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)*
- For a host: stores that host’s IP address, name
- Can perform name/address translation for that host’s name
Distributed, hierarchical database

Client wants IP for www.amazon.com; 1st approx:
- Client queries a root server to find .com DNS server
- Client queries .com DNS server to get amazon.com DNS server
- Client queries amazon.com DNS server to get IP address for www.amazon.com
TLD and Authoritative Servers

- **Root servers**: On top of hierarchy. Know which servers are responsible for a particular Top-level domain.

- **Top-level domain (TLD) servers**: responsible for com, org, net, edu, etc, and all top-level country domains uk, fr, ca, jp.

- **Authoritative DNS servers**: organization’s DNS servers, providing authoritative hostname to IP mappings for organization’s servers (e.g., Web and mail).

- **Local DNS servers**
  - Do not strictly belong to hierarchy
  - When a host makes a DNS query, query is sent to its local DNS server
    - Acts as a proxy, forwards query into hierarchy.
Recursive queries

recursive query:
- puts burden of name resolution on contacted name server
- heavy load?

iterated 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
Iterative queries

Recursive query:
- Puts burden of name resolution on contacted name server
- Heavy load?

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
DNS: caching and updating records

- Once (any) name server learns mapping, it *caches* mapping
  - Cache entries timeout (disappear) after some time
- Update/notify mechanisms under design by IETF
  - RFC 3007 (Feb. 2004)
Inside the DNS Protocol

- Uses UDP Port 53
  (TCP: only for server-to-server traffic or large volumes)
- Limited Packet Size (about 500 Bytes)
- Same packet/message format for both queries and responses
- Association of queries with responses by identification field
Inside the DNS protocol: DNS packet

<table>
<thead>
<tr>
<th></th>
<th>Identification</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>number of questions</td>
<td>number of answer RR's</td>
</tr>
<tr>
<td></td>
<td>number of authority RR's</td>
<td>number of additional RR's</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
  - \textit{name} is hostname
  - \textit{data} is IP address

- Type=CNAME
  - for alias

- Type=MX
  - for mail

- Type=NS
  - \textit{name} is domain (e.g., foo.com)
  - \textit{data} is IP address of authoritative name server for this domain
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 = pack "cccc", 65, 66, 67, 68;  # $out eq "ABCD"
$out = pack "c4", 65, 66, 67, 68;    # same thing
$out = pack ("B8ccc", "01000001, 66, 67,68) # 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";  # a string
$len = 25;    # an integer, not a string
$out = pack("B8n", $flags, $len);
$out .= 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);
$client->recv();
$client->close();
Further Reading

- DNS:
    (preliminary version of 1st ed online at: http://www.net.t-labs.tu-erlin.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