***project***

**Implementation of a DNS Relay**

**Course title:** *Implementation of a DNS Relay*

**Name:** *Chen Qiwen*

*Chu Jianhua*

*Chen Yiming*

**Date：***2017.7.3-2017.7.10*

1. **Overview**

**1.1 Basic requirements**

Our project is aimed at implementing a DNS relay which have two main functions: One is to receive DNS queries from DNS clients (or DNS resolvers) then forward them to a given DNS server; the other is to receive DNS responses from DNS servers then forward them to the resolver.

query

query

DNS Relay

DNS Server

Resolver

response

response

There are also three cases to meet:

case1 is for domain name including distributed database;

case2 is for IP address 0.0.0.0.;

case3 is for no domain in local DNS.

According to these three cases, our team think up different precepts to meet three cases. What’s more, for requirements, our team use socket in our program which builds, receives and sends DNS data packet.

**1.2 Target**

**Step 1: Understanding DNS**

Domain Name System:

A distributed database providing mapping between Domain name and IP address.

An application protocol used by hosts and name servers to communicate to resolve names. The DNS protocol runs on top of the UDP protocol, using the port number 53. Message formats reference RFC 1053.

**Step 2: Using Wireshark to understand the communications**

Wireshark: A network protocol analyzer (packet sniffer). It is able to capture packets transferred on the network and display packet fields and their meanings. Used for network troubleshooting, analysis, software and communications protocol development, and education.

**Step 3: using Socket programming**

Socket: An extension to I/O system of operating system, enabling communication between processes and machines. Socket interface is the API of TCP/IP network and used to build the communication on TCP/IP network.

**Step 4: Running the program**

Get the IP address of our DNS server and IP address. Configure DNS server as our IP address. And Running our program, setting local DNS server, then use ping, nslookup, web browser to test our program

**1.3. Implementation**

For the Windows operating system,

In cmd, execute the following command:

Java dnsrelay [-d | -dd] [dns-server-ipaddr] [filename]

For the Linux/ Mac OS operating system, under the super model

gcc –o dnsrelay desrelay.c

./dnsrelay

**2. Requirements Analysis**

**2.1 Development environment**

·The running environment: Windows 10 and Linux

·The programming language: java and C

·The compile tools: Dev C++ and Eclipse

Our team use two different language to implements the final project, we also implements these two language in two different environments: Windows and Linux.

1. **2 Functional requirements in details**

Our team design a DNS relay server program: the function is reading the domain name form resolver meanwhile sending back the right IP address. We only use UDP protocol in our program. When the client put into the domain which they want to search (using upper-look query the DNS relay).Our team also choose the correct ways to search the IP address.

Three cases:

·case1: If the domain name include in the database (our own DSN relay server).

→send back corresponding IP address by DNS relay server.

·case2: If the IP address is 0.0.0.0.

→send back no such name (reply code =0011) by DNS relay server.

·case3: If the domain name not include in the database.

→forward query to local DNS server, send back packet to outer DNS server for research and do research again by DNS relay server.

* 1. **Task decomposition and analysis**

Our team divided the whole project into three indispensable tasks: local database operation, DNS relay operations and DNS server operations.

·local database operation

First, query’s domain should be determined whether it is in the local database. The local database will store IP address and domain (the domain names have been selected) be selected when a query has been sent from the resolver.

·DNS relay

DNS relay is core task of the whole project. If the query of domain includes in distributed database and IP address is valid, it will return the IP address and send the result to the resolver. But when the query of domain includes in database but the IP address is 0.0.0.0., it will return an error message and send it to the resolver. If the query of domain does not include in database, it will send the query to DNS server for further operation. In conclusion, According to these three cases, our team think up different precepts to meet three cases.

·DNS server

Our team use DNS server to solve the problem that the query whose domain cannot be found in the local database and operates on further query. When the DNS server finds out the corresponding IP address, it should send the query to the DNS relay.

1. **Preliminary Design**
   1. **Decomposition of functional modules**

Our program has main five functional modules: initialization module, receive module, search module, upload module, respond module.

·Initialization Module

When the program is going to run, the program will bind to port 53, and it will read local DNS file, store the content into memory, and start to receive the packets which are send to it.

·Receive Module

When the data is coming, putting data into a specific location as the format, and build a analysis function to judge the packet is a respond packet or a query packet, for the next step to process it.

·Search Module

When the command happens, the program can receive a data packet in DNS standard which includes the website that the client wants to search. Command response module consider those requests by extracting the target website and compare it with local cache. If we can find it, we will send it to the resolver. Otherwise, we will send to query to the resolver.

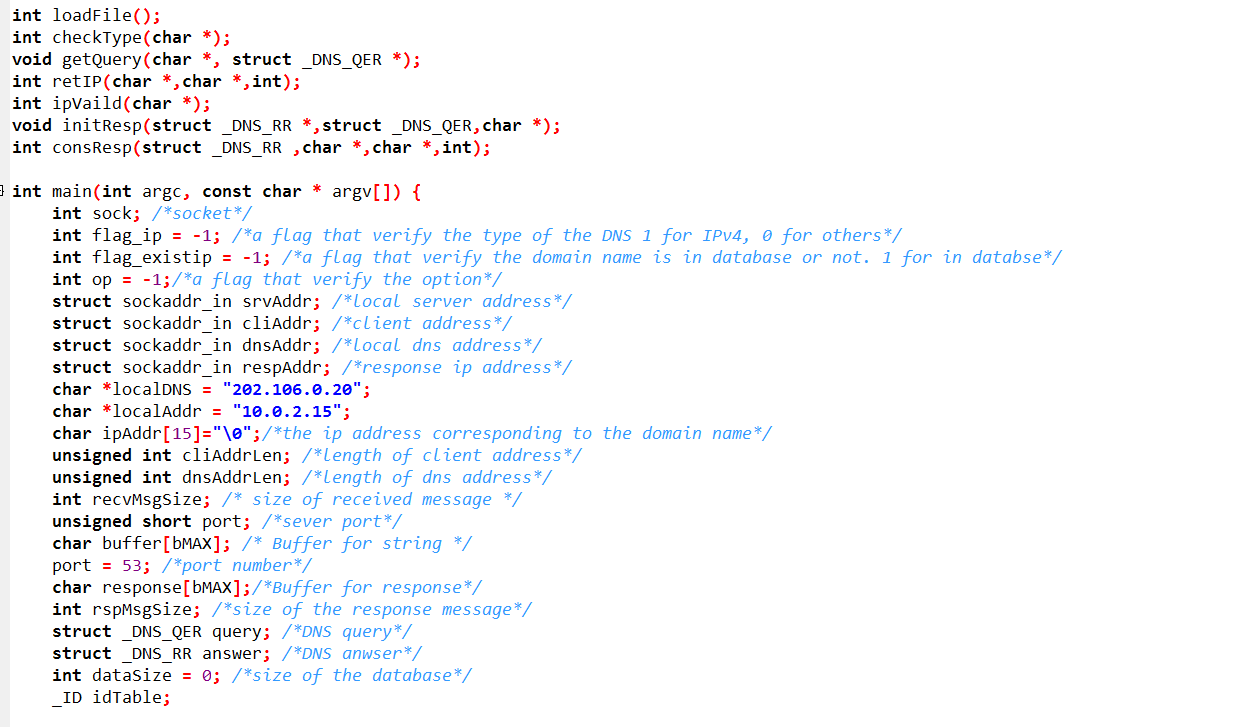
·Upload Module

When the local cache doesn’t have the target website address, the delay sends it to external resolver. It records remain packers ID, header ID and sends query packet to external server.

·Respond Module

When the local cache has the target website address, it gets the location of the saving address of the target website in local cache from the command response. It can read the IP address from the cache, it constructs a standard DNS response package with the target website IP address and sends it back to client.

* 1. **Design of data structures**

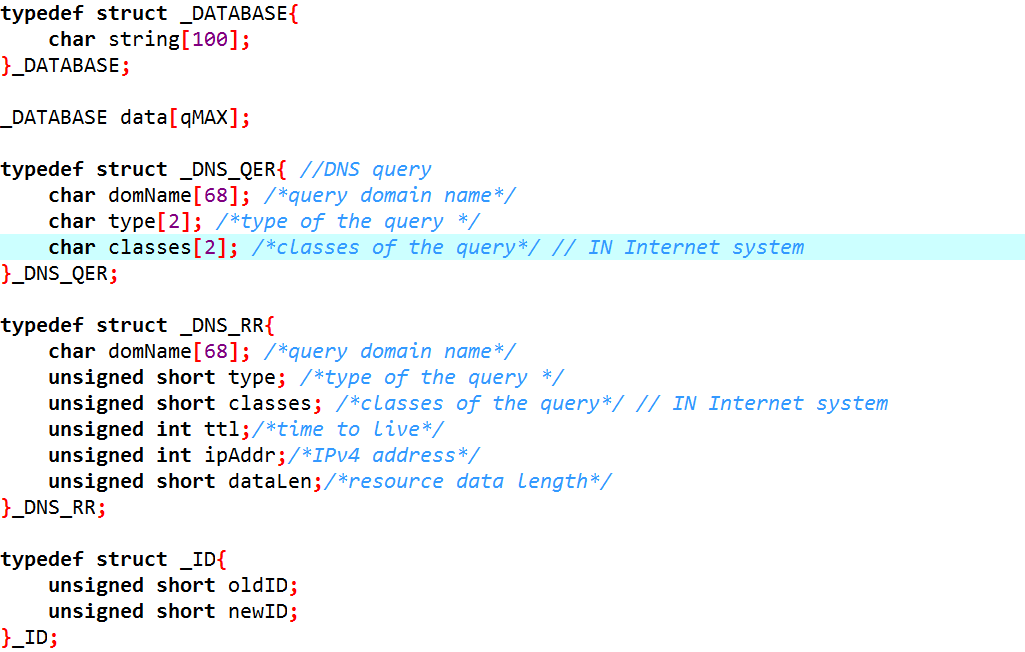


The structure of UDP protocol can be seen in the following, above codes show all the relevant data structure defined in the type of C program.

**4 Detailed Design**

**4.1 Initialization Module**

A. First, define the following struct types according to the UDP protocol:

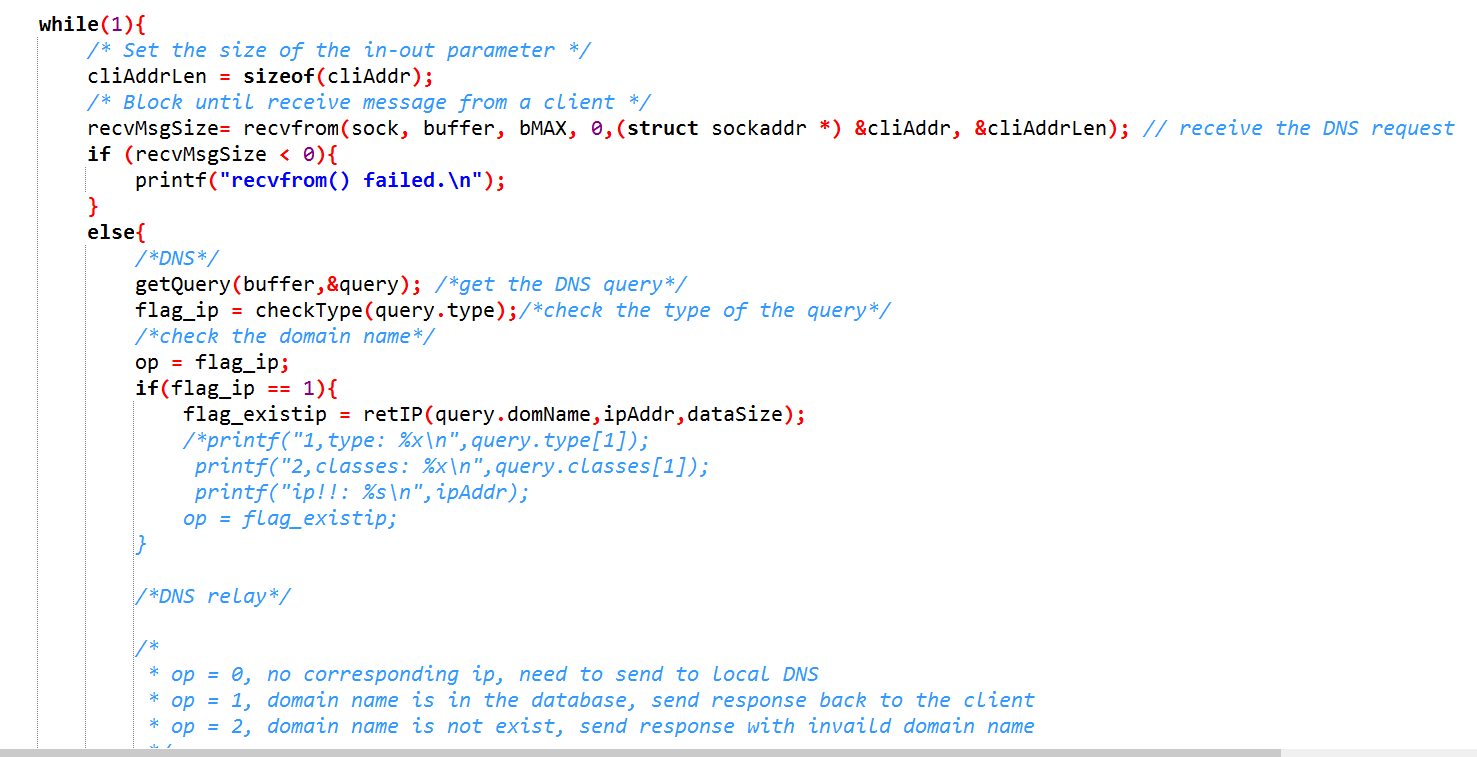


B. Initialize Socket, linking to port 53 which is represented by predefined parameter



**4.2 Receive Module**

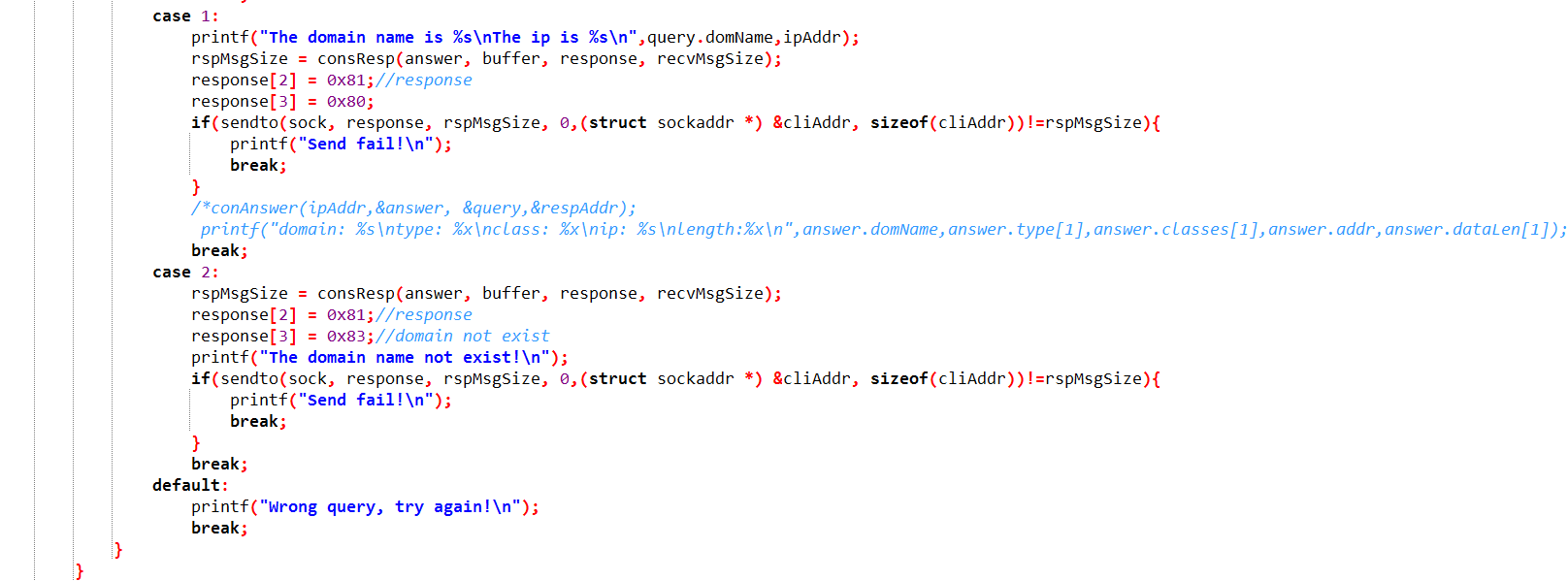
Permanently listening to the interface and receive the relevant data, then choose the case the packet belongs to:



Case0:

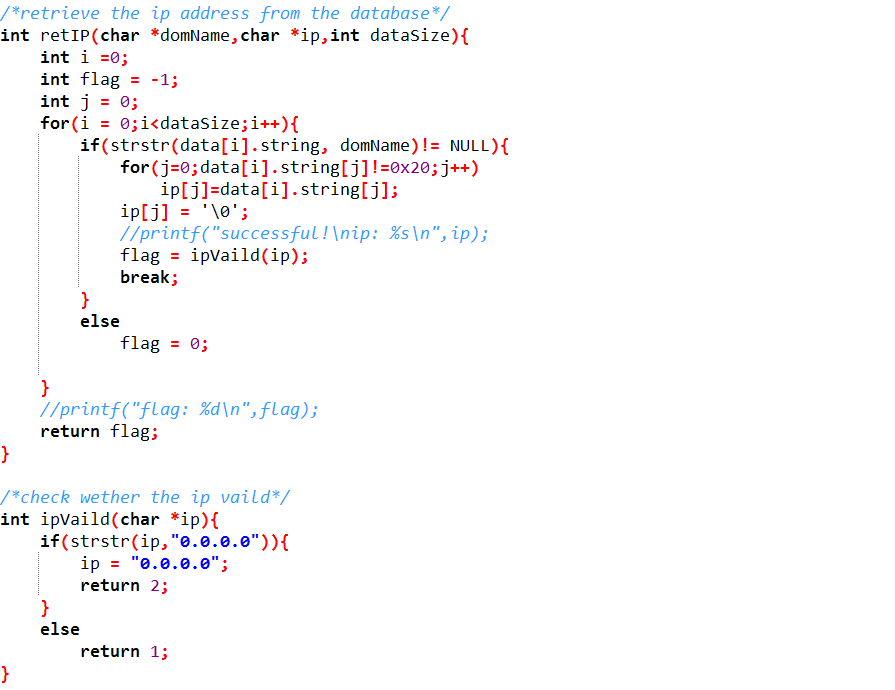


Case 1 and Case 2:



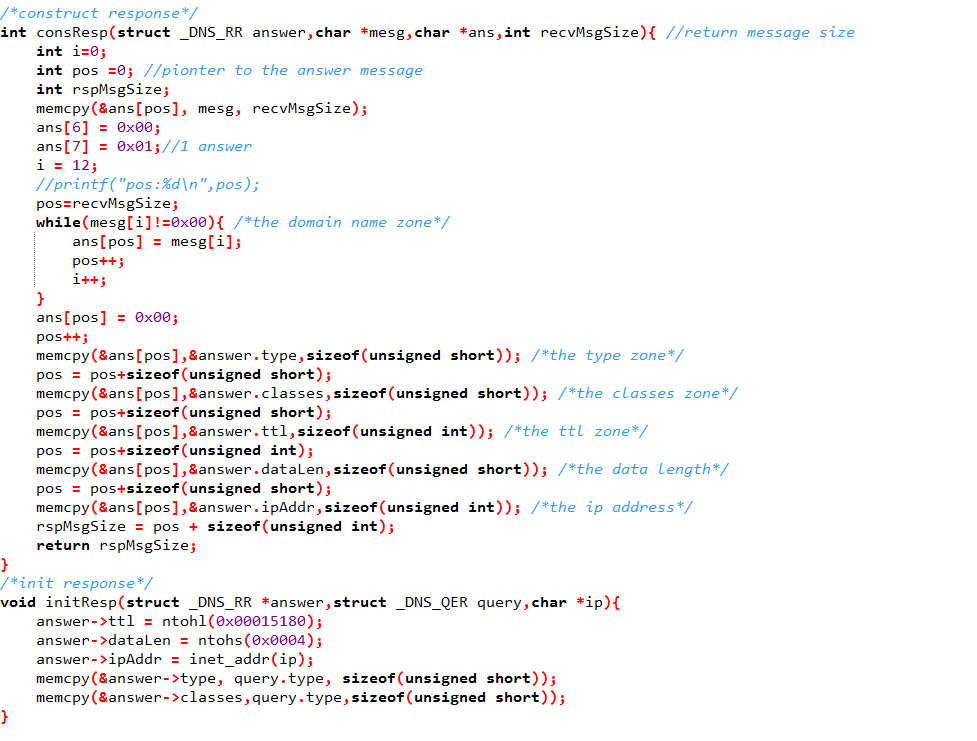
**4.3 Search Module**

Every time receiving a request from the resolvers, the relay part will search the database (DNSRelay.txt) to determine the existence and type of the relevant domain name:



**4.4 Respond Module**

Initialize and construct the respond to the relevant resolver:



**5. Results**

Step 1: Modify the local host to “127.0.0.1” on network service.

In Windows



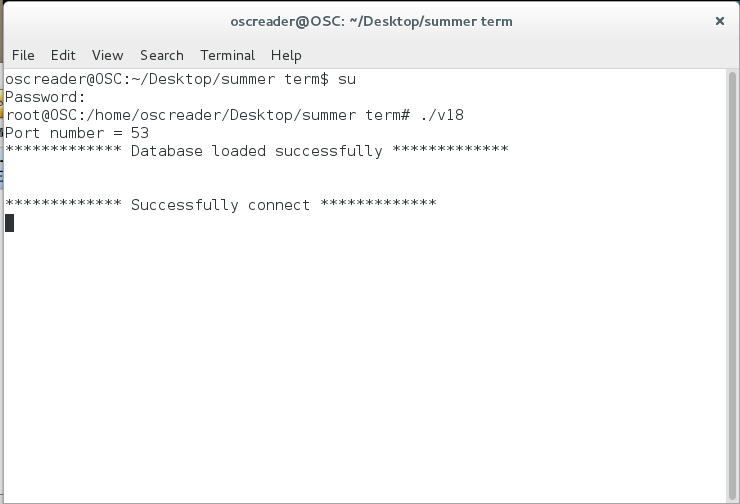
In Linux

In command line entering

vi /etc/resolv.conf

changing the dns address

Step 2: Run the C program.

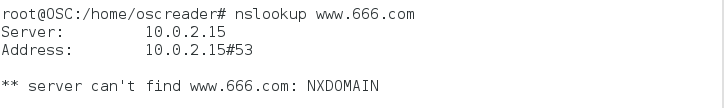


Step 3: Examine the validation of the program by checking different domain input. According to the requirement, we consider three cases.

Case 1: The query of the domain exists in the database and the corresponding IP address is valid.



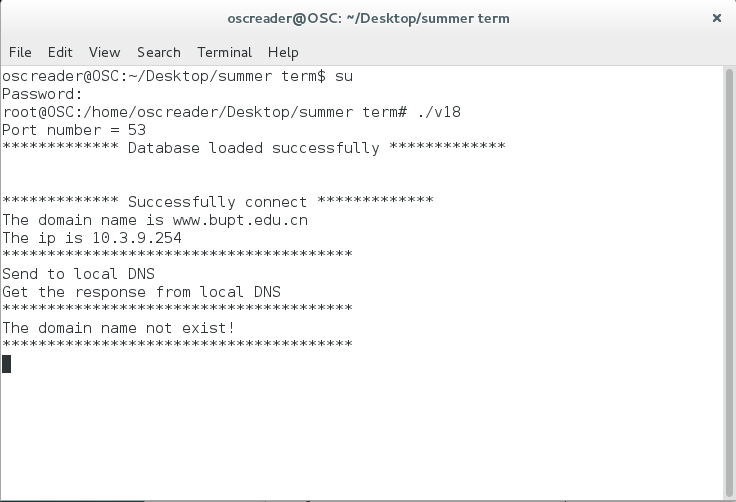
Case 2: The query of the domain is exist in the database but the IP address is 0.0.0.0 which is invalid.



Case 3: The query of the domain doesn’t exist in the database.



Then all the processes are recorded in the initial command line:



**6 Summary and Conclusion**

Qiwen Chen

Responsibilities: I have read many documents about DNS, then thought carefully about these whole project structures, and I communicated with Jianhua frequently to divide this whole structure into several modules. I did the main part of C programming.

Self-evaluation:

In this process, I went through lots of websites to get relevant knowledge of DNS, furthermore, I learned many API and practiced my C programming skills. To be honest, as I read many websites about DNS relay and I thought about them carefully, I got a clear understanding about DNS working mechanism.

Jianhua Chu

Responsibilities: I have read the PPT given to us for many times to obtain knowledge about DNS, I think the PPT is detailed and I learned a lot from it. In addition, I finished the Java programming. Furthermore, I and Yiming learned to test the program and analysis using wireshark with the help from Qiwen.

Self-evaluation:

As I read the PPT for many times, and I communicated with Qiwen and discussed some questions about DNS, I thought Qiwen has a great understanding about it and I learned many from her. Apart from it, I practiced skills of java programming.

Yiming Chen

Responsibilities: I learned how to use wireshark and how to test the program with the help from my teammates. Besides, I programmed part of Initialization Module.

Self-evaluation:

I learned something about how to analysis the command arguments and I practiced java programming. I read the PPT, and with the help from my teammates, I obtained some knowledge about DNS.

**Appendix: Source Codes**

**C:**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <string.h>**

**#include <unistd.h>**

**#include <sys/socket.h> /\* socket(),sendto(),recvfrom()\*/**

**#include <arpa/inet.h> /\* sockaddr\_in and inet\_addr() \*/**

**#define bMAX 512 /\* longest string to DNS request \*/**

**#define qMAX 300**

**typedef struct \_DATABASE{**

**char string[100];**

**}\_DATABASE;**

**\_DATABASE data[qMAX];**

**typedef struct \_DNS\_QER{ //DNS query**

**char domName[68]; /\*query domain name\*/**

**char type[2]; /\*type of the query \*/ //**

**char classes[2]; /\*classes of the query\*/ // IN Internet system**

**}\_DNS\_QER;**

**typedef struct \_DNS\_RR{**

**char domName[68]; /\*query domain name\*/**

**unsigned short type; /\*type of the query \*/**

**unsigned short classes; /\*classes of the query\*/ // IN Internet system**

**unsigned int ttl;/\*time to live\*/**

**unsigned int ipAddr;/\*IPv4 address\*/**

**unsigned short dataLen;/\*resource data length\*/**

**}\_DNS\_RR;**

**typedef struct \_ID{**

**unsigned short oldID;**

**unsigned short newID;**

**}\_ID;**

**int loadFile();**

**int checkType(char \*);**

**void getQuery(char \*, struct \_DNS\_QER \*);**

**int retIP(char \*,char \*,int);**

**int ipVaild(char \*);**

**void initResp(struct \_DNS\_RR \*,struct \_DNS\_QER,char \*);**

**int consResp(struct \_DNS\_RR ,char \*,char \*,int);**

**int main(int argc, const char \* argv[]) {**

**int sock; /\*socket\*/**

**int flag\_ip = -1; /\*a flag that verify the type of the DNS 1 for IPv4, 0 for others\*/**

**int flag\_existip = -1; /\*a flag that verify the domain name is in database or not. 1 for in databse\*/**

**int op = -1;/\*a flag that verify the option\*/**

**struct sockaddr\_in srvAddr; /\*local server address\*/**

**struct sockaddr\_in cliAddr; /\*client address\*/**

**struct sockaddr\_in dnsAddr; /\*local dns address\*/**

**struct sockaddr\_in respAddr; /\*response ip address\*/**

**char \*localDNS = "202.106.0.20";**

**char \*localAddr = "10.0.2.15";**

**char ipAddr[15]="\0";/\*the ip address corresponding to the domain name\*/**

**unsigned int cliAddrLen; /\*length of client address\*/**

**unsigned int dnsAddrLen; /\*length of dns address\*/**

**int recvMsgSize; /\* size of received message \*/**

**unsigned short port; /\*sever port\*/**

**char buffer[bMAX]; /\* Buffer for string \*/**

**port = 53; /\*port number\*/**

**char response[bMAX];/\*Buffer for response\*/**

**int rspMsgSize; /\*size of the response message\*/**

**struct \_DNS\_QER query; /\*DNS query\*/**

**struct \_DNS\_RR answer; /\*DNS anwser\*/**

**int dataSize = 0; /\*size of the database\*/**

**\_ID idTable;**

**printf("Port number = %d\n",port);**

**/\*load the local DNS database\*/**

**dataSize = loadFile();**

**/\* test**

**int l = 0;**

**for(l =0;l<dataSize;l++){**

**printf("%d!: %s\n",l,data[l].string);**

**} end of test\*/**

**/\*create socket\*/**

**sock = socket(PF\_INET,SOCK\_DGRAM,0);**

**if(sock < 0){**

**printf("socket() failed!");**

**exit(1);**

**}**

**/\*construct local address structure\*/**

**memset(&srvAddr, 0, sizeof(srvAddr));**

**srvAddr.sin\_family = AF\_INET;**

**srvAddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);**

**srvAddr.sin\_port =htons(port);**

**/\*construct response address structure\*/**

**memset(&respAddr, 0, sizeof(respAddr));**

**respAddr.sin\_family = AF\_INET;**

**respAddr.sin\_addr.s\_addr = inet\_addr(localAddr);**

**respAddr.sin\_port = htons(port);**

**/\*construct the loca DNS address structure\*/**

**memset(&dnsAddr, 0, sizeof(dnsAddr));**

**dnsAddr.sin\_family = AF\_INET;**

**dnsAddr.sin\_addr.s\_addr = inet\_addr(localDNS);**

**dnsAddr.sin\_port =htons(port);**

**/\*bind to the local address\*/**

**if ((bind(sock, (struct sockaddr \*) &srvAddr, sizeof(srvAddr))) < 0){**

**printf("bind() failed.\n");**

**exit(1);**

**}**

**printf("\*\*\*\*\*\*\*\*\*\*\*\*\* Successfully connect \*\*\*\*\*\*\*\*\*\*\*\*\*\n");**

**while(1){**

**/\* Set the size of the in-out parameter \*/**

**cliAddrLen = sizeof(cliAddr);**

**/\* Block until receive message from a client \*/**

**recvMsgSize= recvfrom(sock, buffer, bMAX, 0,(struct sockaddr \*) &cliAddr, &cliAddrLen); // receive the DNS request**

**if (recvMsgSize < 0){**

**printf("recvfrom() failed.\n");**

**}**

**else{**

**/\*DNS\*/**

**getQuery(buffer,&query); /\*get the DNS query\*/**

**flag\_ip = checkType(query.type);/\*check the type of the query\*/**

**/\*check the domain name\*/**

**op = flag\_ip;**

**if(flag\_ip == 1){**

**flag\_existip = retIP(query.domName,ipAddr,dataSize);**

**/\*printf("1,type: %x\n",query.type[1]);**

**printf("2,classes: %x\n",query.classes[1]);**

**printf("ip!!: %s\n",ipAddr);**

**op = flag\_existip;**

**}**

**/\*DNS relay\*/**

**/\***

**\* op = 0, no corresponding ip, need to send to local DNS**

**\* op = 1, domain name is in the database, send response back to the client**

**\* op = 2, domain name is not exist, send response with invaild domain name**

**\*/**

**initResp(&answer, query,ipAddr);**

**switch (op) {**

**case 0:**

**memcpy(&idTable.oldID , buffer, sizeof(unsigned short));**

**idTable.newID = idTable.oldID + 1;**

**memcpy(response, buffer, recvMsgSize);**

**printf("Send to local DNS\n");**

**/\*retransmit the DNS request to the local DNS\*/**

**memcpy(&response[0], &idTable.newID, sizeof(unsigned short));**

**rspMsgSize = recvMsgSize;**

**if(sendto(sock, response, rspMsgSize, 0,(struct sockaddr \*) &dnsAddr, sizeof(dnsAddr))!=rspMsgSize){**

**printf("Send fail!\n");**

**break;**

**}**

**/\* Set the size of the in-out parameter \*/**

**dnsAddrLen = sizeof(dnsAddr);**

**/\*block until receieve the response from the local DNS\*/**

**recvMsgSize= recvfrom(sock, buffer, bMAX, 0,(struct sockaddr \*) &dnsAddr, &dnsAddrLen); //receive the DNS response**

**if (recvMsgSize < 0){**

**printf("recvfrom() failed.\n");**

**break;**

**}**

**printf("Get the response from local DNS\n");**

**memcpy(response, buffer, recvMsgSize);**

**memcpy(&response[0], &idTable.oldID, sizeof(unsigned short));**

**rspMsgSize = recvMsgSize;**

**/\*send the response to the local client\*/**

**if(sendto(sock,response, rspMsgSize, 0,(struct sockaddr \*) &cliAddr, sizeof(cliAddr))!=rspMsgSize){**

**printf("Send fail!\n");**

**break;**

**}**

**break;**

**case 1:**

**printf("The domain name is %s\nThe ip is %s\n",query.domName,ipAddr);**

**rspMsgSize = consResp(answer, buffer, response, recvMsgSize);**

**response[2] = 0x81;//response**

**response[3] = 0x80;**

**if(sendto(sock, response, rspMsgSize, 0,(struct sockaddr \*) &cliAddr, sizeof(cliAddr))!=rspMsgSize){**

**printf("Send fail!\n");**

**break;**

**}**

**/\*conAnswer(ipAddr,&answer, &query,&respAddr);**

**printf("domain: %s\ntype: %x\nclass: %x\nip: %s\nlength:%x\n",answer.domName,answer.type[1],answer.classes[1],answer.addr,answer.dataLen[1]);**

**break;**

**case 2:**

**rspMsgSize = consResp(answer, buffer, response, recvMsgSize);**

**response[2] = 0x81;//response**

**response[3] = 0x83;//domain not exist**

**printf("The domain name not exist!\n");**

**if(sendto(sock, response, rspMsgSize, 0,(struct sockaddr \*) &cliAddr, sizeof(cliAddr))!=rspMsgSize){**

**printf("Send fail!\n");**

**break;**

**}**

**break;**

**default:**

**printf("Wrong query, try again!\n");**

**break;**

**}**

**}**

**printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");**

**//printf("From client %s:%s\n",inet\_ntoa(cliAddr.sin\_addr),buffer); //print the DNS request**

**}**

**}**

**/\*construct response\*/**

**int consResp(struct \_DNS\_RR answer,char \*mesg,char \*ans,int recvMsgSize){ //return message size**

**int i=0;**

**int pos =0; //pionter to the answer message**

**int rspMsgSize;**

**memcpy(&ans[pos], mesg, recvMsgSize);**

**ans[6] = 0x00;**

**ans[7] = 0x01;//1 answer**

**i = 12;**

**//printf("pos:%d\n",pos);**

**pos=recvMsgSize;**

**while(mesg[i]!=0x00){ /\*the domain name zone\*/**

**ans[pos] = mesg[i];**

**pos++;**

**i++;**

**}**

**ans[pos] = 0x00;**

**pos++;**

**memcpy(&ans[pos],&answer.type,sizeof(unsigned short)); /\*the type zone\*/**

**pos = pos+sizeof(unsigned short);**

**memcpy(&ans[pos],&answer.classes,sizeof(unsigned short)); /\*the classes zone\*/**

**pos = pos+sizeof(unsigned short);**

**memcpy(&ans[pos],&answer.ttl,sizeof(unsigned int)); /\*the ttl zone\*/**

**pos = pos+sizeof(unsigned int);**

**memcpy(&ans[pos],&answer.dataLen,sizeof(unsigned short)); /\*the data length\*/**

**pos = pos+sizeof(unsigned short);**

**memcpy(&ans[pos],&answer.ipAddr,sizeof(unsigned int)); /\*the ip address\*/**

**rspMsgSize = pos + sizeof(unsigned int);**

**return rspMsgSize;**

**}**

**/\*init response\*/**

**void initResp(struct \_DNS\_RR \*answer,struct \_DNS\_QER query,char \*ip){**

**answer->ttl = ntohl(0x00015180);**

**answer->dataLen = ntohs(0x0004);**

**answer->ipAddr = inet\_addr(ip);**

**memcpy(&answer->type, query.type, sizeof(unsigned short));**

**memcpy(&answer->classes,query.type,sizeof(unsigned short));**

**}**

**/\*get query form the DNS request\*/**

**void getQuery(char \*mesg, struct \_DNS\_QER \*query){**

**int qpos = 12; // query pointer, the DNS header occupied 12 bytes, then is the DNS query**

**char name[68];**

**int i = 0;**

**while(mesg[qpos] != 0x00){ /\*get the domain name\*/**

**if(mesg[qpos]== 0x2D || (mesg[qpos]>=0x30 && mesg[qpos]<=0x39 )||(mesg[qpos]>=0x41 && mesg[qpos]<=0x7A ))**

**name[i]=mesg[qpos];**

**else**

**name[i]= 0x2E;**

**qpos++;**

**i++;**

**}**

**name[i] = '\0';**

**memcpy(query->domName, &name[1], strlen(name));**

**query->domName[i] = '\0';**

**//printf("domain name :%s\n",query->domName);**

**/\*get the type of the query\*/**

**memcpy(query->type, &mesg[++qpos], sizeof(unsigned short));**

**qpos = qpos + sizeof(unsigned short);**

**/\*get the classes of the query\*/**

**memcpy(query->classes, &mesg[qpos], sizeof(unsigned short));**

**}**

**/\*check the type of the DNS request\*/**

**int checkType(char \*type){**

**int op = 0;**

**if(type[0]==0x00&&type[1]==0x01)**

**op = 1; //IPv4**

**return op;**

**}**

**/\*load the local DNS database\*/**

**int loadFile(){**

**FILE \*file;**

**int i=0;**

**file = fopen("DNSrelay.txt", "r");**

**if(!file){ //check whether the file open succesffully**

**printf("Database open fail");**

**exit(1);**

**}**

**while(!feof(file)){**

**fgets(data[i].string,100,file); //read on line from the database**

**if(strlen(data[i].string)>=1){**

**data[i].string[strlen(data[i].string)-1] = '\0';**

**}**

**i++;**

**}**

**printf("\*\*\*\*\*\*\*\*\*\*\*\*\* Database loaded successfully \*\*\*\*\*\*\*\*\*\*\*\*\*\n\n\n");**

**return (i-1);**

**}**

**/\*retrieve the ip address from the database\*/**

**int retIP(char \*domName,char \*ip,int dataSize){**

**int i =0;**

**int flag = -1;**

**int j = 0;**

**for(i = 0;i<dataSize;i++){**

**if(strstr(data[i].string, domName)!= NULL){**

**for(j=0;data[i].string[j]!=0x20;j++)**

**ip[j]=data[i].string[j];**

**ip[j] = '\0';**

**//printf("successful!\nip: %s\n",ip);**

**flag = ipVaild(ip);**

**break;**

**}**

**else**

**flag = 0;**

**}**

**//printf("flag: %d\n",flag);**

**return flag;**

**}**

**/\*check wether the ip vaild\*/**

**int ipVaild(char \*ip){**

**if(strstr(ip,"0.0.0.0")){**

**ip = "0.0.0.0";**

**return 2;**

**}**

**else**

**return 1;**

**}**

**Java:**

package DNSRelay;

import java.io.BufferedReader;

import java.io.File;

import java.io.FileInputStream;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.HashMap;

import java.util.Map;

public class Check {

public static Map<String, String> ipTable = new HashMap<String, String>();

public static long fileSize;

public static void readData(String path) throws IOException

{

String line = "";

File f = new File(path);

fileSize = f.length();

if(!f.exists())

{

System.out.println("File doesn\'t exist!");

}

else

{

FileInputStream fis = new FileInputStream(f);

InputStreamReader isr = new InputStreamReader(fis);

BufferedReader br = new BufferedReader(isr);

while((line=br.readLine())!=null)

{

String[] ip = line.split(" ");

String ipAddress = ip[0];

String ipDomainName = ip[1];

ipTable.put(ipDomainName,ipAddress);

}

fis.close();

isr.close();

br.close();

}

}

}

**package** DNSRelay;

**public** **class** Convert {

/\*\*

\* Byte to Integer

\*/

**public** **static** **int** byte2Int(**byte**[] array, **int** start)

{

**final** **int** length = 1;

**int** result = 0;

**byte** loop;

**for** (**int** i = start; i < start + length; i++) {

loop = array[i];

**int** offSet = length - (i - start) -1;

result += (loop & 0xFF) << (8 \* offSet);

}

**return** result;

}

/\*\*

\* Integer to Byte

\* **@param** value Integer

\* **@param** array Byte array

\* **@param** start Start index in array

\* **@return** Offset

\*/

**public** **static** **int** int2Byte(**int** value, **byte**[] array, **int** start) {

**final** **int** length = 4;

**byte** loop;

**for** (**int** i = start; i < start + length; i++) {

**int** offSet = length - (i - start) -1;

loop = (**byte**) ((**byte**) (value >> (8 \* offSet)) & 0xFF);

array[i] = loop;

}

**return** length;

}

/\*\*

\* Integer to Byte

\* **@param** value Integer

\* **@return** byte[]

\*/

**public** **static** **byte**[] int2Byte(**int** value) {

**byte**[] array = **new** **byte**[4];

*int2Byte*(value, array, 0);

**return** array;

}

/\*\*

\* byte2Short

\*/

**public** **static** **short** byte2Short(**byte**[] array, **int** start)

{

**final** **int** length = 2;

**short** result = 0;

**byte** loop;

**for** (**int** i = start; i < start + length; i++) {

loop = array[i];

**int** offSet = length - (i - start) -1; //(i - start);

result += (loop & 0xFF) << (8 \* offSet);

}

**return** result;

}

/\*\*

\* Short to Byte

\* **@param** value Short

\* **@param** array Byte array

\* **@param** start Start index in array

\* **@return** Offset

\*/

**public** **static** **int** short2Byte(**short** value, **byte**[] array, **int** start) {

**final** **int** length = 2;

**byte** loop;

**for** (**int** i = start; i < start + length; i++) {

**int** offSet = length - (i - start) -1;

loop = (**byte**) ((**byte**) (value >> (8 \* offSet)) & 0xFF);

array[i] = loop;

}

**return** length;

}

/\*\*

\* Short to Byte

\* **@param** value Short

\* **@return** Byte[]

\*/

**public** **static** **byte**[] short2Byte(**short** value) {

**byte**[] array = **new** **byte**[2];

*short2Byte*(value, array, 0);

**return** array;

}

/\*\*

\* byte2Long

\*/

**public** **static** **long** byte2Long(**byte**[] b, **int** start){

**long** num = 0;

**for** (**int** ix = start; ix < start+8; ++ix) {

num <<= 8;

num |= (b[ix] & 0xff);

}

**return** num;

// long num = 0;

// for (int ix = start + 8 - 1; ix >= start+0; --ix) {

// num <<= 8;

// num |= (b[ix] & 0xff);

// }

// return num;

}

/\*\*

\* byte2Double

\*/

**public** **static** **double** byte2Double(**byte**[] b, **int** start){

**long** l = Convert.*byte2Long*(b, start);

**return** Double.*longBitsToDouble*(l);

}

/\*\*

\* **@param**

\* **@param**

\* **@return**

\*/

**public** **static** **float** byte2Float(**byte**[] b, **int** start) {

**int** i = Convert.*byte2Int*(b, start);

**return** Float.*intBitsToFloat*(i);

}

/\*\*

\* **@param**

\* **@param**

\* **@param**

\* **@return**

\*/

**public** **static** **char**[] byte2Char(**byte** [] b, **int** start, **int** length) {

**char** [] c = **new** **char**[length];

**for**(**int** i = 0; i < length; i++) {

c[i] = (**char**) b[start + i];

}

**return** c;

}

/\*\*

\* **@param**

\* **@param**

\* **@param**

\* **@param**

\* **@return**

\*/

**public** **static** **int** char2Byte(**char**[] c, **byte** [] b, **int** start, **int** length) {

**for**(**int** i = 0; i < length; i++) {

b[start + i] = (**byte**) c[i];

}

**return** length;

}

/\*\*

\* **@param**

\* **@param**

\* **@param**

\* **@return**

\*/

**public** **static** String byte2String(**byte** [] b, **int** start, **int** length) {

**return** String.*valueOf*(*byte2Char*(b, start, length));

}

/\*\*

\* **@param**

\* **@param**

\* **@param**

\* **@param**

\* **@return**

\*/

**public** **static** **int** string2Byte(String s, **byte** [] b, **int** start)

{

**if** (s == **null**) {

**return** 0;

}

**else** **if** (s.length() <= 0) {

**return** 0;

}

**else** {

**char** [] c = s.toCharArray();

**return** *char2Byte*(c, b, start, c.length);

}

}

/\*\*

\* **@param** source

\* **@param** start

\* **@param** length

\* **@return**

\*/

**public** **static** **byte** checkSum(**byte** [] source, **int** start, **int** length) {

**byte** checkSum = 0;

**for** (**int** i = start; i < start + length; i++) {

checkSum = (**byte**) (checkSum ^ source[i]);

}

**return** checkSum;

}

}

package DNSRelay;

import java.io.IOException;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

import java.net.UnknownHostException;

import java.text.SimpleDateFormat;

import java.util.HashMap;

import java.util.Map;

public class DNSRelay {

// DNS ip address and port

public static final String DNS\_IP = "202.106.0.20";

public static final int DNS\_PORT = 53;

// local port

public static final int LOCAL\_PORT = 53;

// maximum packet size

private static final int DATA\_LEN = 4096;

byte[] inBuff = new byte[DATA\_LEN];

// receive pacekt

private DatagramPacket inPacket = new DatagramPacket(inBuff, inBuff.length);

// relay packet

private DatagramPacket outPacket;

// analysis domain name

private String domainNameStr;

// resolver ip address and port

private InetAddress resolverAddress;

private int resolverPort;

// ipv6 flag

private boolean IPv6\_Flag = false;

// cursor that represents the current location to be resolved

int udpCursor;

// <key, value> = <packet.id, packet.socket>

private Map<Integer, IDTransition> idMap = new HashMap<Integer, IDTransition>();

SimpleDateFormat time=new SimpleDateFormat("HH:mm:ss");

public String getDomainName(byte[] buf) {

// domain name

String domainName = "";

udpCursor = 12;

// length

int length = Convert.byte2Int(buf, udpCursor);

while (length != 0) {

udpCursor++;

domainName = domainName

+ Convert.byte2String(buf, udpCursor, length) + ".";

udpCursor += length;

length = Convert.byte2Int(buf, udpCursor);

}

udpCursor++;

if (buf[udpCursor] == 0x00 && buf[udpCursor + 1] == 0x1c) {

IPv6\_Flag = true;

}

udpCursor += 4;

// delete the last '.'

return domainName.substring(0, domainName.length() - 1);

}

public void init() {

DatagramSocket socket = null;

try {

// bind to port 53

socket = new DatagramSocket(LOCAL\_PORT);

// keep listening

while (true) {

// receive pacet

socket.receive(inPacket);

// obtain data in the pacet

byte[] sendData = inPacket.getData();

// judge the type

if (((sendData[2] & 0x80) == 0x00)) { // query

System.out.println("\nrecieve time： " + new java.util.Date());

// get the domain name

domainNameStr = getDomainName(sendData);

System.out.println("domain name: " + domainNameStr);

// store the address and port of resolver

resolverAddress = inPacket.getAddress();

resolverPort = inPacket.getPort();

// if find

if (Check.ipTable.containsKey(domainNameStr)) {

// obtain the relevant ip address of the domain name

String LocalDNSipAddress = Check.ipTable

.get(domainNameStr);

if (LocalDNSipAddress.equals("0.0.0.0")) {// if IP = 0.0.0.0

System.out.println("function:" + "shield"); // code = 0011

// response (flag=0x8183)

sendData[2] = (byte) (sendData[2] | 0x81);

sendData[3] = (byte) (sendData[3] | 0x83);

// send data packet

outPacket = new DatagramPacket(sendData,

sendData.length, resolverAddress,

resolverPort);

socket.send(outPacket);

IPv6\_Flag = false;

} else {// if not 0.0.0.0 response with no error

// new packet

byte[] finalData = new byte[udpCursor + 16];

int cur = 0; // answer cursor

if (IPv6\_Flag) {// if ipv6

System.out.println("function:" + "IPV6 local response");

// response (flag=0x8180)

// Authority count = 1

sendData[2] = (byte) (sendData[2] | 0x81);

sendData[3] = (byte) (sendData[3] | 0x80);

sendData[8] = (byte) (sendData[8] | 0x00);

sendData[9] = (byte) (sendData[9] | 0x01);

System.arraycopy(sendData, 0, finalData, cur,

udpCursor);

IPv6\_Flag = false;

} else {// if IPv4

System.out.println("function:" + "IPV4 local response");

// response (flag=0x8180)

// Answer count = 1

sendData[2] = (byte) (sendData[2] | 0x81);

sendData[3] = (byte) (sendData[3] | 0x80);

sendData[6] = (byte) (sendData[6] | 0x00);

sendData[7] = (byte) (sendData[7] | 0x01);

System.arraycopy(sendData, 0, finalData, cur,

udpCursor);

// save name

cur += udpCursor;

short name = (short) 0xc00c;

System.arraycopy(Convert.short2Byte(name), 0,

finalData, cur, 2);

// add typeA

cur += 2;

short typeA = (short) 0x0001;

System.arraycopy(Convert.short2Byte(typeA), 0,

finalData, cur, 2);

// add classA

cur += 2;

short classA = (short) 0x0001;

System.arraycopy(Convert.short2Byte(classA), 0,

finalData, cur, 2);

// add timeLive

cur += 2;

int timeLive = 0x00015180;

System.arraycopy(Convert.int2Byte(timeLive), 0,

finalData, cur, 4);

// add responseIPLen

cur += 4;

short responseIPLen = (short) 0x0004;

System.arraycopy(

Convert.short2Byte(responseIPLen), 0,

finalData, cur, 2);

// add responseIP

cur += 2;

byte[] responseIP = InetAddress.getByName(

Check.ipTable.get(domainNameStr))

.getAddress();

System.arraycopy(responseIP, 0, finalData, cur,

4);

cur += 4;

}

// send UDP data packet

outPacket = new DatagramPacket(finalData,

finalData.length, resolverAddress,

resolverPort);

socket.send(outPacket);

}

} else {// if not found in the local txt

// send to remote DNS server

outPacket = new DatagramPacket(sendData,

sendData.length, InetAddress.getByName(DNS\_IP),

DNS\_PORT);

socket.send(outPacket); System.out.println("Relay time: " + new java.util.Date());

IPv6\_Flag = false;

System.out.println("function:" + "send to remote DNS server");

// id存储

IDTransition idTransition = new IDTransition(

(int) Convert.byte2Short(sendData, 0),

resolverPort, resolverAddress);

idMap.put(idTransition.getOldID(), idTransition);

}

} else {// response

// receive the data packet

int responseID = Convert.byte2Short(sendData, 0);

// outPacket = new DatagramPacket(sendData,

// sendData.length, resolverAddress, resolverPort);

// socket.send(outPacket);

if (idMap.containsKey(responseID)) {

IDTransition id = idMap.get(responseID);

// Relay the remote DNS server's response

outPacket = new DatagramPacket(sendData,

sendData.length, id.getAddr(), id.getPort());

socket.send(outPacket);

}

}

}

} catch (Exception e) {

socket.close();

e.printStackTrace();

}

}

public static void main(String[] args) throws UnknownHostException {

try {

Check.readData("dnsrelay.txt");// Read data

} catch (IOException e) {

e.printStackTrace();

}

System.out.println("Name Server: " + DNS\_IP);

System.out.println("Debug Level: 0");

System.out.println("Bind UDP port " + LOCAL\_PORT + " ...OK!");

System.out.println("Try to load table \"dnsrelay.txt\" " + " ...OK!");

System.out.println(Check.ipTable.size() + " names," + "occupy "

+ Check.fileSize + " bytes memory");

System.out.println("========================================");

(new DNSRelay()).init();// Start

}

}

**package** DNSRelay;

**import** java.net.InetAddress;

**public** **class** IDTransition {

**private** **int** oldID;

**private** **int** port;

**private** InetAddress addr;

**public** IDTransition(**int** oldID, **int** port, InetAddress addr) {

**this**.oldID = oldID;

**this**.port = port;

**this**.addr = addr;

}

**public** **int** getOldID() {

**return** oldID;

}

**public** **int** getPort() {

**return** port;

}

**public** InetAddress getAddr() {

**return** addr;

}

}