CSCE 463/612
Networks and Distributed Processing
Spring 2024

Application Layer V

Dmitri Loguinov
Texas A&M University

February 16, 2024

Chapter 2: Roadmap

2.1 Principles of network applications

2.2 Web and HTTP

2.3 FTP

2.4 Electronic Mail

- SMTP, POP3, IMAP

2.5 DNS

2.6 P2P file sharing

2.7 Socket programming with TCP

2.8 Socket programming with UDP

2.9 Building a Web server

DNS Records

Replace A with AAAA for IPv6

DNS: distributed database of resource records (RR)

(name, value, type, ttl)

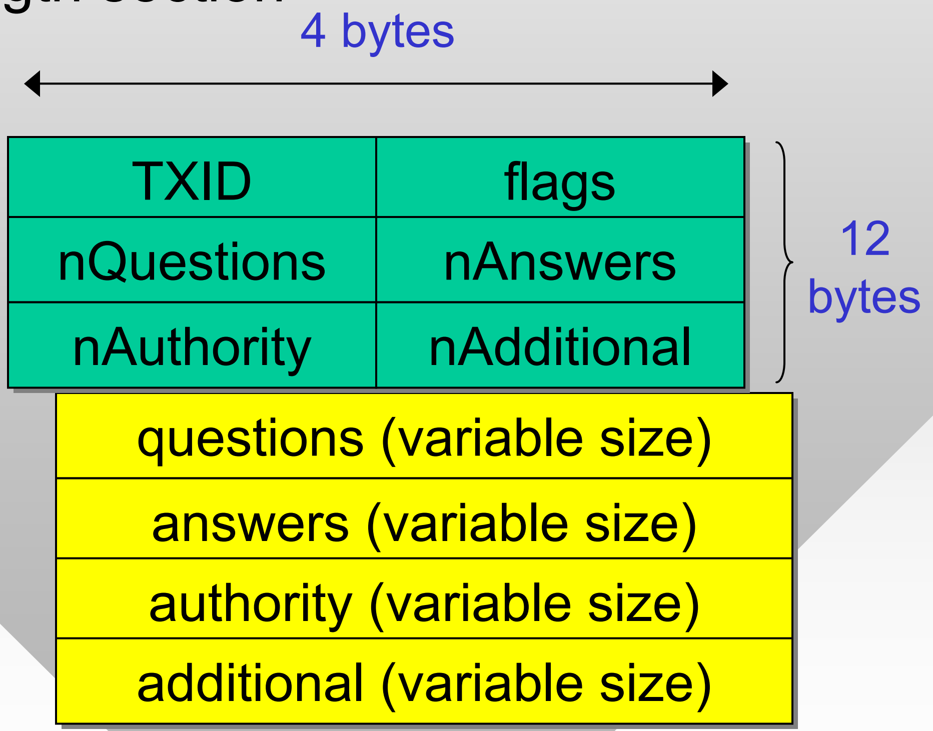
- Type A
  - name = host
  - value = IPv4 address (4 byte DWORD)
- Type NS
  - name = domain
  - value = hostname of authoritative name server for this domain
- Type CNAME
  - name = host
  - value = host it’s aliased to
  - Reduces manual effort to change IPs and other records
- Type MX
  - name = domain
  - value = name of SMTP server associated with domain

Reverse Queries

- Reverse DNS lookups are performed using a special construction of a fake DNS name
  - Reason: DNS resolves names from right to left with the semantics of going from the most general to the most specific
  - In IPs, the MSB is most general, LSB is most specific
- The IP address is reversed and is followed by “in-addr.arpa” (or “ip6.arpa” for IPv6)
  - Example: 128.194.135.65 is requested as 65.135.194.128.in-addr.arpa
  - The query type must be set to PTR
- RFC 1035 (1987) describes DNS headers/commands
  - Also see http://www.networksorcery.com/enp/protocol/dns.htm

DNS Protocol, Messages

- Query and reply messages use same format
  - Packet starts with a fixed DNS header (12 bytes)
  - Followed by a variable-length section
- Transaction ID (TXID)
  - 16-bit number assigned by client to each query
  - Echoed by server in response packet
- Flags specify the type of request being made and response status
- The other 4 fields provide a count of records in each variable-size section



4 bytes

12 bytes

TXID	flags
nQuestions	nAnswers
nAuthority	nAdditional

questions (variable size)

answers (variable size)

authority (variable size)

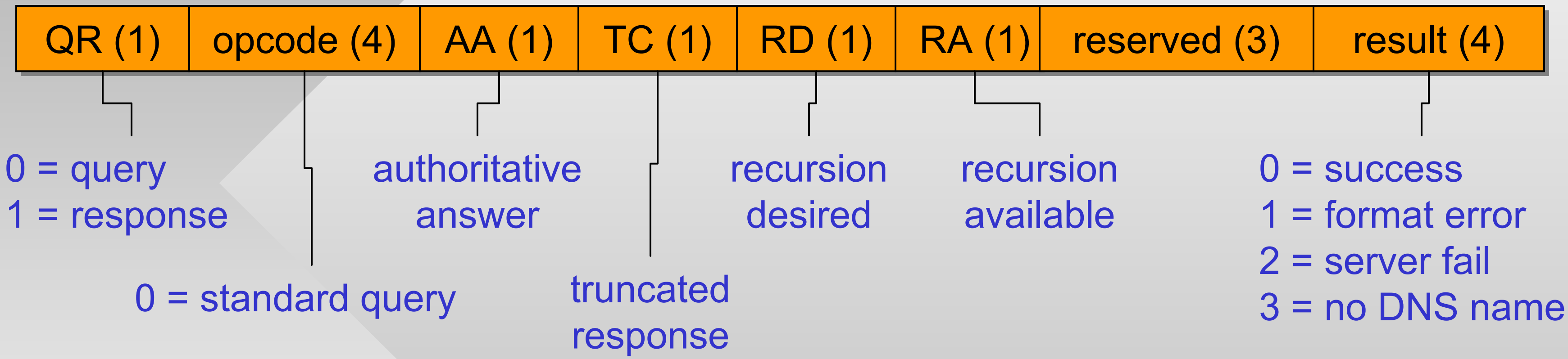
additional (variable size)

DNS Protocol, Messages

TX ID	flags
nQuestions	nAnswers
nAuthority	nAdditional
questions (variable size)	
answers (variable size)	
authority (variable size)	
additional (variable size)	

- Queries contain only the question section
  - Most servers expect one question per packet
- Response packets always repeat the question
  - Safety mechanism if TXID runs into collision at the client
- Authority section carries NS record(s)
  - Used during iterative lookups to specify the next DNS server to query (similar to HTTP redirects)
- All numbers are in network byte order
  - Use proper conversion (i.e., htons() in this case)

DNS Flags



QR (1)	opcode (4)	AA (1)	TC (1)	RD (1)	RA (1)	reserved (3)	result (4)
0 = query 1 = response	0 = standard query	authoritative answer	truncated response	recursion desired	recursion available		0 = success 1 = format error 2 = server fail 3 = no DNS name

- For binary fields, 1 = true and 0 = false
- For query packets:
  - Set RD = 1; all other fields are zero
  - Specify nQuestions = 1
  - Correctly create the actual question and append it to the header in the packet buffer

Nslookup Usage (Windows)

- nslookup -querytype=mx cs.tamu.edu

```
Server:  gw.irl.cs.tamu.edu
Address:  128.194.135.72

Non-authoritative answer:
cs.tamu.edu MX preference = 100, mail exchanger = smtp-relay.tamu.edu
cs.tamu.edu MX preference = 10, mail exchanger = pine.cs.tamu.edu

smtp-relay.tamu.edu     internet address = 165.91.143.199
pine.cs.tamu.edu        internet address = 128.194.138.12
```

cached answers and additional records

smaller preference value means higher priority

- nslookup -querytype=hinfo cs.tamu.edu

```
Server:  gw.irl.cs.tamu.edu
Address:  128.194.135.72

cs.tamu.edu
        primary name server = dns1.cs.tamu.edu
        responsible mail addr = root.cs.tamu.edu
        serial  = 2006090513
        refresh = 1800 (30 mins)
        retry   =  900 (15 mins)
        expire  = 1209600 (14 days)
        default TTL = 3600 (1 hour)
```

Nslookup Usage (Windows)

- nslookup -querytype=ptr 12.138.194.128.in-addr.arpa

```
Server:  gw.irl.cs.tamu.edu
Address:  128.194.135.72

Non-authoritative answer:
12.138.194.128.in-addr.arpa     name = mail.cs.tamu.edu
12.138.194.128.in-addr.arpa     name = pine.cs.tamu.edu
12.138.194.128.in-addr.arpa     name = pophost.cs.tamu.edu
12.138.194.128.in-addr.arpa     name = mailhost.cs.tamu.edu
12.138.194.128.in-addr.arpa     name = pop.cs.tamu.edu
12.138.194.128.in-addr.arpa     name = imap.cs.tamu.edu
```

nslookup performs string reversal transparently, but hw2 will need to do this explicitly

- nslookup -querytype=ptr 12.1.55.186

```
Server:  s18.irl.cs.tamu.edu
Address:  128.194.135.58

Non-authoritative answer:
186.55.1.12.in-addr.arpa        canonical name = 186.184/29.55.1.12.in-addr.arpa

186.184/29.55.1.12.in-addr.arpa name = outlook.milestonescientific.com
```

Using UDP

- DNS runs over UDP that has no connection phase
  - Each request and response is exactly 1 packet
  - Calls to recvfrom() and sendto() correspond to receiving/ sending 1 packet from/to a socket
  - No need to loop on receive
- General idea:

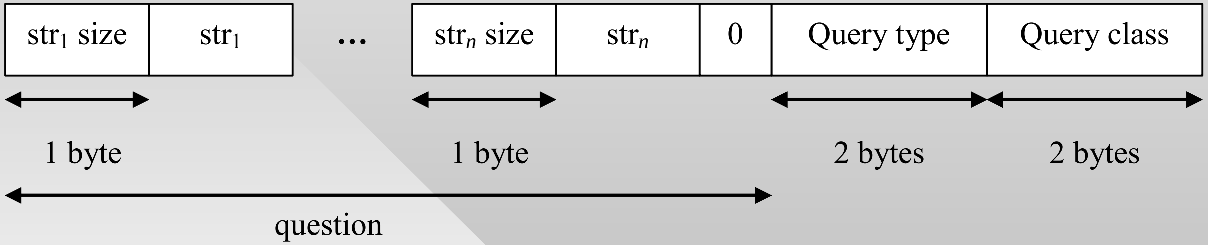
```
sock = socket (AF_INET, SOCK_DGRAM, 0);
// bind sock to port 0 – see the handout
len = CreateRequest(buf, hostname);
while (work to be done) {
  sendto (sock, buf, len, 0, &addressTo, ...);
  ...
  if (select (...) > 0) {
            recvfrom (sock, recvBuf, ..., 0, &addressFrom...);
            parseResponse (recvBuf);
  }
}
closesocket (sock);
```

Homework #2

TX ID	flags
nQuestions	nAnswers
nAuthority	nAdditional

questions (variable size)
answers (variable size)
authority (variable size)
additional (variable size)

- Unlike HTTP, all fields are binary
  - Make sure to refresh pointer usage
- Question format:



str$_1$ size	str$_1$	…	str$_n$ size	str$_n$	0	Query type	Query class
1 byte			1 byte			2 bytes	2 bytes

question

- Create structs for fixed headers
  - Fill in the values (flags: DNS_QUERY and DNS_RD, nQuestions = 1)
  - Allocate memory for the packet
  - Write question into buffer

```
class QueryHeader {
    u_short type;
    u_short class;
};
```

```
class FixedDNSheader {
    u_short ID;
    u_short flags;
    u_short questions;
    ...
};
```

Homework #2

- High-level operation for DNS questions:

```
char packet [MAX_DNS_LEN];              // 512 bytes is max
char host[] = “www.google.com”;
int pkt_size = strlen(host) + 2 + sizeof(FixedDNSheader) + sizeof(QueryHeader);

// fixed field initialization
FixedDNSheader *dh = (FixedDNSheader *) packet;
QueryHeader *qh = (QueryHeader*) (packet + pkt_size - sizeof(QueryHeader));
dh->ID = ...
dh->flags = ...
...
qh->type = ...
qh->class = ...

// fill in the question
MakeDNSquestion (dh + 1, host);
// transmit to Winsock
sendto (sock, packet, ...);
```

- If packet is incorrectly formatted, you will usually get no response; use Wireshark to check outgoing packets

Homework #2

```
class DNSanswerHdr {
          u_short type;
          u_short class;
          u_int ttl;
          u_short len;
};
```

- Formation of questions:

```
makeDNSquestion (char* buf, char *host) {
      while(words left to copy){
            buf[i++] = size_of_next_word;
            memcpy (buf+i, next_word, size_of_next_word);
            i += size_of_next_word;
      }
      buf[i] = 0;    // last word NULL-terminated
}
```

- Answers start with an RR name, followed by a fixed *DNS answer header*, followed by the answer itself
  - Uncompressed answer (not common)

```
0x3 “irl” 0x2 “cs” 0x4 “tamu” 0x3 “edu” 0x00
<DNSanswerHdr> <ANSWER>
```

  - Compressed (2 upper bits 11, next 14 bits jump offset)

```
0xC0 0x0C <DNSanswerHdr> <ANSWER>
```

- For type-A questions, the answer is a 4-byte IP

Homework #2

```
class DNSanswerHdr {
          u_short type;
          u_short class;
          u_int ttl;
          u_short len;
};
```

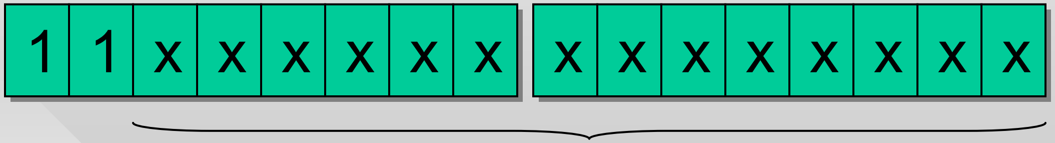
- To check the header
  - Hex printout on screen
  - Wireshark
- What is `sizeof(DNSanswerHdr)`?
  - The actual size is 10 bytes, but the compiler will align/pad it to 4-byte boundary (so 12)
- Remember to change struct packing of all classes that define binary headers to 1 byte

```
#pragma pack(push,1)
// define headers here
#pragma pack(pop)
```

- Caveats (must be properly handled):
  - Exceeding array boundaries on jumps
  - Infinite looping on compressed answers

Homework #2

- How to check if compressed and read 14-bit offset?
  - Suppose array `char *ans` contains the reply packet
  - The answer begins within this array at position `curPos`



1	1	x	x	x	x	x	x	x	x	x	x	x	x	x	x

14 bits

```
char *ans;  // points to reply buffer
if (ans [curPos] >= 0xC0)
    // compressed; so jump
else
    // uncompressed, read next word
```

```
char *ans;  // points to reply buffer
if ( (ans [curPos] >> 6) == 3)
    // compressed; so jump
else
    // uncompressed, read next word
```

```
// computing the jump offset
int off = ( (ans[curPos] & 0x3F) << 8) + ans[curPos + 1];
```

- The first two checks will generally fail
  - Use only unsigned chars when reading buffer!

Homework #2

- Note that jumps may appear mid-answer

  `0x3 “irl” 0xC0 0x22 <DNSanswerHdr> <ANSWER>`
- Jumps may be nested, but must eventually end with a 0-length word
  - Need to remember the position following the very first jump so that you can come back to read DNSanswerHdr
- Replies may be malicious or malformatted
  - Homework must avoid crashing
- If AAAA (IPv6) answers are present, skip
  - Use `DNSanswerHdr::len` to jump over unknown types
- Caution with TAMU VPN
  - Malformed packets are filtered out