实验六 VPN 设计、实现、分析

时间：2014.12.27
徐栋　121220312　dc.swind@gmail.com

【实验目标】

本实验主要目的是设计和实现一个简单的虚拟专用网络的机制，并与已有的标准实现（如PPTP）进行比较，进而让学生进一步理解VPN的工作原理和内部实现。

【网络拓扑配置】

同实验教材一样。

节点名	虚拟设备名	IP	Netmask
PC1	eth0	10.0.0.2	255.255.255.0
VPNServer1	eth0	10.0.0.1	255.255.255.0
	eth1	192.168.0.2	255.255.255.0
NETWORK	eth0	192.168.0.1	255.255.255.0
	eth1	172.0.0.1	255.255.255.0
VPNServer2	eth0	172.0.0.2	255.255.255.0
	eth1	10.0.1.1	255.255.255.0
PC2	eth0	10.0.1.2	255.255.255.0

表 1 - 路由配置信息

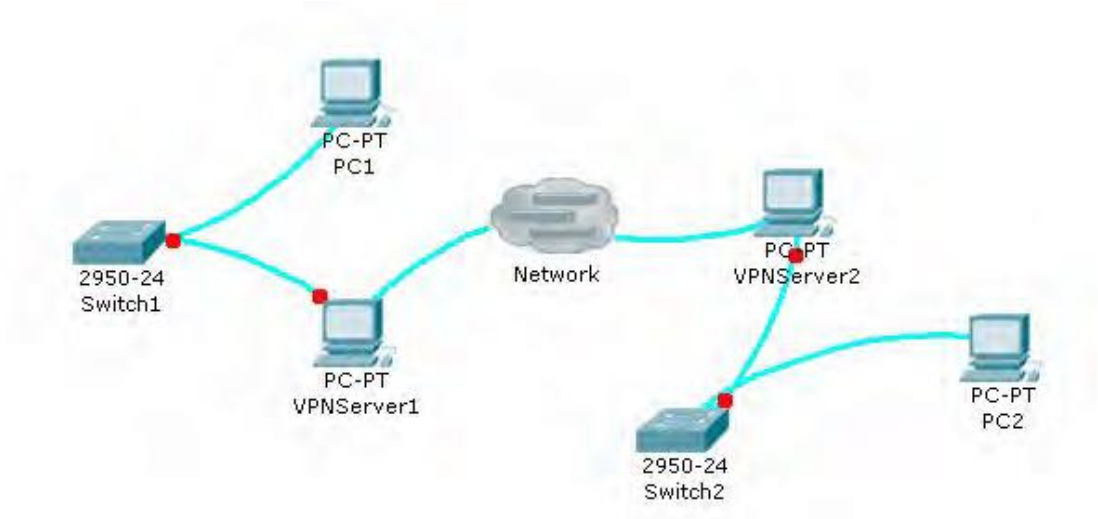


图 0.1 - 拓扑图

【数据结构、配置文件说明】

静态路由规则，确定转发规则

```
struct route_item{
    char destination[16];          //des_ip 目的 ip
    char gateway[16];              //gateway 转发端口 ip
    char netmask[16];              //netmask 掩码
    int interface;                 //转发端口
}route_info[ROUTE_INFO_MAX];
```

Arp 表，用于确定转发至下一跳的物理地址

```
struct arp_table_item{
    char ip_addr[16];                //gateway IP
    unsigned char mac_addr[18];      //NEXT HOP MAC
}arp_table[ARP_SIZE_MAX];
```

网络设备信息，包括物理地址，ip 地址，下一跳 ip 地址，及接口号

```
struct device_info{
    unsigned char mac[18];         //LOCAL MAC
    char ip[16];                   //LOCAL IP
    char nextip[16];               //NEXT HOP IP
    int interface;                 //LOCAL INTERFACE
}device[DEVICE_MAX];
```

自定义 ARP 数据报格式

```
struct arp_header{
    unsigned char dst_mac[6];      //Ethernet des mac address
    unsigned char src_mac[6];      //Ethernet src mac address
    u_int16_t mac_type;            //Ethernet type

    u_int16_t hd_type;             //arp hardware type
    u_int16_t pro_type;            //arp protocol type
    u_int8_t hd_arrl;              //hardware address length
    u_int8_t pro_arrl;             //protocal address length
    u_int16_t arp_type;            //arp type
    unsigned char arp_src_mac[6]; //arp src mac address
    in_addr_t src_ip;              //arp src ip address
    unsigned char arp_dst_mac[6]; //arp dst mac address
    in_addr_t dst_ip;              //arp dst ip address

    unsigned char unuse[18];       //填充
};
```

配置文件位于代码 init()函数中。

（没有单独的读取文件，而是内置在代码中，无本质区别）

route_item_index 为静态路由规则数目。

以下每三行为设置一条静态路由规则的格式

```
strcpy(route_info[0].destination , "10.0.1.2");
strcpy(route_info[0].gateway , "10.0.1.1");
route_info[0].interface = 1;
```

设置对应网络设备 eth0(device[0])、eth1(device[1])的 ip 及下一跳 ip

```
strcpy(device[0].ip, "172.0.0.2");
strcpy(device[0].nextip, "172.0.0.1");
strcpy(device[1].ip, "10.0.1.1");
strcpy(device[1].nextip, "10.0.1.2");
```

设置完成

【程序设计思路】

代码在 Lab4 代码的基础上进行扩展，首先执行 Part 1，进行一些必要的初始化操作，然后执行 Part 3 的 start_receive 函数开始程序的主体部分。其中一部分运行流程在结果分析中介绍。

Part 1 程序开始

Main()函数首先创建一个链路层 socket、一个 IP 网络层 socket，收发所有包。

然后依次调用：

get_eth_info() 获取设备 eth0、eth1 的端口号及 MAC 地址
init() 读取配置文件
init_send() 通过向两端发送 2 个 arp 包进行 arp 初始化
start_receive() 转发开始

Part 2 init_send() 发送用于初始化的 arp 包

使用创建的链路层 socket 进行发送。

根据自定义的 ARP 数据报格式进行各个字段的赋值，赋值完成后，创建 struct sockaddr_ll connection; 对链接初始化后，使用 sendto 在 socket 上发包。对每个端口发送一个 arp 包。

Part 3 start_receive() 转发开始

```
While(1)
    Receive                     //接收包
    Get packet info             //分析包并且判断接收地址是否是自己
    If IP                       //如果是 IP 则进行如下处理
        Check which eth         //确定需要 repack 还是 unpack
        If repack               //向外发送，repack_packet
            repack_packet()
        If unpack               //向内发送，unpack_packet
            unpack_packet()
    IF ARP                      //根据信息进行 arp 初始化
        Init_arp()
```

是 IP 协议的话，判断是哪种情况，如果是从内部向外访问，则调用 repack_packet 函数进行 VPN 封装（执行 Part 5），如果是从外部向内访问，则调用 unpack_packet 函数进行 VPN 解封（执行 Part 6）。

如果是 ARP 协议，执行 Part 4.

Part 4 init_arp()

判断是哪一端口进入的，然后根据目的 ip 判断是否是有用信息，若是有用（即 init_send()的返回包）则进行 arp table 的初始化。

Part 5 repack_packet()

这里我尝试过两种方式，将收到的帧中的 IP 包外面再包装一层 IP 报头，这里需要说明的是，可以在两层 IP 报头之间增加一层自定义 VPN 报头，可以使得程序的兼容性更好，但是临近期末，时间有限，所以未添加 VPN 报头，当然实验要求是完成了的。

继续说两种方式，一种是在 IP socket 层发送，一种是在 eth 链路层 socket 发送，我先尝试了后者，但是一直没有成功，在 network 路由上虽然可以收到包装好的数据包，但是会丢弃不转发。使用 ping 等方式始终未找到原因。因而最终实现是使用了 IP socket。

此函数中是处理链路层的代码，因为最后使用了 IP 层 socket，所以调用了一个名为 init_send_send()的函数进行的处理。

处理方式为，在收到的 IP 包外面添加一个 IP 包，包的源地址为当前 VPNserver，目的地址为内层 IP 包目的地址所在 VPNServer 的 IP 地址。然后发送。

Part 6 unpack_packet()

Unpack 的过程与 repack 相反，丢掉外层 IP 头部，然后根据内层 IP 的地址，查询静态路由表 route_info 和 ARP 表，确定下一跳的 MAC 地址，然后通过链路层的 socket 进行转发，将处理好的链路层数据包调用 resend()函数转发。

Part 7 resend() 转发

根据路由规则，确定收到包的链路层的发送地址和目的地址，然后转发。发送包与 Part 2 中类似，初始化好 connection 后进行 sendto。

【运行结果】

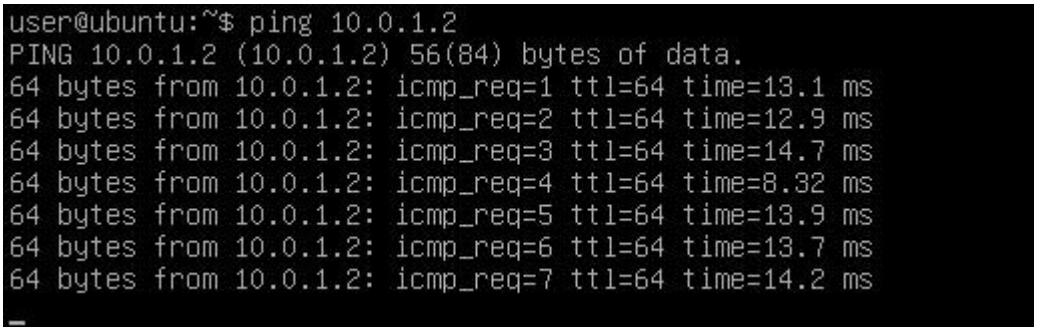
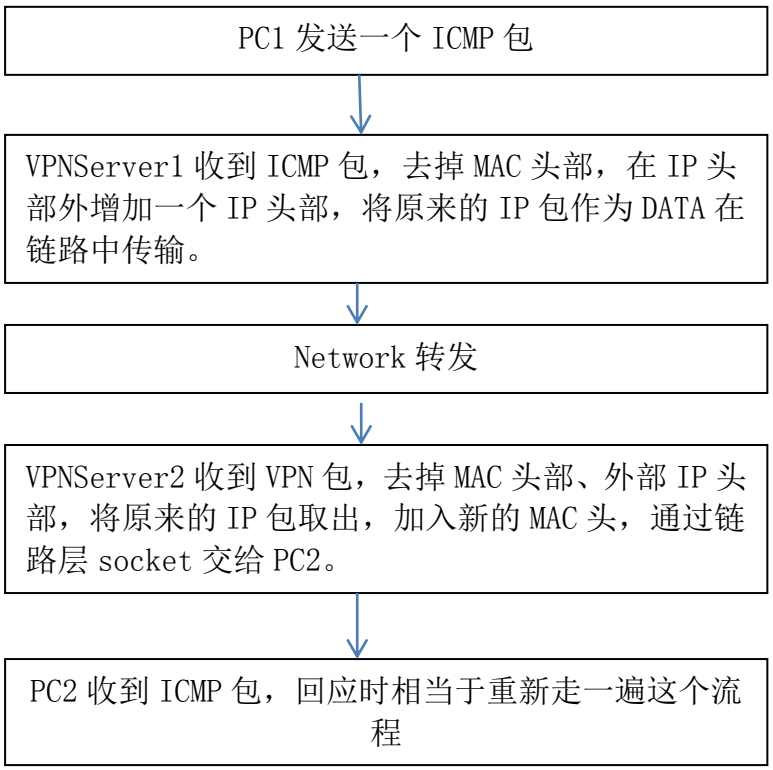


图 1.1 - PC1 PING PC2

分析：

因为没有在代码外配置转发规则，所以在不运行 VPNServer 上的代码前是 ping 不通的，在运行之后的执行流程如下：



【参考资料】

实验教材及教材提供的参考资料。

【对比样例程序】

未找到样例程序，独立实现。

【个人创新及思考、应用场景】

1. 首先考虑要完全脱离 MAC 层的信息，所以在配置信息上进行了修改，只需要获取 ip 层的配置信息。端口信息等通过 ioctl 自动获取。

2. ARP 信息通过初始发送 ARP 数据报进行 ARP 初始化。实现了模拟发送 ARP 数据报的功能。

3. 在处理无 ARP 信息第一次丢包上，考虑到静态的网络环境下，其转发的下一跳是固定的，所以进行初始化的时候，提前主动发送 ARP 数据报，获取 arp 表的相应信息。

4. 在两个 VPN 之间增加一个 NETWORK 路由使得适应范围更广，更切合实际。

5. 原 IP 包外直接加一个 IP 报头虽然可以实现，但是兼容性、扩展性不够好，因此在两个 IP 报头之间增加一个 VPN 头部，可以使得实现更切合实际。

应用方面可以配置一个 VPN，等等。