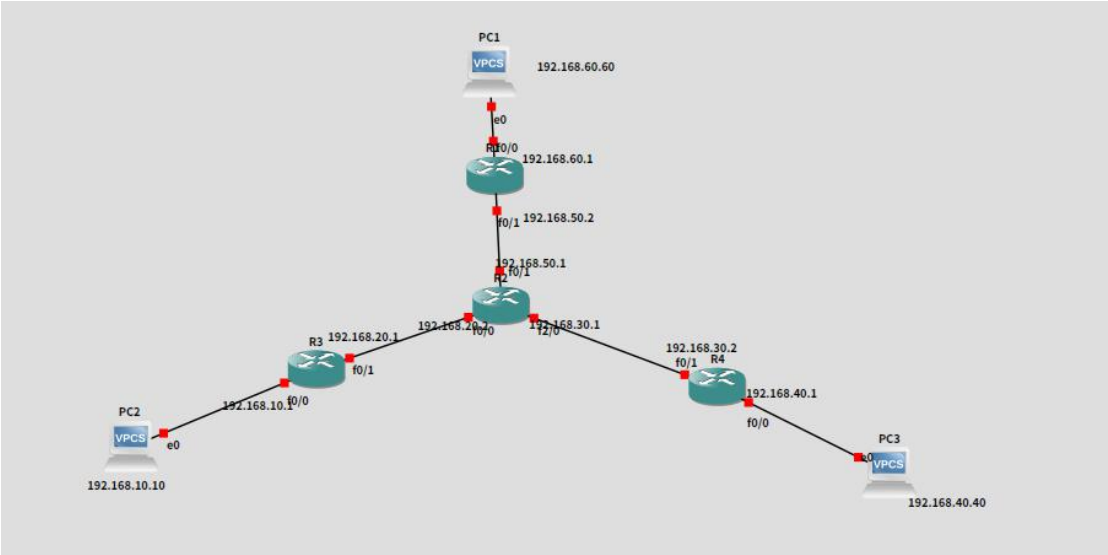
1．准备工作

（1）GNS3 搭建的网络拓扑用到的模拟设备有 4 台 c7200 路由器 R1、R2、R3、R4，3 台 PC 机 PC1、PC2、PC3，网络拓扑结构图如图



（2）R1、R2、R3、R4 的接口和静态路由配置，同时配置 PC 的 IP 地址

```
Gateway of last resort is 192.168.30.1 to network 0.0.0.0

C    192.168.30.0/24 is directly connected, FastEthernet0/1
S    192.168.60.0/24 [1/0] via 192.168.30.1
S    192.168.10.0/24 [1/0] via 192.168.30.1
C    192.168.40.0/24 is directly connected, FastEthernet0/0
S*   0.0.0.0/0 [1/0] via 192.168.30.1
R4(config)#do show ip int b
Interface                  IP-Address      OK? Method Status                Prot
ocol
FastEthernet0/0            192.168.40.1    YES NVRAM  up                    up

FastEthernet0/1            192.168.30.2    YES NVRAM  up                    up
```

（3）运行起来后将 GNS3 和 Wireshark 关联起来，捕获其中一条链路，保存为 pcap 文件，以便作为输入模块中传入的文件，同时保存路由表为 route 文件。

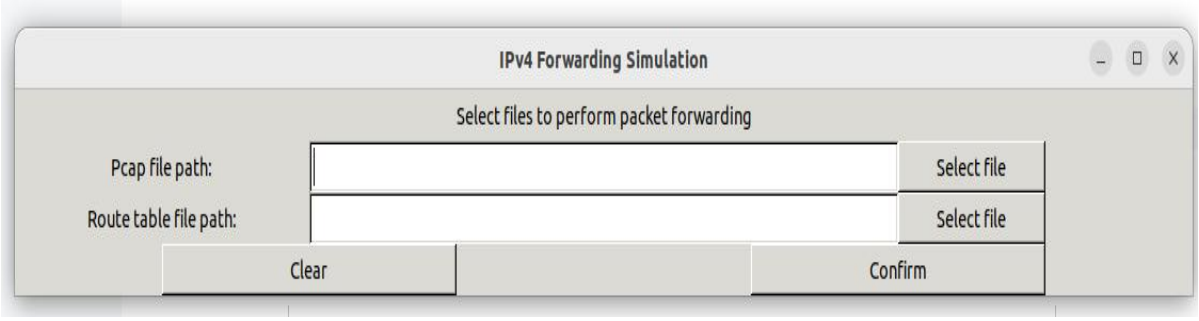
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.40.40	192.168.10.10	ICMP	98	Echo (ping) request id=0x1dc6, seq=1/256, ttl=64 (reply in 2)
2	0.040155	192.168.10.10	192.168.40.40	ICMP	98	Echo (ping) reply id=0x1dc6, seq=1/256, ttl=61 (request in 1)
3	1.041286	192.168.40.40	192.168.10.10	ICMP	98	Echo (ping) request id=0x1ec6, seq=2/512, ttl=64 (reply in 4)
4	1.076580	192.168.10.10	192.168.40.40	ICMP	98	Echo (ping) reply id=0x1ec6, seq=2/512, ttl=61 (request in 3)
5	2.077249	192.168.40.40	192.168.10.10	ICMP	98	Echo (ping) request id=0x1fc6, seq=3/768, ttl=64 (reply in 6)
6	2.113258	192.168.10.10	192.168.40.40	ICMP	98	Echo (ping) reply id=0x1fc6, seq=3/768, ttl=61 (request in 5)
7	3.114274	192.168.40.40	192.168.10.10	ICMP	98	Echo (ping) request id=0x20c6, seq=4/1024, ttl=64 (reply in 8)
8	3.149847	192.168.10.10	192.168.40.40	ICMP	98	Echo (ping) reply id=0x20c6, seq=4/1024, ttl=61 (request in 7)
9	4.150269	192.168.40.40	192.168.10.10	ICMP	98	Echo (ping) request id=0x21c6, seq=5/1280, ttl=64 (reply in 10)
10	4.186090	192.168.10.10	192.168.40.40	ICMP	98	Echo (ping) reply id=0x21c6, seq=5/1280, ttl=61 (request in 9)

2. 功能模块

分为四个模块

1. 图形界面:选择文件按钮,清空按钮,确认按钮

软件在输入模块中解析 pcap 文件和获取路由信息，需要用户自行选择文件路径。页面设计两个选择文件路径的输入框，分别是 pcap 文件路径输入框和路由表文件路径输入框，选择文件路径可以展示当前文件夹内所有文件，以及可以跳转文件夹。为防止用户选择路径出错，设计清除输入框内路径的按钮。也需要设计确认按钮，当用户正确选择文件路径后，进行确认后，才可将文件路径传入输入模块。



2. 输入模块:解析 pcap 文件,解析存储路由信息的文件

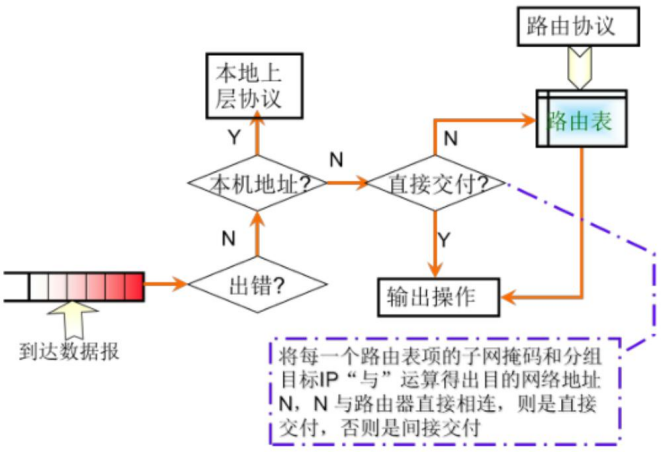
输入模块完成了对 pcap 文件的处理，将 pcap 文件里的数据解析到结构体内，再提取出 IP 数据报传入到分组处理模块。

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>A</td><td>B</td><td>C</td><td>D</td><td>E</td><td>F</td></tr>
<tr><td colspan="16">前24个字节包为数据包包头，包含了一些文件信息</td></tr>
<tr><td colspan="8">但对数据报的分析没有作用</td><td colspan="8">数据报报头（16字节）</td></tr>
<tr><td colspan="8"></td><td colspan="8">以太网帧头（14字节）</td></tr>
<tr><td colspan="6">以太网帧头</td><td colspan="10">IP头（一般20字节）</td></tr>
<tr><td colspan="10">IP头</td><td colspan="6">TCP头</td></tr>
<tr><td colspan="14">TCP头</td><td colspan="2">数据域</td></tr>
<tr><td colspan="16">数据域</td></tr>
<tr><td colspan="8"></td><td colspan="8">数据报报头（16字节）</td></tr>
<tr><td colspan="8"></td><td colspan="8">下一数据报的以太网帧头</td></tr>
<tr><td colspan="5">以太网帧头</td><td colspan="11">下一个数据报的IP头（一般20字节）</td></tr>
<tr><td colspan="9">IP头</td><td colspan="7">下一数据报的TCP头</td></tr>
<tr><td colspan="12">TCP头</td><td colspan="4">数据域</td></tr>
<tr><td colspan="10">数据域</td><td colspan="6">数据报报头</td></tr>
</table>

pcap 文件格式图

3. 分组处理模块:接受 IP 数据包,处理 IP 数据包

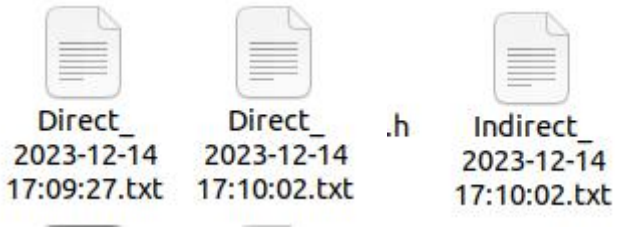
分组处理模块中接受功能是直接从输入模块传入 IP 数据报，处理模块则是完全依据分组转发算法实现的。



节点中的 IPv4 分组交付过程

4. 输出模块:信息写入文件

输出模块中输出的文件类型可以根据处理功能的流程图设计为 4 个输出文件分别为直接交付、间接交付、无路由丢弃、TTL 为 0 丢弃。并且文件内给数据报标上序号，展现接受数据报的时间、源地址和目的地址，并且还需要展示下一跳。从而让用户直观的看到数据的信息。



3.关键代码及解释

interface.c:

设置窗口大小，按钮位置，触发函数，以及主题设置等操作。

```
gtk_init(&argc, &argv);
gtk_settings_set_string_property(gtk_settings_get_default(), "gtk-theme-name", "Mist", "Mist");

window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
g_signal_connect(GTK_OBJECT(window), "destroy", G_CALLBACK(gtk_main_quit), NULL);
gtk_window_set_title(GTK_WINDOW(window), "IPv4 Forwarding Simulation");
gtk_window_set_position(GTK_WINDOW(window), GTK_WIN_POS_CENTER);

table = gtk_table_new(4, 8, TRUE);
gtk_container_add(GTK_CONTAINER(window), table);
```

GTK 按钮点击事件的回调函数

如果传入 1 则为“clear”，清空文件路径。

如果传入 2 则为“confirm”,读取 fileRoute 和 filePcap 中的内容。

```
// Clicking different buttons triggers different actions
void on_button_clicked(GtkWidget *button, gpointer data)
{
    if ((int)data == 1) {
        gtk_entry_set_text(GTK_ENTRY(pcapFile), "");
        gtk_entry_set_text(GTK_ENTRY(routeFile), "");
    } else if ((int)data == 2) {
        const gchar *pcapName = gtk_entry_get_text(GTK_ENTRY(pcapFile));
        const gchar *routeName = gtk_entry_get_text(GTK_ENTRY(routeFile));
        g_print("Reading data: %s\n", pcapName);
        g_print("file-name: %s\n", pcapName);
        g_print("route-file-name: %s\n", routeName);

        g_print("Copying strings\n");
        copyStr(fileRoute, routeName);
        copyStr(filePcap, pcapName);

        printf("fileRoute: %s, filePcap: %s\n", fileRoute, filePcap);
      You, 14分钟前 • Uncommitted changes
        read_Route(fileRoute);
        read_pcap(filePcap);
    }
}
```

readPacp.c:

定义了一些结构体和变量，struct pcap_pkthdr、FramHeader_t、IPHeader_t、IP_Link。然后通过 malloc 函数为这些结构体分配内存空间

读取 IP 数据报头结构，存储在 ip_header 中。将源 IP 和目的 IP 地址转换为字符串格式，并打印相关信息。

接下来，调用 stud_fwd_deal 函数处理此 IP 数据包，并输出一些信息，如数据包序号、版本长度、TTL、时间戳、源 IP 和目的 IP 地址以及 IP 协议类型等。

```c
//数据帧头
typedef struct FramHeader_t
{ //Pcap捕获的数据帧头
    u_int8 DstMAC[6]; //目的MAC地址
    u_int8 SrcMAC[6]; //源MAC地址
    u_short FrameType;    //帧类型
} FramHeader_t;
```

```c
//IP数据报头
typedef struct IPHeader_t
{ //IP数据报头
    u_int8 Ver_HLen;        //版本+报头长度
    u_int8 TOS;             //服务类型
    u_int16 TotalLen;       //总长度
    u_int16 ID;          //标识
    u_int16 Flag_Segment;    //标志+片偏移
    u_int8 TTL;             //生存周期
    u_int8 Protocol;        //协议类型
    u_int16 Checksum;       //头部校验和
    u_int32 SrcIP;      //源IP地址
    u_int32 DstIP;      //目的IP地址
} IPHeader_t;

typedef struct IP_Link
{
    IPHeader_t *IP_Data;
```

```c
//pcap_pkt_header 16 byte
memset(ptk_header, 0, sizeof(struct pcap_pkthdr));
if (fread(ptk_header, 16, 1, fp) != 1) //读pcap数据包头结构
{
    printf("\nread end of pcap file\n");
    break;
}

pkt_offset += 16 + ptk_header->caplen;   //下一个数据包的偏移值

//读取pcap包时间戳，转换成标准格式时间
struct tm *timeinfo;
time_t t = (time_t)(ptk_header->ts.tv_sec);
timeinfo = localtime(&t);

strftime(my_time, sizeof(my_time), "%Y-%m-%d %H:%M:%S", timeinfo); //获取时间
//printf("%s\n", my_time);

//数据帧头 14字节
fseek(fp, 14, SEEK_CUR); //忽略数据帧头

//IP数据报头 20字节
memset(ip_header, 0, sizeof(IPHeader_t));

if (fread(ip_header, sizeof(IPHeader_t), 1, fp) != 1)
{
    printf("%d: can not read ip_header\n\n", i);
    break;
}

printf("DstIP:%d\n",ip_header->DstIP);
inet_ntop(AF_INET, (void *)&(ip_header->SrcIP), src_ip, 16);

inet_ntop(AF_INET, (void *)&(ip_header->DstIP), dst_ip, 16);
```

```
    stud_fwd_deal(ip_header, i, my_time);

    ip_proto = ip_header->Protocol;
    verLen = (int)(((ip_header->Ver_HLen) & 0xf0) >> 4);
    ttl = (int)ip_header->TTL;

    printf("第%d个IP数据包, verLen:%d, TTL:%d, ",i, verLen, ttl);
    printf("time:%s, src_ip:%s, dst_ip:%s, ip protocol:%d\n",my_time, src_ip, dst_ip, ip_
unsigned int address;
    inet_pton(AF_INET,src_ip,&address);
```

const char * inet_ntop(int family, const void *addrptr, char *strptr, size_t len);

//将数值格式转化为点分十进制的 ip 地址格式

返回值：若成功则为指向结构的指针，若出错则为 NULL

从数值格式（addrptr）转换到表达式（strptr)。inet_ntop 函数的 strptr 参数不可以是一个空指针。调用者必须为目标存储单元分配内存并指定其大小，调用成功时，这个指针就是该函数的返回值。len 参数是目标存储单元的大小，以免该函数溢出其调用者的缓冲区。如果 len 太小，不足以容纳表达式结果，那么返回一个空指针，并置为 errno 为 ENOSPC。

readRoute.c:

将导出的路由表信息保存在 txt 文件中，根据每行不同的首字母去解析。C 表示直连地址，L 表示本机地址，S 表示静态路由。

```
    if(buf[0] == 'C' || buf[0] == 'L'){
        printf("%s\n",buf);
        char *p = strtok(buf," ");
        while(p = strtok(NULL, " ")){
            sscanf(p,"%[0-9.]/%d",dest,&masklen);
        }
        printf("dest: %s,masklen:%d\n",dest,masklen);
        if(dest != NULL){
            struct route_msg *msg;
            msg = (struct route_msg *)malloc(sizeof(struct route_msg));
            strcpy(msg->dest, dest);
            strcpy(msg->mask, getMask(masklen));
            msg->masklen = masklen;
            strcpy(msg->nexthop,"直接交付");
            route_add(msg);
            printf("msg->dest:%s,msg->mask:%s,msg->masklen:%d,msg->nexthop:%s",msg->dest,msg->mask,msg->masklen,m
        }


    } else if(buf[0] == 'S'){
        printf("%s\n",buf);
        char temp1[1024],temp2[1024],temp3[1024];
        sscanf(buf,"%s %[0-9.]/%d %s %s %[0-9.]",temp1,dest,&masklen,temp2,temp3,nexthop);
        printf("temp1:%s,dest:%s,masklen:%d,temp2:%s,nexthop:%s\n",temp1,dest,masklen,temp2,nexthop);
        if(nexthop != NULL){
```

forward.c:

处理传入的 IP 数据包

从 IP 数据报的首部字段中提取目的主机的 IP 地址 D，并与目的主机的地址掩码进行与操作，得到目的网络地址 N。

如果 N 与路由器的路由表中的网络地址相匹配，表示目的网络地址是当前路由

器直接相连的网络地址，则选择直接交付数据报到该网络地址上。
如果 N 不与当前路由器直接相连的网络地址匹配，则执行间接交付。
在路由器的路由表中查找是否存在目的地址为 D 的特定主机路由。如果有，则将数据报传送给路由表中指定的下一跳路由器。如果不存在特定主机路由，继续查找是否存在到达网络 N 的路由。如果有，则将数据报传送给路由表中指定的下一跳路由器。如果不存在到达网络 N 的路由，继续查找是否存在默认路由。如果有，默认路由指定了下一跳路由器。
如果连默认路由也不存在，则报告转发分组出错。

```
38  void route_Init();
39
40  void route_add(route_msg *proute);
41
42  int stud_fwd_deal(IPHeader_t *pBufferm,int number, char *my_time);
43
44  int fwd_DiscardPkt(IPHeader_t *pBuffer,int type,int number,char *my_time,char *next_Hop);
```

之前虽然在课程中在理论上学习了 IP 协议，基本 掌握了 IP 分组交付路由表的建立等概念，但是本次课设通过网络拓扑和代码能更加直观深入的了解路由与交换。同时，本次课设也正式激起了我对网 络这个领域的兴趣，想更多的探索网络中每一层都需要哪些协议都需要实现什么功能，各层协议之间是否有配合等等。

本次对于工具的安装与使用也是一个考验，期间帮助同学使用 Linux 系统的安装与配置，GNS3 中的 IP 地址以及路由表配置只能使用命令行配置， 这使我也学会了一些网络配置的一些基本命令。同时也学会了使用 wireshark 这个网络专业必不可少的工具，学会了对其进行抓包，分析，以及过滤的操作。