Architecture of Distributed Systems 2016-2017

Naming

Original : J.J Lukkien

Revision: R.H. Mak

Agenda

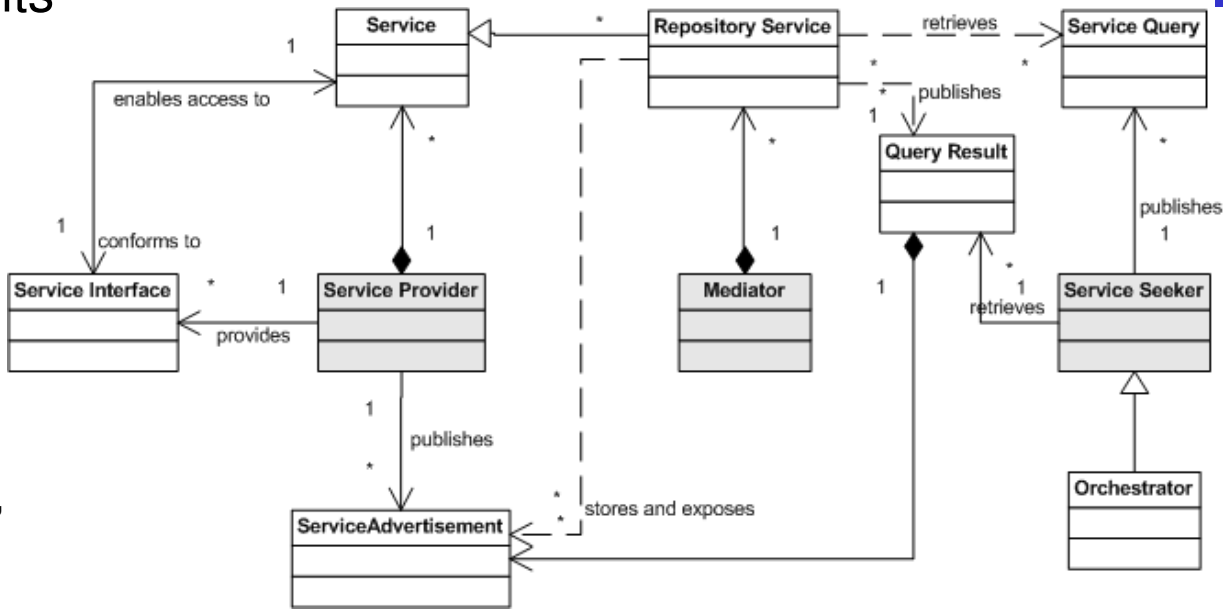
- Service discovery
- On naming
- Distributed flat resolution
  - DHT (distributed hashtable) of Chord
- Distributed structured resolution
  - File systems
  - DNS
- mDNS + DNS-SD

Service discovery model

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

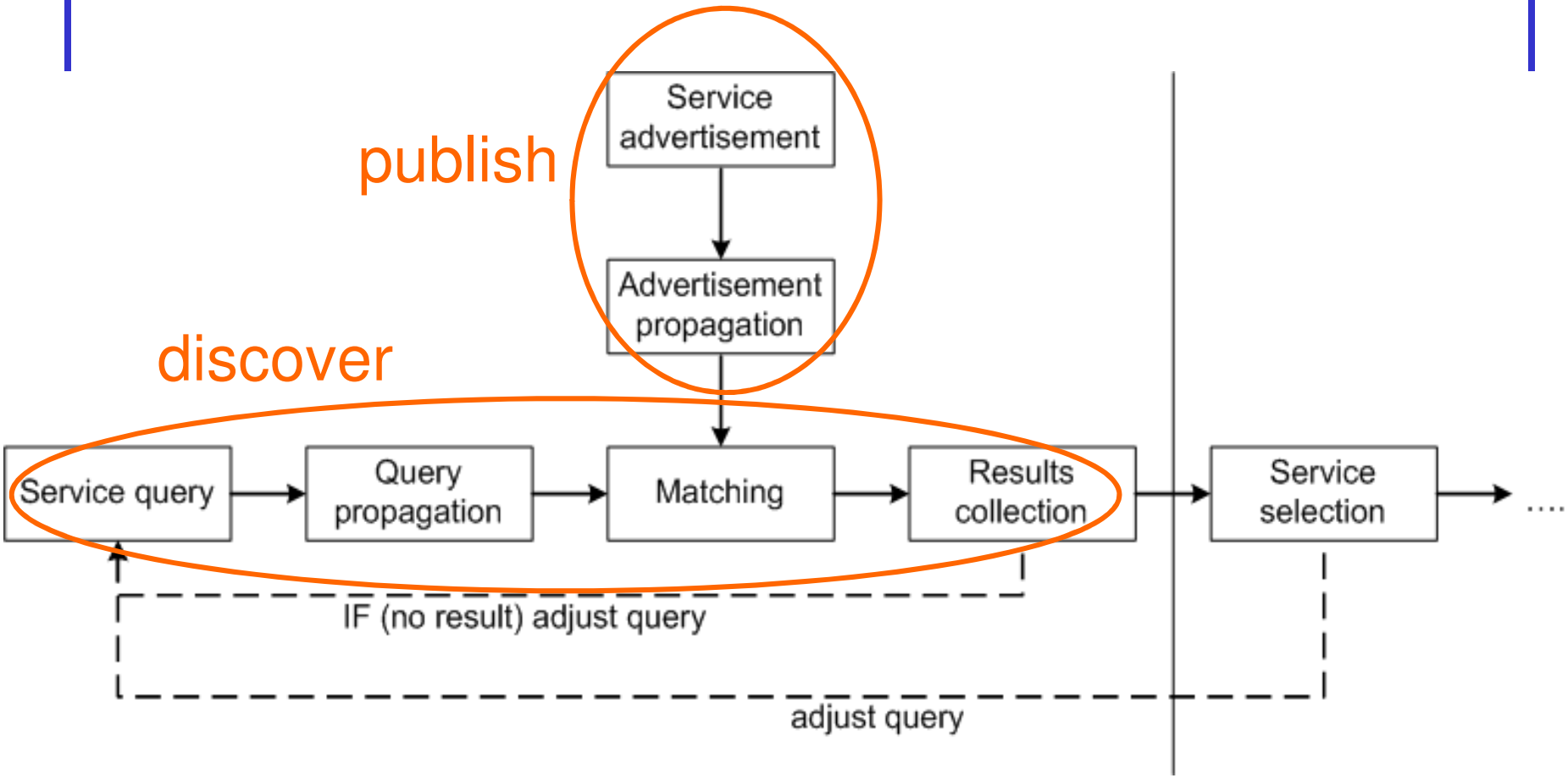
Architectural elements

- Three parties:
  - provider, seeker,
  - mediator (broker)
- Tasks:
  - advertisement
    - of service & query,
  - propagation,
    - of service & query,
  - matching,
  - evaluation



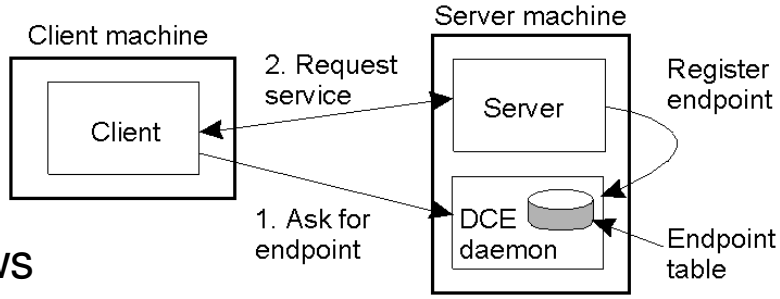
Slides based on work by Melissa Tjiong, SAN, TU/e

Service discovery process



Discovery techniques

Client directly contacts *known access point (AP)* for server access
- based on globally accepted list of services
- example IANA: services coupled to fixed transport port, e.g., ftp: 21, telnet: 23 etc.



(a)

This can be the AP of a mediator that knows the AP of the service in which the client is really interested. Two possible schemes are:

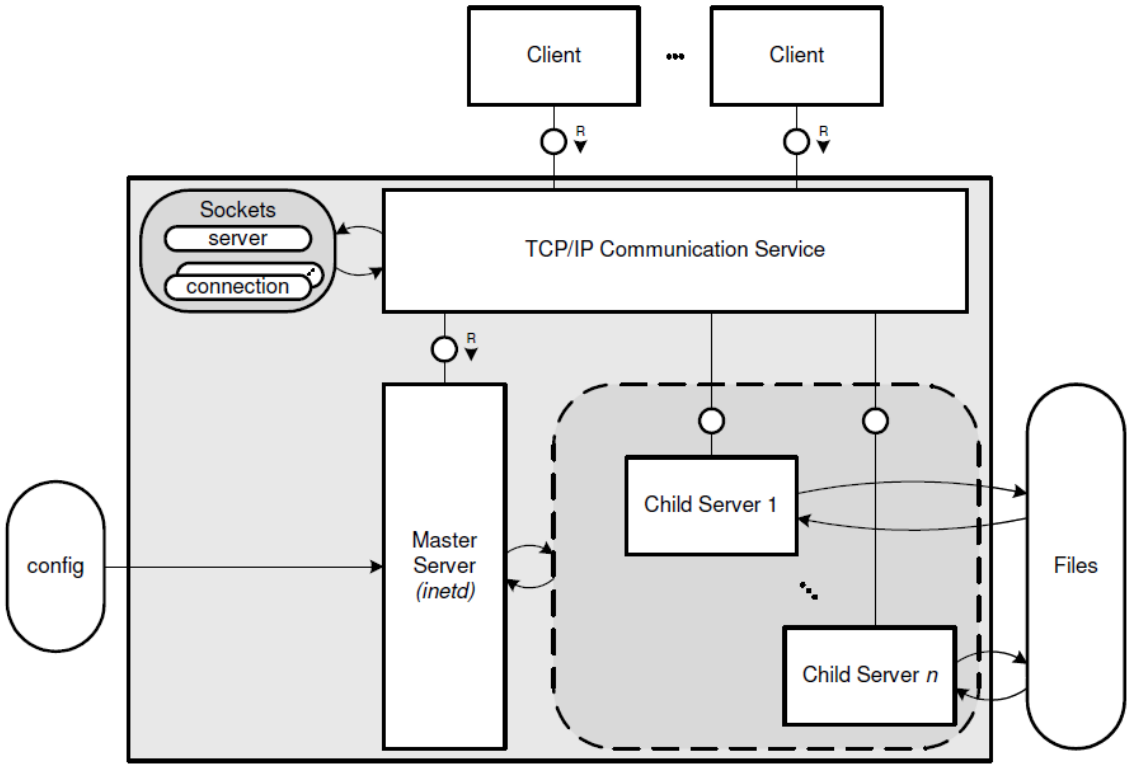
a) Server registers at repository
   - can be on different machine

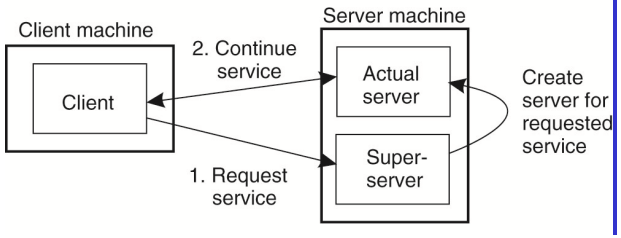
   client contacts repository for access point, e.g., DCE, DNS, SLP

b) 'Superserver', listens to many access points
   - superserver instantiates particular ser on request and hands-over the reque
   - example: *inetd* (internet daemon)
   - just start the server beforehand on the known access point

Note: both the repository (a) and the superserver (b) are themselves servers.

Inetd example (create server upon request)

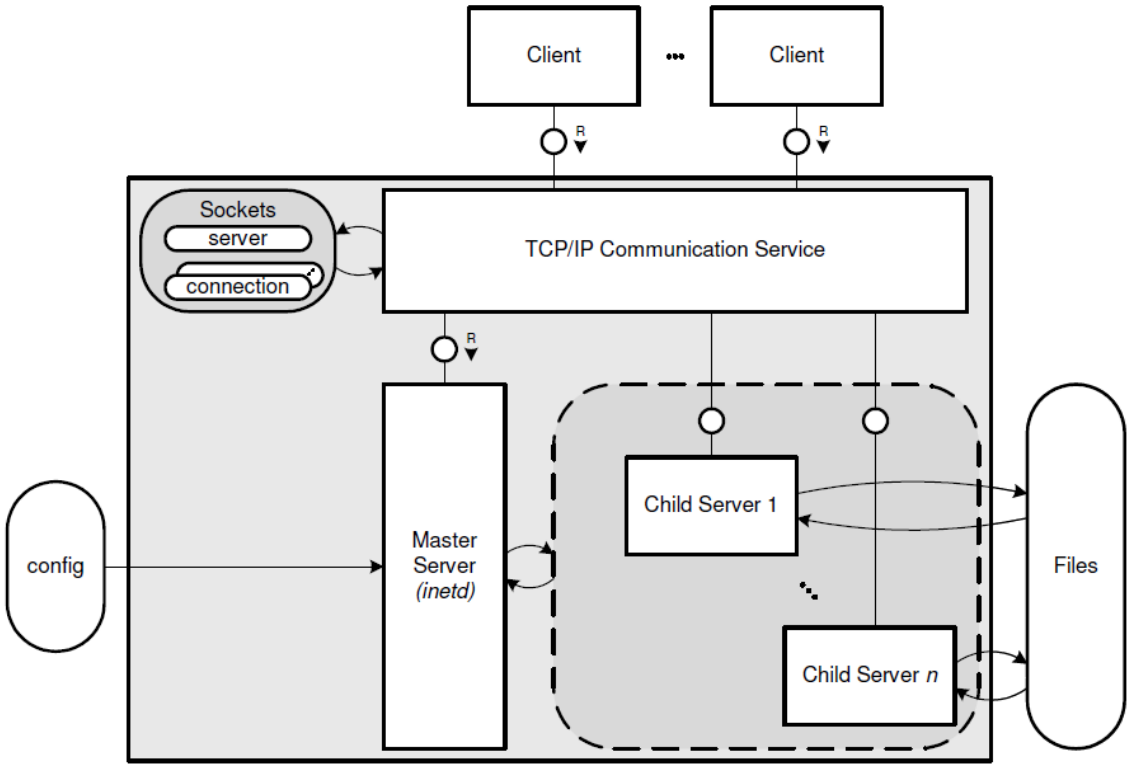


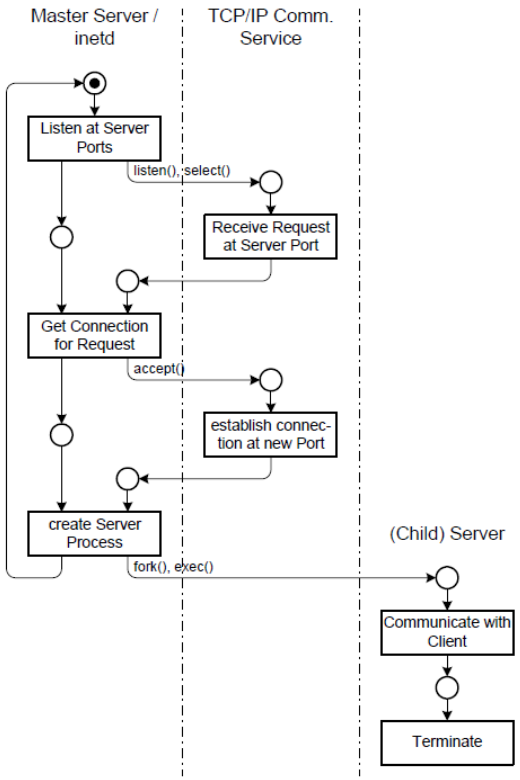


(b)

Picture from 'The Apache Modeling Project',
Grone, Knopfel, Kugel, Schmidt, Potsdam University

Inetd example (create server upon request)

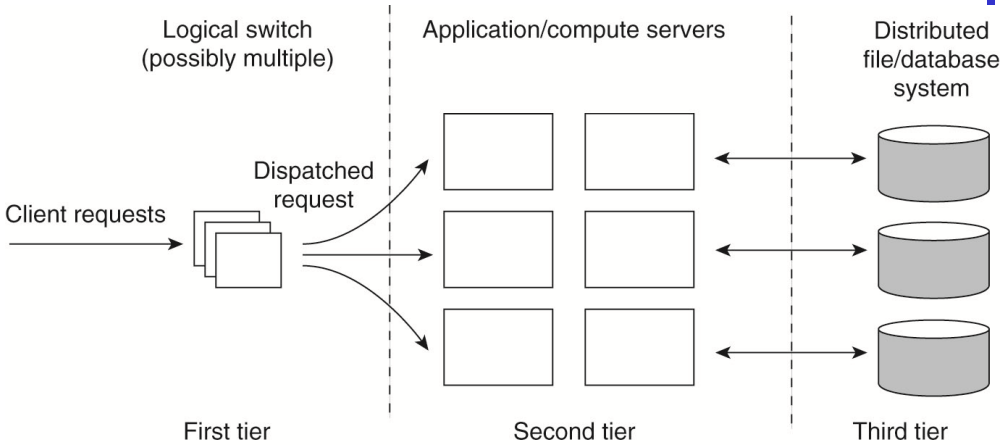




Pictures from 'The Apache Modeling Project',
Grone, Knopfel, Kugel, Schmidt, Potsdam University

Distributed servers (access to clusters)

- Goal of cluster
  - improve performance
    - handle larger load
    - decrease delay
  - improve reliability
  - improve availability

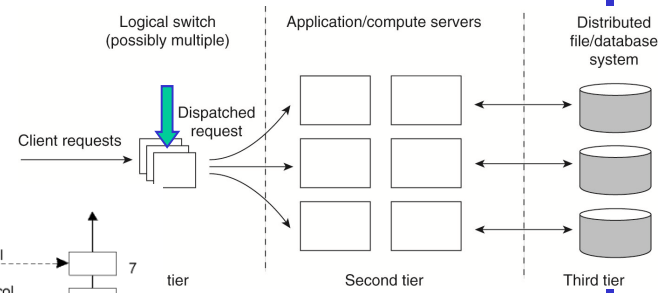


Server cluster general architecture

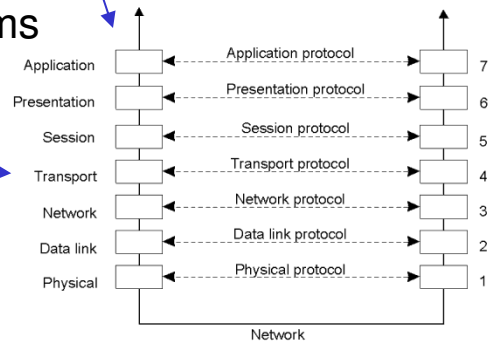
- Problems to be solved:
  - transparent (for client) request dispatching
    - particularly: dealing with the single access-point
  - security
  - management
    - see Planetlab case study in the book

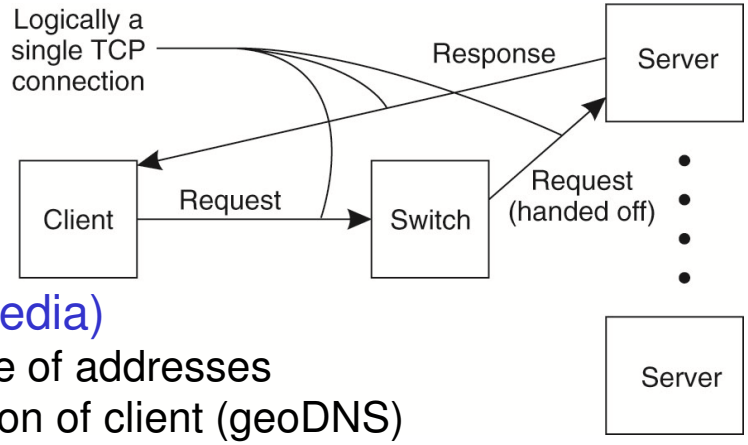
Cluster access

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



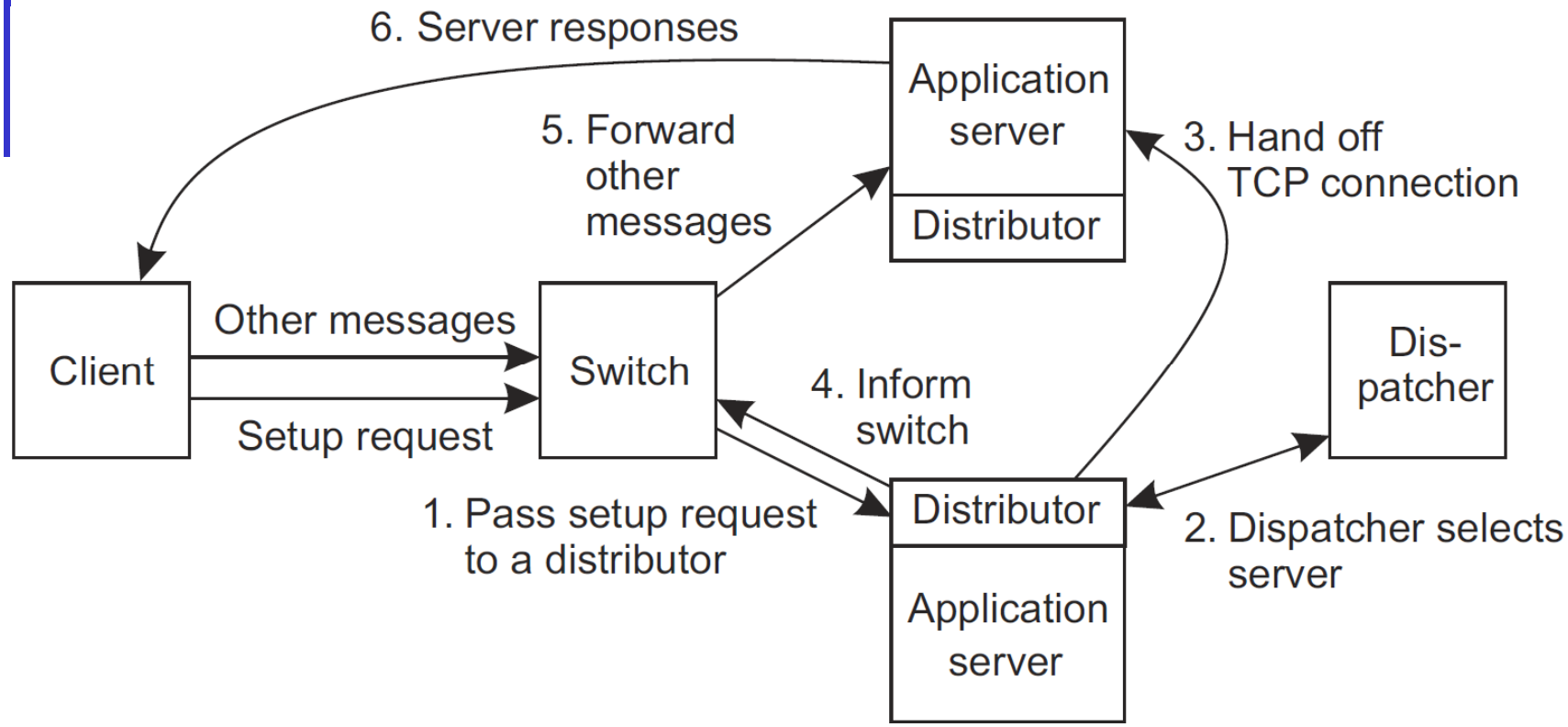
- TCP handoff
  - incoming transport-layer request passed to the server
  - needs protocol adaptation
    - 'spoofing' of the destination ID
    - switch routes all request traffic
    - OS support required (both sides)





- 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)

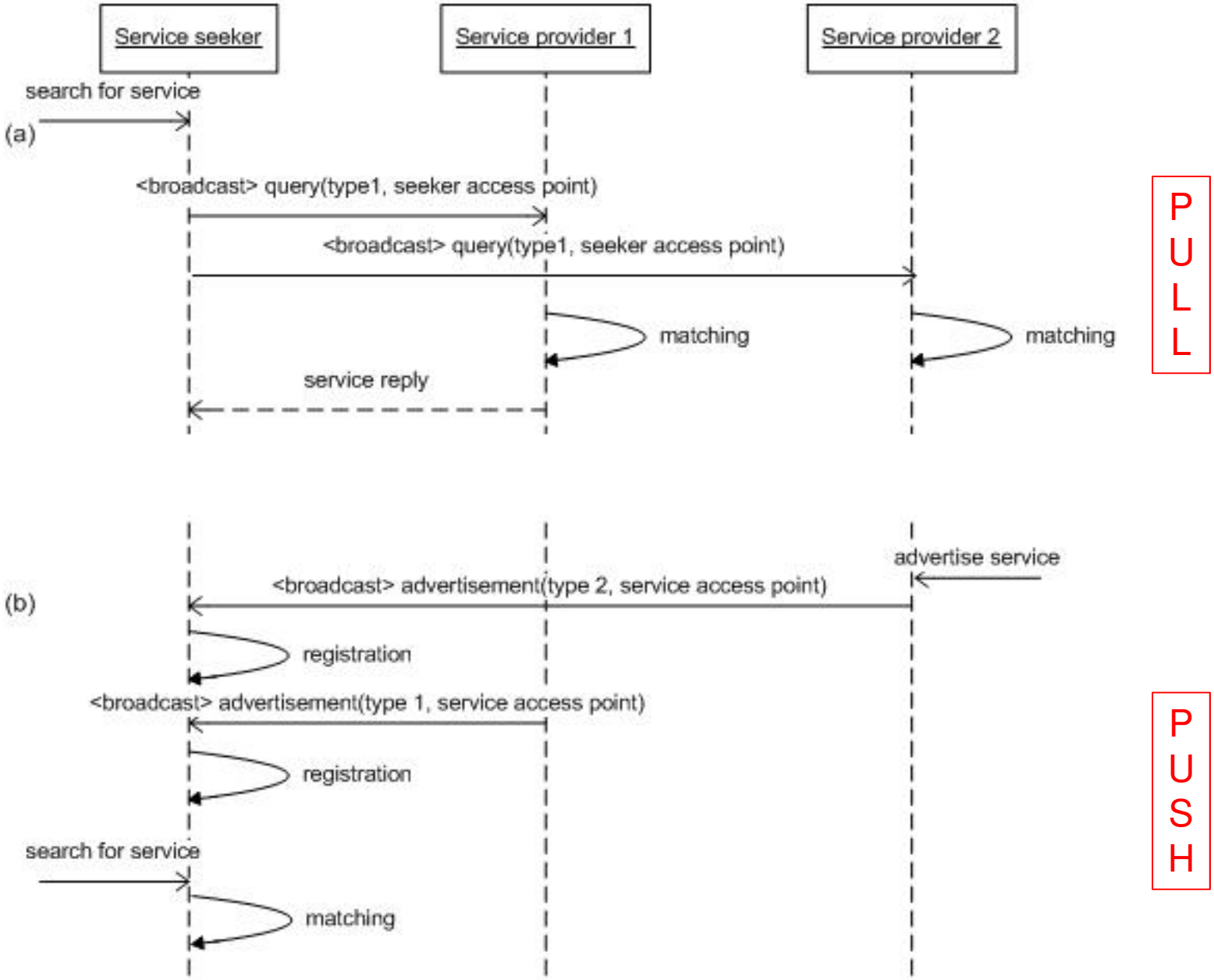
Content-aware dispatching



Further discovery 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
    - use multicast as query
  - examples: unmanaged SLP, DHCP, SSDP (UPnP), Apple Bonjour (rendez-vous), mDNS
- Important for *bootstrapping, e.g. to acquire an IP-address*
- 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

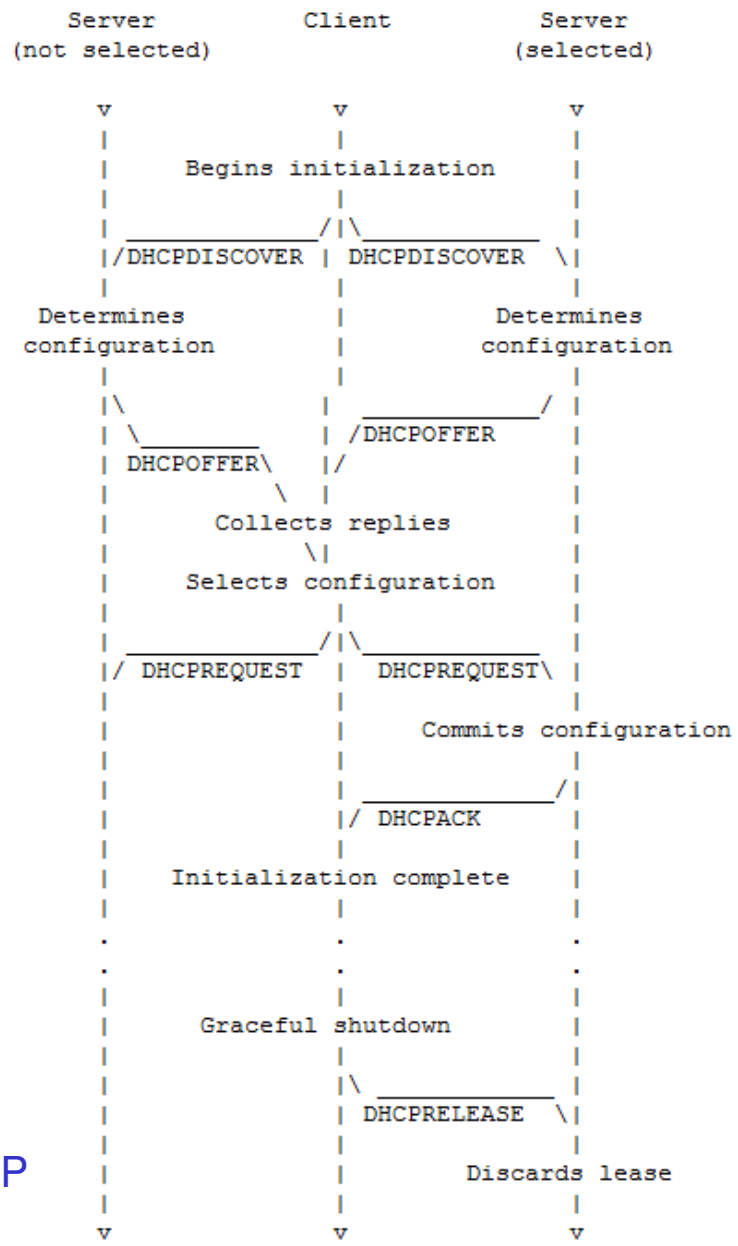
Immediate architecture



DHCP

- Immediate, bootstrapping protocol, provides
  - IP address, with a lease
  - addresses of *caching DNS servers*
- Client side broadcast on local net
- Offers by multiple local servers
- Client side selection and contacting of server (step not shown in previous slide)

from RFC 2131: DHCP



Agenda

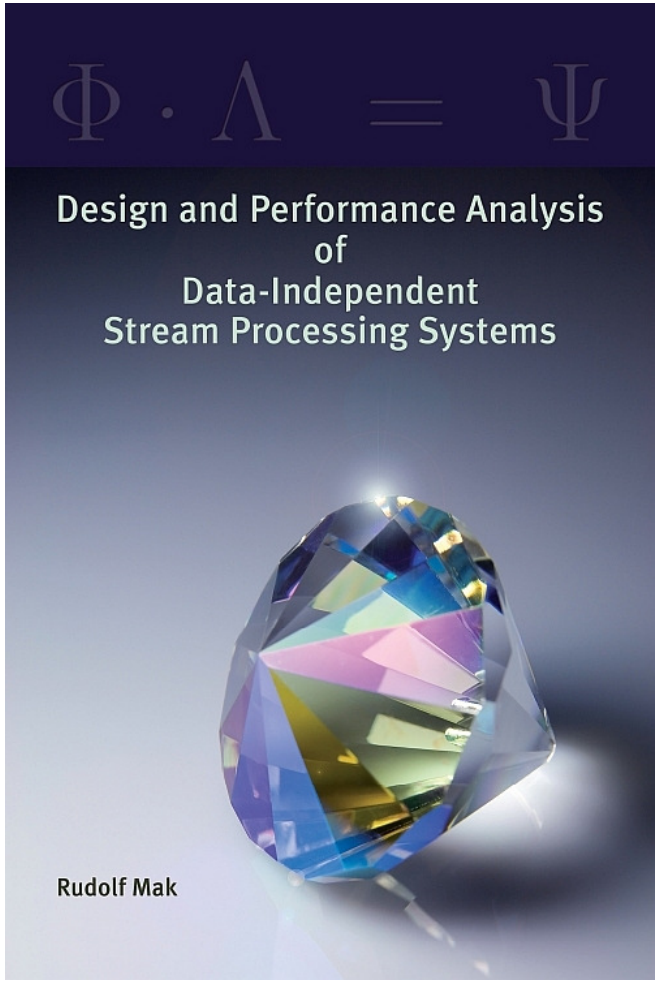
- Service discovery
- On naming
- Distributed flat resolution
  - DHT (distributed hashtable) of Chord
- Distributed structured resolution
  - File systems
  - DNS
- mDNS + DNS-SD

Naming

- A *name* serves as *reference* to an *entity*
  - entity: i.e., anything that can be operated upon
  - names are organized into a name space

- Motivation:
  - Delay binding
    - *mobility transparency*: name remains constant under mobility
    - *replication transparency*: map name to several different entities
  - Lookup
    - *location transparency:* separate name for entity and its access point
    - name serves as a means to locate an entity,
    - In particular, to resolve the name of the entity to its address
  - Human-friendliness

Names are not uniquely identifying



My thesis, not of any particular interest to this course

Names are not uniquely identifying

But if you google
“Rudolf Mak thesis”
you probably find

Ghent University
Faculty of Medicine and Health Sciences

Specific health services for sex workers:
from theory to practice

Rudolf Mak

Thesis submitted in fulfilment of the requirements
for the degree of doctor in medical sciences

Gent, November 22, 2004

Prof. Dr. G. De Backer (promotor)
Prof. Dr. A. Meheus (co-promotor)

Entities and identifiers

- *E:* Entities
  - the collection of 'things' that need to be referred
    - resources, devices, network ports, ...
- *I:* Identifiers
  - the namespace for entities
- A true identifier has the following properties:
  - each identifier refers to at most one entity
  - each entity is referred to by at most one identifier (unique)
  - an identifier always refers to the same entity: is never reused

So, there exists an injective and static mapping $Id: E \rightarrow I$

- Examples:
  - MAC address as identifier for a network card,
  - ISBN number of a text

Access points and addresses

- *X:* Access points
  - an 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*
  - an entity can change its access point(s) over time
- *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. (ARP-protocol)
  - The network card is found directly through this identifier.

Name system/service

- A name system maintains a name-to-address binding
  - In its simplest form a very big table
  - Often in the form of a distributed database

- A *name service* answers queries for attributes of named entity
  - Typically, but not necessarily the address
  - A name server may only answer to queries for part of the name space
    - its domain of authority
  - May invoke other servers to assists in answering the query
    - In case of a distributed database

Name space

- Set (of valid names)
  - specification/algorithm as how to validate/generate names
- Methods
  - flat naming
    - simply a set of unstructured meaningless names
  - structured naming
    - hierarchical, mostly, e.g., /user/johanl/....
      - *relative* name: relative with respect to a given context
      - *absolute* name: complete
    - prefixing
      - embedding a set of names uniquely into a larger set, e.g.,

```
<gm:forcast xmlns:gm="http://GlobalMeteo.com">
    <gm:rain> … </gm:rain> …
</gm:forcast>
```

Hierarchic name space

Advantages

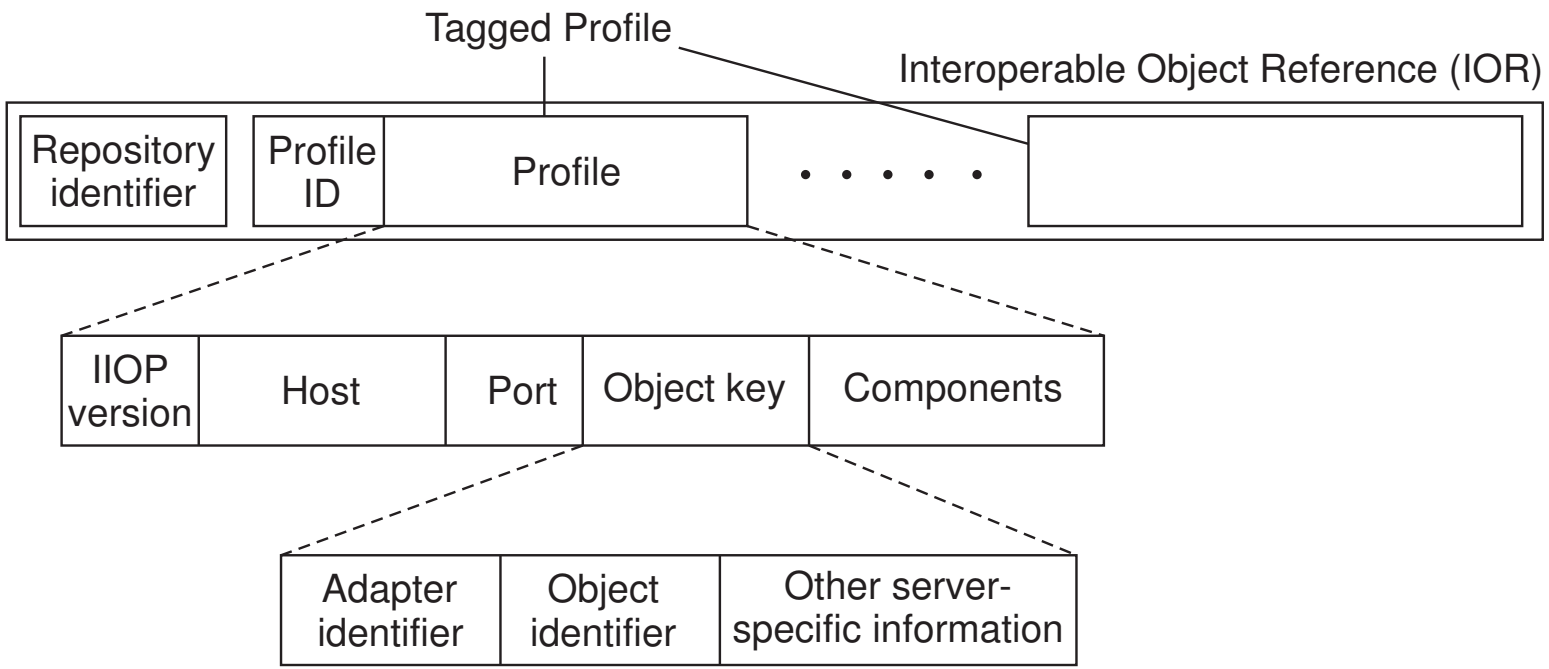
- Unbounded in size
  - keep adding new levels of hierarchy
- Better management
  - Can reflect organizational hierarchy
  - Name resolution can be done in steps
  - Each part is resolved to a small context
  - Name resolution can be distributed over organizations

Disadvantage

- Detection of invalid names may take longer
  - After partial resolution

IOR with specific information for IIOP

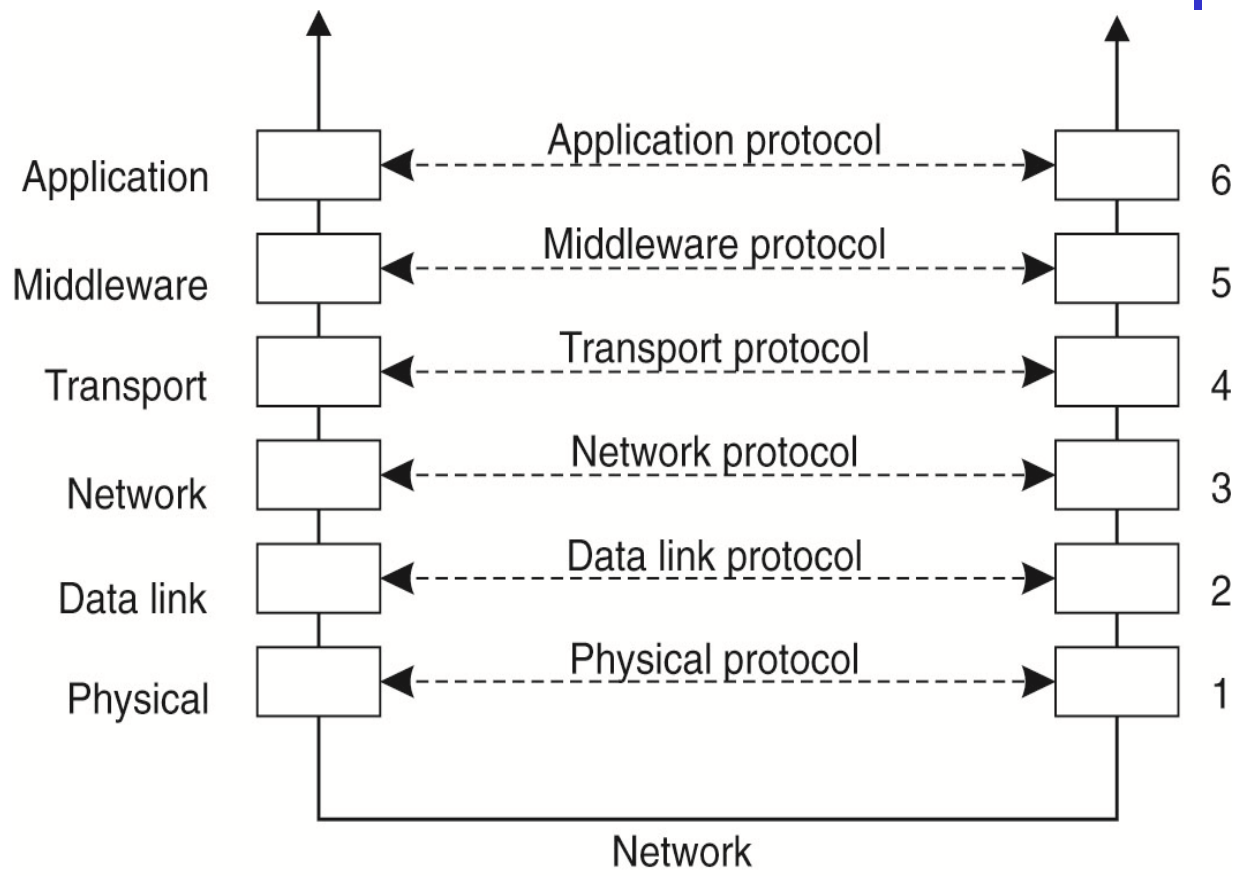
- Another example of a hierarchical structured name



Taken from: Tanenbaum v Steen, fig 10-11

Naming in the Internet stack

Structured/flat	Name
structured, both	DN, URL
(structured, flat)	IOR, Hash
(structured, flat)	(IP, port)
structured	IP address
flat	MAC address



51.4477965, 5.4843428

Binding and 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 ARP: what is the IP address of this MAC address?
    - given identifier, determine access point
    - in principle: reverse DNS: what is the DNS name of this IP address?
      - not entirely flat, however, as still information is remaining in the IP address
      - and the 'in-addr.arpa' domain supports reverse lookup
- **Closure mechanism:** knowing where and how to start resolution

Agenda

- Service discovery
- On naming
- Distributed flat resolution
  - DHT (distributed hashtable) of Chord
- Distributed structured resolution
  - File systems
  - DNS
- mDNS + DNS-SD

Flat naming and resolution

Problem: given identifier, find access point

- possibly, via an intermediate structured address

Solution: search (see: service discovery)

- broadcast: 'ask everyone'
  - limited physical scope
  - example: ARP (maps IP-address to physical (MAC) address.)
- multicast
  - e.g. overlay networks
  - limited logical scope
  - example: searching in P2P overlay
  - may combine with *repositories* storing *associations:* (identifier, address) pairs
- 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 E (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 $id(N) = \{id(n) | n \in N\}$
  - The identifier $id(e)$ of an entity $e \in E$ is called its key

- An entity e with key *k* has an associated <u>node</u>, the one with the next largest or equal *id,* modulo 2^m
  - $\boldsymbol{succ(k) = \begin{cases} min\,\{n \in id(N) \mid n \geq k\} & if\ k \leq max(id(N)) \\ min(id(N)) & otherwise \end{cases}}$

  Function name *succ* is used for consistency.
  Not a very good name: *home node* or *owner node* would be better.

Example: distributed storage and lookup

- 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:

    $pred(p) = y \equiv (succ(y+1) = p \text{ and } succ(y) = y)$
  - successor and predecessor are nodes!

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

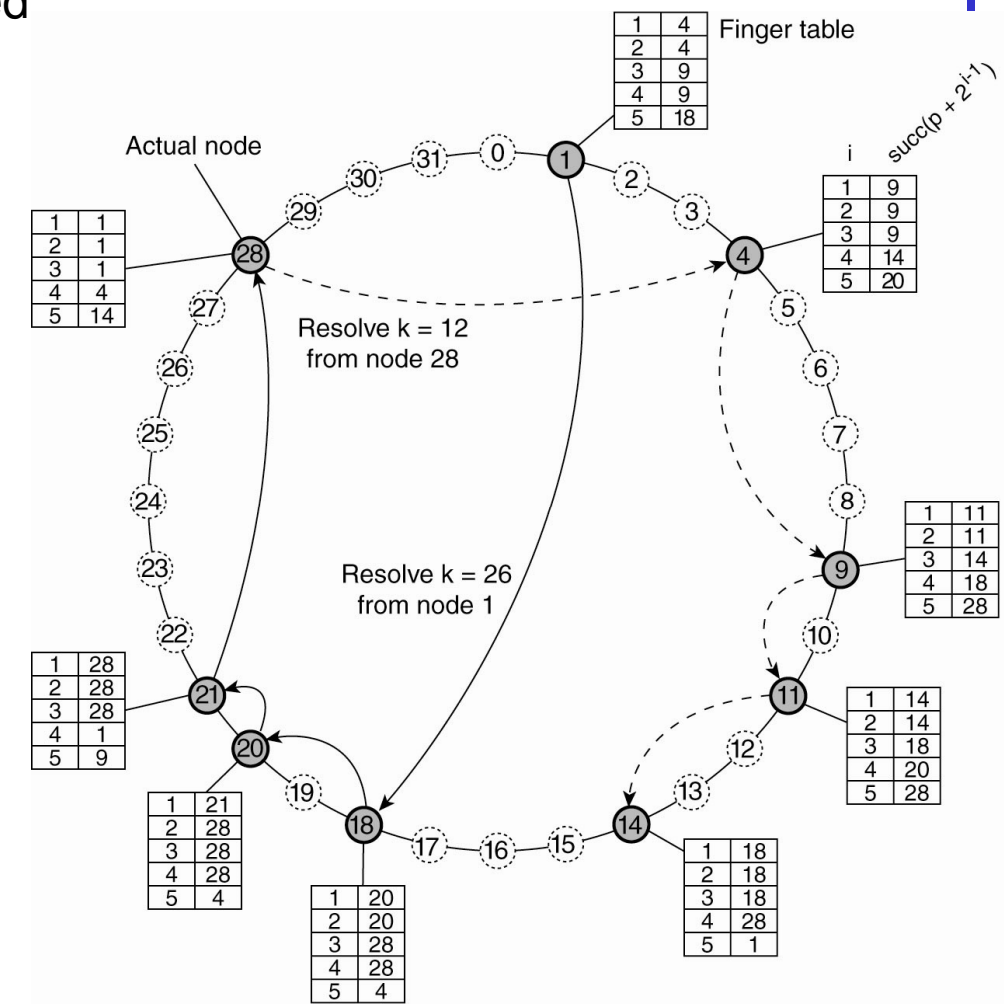
- Hence, the problem is:
  - given *k*, find (the address of) *succ(k)*

Example

- Assume *m = 4, N = { 2, 5, 8, 12 }*
- *succ(4), succ(5), succ(6), succ(13)?*
  - *succ(4) = 5, succ(5) = 5, succ(6) = 8, 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*
  - *13, 14, 15, 0, 1, 2*
- *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 *succ(p+1)* and *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 $succ(p+2^{i-1}),\ 1< i \leq m$
    - Chord system: 'finger tables'
- Issues in:
  - entity insert: simple
  - node insert:
    - table updates in DHT
    - relocate entities
  - routing efficiency - use the underlying network efficiently



Exercise

Given: $m = 6, \quad id(N) = \{3, 8, 23, 44, 51, 53\}$.

Question: What is the average number of hops taken to look up a resource, i.e., resolve a key, starting from node 51, assuming each node knows it predecessor?

Answer:

For $45 \leq k \leq 51$, it takes 0 hops: (*succ*(*k*) = 51)

For *k* = 52*;* 53*;* 3*;* 23, it takes 1 hop: (51 -> *succ*(*k*))

For $54 \leq k \leq 2$(= 66 mod 64), it takes 2 hops: (51 -> 53 -> *succ*(*k*))

For $4 \leq k \leq 8$, it takes 2 hops: (51 -> 3 -> *succ*(*k*))

For $9 \leq k \leq 22$, it takes 3 hops: (51 -> 3 -> 8 -> *succ*(*k*))

For $24 \leq k \leq 44$, it takes 2 hops: (51 -> 23 -> *succ*(*k*))

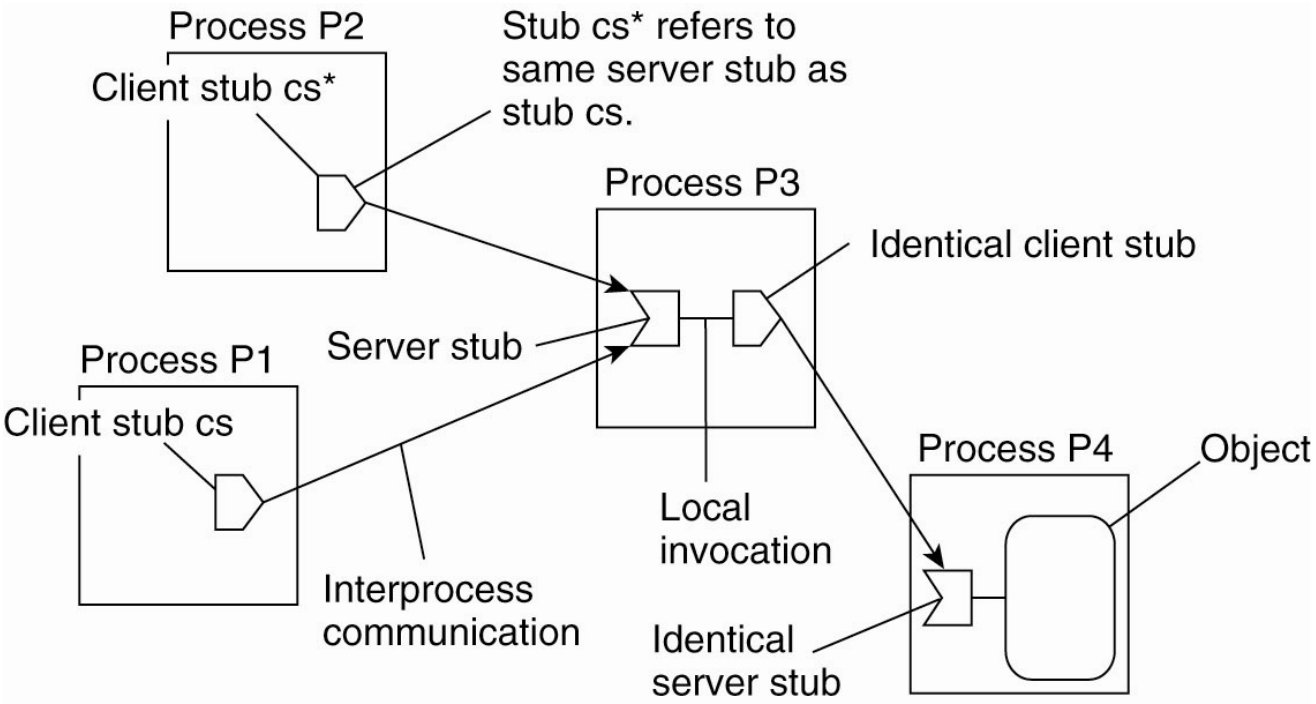
So the average becomes ((7*0)+(4*1)+((13+5+21)*2)+(14*3))/64 = 124/64

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?)

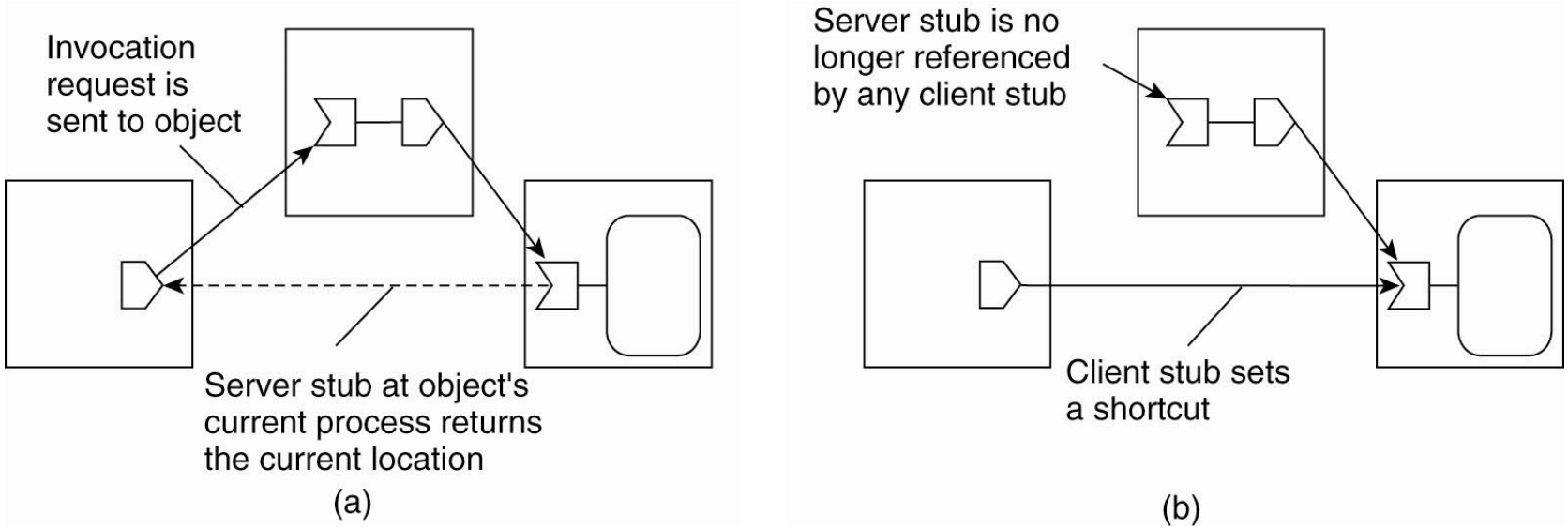
Follow me via stubs

Mobile object replaced by stub that forwards invocations, when object moved from Process P3 to P4



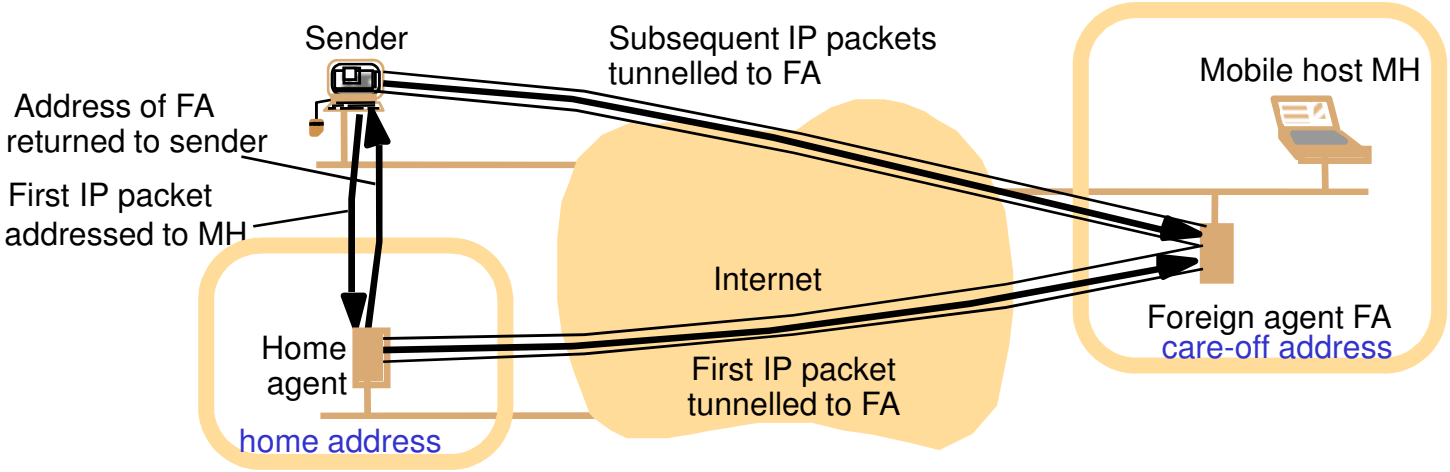
Pointer fusion

- final stub responds with new reference
  - not transparent
  - requires garbage collection



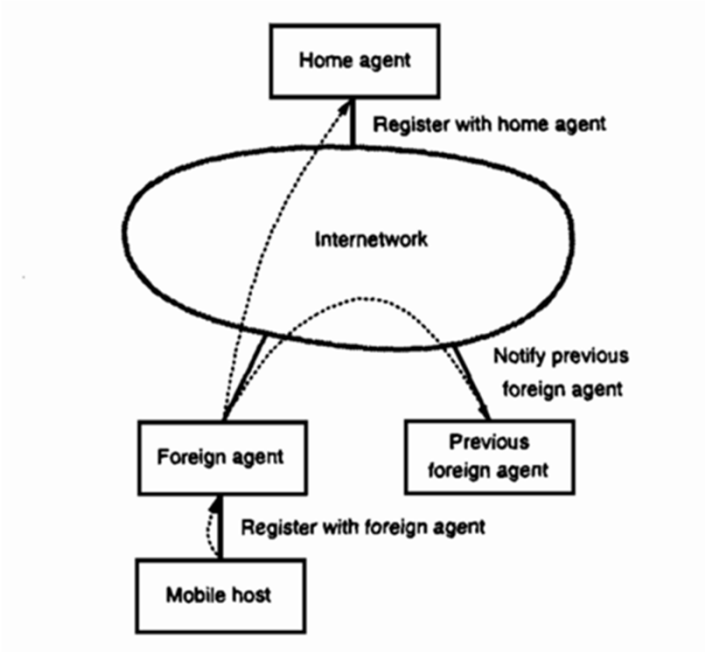
The Mobile IP routing mechanism

- *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
  - All or only the first packet, in which case it informs the client of the care-off address.
- 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 IP

- 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



Pictures taken form Helsinki University of Technology, Seminar on MM

For detailed information see https://tools.ietf.org/html/rfc3344

Hierarchical naming systems

- Graph: labelled, 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
    - "/", "."

Agenda

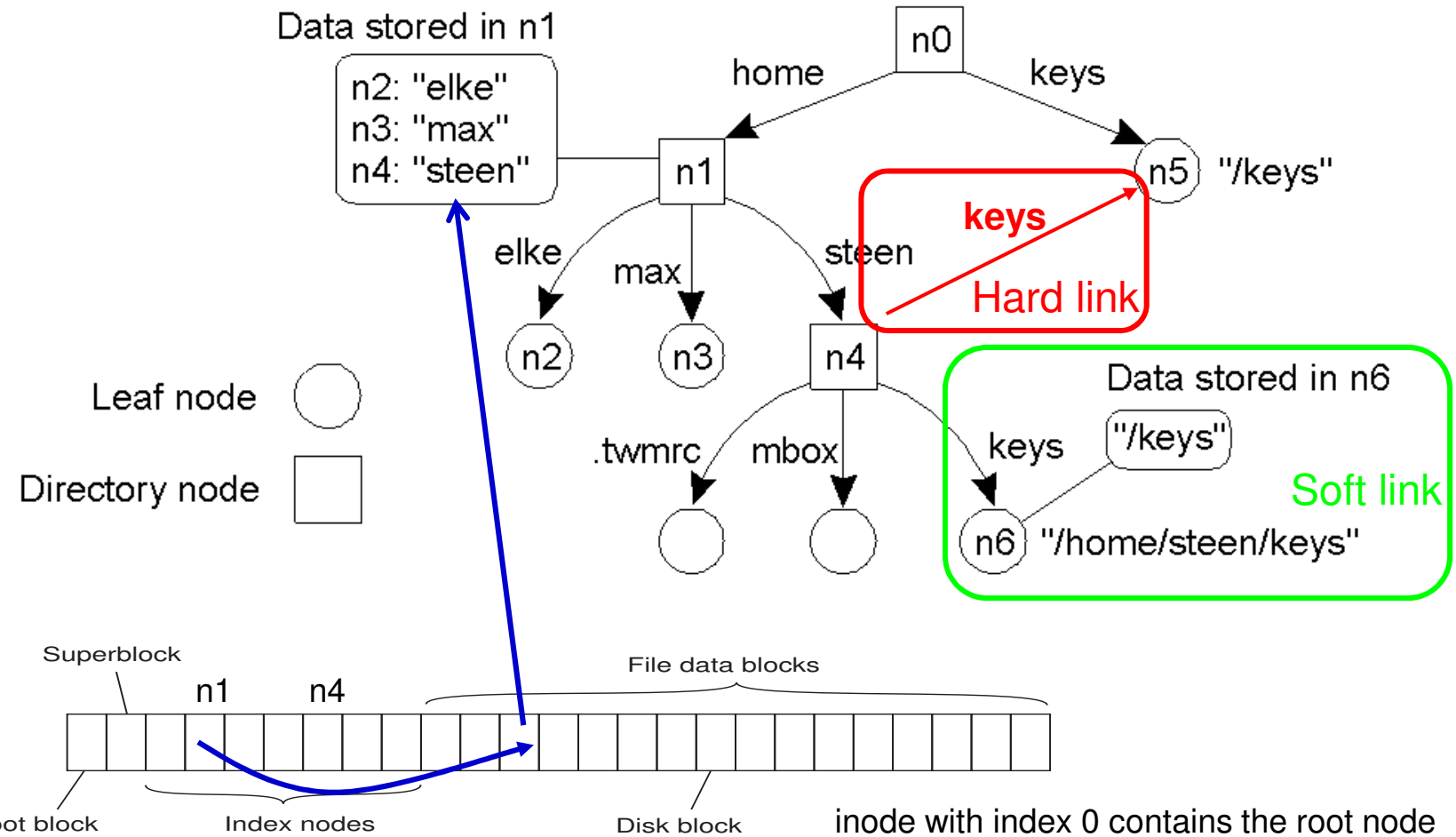
- Service discovery
- On naming
- Distributed flat resolution
  - DHT (distributed hashtable) of Chord
- Distributed structured resolution
  - File systems
  - DNS
- mDNS + DNS-SD

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
- ftp URL
  - combines several naming systems
  - Question: what is the resolution procedure here? and the closure?

File system: naming graph

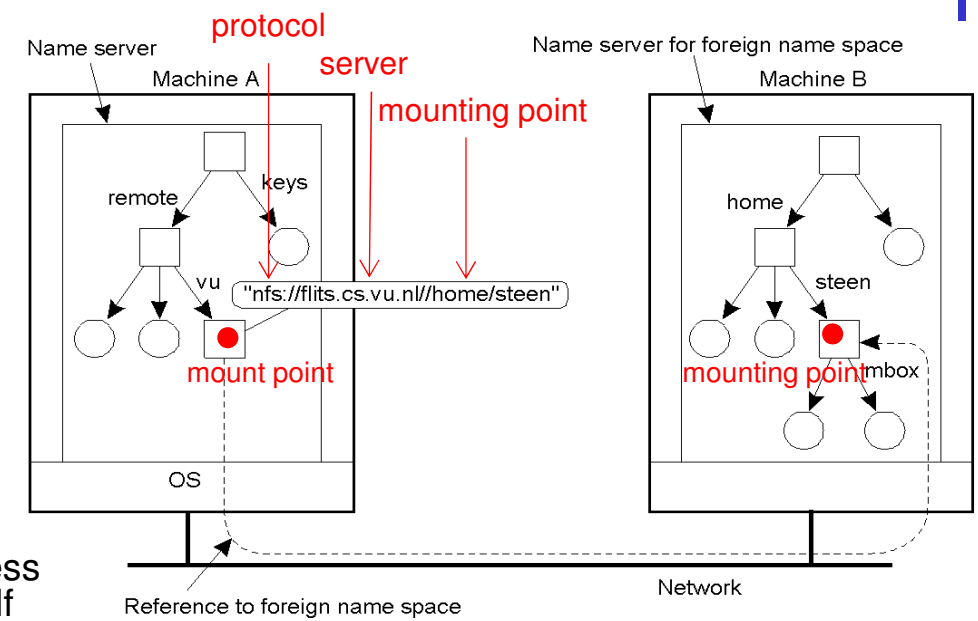
link ("/keys", "home/steen/keys")



inode with index 0 contains the root node

Joining namespaces

- Direct catenation, as in some URL schemes
  - ftp://ftp.win.tue.nl/johanl
- 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 service 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

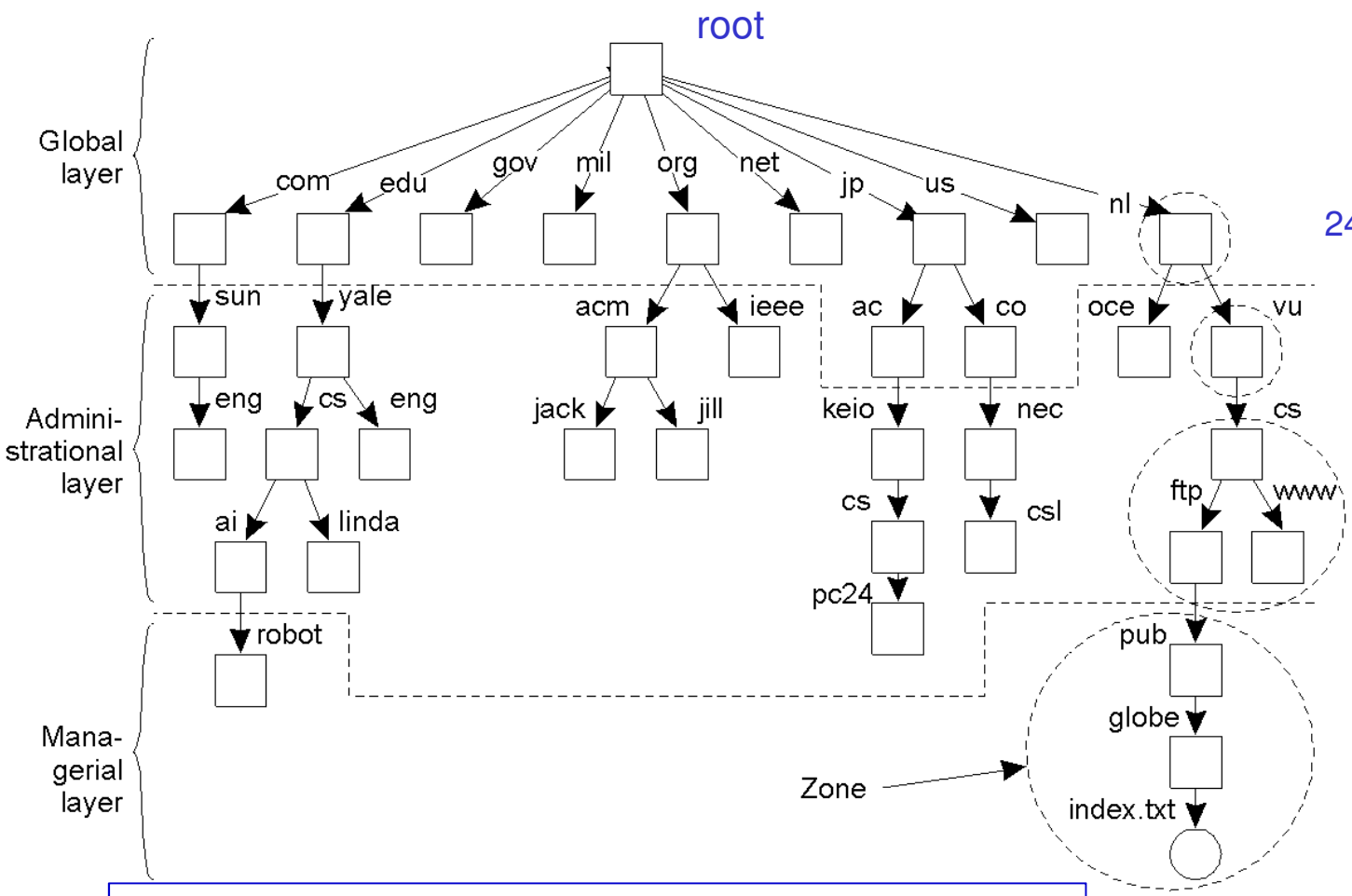
- For performance, scalability and dependability:
  - distribute the data (i.e., directory nodes and subgraphs)
  - distribute the process of name resolution

- Levels in the naming graph
  - Global
    - jointly managed by several administrations
    - changes are rare
  - Administrational
    - nodes managed by a single administration
  - Managerial
    - nodes within single administration mapped onto local nameservers
    - frequent changes

Comparison of layers

Item	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

Example: DNS namespace



(2010)
root zone:
originally 7 generic
top-level domains
248 country-code TLDs

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

For more see: https://icannwiki.org/Portal:TLDs

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 countries or special organizations
  - Administrational
    - 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

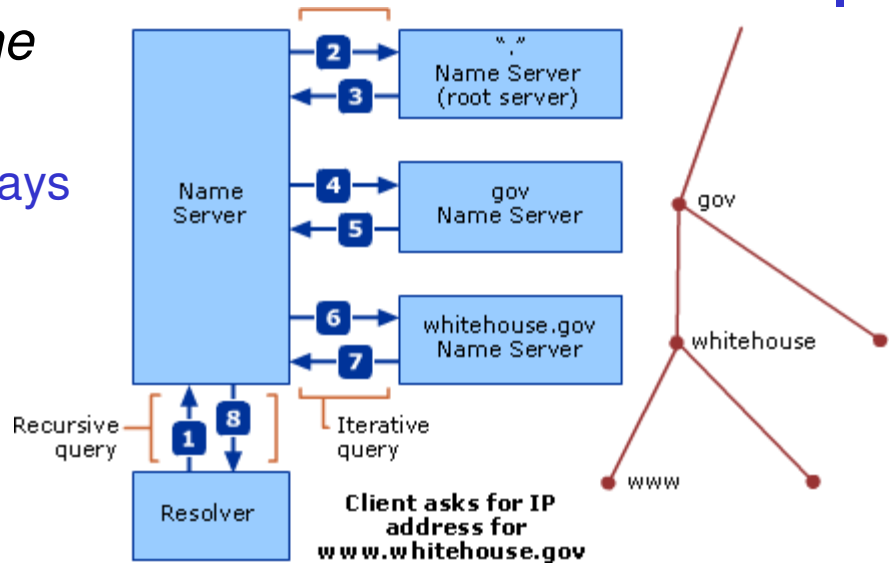
DNS, extra-functional properties

- 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
  - applied tactics
    - name space partitioning (distributed database), server replication, caching

Rudolf Mak TU/e Computer Science

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*

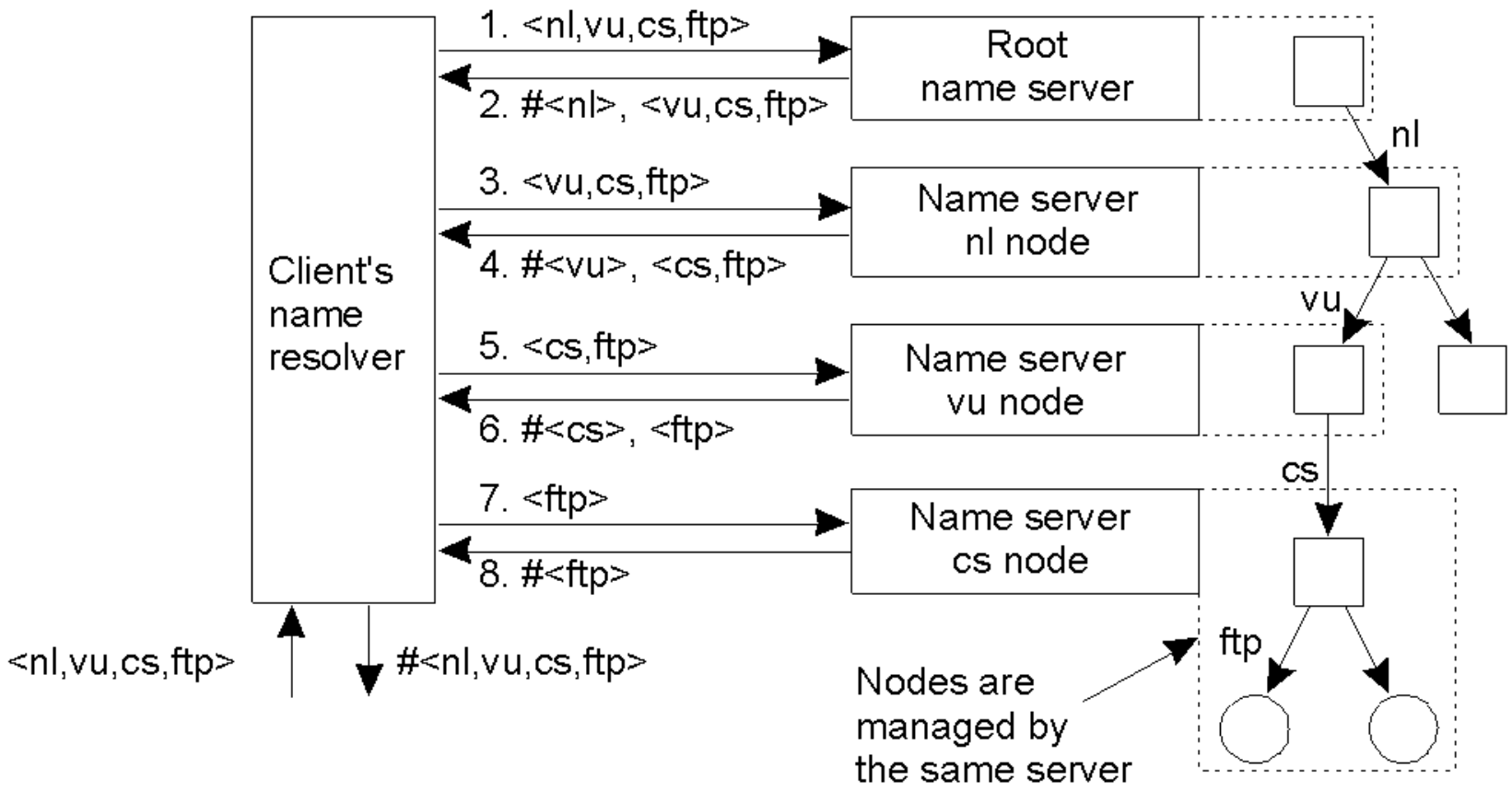


model in process view from Microsoft technet

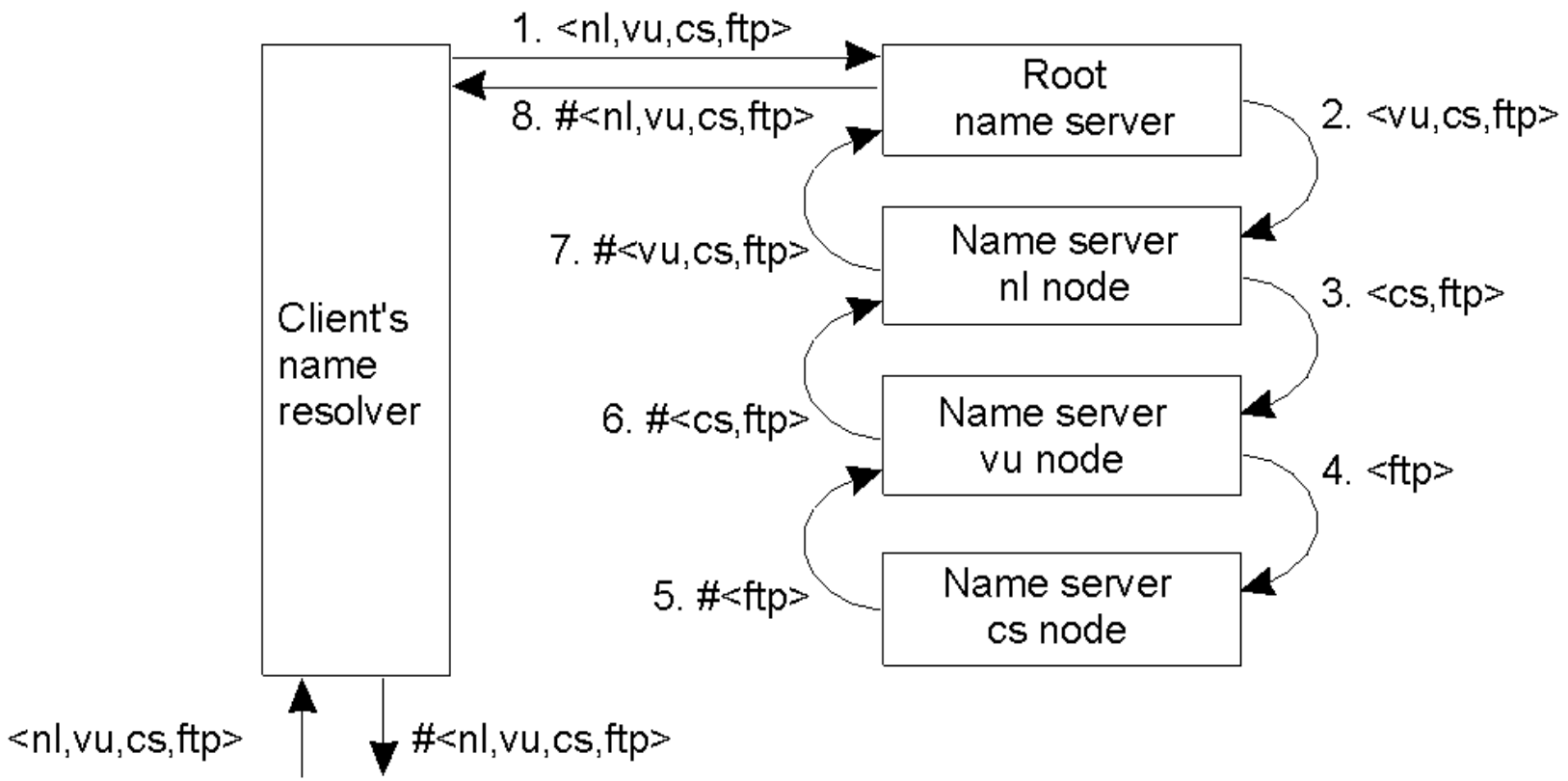
Distributed resolution

- Query: **resolve(dir, name1, name2, ...)**
- Server responsible for directory node **dir** performs one step, i.e., resolves **name1** 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


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>
Client's name resolver
Root name server
Name server nl node
Name server vu node
Name server cs node
nl
vu
cs
ftp
<nl,vu,cs,ftp>
#<nl,vu,cs,ftp>
Nodes are managed by the same server

Recursive

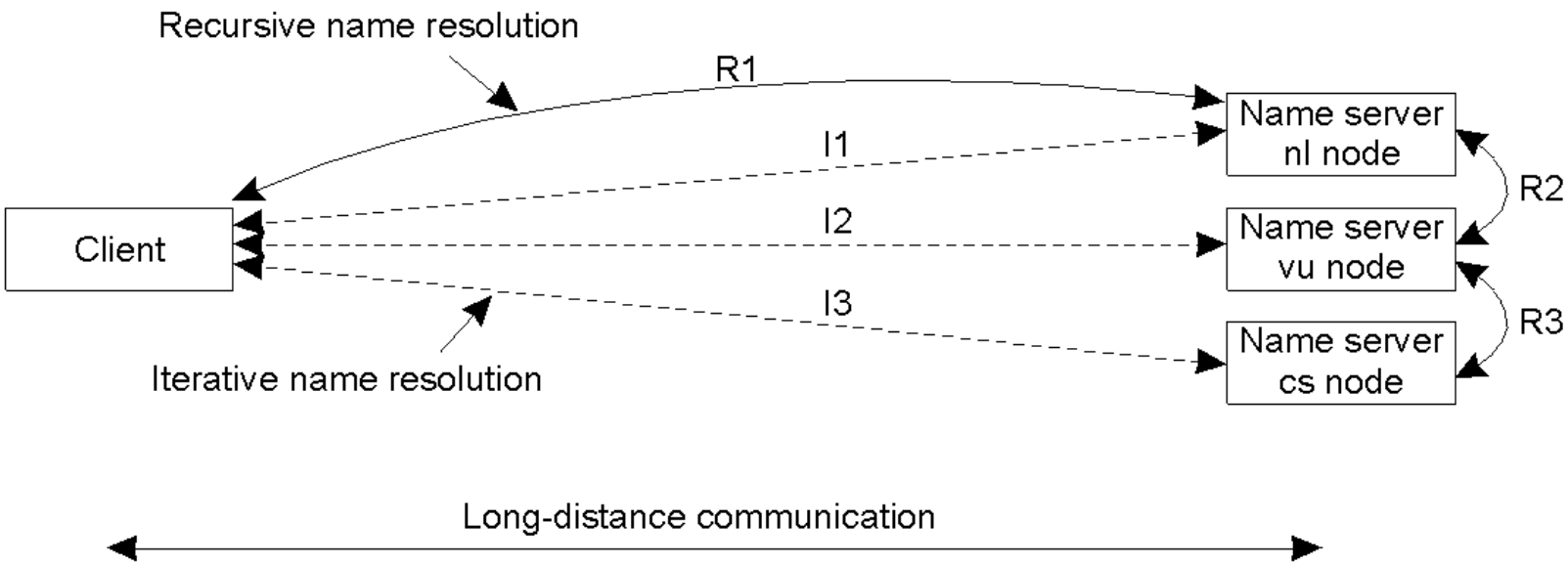

1. <nl,vu,cs,ftp>
8. #<nl,vu,cs,ftp>
Root name server
2. <vu,cs,ftp>
7. #<vu,cs,ftp>
Name server nl node
3. <cs,ftp>
Client's name resolver
6. #<cs,ftp>
Name server vu node
4. <ftp>
5. #<ftp>
Name server cs node
<nl,vu,cs,ftp>
#<nl,vu,cs,ftp>

Caching

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>

- 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



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 types in DNS

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 domain
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

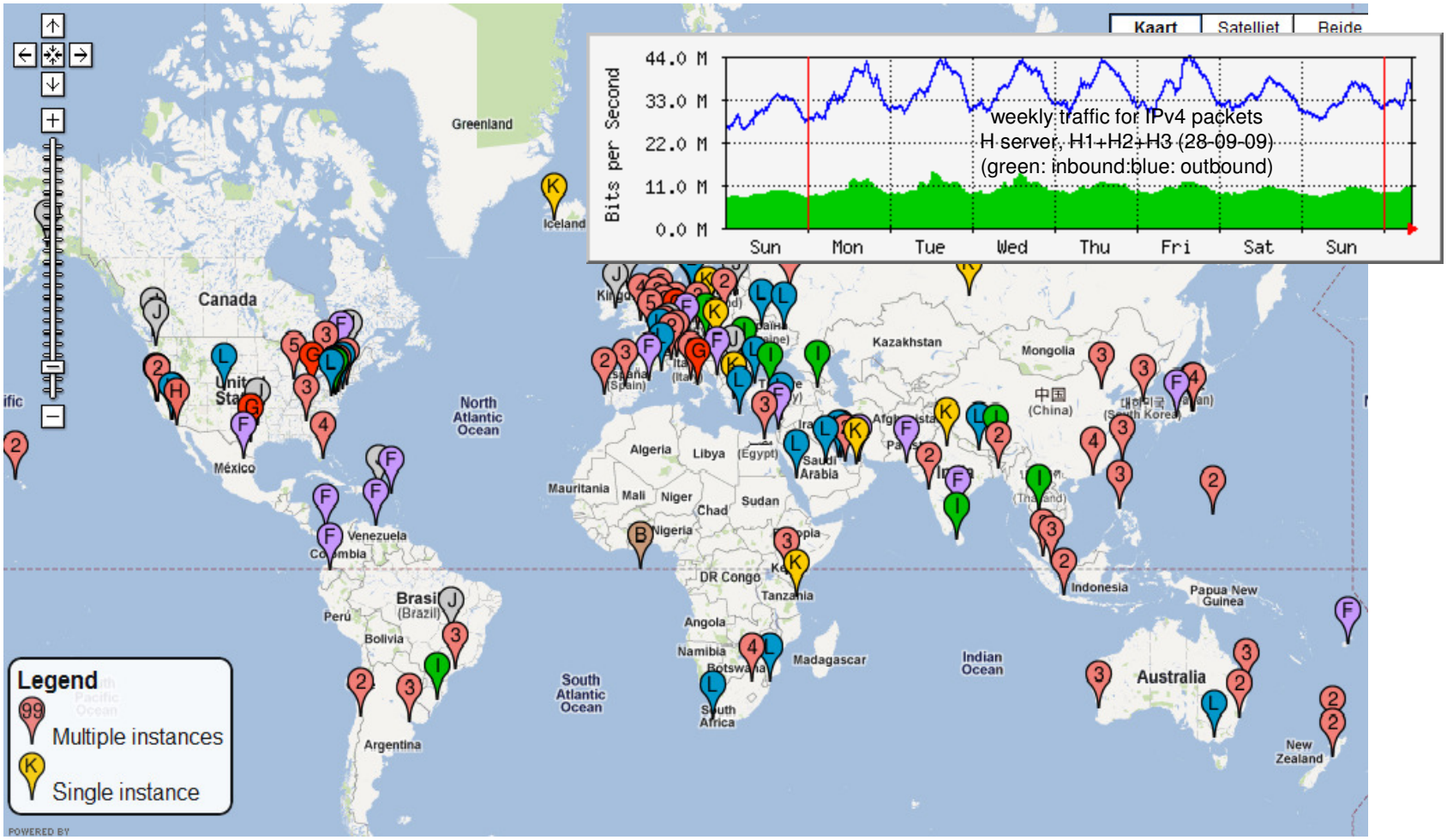
Example database for domain cs.vu.nl

Name	Record type	Record value
cs.vu.nl.	SOA	star.cs.vu.nl. hostmaster.cs.vu.nl. 2005092900 7200 3600 2419200 3600
cs.vu.nl.	TXT	"Vrije Universiteit - Math. & Comp. Sc."
cs.vu.nl.	MX	1 mail.few.vu.nl.
cs.vu.nl.	NS	ns.vu.nl.
cs.vu.nl.	NS	top.cs.vu.nl.
cs.vu.nl.	NS	solo.cs.vu.nl.
cs.vu.nl.	NS	star.cs.vu.nl.
star.cs.vu.nl.	A	130.37.24.6
star.cs.vu.nl.	A	192.31.231.42
star.cs.vu.nl.	MX	1 star.cs.vu.nl.
star.cs.vu.nl.	MX	666 zephyr.cs.vu.nl.
star.cs.vu.nl.	HINFO	"Sun" "Unix"
zephyr.cs.vu.nl.	A	130.37.20.10
zephyr.cs.vu.nl.	MX	1 zephyr.cs.vu.nl.
zephyr.cs.vu.nl.	MX	2 tornado.cs.vu.nl.
zephyr.cs.vu.nl.	HINFO	"Sun" "Unix"

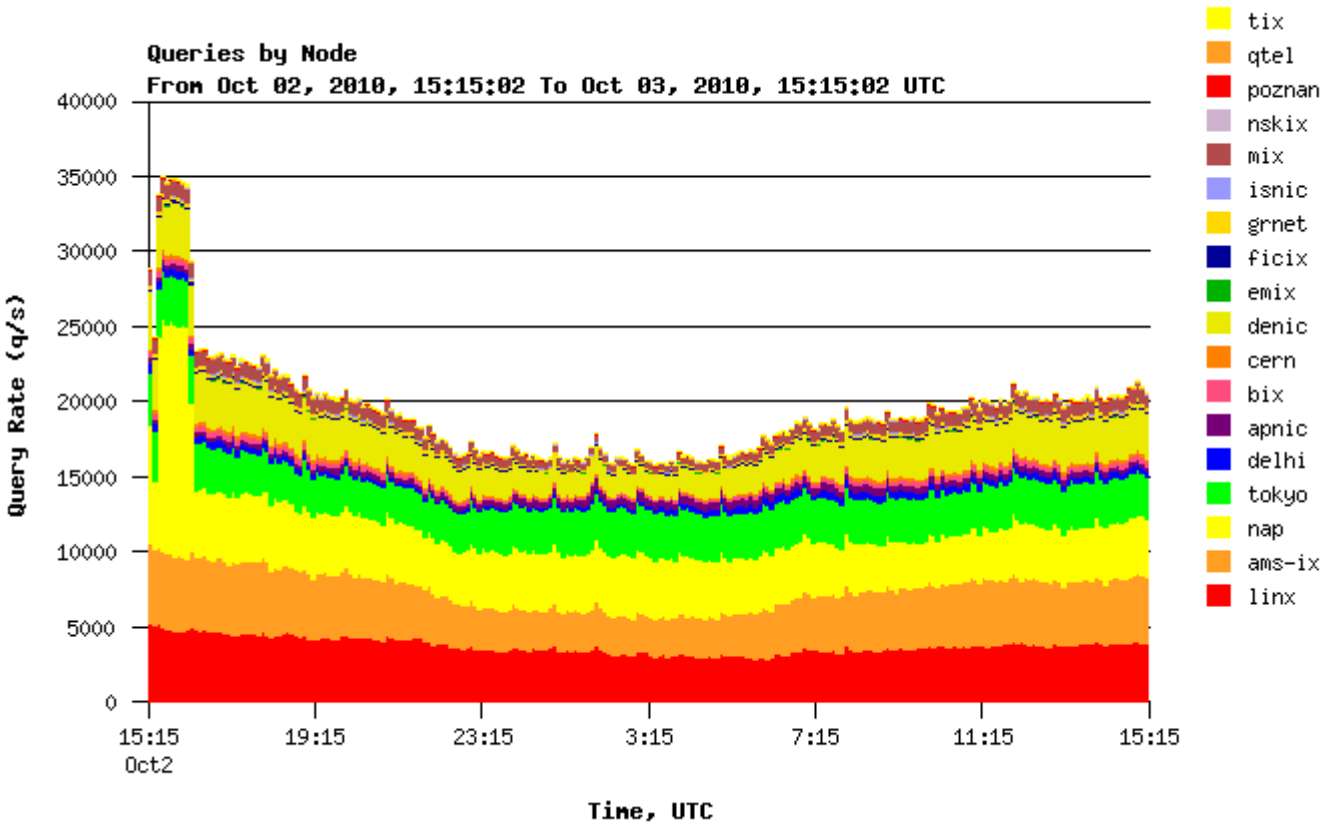
Example (cnt'd)

ftp.cs.vu.nl.	CNAME	soling.cs.vu.nl.
www.cs.vu.nl.	CNAME	soling.cs.vu.nl.
soling.cs.vu.nl.	A	130.37.20.20
soling.cs.vu.nl.	MX	1 soling.cs.vu.nl.
soling.cs.vu.nl.	MX	666 zephyr.cs.vu.nl.
soling.cs.vu.nl.	HINFO	"Sun" "Unix"
vucs-das1.cs.vu.nl.	PTR	0.198.37.130.in-addr.arpa.
vucs-das1.cs.vu.nl.	A	130.37.198.0
inkt.cs.vu.nl.	HINFO	"OCE" "Proprietary"
inkt.cs.vu.nl.	A	192.168.4.3
pen.cs.vu.nl.	HINFO	"OCE" "Proprietary"
pen.cs.vu.nl.	A	192.168.4.2
localhost.cs.vu.nl.	A	127.0.0.1

Root servers (19-10-11) and replications



k-servers traffic



See also http://k.root-servers.org/

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 (request goes out b response is blocked)
  - 'stub' servers call a root server directly
- Servers are fast (see figure)
- Many servers are replicated
  - e.g. F has 46 sites (in 2009)

DNS Root Server Speed

The UltraTools DNS root server speed tool tests the speed across each of the DNS Root Servers from the test servers.

Name Server	AAAA	CNAME	MX	SOA	A	SPF	SRV	TXT
a.root-servers.net.	86	86	87	87	87	86	86	87
m.root-servers.net.	180	182	180	181	194	183	180	185
l.root-servers.net.	100	91	100	100	105	91	97	90
h.root-servers.net.	4	5	5	5	5	5	5	5
k.root-servers.net.	189	191	191	183	185	191	186	184
e.root-servers.net.	79	-	-	87	81	88	93	79
g.root-servers.net.	308	307	179	194	193	307	182	188
c.root-servers.net.	2	3	2	3	4	3	4	5
i.root-servers.net.	79	77	75	77	76	76	75	77
f.root-servers.net.	82	82	80	83	-	79	81	80
d.root-servers.net.	3	3	3	3	3	3	3	3
j.root-servers.net.	113	107	115	109	109	109	109	108
b.root-servers.net.	70	72	72	69	67	72	72	73
Min. Time(ms)	**2**	**3**	**2**	**3**	**3**	**3**	**3**	**3**
Max. Time(ms)	**308**	**307**	**191**	**194**	**194**	**307**	**186**	**188**
Avg. Time(ms)	**99**	**100**	**90**	**90**	**92**	**99**	**90**	**89**

from ultratools.com in October 2011

Scaling

- Response time depends on depth of search
- Global servers get lots of requests
  - need to limit traffic per 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 frequently
    - e.g. 24h-48h for TLD
    - highly mobile devices – cross domain - would not benefit from caching

Packet format

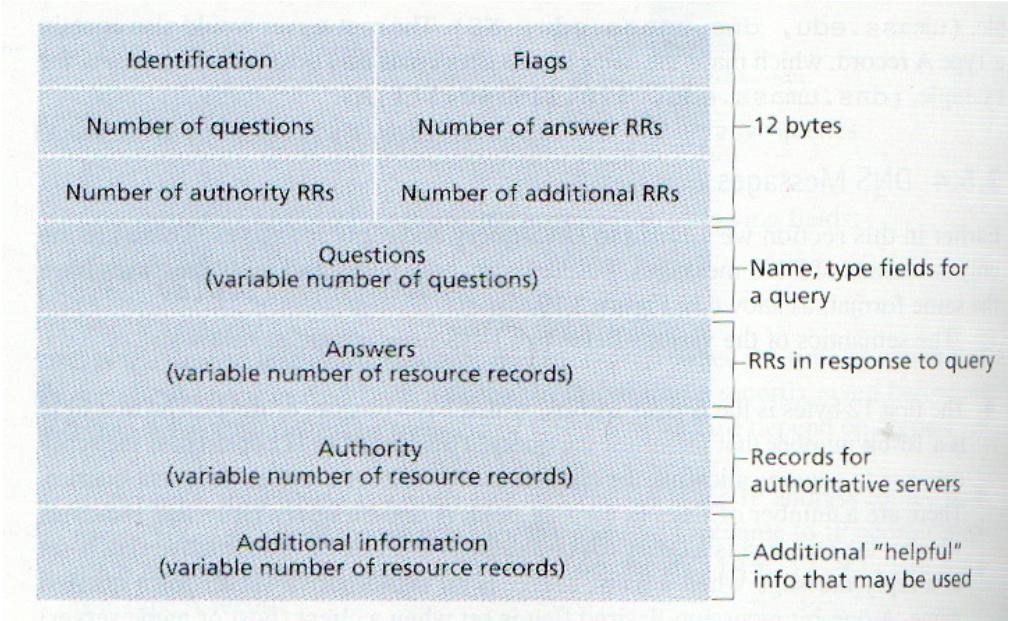
UDP carrier

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)
- response code (4)

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 set of *records* consisting of a list of (attribute, value) pairs is called a *directory*
  - service is termed: *directory service*, in contrast to a *naming service*
    - OSI X.500 directory service
    - LDAP, Lightweight Directory Access Protocol
      - MS Active Directory

Agenda

- Service discovery
- On naming
- Distributed flat resolution
  - DHT (distributed hashtable) of Chord
- Distributed structured resolution
  - File systems
  - DNS
- mDNS + DNS-SD

mDNS

- A service that provides DNS-like queries for DNS-like resource records using DNS-like query-response messages over UDP.
- The service is link local, i.e., provided on a single broadcast domain of a local area network.
- Gives users authority to equip their computers with a domain name of the form “single-dns-label.local”

  -- provided this does not give rise to local name clashes --

For detailed information see: Internet Engineering Task Force, RFC 6762

mDNS (ctnd)

Namespaces

- A zero configuration protocol can be used to assign IP-addresses
- There is a reserved top-level domain `local.` that can be used to give hosts on the local link a domain name
  - possibly in addition to another domain name
  - Similar to assigning hosts IP4-addresses in the range 169.254/16

Hosts

- form a communication group
  - on a local link, i,e., a confined segment of the local network,
  - identified by the multicast (group) address 224.0.0.251
- run so-called responders that are both clients and servers
  - as a server, a responder listens on the fixed port: 5353
  - as a client a responder issues DNS-queries to the group
- can be perceived as a (local) P2P system

mDNS (ctnd)

- Queries
  - one-shot queries
  - terminate upon the first response
  - must not be issued from port 5353
- Continuous queries
  - use retransmission until a sufficient number of answers (this depends on the type of operation) has been received
  - indicate known answers in the query, to reduce response traffic
- Database
  - together, the responders maintain a distributed cache that contains the DNS-like database.
  - eventual consistent for shared resources (RRs)
    - when there exist an authoritative host
  - TTL used to determine staleness of resource records
    - records whose TTL has become zero are deleted

mDNS

- Database content
  - Added directly by system adminstrators
  - Added by zero configuration
  - Added by users through via client APIs
    - `dns-sd -R …`
    - `avahi-publish …`

Note. Explicit removal of mDNS database content is not possible.

Avahi, e.g., does not offer the possibility to send good-bye packets. Only expiration of the (RR-)TTL will cause removal, but this value is large. On the other hand, killing the avahi process that sends the periodic advertisement will result in a goodbye packet being send. This, however, does not cater for other ways in which the service may become unreachable, such as network outages, etc..

DNS-SD

- Specifies how DNS resource records are named and structured to facilitate service discovery
- Uses the SRV, PTR and TXT resource records from DNS to create a service repository
- Works both with unicast and multicast
- Provides zero configuration when used in combination with mDNS for service discovery on a local network
  - DHCP server provides a domain that can be queried by the new client for services

.

DNS-SD records

- SRV records are used to specify the service instances
  - Using extension of the standard DNS conventions
    - Service Instance Name = <Instance> . <Service> . <Domain>
- TXT records define attributes of the service
  - There must be at least one TXT record per SRV record
    - usually precisely one
  - Using the key=value format
- PTR records are used to distinguish between instances of a service
  - e.g., when the location where the service is provided matters,
    - print services, display services, sensors, actuators, …
  - This extra level of indirection is special to DNS-SD

DNS-SD Service instance enumeration

By querying PTR records instead of SRV records

- the query states the generic service, viz., service type (`_http`) and transport protocol (`_tcp`) , and the domain (`dns-sd.org`)
- the response lists all instances found in PTR records, which together serve as a directory of instances for the generic service

What web pages are being advertised from dns-sd.org?

```
nslookup -q=ptr _http._tcp.dns-sd.org.
_http._tcp.dns-sd.org
      name = Zeroconf._http._tcp.dns-sd.org
_http._tcp.dns-sd.org
      name = Multicast\032DNS._http._tcp.dns-sd.org
_http._tcp.dns-sd.org
      name = Service\032Discovery._http._tcp.dns-sd.org
_http._tcp.dns-sd.org
      name = Stuart's\032Printer._http._tcp.dns-sd.org
```

<Instance> <Service> <Domain>

DNS-SD

- Service type enumeration
  - a special meta query that has as its response the list of services to be found in the database
    - -q=ptr "`_services._dns-sd._udp`.<Domain>"
    - Domain can, e.g., be `local.`
    - Browser clients can use this feature to display a list of services
      - e.g. `ahavi-browse` …

For detailed information see:

Internet Engineering Task Force (IETF), RFC 6763