Naming

Today

- What’s in a name
- Flat naming
- Structured naming
- Attribute-based naming

Names, identifiers and addresses

- Names are used to denote entities in a distributed system
  - Hosts, printers, files, processes, users ….
- To operate on an entity, e.g. print a file, we need to access it at an access point
  - An entity can offer more than one access points (think of telephone numbers)
- Access points are entities that are named by means of an address (telephone numbers)
- A location-independent name for an entity *E, is independent* from 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 the following properties
  – Each identifier refers to at most one entity
  – Each entity is referred to by at most one identifier
  – An identifier always refers to the same entity (no reusing)
- An identifier need not necessarily be a pure name, i.e., it may have content

Flat naming

- Given an essentially unstructured name (e.g., an identifier), how can we locate its associated access point?
  - Simple solutions (broadcasting)
  - Home-based approaches
  - Distributed Hash Tables (structured P2P)
  - Hierarchical location service

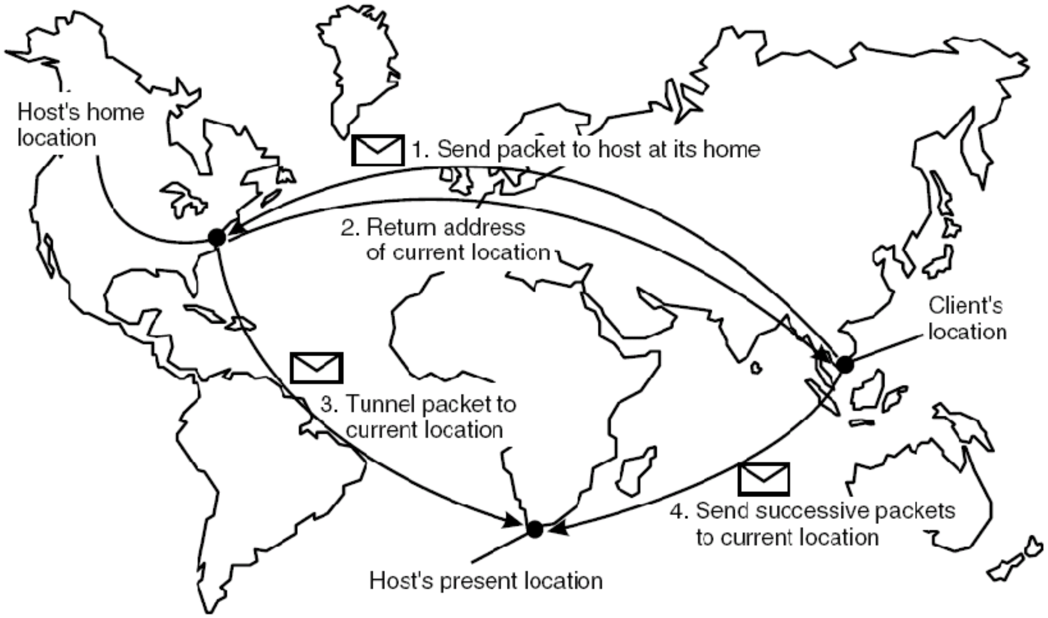
Simple solutions

- Broadcasting – simply 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
- Forwarding pointers – each time an entity moves, it leaves behind a pointer telling where it has gone to.
  - Dereferencing can be made entirely transparent to clients by simply following the chain of pointers
  - Update a client's reference as soon as present location has been found
  - Geographical scalability problems:
    - Long chains are not fault tolerant
    - Increased network latency at dereferencing

  Essential to have separate chain reduction mechanisms

Home-based approaches

- Single-tiered scheme – let a home keep track of where the entity is:
  - An entity's home address is registered at a naming service
  - The home registers the foreign address of the entity
  - Clients always contact the home first, and then continues with the foreign location



Home-based approaches

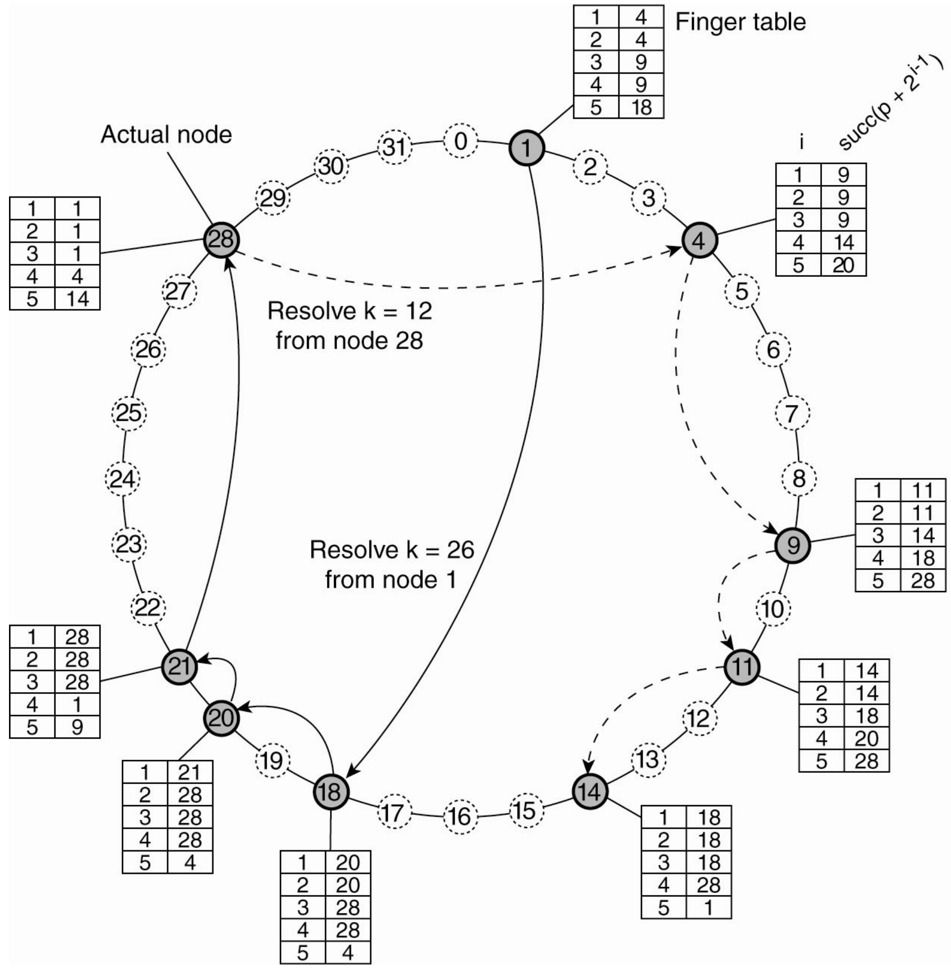
- Two-tiered scheme – keep track of visiting entities:
  - Check local visitor register first
  - Fall back to home location if local lookup fails
- Problems with home-based approaches:
  - The home address has to be supported as long as the entity lives.
  - The home address is fixed, which means an unnecessary burden when the entity permanently moves to another location
  - Poor geographical scalability (the entity may be next to the client)

Distributed Hash Tables (DHT)

- Consider the organization of nodes into a logical ring (Chord)
  - 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)*
- Non-solution: Let node id keep track of *succ(id)* (and pred) and do a linear search along the ring
- Finger tables – each node p maintains a finger table $FT_p[]$ with at most m entries: $FT_p[i] = succ(p + 2^{i-1})$
- This are basically shortcuts to nodes in the identifier space
- $FT_p[i]$ points to the first node succeeding p by at least 2^{i-1}.
  - To look up key k, 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 $FT_p[1]$

DHTs

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

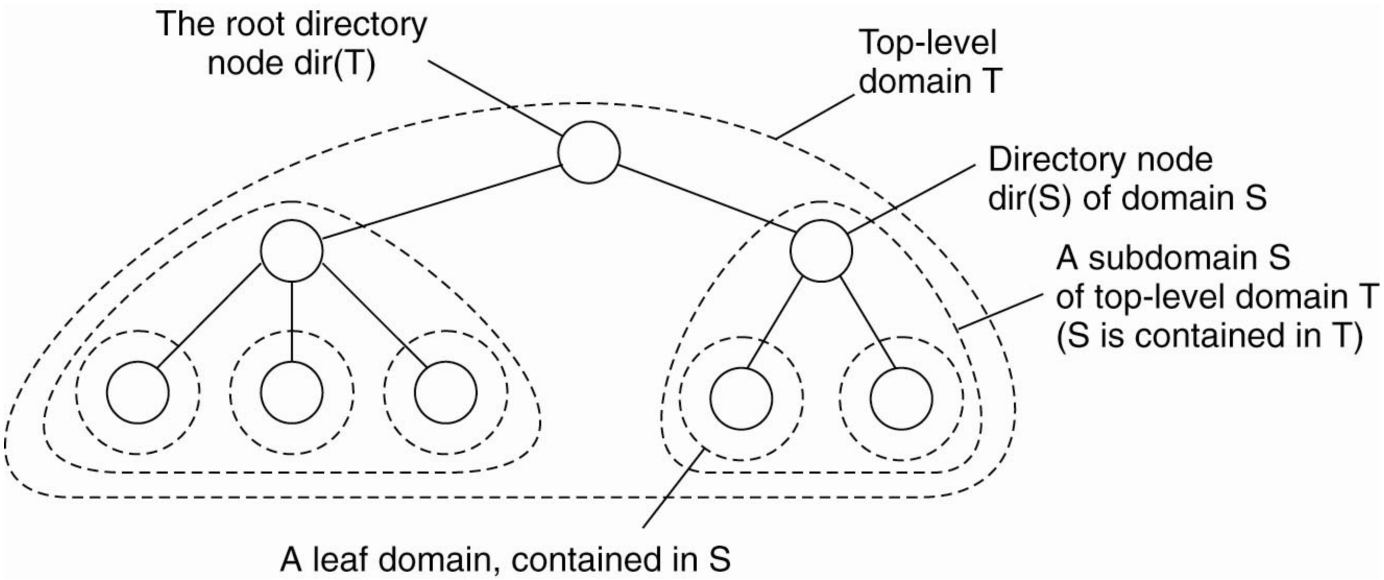


Exploiting network proximity

- The logical organization of nodes in the overlay may lead to erratic message transfers in the underlying
  - 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.

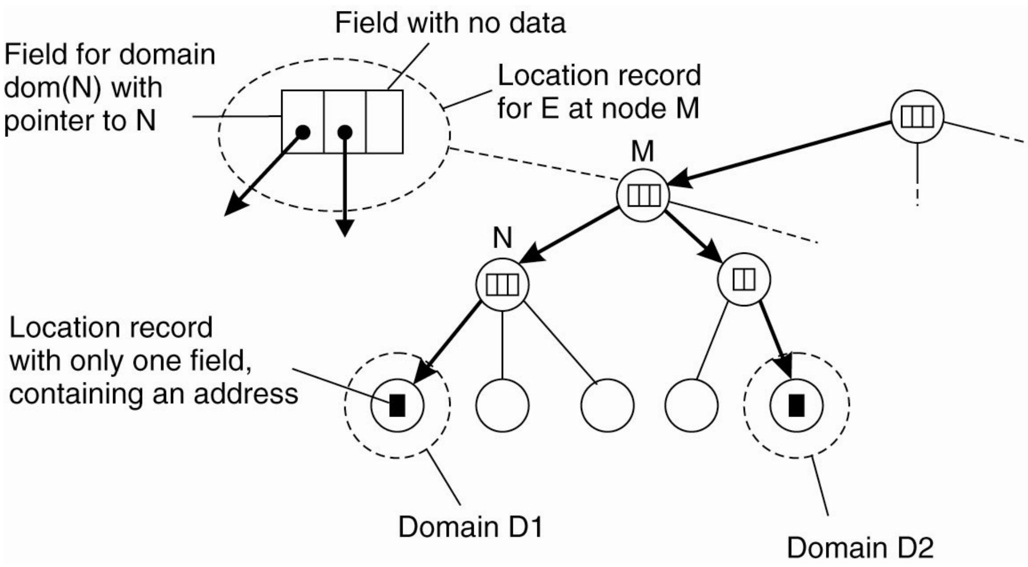
Hierarchical location system

- 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.



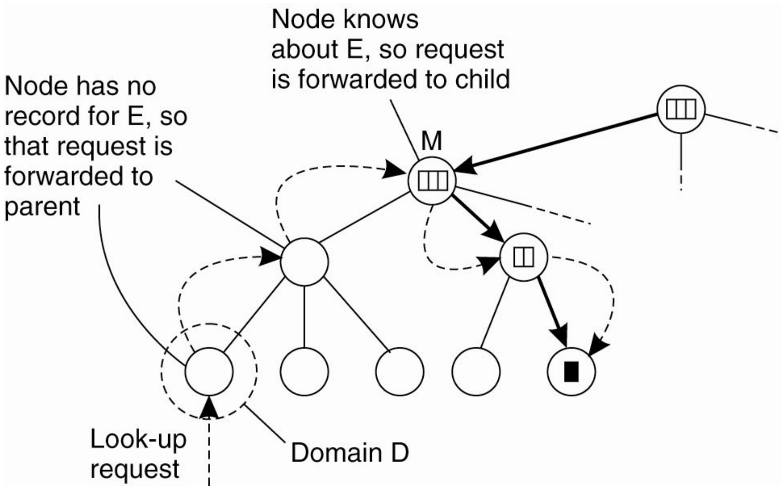
HLS – Tree organization

- The address of an entity is stored in a leaf node, or in an 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

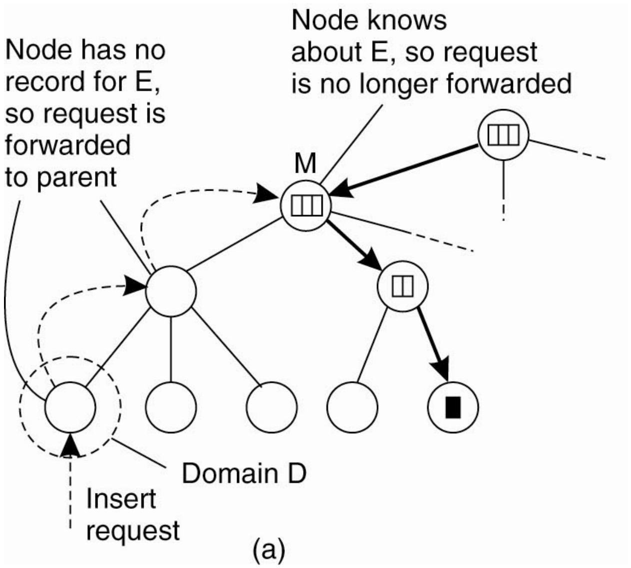


HLS lookups and inserts

•Start lookup at local leaf node
•If node knows it, follow downward
pointer, otherwise go one up
•Upward lookup always stops at root

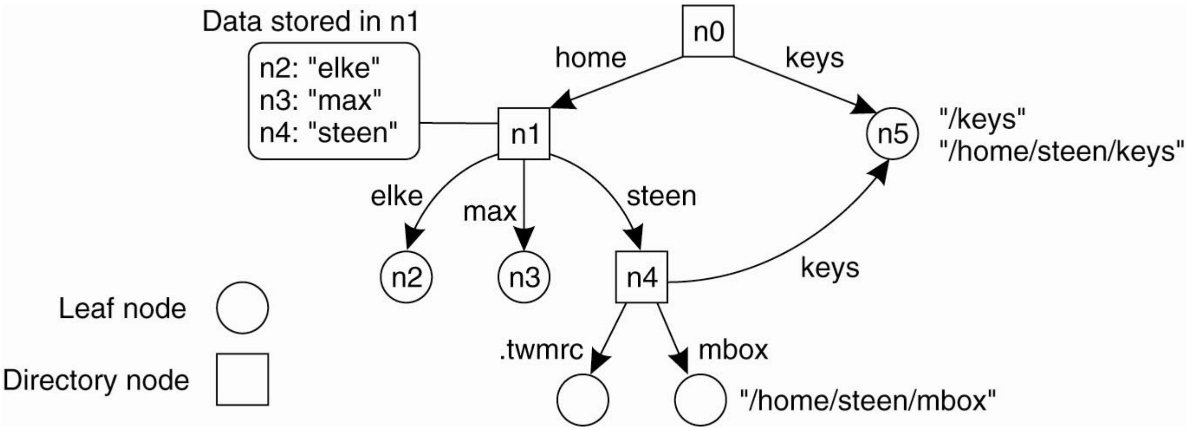


• Insertion initiated in leaf domain D – node dir(D)
•This forwards to parent, … until it reaches directory node M
•Request is push down with each node creating a location record



Name space

- A graph in which a leaf node represents a (named) entity. A directory node is an entity that refers to other nodes
- A directory node contains a (directory) table of *(edge label, node identifier) pairs.*
- We can easily store all kinds of attributes in a node, describing aspects of the entity the node represents:



Name resolution & linking

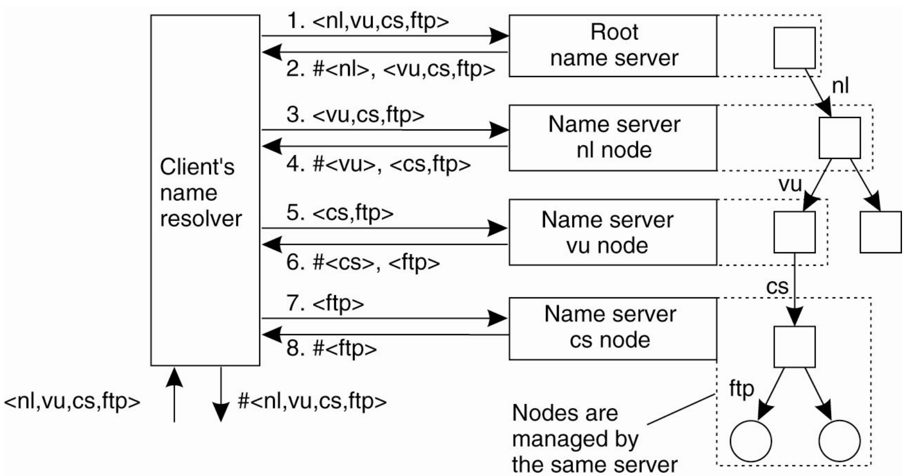
- Problem: To resolve a name we need a directory node. How do we actually find that (initial) node?
- Closure mechanism: The mechanism to select the implicit context from which to start name resolution:
  - ww.cs.vu.nl: start at a DNS name server
  - /home/steen/mbox: start at the local NFS file server
- 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 *O* to contain a name of another node:
  - First resolve *O's name (leading to O)*
  - Read the content of *O, yielding name*
  - Name resolution continues with name

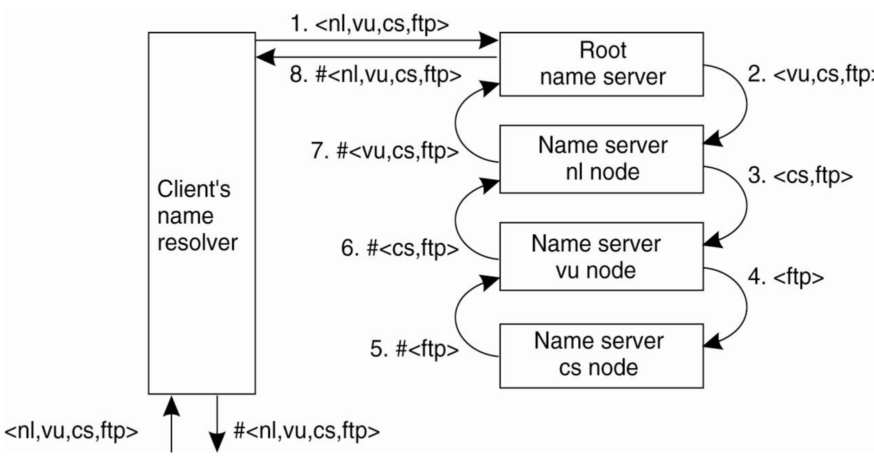
Name space implementation

- Basic issue: Distribute the name resolution process as well as name space management across multiple machines, by distributing nodes of the naming graph.
- Consider a hierarchical naming graph and distinguish three levels:
  - Global level: Consists of the high-level directory nodes. Main aspect is that these directory nodes have to be jointly managed by different administrations
  - Administrational level: Contains mid-level directory nodes that can be grouped in such a way that each group can be assigned to a separate administration.
  - Managerial level: Consists of low-level directory nodes within a single administration. Main issue is effectively mapping directory nodes to local name servers.

Interactive and recursive resolution

- Interactive – client drives the resolution
  - Caching by clients
  - Potentially costly communication
- Recursive – the server does
  - Higher performance demand on servers
  - More effective caching
  - Reduced communication costs





Scalability issues

- Size – ensure that servers can handle a large number of requests per time unit
- Solution: Assume, at least at high levels, that content of nodes hardly ever changes – leverage replication and start name resolution at the nearest server
- Observation: An important attribute of many nodes is the address where the represented entity can be contacted. Replicating nodes makes large-scale traditional name servers unsuitable for locating mobile entities
- Geographical – ensure that the name resolution process scales across large geographical distances

Attribute-based naming

- In many cases, it is much more convenient to name, and look up entities by means of their attributes – traditional directory services
- Lookup operations can be extremely expensive, as they require to match requested attribute values, against actual attribute values
- Solutions:
  - Implement basic directory service as database, and combine with traditional structured naming system – LDAP
  - Entities' descriptions are translated into attribute-value trees which are encoded into a set of unique hash ids for a DHT – INS/Twine, SWORD, Mercury

Summary

- Naming is central to computer systems in general and distributed systems in particular
- How do you name things?
- How do you find what you are looking for?
- What if that’s a moving target?
- How do you implement name/directory services in an scalable manner?