DNS: Domain Name System

Domain Name System:
- Map between symbolic domain name and IP address
- *Distributed database*: implemented in hierarchy of many name servers
- *Application-layer protocol*: host, routers, name servers communicate to *resolve* names (address/name translation)
  - Core Internet function implemented as application-layer protocol
DNS name servers

No server has all name-to-IP address mappings

Local name servers (Resolvers):
- Each ISP, company has local (default) name server
- Query first goes to local name server

Authoritative name server:
- Authority for a zone (= domain)
- Can, e.g., perform name/address translation for a host’s name
Distributed, hierarchical database

Example: Client wants IP address for www.ietf.org
- Client queries a root server, response contains .org DNS servers
- Client queries .org DNS server, response contains ietf.org DNS servers
- Client queries ietf.org DNS server, response contains IP address for www.ietf.org
Server types, zones and domains

- **Authoritative DNS servers:**
  - Responsible for a zone
  - Provide authoritative answers, e.g.,
    - Root servers: On top of DNS hierarchy. Know which servers are responsible for a particular top-level domain
    - Top-level domain (TLD) servers: responsible for com, org, net, edu, ..., and all country code top-level domains (ccTLD) de, uk, fr, ca, jp, ...
    - Organizations’ name servers

- **Resolving DNS servers (aka Resolver, “cache”)**
  - Perform domain name resolution on behalf of a client’s stub resolver
  - Very often cache answers
Recursive queries

Recursive query:
- puts burden of name resolution on contacted name server
- To local resolver

Iterative query:
- contacted server replies with name of server to contact
- “I don’t know this name, but ask this server”

Host at cis.poly.edu wants IP address for gaia.cs.umass.edu

Resolver dns.poly.edu

Requesting host cis.poly.edu

SLD DNS server dns.cs.umass.edu

TLD DNS server

root DNS server
Iterative queries

Recursive query:
- Puts burden of name resolution on contacted name server
- To local resolver

Iterative query:
- Contacted server replies with name of server to contact
- “I don’t know this name, but ask this server”

Host at cis.poly.edu wants IP address for gaia.cs.umass.edu
Replication

- Common case: several authoritative servers per zone
  - One primary/master server
  - Many secondaries/backups/slaves/...
  - Slave servers synchronize with master after timeout and notifies

- Caching on resolvers: once a resolving name server learns a mapping, it *caches* mapping
  - Cache entries timeout (disappear) after some time
Inside the DNS Protocol

- Uses UDP Port 53  
  (TCP: only for server-to-server traffic or large volumes, and as a fallback)
- Limited Packet Size (about 500 Bytes, can become larger through extensions)
- Same packet/message format for both queries and responses
- Association of queries with responses by identification field: „query id“
Inside the DNS protocol: DNS packet

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>identification</td>
<td>ID of the request</td>
</tr>
<tr>
<td>flags</td>
<td>Flags: rec, auth, add, op</td>
</tr>
<tr>
<td>number of questions</td>
<td>Number of questions</td>
</tr>
<tr>
<td>number of answer RR’s</td>
<td>Number of answer records</td>
</tr>
<tr>
<td>number of authority RR’s</td>
<td>Number of authority records</td>
</tr>
<tr>
<td>number of additional RR’s</td>
<td>Additional records</td>
</tr>
</tbody>
</table>

- questions
- answers (variable numbers of resource records)
- authority (variable number of resource records)
- additional (variable number of resource records)
Inside the DNS protocol: DNS records

distributed db storing resource records (RR)

RR format: \((name, type, class, ttl, length, data)\)

- For all practical purposes: Class=IN (Internet)

- Type=A
  - name is hostname
  - data is IP address

- Type=NS
  - name is domain (e.g., foo.com)
  - data is name of authoritative DNS server for this domain

- Type=CNAME
  - for alias

- Type=MX
  - for mail
Perl Continued

- pack()/unpack()
- UDP Socket Programming()
**pack()**

- $data = \text{pack($template, @list)}$

- pack() takes a list of scalars (@list) and packs them into a binary structure (e.g., a bitfield) according to template.

- template specifies how wide the elements of the bitfields are, and how to interpret the results

- unpack() is the reverse operation
pack() -- Examples

$out = \text{pack "cccc", 65, 66, 67, 68}; \quad \# \text{"ABCD"}$
$out = \text{pack "c4", 65, 66, 67, 68}; \quad \# \text{same thing}$
$out = \text{pack ("B8ccc", "01000001", 66, 67, 68); \# \text{same thing}$

$(01000001_2 == 65_{10})$

# a 8-bit field with flags, followed by a 16 bit length field in
# network byte order
$flags="10011001"; \quad \# \text{a string}$
$len = 25; \quad \# \text{an integer, not a string}$
$out = \text{pack("B8n", $flags, $len);}$
$out .= \text{pack ...... \# add some other stuff}$
pack()

Some frequently used template characters:

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>Bit string, ascending bit order inside each byte</td>
</tr>
<tr>
<td>B</td>
<td>Bit string, ascending bit order inside each byte</td>
</tr>
<tr>
<td>C</td>
<td>Unsigned character / 8 bit</td>
</tr>
<tr>
<td>n</td>
<td>Short (16 bit) in network byte order</td>
</tr>
<tr>
<td>N</td>
<td>Long (32 bit) in network byte order</td>
</tr>
<tr>
<td>S</td>
<td>Unsigned short in host byte order</td>
</tr>
<tr>
<td>I</td>
<td>Unsigned integer in host byte order</td>
</tr>
</tbody>
</table>

Complete list: perldoc –f pack or try perldoc perlpacktut
IO::Socket::INET – UDP Client

use IO::Socket::INET;

$client = IO::Socket::INET->new(PeerAddr => "dns.hier.de",
    PeerPort => 53,
    Type => SOCK_DGRAM,
    Proto => "udp");

$client->send($dnspacket);
$answer_packet = $client->recv();
$client->close();
Further Reading

- DNS:
  - Kurose & Ross: Computer Networking, 4th ed. (preliminary version of 1st ed online at: http://www.net.t-labs.tu-berlin.de/teaching/computer_networking/)
  - RFC 1034 and RFC 1035

- Perl / pack / unpack / socket programming
  - perldoc IO::Handle
  - perldoc IO::Socket
  - perldoc IO::Socket::Inet
  - perldoc -f pack
  - perldoc -f unpack
  - perldoc perlpacktut