Wire?

Wire == Network

- Not the 1980s rock band ;-) 
- nor the TV series ;-) 

- Queries and Responses are packaged into packets 
- Packets are transferred over the wire/air 
  - OSI Layer 2: can be anything 
  - OSI Layer 3: IPv4 or IPv6 
  - OSI Layer 4: UDP or TCP

How to package DNS messages

1. Define what information you want to exchange
2. Specify a format in which to encode that information
   - Serialization, “flattening” data structures
3. Implement that format in software
4. Start doing DNS :)
DNS Message packet format

<table>
<thead>
<tr>
<th>Header section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question section</td>
</tr>
<tr>
<td>Answer section</td>
</tr>
<tr>
<td>Authority section</td>
</tr>
<tr>
<td>Additional section</td>
</tr>
</tbody>
</table>

DNS packet header

<table>
<thead>
<tr>
<th>0</th>
<th>15</th>
<th>16</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Flags</td>
<td></td>
<td></td>
</tr>
<tr>
<td>QDCOUNT</td>
<td>ANCOUNT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NSCOUNT</td>
<td>ARCOUNT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DNS header

DNS header fields

<table>
<thead>
<tr>
<th>ID</th>
<th>Transaction Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flags</td>
<td>See next slide</td>
</tr>
<tr>
<td>QDCOUNT(^1)</td>
<td>Number of questions</td>
</tr>
<tr>
<td>ANCOUNT</td>
<td>Number of answers</td>
</tr>
<tr>
<td>NSCOUNT</td>
<td>Number of authority records</td>
</tr>
<tr>
<td>ARCOUNT</td>
<td>Number of additional records</td>
</tr>
</tbody>
</table>

DNS header flags

<table>
<thead>
<tr>
<th>Bit(s)</th>
<th>Mnemonic</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>QR</td>
<td>Query(0) or Response(1)</td>
</tr>
<tr>
<td>1-4</td>
<td>OPCODE</td>
<td>Kind of Query (see next slide)</td>
</tr>
<tr>
<td>5</td>
<td>AA</td>
<td>Authoritative Answer</td>
</tr>
<tr>
<td>6</td>
<td>TC</td>
<td>Truncation or Truncated Response</td>
</tr>
<tr>
<td>7</td>
<td>RD</td>
<td>Recursion Desired</td>
</tr>
<tr>
<td>8</td>
<td>RA</td>
<td>Recursion Available</td>
</tr>
<tr>
<td>9</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>10</td>
<td>AD</td>
<td>Authentic Data (DNSSEC)</td>
</tr>
<tr>
<td>11</td>
<td>CD</td>
<td>Checking Disabled (DNSSEC)</td>
</tr>
<tr>
<td>12-15</td>
<td>RCODE</td>
<td>Result Code</td>
</tr>
</tbody>
</table>

\(^1\)It is unclear what the “D” stands for
DNS opcodes

<table>
<thead>
<tr>
<th>Value</th>
<th>Mnemonic</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Query</td>
<td>Standard query</td>
</tr>
<tr>
<td>1</td>
<td>IQuery</td>
<td>Inverse Query (obsolete)</td>
</tr>
<tr>
<td>2</td>
<td>Status</td>
<td>Status query (not standardized)</td>
</tr>
<tr>
<td>4</td>
<td>Notify</td>
<td>Change of master data</td>
</tr>
<tr>
<td>5</td>
<td>Update</td>
<td>Dynamic update</td>
</tr>
</tbody>
</table>

DNS result codes

<table>
<thead>
<tr>
<th>Value</th>
<th>Mnemonic</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NoError</td>
<td>No Error</td>
</tr>
<tr>
<td>1</td>
<td>FormErr</td>
<td>Format Error</td>
</tr>
<tr>
<td>2</td>
<td>ServFail</td>
<td>Server Failure</td>
</tr>
<tr>
<td>3</td>
<td>NXDomain</td>
<td>Non-eXistent Domain</td>
</tr>
<tr>
<td>4</td>
<td>NotImp</td>
<td>Not Implemented</td>
</tr>
<tr>
<td>5</td>
<td>Refused</td>
<td>Query Refused</td>
</tr>
<tr>
<td>6-10</td>
<td>…</td>
<td>Related to dynamic updates</td>
</tr>
<tr>
<td>11-15</td>
<td>…</td>
<td>Not assigned</td>
</tr>
<tr>
<td>16-…</td>
<td>…</td>
<td>Extended result codes (EDNS0)</td>
</tr>
</tbody>
</table>

Queries

- QDCOUNT should always be 1
  - Multiple queries have undefined behaviour
    - for instance there is only one result code
- Query consists of
  - QNAME (sequence of labels, encoded with length/value pairs)
    - Ending when length == 0, representing the empty root label
  - QTYPE (2 bytes)
  - QCLASS (2 bytes, almost always IN (=1))

Answers, Authorities and Additionals (1/2)

- Answers
  - Answers to question(s)
  - Special treatment of CNAMEs, DNAMEs and wildcards
- Authorities
  - Adds NS records as referral information
  - May add SOA and NS records for NXDOMAIN or NODATA
- Additional
  - Courtesy information
  - Dangerous... if accepted too easily, especially if the information is not related to the question
Each of these is a list of resource records
- Answer section
- Authority section
- Additional section

Data per resource record
- NAME, TYPE, CLASS (as in queries)
- TTL (4 bytes)
- RDLENGTH (2 bytes)
- RDATA (RDLENGTH bytes)

Use of zone transfers
- Copy data from master server to slave server
  - ns1.os3.nl. (master)
  - ns2.os3.nl. (slave)
  - ns1.zurich.surf.net. (slave)
- Zone transfers are often limited to slave servers
  - DNS data: public or semipublic?
  - You can prevent zone transfers using ACLs on the IP level

How zones are transferred
- Pull
  - When starting without a cached copy of the zone data
  - When data has changed
    - Serial number used to decide whether data has changed
  - DNS query type AXFR or IXFR
- Push
  - Tells slave servers to pull ;)
  - DNS opcode 4 ("notify")

Encoding of domain names
- Specify length of labels in label encoding
  - So a domain name is encoded as
    - `<length> <label> <length> <label> ...`
    - `3 "www" 3 "os3" 2 "nl" 0`, which amounts to
      - `3 'w' 'w' 'w' 3 'o' 's' '3' 2 'n' 'l' 0`
      - `3 119 119 119 3 111 115 51 2 114 108 0`, where
        - Lengths are at most 63 and encoded as big endian
        - `X` is the ASCII value of character X
  - Note the difference between 3 and `3'`
“Normal label length” encoding

- First byte used for length
  - First 2 bits are flags
    - 00 means “normal label length”
  - Remaining 6 specify label length
    - Hence the maximum label length of $2^6 - 1 = 63$ octets
- Remaining bytes contain the label itself
  - Number of remaining bytes is encoded in first byte of the label

---

Label types

- First two bits of the length byte denote the label type
  - A label length may not exceed 63 octets (6 bits needed)
  - 00: Normal label length
  - 11: Compressed label$^2$: 6+8 bits used as pointer
  - 01: Extended label type (EDNS0)
    - 01000001: Binary labels (for use with IPv6 PTR types)
    - Binary labels were deprecated in April 2013 (RFC 6891)
  - 10: Unallocated label type

---

$^2$This is a misnomer, because the domain name (not the label) is “compressed”
Compressed encoding

- Domain name with compressed encoding has fixed length of 2 bytes
  - First 2 bits are flags
    - 11 means "compressed label"
  - Remaining 6 bits + 8 subsequent bits are used as a pointer
    - Points to domain name at another position in the packet from where the domain name construction is continued
    - Value is offset from beginning of DNS packet which starts right after the UDP or TCP header
  - Saves space when a domain name is used more than once
    - For instance used to get 13 root name servers within 512 bytes

Compressed encoding

Wildcards (1)

From RFC 1034, section 4.3.3
The contents of the wildcard RRs follows the usual rules and formats for RRs. The wildcards in the zone have an owner name that controls the query names they will match. The owner name of the wildcard RRs is of the form "*.<anydomain>", where <anydomain> is any domain name. <anydomain> should not contain other * labels, and should be in the authoritative data of the zone. The wildcards potentially apply to descendants of <anydomain>, but not to <anydomain> itself. Another way to look at this is that the "*" label always matches at least one whole label and sometimes more, but always whole labels.
Wildcards (2)

From RFC 1034, section 4.3.3
Wildcard RRs do not apply:

- When the query is in another zone.
  That is, delegation cancels the wildcard defaults.
- When the query name or a name between the wildcard domain and the query name is known3 to exist. For example, if a wildcard RR has an owner name of ".*.X", and the zone also contains RRs attached to B.X, the wildcards would apply to queries for name Z.X (presuming there is no explicit information for Z.X), but not to B.X, A.B.X, or X.

---

3original text, should be “known”

Problems with wildcards in RFC 1034

- Notions are intuitive, not well-defined
  - When does a domain name “exist”?
  - How does matching work exactly?
  - What about empty non-terminals?
- RFC 4592 tries to clarify all of this
  - Defines “existence of a domain name”
  - Defines “asterisk label” and “wildcard domain name”
  - Defines “source of synthesis” and “closest encloser”

Wildcard synthesis in query matching algorithm

From RFC 1034, section 4.3.2, algorithm step 3.c
If at some label, a match is impossible (i.e., the corresponding label does not exist), look to see if a the "*" label exists. If the "*" label does not exist, check whether the name we are looking for is the original QNAME in the query or a name we have followed due to a CNAME. If the name is original, set an authoritative name error in the response and exit. Otherwise just exit.
If the "*" label does exist, match RRs at that node against QTYPE. If any match, copy them into the answer section, but set the owner of the RR to be QNAME, and not the node with the "*" label. Go to step 6.

Wildcard supporting definitions

Definitions

- A domain name exists if the name itself or any of its descendants has at least one RR
  - In particular empty non-terminals exist
- An asterisk label is a label of length 1 containing as only octet the ASCII equivalent of "*
- A wildcard domain name is a domain name with an asterisk label as its leftmost label
- The closest encloser of a query name is the longest matching ancestor that exists
- The source of synthesis of a query name is the domain name "*.<closest encloser>", which may or may not exist
$ORIGIN example.
example. 3600 IN SOA <SOA RDATA>
    example. 3600 NS ns.example.com.
    example. 3600 NS ns.example.net.
*.example. 3600 TXT "this is a wildcard"
    *.example. 3600 MX 10 host1.example.
    sub.*.example. 3600 TXT "... not a wildcard"
    host1.example. 3600 A 192.0.2.1
    _ssh._tcp.host1.example. 3600 SRV <SRV RDATA>
    _ssh._tcp.host2.example. 3600 SRV <SRV RDATA>
    subdel.example. 3600 NS ns.example.com.
    subdel.example. 3600 NS ns.example.net.

DNS limitations

▶ DNS is usually based on UDP
  ▶ RFC 1035 maximum size is 512 bytes of DNS content
  ▶ This limits the number of anycast IP addresses used
  ▶ Option to use TCP was present from the start
  ▶ but was not recommended for ordinary use
▶ DNS has weak security
  ▶ DNS packets can easily be spoofed
  ▶ Initially no support for message authentication except for a (clear text) Transaction ID

Extension Mechanisms for DNS (EDNS0)

▶ EDNS0 was first specified in RFC 2671, now replaced by RFC 6891
  ▶ Extends maximum size of UDP-based requests and responses
  ▶ Extends result codes, possible flags and label types
  ▶ Used by DNSSEC for DO (DNSSEC OK) extended flag
▶ Uses a "pseudo"-OPT-RR
  ▶ TYPE 41
  ▶ CLASS reused for UDP message size
  ▶ TTL reused for extended result codes and flags
  ▶ RDATA used for options as attribute-value pairs

RFC 4592 example queries

<table>
<thead>
<tr>
<th>QNAME</th>
<th>QTYPE</th>
<th>Synthesized?</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>host3.example.</td>
<td>MX</td>
<td>yes</td>
<td>RDATA</td>
</tr>
<tr>
<td>host3.example.</td>
<td>A</td>
<td>yes</td>
<td>NoData</td>
</tr>
<tr>
<td>foo.bar.example.</td>
<td>TXT</td>
<td>yes</td>
<td>RDATA</td>
</tr>
<tr>
<td>host1.example.</td>
<td>MX</td>
<td>no</td>
<td>NoData</td>
</tr>
<tr>
<td>sub.*.example.</td>
<td>MX</td>
<td>no</td>
<td>NoData</td>
</tr>
<tr>
<td>_telnet._tcp.host1.example.</td>
<td>SRV</td>
<td>no</td>
<td>NXDOMAIN</td>
</tr>
<tr>
<td>host.subdel.example.</td>
<td>A</td>
<td>no</td>
<td>referral</td>
</tr>
<tr>
<td>ghost.*.example.</td>
<td>MX</td>
<td>no</td>
<td>NXDOMAIN</td>
</tr>
</tbody>
</table>
Message Authentication

- TSIG mechanism added in RFC 2845/4635
  - calculates HMAC-MD5/SHA over the complete DNS packet
  - adds this as a "pseudo"-TSIG-RR
  - uses secret keys
  - may use a "pseudo"-TKEY-RR for key exchange (RFC 2930)
- SIG(0) mechanism added in RFC 2931
  - uses public keys
  - uses DNSSEC like mechanisms
  - extends DNSSEC to cover complete DNS packets