计算机网络实验报告

实验六

VPN 设计、实现与分析

学号：141220065

姓名：刘博

时间：2016.5.23

1. 实验目的：

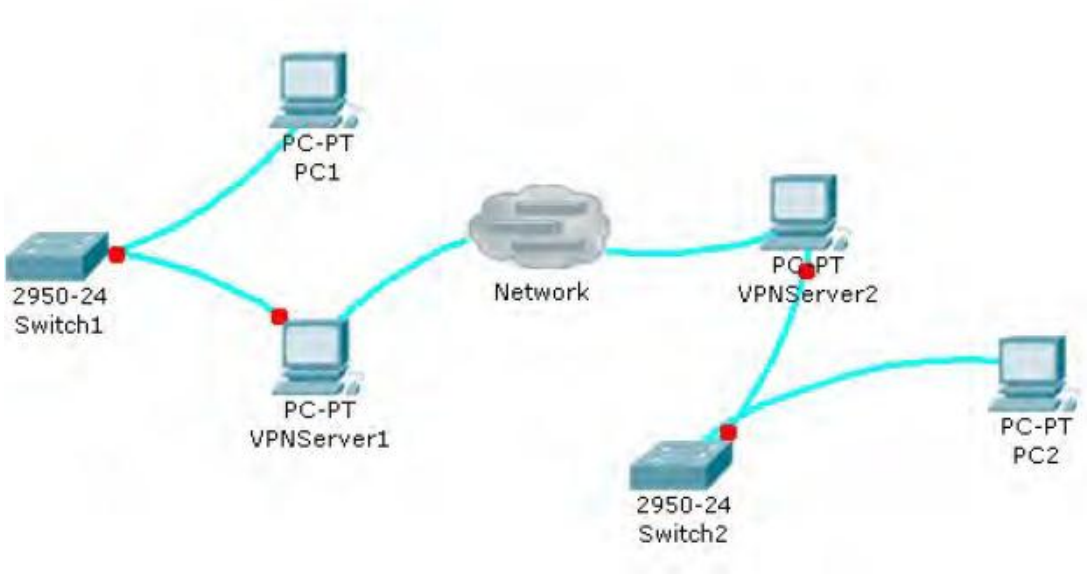
设计和实现一个简单的虚拟专用网络机制，并与已有的标准实现进行比较，进而进一步理解 VPN 的工作原理和内部实现细节。

2. 实验环境：

Ubuntu-12.04（自己电脑）

3. 网络拓扑配置：

节点名	虚拟设备名	IP	子网掩码
PC1	Pc1	10.0.0.2	255.255.255.0
VPN_server1	VPN1	10.0.0.1 192.168.0.2	255.255.255.0 255.255.255.0
Network	Network	192.168.0.1 172.0.0.1	255.255.255.0 255.255.255.0
VPN_server2	VPN2	172.0.0.2 10.0.1.1	255.255.255.0 255.255.255.0
PC2	Pc2	10.0.1.2	255.255.255.0



4. 路由配置文件：

为所有的节点设置 ip 地址和路由规则：

PC1：ifconfig eth0 10.0.0.2 netmask 255.255.255.0

Route add default gw 10.0.0.1

VPN_server1:ifconfig eth0 10.0.0.1 netmask 255.255.255.0

Ifconfig eth1 192.168.0.2 netmask 255.255.255.0

Route add default gw 192.168.0.1

Network:ifconfig eth0 192.168.0.2 netmask 255.255.255.0
Ifconfig eth1 172.0.0.1 netmask 255.255.255.0
Echo 1 > /proc/sys/net/ipv4/ip_forward

VPN_server2:ifconfig eth0 172.0.0.2 netmask 255.255.255.0
Ifconfig eth1 10.0.1.1 netmask 255.255.255.0
Route add default gw 172.0.0.1

PC2:ifconfig eth0 10.0.1.2 netmask 255.255.255.0
Route add default gw 10.0.1.1

5. 数据结构说明：

以太网帧头部：

```
typedef struct ethernet {
        unsigned char dst[6];
        unsigned char src[6];
        unsigned char type[2];
} eth_head;
```

Dst：目的 mac 地址；
Src：源 mac 地址；
Type：上层协议类型；

Ip 头部：

```
typedef struct IP {
        unsigned char IHL : 4;                  //Internet Head Length
        unsigned char version : 4;              //IP Version
        unsigned char ECN : 2;
        unsigned char DS : 6;                   //Determine Service
        unsigned short length;                  //Length of datagram
        unsigned short label;
        unsigned short offset : 13;
        unsigned char tag : 3;
        unsigned char live;
        unsigned char protocol;
        unsigned short check_sum;
        unsigned char ip_src[4];                //Source Address
        unsigned char ip_dst[4];                //Destination Address
} ip_head;
```

IHL：网络头部长度；
Version：ip 版本号；
Live：ttl 数据报生存期；
Protocol：上层协议类型；
Check_sum：校验和；

Ip_src：源 ip 地址；
Ip_dst：目的 ip 地址；

ARP 头部：

```
typedef struct ARP {
        unsigned char mac_target[6];
        unsigned char mac_source[6];
        unsigned short ethertype;
        unsigned short hw_type;
        unsigned short proto_type;
        unsigned char mac_addr_len;
        unsigned char ip_addr_len;
        unsigned short op;
        unsigned char mac_sender[6];
        unsigned char ip_sender[4];
        unsigned char mac_receiver[6];
        unsigned char ip_receiver[4];
        unsigned char padding[18];
} arp_head;
```

Mac_target：目标 mac 地址；
Mac_source：源 mac 地址；
Hw_type：硬件类型（网卡编号）；
Proto_type：协议类型；
Mac_addr_len：mac 地址长度；
Ip_addr_len：ip 地址长度；
Op：操作类型；
Mac_sender：mac 发送者地址；
Ip_sender：ip 发送者地址；
Mac_receiver：mac 接受者地址；
Ip_receiver：ip 接受者地址；
Padding：附加段；

路由表项：

```
typedef struct route_item {
        unsigned char destination[16];
        unsigned char gateway[16];
        unsigned char netmask[16];
        unsigned char interface[16];
} route_item;
```

Destination：目的 ip 地址；
Gateway：网关地址；
Netmask：网络掩码；
Interface：网络接口（网卡名）；

Arp 表项：

```
typedef struct arp_item {
        unsigned char ip_addr[16];
        unsigned char mac_addr[18];
} arp_item;
```

Ip_addr：ip 地址；
Mac_addr：mac 地址；

Device 表项：

```
typedef struct device_item {
        unsigned char interface[16];
        unsigned char ip_addr[16];
        unsigned char mac_addr[18];
        int is_entrance;
} device_item;
```

Interface：网卡接口（网卡名）；
Ip_addr：网卡 ip 地址；
Mac_addr：网卡 mac 地址；
Is_entrance:网卡是否是外部接口；

VPN 路由表项：

```
typedef struct VPN_route {
        unsigned char dst[16];
        unsigned char VPN_dst[16];
        unsigned char netmask[16];
} VPN_item;
```

6. 配置文件说明：

所有配置文件都在 configuration 文件夹下，其中 VPN_server 分别有独立的配置文件；Route，device 表的内部与实验四基本相同，新加入了 VPN 表，用来保存 VPN 的虚拟路由信息。

7. 程序设计思路及运行流程：

设计思路大体与实验四相似，只是在实验四的静态路由转发基础上加上一层 VPN 的封装即可；
首先路由器进行初始化，直接将路由表和设备表进行读取和赋值，同时加上读取 VPN 表并保存：

```
void read_settings(void) {
        FILE *fp = fopen("../configuration.file/route_table.txt","r");
        while(!feof(fp))
        {
                route_item *p = (route_item *)route_info + route_item_index;
                fscanf(fp,"%s %s %s %s",p->destination,p->gateway,p->netmask,p->interface);
                route_item_index++;
        }
        route_item_index--;
        //route item index for all route items
        fclose(fp);
        fp = fopen("../configuration.file/device_table.txt","r");
        while(!feof(fp))
        {
                device_item *p = (device_item *)device + device_item_index;
                fscanf(fp,"%s %s %s %d",p->interface,p->ip_addr,p->mac_addr,&p->is_entrance);
                strcpy(arp_table[arp_item_index].ip_addr,p->ip_addr);
                strcpy(arp_table[arp_item_index].mac_addr,p->mac_addr);
                device_item_index++;
                arp_item_index++;
        }
        device_item_index--;
        //device item index for all device items
        fclose(fp);
        fp = fopen("../configuration.file/VPN_route.txt","r");
        while(!feof(fp))
        {
                VPN_item *p = (VPN_item *)VPN_route_info + VPN_item_index;
                fscanf(fp,"%s %s %s",p->dst,p->VPN_dst,p->netmask);
                VPN_item_index++;
        }
        VPN_item_index--;
        fclose(fp);
}
```

初始化完成后，判断收到的包裹是否是 VPN 包裹，如果是则进行解包，如果不是，则进行封装后选择转发或不转发；

```
int detect_packet(ip_head *ip) {
        unsigned char local_net[16];
        unsigned char local_gw[16];
        unsigned char over_net[16];
        unsigned char over_gw[16];
        int i;
        for(i = 0; i < device_item_index; i++)
        {
                if(device[i].is_entrance)
                        strcpy(local_net,device[i].ip_addr);
                else
                        strcpy(local_gw,device[i].ip_addr);
        }
        for(i = 0; i < VPN_item_index; i++)
        {
                strcpy(over_net,VPN_route_info[i].dst);
                strcpy(over_gw,VPN_route_info[i].VPN_dst);
                unsigned int dst_addr;
                unsigned int src_addr;
                unsigned int mask;
                inet_aton(local_net,(struct in_addr *)&src_addr);
                inet_aton(over_net,(struct in_addr *)&dst_addr);
                inet_aton(VPN_route_info[i].netmask,(struct in_addr *)&mask);
                if((((*(unsigned int *)ip->ip_src) & mask) == (src_addr & mask)) && (((*(unsigned int *)ip->ip_dst) & mask) == (dst_addr &
mask)))
                        return PACK;
                inet_aton(local_gw,(struct in_addr *)&dst_addr);
                inet_aton(over_gw,(struct in_addr *)&src_addr);
                if((((*(unsigned int *)ip->ip_src) & mask) == (src_addr & mask)) && (((*(unsigned int *)ip->ip_dst) & mask) == (dst_addr &
mask)))
                        return UNPACK;
        }
        return -1;
}
```

判断方式就是看包裹的目的 IP 和源 IP，如果目的 IP 和源 IP 都是在两个 VPN 子网之内的（例如 10.0.0.2 和 10.0.1.2）那么就判断是封装前的 VPN 包，将其封装成 VPN 包即可；

如果目的 IP 和源 IP 都是两个 VPN 的网关地址（例如 192.168.0.2 和 172.0.0.2）那么就判断是封装后的 VPN 包，则将其解包即可；

如果目的 IP 和源 IP 不满足以上两个条件，则判断该包与 VPN 无关，则正常转发即可。

下面看看如何封装与解封：

1. 封装：

```
if(operation == PACK)
{
        printf("PACK\n");
        int VPN_temp_index = 0;
        VPN_temp_index = find_VPN_ip(ip);
        if(VPN_temp_index == -1)
        {
                printf("not in VPN route table\n");
        }
        else
        {
                //pack a new VPN packet
                eth_head *VPN_eth = (void *)VPN_packet;
                //copy_eth_ip_icmp
                memcpy((void *)VPN_eth,(void *)eth,sizeof(eth_head) + sizeof(ip_head) + 16);
                //change ip_dst
                ip_head *VPN_ip = (void *)((unsigned char *)VPN_eth + sizeof(eth_head));
                copy_ip(VPN_route_info[VPN_temp_index].VPN_dst,VPN_ip->ip_dst);
                int temp;
                for(temp = 0; temp < device_item_index; temp++)
                {
                        if(!device[temp].is_entrance)
                                copy_ip(device[temp].ip_addr,VPN_ip->ip_src);
                }
                //change ip_length
                unsigned short len = ntohs(ip->length) + sizeof(ip_head) + 16;
                VPN_ip->length = htons(len);
                //copy data
                char *cat = (char *)VPN_ip + sizeof(ip_head) + 16;
                memcpy(cat,buffer + sizeof(eth_head),2048 - (sizeof(eth_head) + sizeof(ip_head) + 16));
                eth = VPN_eth;
                ip = VPN_ip;
                n_read = n_read + sizeof(ip_head) + 16;
        }
}
```

首先明确我们需要将一个原来的数据报（ping 包）作为新的 VPN 数据包的负载，即将原来的 ping 的 IP 包放入新的 VPN 包的 DATA 部分，然后添加新的 mac 地址和 ip 地址，所以我们可以对原来的数据包进行改造，适当的扩展即可完成：

首先 mac 部分由于与 VPN 的关系不大，所以 mac 地址部分不要进行额外的修改，所以可以将原来的数据包的 mac 地址完整拷贝到新的数据包的 mac 地址；

对于 ip 地址的改造是这里改造的重点，首先在网络上跑的数据包其 ip 地址均应该是网络上可寻址的 ip 地址，内网地址需要更改，所以将 ip 部分的 src 改成本地 VPN 的网关地址，将 dst 改成目标 ip 的 VPN 的网关地址；

由于对 ip 包进行了修改（数据部分增加）所以需要更改 ip 头部的 length 部分；

其次需要修改所有 ip 内部的校验和信息，生存时间（包括 icmp 的校验和）

然后将生成的完整的 VPN 包按照标准的路由数据包转发即可（利用实验四已有的部分）注意数据发送的大小已经改变；

2. 解封：

```
else if(operation == UNPACK)
{
        printf("UNPACK\n");
        //unpack a VPN packet
        eth_head *VPN_eth = (void *)VPN_packet;
        //copy_eth_ip_icmp
        memcpy((void *)VPN_eth,(void *)eth,sizeof(eth_head) + sizeof(ip_head) + 16);
        //change ip_dst
        ip_head *VPN_ip = (void *)((unsigned char *)VPN_eth + sizeof(eth_head));
        ip_head *pre_ip = (void *)((unsigned char *)eth + sizeof(eth_head) + sizeof(ip_head) + 16);
        memcpy(VPN_ip->ip_dst,pre_ip->ip_dst,4);
        memcpy(VPN_ip->ip_src,pre_ip->ip_src,4);
        //change ip_length
        unsigned short len = ntohs(pre_ip->length);
        VPN_ip->length = htons(len);
        //copy data
        char *cat = (char *)VPN_ip + sizeof(ip_head) + 16;
        memcpy(cat,buffer + sizeof(eth_head) + (sizeof(ip_head) + 16) * 2,2048 - (sizeof(eth_head) + (sizeof(ip_head) + 16)
2));
        eth = VPN_eth;
        ip = VPN_ip;
        n_read = len + sizeof(eth_head);
}
```

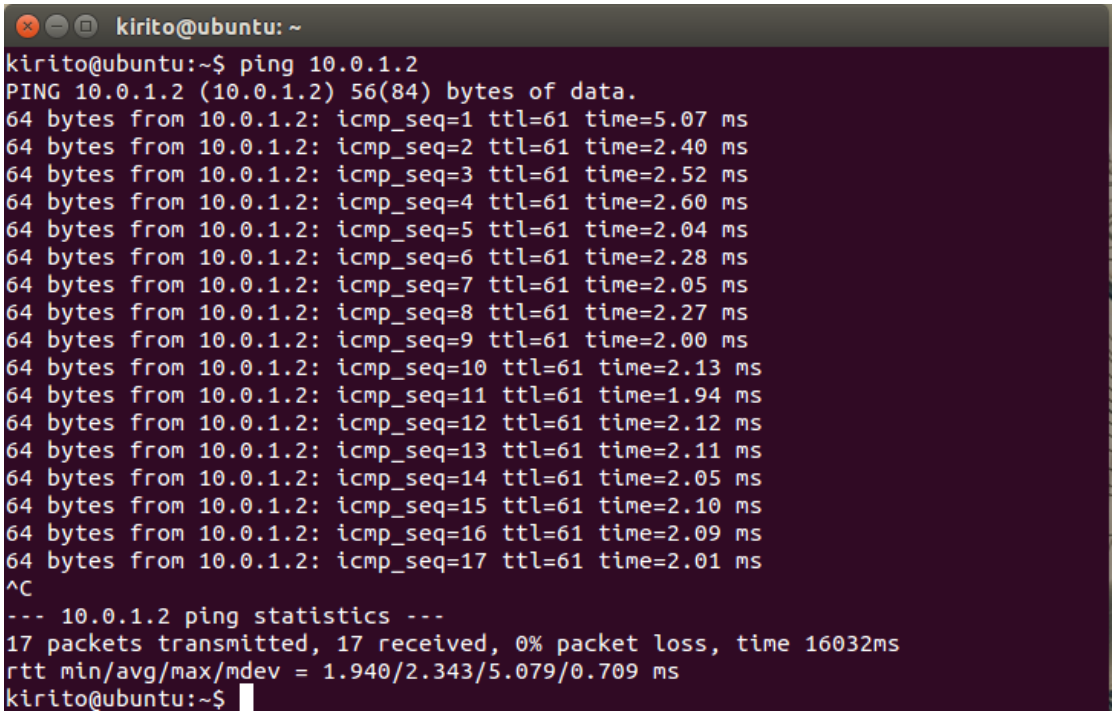
如果我们收到了一个来自外部的 VPN 数据包，那么我们就需要将它进行解封，我们需要的数据包实际上是该 VPN 包的负载；所以我们只需要将这个 VPN 外部的东西舍弃，然后将其负载重新提取；

首先需要将负载部分的 ip，icmp 部分提取出来并且拷贝到真正的 ip，icmp 处，然后修改 ip 长度，校验和，生存时间等信息，将新的 ip 包作为真正的 ip 包进行转发即可；

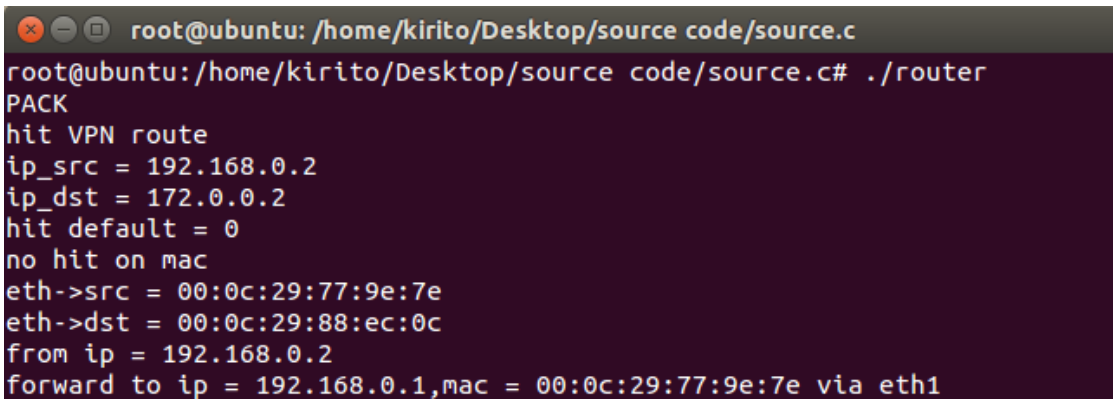
当 VPN 的封装和解封结束后，剩下的就是正常的路由转发功能，则可以直接利用实验四的代码即可；这里不再进行赘述。

8．运行结果截图：

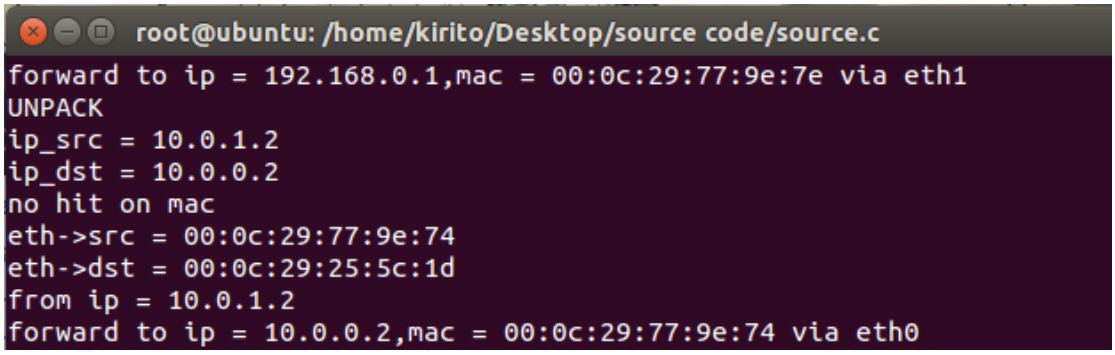
首先将 VPN1 和 VPN2 的 router 程序运行；
然后从 pc1 ping pc2：
运行结果如下：



可以看到 pc1（10.0.0.2）可以成功 ping 到 pc2（10.0.1.2）
并且从两个 pc 处看起来所有的数据包就像是直接互相发送的而没有进行修改，实际上是通过了 VPN 之后才可以建立的连接；
下面看一下 VPN 上的程序输出：



首先程序 PACK 一个 VPN 包，PACK 后的 VPN 包 ip 地址是 192.168.0.2->172.0.0.2 然后通过网关 192.168.0.1 成功发送到了 network 上；



过了一会程序收到了来自外部的（ping reply）VPN 包，所以先 UNPACK 该包，UNPACK 后该 ip 包的 ip 地址是 10.0.0.2->10.0.1.2 然后成功发送到 10.0.0.2（pc1）上。

9. 相关参考资料：

（百度，google 等）；

10. 对比样例程序：

本次实验无任何样例程序；

11. 代码个人创新及思考：

这次程序主要是注意修改 ip 地址时需要进行注意，如果仅仅只是修改 ip 地址而不去修改其他的 icmp 校验和，ip 数据长等信息，则生成的数据包不会通过转发，会变成以太网错误帧，所以要注意进行相应的 ip 头部，icmp 头部改变；

其次对于数据包判断，如何判断一个数据包是否是 VPN 数据包，对于收到的和发送的数据包如何进行封装和解封都是关键；

其次，如果仅仅只是修改新的 VPN 包的目的 ip 地址，而不修改源 ip 的地址，那么在 network 上不会进行转发，原因是路由表内没有源 ip 地址的路由信息，所以必须将新 VPN 包的 ip 源地址也改成网关地址（借用），这样才能直接在 network 上跑。