Yao Shubin (2018213086)

Xiao Xuezhi (2018213083)

Liu Zhanghan (2018213092)

Design and implementation of a DNS Relay

Design and implementation of a DNS Relay

**Content**

[1. Overview 3](#_Toc487909007)

[1.1. Brief requirement 3](#_Toc487909008)

[1.2. Target 3](#_Toc487909009)

[1.3. Implementation 3](#_Toc487909010)

[2. Requirements Analysis 3](#_Toc487909011)

[2.1. Detailed nonfunctional requirement 3](#_Toc487909012)

[2.2. Detailed functional requirement 4](#_Toc487909013)

[3. Preliminary Design 5](#_Toc487909014)

[3.1. Decomposition of functional modules 5](#_Toc487909015)

[3.1.1. Required domain name not in local database 5](#_Toc487909016)

[3.1.2. Required domain name in black list 5](#_Toc487909017)

[3.1.3. Required domain name in the list 5](#_Toc487909018)

[3.2. Relationship and interface between the modules 6](#_Toc487909019)

[3.3. Overall Flowchart 7](#_Toc487909020)

[3.4. Design of data structures 7](#_Toc487909021)

**4.** **DETAILED DESIGN…………………………………………………………………….10**

[**5. Test the program 13**](#_Toc487909023)

[6. Summary and Conclusion 16](#_Toc487909026)

[7. Conclusion and Implementation 16](#_Toc487909029)

[Appendix 17](#_Toc487909030)

# Overview

## Brief requirement

Since the basic requirement is to implement relay between Resolver and DNS Server. So the major parts need to be considered are (Figure 1):

* + - How to receive DNS queries from DNS client (Resolver) and push them to the given DNS server.
    - How to receive DNS responses from DNS server and forward them to the Resolver.

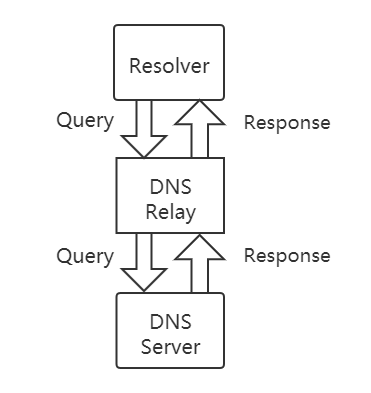


Figure 1 Relationship of DNS Relay and Resolver or Server

After analyses, there are four cases need to be considered:

* + - Case 1: If domain name included in the local database (e.g. dnsrelay.txt), sending back corresponding IP address
    - Case 2: if found, for IP address 0.0.0.0, sending back “no domain name” (reply code =0011)
    - Case 3: if domain name not included in the database, forward query to local DNS server
    - Case4: if domain name is in type of IPV6 and is also in the database, forward query to local server as well as sending back corresponding IP address (additional case)

## Target

**1: Understanding DNS:**

* + - DNS hierarchical structure;
    - DNS function;
    - DNS message format;
    - DNS protocol transport;

**2: How DNS severs working**

* + - Mastering Socket programming using UDP;
    - Understanding UDP protocol and message format;
    - Understanding the principle of Socket;
    - Using UDP-based sockets API;

**3: Skills**

* + - Using proper programming language in suitable environment such as Java in windows command line and having a good communication within the team.

## Implementation

In cmd, execute the following command:

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

# Requirements Analysis

# 2.1 Group Discuss Process

In the project process, our group cooperation is divided into three phases, they are: the project preparation phase, the project implementation phase, the project completion phase. The first two phases are completed through online discussion, and the last phase is completed after we back to school.

a) Project preparation phase：

In the preparation phase of the project, we separately prepared and reviewed the knowledge required for the project, with the help of the teacher's explanation and extra-curricular information, we independently learned the DNS concept and technology, socket programming, and reviewed the contents of the UDP protocol and another related knowledge.

b) Protect implementation phase:

In the implementation phase of the project, we discussed how to implement our DNS relay, concentrate on Thread, socket, and way to analyze packet.

Thread: We discussed that our DNS relay used multi-thread by thread pool. When DNS received a packet from client, it created a thread for this packet, which is responsible for the whole process of the package delivery. Since when we analyzed the packet using Wireshark, we found that sending one query would produce several packets. We discussed that if we used single thread in our relay, several packets may cause congestion. Multi-thread improves the efficiency of package delivery.

Socket: We discussed that out DNS relay used multi socket. One socket for communication between client and relay. And when query needs to forward to DNS server, it would create a socket for communication between relay and server.

Then, we had a division of work discussion, sharing our expertise in technology and areas and based on this we divide the work. After that, we carry on the project implementation according to the division of work, each of us completes our own tasks. We plan to hold a group discussion once a week to discuss the progress of the project and the problems encountered during the week, and solve them one by one.

c)Project completion phase :

In the completion phase of the project, we integrate and summarize our code on the Visual Studio Code Platform, solve the Code bug, implement the functions needed by the project, debug and test.

## 2.2 Detailed nonfunctional requirement

* + - OS: Windows
    - Programming language: Java

The operating system of development is Windows. And we use JAVA as our development language for its convenience. And all operations can be done by command line.

## Detailed functional requirement

* + - **Initialization requirement**

Since we need to initialize states to assure the program executed correctly:

Read DNS server IP address and path of local database

* Initialize Socket: Open socket; bind to port 53
* Get the size of received packet and packet to be sent.
* Print the basic information about the program, which are stored in matrix, including DNS server IP, bound local UDP port, local database and the size of current database
* Close the socket connection and exit program, when an exception is thrown
* Listen to port 53(UDP port) until a packet is received and then repeat this operation.
  + - **Packet parser requirement**

Used for parsing the packet attributes:

* Find out it is a response packet or a query packet, and sent to corresponding handler
* Extract the domain name from packet
* Determine whether domain name is found in local database
* Calculate the length of payload of UDP packet
  + - **Local DNS server requirement**

Responsible for dealing with domain name in local database:

* Recognize if the related IP address is 0.0.0.0 then construct the different response packet to resolver according to different situation.
* If the IP address found in the database equals is to 0.0.0.0, then it should only modify flag to 0x8183 and leave the answer count still equals to 0 with no change.
* If the IP address found in the database doesn’t equals to 0.0.0.0, then it should change much more parts of the query package, including changing the flag to 0x8180, changing the answer count to 1, and attaching the answer section containing the retrieving IP address
* Use one socket to send response for IPV4 and IPV6 request to prevent duplicated response, which need a flag (Boolean) variable to distinguish them.
* For example, if it’s a IPV6 request such as [www.bupt.edu.cn](http://www.bupt.edu.cn) with an IP in the database, the IPV6 domain name will make the DNS return the IP stored in the database as an IPV4 address, meanwhile DNS will work as a relay to send a packet to ask upper network and return the response as the IPV6 address. We will talk more in the testing chapter.
  + - **DNS relay requirement**

Responsible for dealing with domain name not in local database:

* Because it allows concurrent query, the relay should have the ability to store the original Client address (resolver address) corresponding to its original ID and matching ID which is the index in the ‘IDStored’
* When a response packet sent by DNS sever is received, there is a lookup table finding the corresponding old ID and Socket Address
* Directly forward the response packet without changing the answer filed
* Can deal with both IPV4 and IPV6 request

# Preliminary Design

## Decomposition of functional modules

Mainly three situations for domain name are required by the client. And we decompose them as following:

### If the required domain name cannot be found in local database

Function: Serves as a DNS Relay Steps:

* + - 1. Receive the DNS query
      2. Unpacked the query and get domain name
      3. Search the domain name and compare in the local database list
      4. Send to DNS server
      5. Receive the response of DNS server and forward to client
* **Tips：**

The process of the package in the program is a non-blocking single thread, so that the packet can be processed quickly and the corresponding time-out processing is carried out. If a packet loss or reply error occurs, the request is automatically retransmitted over a period of time.

### Black list website interception function

The function module is to achieve the interception function. The main parts as follow:

* + - 1. Receive the DNS query
      2. Analyze the query and get the information like flags, domain name etc.
      3. Search the domain name and compare to the local database
      4. Find its IP address is 0.0.0.0
      5. Set its RCODE as 3, which means name in query does not exist.
      6. Set answer account as 0
      7. Produce the packet to send response to DNS client

### Local Database Query function

The function module is to achieve local database domain name-IP query. The main parts as follow:

* + - 1. Receive the DNS query
      2. Analyze the query and get the information like flags, domain name etc.
      3. Search the domain name and compare to the local database
      4. Find its IP address
      5. Produce the packet to send response to DNS client.

### DNS server query function

The function module is to achieve domain name-IP address query which is not in local database. The main parts as follow:

* + - 1. Receive the DNS query
      2. Analyze the query and get the information like flags, domain name etc.
      3. Search the domain name and compare to the local database
      4. IP address is not found
      5. Send query packet to DNS server
      6. Server send back response
      7. Relay send response packet to client

## Relationship and interface between the modules

Basic function of DNS Relay is to receive DNS queries from DNS (Resolver) and forward them to a given DNS server as well as receiving DNS responses from DNS server and forward them to the Resolver



Figure 2 Relationship of DNS Relay and Resolver or Server

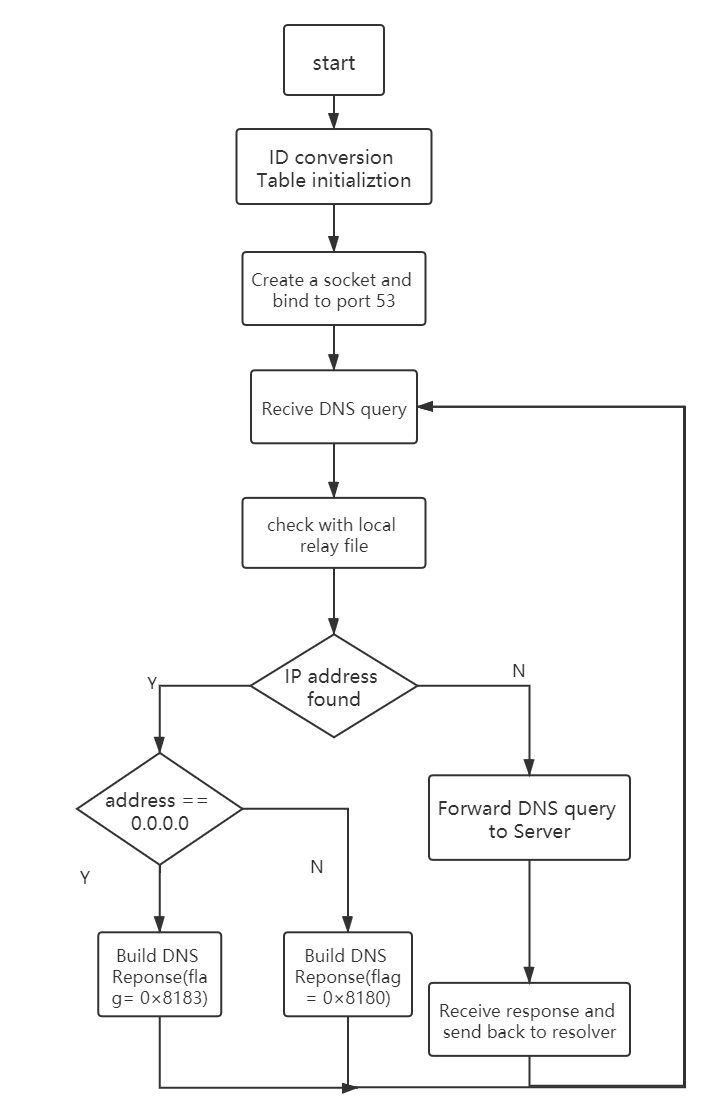
In this model, DNS Relay plays three different roles which has mentioned above, which include a DNS Server, a DNS Filter and a Relay itself.

## Overall Flowchart

Basic description of steps:

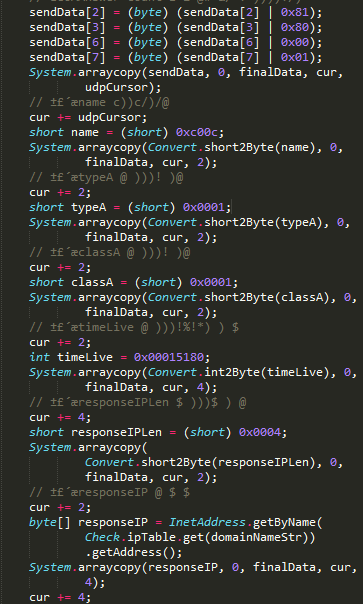
1. ID Conversion Table Initialization.
2. For DNS relay and DNS server creates the sockets.
3. Bind IP address of local DNS socket to port.
4. Receive query from the connected socket.
5. Get the domain name from query and search in Resolution Table.
   * + - If not found, resend DNS queries to DNS server. Forward to client after getting query results.
       - If found, judge whether the domain name is in black list or not:
         * If it is in black list, set answer count as 0.
         * If not, set answer count as 1 and make a UDP packet and response to resolver.
6. Go back to step (d)

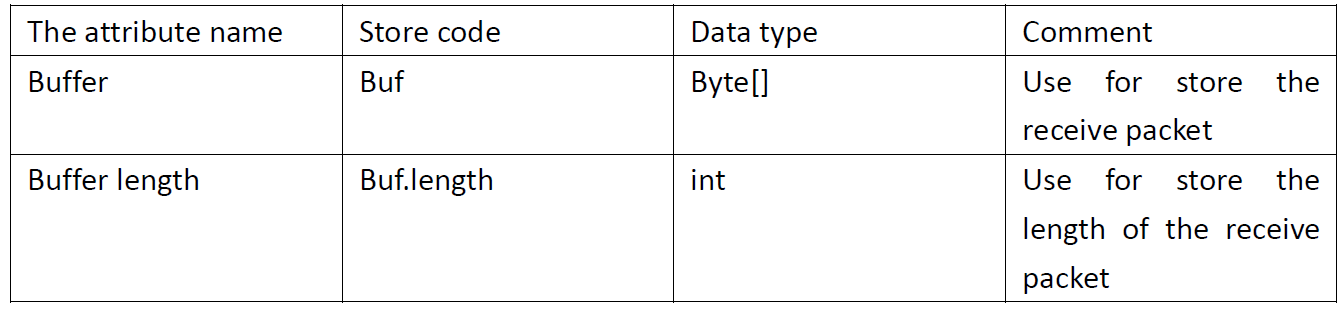
The flowchart is as showed below.



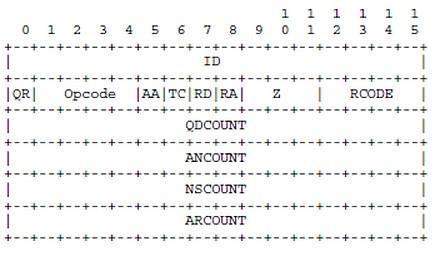
## Design of data structures

### Header





This data structure is designed based on Message Format from RFC 1035. The Format is as following:



**ID**: 16-bit field used to correlate queries and responses.

**QR**: 1-bit field that identifies the message as query (0) or response

**OPCODE**: 4-bit field that describes the type of query:

* 0: Standard query (name to address)
* 1: Inverse query (address to name)
* 2: Server status request

**AA**: authoritative Answer. 1-bit field. When set to 1, identifies this response is made by an authoritative name server.

**TC**: Truncation. 1-bit field. When set to 1, indicates the message has been truncated due to length greater than that permitted.

**RD**: Recursion Desired. 1-bit field. Set to 1 by the resolver to request recursive service by the name server.

**RA:** Recursion Available. 1-bit field. Set to 1 by name server to indicate recursive query support is available.

**Z**: 3-bit field. Reserved for future use. Must be set to 0.

**RCODE**: Response Code. 4-bit field that is set by the name server to identify the status of the query:

* + 0: No error condition.
  + 1: Unable to interpret query due to format error.
  + 2: Unable to process due to server failure.
  + 3: Name in query does not exist.
  + 4: Type of query not supported.
  + 5: Query refused for policy reasons.

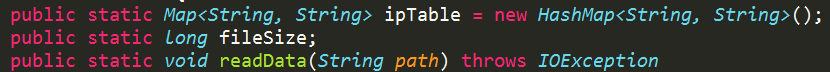
**QDCOUNT** (Question count): 16-bit field that defines the number of entries in the question section.

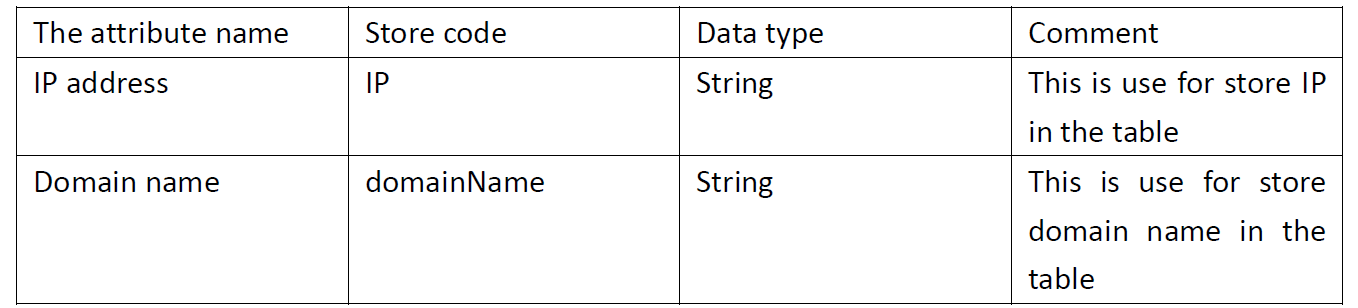
**ANCOUNT** (Answer count): 16-bit field that defines the number of resource records in the answer section.

**NSCOUNT** (Authority count): 16-bit field that defines the number of name server resource records in the authority section.

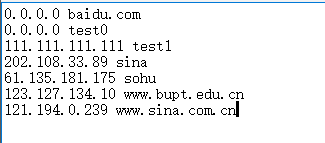
**ARCOUNT** (Additional count): 16-bit field that defines the number of resource records in the additional records section.

* + 1. **Domain Name Resolution Table**





This data structure is designed based on local database. When a domain name is in the black list, it is corresponding to IP 0.0.0.0.

By the structure, we can search a local database to determine whether need to send query to DNS server or not. The following picture is the local database file: 

* + 1. **The dictionary of building a new packet for socket**

This data structure is used to store the ID information of message, in order to ensure the concurrency of query can be responded correctly between relay and DNS server when the domain name is not in the local database.

**4. Detailed design**

**4.1 Initialization and Receive query**

ID Conversion Table Initialization

load the “dnsrelay.txt” into the memory

Create the sockets of DNS relay and DNS server and initialize

Create UDP packet and bind to the port 53

Receive packet from client in always true loop

Extract the UDP part from the received packet

Then get DNS data from UDP and named as “senddata”

Judge if it is a query or response by bit operation

If it is a query, then store the address and port of the packet

 If it is response, send forward to the resolver.



* 1. **Required domain name in white list or black list**

**** Judge whether the IP equals to 0.0.0.0.

****  If yes, shield the packet by changing the flag into 8183.

**** If no, judge whether the packet is of IPv4 or IPv6 type.

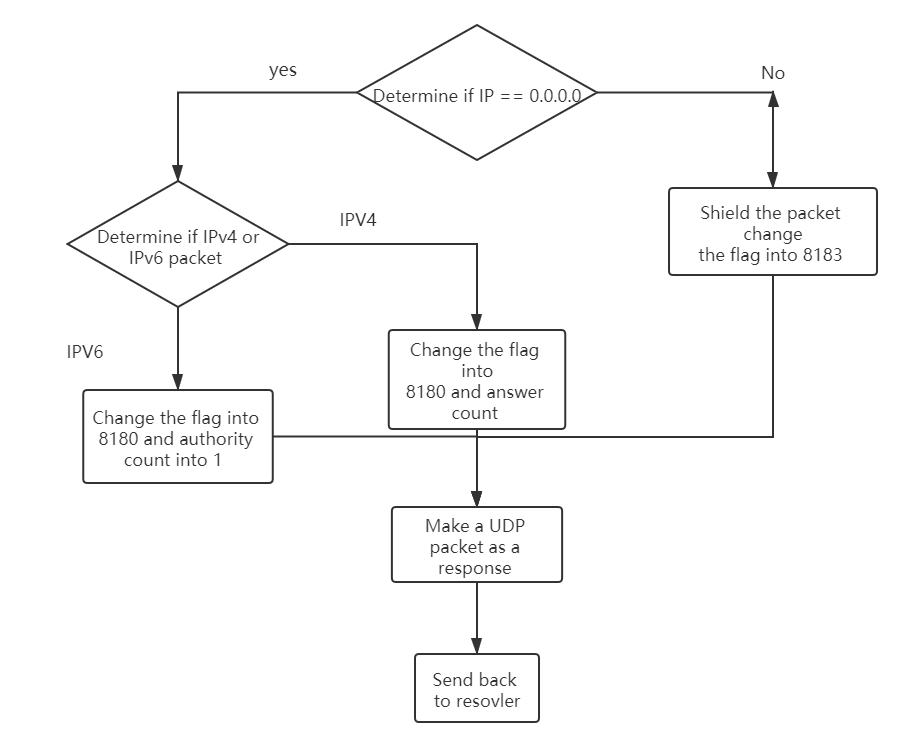
 If IPv6, change the flag into 8180 and authority count into 1.

 If IPv4, change the flag into 8180 and answer count into 1.

**** Then make a UDP packet as a response.

**** Send back to resolver by the address and port stored in Model 1.

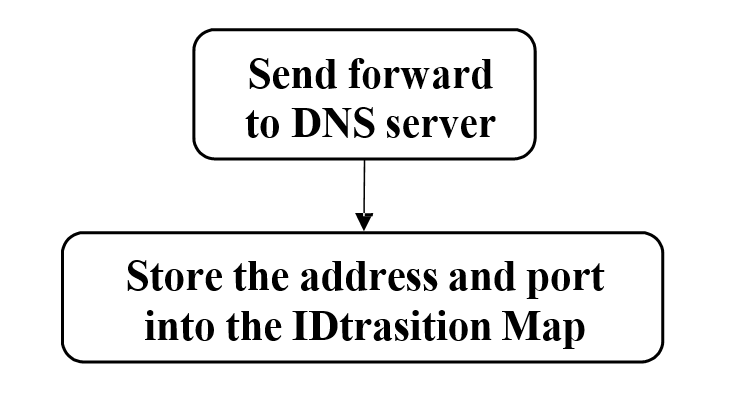
Flowchart:

****

**4.3 Required domain name not in local database**

**** Use the address and port stored in the Model 1 to send the packet forward to the DNS server.

****Store the address and port in the IDtransition and ID map

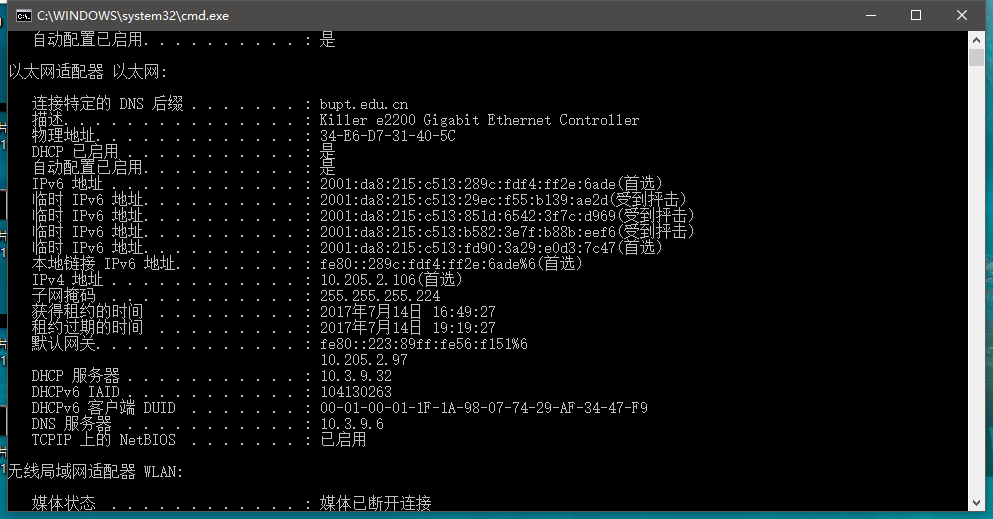
****

**5. Results**

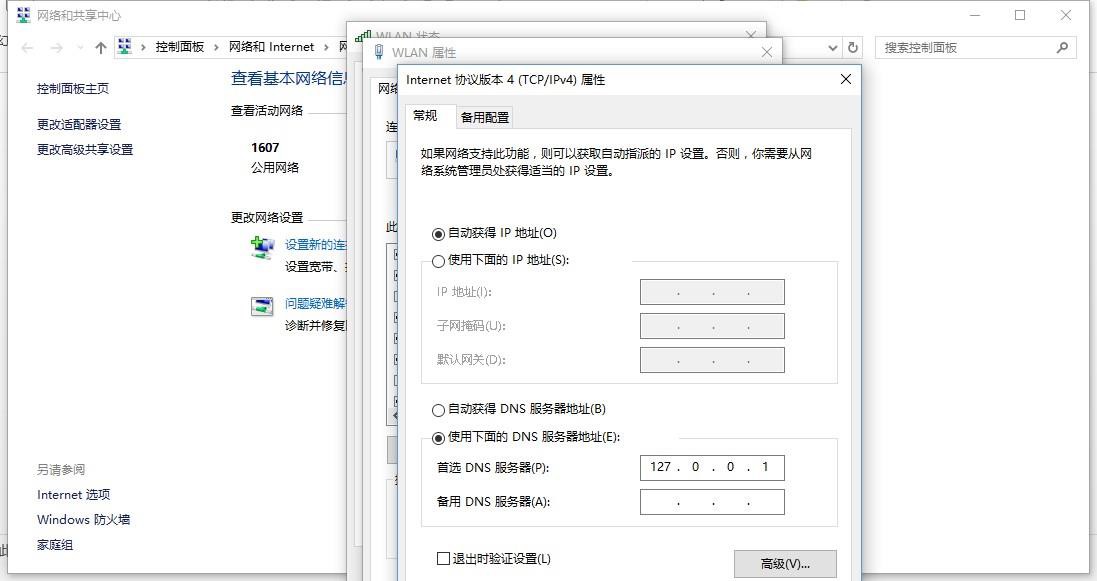
**5.1 Running the program**

1. Get the IP address of our DNS server (e.g. 192.168.2.2 ) and our IP address (e.g. 192.168.2.110 )

c: > ipconfig /all



1. configure our DNS server as our IP address (e.g. 192.168.2.110 or 127.0.0.1)



1. Running our program, setting local DNS server as 202.106.0.20
2. Using nslookup to test our program

## 5. 2 Test the program

Part of data in relay.txt:

## C:\Users\ss\AppData\Local\Microsoft\Windows\INetCache\Content.Word\1.png

* + - Test a domain name whose IP address can be found and equals to 0.0.0.0.

For example, test “test0”. In relay.txt, its IP address is 0.0.0.0:

nslookup baidu.com

Because of exist of related IP address, 0.0.0.0, in local database, it will receive “NO such name”response:

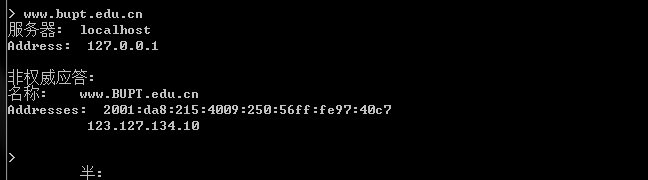
## C:\Users\ss\AppData\Local\Microsoft\Windows\INetCache\Content.Word\微信图片_20170714172003.png

In our relay program, the debug information shows (because of both request of IPV6 and IPV4, program will receive two request

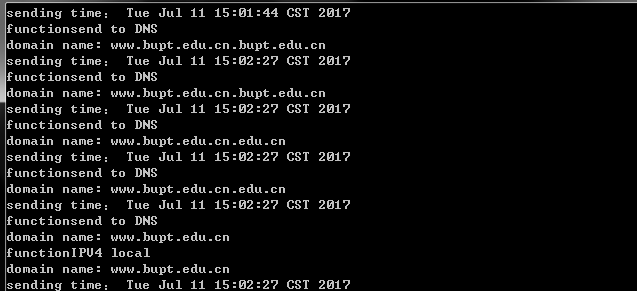
* + - Test a domain name whose IP address can be found and not equals to 0.0.0.0

nslookup www.bupt.edu.cn

Because of exist of related IP address in local database, it will response packet sent by our program:



In our relay program, the debug information shows (because of both request of IPV6 and IPV4, program will receive two request):

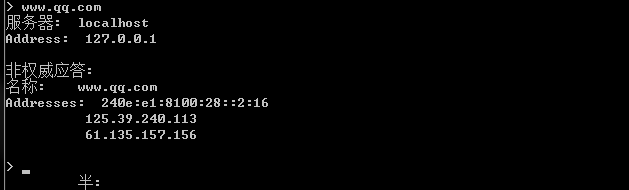


* + - Test a domain name whose IP address cannot be found

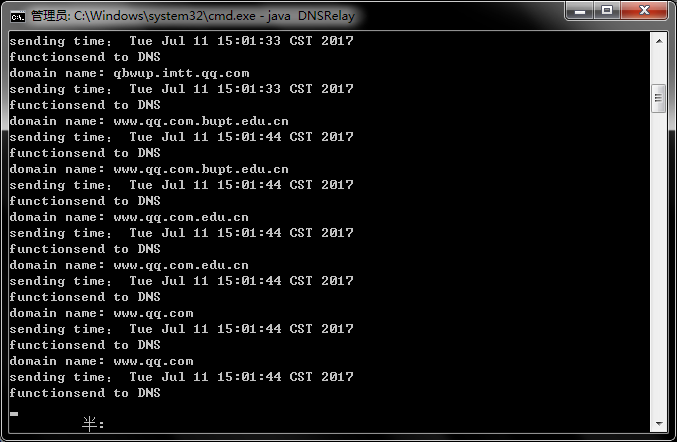
For example, test [www.qq.com](http://www.qq.com):

nslookup www.qq.com

Because of nonexistence of related IP address in local database, our program will forward this quest to DNS server:



In our relay program, the debug information shows (because of both request of IPV6 and IPV4, program will receive two request):



# 6. Summary and Conclusion

 **Yao Shubin**

Task Assignment:

I mainly response for construct whole code structure and divide the module into several sub-modules, then write the code. Meanwhile, I also learn some knowledge about the socket programming and write code related to DNS relay such as data packet reception and transmission. Besides, based on Xiao Liu’s requirement analysis, I refine it into preliminary design of the whole project and write the report of preliminary design. And I understand details of Domain Name System and master of details of RFC 1035 protocol and DNS message format.

Self-evaluation:

From this project, I learn a lot about how to use JAVA language to achieve the DNS relay and gain some experience about the socket programming. By analyzing the whole project, I have a clear comprehension on the DNS relay working mechanism and how to modify the local DNS relay which is helpful in further computer network study.

### Xiao Xuezhi

### Task Assignment:

I read some documents about the communication process between DNS relay and external DNS which is helpful to construct the connections. And I am responsible for designing and coding tools classes, including transforming byte to int and byte to short; responsible for design and coding the process of UDP connection, sending and receiving packets. Moreover, I also write some code which is related to data packet encapsulation and I summarize the preliminary design worked by Da Han and write the detail design.

Self-evaluation:

I have a good knowledge of the computer network communication in UDP layer and can write some code to achieve its function. Meanwhile, I also learn a lot about the DNS service and its working mechanism.

 **Liu Zhanghan**

Task Assignment:

I analyze the project and give a brief requirement of this project for further analysis and read some documents about DNS concept. I also write some code about the achievement of some small function and write the requirements analysis. Responsible for designing and coding the part related to dealing with receiving packets and response packet and extracting the domain name from DNS packet; responsible testing program and recording results.

Self-evaluation:

I learn a lot about DNS working mechanism and gain a lot of experience on JAVA programming language on this project which gives me a clear understanding on the computer network.

# 7. Conclusion and Implementation

At the beginning, we peruse the requirements of DNS Relay system immediately after class. After finishing analyzing requirements, we look through the communications between DNS resolver, DNS relay and DNS server. And get the basic idea of how to implement the system. Besides DNS protocol, we also reviewed UDP protocol studied in Adhoc network, since DNS is based on UDP transport layer. So we are about to programming the system.

As we begin to programming, there are quite a lot situation which we haven’t thought through in advance. However, after debating and consulting on a certain extent, we managed to overcome the obstacles. For example, in the transport layer, when the client sends the packet to the relay and then to the server, the port number used is 53. However, when the packet is returned to the client, the port number needs to be obtained through the packet sent. We found this problem in time and improved the relay.

During programming, we met several problems especially the IP address will show in 2 times which is caused by the judgement of the packet type in IPv4 and IPv6. Another problem is about the concurrency of the received packet, in order to avoid the override of original address, we use IDStored class to store them by hash map.

In general, during the course, we reviewed the basic knowledge of computer network, and we reviewed socket programming and so on, enjoying the efficiency of pair programming. For further improvement, we will carry on the long-term programming practice, simultaneously grasps the computer and the network knowledge, we will also do long-lasting programming practice and catch the knowledge of computer and network at the same time. The understanding of TCP protocol is also important in our studying life.

# Appendix

# 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 DNSSystem {

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

# public static final int DNS\_PORT = 53;

# public static final int LOCAL\_PORT = 53;

# private static final int DATA\_LEN = 4096;

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

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

# private DatagramPacket outPacket;

# private String domainNameStr;

# private InetAddress resolverAddress;

# private int resolverPort;

# private boolean IPv6\_Flag = false;

# int udpCursor;

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

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

# 

# 

# public String getDomainName(byte[] buf) {

# String domainName = "";

# udpCursor = 12;

# 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;

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

# }

# public void init() {

# DatagramSocket socket = null;

# try {

# socket = new DatagramSocket(LOCAL\_PORT);

# while (true) {

# socket.receive(inPacket);

# byte[] sendData = inPacket.getData();

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

# domainNameStr = getDomainName(sendData);

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

# resolverAddress = inPacket.getAddress();

# resolverPort = inPacket.getPort();

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

# String LocalDNSipAddress = Check.ipTable.get(domainNameStr);

# if (LocalDNSipAddress.equals("0.0.0.0")) {

# System.out.println("function£º" + "blacklist");

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

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

# outPacket = new DatagramPacket(sendData,

# sendData.length, resolverAddress,

# resolverPort);

# socket.send(outPacket);

# IPv6\_Flag = false;

# } else {

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

# int cur = 0;

# if (IPv6\_Flag) {

# outPacket = new DatagramPacket(sendData,sendData.length, InetAddress.getByName(DNS\_IP),DNS\_PORT);

# socket.send(outPacket); System.out.println("sending time£º " + new java.util.Date());

# IPv6\_Flag = false;

# System.out.println("function" + "send to DNS");

# IDStored idStored = new IDStored(

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

# resolverPort, resolverAddress);

# idMap.put(idStored.getOldID(), idStored);

# } else {

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

# 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);

# cur += udpCursor;

# short name = (short) 0xc00c;

# System.arraycopy(Convert.short2Byte(name), 0,finalData, cur, 2);

# cur += 2;

# short typeA = (short) 0x0001;

# System.arraycopy(Convert.short2Byte(typeA), 0,finalData, cur, 2);

# cur += 2;

# short classA = (short) 0x0001;

# System.arraycopy(Convert.short2Byte(classA), 0,finalData, cur, 2);

# cur += 2;

# int timeLive = 0x00015180;

# System.arraycopy(Convert.int2Byte(timeLive), 0,finalData, cur, 4);

# cur += 4;

# short responseIPLen = (short) 0x0004;

# System.arraycopy(Convert.short2Byte(responseIPLen), 0,finalData, cur, 2);

# cur += 2;

# byte[] responseIP = InetAddress.getByName(Check.ipTable.get(domainNameStr)).getAddress();

# System.arraycopy(responseIP, 0, finalData, cur,4);

# cur += 4;

# }

# outPacket = new DatagramPacket(finalData,finalData.length, resolverAddress,resolverPort);

# socket.send(outPacket);

# }

# } else {

# outPacket = new DatagramPacket(sendData,

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

# DNS\_PORT);

# socket.send(outPacket); System.out.println("sending time£º " + new java.util.Date());

# IPv6\_Flag = false;

# System.out.println("function" + "send to DNS");

# IDStored idStored = new IDStored((int) Convert.byte2Short(sendData, 0),resolverPort, resolverAddress);

# idMap.put(idStored.getOldID(), idStored);

# }

# } else {

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

# if (idMap.containsKey(responseID)) {

# IDStored id = idMap.get(responseID);

# 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");

# }catch (IOException e) {

# e.printStackTrace();

# }

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

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

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

# + Check.fileSize + " bytes memory");

# System.out.println("---------------------------------------------------");

# (new DNSSystem()).init();

# }

# }

# 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();

# 

# }

# }

# }

# 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;

# }

# 

# 

# import java.net.InetAddress;

# public class IDStored {

# 

# private int oldID;

# private int port;

# private InetAddress addr;

# 

# public IDStored(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;

# }

# }

# 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);

# }

# 

# /\*\*

# \* 字节转换为浮点

# \*

# \* @param b 字节（至少4个字节）

# \* @param start 开始位置

# \* @return 浮点

# \*/

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

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

# return Float.intBitsToFloat(i);

# }

# 

# /\*\*

# \* 字节转换为字符串

# \* @param b 字节

# \* @param start 起点

# \* @param length 长度

# \* @return 字符串

# \*/

# 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;

# }

# 

# /\*\*

# \* 字符串转换为字节

# \* @param c 字符串

# \* @param b 字节

# \* @param start 字节中起点

# \* @param length 字符串长度

# \* @return 字节中的偏移量

# \*/

# 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;

# }

# 

# /\*\*

# \* 字符串转换为字节

# \* @param b 字节

# \* @param start 起点

# \* @param length 长度

# \* @return 字符串

# \*/

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

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

# }

# /\*\*

# \* 字符串转换为字节

# \* @param s 字符串

# \* @param b 字节

# \* @param start 字节中起点

# \* @param length 字符串长度

# \* @return 字节中的偏移量

# \*/

# 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);

# }

# }

# 

# /\*\*

# \* 获得异或校验和的值

# \* @param source 需要校验的数据源

# \* @param start 校验起点

# \* @param length 校验长度

# \* @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;

# }

# }