“DNS中继服务器”的实现

2017211302 2017210078 林　杨
2017211303 2017211175 王　瑞
2017211304 2017211219 范乾一

2019 年 10 月 1 日

7. 参考

1.题目概述

设计一个 DNS 服务器程序,读入"域名-IP 地址"对照表,当客户端查询域名对应的 IP 地址时,用域名检索该对照表,三种检索结果:

- 检索结果为 IP 地址 0.0.0.0,则向客户端返回「域名不存在」的报错消 息（不良网站拦截功能）
- 检索结果为普通 IP 地址,则向客户返回这个地址（服务器功能）
- 表中未检到该域名,则向因特网 DNS 服务器发出查询,并将结果返给 客户端（中继功能）
- 需考虑多个计算机上的客户端会同时查询,需要进行消息ID的转换.

2.理论基础

2.1 DNS 简介

由于 IP 地址不易读且不固定,若直接使用 IP 地址,人们日常连接网络或是识别参与分布式应用的主机等问题将会十分困难.为了解决这个问题,互联网允许使用主机名称来识别主机.通过对名称的解析来完成其与 IP 地址的互相和转换.

互联网中最通用的一种名称解析系统是家喻户晓的域名系统 (DNS）.DNS 是一个分布式的client-server 网络数据库.建立于 TCP/IP 上的应用程序使用它来完成主机名称和 IP 地址之间的互相转换.既然称之为分布式的系统,DNS 在提供了允许客户机和服务器相互通信的协议的同时,也提供了服务器之间交互信息的协议.

DNS 建立于 TCP/IP协议之上.在请求 TCP 打开一个连接或使用 UDP 发送一个单播数据报之前,应用程序必须将主机名称转换为 IPv4 与/或 IPv6 地址.

DNS 主要由三部分组成,分别是域名空间和资源记录(DOMAIN NAME SPACE和 RESOURCE RECORDS),名称服务器(NAME SERVERS)和解析器(RESOLVERS).下面将对这几部分进行逐一说明.

2.2 DNS 名称空间与命名语法

称空间是一棵域名树,位于顶部的树根未命名.树的最高层是所谓的顶级域名（TLD）.

DNS 名称树中 TLD 下面的名称进一步划分成组,称为子域名.一个域名包含一系列的由点分开的 标签.名称代表名称层级中的一个位置,句点是层次结构分隔符,并且按名称中自右至左的顺序沿树下降.每个标签最多可到 63 个字符长.

2.3 DNS 服务器和区域

部分 DNS 名称空间的管理责任分配给个人或组织,他们通过安置 DNS 服务器来存储名称空间的相关信息,以便本地或者外界互联网用户查询名称.

在 DNS 服务器中,管理授权的单位称为区域.一个区域是 DNS 名称空间的一棵子树,它可以独立管理而不受其他区域影响.每一个域名都存在于某个区域中,每当一个新记录 添加到区域中时,该区域的 DNS 服务器为该新条目分配一个域名和IP,并且将这些信息保存到名称服务器的数据库中.

2.4 缓存

DNS 服务器会缓存它们学习的区域信息,直到该条目的TTL到时间限制为止.

使用缓存的信息来应答查询请求,可以减少 DNS 消息的流量.每个 DNS 记录有自己的 TTL 以控制其缓存的时间.

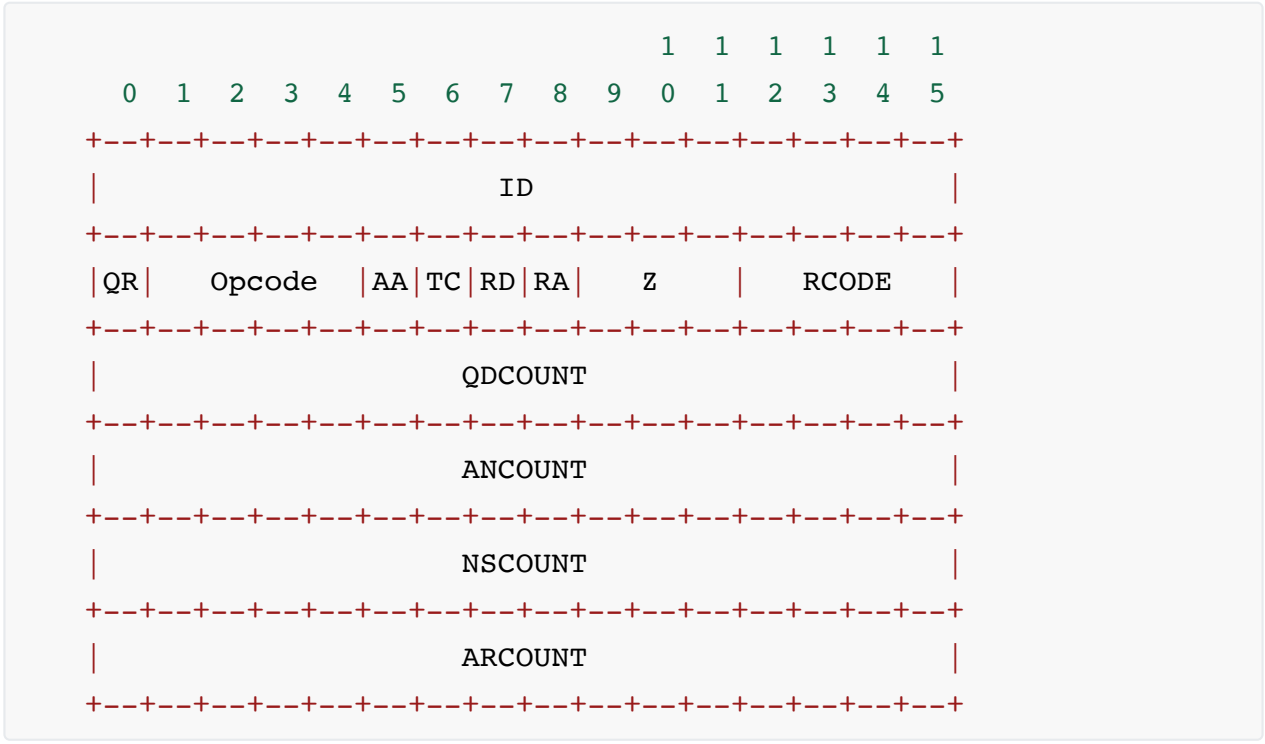
缓存同时适用于成功的解析和不成功的解析（否定缓存）,如果一个请 求返回域名不存在时,将该事实缓存下来,之后再次请求该不存在的域名时 可以直接返回,从而降低流量.基于这个原理,我们可以对 DNS 进行污染,以屏蔽某些不良网站的访问.

2.5 DNS 协议

DNS 协议主要由两个主要部分组成,对 DNS 特定名称查询的`查询/响应协议`和 DNS 服务器用于`交换数据库记录的协议`.

2.5.1 DNS 消息格式

基本的 DNS 消息格式如下所示.基本的 DNS 消息格式以固定的 12 字



节头部开始,其后跟随 4 个可变长度的区段：**问题、回答、授权记录、额外记录**.

名称和标签

DNS 消息末尾的可变长度区段包含问题、回答、授权信息和 额外信息.每一个问题和 RR 以它所涉及的名称开始.每个名称由一系列 的标签组成.标签类型有两种：数据标签和压缩标签.数据标签包含构成一 个标签的字符；压缩标签充当指向其他标签的指针.当相同字符串的多个副 本在多个标签中出现时,压缩标签有助于节省 DNS 信息的空间.

数据标签

每个数据标签以 1 字节的计数开始,该计数指定了紧随其后的字节数目,名称以值为 0 的字节结束.例如,名称 `www.leetcode.com` 的编 码如 `3www8leetcode3com0` 所示.对于数据标签,每个标签的长度字节的值必须在 0 到 63 之间.

值得注意的是,`3www8leetcode3com0` 中的 3,8,3,0 是整数类型而非字符类型.

压缩标签

在许多情况下,DNS 响应消息在涉及相同的名称 时,DNS 消息中的相同字符就会重复.为了避免冗余和节省空间,使用了一种压缩机制.DNS 消息中,通过指针进行跳转以进行压缩.

若前面的单一计数字节的 2 个高位为 `11`,剩余的位与随后的字节中的位组合形成一个 14 位的指针.该指针代表一个偏移量,给出了距离 DNS 消息开始处的字节数.压缩标签能够指向距离开始处多达 16383 个字节的位置,继续补齐剩余的标签.

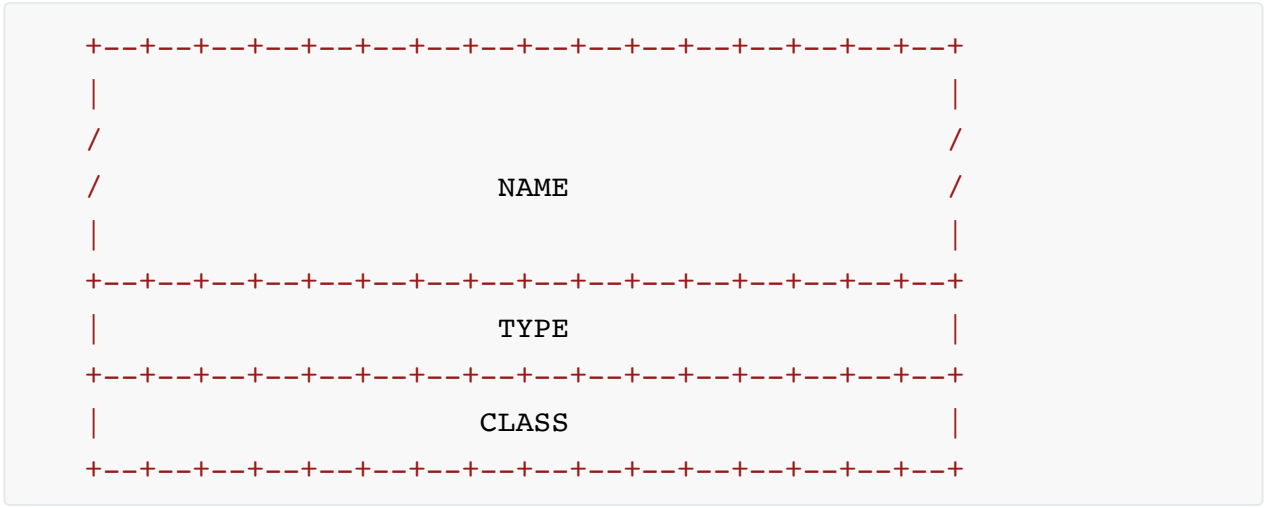
2.5.2 UDP

对于 UDP 来说,DNS 的知名端口号是 53,常见的格式是使用如下所 示的 UDP/IPv4 的数据报结构.

```
            +-----------------------------------------+
IPv4 Header |                                         |
            +-----------------------------------------+
UDP Header  |                                         |
            +-----------------------------------------+
Header      | OPCODE=SQUERY, RESPONSE, AA             |
            +-----------------------------------------+
Question    | QNAME=SRI-NIC.ARPA., QCLASS=IN, QTYPE=NS |
            +-----------------------------------------+
Answer      | <empty>                                 |
            +-----------------------------------------+
Authority   | <empty>                                 |
            +-----------------------------------------+
Additional  | <empty>                                 |
            +-----------------------------------------+
```

2.5.3 查询字段格式

查询字段中每个问题的格式如下所示.



查询名称是要被查询的域名,使用我们之前描述的标签的编码,即数据标签或压缩标签.

每个问题都有查询类型和查询类.类的值是 1、254 或 255,分别表示互联网类、没有类或所有类,其他值通常不用于 TCP/IP 网络.

查询类型字段包含一个值,指明正在执行的查询类型.最常见的查询类型是 A（如果启用 IPv6 的 DNS 解析,则是 AAAA）,这意味着需要一个与查询名称对应的 IP 地址.

2.5.4 回答、授权和额外信息字段格式

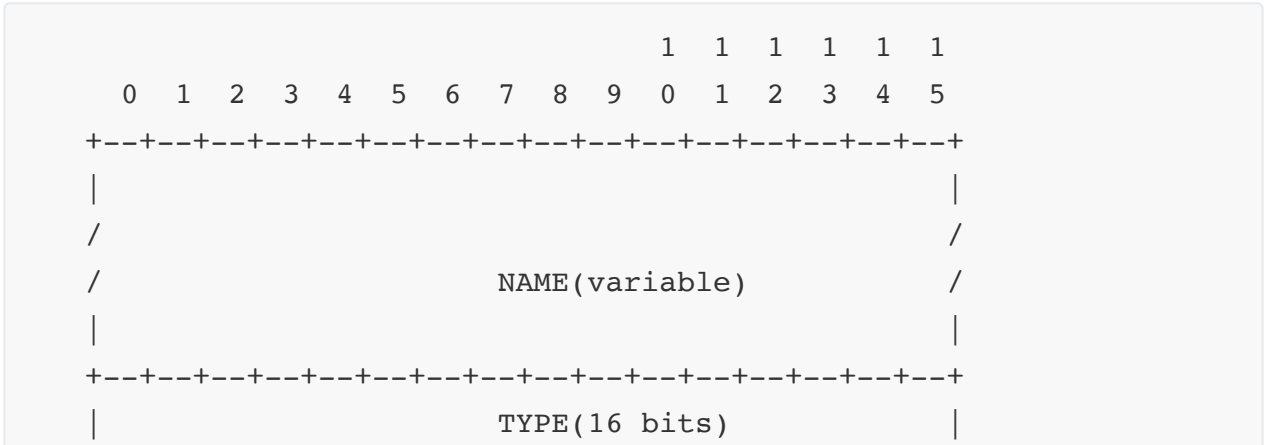
DNS 报文中的最后三个区段——回答、授权和额外信息,是包含 RR 的 集合.每个资源记录RR的格式如下所示.

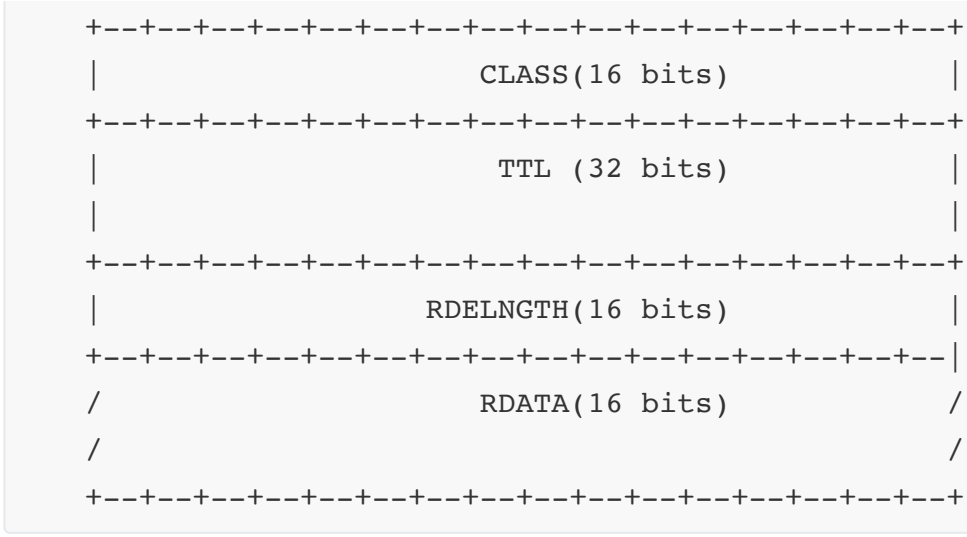
名称字段是随后的资源数据对应的域名,它与我们之前描述的名称和标签的格式相同.

类型字段指定为 RR 类 型代码中的一个.这些和我们之前描述的查询类型值相同.对于互联网数据来说,类字段是 1.

TTL 字段是 RR 可以被缓存的秒数.

资源数据长度字段指定了资源数据字段中包含的字节数.数据的格式取决于类型.例如,A 记录（类型 1）在 RDATA 域中有一个 32 位的 IPv4 地址.AAAA 则表明 RDATA 中有一个 IPv6 地址.




CLASS(16 bits)
TTL (32 bits)
RDELNGTH(16 bits)
RDATA(16 bits)

3. 功能设计

3.1 不良网站拦截

若在本地域名-IP 记录表`nameTable`中查找到的域名对应的 IP 为 0.0.0.0 时,说明该域名不存在.将报文的响应码设置为 3 发送至客户端,表示当前查找的域名不存在.由此,利用这个机制,我们在本地域名-IP 记录表中将不良网站的 IP 地址设置为 0.0.0.0,就实现了不良网站的拦截功能.

3.2 服务器

若在本地 域名-IP 记录表`nameTable`中匹配到所查询的域名,且检索结果为普通 IP 地址时(非不良网站),表示我们在本地表中找到了域名对应的地址.将该地址置入报文的 ANSWER 字段并将报文返回给客户端.

3.3 中继

当在本地nameTable表中匹配不到所查询的域名时,转发该报文到远端服务器.

为了实现多客户端并发查询,即允许第一个查询尚未得到答案就响应另一个客户端的查询请求.我们通过在中继服务器中进行 ID 的转换的方法实现了并发查询.用一个本地 currentID 值替换原 ID 发送给源端服务器,并将 currentID 与旧 ID 的关系记录到 Record 中.当收到一个来自远端服务器的报文时,查表表确定旧 ID 以及客户端的地址,将 ID 字段修改为之前的 ID,发送回原客户端.

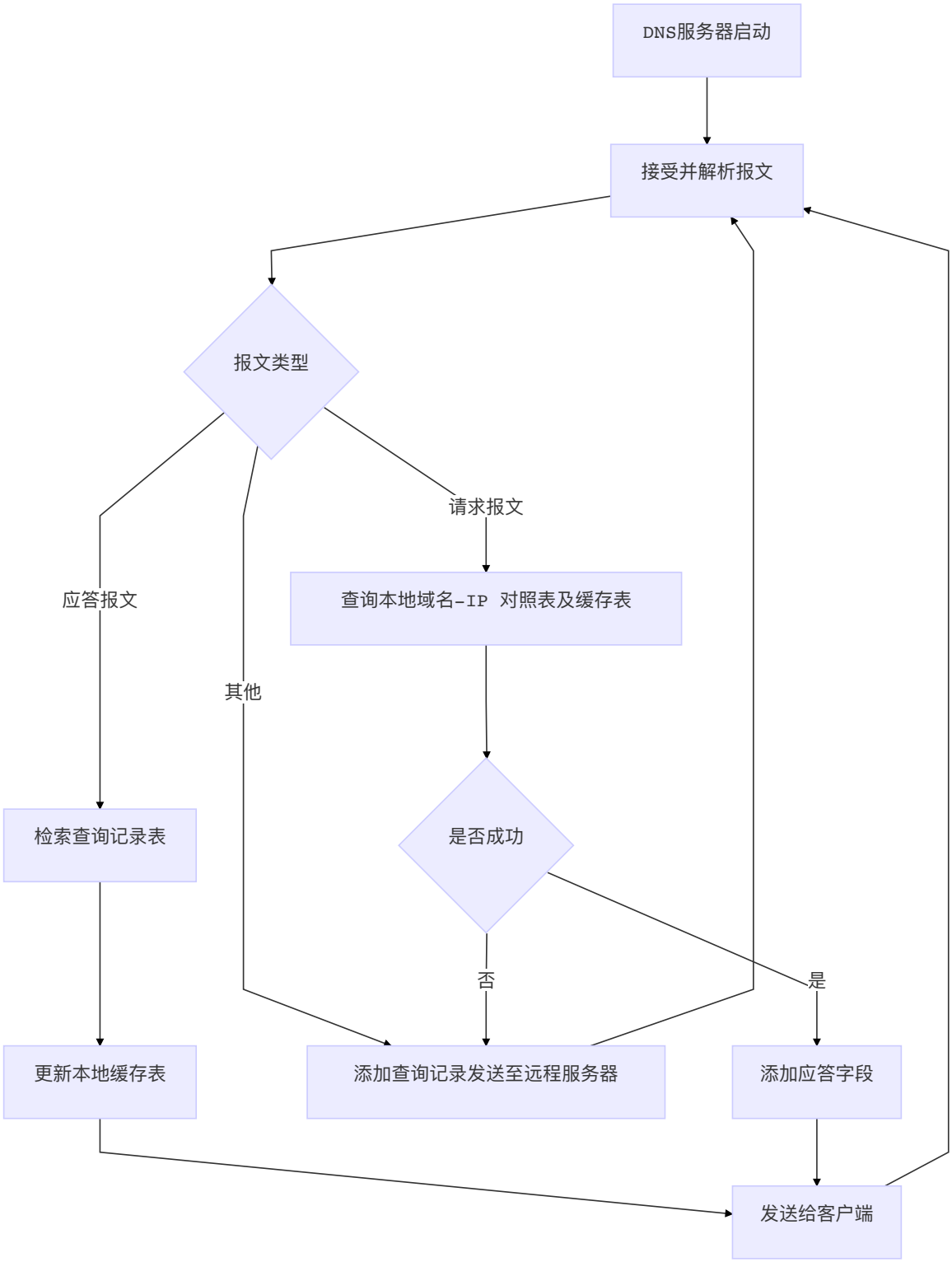
3.4 缓存

为减少 DNS 消息的流量,我们 DNS 具有了缓存的功能.缓存的信息保存表项`nameTable`之中.当我们的 DNS 中继服务器收到远端服务器的 DNS 消息时,则从buffer中提取出 ID,IP,域名以及TTL.

之后再收到客户端的查询时,我们检查本地的 域名-IP 记录表`nameTable`,如果成功找到则将 IP 返回,否则再向远端服务器发送询问.每次查询到结果都检查 TTL观察其是否仍有效.

4. 模块划分 & 代码详解

4.0 流程图 & 对模块划分的总述



模块名	模块功能
socket_manager	负责网络通信
name_table	负责本地域名-IP 对照表以及缓存表的相关操作
record_table	负责 DNS 中继服务器中查询记录表的相关操作
message	负责字节流和各种结构体之间的转换
parser	负责报文进行解析,并发送相应的应答包

全体函数

```
int getopt(int argc, char **argv);
char *mkcopy(const char *src);
void printNameTable();
void init(const char *path);
void freeSpace();
void init(const char *path);
void freeSpace();
void get16bits(uint8_t **ptr, uint16_t *value);
void get32bits(uint8_t **ptr, uint32_t *value);
void ExtractHeader(uint8_t **ptr);
void extractName(uint8_t **ptr, char **name);
void ExtractQuestion(uint8_t **ptr);
void printName(uint8_t *name);
void ExtractRR(uint8_t **ptr);
void ExtractMessage();
void printMessage(time_t *time);
char *transName(uint8_t *name);
int findTable(char *name);
void createAnswer(char *IP, char *name);
void put16bits(uint16_t value, uint8_t **ptr, int *buffersize);
void put32bits(uint16_t value, uint8_t **ptr, int *buffersize);
void constructHeader(uint8_t **ptr, int *buffersize);
void constructQuestion(uint8_t **ptr, int *buffersize);
void constructRR(uint8_t **ptr, int *buffersize);
int constructMessage(char *IP, char *name);
void insertRecord();
void insertEntry(uint32_t IP, char *name);
int findRecord(uint16_t ID, SOCKADDR_IN *temp, char **name);
void sendToServer(int bytes);
```

```
void sendAnswer(int recvbytes);
void recvMessage();
void analyzeResponse(int bytes);
```

4.1 socket_manager

该模块负责网络通信.大部分有关 socket 的配置及使用都包含在该模块之中.

```
void sendToServer(int bytes);
void sendAnswer(int recvbytes);
void recvMessage();
```

`sendToServer(int bytes);`发送存储在buffer 中的内容给上级服务器. `void sendAnswer(int recvbytes);`发送存储在 buffer 中的内容给请求服务器. `void recvMessage();`接受消息并存储到 buffer 中.

无需指定发送方和接收方,函数内部会自动根据存储在 buffer 中的ID 值进行转发.

4.2 name_table

该模块负责维护一个 DNS本地域名映射到期对应 IP 的对照表以及缓存表的相关操作.

该模块使用的数据结构如下:

```
typedef struct Entry {
  char IP[16];
  char *name;
} Entry;

struct NameTable {
  Entry nametable[MAX_TABLEENTRY];
  int size;
} nameTable;
```

本模块使用的函数定义如下:

```
void init(const char *path);
int findTable(char *name);
void insertEntry(uint32_t IP, char *name);
```

`init()` 用于根据`DNSrelay.txt`中所预存的信息初始化`nameTable`表.对`DNSrelay.txt` 的读写分散在其余各个函数中.

`insertEntry(uint32_t IP, char *name)`用以插入一个表项.

4.3 record_table

该模块负责 DNS 中继服务器中查询记录表的相关操作.

该模块使用的数据结构如下:

```
uint16_t currentID;

typedef struct Record {
  uint16_t currentId;
  SOCKADDR_IN senderAddr;
  uint16_t ID;
  char *name;
} Record;

struct RecordTable {
  Record recordtable[MAX_RECORD];
  int size;
} recordTable;
```

我们直接将每一个requset 的的 currentID,即 DNS 服务器本地保存的,用以实现并发中继的 ID,保存在 Record 结构体中.

本模块使用的函数定义如下:

```
void insertRecord();
int findRecord(uint16_t ID, SOCKADDR_IN *temp, char **name);
```

`insertRecord()`将 currentID 加入到一条 Record 中,并将整条 Record 加入到 table . 全部信息存储在报文 message 中.

`findRecord()` 用于当收到一条服务器返回的response时,在 table 中查询其对应的原 ID 和客户端地址.

4.4 message

该模块负责结构体和字节流的转换.我们通过 UDP 发送和接收的是字节流,而在程序里处理的报文是抽象成结构体了的.需对二者进行转换.

该模块使用的数据结构如下:

```
typedef struct DNSHeader {
  uint16_t ID;
  uint16_t QR : 1;
  uint16_t Opcode : 4;
  uint16_t AA : 1;
  uint16_t TC : 1;
  uint16_t RD : 1;
  uint16_t RA : 1;
  uint16_t Z : 3;
  uint16_t RCODE : 4;
  uint16_t QDCOUNT;
  uint16_t ANCOUNT;
  uint16_t NSCOUNT;
  uint16_t ARCOUNT;
} DNSHeader;

typedef struct Question {
  uint8_t *QNAME;
  uint16_t QTYPE;
  uint16_t QCLASS;
} Question;

typedef struct ResourceRecord {
  char *NAME;
  uint16_t TYPE;
  uint16_t CLASS;
  uint32_t TTL;
  uint16_t RDLENGTH;
  uint8_t *RDATA;
} ResourceRecord;

struct Message {
  DNSHeader dnsheader;
  Question *question;
  ResourceRecord *answer;
  ResourceRecord *authority;
  ResourceRecord *additional;
  SOCKADDR_IN senderAddr;
} message;

typedef struct Entry {
  char IP[16];
```

```
  char *name;
} Entry;

struct NameTable {
  Entry nametable[MAX_TABLEENTRY];
  int size;
} nameTable;

typedef struct Record {
  SOCKADDR_IN senderAddr;
  uint16_t ID;
  char *name;
} Record;

struct RecordTable {
  Record recordtable[MAX_RECORD];
  int size;
} recordTable;
```

本模块中结构体与字符流之间的转换函数声明如下.

```
void get16bits(uint8_t **ptr, uint16_t *value);
void get32bits(uint8_t **ptr, uint32_t *value);
void ExtractHeader(uint8_t **ptr);
void extractName(uint8_t **ptr, char **name);
void ExtractQuestion(uint8_t **ptr);
void printName(uint8_t *name);
void ExtractRR(uint8_t **ptr);
void ExtractMessage();

char *transName(uint8_t *name);
void createAnswer(char *IP, char *name);
void put16bits(uint16_t value, uint8_t **ptr, int *buffersize);
void put32bits(uint16_t value, uint8_t **ptr, int *buffersize);
void constructHeader(uint8_t **ptr, int *buffersize);
void constructQuestion(uint8_t **ptr, int *buffersize);
void constructRR(uint8_t **ptr, int *buffersize);
int constructMessage(char *IP, char *name);
```

此模块不过是从字节流中抽取出相关信息赋值到结构体中,无需特别说明之处.

4.5 parser

`parser` 模块负责对接收的报文进行解析,并发送相应的包.

该模块使用的主要变量如下.

```
#define MAX_BUFFERSIZE 1024
uint8_t *buffer;
SOCKADDR_IN serverAddr;
struct Message {
  DNSHeader dnsheader;
  Question *question;
  ResourceRecord *answer;
  ResourceRecord *authority;
  ResourceRecord *additional;
  SOCKADDR_IN senderAddr;
} message;
time_t t;
```

其中,buffer 存储当前报文的字节流, MAX_BUFFERSIZE 表示 buffer 的大小,serverAddr 表示当前报文的发送方地址,t 表示收到当前报文的时间.message 表示当前报文的结构体.

该模块使用的主要函数如下:

```
void recvMessage();
void analyzeResponse(int bytes);
void sendAnswer(int recvbytes);
void sendToServer(int bytes);
```

其中,`main` 函数中一直调用 `recvMessage` 函数以解析各种信息.

```
  while (1) {
    recvMessage();
  }
```

根据报文类型进行进一步判断.

• REQUEST：若报文的 QR 和 OPCODE 字段均为 0,说明这是一条request.我们进入 `sendAnswer()` 部分. • RESPONSE：若报文的 QR 字段为 1,表示这是一条应答.我们进入 `analyzeResponse()` 部分. • OTHER：如果不是前两种报文,我们的当前 DNS 中继服务器是无法处理的,发往远端服务器.我们进入`sendToServer()`..

5. 测试样例

以下测试用例是在"将本地主机所对应的 DNS 服务器改成本地 IP"下进行的.

执行`nslookup`指令,开始访问 dns 服务器. `➜ ~ nslookup - 127.0.0.1`

5.1 不良网站拦截功能

首先是正常网络访问结果:

```
> www.baidu.com
Server:         127.0.0.1
Address:        127.0.0.1#53

Name:   www.baidu.com
Address: 39.156.66.14
```

在`dnsrelay.txt`中屏蔽京东官网后,再访问得到的结果为:

```
> www.jd.com
Server:         127.0.0.1
Address:        127.0.0.1#53

** server can't find www.jd.com: NXDOMAIN
```

我们又手动登入了一下京东官网,验证了京东今天确实还没倒闭,也说明了我们确实实现了不良网站蓝洁功能.

5.2 服务器功能

`dnsrelay.txt`中存在表项`39.156.69.79 baidu.com`.基于此,我们查询`baidu.com`得到结果如下:

```
> baidu.com
Server:         127.0.0.1
Address:        127.0.0.1#53

Name:   baidu.com
Address: 39.156.69.79
```

通过比对查询结果所获得的 IP 地址发现无误,我们成功实现了服务器功能

5.3 中继功能

要检测我们的服务器实现了中继功能,我们访问一个 `dnsrelay.txt` 中没有的网站. 我们查询 `leetcode.com`.通过请求远端服务器,我们成功获得了 IP 地址,说明我们成功实现了中继功能.

```
> leetcode.com
Server:         127.0.0.1
Address:        127.0.0.1#53

Non-authoritative answer:
Name:   leetcode.com
Address: 104.18.35.28
Name:   leetcode.com
Address: 104.18.34.28
```

5.4 缓存功能

再次查询`leetcode.com`,于 DNS 输出端口得到如下信息.

```
find the record in local name table
the message is sent to the server
Time: Wed Nov 20 16:37:58 2019
Count: 8
```

说明我们已成功缓存`leetcode.com`的 IP 地址.`nslookup`查询结果如下:

```
> leetcode.com
Server:         127.0.0.1
Address:        127.0.0.1#53

Name:   leetcode.com
Address: 104.18.35.28
```

查询 `dnsrelay.txt`,我们发现确实多了一条条目`104.18.35.28 leetcode.com`,说明我们成功实现了缓存功能.

6. 遇到的问题 & 心得

遇到的问题

处理并发

一开始没有理解题目中的要求"考虑多个计算机上的客户端会同时查询,需要进行消息ID的转换"中的后半句.单纯就是知道需要能够处理并发查询过程.于是自然而然就想到通过多线程来处理.但在多线程编程实现上我们卡了挺长的时间.

事实上"需要进行消息ID的转换"与其说是要求,不如说就是提示及实现方式.通过几日的思索,我们发现确实只需要在 DNS 服务器里对每个request 存储一个本地 ID (currentID),并用此 ID 替换原 ID 进行中继处理,就能绝对保证 ID 重复问题了.

一个神奇的 bug

我们在一个的 bug 上卡住了很久.因为这个 bug 的呈现方式实在过于诡谲.这个问题是在加入 currentID 后出现的.

具体表现如下:

在未加入 currentID 的版本中,一切能正常运行.但偶尔会出现 timeout 的情况. 在加入 currentID 的版本中,却发生了一定会出现一到两次 timeout 的异常.

一开始我们认为出现异常的原因是,在引入并发功能之后,暴露出了系统中某一部分隐藏的 bug.于是我们对系统各个部分都进行整体的排查,花费了很多很多时间都没有成功 debug.我们又对系统运行流程进行模拟,以及进行单步调试,也发现不了为何会出现这种 bug.

由于我们仅仅做了一个 currentID 的小改动,我们意识到问题不是非常严峻,就是一个粗心烦的错误而已.最后是通过逐行分析.发现是在`insertrecord()` 里面重header放在了插入记录前面.如此,无法将查询的结果送回发出 request的服务器.那为什么只发生一次后就成功了呢?是因为得到的结果已经加入`nameTable`中,没有进入中继功能直接返还了.

这个 bug 的意义在于,让我们了解了如何在一个动态的运行的系统下面进行 debug .

linux 环境与 win 环境之间的转换

由于我们小组各个电脑环境不同,一开始各自在linux 下和 win 下开始写.等到拼接讨论的时候发现除了套接字API不一样之外,还发现 windows 缺少挺多 linux 下自带的函数,如 optget 等.于是我们还手工实现了这些函数,花费了一点时间.

有关验收时提出的问题

验收时高老师提出了一个问题:除了用报文中的信息外,还有什么方式可以分辨请求报文和应答报文?

由于验收时时间紧迫,没有回答完整,这里做详细说明.以网络体系结构的思路进行分析,按分层思想作答即可.那么按层划分有:

0. 我们使用报文的内容进行分辨，可以看作是利用应用层的信息进行分辨。
1. 于传输层中按端口号分辨.由于是用 UDP,那么自然包括端口号信息.通过识别端口号即可分辨报文是来自服务端还是客户端.我们可以根据检测源端口号是否为 53 来检测报文是否来自远端服务器.我们也可以新开一个端口,用 54 端口专门发送请求报文,那么只要是 54 端口号接收的信息都将是来自远端服务器的报文.如此，53 收到的是请求报文，54收到的是默认DNS服务器的应答报文
2. 于网络层中按 IP 地址分辨.只要知道服务器 IP 地址便可以分辨.

心得

林杨

对于网络协议的学习,需要紧紧结合实践.若只是简单的记背,没有写过代码,没有亲自对报文信息进行解析,处理,那么整个学习过程便如无本之木一样.本次课程设计的整个过程,让我真正体会到了DNS报文中各个字段的含义,了解了DNS服务器的运作流程.这些认知,我相信会为未来的学习打下很好的基础.

从拿到实验题目开始,就觉得这个实验要做的东西并不很难,无非是中继转发+本地查询+成帧 发回这几个功能而已.但到了真正开始干的时候,却发现并没有这么简单.

我们很快地确定了代码框架,但是写起来的时候,在很多具体的实现上都遇到了一些麻烦,首先是网络字节流存储的逆序问题,在解析报文与构建报文时,要精确的算出包中每一个字段的具体位置,并配合位运算,提取出字节中对应比特的数据,实现起来十分费劲.

由于使用c语言编写程序,在编写过程中,由于涉及对buffer的解析与构建,需要大量的使用指针,这也常常导致程序出现例如非法访问内存之类的问题,而这类问题又常常难以定位,在调试程序的时候给我们带来了很大的麻烦.

在整个实验过程中,占据我们最多时间的是debug这个环节,由于涉及网络编程,许多问题难以定位,我们常常只能把出错的报文打印出来,逐位分析可能是程序的那一部分出了问题,例如,我们发现在添加了id转换功能之后,请求总是出现time out,但是还是可以正确的返回结果,最后发现是由于地址转换失败,第一次发出的请求没法转发回发出请求的服务器,但是得到的域名-IP记录进了cache中,超时重传后的查询请求直接在cache中找到了结果,不需要向外网的DNS服务器进行转发,所以能得到正常的结果.

本次课程设计让我深刻理解了DNS协议,我们完整地经历了框架设计--代码实现--调试debug--迭代更新这一整个程序开发流程.这个过程对我们代码能力的提升帮助很大.

王瑞

在本次课程设计中,我主要负责代码编写和调试的工作.一开始,我对于DNS的基本原理和套接字编程并不十分熟悉,因而查阅了大量的资料才明确了DNS解析的工作流程和代码的编写逻辑.

在明确了目标之后,代码的编写可以说是十分清晰和顺畅的,不过由于采用的是C语言,我们缺少了很多可用的函数接口和数据结构,需要自己编写这一部分的代码.另外,对于动态分配的内存,我们不可避免地需要处理很多指针相关的操作,手动进行内存的管理,这为编程和调试都带来的一定的复杂性.

在调试过程中,我们遇到了相当多的问题,最多的便是内存管理不慎而造成的内存的非法访问,此外我们还遇到了无法绑定套接字,RDATA数据域错误,DNS请求超时等问题.一个个看似简单的问题都花费了我们较长的时间去排查,而在解决问题的过程中,我对于套接字编程和DNS也有了更深的理解,对于C语言的指针操作也更加熟练. 这次课程设计给了我们一个很好的机会将课本学习的理论知识迁移到实践之中,通过编写代码和进行调试,我的工程实践能力也有了一定的提高.

范乾一

我是这篇文档的作者,我相信我的心得与思考已经在这篇文稿的字里行间有所体现.而最令我印象深刻的体验便是上面遇到的问题里所述的那样,即如何通过分析一个正在进行中的系统所反馈的信息进行 Debug.

De 了这么多 Bug,事实上已经有经验了.对于一些及其神奇的错误的,我有如下不成熟见解.第一便是,错误往往不是非常高神的机制所引发的,往往就是一些错行,错位造成的.第二是,Debug 的平均意义上最快的方法就是将所有可能的状态及其之间的转换关系构造出来,非常谨慎地捋一遍.如果代码量小,个人感觉更快的方法便是直接重写一遍,因为我认为重写是更严谨地更精确的分析,需要知道每一步具体代表的意义.

总而言之,通过本次课程设计,我深刻理解了DNS协议.这实现一个完整系统的过程对我代码能力有很大提升.

7. 参考

1. RFC1034
2. RFC1035
3. http://en.wikipedia.org/wiki/Domain_Name_System
4. socket 编程 (wiki pedia,百度百科...)
5. nslookup 手册
6. https://github.com/Poeroz