity.
- Optimized chain: client accesses entity directly via shortened pointer.
- Reduces network latency and improves fault tolerance.
- Common in industrial distributed directories to track moving entities.



Home-Based Approaches

Concept Overview

- **Single-tiered scheme:** A single home node is responsible for tracking the location of each entity.
- The **home** acts as a reference point that maintains up-to-date information about the entity's current location.
- The entity's **home address** is registered at a global **naming service**.
- Whenever the entity moves, the home updates and registers the entity's new **foreign address**.

Key Insights

- Reduces complexity with centralized tracking.
- May create a bottleneck at the home location.
- Effective for smaller, stable networks.

Detailed Operation of Home-Based Scheme

How It Works

1. **Entity Registration:** Each entity has a fixed home node that stores its permanent address.
2. **Location Update:** When an entity moves to a new network, it sends its new foreign address to its home.
3. **Client Request:** Clients query the home to resolve the current location of the entity.
4. **Redirection:** The home returns the foreign address
5. **Caching:** Clients may cache the foreign address for future direct contact.

Practical Example

- A **mobile agent** migrating between servers updates its home node each time it relocates.
- Clients contact the home once and then interact directly with the current host.

Advantages and Limitations of Home-Based Approaches

Overview

- Provides a **central point** for tracking entity locations.
- Works best in **stable or small-scale** environments.
- However, may lead to **bottlenecks** and higher latency if heavily loaded.

Practical Insight

- Used in **Mobile IP** for device location updates.
- Can be enhanced by **replicating home nodes**.
- Ideal demo: small network showing home redirecting client to foreign node.

AASTU

Applications of Home-Based Location Management

Use Cases

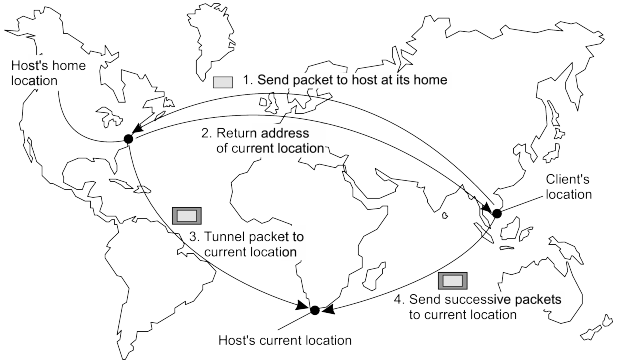
- **Mobile IP:** The home agent maintains a mapping between the mobile device's permanent IP and its current care-of address.
- **Cloud VM Migration:** Virtual machines update a central registry.
- **IoT Device Tracking:** Smart sensors register with a central home hub for location-aware operations.
- **Distributed Gaming Servers:** Player state is tracked by a home server.

Visualization Suggestion

- Display a short animation showing "Home $\rightarrow$ Foreign $\rightarrow$ Client" interaction.
- Demonstrate a client querying home and redirecting to new location.

AASTU

The principle of mobile IP



M. van Steen and A. S. Tanenbaum, *Distributed Systems*, 4th ed., 2024.

AASTU

Illustrative Example: Chord

Concept Overview

- **Chord** organizes many nodes into a **logical ring structure**.
- Each node is assigned a random **m-bit identifier**.
- Every entity (data item) has a unique **m-bit key**.
- The entity with key k is managed by the node with the smallest ID $\geq k$, known as its **successor** succ(k).

Practical Note

- Nodes conceptually form a **circular ring**.
- A simple (non-optimized) approach: each node tracks only its **immediate neighbor** and performs a **linear search**.
- We denote each node as p (node with identifier p).

AASTU

Chord Finger Tables

Principle and Structure

- Each node p maintains a **finger table** $\mathrm{FT}_p[]$ with at most m entries.
- Each entry i points to the **successor node** of $(p + 2^{i-1})$ on the ring:

$$\mathrm{FT}_p[i] = \mathrm{succ}(p + 2^{i-1})$$

- This enables **logarithmic lookup** instead of linear traversal.

Example and Interpretation

- Suppose $m = 4$ (IDs range from 0 to 15), and node $p = 3$.
- Finger table of node 3:

$$\begin{aligned} i = 1: &\quad \mathrm{succ}(3+1) = \mathrm{succ}(4) \\ i = 2: &\quad \mathrm{succ}(3+2) = \mathrm{succ}(5) \\ i = 3: &\quad \mathrm{succ}(3+4) = \mathrm{succ}(7) \\ i = 4: &\quad \mathrm{succ}(3+8) = \mathrm{succ}(11) \end{aligned}$$

AASTU

Chord Lookup Using Finger Tables

Lookup Example

- Find the node responsible for key $k = 11$.
- Assume $m = 4$ and start at node $p = 3$.
- Finger table of node 3:

$$[4, 5, 7, 11]$$

- Node 3 forwards to node 7, which finds successor node 11.
- Result: key 11 found in **2 hops**.

Interpretation and Notes

- Each hop reduces the remaining search space exponentially.
- Average lookup time: $O(\log N)$.
- Efficient even for large distributed systems.

AASTU

Chord Performance and Fault Tolerance

Performance Highlights

- Lookup grows logarithmically: for $N = 1024$, $\log_2 1024 = 10$ hops.
- For $N = 64$, average lookup $= 6$ hops.
- Replication on successors improves fault tolerance.

Interpretation

- Chord maintains efficiency even under frequent node churn.
- Recovery handled locally by neighboring nodes.
- Scalable and resilient design for peer-to-peer systems.

AASTU

Chord Finger Table Lookup Process

Principle

- Node p maintains a **finger table** $\mathrm{FT}_p[]$ of m entries.
- Each entry points to the first node at $p + 2^{i-1}$:

$$\mathrm{FT}_p[i] = \mathrm{succ}(p + 2^{i-1})$$

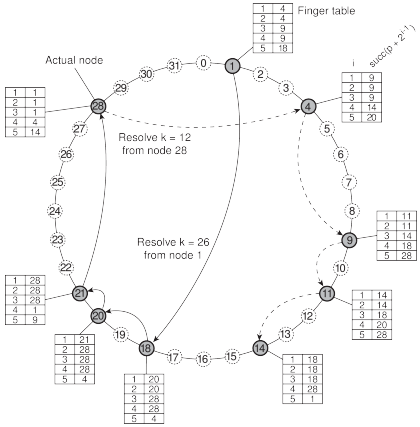
- Lookup of key k: forward to the largest $\mathrm{FT}_p[j] \leq k$.

Example

- Let $m = 4$, node $p = 3$, $\mathrm{FT}_3 = [4, 5, 7, 11]$.
- Lookup key $k = 9$:

$$7 \leq 9 < 11 \Rightarrow \text{forward to node } 7$$

Chord lookup example Resolving key 26 from node 1 and key 12 from node 28



M. van Steen and A. S. Tanenbaum, *Distributed Systems*, 4th ed., 2024.

Exploiting Network Proximity

Problem

- Overlay nodes may be far apart in the underlying network.
- Node p and succ$(p+1)$ could be geographically distant.
- Causes inefficient and erratic message transfers.

Solutions

- Topology-aware node assignment: assign IDs so nearby nodes in ID space are also close in network.
- Proximity routing: maintain multiple successors and forward to the closest.
- Proximity neighbor selection: choose the closest neighbor when there is a choice.

AASTU

Exploiting Network Proximity

Problem

- Overlay nodes may be far apart in the underlying network.
- Node p and succ$(p+1)$ could be geographically distant.
- Causes inefficient and erratic message transfers.

Solutions / Example

- **Topology-aware:** Nodes with nearby IDs placed close in the network.
- **Proximity routing:** Forward to the nearest successor (e.g., node 5 instead of 8).
- **Neighbor selection:** Choose the closest neighbor with lowest latency.

AASTU

Hierarchical Location Services (HLS)

Basic Idea

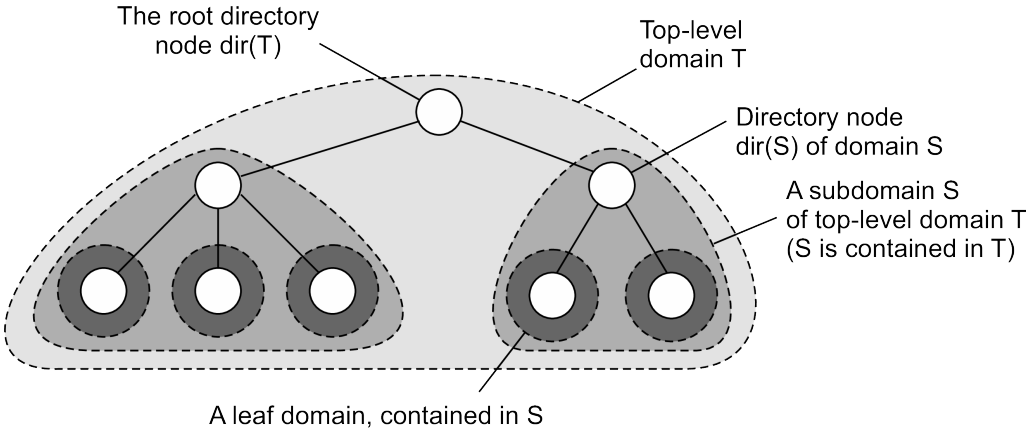
- Divide the network into hierarchical domains.
- Each domain has a dedicated directory node.
- Build a large-scale search tree for locating entities efficiently.

Example / Insight

- Lookup starts at the top-level directory and moves down the hierarchy.
- Reduces search scope compared to flat location services.
- Scales efficiently for very large networks.

AASTU

Principle



M. van Steen and A. S. Tanenbaum, *Distributed Systems*, 4th ed., 2024.

AASTU

HLS: Tree Organization

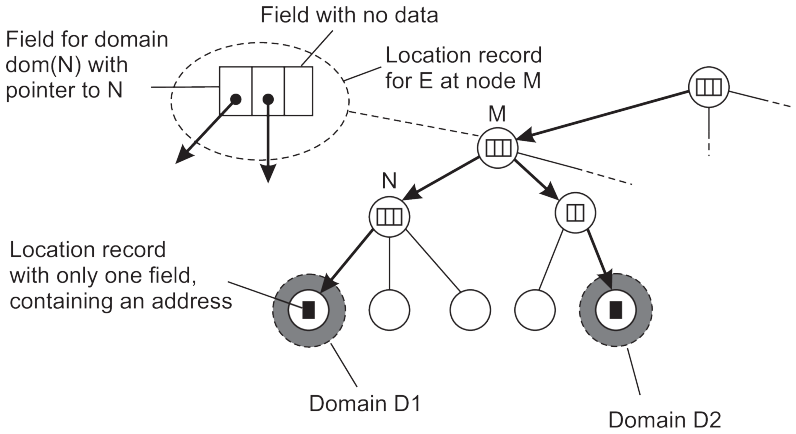
Invariants

- Entity E's address is stored in a leaf or intermediate node.
- Intermediate nodes point to a child only if the subtree contains E.
- The root maintains knowledge of all entities.
- Handles entities with multiple addresses across leaf domains.

Example / Insight

- Lookup follows pointers from root to relevant leaf nodes.
- Efficiently narrows search space for large networks.
- Supports entities with multiple addresses transparently.

Principle



M. van Steen and A. S. Tanenbaum, *Distributed Systems*, 4th ed., 2024.

HLS: Lookup Operation

Basic Principles

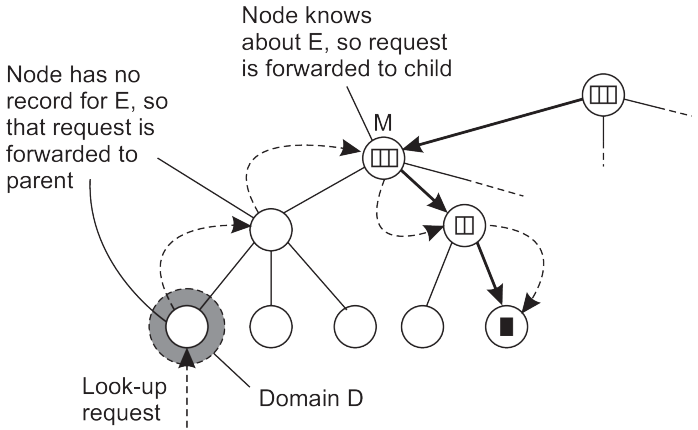
- Start lookup at the local leaf node.
- If the node knows about entity E, follow downward pointer.
- Otherwise, move upward in the hierarchy.
- Upward lookup always stops at the root.

Example / Insight

- Lookup begins at leaf and traverses pointers toward E.
- Root node acts as the ultimate reference for all entities.
- Minimizes unnecessary searches in unrelated subtrees.

AASTU

Principle



M. van Steen and A. S. Tanenbaum, *Distributed Systems*, 4th ed., 2024.

HLS: Insert Operation

Basic Steps

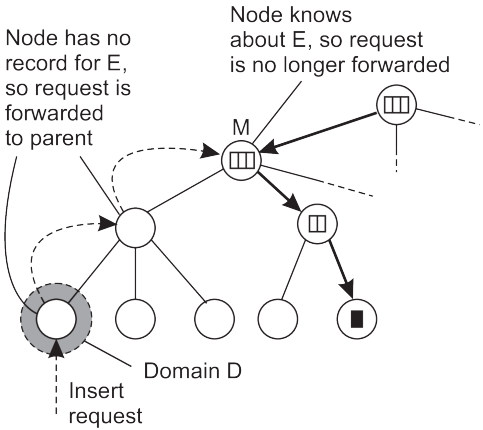
- Forward the insert request to the first node that knows about entity E.
- Establish a chain of forwarding pointers down to the leaf node.

Example / Insight

- Leaf node eventually stores E's address.
- Forwarding chain ensures efficient future lookups and updates.

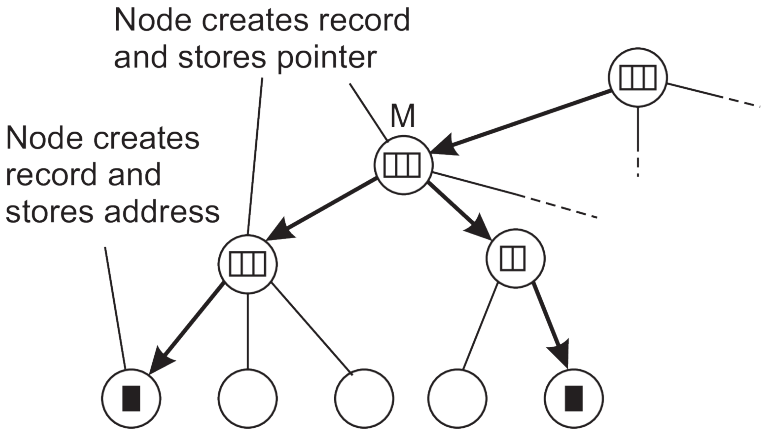
AASTU

HLS: Insert operation



M. van Steen and A. S. Tanenbaum, *Distributed Systems*, 4th ed., 2024.

HLS: Insert operation



M. van Steen and A. S. Tanenbaum, *Distributed Systems*, 4th ed., 2024.

Can HLS Scale?

Solution

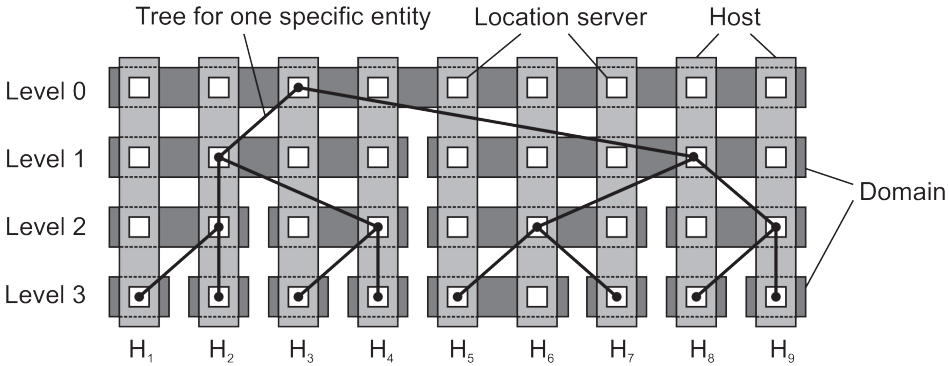
- Denote level k domains as $D_k = \{D_{k,1}, D_{k,2}, \ldots, D_{k,N_k}\}$.
- $N_0 = |D_0| = 1$, the root level.
- Each level k partitions hosts into N_k subsets.
- Each host runs a location server for exactly one domain $D_{k,i}$.

Insight

- Hierarchical partitioning allows HLS to scale to large networks.
- Each server handles only a subset of hosts at its level.

AASTU

HLS: Insert operation



M. van Steen and A. S. Tanenbaum, *Distributed Systems*, 4th ed., 2024.

Security in Flat Naming: Basics

Challenges

- Name-resolution process must be trusted to avoid incorrect associations.
- Two main approaches:
  - Secure the identifier-to-entity association to prevent impersonation.
  - Secure the name-resolution process to prevent tampering or spoofing.
- Without proper measures, flat names may point to wrong entities.

Insight

- Flat names alone cannot guarantee correctness or authenticity.
- Security must be enforced both at the identifier level and during resolution.
- These measures prevent attacks such as spoofing, man-in-the-middle, or unauthorized updates.

Self-Certifying Names and Name-Resolution

Self-Certifying Names

- Flat names include a value derived from the associated entity.
- Examples:
  - Read-only entities:
    $id(entity) = hash(data)$
  - Other entities:
    $id(entity) = publickey(entity)$
- Enables verification using digital signatures.
- Prevents impersonation and ensures authenticity of the entity.

Securing Name-Resolution

- More complex than securing identifiers alone.
- Requires ensuring the resolution path is trustworthy.
- Defers to secure DNS mechanisms.
- Prevents attacks such as spoofing.

Naming Graph

Basic Concept

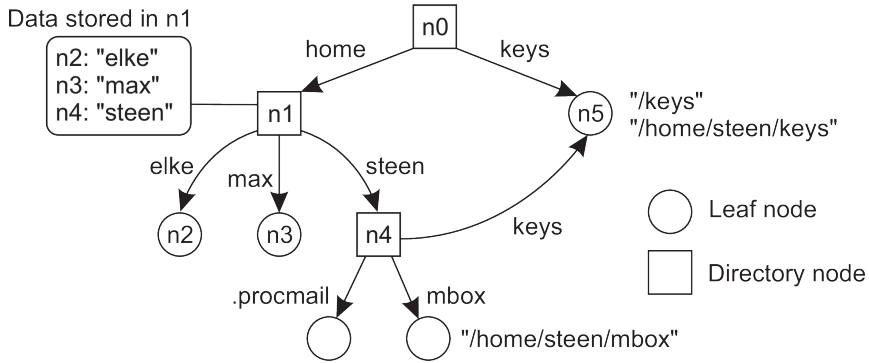
- A leaf node represents a named entity in the system.
- A directory node refers to other nodes, acting like an internal pointer.
- The graph is organized with a single root node at the top.
- Supports hierarchical and distributed naming efficiently.

Insight / Example

- Lookup starts at the root and traverses directory nodes to reach leaf nodes.
- Directory nodes may point to multiple leaves or other directories.
- Enables structured, scalable, and manageable naming systems.

AASTU

Naming Graph



M. van Steen and A. S. Tanenbaum, *Distributed Systems*, 4th ed., 2024.

Questions?

Thank you for your attention!

References

Suggested References

1. M. van Steen and A. S. Tanenbaum, *Distributed Systems*, 4th ed., Version 4.02, February 2024.
2. G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, *Distributed Systems: Concepts and Design*, 5th ed., Addison-Wesley, 2012.
3. N. Lynch, *Distributed Algorithms*, 2nd ed., Morgan Kaufmann, 1996.
4. R. E. Bryant and D. O'Hallaron, *Computer Systems: A Programmer's Perspective*, 4th ed., Pearson, 2024.

AASTU