Architectures of Distributed Systems
2010/2011

Naming & references

Johan Lukkien
Agenda

• Service discovery
• On naming
• Distributed flat resolution
  – DHT (distributed hashtable) of Chord
• Distributed structured resolution
  – DNS
Service discovery model

• The basic problem: how do two parties, that don’t know each other, meet?

• Three roles:
  – publisher, seeker, mediator (broker)

• Tasks:
  – service/query advertisement, (propagation), matching, evaluation

Slides based on work by Melissa Tjong, SAN, TU/e
Service discovery process

Service query → Query propagation → Matching → Results collection → Service selection

publish

find

IF (no result) adjust query

adjust query

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Informatica, System Architecture and Networking
Discovery techniques

- Discover and contact, alternatives
  - Server registers at repository (a)
    - can be on different machine
  - client contacts repository for access point
    - e.g. DCE, DNS, SLP
  - client contacts known access point for server access
    - needs some accepted list of services
    - example: services coupled to fixed transport port
      - e.g., ftp: 21, telnet: 23 etc.
  - Server:
    - ‘Superserver’, listens to many access points (b)
      - Superserver instantiates particular server on request
      - example: inetd (internet daemon)
    - just start the server beforehand on the known access point

- Note:
  - the repository (a) or superserver (b) is again a server itself
Inetd (create server upon request)

![Diagram of Inetd](image)

Picture from 'The Apache Modeling Project',
Grone, Knopfel, Kugel, Schmidt, Potsdam University
Further alternatives: immediate

- Interested parties find each other through *multicasting (broadcasting)*
  - server or client, or both,
    - multicast presence periodically or
    - single multicast of each party upon joining a community (network, other group) or
    - multicast as query
  - examples: unmanaged SLP, DHCP, SSDP (UPnP), Apple Bonjour (rendez-vous)

- Important for *bootstrapping*

- Advantage:
  - fully distributed
  - no central state maintained
  - ‘zero configuration’

- Disadvantage
  - each contender implements the entire protocol, including storage
    - no shared services
  - scalability
    - performance [what are performance measures of interest here?]
  - limited to multicast (broadcast) scope
Immediate architecture

(a) search for service

<broadcast> query(type1, seeker access point)

<broadcast> query(type1, seeker access point)

service reply

matching

matching

(b) advertise service

<broadcast> advertisement(type 2, service access point)

registration

<broadcast> advertisement(type 1, service access point)

registration

search for service

matching
DHCP

- Immediate, bootstrapping protocol, provides
  - IP address, with a lease
  - addresses of caching DNS servers
- Client side broadcast on local net
- Client side selection of server (not shown in previous diagram)

from RPF2131: DHCP
After discovery: server cluster

- Goals
  - improve performance
    - scalability: increase number of requests served
    - decrease delay
  - improve reliability
  - improve availability

- Problems to be solved:
  - transparent (for client) distribution of request
    - particularly: dealing with the single access-point
  - security
  - management
    - see Planetlab case study in the book
Cluster access

- Handoff based on application request
  - supports dedicated servers
  - requires request interpretation
    • potential performance problems

- TCP handoff
  - incoming transport-layer request passed to the server
  - needs protocol adaptation
    • ‘spoofing’ of the destination ID
    • switch routes all request traffic
    • OS support (both sides)

- Multiple access points
  - DNS load balancing (e.g. Google, Wikipedia)
    • DNS resolution cycles through a sequence of addresses
    • DNS resolution uses geographic information of client (geoDNS)
Agenda

• Service discovery
• On naming
• Distributed flat resolution
  – DHT (distributed hashtable) of Chord
• Distributed structured resolution
  – DNS
Naming

• A name serves as reference to an entity
  – entity: i.e., anything that can be operated upon

• Motivation:
  – Delay binding
    • mobility transparency: name remains constant under mobility
    • replication transparency: map name to several different entities
    • location transparency
  – Lookup
    • name serves as a means to locate an entity
  – Human-friendliness
<table>
<thead>
<tr>
<th>Structured/flat</th>
<th>Name</th>
<th>Application</th>
<th>Middleware</th>
<th>Transport</th>
<th>Network</th>
<th>Data link</th>
<th>Physical</th>
<th>Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>structured, both</td>
<td>DNS, URL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(structured, flat)</td>
<td>(IP, port)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>structured</td>
<td>IP address</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>flat</td>
<td>MAC address</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Naming in the Internet stack**

5-Oct-10

Johan J. Lukkien, j.j.lukkien@tue.nl

TU/e Computer Science, System Architecture and Networking
Namespace

• Set (of names)
  – Specification/algorithm as how to generate names

• Methods
  – flat naming
    • simply an unstructured set of names
  – structured naming
    • hierarchical, mostly, e.g., /user/johanl/....
      – relative name: relative with respect to a given prefix
      – absolute name: complete
  – prefixing
    – embedding a set of names uniquely into a larger set
      » e.g. <gm:weerbericht xmlns:gm="http://GlobalMeteo.com">
Entities and identifiers

- **E**: Entities
  - the collection of ‘things’ that need to be referred
    - resources, devices, network ports, ...
- **I**: Identifiers
  - the namespace for entities

- An identifier for an entity is unique; identifiers are not re-used
  - e.g., MAC address as identifier for a network card

- \( Id: E \rightarrow I \)
  - \( Id \) is injective and static
Access points and addresses

• **X:** Access points
  – a access point represents a place where an entity can be accessed
  – an entity can have several access points
    • loosens the tight relation: entity-identifier
    • an entity is associated with a subset of $X$

• **A:** Addresses
  – a namespace for access points
  – an address of an access point is unique
    • Though not always! This leads to ambiguity in the resolution procedure.
    • Examples?
  – addresses of access points may change over time

• Example
  – A PC has several access points (network cards). Each network card receives dynamically an IP address. Resolution of the IP address yields the identifier of the network card.
  – The access point of the network card is found directly through this identifier.
Binding, Resolution

• **Binding**: (the establishment of) the relationship between a reference (a name) and a referred object (another name or an access point)
  – *symbolic*: referred object is again a similar reference
  – *aliasing*: several names for the same object

• **Resolution**: given a name, determine referred object
  – ‘structured’ resolution: follow structure and levels in naming hierarchy, e.g.
    • given a DNS name, find Transport address, then find MAC address
  – ‘flat’ resolution: flat naming, no hierarchy support, e.g.,
    • reverse DNS: what is the DNS name of this IP address?
      – not entirely flat though as still information is remaining in the IP address
    • reverse ARP: what is the IP address of this MAC address?
    • given identifier, determine access point

• **Closure**: starting point in resolution
Agenda

• Service discovery
• On naming
• Distributed flat resolution
  – DHT (distributed hashtable) of Chord
• Distributed structured resolution
  – DNS
Flat naming and resolution

- Problem: given identifier, find access point
  - possibly, via an intermediate structured address

- Search (see: service discovery)
  - broadcast: ‘ask everyone’
    - limited physical scope
    - example: ARP
  - multicast
    - e.g. overlay networks
    - limited logical scope
    - example: searching in P2P overlay
    - may combine with repositories storing associations (identifier, address)
  - table lookup – hashing, tree-search
    - hashing inside repositories
    - distributed hash tables
    - distributed, tree-like repositories
Example: distributed storage and lookup

- Flat name space of $m$-bit identifiers
- Set of nodes, $N$, set of other entities (files, objects, processes, ...)

- Each node or entity $e$ has an identifier $id(e)$
  - typically a random assignment
  - we refer to nodes by their identifier; hence, ‘node $p$’ means ‘the node with identifier $p$’
  - the set from which we take nodes is therefore $id(N)$

- Identifier $k$ has an associated node, the one with the next largest or equal $id$, modulo $2^m$
  - $succ(k) = (\min n \in id(N), n \geq k: n)$ if $k \leq \max(id(N))$
  - $\min n \in id(N)$ otherwise

- Using this function $succ$ we define successor and predecessor of a node $p$
  - successor of $p$ is $succ(p+1)$
  - predecessor of $p$, $pred(p)$ is defined as follows: $pred(p) = y \equiv succ(y+1) = p$

- An entity with identifier $k$ is managed by (stored by) node $succ(k)$

- Hence, the problem is: given $k$, find (the address of) $succ(k)$
Example

• Assume $m = 4$, $N = \{2, 5, 8, 12\}$
• $\text{succ}(4)$, $\text{succ}(5)$, $\text{succ}(6)$, $\text{succ}(13)$?
  – $\text{succ}(4) = 5$, $\text{succ}(5) = 5$, $\text{succ}(6) = 8$, $\text{succ}(13) = 2$
• Successor of 2, 5, 8, 12?
  – 5, 8, 12, 2
• Predecessors of 2, 5, 8, 12?
  – 12, 2, 5, 8
• 2 stores
  – 0, 1, 2, 13, 14, 15
• 5 stores
  – 3, 4, 5
• 8 stores
  – 6, 7, 8
• 12 stores
  – 9, 10, 11, 12
Distributed location lookup

- Logically the nodes are organized in a pipelined or cyclic fashion

- Linear search
  - node $p$ stores addresses of $\text{succ}(p+1)$ and $\text{pred}(p)$

- ‘Binary’ search (distributed hash table, DHT)
  - node $p$ stores addresses of nodes that take large steps (chords) along the circle
  - routing table with entries for $\text{succ}(p+2^{-i})$, $1 < i \leq m$
    - Chord system: ‘finger tables’

- Issues in:
  - entity insert: simple
  - node insert:
    - table updates in DHT
    - relocate entities
  - routing efficiency - use the underlying network efficiently
Flat naming and mobility

• Leave ‘follow me’ to new location
  – typically as part of middleware to establish location transparency as well as mobility transparency
  – sensitive to
    • broken links (dependability)
      – all intermediates must be online
    • long chains (efficiency, scalability)
      – all intermediates must maintain state

  – solutions:
    • pointer fusion
      – distributed object based systems
    • keep first indirection up-to-date
      – mobile IP: maintain a reference to current location at home location
      – (Question: how transparent can this be?)
- Mobile object replaced by stub that forwards invocations

- Pointer fusion: final stub responds with new reference (below)
  - not transparent
Mobile IP

- **Home agent** (HA) maintains binding between a fixed *home address* and the current *care-off* address of a mobile node.

- HA receives packets from a client (‘correspondent host’) on the home address and passes them on to current care-off address, via an IP tunnel.

- The *foreign agent* maintains the binding (mobile node, care-off address) at the current location. Typically, it is addressed via the care-off address and forwards the traffic via the local link.

- Mobile node uses home address as source in its datagrams.

- This establishes *triangular routing*.

- **Agent discovery**: immediate, advertisement of both query and agent

- **Agent handoff**: upon registering with a new foreign agent (2nd figure)
Agenda

- Service discovery
- On naming
- Distributed flat resolution
  - DHT (distributed hashtable) of Chord
- Distributed structured resolution
  - DNS
Hierarchical naming systems

- Graph: labeled, directed, acyclic
  - edge labels: partial names
  - paths: names

- Leaf nodes
  - access point
  - name in new naming system
  - name in same naming system ('symbolic')

- Directory nodes
  - access point to directory structure
    - references to child nodes
  - often represented by a special symbol
    - "/", "."
Name space examples

• File systems
  – usually, hierarchy
  – resolution: through file-tables
  – closure: e.g. superblock

• Internet addresses
  – basic hierarchy
  – binding: e.g. DHCP
  – resolution & closure: ARP

• Internet names
  – hierarchy
  – binding and resolution: DNS (yields Internet-address)
  – closure: local DNS server

• URL
  – combines several naming systems
  – Question: what is the resolution procedure here? and the closure?
Filesysteem

Data stored in n1
n2: "elke"
n3: "max"
n4: "steen"

n1

home

n0

keys

n5

"/keys"

elke

max

steen

(n2)

(n3)

(n4)

Data stored in n6

"/home/steen/keys"

Leaf node

Directory node

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking
Joining namespaces

• Direct catenation, as in URLs

• Use name of system as a prefix
  – systemA:/home/..., systemB:/home/....
  – resolve in new root

• Symbolic linking
  – store reference in second name system as referred object
    • e.g. contents of /home/johanl/aap is oracle1://johanl
    • similar in effect to mounting, but less dynamic: stored within the file itself

• Mounting
  – (tell the resolution procedure that) from a certain prefix $p$ on, a new resolution is used
    • mainly modification of resolution procedure
    • closure from $p$ onward stored with $p$ in a table of the resolver
    • transparent resolution
A distributed naming service

- A naming services provides
  - operation for adding, removing, and lookup of names
  - a closure mechanism: where does the search start?
    - Example closures: local, known fileserver, local, known DNS server, known gateway etc., just a context, ....
    - notice: at the place where two namespaces join, a closure mechanism must be included

- For performance, scalability and dependability: distribute...
  - ...the data (i.e., directory nodes and subgraphs)
  - ...the process of name resolution

- Levels in the graph
  - Global
    - jointly managed by several administrations
    - changes are rare
  - Administrative
    - nodes managed by a single administration
  - Managerial
    - nodes within single administration mapped onto local nameservers
    - frequent changes
## Comparison of layers

<table>
<thead>
<tr>
<th>Item</th>
<th>Global</th>
<th>Administrational</th>
<th>Managerial</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geographical scale of network</td>
<td>Worldwide</td>
<td>Organization</td>
<td>Department</td>
</tr>
<tr>
<td>Total number of nodes</td>
<td>Few</td>
<td>Many</td>
<td>Vast numbers</td>
</tr>
<tr>
<td>Responsiveness to lookups</td>
<td>Seconds</td>
<td>Milliseconds</td>
<td>Immediate</td>
</tr>
<tr>
<td>Update propagation</td>
<td>Lazy</td>
<td>Immediate</td>
<td>Immediate</td>
</tr>
<tr>
<td>Number of replicas</td>
<td>Many</td>
<td>None or few</td>
<td>None</td>
</tr>
<tr>
<td>Is client-side caching applied?</td>
<td>Yes</td>
<td>Yes</td>
<td>Sometimes</td>
</tr>
</tbody>
</table>
Example: DNS namespace

root zone:
20 generic top-level domains
248 country-code TLDs

Domain: subtree
Zone: delegated portion of a domain
(though terms are often used interchangeably)
DNS

- It is organized as a worldwide collection of name servers, collectively responsible for
  - maintaining the tree-like database of DNS
  - responding to queries aimed at finding a host IP address, as well as several other services (e.g. mail server)
- Levels in the graph
  - Global
    - the ‘roots’; jointly managed by several administrations
      - 13 replicated servers, named A-M
      - using anycast communication with clients
    - ~270+ top level domains, managed by
  - Administrative
    - nodes managed by a single administration (e.g. TU/e)
  - Managerial – book: not part of DNS
    - frequently changing local resources: part in URL after the DNS name
    - file system: e.g. your local website
DNS, extra-functional properties

• modification by different administrations
• extreme robustness
  – even under breaking apart of the distributed system
• scalability
  – usage parameters:
    • geographical spread
    • # machines
    • # queries
  – metrics:
    • query response time
  – scalability criterion
    • constant on average, at worst an occasional spike
Name servers

- Name server serves a certain zone
  - authoritative or cached answer
  - authoritative server for a host: always maintains a record of that host
- Root servers: are authoritative for root and top-level domains
- Name servers are configured with addresses of root servers
- Name servers for a domain
  - know their child domains
  - are typically replicated: primary (initialized from file) and secondary server (synchronizes with primary)
- DNS is made available to applications as a middleware service
  - typically a library (resolver) that calls upon a configured name server
  - that server often uses BIND (Berkeley Internet Name Daemon), named
Distributed resolution

- Query: `resolve(dir, name1, name2, ...)`

- Server responsible for directory node `dir` performs one step and finds `newServer`

  - **Iterative**: return `newServer` to client
    - caching: information becomes distributed
    - communication cost: from client to all named servers

  - **Recursive**: pass remaining request on to `newServer`; pass result on to client
    - caching: follows name structure
    - communication cost:
      - exploits locality in name
      - but repeated queries do not enjoy optimization
Iterative

Client's name resolver

1. <nl,vu,cs,ftp>
2. #<nl>, <vu,cs,ftp>
3. <vu,cs,ftp>
4. #<vu>, <cs,ftp>
5. <cs,ftp>
6. #<cs>, <ftp>
7. <ftp>
8. #<ftp>

Root name server

Name server nl node

Name server vu node

Name server cs node

Nodes are managed by the same server

5-Oct-10

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking
Recursive
### Caching

<table>
<thead>
<tr>
<th>Server for node</th>
<th>Should resolve</th>
<th>Looks up</th>
<th>Passes to child</th>
<th>Receives and caches</th>
<th>Returns to requester</th>
</tr>
</thead>
<tbody>
<tr>
<td>cs</td>
<td>&lt;ftp&gt;</td>
<td>#&lt;ftp&gt;</td>
<td>--</td>
<td>--</td>
<td>#&lt;ftp&gt;</td>
</tr>
<tr>
<td>vu</td>
<td>&lt;cs,ftp&gt;</td>
<td>#&lt;cs&gt;</td>
<td>&lt;ftp&gt;</td>
<td>#&lt;ftp&gt;</td>
<td>#&lt;cs&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>#&lt;cs, ftp&gt;</td>
</tr>
<tr>
<td>ni</td>
<td>&lt;vu,cs,ftp&gt;</td>
<td>#&lt;vu&gt;</td>
<td>&lt;cs,ftp&gt;</td>
<td>#&lt;cs&gt;</td>
<td>#&lt;vu&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>#&lt;cs,ftp&gt;</td>
<td>#&lt;vu,cs, ftp&gt;</td>
</tr>
<tr>
<td>root</td>
<td>&lt;ni,vu,cs,ftp&gt;</td>
<td>#&lt;nl&gt;</td>
<td>&lt;vu,cs,ftp&gt;</td>
<td>#&lt;vu&gt;</td>
<td>#&lt;nl&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>#&lt;vu,cs&gt;</td>
<td>#&lt;nl,vu&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>#&lt;vu,cs,ftp&gt;</td>
<td>#&lt;nl,vu,cs&gt;</td>
</tr>
</tbody>
</table>

- For the recursive case: see table
- For the iterative case: the iterative server caches all information
  - hence: client asks *local name server* (and not the root server) in a recursive way
  - name server operates iteratively to improve caching
Communication cost

Recursive name resolution

Client

Iterative name resolution

Name server nl node

Name server vu node

Name server cs node

Long-distance communication
Root servers (25-09-09) and replications

5-Oct-10

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking
k-servers traffic

Queries by Node
From Oct 02, 2010, 15:15:02 To Oct 03, 2010, 15:15:02 UTC

Query Rate (q/s)

Time, UTC

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking
Some information about root servers

- Investigation in 2002 on f-servers: 19% of the calling hosts and 98% of all queries is not legitimate, e.g.,
  - query a local address
  - query an IP address rather than a DNS name
  - query a non-existent TLD
  - lousy installations of DNS clients or erroneous implementations
    - e.g. filtering inbound DNS but not outbound traffic (blocking responses)
    - ‘stub’ servers call a root server directly
- Servers are fast (see figure)
- Many servers are replicated
  - e.g. F has 46 sites (in 2009)
Scaling

• Response time depends on depth of search

• Global servers get lots of requests
  – need to limit traffic / source

• Strategies
  – replicate
    • multiple servers at global and administrational level
    • using anycast and geoDNS
      – geoDNS: respond with server IP closest to query source

  – closure mechanism: start at local server that uses caching
  – addresses are assumed not to change very frequent
    • e.g. 24h-48h for TLD
    • does not work with highly mobile devices
DNS: database

• Each domain has a set of associated resource records
  – for a single host: just its IP address

• A name server can give an authoritative answer about the resource records it manages
  – for which is has a Start of Authority record

• Record contains
  – domain name
  – time to live
  – class – mostly IN(ternet)
  – type
  – value
# Resource record in DNS

<table>
<thead>
<tr>
<th>Type of record</th>
<th>Associated entity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOA</td>
<td>Zone</td>
<td>Holds information on the represented zone</td>
</tr>
<tr>
<td>A</td>
<td>Host</td>
<td>Contains an IP address of the host this node represents</td>
</tr>
<tr>
<td>MX</td>
<td>Domain</td>
<td>Refers to a mail server to handle mail addressed to this domain</td>
</tr>
<tr>
<td>SRV</td>
<td>Domain</td>
<td>Refers to a server handling a specific service</td>
</tr>
<tr>
<td>NS</td>
<td>Zone</td>
<td>Refers to a name server that implements the represented zone</td>
</tr>
<tr>
<td>CNAME</td>
<td>Node</td>
<td>Symbolic link with the primary name of the represented node</td>
</tr>
<tr>
<td>PTR</td>
<td>Host</td>
<td>Contains the canonical name of a host</td>
</tr>
<tr>
<td>HINFO</td>
<td>Host</td>
<td>Holds information on the host this node represents</td>
</tr>
<tr>
<td>TXT</td>
<td>Any kind</td>
<td>Contains any entity-specific information considered useful</td>
</tr>
</tbody>
</table>
Example database for domain cs.vu.nl

<table>
<thead>
<tr>
<th>Name</th>
<th>Record type</th>
<th>Record value</th>
</tr>
</thead>
<tbody>
<tr>
<td>cs.vu.nl.</td>
<td>SOA</td>
<td>star.cs.vu.nl. hostmaster.cs.vu.nl.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2005092900 7200 3600 2419200 3600</td>
</tr>
<tr>
<td>cs.vu.nl.</td>
<td>TXT</td>
<td>&quot;Vrije Universiteit - Math. &amp; Comp. Sc.&quot;</td>
</tr>
<tr>
<td>cs.vu.nl.</td>
<td>MX</td>
<td>1 mail.few.vu.nl.</td>
</tr>
<tr>
<td>cs.vu.nl.</td>
<td>NS</td>
<td>ns.vu.nl.</td>
</tr>
<tr>
<td>cs.vu.nl.</td>
<td>NS</td>
<td>top.cs.vu.nl.</td>
</tr>
<tr>
<td>cs.vu.nl.</td>
<td>NS</td>
<td>solo.cs.vu.nl.</td>
</tr>
<tr>
<td>cs.vu.nl.</td>
<td>NS</td>
<td>star.cs.vu.nl.</td>
</tr>
<tr>
<td>star.cs.vu.nl.</td>
<td>A</td>
<td>130.37.24.6</td>
</tr>
<tr>
<td>star.cs.vu.nl.</td>
<td>A</td>
<td>192.31.231.42</td>
</tr>
<tr>
<td>star.cs.vu.nl.</td>
<td>MX</td>
<td>1 star.cs.vu.nl.</td>
</tr>
<tr>
<td>star.cs.vu.nl.</td>
<td>MX</td>
<td>666 zephyr.cs.vu.nl.</td>
</tr>
<tr>
<td>star.cs.vu.nl.</td>
<td>HINFO</td>
<td>&quot;Sun&quot; &quot;Unix&quot;</td>
</tr>
<tr>
<td>zephyr.cs.vu.nl.</td>
<td>A</td>
<td>130.37.20.10</td>
</tr>
<tr>
<td>zephyr.cs.vu.nl.</td>
<td>MX</td>
<td>1 zephyr.cs.vu.nl.</td>
</tr>
<tr>
<td>zephyr.cs.vu.nl.</td>
<td>MX</td>
<td>2 tornado.cs.vu.nl.</td>
</tr>
<tr>
<td>zephyr.cs.vu.nl.</td>
<td>HINFO</td>
<td>&quot;Sun&quot; &quot;Unix&quot;</td>
</tr>
</tbody>
</table>
## Example (cnt’d)

<table>
<thead>
<tr>
<th>Host Name</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ftp.cs.vu.nl</td>
<td>CNAME</td>
<td>soling.cs.vu.nl</td>
</tr>
<tr>
<td><a href="http://www.cs.vu.nl">www.cs.vu.nl</a></td>
<td>CNAME</td>
<td>soling.cs.vu.nl</td>
</tr>
<tr>
<td>soling.cs.vu.nl</td>
<td>A</td>
<td>130.37.20.20</td>
</tr>
<tr>
<td>soling.cs.vu.nl</td>
<td>MX</td>
<td>1 soling.cs.vu.nl</td>
</tr>
<tr>
<td>soling.cs.vu.nl</td>
<td>MX</td>
<td>666 zephyr.cs.vu.nl</td>
</tr>
<tr>
<td>soling.cs.vu.nl</td>
<td>HINFO</td>
<td>&quot;Sun&quot; &quot;Unix&quot;</td>
</tr>
<tr>
<td>vucs-das1.cs.vu.nl</td>
<td>PTR</td>
<td>0.198.37.130.in-addr.arpa.</td>
</tr>
<tr>
<td>vucs-das1.cs.vu.nl</td>
<td>A</td>
<td>130.37.198.0</td>
</tr>
<tr>
<td>inkt.cs.vu.nl</td>
<td>HINFO</td>
<td>&quot;OCE&quot; &quot;Proprietary&quot;</td>
</tr>
<tr>
<td>inkt.cs.vu.nl</td>
<td>A</td>
<td>192.168.4.3</td>
</tr>
<tr>
<td>pen.cs.vu.nl</td>
<td>HINFO</td>
<td>&quot;OCE&quot; &quot;Proprietary&quot;</td>
</tr>
<tr>
<td>pen.cs.vu.nl</td>
<td>A</td>
<td>192.168.4.2</td>
</tr>
<tr>
<td>localhost.cs.vu.nl</td>
<td>A</td>
<td>127.0.0.1</td>
</tr>
</tbody>
</table>
Packet format

• UDP carrier
• Header:
  – id(16),
  – query?(1),
  – opcode (standard, inverse, server status)(4)
  – authoritative?(1)
  – truncated?(1) [lost bytes beyond the 512]
  – recursion desired?(1)
  – recursion available(1)
    • server response
• Later, extended with security measures to avoid tampering
  – secure carrier
  – signing

The limit of 512 bytes has consequences
The query: 'who serves the root' ('.', or the empty domain) has as answer: the list of root servers
This list consists of
  NS records: name of name server
  A records: address of name server
Of these, only 13 fit in a single packet
Hence, 13 root servers
Concluding remarks

• A DHT-based implementation of DNS where names are hashed is possible as well
  – this can be done for any naming system in which we can derive some serial representation systematically
  – although the structure then disappears

• The DNS lookup (or structured lookup) is, in fact, based on a key

• Instead, a lookup based on attribute values – including searching and wildcards – can be needed
  – a list of (attribute, value) pairs is called a directory
  – service is termed:, directory services, in contrast to a naming service
    • OSI X.500 directory service
    • LDAP, Lightweight Directory Access Protocol
      – MS Active Directory