实现 DNS 中继服务器

系统的功能设计

设计与实现一个 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
- 检索结果为普通 IP 地址
- 表中未检到该域名

根据这三种检索结果需要实现的功能描述如下:

不良网站拦截功能

当收到本地进程的查询报文时,通过检索域名-IP 地址中的条目可判定查询的域名是否为不良网站,如果检索到的 IP 地址是 0.0.0.0,则实施拦截.实施的方式是将回答报文中的响应码字段(rcode)置为 3,即返回一个表示” 域名不存在” 的回答报文,

服务器功能

如果在对照表中检索到了查询报文询问的域名,则无需询问 DNS 服务器,直接将表中的 IP 地址组装入回答报文发送给询问进程即可.

中继功能

当对照表中不存在询问的域名时,检索不会得到结果.此时需要将查询报文转化后转发给 DNS 服务器,之后将得到的回答报文再次转化后发给询问进程,实现中继.

此外,还需要考虑的两个问题:

多客户端并发

允许多个客户端并发查询,即允许第一个查询尚未得到答案前就启动处理另一个客户端查询请求.这是由 ID 转换模块实现的,详见下文.

超时处理

由于 UDP 的不可靠性,考虑求助外部 DNS 服务器(中继)却不能得到应答或者收到迟到答应的情形.

在我们实现的请求池中,每个请求创建时都会带有预设的过期时间,过期的条目条目将被清除.当接收到迟到答应时,由于请求池中的条目已被清除,该答应会被丢弃.

模块划分和软件流程图

程序划分为解析控制模块(主模块),缓存模块, ID 转换模块,静态表模块.

解析控制模块

即程序的 main 函数,在其中进行命令行参数的解析,接收报文并调用其它模块的接口进行处理,以及各模块的初始化与最后的清理.

支持的命令行参数:

- -d | -dd 指定调试级别 1 或 2,默认为 0
- -a <dns-server-ipaddr> 指定一个 DNS 服务器,默认为 114.114.114
- -f <filename> 指定静态表文件,默认为 dnsrelay.txt

缓存模块

我们实现了 LRU 缓存机制,规则如下:

- 每当从对照表中或从 DNS 服务器得到查询结果且结果为 A 类型时,将此结果插入到缓存队列头部
- 每当接收到查询报文时,先遍历缓存队列,若存在查询的域名的结果则回答之,并将该条目移动到缓存队列头部;否则查找对照表,若检索到,则回答之,并将该条目从缓存队列头部插入;如果仍未检索到,最终会触发中继功能
- 缓存队列中的条目均有 TTL 字段,每当遍历缓存队列时,若发现过期,则删除该条目
- 当向缓存队列插入新条目时,如果超出了预设的缓存大小,则不断删除队列尾部的条目直至队列大小不超过预设缓存大小

我们使用双向链表 list_head 实现缓存队列,链表的实现是在

github.com:torvalds/linux/include/linux/container.h&list.h 的基础上稍加修改后得到的.

ID 转换

为了实现多客户端并发查询,采用了一种 ID 转换算法.只有在触发中继功能时才需要进行 ID 转换.

由于本地的多个进程都会发来查询报文,并且这些报文的 ID 字段是随机产生的,所以存在 ID 冲突的可能性.对于冲突的 ID 需要进行转换,具体规则如下:

- 设置一个大小为 2^16 的请求池,以 0~2^16 -1 为索引
- 每当接收到一个 ID 为 x 的查询报文时,申请请求池的第 x 项,若第 x 项已被占据,则依次尝试第 x+1,x+2,x+3…项,直到成功申请到第 y 项,修改 ID 为 y 后将此报文转发给 DNS 服务器,并将请求的信息记录下来,这包括原始 ID 的值 x
- 每当接受到一个 ID 为 y 的回答报文时,查询请求池的第 y 项,这包含了原始 ID 的值 x 和查询进程的地址,将回答报文的 ID 修改为 x 后发送给此进程

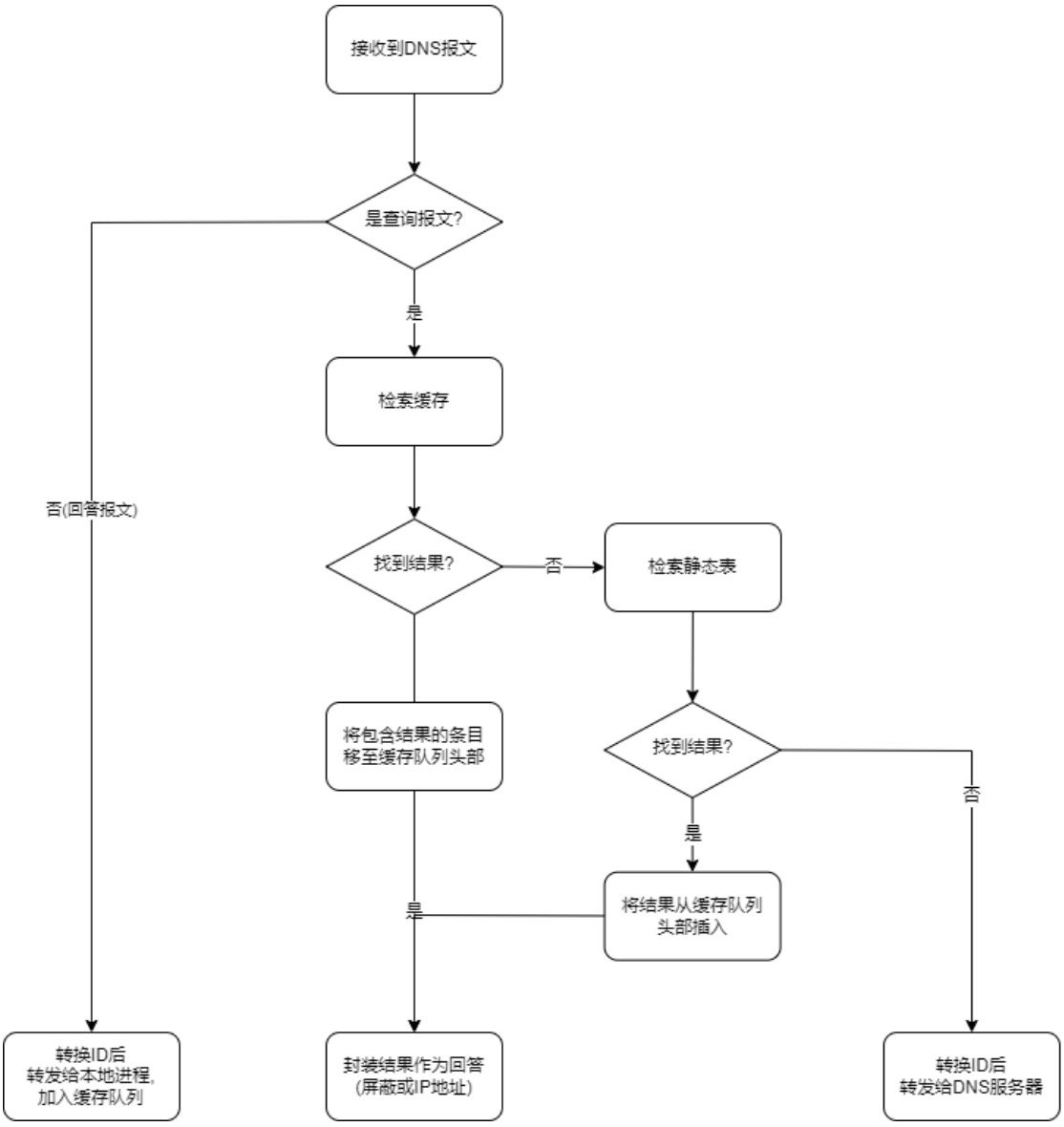
静态表检索

申请一块连续的内存,读入静态表,以域名为关键字,按字典序排序,检索时进行二分查找.

申请的内存大小通过这种方式决定:打开静态表文件(默认为 dnsrelay.txt)后,遍历至 EOF,检查出文件的行数,认定行数就是静态表的条目数,以此得出应该申请的内存大小.

*注意到,上述大部分功能只对 A 类型的 DNS 查询报文产生作用.*为了让主机获得正常的 DNS 服务,我们的 DNS 中继服务器还必须同时处理好其它类型的查询报文.处理方式是将这些报文视作未在对照表中检索到结果,通过中继功能转发给 DNS 服务器,之后将回答转发给询问进程.

软件流程图



测试用例以及运行结果

1. 本地测试

将本机的 DNS 首选服务器设置为 127.0.0.1,运行程序.

Windows:dnsrelay.exe -d

```
17542.129  A type: A     from: 114.114.114.114  ID: 12250 <- 12251     catalog.gamepass.com <- 23.1.236.33
17542.834  Q type: A     from: 127.0.0.1        ID: 1340 -> 1340       www.google.com -> ?
17542.879  Q type: A     from: 127.0.0.1        ID: 1340 -> 1341       www.google.com -> ?
17542.957  A type: A     from: 114.114.114.114  ID: 1340 <- 1340       www.google.com <- 172.217.160.68
17542.959  A type: A     from: 114.114.114.114  ID: 1340 <- 1341       www.google.com <- 172.217.160.68
17543.956  Q type: A     from: 127.0.0.1        ID: 12710 -> 12710     webvpn.bupt.edu.cn -> ?
17543.958  Q type: A     from: 127.0.0.1        ID: 12989 -> 12989     safebrowsing.google.com -> ?
17543.988  Q type: A     from: 127.0.0.1        ID: 12710 -> 12711     webvpn.bupt.edu.cn -> ?
17543.990  A type: A     from: 114.114.114.114  ID: 12989 <- 12989     safebrowsing.google.com <- 172.217.163.46
17544.021  A type: A     from: 114.114.114.114  ID: 12710 <- 12711     webvpn.bupt.edu.cn <- 211.68.69.226
17544.049  A type: A     from: 114.114.114.114  ID: 12710 <- 12710     webvpn.bupt.edu.cn <- 211.68.69.226
17546.655  Q type: A     from: 127.0.0.1        ID: 22581 -> 22581     login.microsoftonline.com -> ?
17546.682  A type: A     from: 114.114.114.114  ID: 22581 <- 22581     login.microsoftonline.com <- 20.190.144.163
```

Linux:sudo ./dnsrelay -d

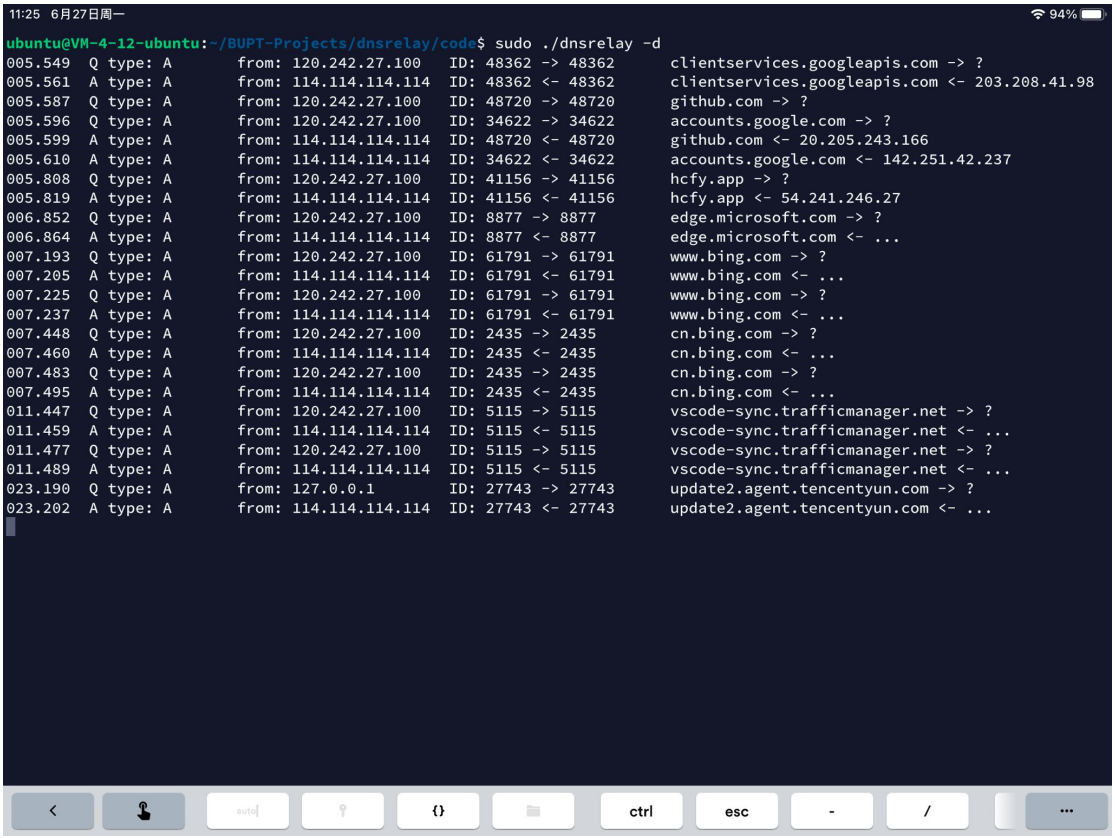
```
027.152  Q type: 28      from: 127.0.0.1        ID: 5202 -> 5202       baidu.com -> ?
027.167  A type: A       from: 114.114.114.114  ID: 4359 <- 4359       baidu.com <- 220.181.38.148
027.167  A type: 28      from: 114.114.114.114  ID: 5202 <- 5202       baidu.com <- ...
027.220  Q type: 12      from: 127.0.0.1        ID: 25376 -> 25376     148.38.181.220.in-addr.arpa -> ?
027.233  A type: 12      from: 114.114.114.114  ID: 25376 <- 25376     148.38.181.220.in-addr.arpa <- ...
028.207  Q type: 12      from: 127.0.0.1        ID: 48840 -> 48840     148.38.181.220.in-addr.arpa -> ?
028.220  A type: 12      from: 114.114.114.114  ID: 48840 <- 48840     148.38.181.220.in-addr.arpa <- ...
029.205  Q type: 12      from: 127.0.0.1        ID: 45121 -> 45121     148.38.181.220.in-addr.arpa -> ?
029.221  A type: 12      from: 114.114.114.114  ID: 45121 <- 45121     148.38.181.220.in-addr.arpa <- ...
029.955  Q type: A       from: 127.0.0.1        found in cache         baidu.com -> ? <- 220.181.38.148
029.955  Q type: 28      from: 127.0.0.1        ID: 2591 -> 2591       baidu.com -> ?
029.971  A type: 28      from: 114.114.114.114  ID: 2591 <- 2591       baidu.com <- ...
030.012  Q type: 12      from: 127.0.0.1        ID: 11520 -> 11520     148.38.181.220.in-addr.arpa -> ?
030.025  A type: 12      from: 114.114.114.114  ID: 11520 <- 11520     148.38.181.220.in-addr.arpa <- ...
030.716  Q type: A       from: 127.0.0.1        found in cache         baidu.com -> ? <- 220.181.38.148
030.716  Q type: 28      from: 127.0.0.1        ID: 10664 -> 10664     baidu.com -> ?
030.731  A type: 28      from: 114.114.114.114  ID: 10664 <- 10664     baidu.com <- ...
030.773  Q type: 12      from: 127.0.0.1        ID: 13755 -> 13755     148.38.181.220.in-addr.arpa -> ?
030.786  A type: 12      from: 114.114.114.114  ID: 13755 <- 13755     148.38.181.220.in-addr.arpa <- ...
032.076  Q type: A       from: 127.0.0.1        found in cache         baidu.com -> ? <- 220.181.38.148
032.076  Q type: 28      from: 127.0.0.1        ID: 53551 -> 53551     baidu.com -> ?
032.093  A type: 28      from: 114.114.114.114  ID: 53551 <- 53551     baidu.com <- ...
032.160  Q type: 12      from: 127.0.0.1        ID: 21227 -> 21227     148.38.181.220.in-addr.arpa -> ?
032.181  A type: 12      from: 114.114.114.114  ID: 21227 <- 21227     148.38.181.220.in-addr.arpa <- ...
035.051  Q type: A       from: 127.0.0.1        ID: 12163 -> 12163     updatecenter.qq.com -> ?
035.051  Q type: 28      from: 127.0.0.1        ID: 12777 -> 12777     updatecenter.qq.com -> ?
035.071  A type: A       from: 114.114.114.114  ID: 12163 <- 12163     updatecenter.qq.com <- 111.30.176.92
035.071  A type: 28      from: 114.114.114.114  ID: 12777 <- 12777     updatecenter.qq.com <- ...
041.232  Q type: A       from: 127.0.0.1        found in cache         updatecenter.qq.com -> ? <- 111.30.176.92
```

在 Windows 下连续运行四个多小时(行首数字表示运行秒数),期间正常使用电脑,没有任何异样的体验.在 Linux 环境下同样正常运行.

经过手动测试,屏蔽功能运行正常.

```
C:\Users\huawei>nslookup
默认服务器:  UnKnown
Address:  127.0.0.1

> cctv1.net
服务器:  UnKnown
Address:  127.0.0.1

*** UnKnown 找不到 cctv1.net: Non-existent domain
> test0.cn
服务器:  UnKnown
Address:  127.0.0.1

*** UnKnown 找不到 test0.cn: Non-existent domain
> test1.cn
服务器:  UnKnown
Address:  127.0.0.1

非权威应答:
名称:    test1.cn
Address:  11.111.11.111

> baidu.com
服务器:  UnKnown
Address:  127.0.0.1

非权威应答:
名称:    baidu.com
Address:  220.181.38.148
```

2. 远程部署测试



通过将程序部署到服务器,并将本机的 DNS 首选服务器设置为服务器的 IP 地址(124.220.210.31)来测试 socket 编程的正确性.

上图是服务器的运行结果,除了最后一对询问与回答来自服务器自身(127.0.0.1),其它的询问来源均为本机(120.242.27.100).

测试成功,没有出现问题.

调试中遇到并解决的问题

1. 操作系统对 DNS 的缓存

我们意外地发现,在 Windows 下短时间内多次 ping 同一个域名时,程序只会收到一次 DNS 查询报文,而在 Linux 下则是每次都会收到,进而在程序实现的缓存中找到回答.

据此推测,Windows 自身也会保留 DNS 缓存,实践后发现如果每次 ping 后都使用 ipconfig/flushdns,程序接收到的报文就与 Linux 下的情况一致了.

最后,根据查找到的资料(https://stackoverflow.com/questions/11020027/dns-caching-in-linux),Windows 是带有系统级别的 DNS 缓存的,而我们测试所用的 Linux 发行版并不带有这一功能,这就能解释观察到的现象了.

2. 对 CNAME 类型报文进行解析的尝试

尽管课程提出的要求只需要通过解析 A 类型报文就可以实现,我们最初还是尝试了对 CNAME 类型进行处理,认为只需要添加一种新的条目类型,与 A 类型进行类似的缓存,检索等操作就可以了.

然而当我们把 CNAME 回答报文中的 RR 字段内容提取出来,参考 RFC 将其转化为人类可读的格式时,发现经常会出现错误.后来查找到一篇对 DNS 报文格式进行详细分析的文章

(https://cabulous.medium.com/dns-message-how-to-read-query-and-response-message-cfebcb4fe817),这才发现为了节省资源,CNAME 类型回答报文的 RR 字段使用了一种优化的存储方式,我们的实现并没有考虑这一点.

最后我们放弃了这一尝试,继续实现其它功能.

总结和心得体会

这是一次很好的网络编程实践经历.

实现这样的一个中继器程序,既要了解 DNS 的工作方式与相关细节,又要依赖 socket 编程进行实现.而在这学期之初,我们小组的三个人在相关领域的知识几乎是空白的状态,既不懂计算机网络的层次结构,也没有什么网络编程方面的经验.但好在时间比较充裕,计算机网络方面的知识在课堂上学习,而在网络编程方面,我们通过学习和参考<<Linux 高性能服务器编程>>(游双)和<<TCP/IP 网络编程>>(尹圣雨)两本书很快就达到了足够的水平.

为了实现方便的跨平台部署,还学习了如何运用宏定义实现条件编译,以及运用类似的方法编写 Makefile,使得编译更加方便.

这些经历大部分对于我们而言都是头一次,所以学到了不少新技术,收获很大.