

**课程设计报告**



**题目：DNS中继服务器的实现**

**班 级：2018211319**

**学 号：2018211747 2018211947 2018211698**

**姓 名：张天秋 王子澄 邓宇涛**

**学 院：计算机学院**

**2020年 9 月 7 日**

1、系统的功能设计

1.1 基本功能

一个 DNS 服务器程序，能够读入 IP 地址-域名 (hosts) 对照表。收到客户端发来的报文，解析其含义，编写一个回复报文发送给客户端。当客户端查询域名对应的 IP 地址时，用域名检索该对照表，有三种可能检索结果：

1. 检索结果 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）

2. 检索结果为普通 IP 地址，则向客户端返回该地址（服务器功能）

3. 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）

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。

可以在程序中加入不同等级的调试信息（如输出时间坐标、序号、客户端IP地址、查询的域名、收到的报文和发送的报文等），根据命令行输入的命令（无/-d/-dd）显示不同等级的调试信息。

1.2 附加功能

在基本功能的基础上，我们附加了两种功能：

* 1. 兼容IPv6查询
  2. 增加LRU缓存池，可以减少DNS流量，快速回复给服务器。当再次收到域名查询时，首先查询LRU缓存池中是否有对应域名，并检查是否为IPv4查询；若查到则将IP返回，否则继续查询hosts文件或发送给远端服务器进行查询。

2 模块设计

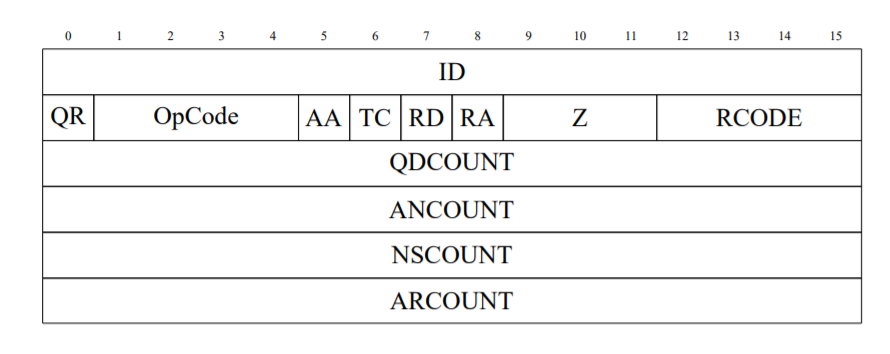
2.1 主模块（main）

初始化各项配置，创建socket对象并初始化，加载IP地址-域名转换表，最后进入服务器工作模块并一直执行。

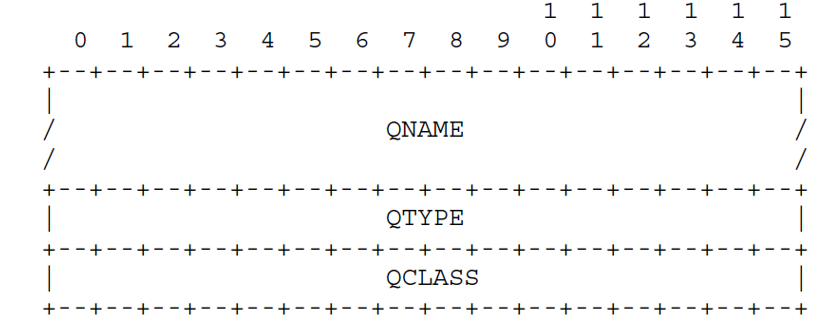
|  |  |
| --- | --- |
| void initCommand(argc, argv) | 该函数处理从命令行读取的参数设置，初始化调试等级、外部DNS服务器的IP地址，并载入IP地址-域名转换表准备读取 |
| void initWSA() | 为了向操作系统说明，我们要用哪个库文件，让该库文件与当前的应用程序绑定，从而就可以调用该版本的*socket*的各种函数了。 |
| void work(int sockfd) | 服务器工作模块，是本服务器的核心处理模块，传入参数为socket对象 |

2.2 数据字段模块

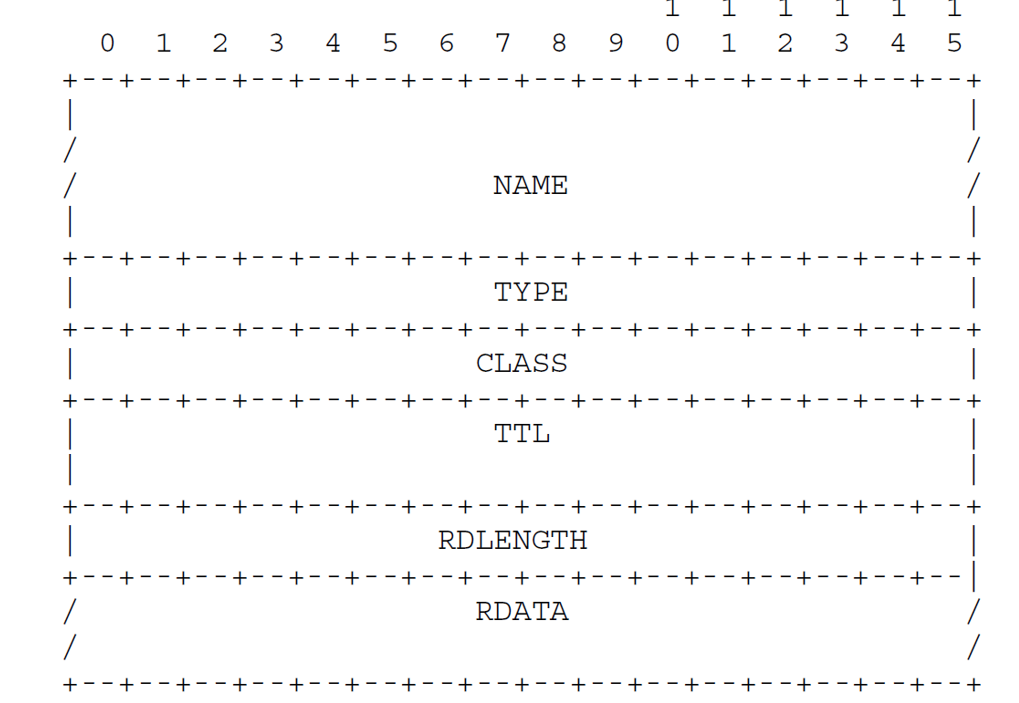
该模块主要声明请求报文和应答报文的结构体，以便将来把得到的字节流解码为相应的结构体元素。还有在服务器内需要建立ID转换表，将客户端与中继服务器的报文ID和外部DNS服务器与中继服务器发送的报文ID进行匹配。



1. **typedef** **struct** DNSHEADER    //DNS报文报头字段
2. {
3. unsigned **short** ID;      //2 bytes
4. unsigned **short** Flag;    //2 bytes
5. /\*  Flag字段共包含以下几个字段
6. QR：0表示查询报，1表示响应报。
7. OPCODE：通常值为0（标准查询），其他值为1（反向查询）和2（服务器状态请求）。
8. AA: 权威答案(Authoritative answer)
9. TC: 截断的(Truncated)，应答的总长度超512字节时，只返回前512个字节
10. RD: 期望递归(Recursion desired)，查询报中设置，响应报中返回，告诉名字服务器处理递归查询。如果该位为0，且被请求的名字服务器没有一个权威回答，就返回一个能解答该查询的其他名字服务器列表，这称为迭代查询
11. RA：递归可用(Recursion Available)，如果名字服务器支持递归查询，则在响应中该比特置为1
12. Z：必须为0，保留字段
13. RCODE: 响应码(Response coded)，仅用于响应报
14. 值为0(没有差错)
15. 值为3表示名字差错。从权威名字服务器返回，表示在查询中指定域名不存在
16. \*/
17. unsigned **short** QDCount; //2 bytes
18. unsigned **short** ANCount; //2 bytes
19. unsigned **short** NSCount; //2 bytes
20. unsigned **short** ARCount; //2 bytes
21. } dnsHeader;



1. **typedef** **struct** DNSQUERY
2. {
3. **char**\* Qname;            //查询域名
4. unsigned **short** Qtype;   //2 bytes
5. /\*
6. A(1) : IPv4
7. AAAA(28) : IPv6
8. \*/
9. unsigned **short** Qclass;  //IN(1), 2 bytes
10. } dnsQuery;



1. **typedef** **struct** DNSRR
2. {
3. **char**\* Name;             //restore Domain Name
4. unsigned **short** Type;    //16 bits
5. unsigned **short** Class;   //16 bits
6. unsigned **int** TTL;       //32 bits
7. unsigned **short** RDLength;//16 bits
8. unsigned **char**\* RData;            //restore IP address
9. } dnsRR;

最终将所有结构体汇总到一个报文结构体中:

1. **typedef** **struct** DNSPacket
2. {
3. dnsHeader header;
4. dnsQuery \*question;
5. dnsRR \*answer;
6. dnsRR \*authority;
7. dnsRR \*additional;
8. } dnsPacket;

ID转换表的结构体和结构数组如下所示:

1. **typedef** **struct** idRecord
2. {
3. unsigned **short** ServerID;            //发送给Server的报文ID号
4. unsigned **short** ClientID;            //发送给Client的报文ID号
5. **struct** sockaddr\_in sa;              //用户的socket address
6. **time\_t** timestamp;                   //时间戳
7. } IDRecord;

10. IDRecord IPTable[RECORD\_SIZE];          //ID转换表

LRU缓存池的结构体如下所示：

1. **typedef** **struct** entry
2. {
3. unsigned **char**\* IP;        //LRU表项中的IP地址
4. **char**\* DN;                 //LRU表项中的域名
5. **struct** entry\* next;       //指向下一个表项结构体的指针
6. } Entry;
8. **typedef** **struct** lruCache
9. {
10. Entry\* head;              //LRU链表头指针
11. Entry\* tail;              //LRU链表尾指针
12. **int** size;                 //LRU缓存池大小
13. } LRUCache;
15. LRUCache lrucache;

2.3 服务器工作模块

负责对收到的请求进行处理和对发送的数据进行填写。这一大模块中包含几个子模块：

2.3.1 解码、编码模块

服务器将收到的字节流进行信息解码，并将网络字节顺序转换为CPU字节顺序，将得到的数据存入相应的结构体字段中。在经过信息处理之后将要发送的信息通过字节顺序转换放入缓存中，再传递给sendto函数。

|  |  |
| --- | --- |
| void Decode(dnsPacket\* Packet, struct sockaddr\_in \*sockFrom, char \*buf, int bufLen) | 编码函数，传入参数为要写入的报文结构体、socket的发送方地址、缓存及缓存长度 |
| void Encode(dnsPacket\* Packet, char \*buf, int \*bufLen) | 解码函数，传入参数为已完善信息的报文结构体、缓存及缓存长度 |

2.3.2 socket通信模块

Socket通信实现模块，实现UDP接收/发送数据包。

|  |  |
| --- | --- |
| void initSock(int\* sockfd, struct sockaddr\_in\* sockINServer) | 用于创建socket对象因传入参数：特网 TCP/IP 地址族         S；: 以数据报为传输形式         I；P: 采用 UDP 协议 |
| void recvPacket(int\* bufLen, int sockfd, char\* buf, int packetSize, struct sockaddr\_in\* sockFrom, socklen\_t\* sockLen) | 包装recvfrom函数，用于接收UDP数据包 |
| void sendPacket(int sockfd, char\* buf, int packetSize, struct sockaddr\_in\* sockTo, socklen\_t\* sockLen) | 包装snedto函数，用于发送UDP数据包 |

2.3.3 中继服务器模块

该模块是服务器的核心，对收到的UDP数据包进行判断，若是来自客户端的IPv4包则首先查询lru缓存池，若找到对应IP则发送结果给客户端并更新缓存池，否则查询本地host，若找到对应IP则发送结果回客户端并更新缓存池，若在lru缓存池和本地host中都未查询到，则转发给上游DNS服务器；若是来自上游DNS服务器的数据包，则查看ID转换表并将ID进行替换，最终将数据包发回给相应的客户端。

通过ID转换表可以处理多个客户端并行请求的情况，通过将客户端的报文ID与产生的独一无二的新ID绑定，从而区分不同的客户端，并在转发应答时通过ID转换表查询到原报文ID和客户端的socket地址，实现报文的正确传输。同时，每一个ID对应表项都设置了TTL，当超过一定的时间后，ID转换表会自动将超时的表项删除，不会出现超时应答仍可得到答案的可能。

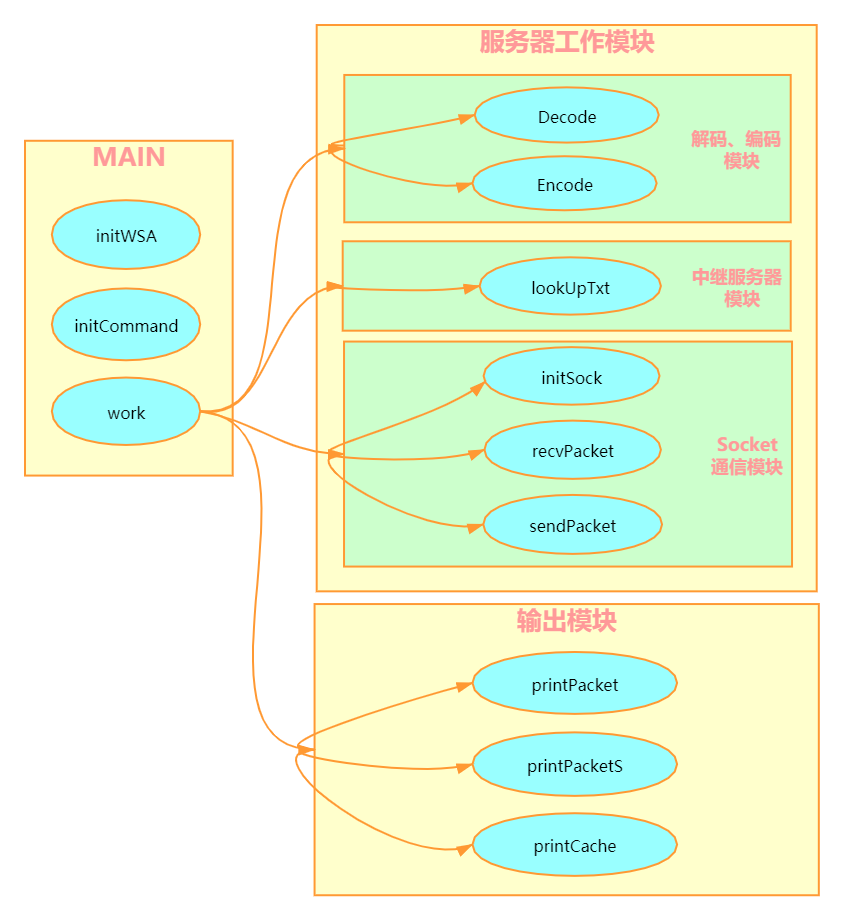
|  |  |
| --- | --- |
| int lookUpTxt(char\* DN, unsigned char\* IP) | 查询IP-域名对应表，若成功找到则返回1，否则返回0。传入参数为查找的域名和找到后存储的IP |

2.4 输出模块

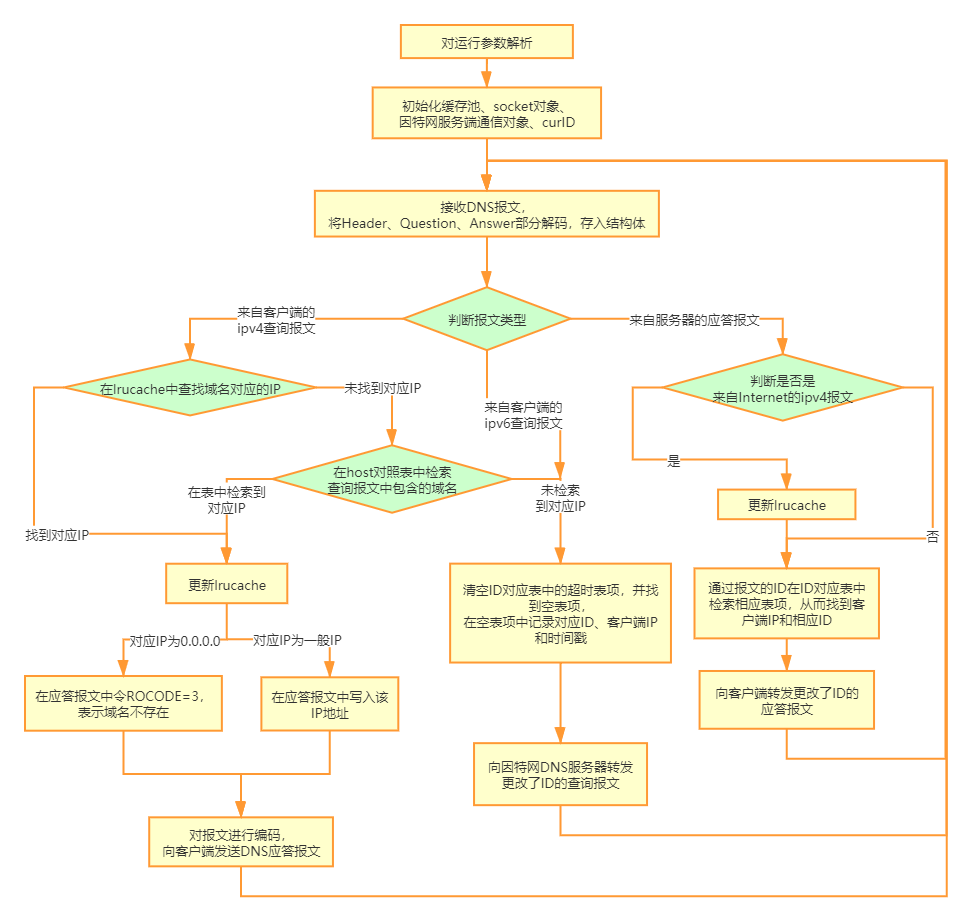
输出模块将需要打印的信息封装在函数中，通过调用函数来实现信息的输出。并且根据调试等级的不同设置了不同程度的输出信息，debugLevel = 0表示无调试信息；debugLevel = 1表示简要调试信息，会展示报文头部和查询段的信息；debugLevel = 2表示详细调试信息，会展示每一个传送报文的字节，利于调试。

|  |  |
| --- | --- |
| void printPacket (const char\* preface, struct sockaddr\_in \*sockFrom, char \*buf, int bufLen) | 打印以字节形式显示的报文(debugLevel >= 2) |
| void printPacketS(const char\* preface, dnsPacket\* Packet, struct sockaddr\_in \*sockFrom, int bufLen) | 打印以结构体形式显示的报文(debugLevel >= 1) |
| void printCache() | 打印lrucache的内容 |

3、函数调用关系图



4、程序设计流程图



5、测试用例与运行结果

测试准备：

* 1. 在系统环境为 MacOS 的本地 PC 上于程序所在文件夹使用 gcc DNS-Server.c -o DNS-Server 编译本程序为名为 DNS-Server 的可执行文件；
  2. 在程序同一目录下新建文本文件 dnsrelay.txt，并在文件中包含“10.105.242.83 [www.bupt.oj.com](http://www.search.com)”、“0.0.0.0 [www.notgood.com](http://www.notgood.com)”两条记录用作示例；
  3. 在该目录内使用 sudo ./DNS-Server -dd 10.3.9.4 运行本程序；
  4. 使用 nslookup [websit] - 127.0.0.1 实现指定本地 IP 地址 127.0.0.1为 DNS 服务器的 DNS 查询行为，进行第一阶段测试；
  5. 在本地 PC 上修改网络 DNS 服务器为本地 IP 地址 127.0.0.1，进行第二阶段测试；
  6. 将程序放置于云服务器上运行，使用多台本地 PC 将其作为 DNS 服务器，进行第三阶段测试。

测试用例：

第一阶段

* 1. 命令 nslookup [www.bupt.oj.c](http://www.bupt.edu.cn)om - 127.0.0.1，测试 DNS 服务器功能；
  2. 命令 nslookup [www.notgood.com](http://www.notgood.com) - 127.0.0.1，测试不良网站拦截功能;
  3. 命令 nslookup [www.bupt.edu.cn](http://www.bupt.edu.cn) - 127.0.0.1，测试中继功能;
  4. 设置 nslookup 的查询模式为查询 iPv6 模式，查询 [www.baidu.com](http://www.baidu.com) 的 IP 地址，测试 iPv6 查询的兼容功能;
  5. 再次命令 nslookup [www.bupt.edu.cn](http://www.bupt.edu.cn) - 127.0.0.1，测试 LRUCache 功能;

第二阶段

* 1. 正常进行浏览，检查程序 debug 输出；
  2. 浏览器地址栏键入 [www.bupt.edu.cn](http://www.bupt.edu.cn) 并打开；

第三阶段

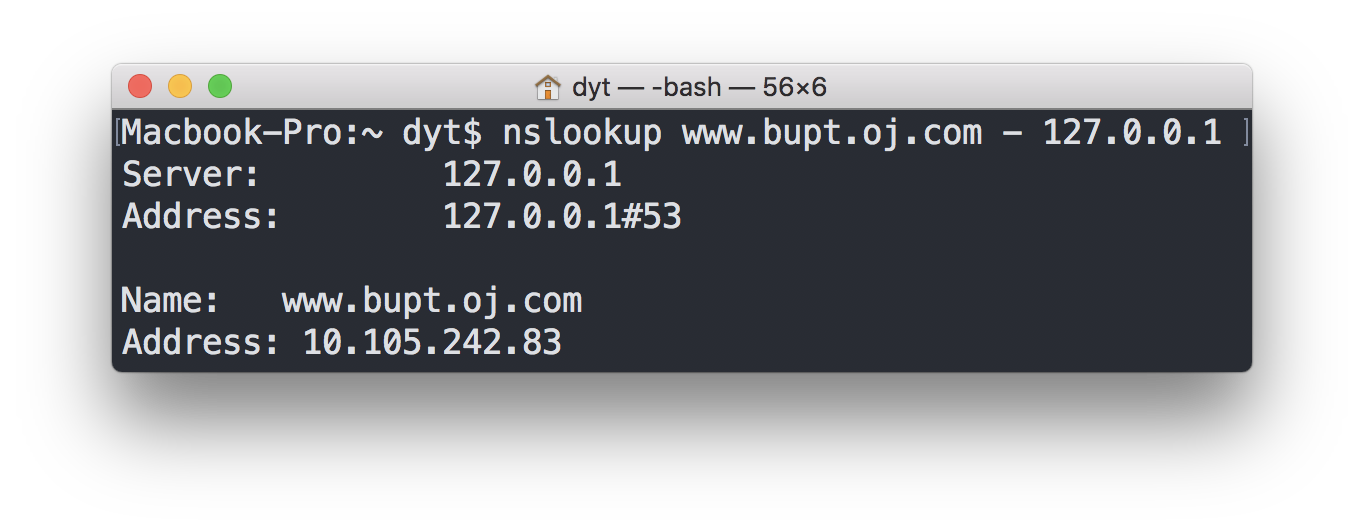
* 1. 在多台本地 PC 中正常进行浏览，检查程序 debug 输出；

测试结果：

第一阶段

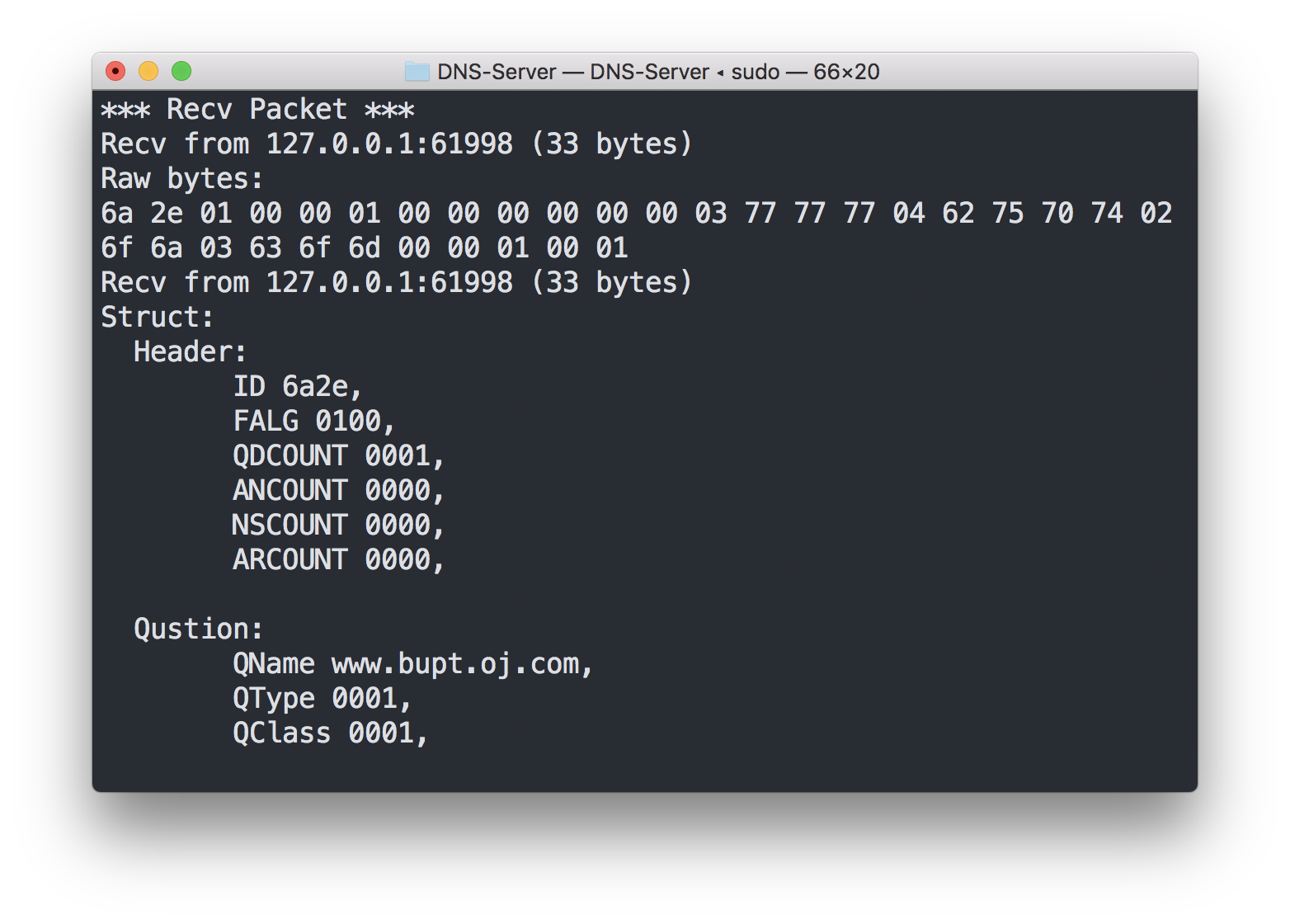
* 1. 命令 nslookup www.bupt.oj.com - 127.0.0.1，测试 DNS 服务器功能；

客户端：

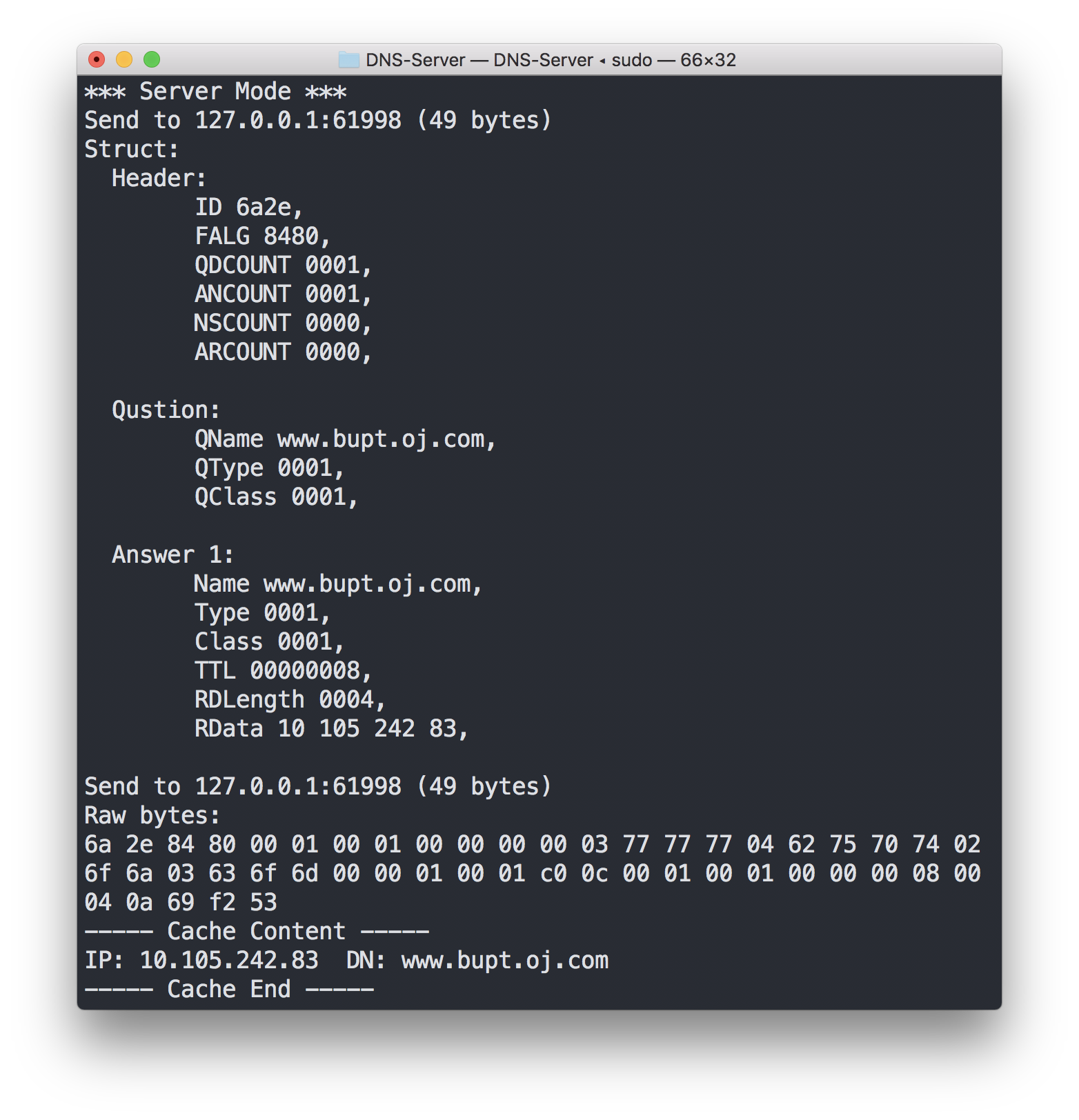


服务端：

接收客户端查询请求报文部分：

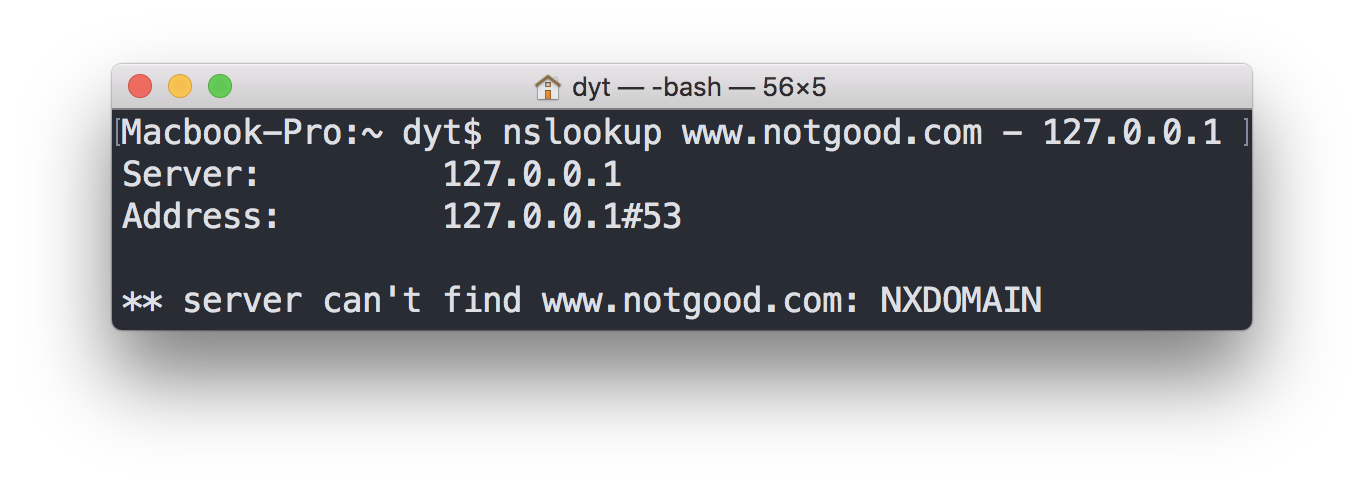


DNS 服务器模式部分 & LRUCache 部分：



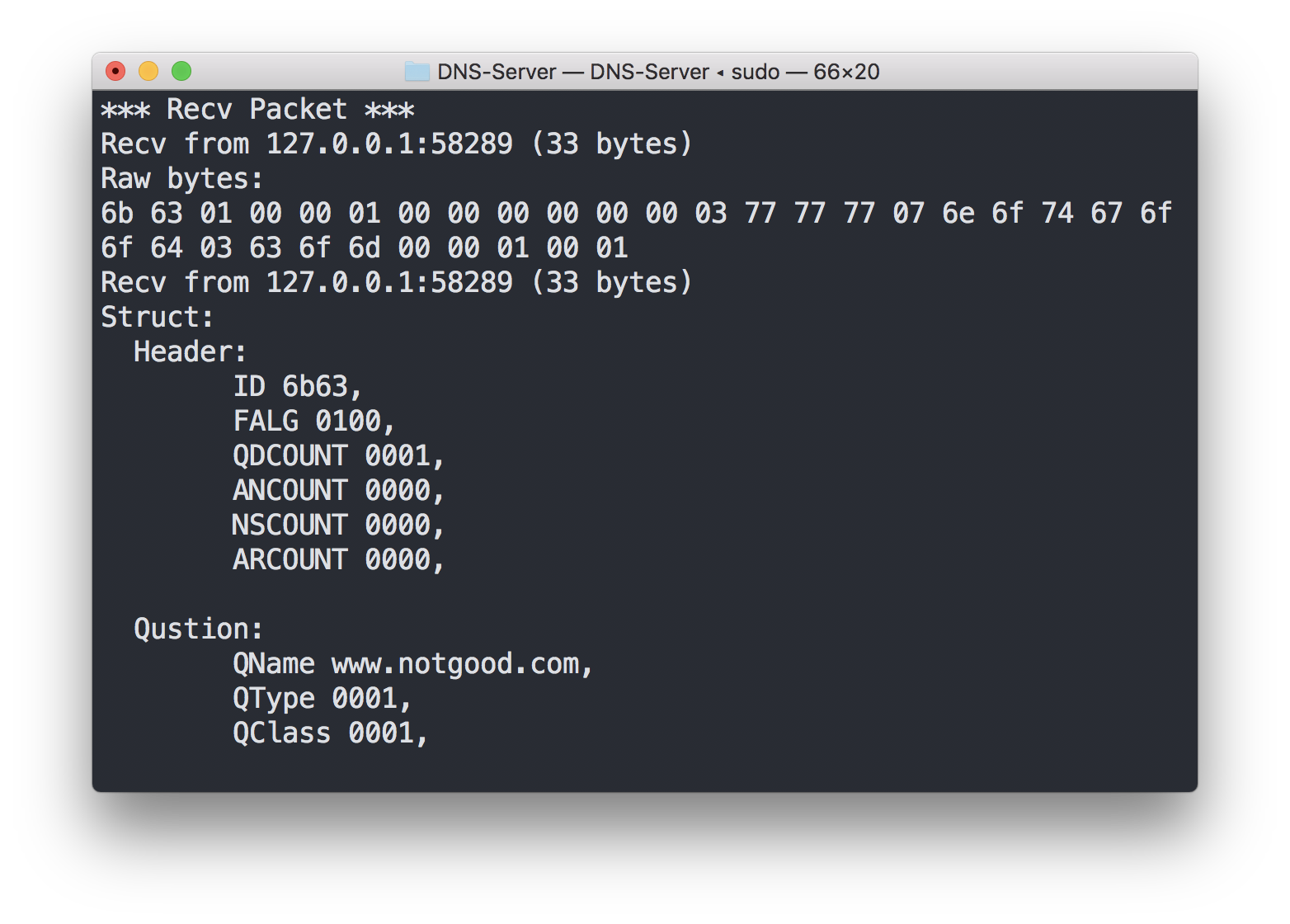
b. 命令 nslookup [www.notgood.com](http://www.notgood.com) - 127.0.0.1，测试不良网站拦截功能;

客户端：

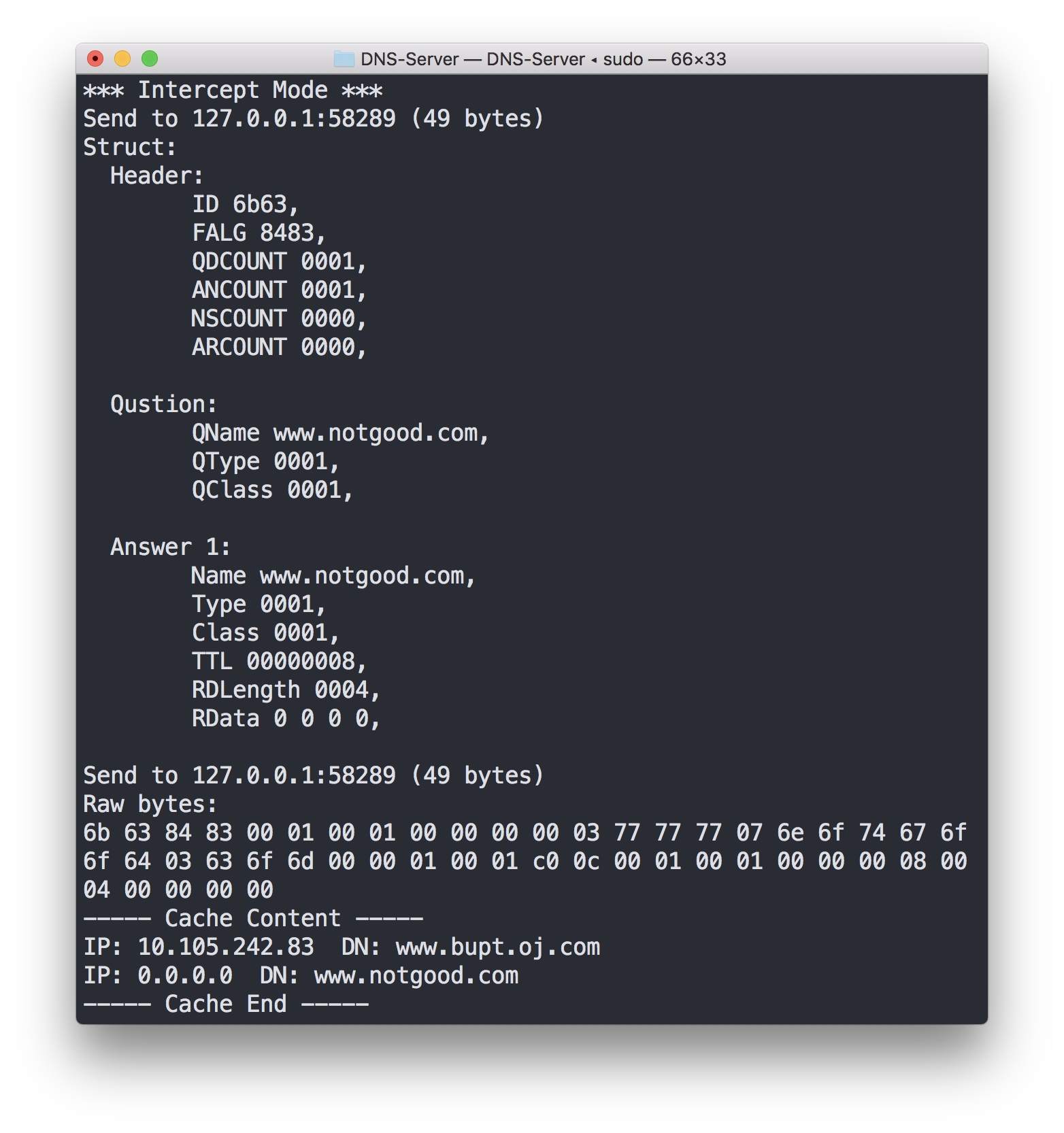


服务端：

接收客户端查询请求报文部分：

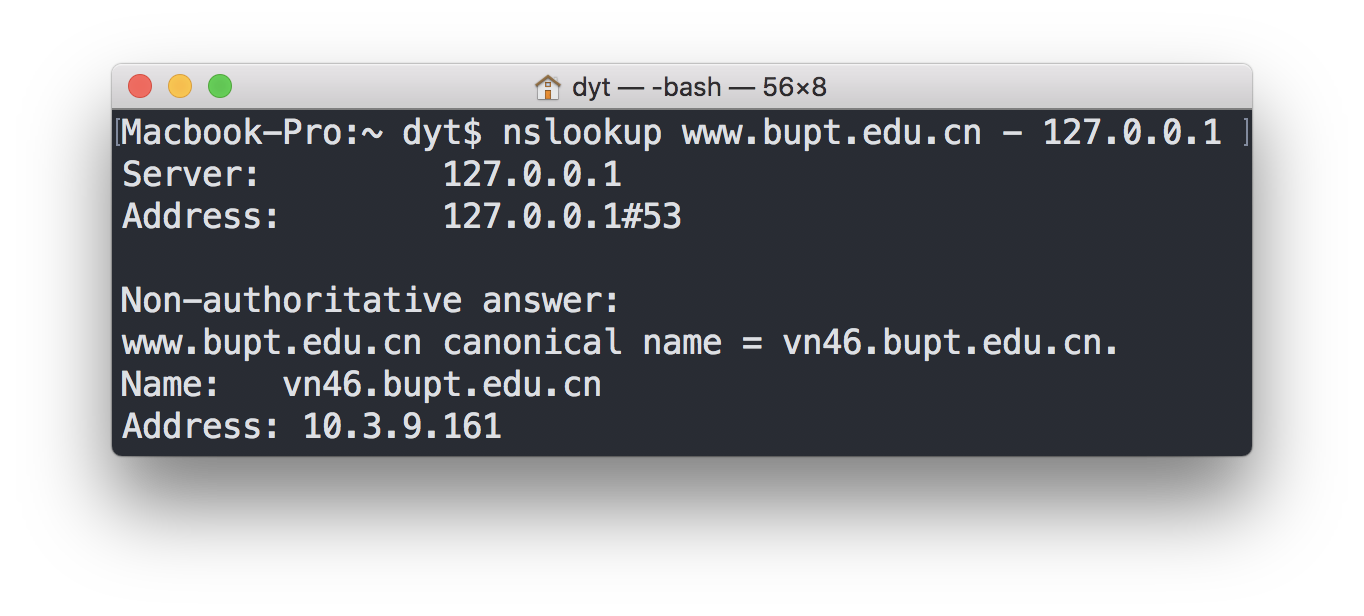


不良网站拦截模式部分 & LRUCache 部分：



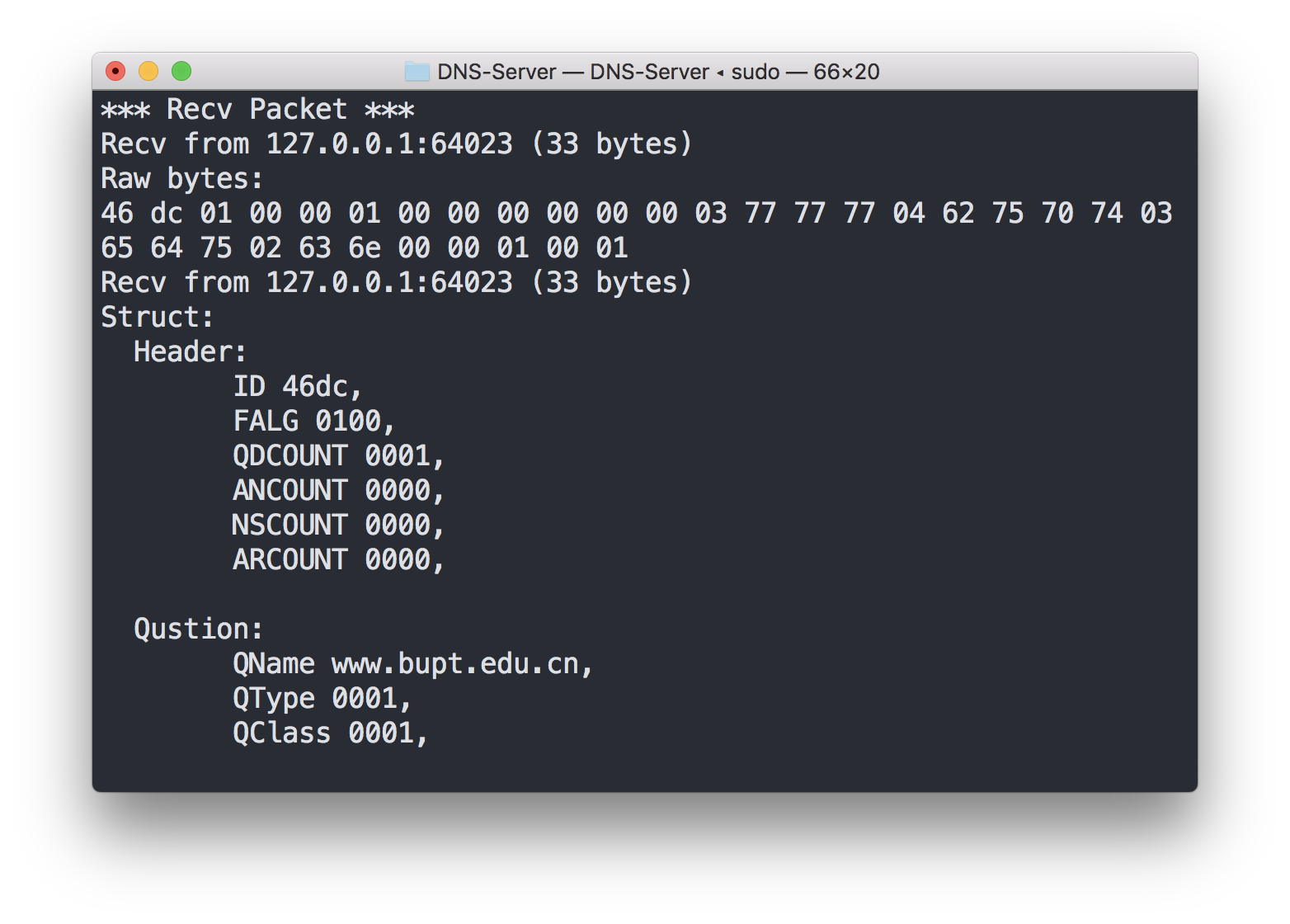
c.命令 nslookup [www.bupt.edu.cn](http://www.bupt.edu.cn) - 127.0.0.1，测试中继功能;

客户端：

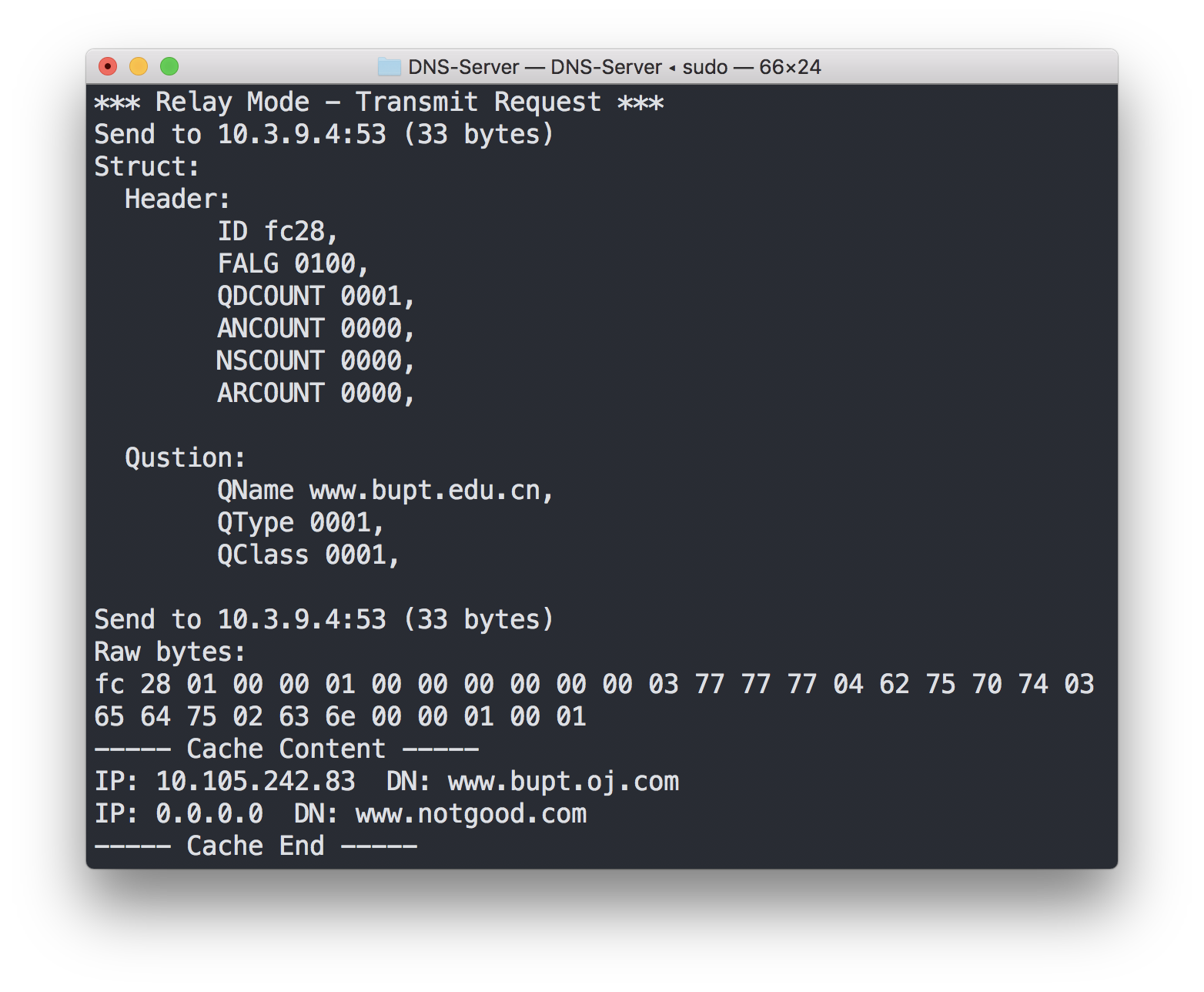


服务端：

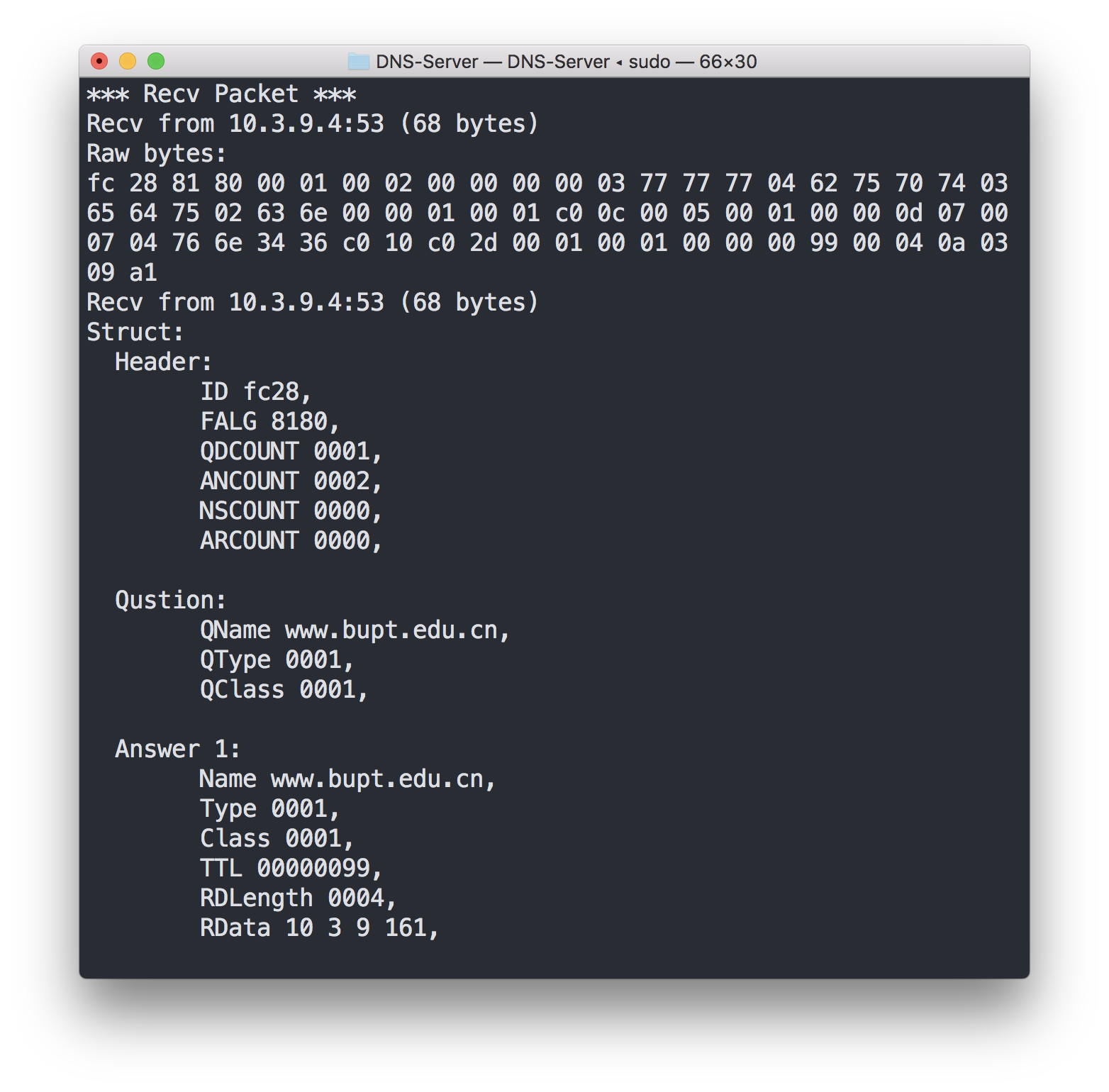
接收客户端查询请求报文部分：



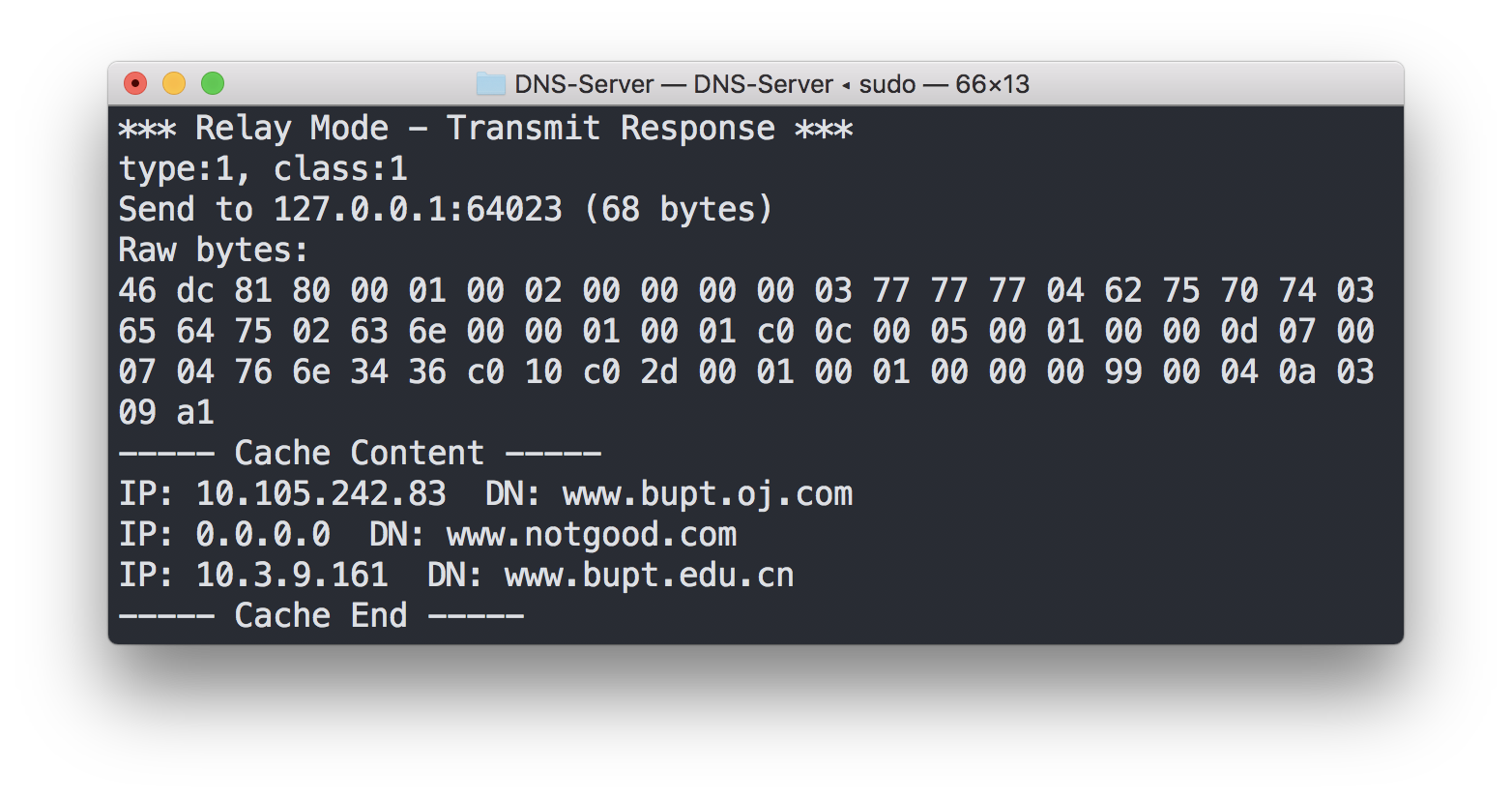
中继模式-转发客户端查询请求报文部分 & LRUCache 部分：



中继模式-接收 Internet DNS 查询响应报文部分 & LRUCache 部分：

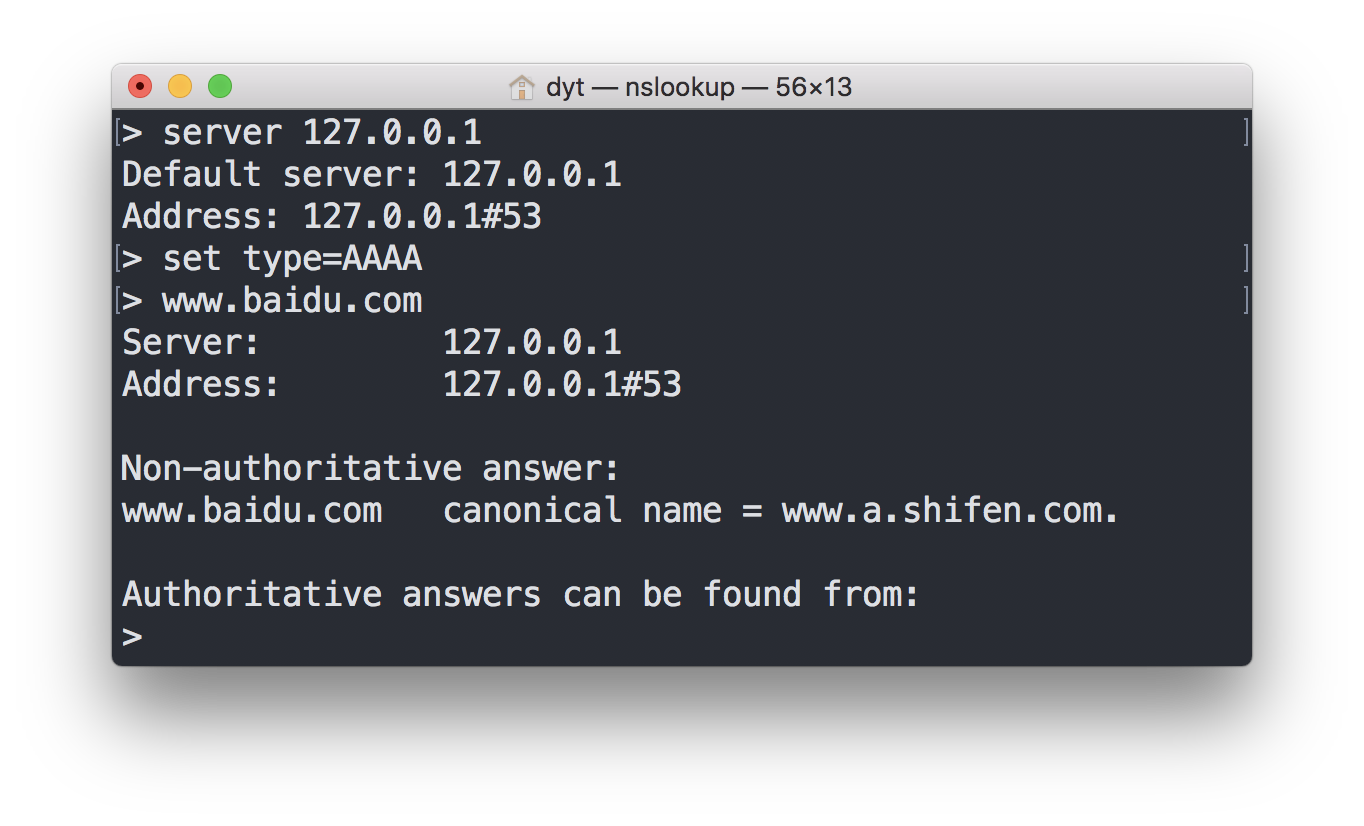


中继模式-转发 Internet DNS 查询响应报文部分 & LRUCache 部分：



d.设置 nslookup 的查询模式为查询 iPv6 模式，查询 [www.baidu.com](http://www.baidu.com) 的 IP 地址，测试 iPv6 查询的兼容功能;

客户端：

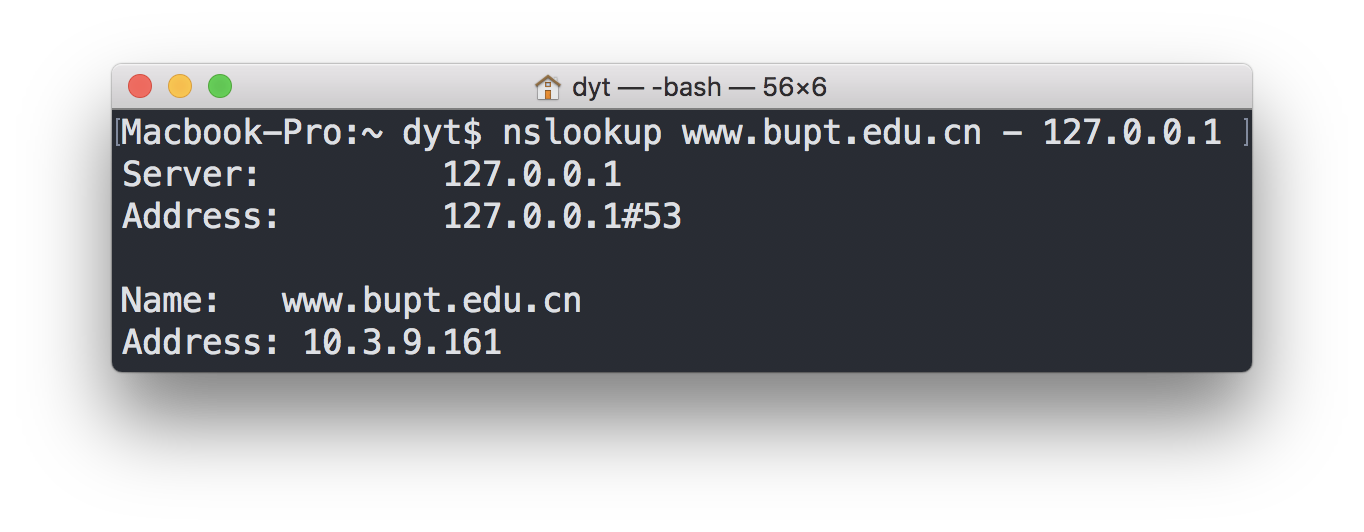


服务端：

同中继模式，略。

e.再次命令 nslookup [www.bupt.edu.cn](http://www.bupt.edu.cn) - 127.0.0.1，测试 LRUCache 功能;

客户端：



服务端：

接收客户端查询请求报文部分：



使用 LRUCache 的 DNS 服务器模式部分 & LRUCache 部分：



第二阶段：

* 1. 正常进行浏览，检查程序 debug 输出；

结果：

可进行正常的网页浏览。

任选一次浏览记录的部分服务端 debug 输出：

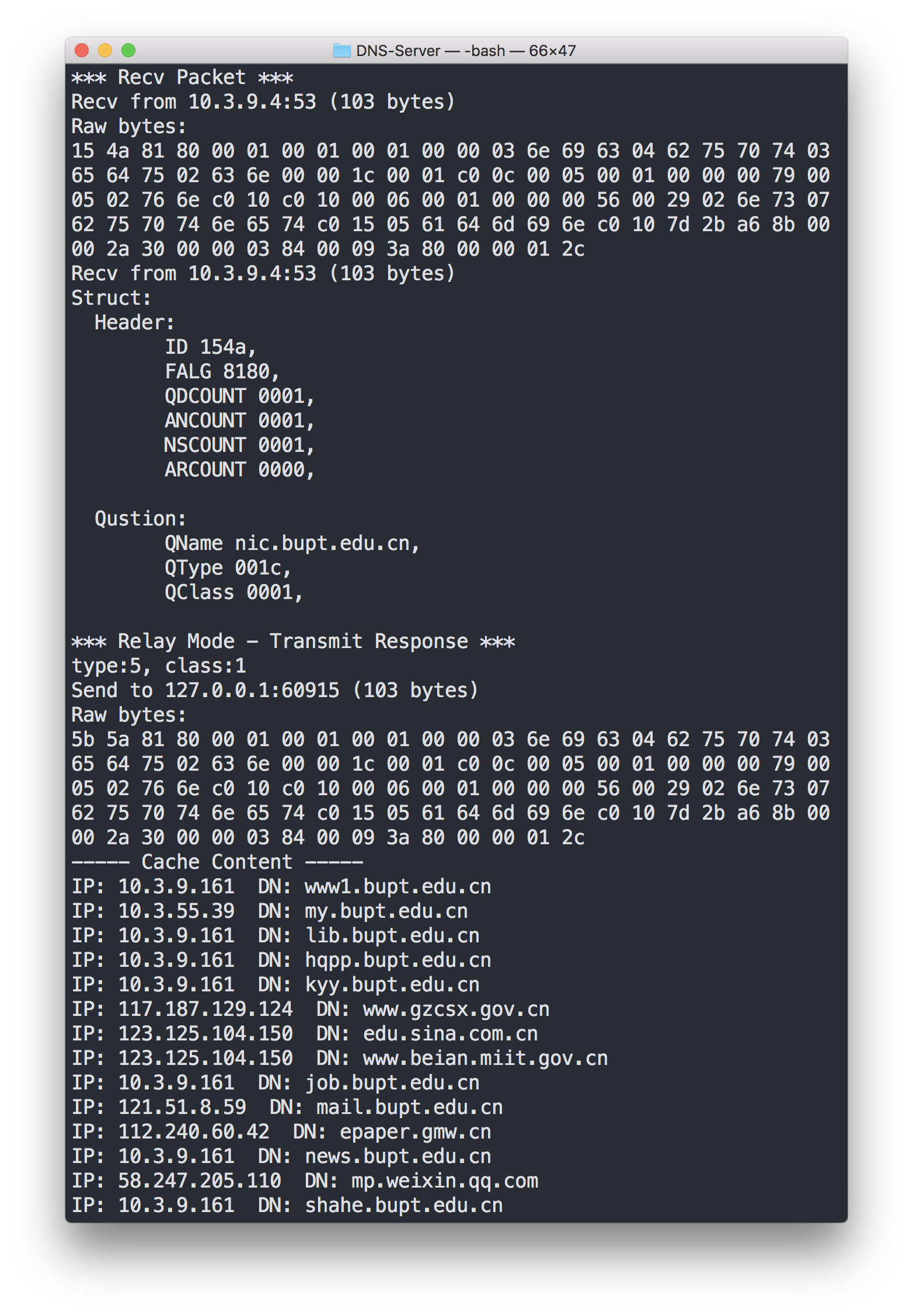


* 1. 浏览器地址栏键入 [www.bupt.edu.cn](http://www.bupt.edu.cn) 并打开；

结果：

可进行正常的网页浏览。

部分服务端 debug 输出：



第三阶段

* 1. 在多台本地 PC 中正常进行浏览，检查程序 debug 输出；

结果：

可进行正常的网页浏览。

服务端 debug 输出：

与第二阶段的测试结果相似。

6、调试中遇到的问题与疑惑

**问题：WinOS下无法成功绑定53端口**

原因：

* 53端口被scvhost进程占用

解决：

* 结束占用53端口的scvhost进程
* 添加initWSA函数，初始化网络环境

**问题：转发应答时无法找到原对应的客户端**

原因：

* 由于会存在多个客户端，不存储客户端的socket 地址会无法发回给正确的客户端

解决：

* 在ID转换表中加入客户端的socket 地址信息，这样在查表后既找到原报文ID号，又可以查询到客户端的socket 地址，直接发给客户端即可

**问题：IPv6查询域名时LRUCache中会存入一条IP-域名表项，而且该IP与前一表项的IP相同**

原因：

该问题本质在于对LRUCache增加表项时未进行IPv4的判断，只有IPv4的查询下才能将IP-域名进行缓存。对于IPv6的查询，我们统一进行转发处理，并不会存入缓存池中。

**疑惑：使用Wireshark抓取DNS应答包时，Answers部分中Name域字节为C00C**

原因：DNS协议使用偏移指针代替重复的字符串达到消息压缩的效果，该指针用两个8bit表示具体规定如下：

+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

| 1 1| OFFSET |

+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

最开始的两个bit必须都为1，目的是为了避免混淆。后面的14bit表示字符串在整个DNS消息包中的偏移量。

**问题：在将**[www.bupt.oj.com](http://www.search.com)**在txt文件中重定向为北邮的OJ系统IP地址10.105.242.83时，使用nslookup、ping等命令均可得到我们人为设置好的IP地址，但用浏览器输入该域名后，网页会显示bad request，并不能成功登录我们期望看到的界面**

原因：浏览器会设置过滤规则，若出现域名和IP不一致的情况时，它不会正常响应。这种机制是为了防止**DNS污染**，所以这种操作并不能实现。

7、总结和心得体会

**王子澄：**

这次实验令我在许多方面受益匪浅。

首先，这次实验给了我一个将理论知识和实践相结合的机会，使我更加深入地理解了DNS协议的运行机制。对于DNS报文的Header、Question和Answer部分，我详细研究了每个字段的作用。对于DNS中继器对报文的三种处理方式，我也有了清晰的认识。这些知识都是设计程序的基础。

其次，运用纯C语言编写程序让我巩固了底层程序的编写方法。由于不能使用C++、python等库函数，我复习了链表、string库函数的用法。因为很久不用，一开始我对于它们的用法的确有些生疏。尤其是在写lru缓存池时，由于一个链表的逻辑错误，我们的程序出现了bug，困扰了我们很久。好在最终我们解决了bug，也意识到了链表逻辑的重要性。

最后，在这次小组分工合作中，我从队友身上学到了很多技巧，从用命令行调用gcc编译程序到代码版本管理的方法。这些技巧在实际开发软件过程中十分实用，也是这次课程设计带给我的一大收获。

**邓宇涛：**

本次实验采用相对较为底层的C语言进行编程实现DNS中继服务器，需要考虑诸多网络通信中的细节，例如网络字节顺序与主机字节顺序的一致性控制、字节流中不同字段的读取等，更考验对协议的理解程度训练了我们思维的完备性。

通过本次实验，在计算机网络的知识方面，我大大加深了对DNS协议机制的理解与对UDP报文传输的理解。在编程语言方面也进一步提高了C语言的编写能力。在调试程序的过程中，我除了在程序中加入调试信息，还运用了Wireshark软件抓包来辅助调试。对于网络相关程序，Wireshark的捕获和分析功能对于测试程序十分有用，可以直观、迅速地为我们呈现报文的信息。通过对比一般DNS服务器的报文与我们设计的中继DNS服务器的报文，我们能更快速的获取到报文的标准格式与调试过程中的报文差错。在程序的测试过程中，我们还认识到了校园网内的AP隔离、防DNS污染等等网络技术，拓展了对我们的网络知识。

**张天秋：**

DNS域名系统是将主机名字与IP进行互相转换的一种分布式数据库，我们在这次课设中实现了一个较为初级的DNS中继服务器，对于身处应用层的DNS在网络通信的方式和位置有了深刻的理解。依靠RFC 1035文档细致的描述，我们可以将所学的计网知识更灵活地运用到实践编程之中，而socket通信提供的接口使得我们将注意力集中在中继服务器本身的设计上。

我主要负责了报文结构体的编写和服务器工作模块中的中继服务器模块编写，由于后期版本相比早期版本有不少功能添加，导致早期的中继模块有很多漏洞。每一次的调试排错，我对于DNS中的细节处理的认识就更为深入。令我印象深刻的一个bug就是使用chrome浏览器时，它会发送IPv4和IPv6的包，而由于一开始中继模块对于IPv6并没有进行处理，导致LRU缓存池会存入奇怪的缓存。后来和队友通过抓包分析出IPv6的包的应答格式后，完善了模块编写，顺利解决了bug。

本次的实验我们通过小组合作的方式顺利地完成了项目，依托于git的远程版本管理，我们得以在疫情居家期间可以无阻碍的对项目进行修改共享；对于不同操作系统及不同编译器带来的一系列编译不兼容问题，我们最终统一选择gcc对程序进行编译，这样减少了我们对于程序逻辑之外的关注程度；在最后的调试阶段，我们分别在MacOS、WinOS以及配置为Ubuntu系统的腾讯云服务器上运行了我们的程序，最后都成功运行，并在最终选择界面简洁的MacOS作为验收时的主机器。