

CS 5523 Lecture 26: Naming

- *Questions on Laboratory 4*
- *Basic terminology*
- *Name spaces*
- *Name service goals*
- *Navigation*
- *DNS – name resolution and representation*
- *Practical issues in name resolution*

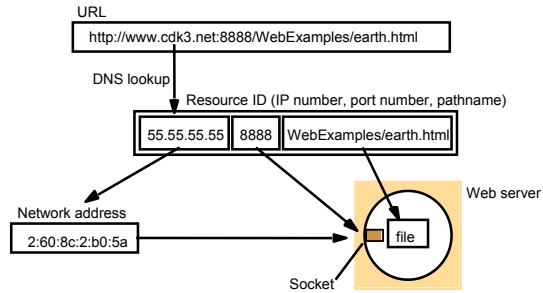
Basic naming terminology

- *name - something that identifies an entity*
- *pure name - uninterpreted bit pattern*
- *non-pure name - contains information about the entity*
- *address - location of an entity*
- *resolving a name - when a name is translated into information about the entity in order to invoke an action on it*
- *binding - association between name and information about entity*

Examples of binding

- *DNS - maps domain names to host attributes (IP address, type of service, time-to-live)*
- *X500 - a directory service that maps a person's name to attributes (email address, etc.)*
- *CORBA Naming and Trading Service - maps the name of a remote to reference and object description*

Figure 9.1
Composed naming domains used to access a resource from a URL



Instructor's Guide for Conrad, Doherty and Kuehling: Distributed Systems: Concepts and Design, Edn. 3
© Addison-Wesley Publishers 2009

Name service basics

- a name service stores one or more naming contexts and responds to requests regarding these contexts
- a naming context is a set of textual names and attributes
- a name space is a collection of valid names recognized by a name service
- a naming domain is a name space which has a single administrative authority

What operations should a name service have?

Why is name management usually separated from other services?

Name spaces in DNS

- hierarchical structure - one or more components or labels separated by periods (.)
- only absolute names - referred relative to global root
- clients usually have a list of default domains that are appended to single-component domain names before trying global root
- allows aliases such as `www.utsa.edu`

Compare name spaces in DNS with Unix file names.

Name service goals:

- *scalable to arbitrary size*
- *have a long lifetime*
- *be highly available*
- *have fault isolation*
- *tolerate mistrust*

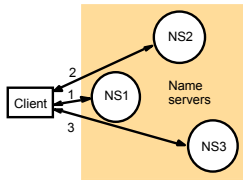
Navigation of name services:

- *name spaces are usually partitioned and cannot be serviced by a single server*
- *navigation - process of locating naming data from a collection of name servers in order to resolve a name*

Types of navigation:

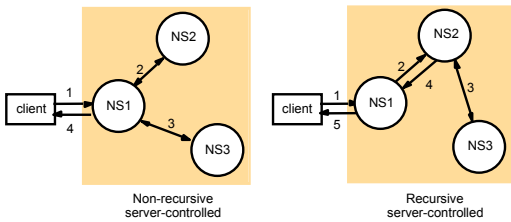
- *iterative - result of query immediately returned to client (with a hint of where to look next if query failed)*
- *multicast - client simultaneously queries a group of servers and waits for the first response*
- *non-recursive server-controlled - client chooses a server who provides iterative navigation on its behalf*
- *recursive server-controlled - servers recursively contact other servers until name is resolved.*

Figure 9.2
Iterative navigation



A client iteratively contacts name servers NS1–NS3 in order to resolve a name

Figure 9.3
Non-recursive and recursive server-controlled navigation



A name server NS1 communicates with other name servers on behalf of a client

DNS - domain name system:

- replaced a single master file scheme
- devised by Mockapetris (1987)
- objects are names of computers
- attributes are IP addresses
- scaling is achieved by a combination of
 - hierarchical partitioning
 - replication of naming data
 - caching

Domain names (last element of hierarchical name):

- *com* - commercial organizations
- *edu* - universities and educational institutions
- *gov* - US government agencies
- *mil* - US military organizations
- *net* - major network support centers
- *org* - organizations not included in first five
- *int* - international organization
- country codes - (e.g., us, uk, fr, etc.)

New domain names (approved by ICANN 11/2000):

- *aero* - air transportation industries
- *biz* - businesses
- *coop* - cooperatives
- *info* - unrestricted
- *museum* - museums
- *name* - individuals
- *pro* - professions such as doctors and lawyers

DNS queries:

- *host resolution* - resolves host names into IP addresses
- *mail host location* - resolves domain names into IP addresses of mail hosts
- *reverse resolution* - returns a host name given an IP address
- *host information* - given a host name, return information about the host
- *well-known services* - return a list of services given a hostname

Give an example of a use of each of these types of queries.

Zone partitioning of the DNS name space:

- zone - contains attribute data for names in domain minus the sub-domains administrated by lower-level authorities:
Example: UTSA has a name server for utsa.edu, but cs.utsa.edu names are resolved by the CS Department server
- at least two name servers that provide authoritative data for the zone
- names of the servers for the sub-domains
- zone management parameters

Authoritative name servers:

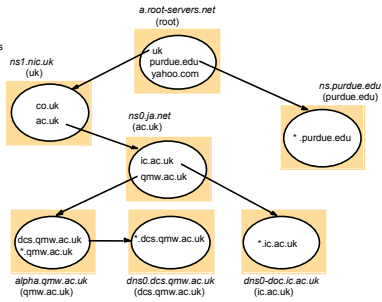
- a server may be an authoritative source for zero or more zones
- data for a zone is entered into a local master file
- the master (primary) server reads the zone data directly from the master file
- secondary authoritative servers download zone data from primary server
- secondary servers periodically check their version number against the master server

Caching in DNS:

- any server can cache any name
- non-authoritative servers note time to live when they cache data
- non-authoritative servers indicate that they are such when responding to clients with cached names

Figure 9.4
DNS name servers

Note: Name server names are in italics, and the corresponding domains are in parentheses. Arrows denote name server entries.



Instructor's Guide for Cindoria, Deffense and Kishberg. Distributed Systems: Concepts and Design. Edn. 3. © Addison-Wesley Publishers 2000.

DNS clients (resolvers):

- Resolvers are usually implemented as library routines (e.g., *gethostbyname*).
- The request is formatted into a DNS record.
- DNS servers use a well-known port.
- A request-reply protocol is used.
- The resolver times out and resends if it doesn't receive a response in a specified time.

Figure 9.5
DNS resource records

Record type	Meaning	Main contents
A	A computer address	IP number
NS	An authoritative name server	Domain name for server
CNAME	The canonical name for an alias	Domain name for alias
SOA	Marks the start of data for a zone	Parameters governing the zone
WKS	A well-known service description	List of service names and protocols
PTR	Domain name pointer (reverse lookups)	Domain name
HINFO	Host information	Machine architecture and operating system
MX	Mail exchange	List of <preference, host> pairs
TXT	Text string	Arbitrary text

Instructor's Guide for Cindoria, Deffense and Kishberg. Distributed Systems: Concepts and Design. Edn. 3. © Addison-Wesley Publishers 2000.

Layout of a DNS database:

- SOA - giving zone parameters
- NS records - giving name servers for domain
- MX records - giving mail host names

Practical issues in name resolution:

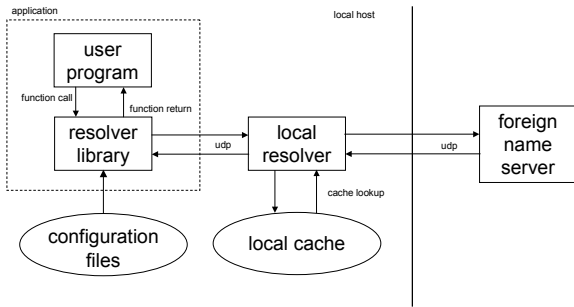
- Name resolution is needed in most communication because users usually specify hosts by name rather than IP address.
- `/etc/resolv.conf` specifies IP addresses of the name servers on most Unix systems.
- A single name may require many inquiries to be resolved.
- Local shared caching can considerably reduce network traffic.
- The name servers must be relatively reliable, or the network won't work.

DNS queries:

- host resolution - resolves host names into IP addresses
- mail host location - resolves domain names into IP addresses of mail hosts
- reverse resolution - returns a host name given an IP address
- host information - given a host name, return information about the host
- well-known services - return a list of services given a hostname

Give an example of a use of each of these types of queries.

Typical name resolution in a user program:



The `/etc/resolv.conf` file for medusa

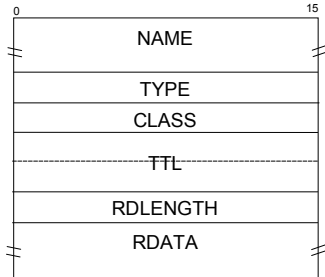
```
domain cs.utsa.edu
search cs.utsa.edu atm.utsa.edu utsa.edu
nameserver 129.115.11.19
nameserver 129.115.102.150
nameserver 128.83.139.9
```

- *domain* – indicates the local default domain name for resolving short names
- *search* – list to search for hostname lookup
- *nameserver* – foreign name servers to be queried in listed order

DNS databases:

- A domain name identifies a node.
- Each information at each node consists of resource records.
- Each resource record (RR) has:
 - owner (domain name)
 - type (A, CNAME, HINFO, MX, NS, PTR, SOA)
 - class (IN = internet)
 - TTL
 - RDATA (describes the information above)

DNS resource records (RR):



DNS name format:

- A domain name is a sequence of labels.
- A label is 1 byte length followed by that number of bytes.
- A domain name ends with a null byte (a zero-length label).
- The labels are printed separated by dots.
- Most DNS comparisons are case insensitive

Example of a DNS name:

2 c s 4 u t s a 3 e d u 0

Sizes defined in `arpa/nameser.h`:

```
/*
 * Define constants based on RFC 883, RFC 1034, RFC 1035
 */
#define NS_PACKETSZ 512 /* maximum packet size */
#define NS_MAXDNAME 1025 /* maximum domain name */
#define NS_MAXCDNAME 255 /* maximum compressed domain name */
#define NS_MAXLABEL 63 /* maximum length of domain label */
#define NS_HFIXEDSZ 12 /* #bytes of fixed data in header */
#define NS_QFIXEDSZ 4 /* #bytes of fixed data in query */
#define NS_RFIXEDSZ 10 /* #bytes of fixed data in r record */
#define NS_INT32SZ 4 /* #bytes of data in a u_int32_t */
#define NS_INT16SZ 2 /* #bytes of data in a u_int16_t */
#define NS_INT8SZ 1 /* #bytes of data in a u_int8_t */
#define NS_INADDRSZ 4 /* IPv4 T.A */
#define NS_IN6ADDRSZ 16 /* IPv6 T.AAAA */
#define NS_CMPRSFLGS 0xc0 /* Flag bits indicating name compression. */
#define NS_DEFAULTPORT 53 /* For both TCP and UDP. */
```

Where would you look for the file `arpa/nameser.h`?

How would you tell the compiler to look somewhere else?

Types defined in arpa/nameser.h:

```
typedef enum ns_type {
    ns_t_a = 1, /* Host address */
    ns_t_ns = 2, /* Authoritative server */
    ns_t_md = 3, /* Mail destination */
    ns_t_mf = 4, /* Mail forwarded */
    ns_t_cname = 5, /* Canonical name */
    ns_t_soa = 6, /* Start of authority zone */
    ns_t_mb = 7, /* Mailbox domain name */
    ns_t_mg = 8, /* Mail group member */
    ns_t_mr = 9, /* Mail rename name */
    ns_t_mx = 10, /* Mail resource record */
    ns_t_wks = 11, /* Well known service */
    ns_t_ptr = 12, /* Domain name pointer */
    ns_t_hinfo = 13, /* Host information */
    ns_t_minfo = 14, /* Mailbox information */
    ns_t_txt = 16, /* Text strings */
    ns_t_rp = 17, /* Responsible person */
    ns_t_afsdb = 18, /* AFS cell database */
    ns_t_x25 = 19, /* X.25 calling address */
    ns_t_isdn = 20, /* ISDN calling address */
    ns_t_rt = 21, /* Router */
    ns_t_nsap = 22, /* NSAP address */
    ns_t_ptr = 21, /* Router */
    ns_t_nsap = 22, /* NSAP address */
    ns_t_nsap_ptr = 23, /* Reverse NSAP lookup */
    ns_t_sig = 24, /* Security signature */
    ns_t_key = 25, /* Security key */
    ns_t_px = 26, /* X.400 mail mapping */
    ns_t_gpos = 27, /* Geographical position */
    ns_t_soa = 28, /* IP6 Address */
    ns_t_loc = 29, /* Location information */
    ns_t_nxt = 30, /* Next domain (security) */
    ns_t_eid = 31, /* Endpoint identifier */
    ns_t_ninloc = 32, /* Nimrod Locator */
    ns_t_srv = 33, /* Server Selection */
    ns_t_atma = 34, /* ATM Address */
    ns_t_naptr = 35, /* Naming Authority Pointer */
    /* Query type values not in resource records */
    ns_t_ixfr = 251, /* Incremental zone transfer */
    ns_t_axfr = 252, /* Transfer zone of authority */
    ns_t_mailb = 253, /* Transfer mailbox records */
    ns_t_maila = 254, /* Transfer mail agent records */
    ns_t_any = 255, /* Wildcard match */
    ns_t_max = 65536
} ns_type;
```

Class field definitions in arpa/nameser.h:

```
/*
 * Values for class field
 */
typedef enum ns_class {
    ns_c_in = 1, /* Internet. */
    ns_c_chaos = 3, /* Class 2 unallocated/unsupported. */
    ns_c_hs = 4, /* MIT Chaos-net. */
    /* Query class values which do not appear in resource records */
    ns_c_none = 254, /* For prereq. sections in update requests */
    ns_c_any = 255, /* Wildcard match. */
    ns_c_max = 65536
} ns_class;
```

Zone partitioning of the DNS name space:

■ zone - contains attribute data for names in domain minus the sub-domains administrated by lower-level authorities:

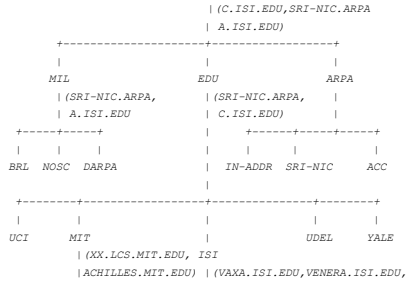
Example: UTSA has a name server for utsa.edu, but cs.utsa.edu names are resolved by the CS Division server

■ at least two name servers that provide authoritative data for the zone

■ names of the servers for the sub-domains

■ zone management parameters

Zone partitioning of the DNS name space:



from RCF 1034: Domain Concepts and Facilities by Mockpetris

Authoritative servers:

- a server may be an authoritative source for zero or more zones
- data for a zone is entered into a local master file
- the master (primary) server reads the zone data directly from the master file
- secondary authoritative servers download zone data from primary server
- secondary servers periodically check their version number against the master server

Sample query of the root server :

```

.      IN      SOA      SRI-NIC.ARPA. HOSTMASTER.SRI-NIC.ARPA. (
                          $T0611
                          1800          ;refresh every 30 min
                          300           ;retry every 5 min
                          604800        ;expire after a week
                          86400)       ;minimum of a day
      NS      A.ISI.EDU
      NS      C.ISI.EDU.
      NS      SRI-NIC.ARPA.
MIL.  86400  NS      SRI-NIC.ARPA.
      86400  NS      A.ISI.EDU.
EDU.  86400  NS      SRI-NIC.ARPA.
      86400  NS      C.ISI.EDU.
      ...Continued from previous column
SRI-NIC.ARPA. A      26.0.0.73      USC-ISIG.ARPA. CNAME C.ISI.EDU.
      A      73.0.0.26.IN-ADDR.ARPA. PTR SRI-NIC.ARPA.
      A      65.0.6.26.IN-ADDR.ARPA. PTR ACC.ARPA.
      MX     0 SRI-NIC.ARPA. 51.0.0.10.IN-ADDR.ARPA. PTR SRI-NIC.ARPA.
      HINFO  DEC-2060 TOP320 52.0.0.10.IN-ADDR.ARPA. PTR C.ISI.EDU.
ACC.ARPA. A      26.6.0.65      103.0.3.26.IN-ADDR.ARPA. PTR A.ISI.EDU.
      HINFO  PDP-11/70 UNIX  A.ISI.EDU. 86400 A 26.3.0.103
      MX     10 ACC.ARPA. C.ISI.EDU. 86400 A 10.0.0.52
      ... Continued in next column
  
```

Sample query of the root server :

```
-----+-----
Header | OPCODE=QUERY, |
      | QNAME=SRI-NIC.ARPA., QCLASS=IN, QTYPE=A |
Question | <empty> |
Answer | <empty> |
Authority | <empty> |
Additional | <empty> |
-----+-----
```

query to c.isi.edu

response

```
-----+-----
Header | OPCODE=QUERY, RESPONSE, AA |
      | QNAME=SRI-NIC.ARPA., QCLASS=IN, QTYPE=A |
Question | <empty> |
Answer | SRI-NIC.ARPA. 86400 IN A 26.0.0.73 |
      | 86400 IN A 10.0.0.51 |
Authority | <empty> |
Additional | <empty> |
-----+-----
```

Sample query of the root server :

query to c.isi.edu: QNAME=SRI-NIC.ARPA, QTYPE=*

response:

```
-----+-----
Header | OPCODE=QUERY, RESPONSE, AA |
      | QNAME=SRI-NIC.ARPA., QCLASS=IN, QTYPE=* |
Question | <empty> |
Answer | SRI-NIC.ARPA. 86400 IN A 26.0.0.73 |
      | A 10.0.0.51 |
      | MX 0 SRI-NIC.ARPA. |
      | HINFO DEC-2060 TOPS20 |
Authority | <empty> |
Additional | <empty> |
-----+-----
```

*The * wildcard for the type indicates all records.*

Sample query of the root server :

query to c.isi.edu: QNAME=BRL.MIL, QTYPE=A

response:

```
-----+-----
Header | OPCODE=QUERY, RESPONSE |
      | QNAME=BRL.MIL, QCLASS=IN, QTYPE=A |
Question | <empty> |
Answer | <empty> |
Authority | MIL. 86400 IN NS SRI-NIC.ARPA. |
      | 86400 NS A.ISI.EDU. |
Additional | A.ISI.EDU. A 26.3.0.103 |
      | SRI-NIC.ARPA. A 26.0.0.73 |
      | A 10.0.0.51 |
-----+-----
```

response contains the address RRs because cs.is.edu isn't authoritative for mil and it didn't have BRL.MIL.

Using nstest to query name servers (example 1):

```
visual8% nstest 129.115.102.150
amedusa.cs.utsa.edu
;; res_mkquery(0, medusa.cs.utsa.edu, 1, 1)
;; res_send()
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 46777
;; flags: rd; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 0
;;      medusa.cs.utsa.edu, type = A, class = IN
;; Querying server (# 1) address = 129.115.102.150
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 46777
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0,
ADDITIONAL: 0;; medusa.cs.utsa.edu, type = A, class = IN
medusa.cs.utsa.edu.      1D IN A      129.115.11.62
```

Using nstest to query name servers (example 2):

```
nmedusa.cs.utsa.edu
;; res_mkquery(0, medusa.cs.utsa.edu, 1, 2)
;; res_send()
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 46778
;; flags: rd; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 0
;;      medusa.cs.utsa.edu, type = NS, class = IN
;; Querying server (# 1) address = 129.115.102.150
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 46778
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 0
;;      medusa.cs.utsa.edu, type = NS, class = IN
cs.utsa.edu.      1D IN SOA      jazz.cs.utsa.edu. root.jazz.cs.utsa.edu. (
424              ; serial
3H               ; refresh
1H               ; retry
1W               ; expiry
1D )             ; minimum
```

The resolver library:

```
cc [ flag ... ] file ... -lresolv -lsocket -lnsl [ library ... ]

#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/nameser.h>
#include <resolv.h>

int res_mkquery(int op, /* QUERY */
               const char *dname, /* domain */
               int class, /* IN */
               int type, /* A, NS, etc. */
               const char *data, /* RR */
               int datalen, /* length of RR */
               struct rrec *newrr, /* use null */
               uchar_t *buf, /* answer */
               int buflen /* answer length */);
```

The resolver library:

```
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/nameser.h>
#include <resolv.h>

int res_send(uchar_t *msg, /* preformatted msg */
             int msglen,
             uchar_t *answer,
             int anslen);
```

The resolver library:

```
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/nameser.h>
#include <resolv.h>

Int res_query(const char *dname,
              int class,
              int type,
              uchar_t *answer,
              int anslen);
```

For next time:

- Read CDK Chapter 9.3-9.5
- Try some examples of nstest before the next class
