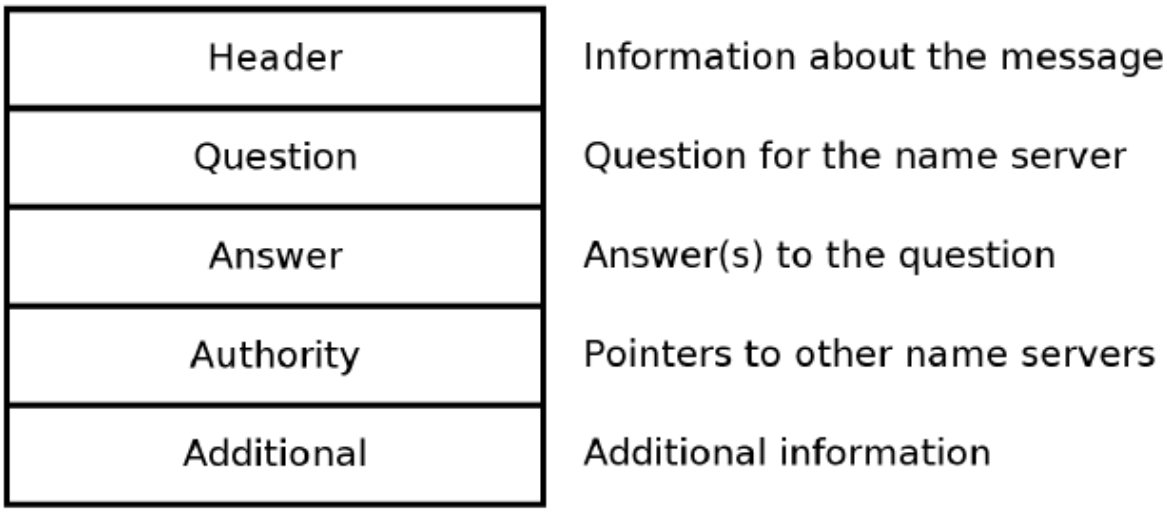
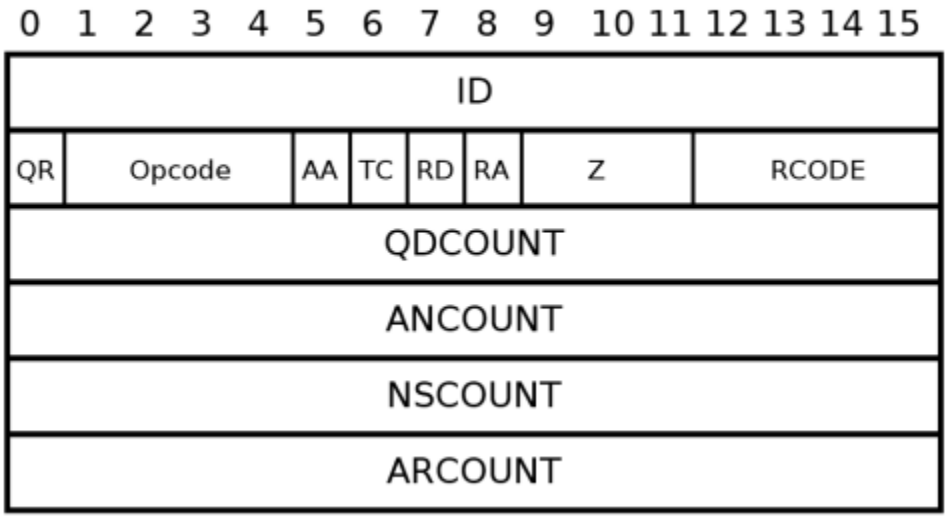
DNS Message Format



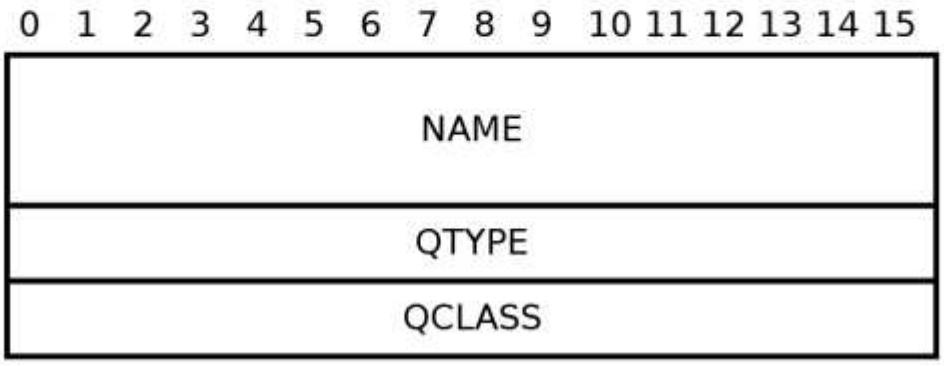
Header Format



0xAB, 0xCD, /* ID */
0x01, 0x00, /* Set recursion */

The DNS **Question Format** consists of a name followed by two 16-bit values—QTYPE and QCLASS. This **Question Format** is illustrated as follows

Question Format



Name Example

0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15
3	'w'
'w'	'w'
7	'e'
'x'	'a'
'm'	'p'
'l'	'e'
3	'c'
'o'	'm'
0	

Here's the complete query message with the domain name cs.lasierra.edu to send to the DNS server. The query length is 33.

```
char query[1024] = {
0xAB, 0xCD, /* ID */
0x01, 0x00, /* Set recursion */
0x00, 0x01, /* QDCOUNT */
0x00, 0x00, /* ANCOUNT */
0x00, 0x00, /* NSCOUNT */
0x00, 0x00, /* ARCOUNT */
0x02, 'c', 's',
0x08, 'l', 'a', 's', 'i', 'e', 'r', 'r', 'a',
0x03, 'e', 'd', 'u',
0x00,
0x00, 0x01, /* QTYPE A (IPv4)=0x01; AAAA (IPv6)=0x1C; all=0xff*/
0x00, 0x01 /* QCLASS */
};
int query_size = 33;
```

Code template steps:

```
getaddrinfo()    // configure remote address DNS server 8.8.8.8:53
socket()         // create socket
sendto()         // send query message to DNS server
recvfrom()       // get response message
print out IP address from the response message
```

```
#define BYTE_TO_BINARY(byte)  \
  ((byte) & 0x80 ? '1' : '0'), \
  ((byte) & 0x40 ? '1' : '0'), \
  ((byte) & 0x20 ? '1' : '0'), \
  ((byte) & 0x10 ? '1' : '0'), \
  ((byte) & 0x08 ? '1' : '0'), \
  ((byte) & 0x04 ? '1' : '0'), \
  ((byte) & 0x02 ? '1' : '0'), \
  ((byte) & 0x01 ? '1' : '0')

void print_raw_message(const char *message, int msg_length) {
     int i;
     for (int i=0; i< msg_length; i+=2) {
          unsigned char r = message[i];
          unsigned char r2 = message[i+1];
          printf("%02d:    %02X   %02X   %c%c%c%c%c%c%c%c
%c%c%c%c%c%c%c%c\n", i, r, r2, BYTE_TO_BINARY(r),BYTE_TO_BINARY(r2));
     }
}

const unsigned char *print_name(const unsigned char *msg,
        const unsigned char *p, const unsigned char *end) {

    if (p + 2 > end) {
        fprintf(stderr, "End of message.\n"); exit(1);}

    if ((*p & 0xC0) == 0xC0) {
        const int k = ((*p & 0x3F) << 8) + p[1];
        p += 2;
        printf(" (pointer %d) ", k);
        print_name(msg, msg+k, end);
        return p;

    } else {
        const int len = *p++;
        if (p + len + 1 > end) {
            fprintf(stderr, "End of message.\n"); exit(1);}

        printf("%.*s", len, p);
        p += len;
        if (*p) {
            printf(".");
            return print_name(msg, p, end);
        } else {
            return p+1;
        }
    }
}

void print_dns_message(const char *message, int msg_length) {
```

```
    if (msg_length < 12) {
        fprintf(stderr, "Message is too short to be valid.\n");
        exit(1);
    }

    const unsigned char *msg = (const unsigned char *)message;

    printf("ID = %0X %0X\n", msg[0], msg[1]);

    const int qr = (msg[2] & 0x80) >> 7;
    printf("QR = %d %s\n", qr, qr ? "response" : "query");

    const int opcode = (msg[2] & 0x78) >> 3;
    printf("OPCODE = %d ", opcode);
    switch(opcode) {
        case 0: printf("standard\n"); break;
        case 1: printf("reverse\n"); break;
        case 2: printf("status\n"); break;
        default: printf("?\n"); break;
    }

    const int aa = (msg[2] & 0x04) >> 2;
    printf("AA = %d %s\n", aa, aa ? "authoritative" : "");

    const int tc = (msg[2] & 0x02) >> 1;
    printf("TC = %d %s\n", tc, tc ? "message truncated" : "");

    const int rd = (msg[2] & 0x01);
    printf("RD = %d %s\n", rd, rd ? "recursion desired" : "");

    if (qr) {
        const int rcode = msg[3] & 0x0F;
        printf("RCODE = %d ", rcode);
        switch(rcode) {
            case 0: printf("success\n"); break;
            case 1: printf("format error\n"); break;
            case 2: printf("server failure\n"); break;
            case 3: printf("name error\n"); break;
            case 4: printf("not implemented\n"); break;
            case 5: printf("refused\n"); break;
            default: printf("?\n"); break;
        }
        if (rcode != 0) return;
    }

    const int qdcount = (msg[4] << 8) + msg[5];
    const int ancount = (msg[6] << 8) + msg[7];
    const int nscount = (msg[8] << 8) + msg[9];
    const int arcount = (msg[10] << 8) + msg[11];

    printf("QDCOUNT = %d\n", qdcount);
```

```
    printf("ANCOUNT = %d\n", ancount);
    printf("NSCOUNT = %d\n", nscount);
    printf("ARCOUNT = %d\n", arcount);


    const unsigned char *p = msg + 12;
    const unsigned char *end = msg + msg_length;

    if (qdcount) {
        int i;
        for (i = 0; i < qdcount; ++i) {
            if (p >= end) {
                fprintf(stderr, "End of message.\n"); exit(1);}

            printf("Query %2d\n", i + 1);
            printf("  name: ");

            p = print_name(msg, p, end); printf("\n");

            if (p + 4 > end) {
                fprintf(stderr, "End of message.\n"); exit(1);}

            const int type = (p[0] << 8) + p[1];
            printf("  type: %d\n", type);
            p += 2;

            const int qclass = (p[0] << 8) + p[1];
            printf(" class: %d\n", qclass);
            p += 2;
        }
    }

    if (ancount || nscount || arcount) {
        int i;
        for (i = 0; i < ancount + nscount + arcount; ++i) {
            if (p >= end) {
                fprintf(stderr, "End of message.\n"); exit(1);}

            printf("Answer %2d\n", i + 1);
            printf("  name: ");

            p = print_name(msg, p, end); printf("\n");

            if (p + 10 > end) {
                fprintf(stderr, "End of message.\n"); exit(1);}

            const int type = (p[0] << 8) + p[1];
            printf("  type: %d\n", type);
            p += 2;

            const int qclass = (p[0] << 8) + p[1];
```

```
            printf(" class: %d\n", qclass);
            p += 2;

            const unsigned int ttl = (p[0] << 24) + (p[1] << 16) +
                (p[2] << 8) + p[3];
            printf("   ttl: %u\n", ttl);
            p += 4;

            const int rdlen = (p[0] << 8) + p[1];
            printf(" rdlen: %d\n", rdlen);
            p += 2;

            if (p + rdlen > end) {
                fprintf(stderr, "End of message.\n"); exit(1);}

            if (rdlen == 4 && type == 1) {
                /* A Record */
                printf("Address ");
                printf("%d.%d.%d.%d\n", p[0], p[1], p[2], p[3]);

            } else if (rdlen == 16 && type == 28) {
                /* AAAA Record */
                printf("Address ");
                int j;
                for (j = 0; j < rdlen; j+=2) {
                    printf("%02x%02x", p[j], p[j+1]);
                    if (j + 2 < rdlen) printf(":");
                }
                printf("\n");

            } else if (type == 15 && rdlen > 3) {
                /* MX Record */
                const int preference = (p[0] << 8) + p[1];
                printf("  pref: %d\n", preference);
                printf("MX: ");
                print_name(msg, p+2, end); printf("\n");

            } else if (type == 16) {
                /* TXT Record */
                printf("TXT: '%.*s'\n", rdlen-1, p+1);

            } else if (type == 5) {
                /* CNAME Record */
                printf("CNAME: ");
                print_name(msg, p, end); printf("\n");
            }

            p += rdlen;
        }
    }
```

```
    if (p != end) {
        printf("There is some unread data left over.\n");
    }

    printf("\n");
}
```