**《信息安全概论》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **2018级计算机科学与技术卓越01班** | | | **姓名** | **秦凡** |
| **实验题目** | **隧道协议仿真程序设计与实现** | | | | | |
| **实验时间** | **2021/5/13** | | **实验地点** | **DS3 401** | | |
| **实验成绩** |  | | **实验性质** | **□验证性 □设计性 □√综合性** | | |
| 教师评价：  □算法/实验过程正确； □源程序/实验内容提交 □程序结构/实验步骤合理；  □实验结果正确； □语法、语义正确； □报告规范；  其他：  评价教师签名： | | | | | | |
| 一、实验目的  目的：   1. 编写IP隧道协议仿真程序 2. IP in IP报文传输仿真   实验开展方式：   1. 以组为单位进行实验，3人1组 2. 课内外结合的方式完成实验，课内完成实验内容、实验要求等讲解，以及实验结果、实验报告检查；课内外结合完成程序设计、编制与调试   实验软硬件环境要求：  要求在四台机器上实现 | | | | | | |
| 二、实验项目内容  实验内容：   1. 编写IP隧道协议仿真程序 2. IP in IP报文传输仿真   实验讲授内容：   1. IP隧道协议原理讲解 2. IP in IP隧道协议仿真实现技术讲解   学生实践内容：   1. 编制IP隧道仿真程序，包括四个程序，IP请求、隧道封装、隧道解封、IP接收等四个仿真程序 2. 进行IP隧道仿真程序部署 3. 进行IP in IP隧道协议演示   编写实验报告 | | | | | | |
| 三、实验过程或算法（源程序）1.IP隧道协议1.1 协议概述 IP隧道技术：是路由器把一种网络层协议封装到另一个协议中以跨过网络传送到另一个路由器的处理过程。  隧道技术是一种数据包封装技术，它是将原始IP包（其报头包含原始发送者和最终目的地）封装在另一个数据包（称为封装的IP包）的数据净荷中进行传输。  所谓隧道，实际上是路由器把一种网络层协议封装到另一个协议中以跨过网络传送到另一个路由器的处理过程。发送路由器将被传送的协议包进行封装，经过网络传送，接受路由器解开收到的包，取出原始协议；而在传输过程中的中间路由器并不在意封装的协议是什么。这里的封装协议，称之为传输协议，是跨过网络传输被封装协议的一种协议，IP协议是IOS唯一选择的传输协议。而被封装的协议在此为IPX协议或者AppleTAlk协议，通常可以称之为乘客协议。需要特别注意的是：隧道技术是一种点对点的链接，因而必须在链接的两端配置隧道协议。  我们假设在站点A和B之间交换数据。在IP协议下，数据包在路由器之间的传递直到到达目的地的过程，其线路是没有经过预先的设计和计划。然而在MPLS（多协议标记交换）协议下，在站点A和B之间传递的IP数据包必须沿由第一个路由器预先建立起来的通路传送。这条通路在IP网络中就像一条中空的隧道，直接连接A和B两个站点。  隧道技术是一种数据包封装技术，它是将原始IP包（其报头包含原始发送者和最终目的地）封装在另一个数据包（称为封装的IP包）的数据净荷中进行传输。在移动IP中，隧道包目的地址就是转交地址，当外地代理（或移动节点）收到这个隧道包后，解封装该包，把里面的净荷提交给移动节点。如图所示。  移动IPv4主要有三种隧道技术，它们分别是：IP in IP、最小封装以及通用路由封装。 1.2 IP in IP协议1.2.1 简介 IP-IN-IP协议描述了一种可在IP数据报中封装另一个IP数据包（作为净负载）的方法。封装通过把路由信息送往某个中间目的地（不是由原IP头部的IP Destination Address域）把正常的IP路由变为数据报。一旦封装后的数据报到达该中间目的地节点，就被拆分，得到原IP数据报,然后原数据报被送到目的地址（由原Destination Address域决定）. 封装与拆分数据报的过程通常称为数据报“隧道”（"tunneling"），封装方和拆分方分别为隧道的端点（“endpoints"）;封装方称为隧道的“入口点”（"entry point"）,拆分方称为隧道的出口点（"exit point"）.  在最常见的隧道中我们有：  其中source, encapsulator, decapsulator和destination是独立的节点。encapsulator 节点称为隧道的“入口点”而decapsulator节点称为隧道的“出口点”.在封装与拆分的过程中同一个隧道可能有多个source-destination对。 1.2.2 在IP中封装IP 为了使IP-in-IP来封装IP数据报,在现存IP头部前面插入外层的IP头部，如下所示：  +---------------------------+  | |  | Outer IP Header |  | |  +---------------------------+ +---------------------------+  | | | |  | IP Header | | IP Header |  | | | |  +---------------------------+ ====> +---------------------------+  | | | |  | | | |  | IP Payload | | IP Payload |  | | | |  | | | |  +---------------------------+ +---------------------------+  外层IP头部中的Source Address和Destination Address标识了隧道的“端口”.内层IP头部的中的Source Address和Destination Addresses标识了数据报的原（最初，original）发送方和接收方.内层IP头部不能被封装方修改（并在向隧道出口传输的过程中保持不变），除非按下面的方法递减TTL.封装后的数据报在隧道传输的过程中IP选项不做任何修改。如果要修改，则在内外层IP头部间插入其他协议头部，如IP认证头部（Authentication header，参考文献[1]）.注意内层IP头部的安全选项可能影响正在封装的（外层）IP头部的安全选项。  **（1）IP头部各域及管理**  外层IP头部由封装方按下面设置:   |  |  | | --- | --- | | **字段** | **含义** | | Version | 版本号为4 | | IHL | 指外层IP头部长度，以32比特为计算单位。 | | TOS | 从内层IP头部复制。 | | Total Length | 指整个IP负载的长度，包括外层IP头，内层IP头和IP负载。 | | Identification, Flags, Fragment Offset | 这三个字段的含义与RFC791的定义相同。注意，如果内层IP头部的DF位置位，外层IP头部的DF位也必须置位。如果内层IP头的DF未置位，外层IP头部的DF位可以置位也可以不置位。 | | Time to Live | 外层IP头部的TTL域设置为发送该数据包到隧道目的端的合适的值。 | | Protocol | 协议编号为4 | | Header Checksum | 外层IP头部的校验字段。 | | Source Address | 执行该IPinIP隧道头封装的隧道入口设备的IP地址 | | Destination Address | 执行该IPinIP隧道头解封装的隧道出口设备的IP地址。 | | Options | 内层IP头部的任何选项字段通常不被复制到外层IP头部。隧道路径上的设备可以添加新的选项字字段。内层IP头部的安全选项字段的类型可能影响外层IP头部的安全选项字段的选择。 |   在封装数据报时,如果隧道作为转发数据报的一部分，内层IP头部的TTL将减1；否则，  在封装的过程中内层TTL保持不变.如果得到的内层IP头部的TTL为0,数据报被丢弃并应该向发送者产生一个Time Exceeded的ICMP信息。不允许封装方对TTL=0的数据报进行封装。内层IP头部中的TTL在拆分的过程中保持不变。拆分后，如果内层数据报TTL=0，拆分方必须丢弃该数据报。拆分后，如果拆分方转发该数据报到它的一个网络接口，它像正常转发IP数据报那样递减TTL。封装方可以使用现存适合的IP机制来把封装后的净载数据传送到隧道的出口点。  特别地，允许使用IP选项,还可以允许分片，除非内层IP头部中设置了"Don't Fragment"位。使用该分片限制是为了使使用路径MTU发现（参考文献[7]）的节点能够得到他们所要寻找的信息。  **（2）路由失败**  在隧道内部的路由环回（Routing loops）特别危险，它们使数据报再次回到封装方。假  设一个数据报到达路由器等待转发,而该路由器认为该数据报在传送之前必须封装 ，那么：   * 如果该数据报的Source Address与路由器自己的任一个网络接口的IP地址匹配，该   路由器不允许为该数据报建立隧道；相反，该数据报应该被丢弃.   * 如果该数据报的Source Address 与隧道的目的IP地址匹配（隧道出口点一般由路由   器根据数据报的IP头部的Destination Address选择），路由器不允许为该数据报建立隧道 ，相反，该数据报应该被丢弃。 2.IP隧道仿真程序框架  3.程序模块实现3.1 IP请求模块 此模块用于处理从传输层接收到到UDP数据包，将其封装为IP数据包，然后对隧道封装模块发出隧道封装请求。  形成IP数据包的代码如下：  // Make Ip packet    unsigned int MakeIpPacket(unsigned int DF, unsigned int MF, unsigned int FragmentOffset, const IP\_Packet ip\_packet, unsigned char \*buf, unsigned char \*IPv4\_Option, long IPv4\_Option\_Len, unsigned char \*IPv4\_Data, short IPv4\_Data\_Len)    {    //第一个byte    unsigned char VersionAndIhl = extendl\_8bit(ip\_packet.IPv4\_Version, 4) + extendl\_8bit(ip\_packet.IPv4\_IHL, 0); //8 bit    memcpy(buf, &VersionAndIhl, sizeof(VersionAndIhl));    //第二个byte    unsigned char IPv4\_TOS = extendl\_8bit(ip\_packet.IPv4\_TOS, 0); //8 bit    memcpy(&buf[1], &IPv4\_TOS, sizeof(IPv4\_TOS)); //1已经被占用    //第三四个byte    unsigned short IPv4\_TotalLength = 20 + 40 + (short)IPv4\_Data\_Len; //ip\_packet.IPv4\_TotalLength;    memcpy(&buf[2], &IPv4\_TotalLength, sizeof(IPv4\_TotalLength)); //3已经被占用    //第五六个byte    unsigned short IPv4\_Identification = ip\_packet.IPv4\_Identification;    memcpy(&buf[4], &IPv4\_Identification, sizeof(IPv4\_Identification)); //5已经被占用    //第7、8个byte    //unsigned short extendl\_16bit(unsigned int bit\_content, int bit\_offset)    unsigned short NoFunc\_DF\_FragmentOffset = extendl\_16bit(ip\_packet.IPv4\_NoFunc, 15) + extendl\_16bit(DF, 14) + extendl\_16bit(MF, 13) + extendl\_16bit(FragmentOffset, 0);    memcpy(&buf[6], &NoFunc\_DF\_FragmentOffset, sizeof(NoFunc\_DF\_FragmentOffset)); //7已经被占用    //第9、10个byte    //unsigned short TimeTolive\_Protocol=extendl\_16bit(ip\_packet.IPv4\_TimeTolive,8)+extendl\_16bit(ip\_packet.IPv4\_Protocol,0);    unsigned short TimeTolive\_Protocol = extendl\_16bit(ip\_packet.IPv4\_Protocol, 8) + extendl\_16bit(ip\_packet.IPv4\_TimeTolive, 0);    memcpy(&buf[8], &TimeTolive\_Protocol, sizeof(TimeTolive\_Protocol)); //9已经被占用    //11 12    unsigned short HeaderCheckSum = ip\_packet.IPv4\_HeaderCheckSum;    memcpy(&buf[10], &HeaderCheckSum, sizeof(HeaderCheckSum)); //11已经被占用    //13 14 15 16    unsigned int SourceAddr = ip\_packet.IPv4\_SourceAddr;    memcpy(&buf[12], &SourceAddr, sizeof(SourceAddr)); //15已经被占用    //17 18 19 20    unsigned int DesAddr = ip\_packet.IPv4\_DesAddr;    memcpy(&buf[16], &DesAddr, sizeof(DesAddr)); //19已经被占用    // option    memcpy(&buf[20], IPv4\_Option, IPv4\_Option\_Len);    // data    memcpy(&buf[20 + IPv4\_Option\_Len], IPv4\_Data, IPv4\_Data\_Len);    //返回字节数    return 20 + IPv4\_Option\_Len + IPv4\_Data\_Len;    }  函数中的参数保存了IP数据包头部字段信息，示例如下：  struct IP\_Packet ip\_packet\_info = {0b0100, 0b1111, 0b00000000, //IPv4\_Version,IPv4\_IHL,IPv4\_TOS    0b0000000000000000, //IPv4\_TotalLength    0b0000000000000000, //IPv4\_Identification    0b0, 0b0, 0b0, 0b0000000000000, //IPv4\_NoFunc,IPv4\_DF,IPv4\_MF,IPv4\_FragmentOffset    0b01010101, 0b10101010, //IPv4\_TimeTolive,IPv4\_Protocol    0b0000000000000000, //IPv4\_HeaderCheckSum    0b11011010010001101111111101100001, //IPv4\_SourceAddr //218.70.255.97    0b01110010001101110101111100001110, //IPv4\_DesAddr //114.55.95.14    0b0, //IPv4\_Option    0b0}; //IPv4\_Data 3.2隧道封装模块 此模块对IP数据包进行进一步的封装，首先从网络接口层解析出IP数据包，然后进行隧道封装，将其封装进一个新的IP数据包。  unsigned char ipinip\_buffer[1500 + 60]; //存放ip in ip数据包  unsigned int IpinIpPacketLen=MakeIpPacket(DF, MF, FragmentOffset, ip\_packet\_info, ipinip\_buffer, ip\_packet\_info.IPv4\_Option, 40, ipv4\_buffer, RestByte+60);  socket\_send\_len += datalink\_layer\_send(ipinip\_buffer, IpinIpPacketLen, sockfd);  其中变量用于保存将IP数据包封装后形成的IPINIP数据包，IPINIP数据包的最大大小为1560 Bytes。其中1500 Bytes用于payload字段，20 Bytes应用头部除可选项以外的字段，身下的40 Bytes是用于可选项的。   3.3隧道解封模块 对于数据包的解析，网络层接收从网络接口层接收数据包，然后对IPINIP数据包的头部进行解析。头部每个字段的长度都是确定的，所以解析较为方便。  解析IPINIP数据包的部分代码如下，其中为从网络接口层接收数据包：  //IPinIP\_Version  unsigned char IPinIP\_Version;  memcpy(&IPinIP\_Version, &payload[0], 1);  printf("IPinIP\_Version: ");  PrintBinary((int)IPinIP\_Version, 7, 4);  printf("**\n**");    //IPinIP\_IHL  unsigned char IPinIP\_IHL;  memcpy(&IPinIP\_IHL, &payload[0], 1);  printf("IPinIP\_IHL: ");  //PrintBinary((int)IPinIP\_Version,3,0); // 0 byte  PrintBinary((int)IPinIP\_Version, 7, 0); // 0 byte  printf("**\n**");  IPinIP\_IHL = IPinIP\_IHL << 4;  IPinIP\_IHL = IPinIP\_IHL >> 4;    //IPinIP\_TOS  unsigned char IPinIP\_TOS;  memcpy(&IPinIP\_TOS, &payload[1], 1);  printf("IPinIP\_TOS: ");  PrintBinary((int)IPinIP\_TOS, 7, 0); // 1 byte  printf("**\n**");    //IPinIP\_TotalLength  unsigned short IPinIP\_TotalLength;  memcpy(&IPinIP\_TotalLength, &payload[2], 2);  printf("IPinIP\_TotalLength: ");  PrintBinary((int)IPinIP\_TotalLength, 15, 0); // 2,3 byte  printf("**\n**");    //IPinIP\_Identification  unsigned short IPinIP\_Identification;  memcpy(&IPinIP\_Identification, &payload[4], 2);  printf("IPinIP\_Identification: ");  PrintBinary((int)IPinIP\_Identification, 15, 0); // 4,5 byte  printf("**\n**");    //IPinIP\_NoFunc  unsigned short IPinIP\_NoFunc;  memcpy(&IPinIP\_NoFunc, &payload[6], 2);  printf("IPinIP\_NoFunc: ");  PrintBinary((int)IPinIP\_NoFunc, 15, 15); // 6,7 byte  printf("**\n**");    //IPinIP\_DF  unsigned short IPinIP\_DF;  memcpy(&IPinIP\_DF, &payload[6], 2);  printf("IPinIP\_DF: ");  PrintBinary((int)IPinIP\_DF, 14, 14); // 6,7 byte  printf("**\n**");    //IPinIP\_MF  unsigned short IPinIP\_MF;  memcpy(&IPinIP\_MF, &payload[6], 2);  printf("IPinIP\_MF: ");  PrintBinary((int)IPinIP\_MF, 13, 13); // 6,7 byte  printf("**\n**");    //IPinIP\_FragmentOffset  unsigned short IPinIP\_FragmentOffset;  memcpy(&IPinIP\_FragmentOffset, &payload[6], 2);  printf("IPinIP\_FragmentOffset: ");  PrintBinary((int)IPinIP\_FragmentOffset, 12, 0); // 6,7 byte  printf("**\n**");    //IPinIP\_TimeTolive  unsigned char IPinIP\_TimeTolive;  memcpy(&IPinIP\_TimeTolive, &payload[8], 1);  printf("IPinIP\_TimeTolive: ");  PrintBinary((int)IPinIP\_TimeTolive, 7, 0); // 8 byte  printf("**\n**");    //IPinIP\_Protocol  unsigned char IPinIP\_Protocol;  memcpy(&IPinIP\_Protocol, &payload[9], 1);  printf("IPinIP\_Protocol: ");  PrintBinary((int)IPinIP\_Protocol, 7, 0); // 9 byte  printf("**\n**");    //IPinIP\_HeaderCheckSum  unsigned short IPinIP\_HeaderCheckSum;  memcpy(&IPinIP\_HeaderCheckSum, &payload[10], 2);  printf("IPinIP\_HeaderCheckSum: ");  PrintBinary((int)IPinIP\_HeaderCheckSum, 15, 0); // 10,11 byte  printf("**\n**");    //IPinIP\_SourceAddr  unsigned int IPinIP\_SourceAddr;  memcpy(&IPinIP\_SourceAddr, &payload[12], 4);  printf("IPinIP\_SourceAddr: ");  PrintBinary((int)IPinIP\_SourceAddr, 31, 0); // 12,13,14,15 byte  printf("**\n**");    //IPinIP\_DesAddr  unsigned int IPinIP\_DesAddr;  memcpy(&IPinIP\_DesAddr, &payload[16], 4);  printf("IPinIP\_DesAddr: ");  PrintBinary((int)IPinIP\_DesAddr, 31, 0); // 16,17,18,19 byte  printf("**\n**");    //IPinIP\_Option  int IPinIP\_Option\_Len = (int)IPinIP\_IHL \* 4 - 20;  unsigned char IPinIP\_Option[40]; //最多40个Byte  memcpy(&IPinIP\_Option, &payload[20], IPinIP\_Option\_Len);  printf("IPinIP\_Option: ");  for (int i = 0; i < IPinIP\_Option\_Len; ++i)  {  PrintBinary((int)IPinIP\_Option[i], 7, 0); // 16,17,18,19 byte  }  printf("**\n**");    //IPinIP\_Data  int IPinIP\_Data\_Len = (int)IPinIP\_TotalLength - (int)IPinIP\_Option\_Len;  unsigned char IPinIP\_Data[1500]; //最多40个Byte  memcpy(&IPinIP\_Data, &payload[20 + IPinIP\_Option\_Len], IPinIP\_Data\_Len); 3.4 IP接收模块 IP接收模块接收被IPINIP隧道解析模块解析得到的，换言之，IP接收模块需要的数据就IPINIP数据包中的payload字段。  其实IP解析于IPINIP都解析很相似，不同之处在于IP接收模块还要进行数据分片的合并处理。  IP解析部分代码如下：  //IPv4\_Version    unsigned char IPv4\_Version;    memcpy(&IPv4\_Version, &IPinIP\_Data[0], 1);    printf("IPv4\_Version: ");    PrintBinary((int)IPv4\_Version, 7, 4);    printf("**\n**");        //IPv4\_IHL    unsigned char IPv4\_IHL;    memcpy(&IPv4\_IHL, &IPinIP\_Data[0], 1);    printf("IPv4\_IHL: ");    //PrintBinary((int)IPv4\_Version,3,0); // 0 byte    PrintBinary((int)IPv4\_Version, 7, 0); // 0 byte    printf("**\n**");    IPv4\_IHL = IPv4\_IHL << 4;    IPv4\_IHL = IPv4\_IHL >> 4;        //IPv4\_TOS    unsigned char IPv4\_TOS;    memcpy(&IPv4\_TOS, &IPinIP\_Data[1], 1);    printf("IPv4\_TOS: ");    PrintBinary((int)IPv4\_TOS, 7, 0); // 1 byte    printf("**\n**");        //IPv4\_TotalLength    unsigned short IPv4\_TotalLength;    memcpy(&IPv4\_TotalLength, &IPinIP\_Data[2], 2);    printf("IPv4\_TotalLength: ");    PrintBinary((int)IPv4\_TotalLength, 15, 0); // 2,3 byte    printf("**\n**");        //IPv4\_Identification    unsigned short IPv4\_Identification;    memcpy(&IPv4\_Identification, &IPinIP\_Data[4], 2);    printf("IPv4\_Identification: ");    PrintBinary((int)IPv4\_Identification, 15, 0); // 4,5 byte    printf("**\n**");        //IPv4\_NoFunc    unsigned short IPv4\_NoFunc;    memcpy(&IPv4\_NoFunc, &IPinIP\_Data[6], 2);    printf("IPv4\_NoFunc: ");    PrintBinary((int)IPv4\_NoFunc, 15, 15); // 6,7 byte    printf("**\n**");        //IPv4\_DF    unsigned short IPv4\_DF;    memcpy(&IPv4\_DF, &IPinIP\_Data[6], 2);    printf("IPv4\_DF: ");    PrintBinary((int)IPv4\_DF, 14, 14); // 6,7 byte    printf("**\n**");        //IPv4\_MF    unsigned short IPv4\_MF;    memcpy(&IPv4\_MF, &IPinIP\_Data[6], 2);    printf("IPv4\_MF: ");    PrintBinary((int)IPv4\_MF, 13, 13); // 6,7 byte    printf("**\n**");        //IPv4\_FragmentOffset    unsigned short IPv4\_FragmentOffset;    memcpy(&IPv4\_FragmentOffset, &IPinIP\_Data[6], 2);    printf("IPv4\_FragmentOffset: ");    PrintBinary((int)IPv4\_FragmentOffset, 12, 0); // 6,7 byte    printf("**\n**");        //IPv4\_TimeTolive    unsigned char IPv4\_TimeTolive;    memcpy(&IPv4\_TimeTolive, &IPinIP\_Data[8], 1);    printf("IPv4\_TimeTolive: ");    PrintBinary((int)IPv4\_TimeTolive, 7, 0); // 8 byte    printf("**\n**");        //IPv4\_Protocol    unsigned char IPv4\_Protocol;    memcpy(&IPv4\_Protocol, &IPinIP\_Data[9], 1);    printf("IPv4\_Protocol: ");    PrintBinary((int)IPv4\_Protocol, 7, 0); // 9 byte    printf("**\n**");        //IPv4\_HeaderCheckSum    unsigned short IPv4\_HeaderCheckSum;    memcpy(&IPv4\_HeaderCheckSum, &IPinIP\_Data[10], 2);    printf("IPv4\_HeaderCheckSum: ");    PrintBinary((int)IPv4\_HeaderCheckSum, 15, 0); // 10,11 byte    printf("**\n**");        //IPv4\_SourceAddr    unsigned int IPv4\_SourceAddr;    memcpy(&IPv4\_SourceAddr, &IPinIP\_Data[12], 4);    printf("IPv4\_SourceAddr: ");    PrintBinary((int)IPv4\_SourceAddr, 31, 0); // 12,13,14,15 byte    printf("**\n**");        //IPv4\_DesAddr    unsigned int IPv4\_DesAddr;    memcpy(&IPv4\_DesAddr, &IPinIP\_Data[16], 4);    printf("IPv4\_DesAddr: ");    PrintBinary((int)IPv4\_DesAddr, 31, 0); // 16,17,18,19 byte    printf("**\n**");        //IPv4\_Option    int IPv4\_Option\_Len = (int)IPv4\_IHL \* 4 - 20;    unsigned char IPv4\_Option[40]; //最多40个Byte    memcpy(&IPv4\_Option, &IPinIP\_Data[20], IPv4\_Option\_Len);    printf("IPv4\_Option: ");    for (int i = 0; i < IPv4\_Option\_Len; ++i)    {    PrintBinary((int)IPv4\_Option[i], 7, 0); // 16,17,18,19 byte    }    printf("**\n**");        //IPv4\_Data    int IPv4\_Data\_Len = (int)IPv4\_TotalLength - (int)IPv4\_Option\_Len;    unsigned char IPv4\_Data[1500]; //最多40个Byte    memcpy(&IPv4\_Data, &IPinIP\_Data[20 + IPv4\_Option\_Len], IPv4\_Data\_Len);  IP的分片的拼接，主要根据MF字段和DF字段进行拼接。如果MF的值为1，那么说明后面还有数据包的分片；如果MF的值为0，说明当前是最后一个分片，拼接结束。部分代码如下：  //开始拼接    unsigned short DF, MF, FragmentOffset;    DF = GetBinary(IPv4\_DF, 14, 14);    MF = GetBinary(IPv4\_MF, 13, 13);    FragmentOffset = GetBinary(IPv4\_FragmentOffset, 12, 0);    printf("MF= %d**\n**", MF);    if (MF == 1)    { //后面还有分片    UDP\_LEN += IPV4\_DATA\_MAXSIZE;    for (int i = FragmentOffset \* IPV4\_DATA\_MAXSIZE, l = 0; i < ((FragmentOffset + 1) \* IPV4\_DATA\_MAXSIZE - 1); i++, l++)    {    UDP\_PACKET[i] = IPv4\_Data[l];    }    }    else    { //最后一个分片    UDP\_LEN += IPv4\_Data\_Len;    for (int i = FragmentOffset \* IPV4\_DATA\_MAXSIZE, l = 0; i < (FragmentOffset \* IPV4\_DATA\_MAXSIZE + IPv4\_Data\_Len); i++, l++)    {    UDP\_PACKET[i] = IPv4\_Data[l];    }    }    transfer\_layer\_receriver(UDP\_PACKET, UDP\_LEN);    //结束拼接 | | | | | | |
| 四、实验结果及分析和（或）源程序调试过程4. IP隧道仿真程序部署 在文件根目录（即client.cpp或server.cpp的目录下）打开两个终端，按下步骤依次操作。  （1）在任意一个终端输入Make命令，生成输出文件；    （2）获取PC2，PC3，PC4的IP地址，在控制台输入命令”ifconfig”，其中ens33即为要使用的IP地址。    （3）开始连接四台电脑  ① 在PC4终端运行”./output/server”    ② 在PC3两个终端分别运行” ./output/server”与”./ouput/client [PC4的IP地址]”      ③ 在PC2两个终端分别运行” ./output/server”与”./ouput/client [PC3的IP地址]”    ④ 在PC1终端运行”./ouput/client [PC1的IP地址]”    这就就完成了四台电脑的Socket连接了，下面进行IP in IP隧道协议演示。 5. IP in IP隧道协议演示5.用户发送的信息 “This is a test message” （1）**PC1：**IP模块：用户发送的信息 “Play tiger tonight”，运行后的截图如下：    （2）**PC2：**IP in IP 封装模块    （3）**PC3：**IP in IP 解封模块    （4）**PC4：**IP接收模块   6. 总结 通过编制IP隧道仿真的四个程序：IP请求、隧道封装、隧道解封、IP接收等四个仿真程序，并且在四台电脑上进行IP隧道仿真程序部署，进行IP in IP隧道协议演示，更为彻底地了解和掌握了IP in IP协议。 | | | | | | |

备注：

1、教师在布置需撰写实验报告的实验前，应先将报告书上的“实验题目”、“实验性质”、“实验目的”、“实验项目内容”等项目填写完成，然后再下发给学生。

2、教师在布置需撰写报告的实验项目时，应告知学生提交实验报告的最后期限。

3、学生应按照要求正确地撰写实验报告：

* 1. 在实验报告上正确地填写“实验时间”、“实验地点”等栏目。
  2. 将实验所涉及的源程序文件内容（实验操作步骤或者算法）填写在“实验过程或算法（源程序）”栏目中。
  3. 将实验所涉及源程序调试过程（输入数据和输出结果）或者实验的分析内容填写在“实验结果及分析和（或）源程序调试过程”栏目中。
  4. 在实验报告页脚的“报告创建时间：”处插入完成实验报告时的日期和时间。
  5. 学生将每个实验完成后，按实验要求的文件名通过网络提交（上载）到指定的服务器所规定的共享文件夹中。每个实验一个电子文档，如果实验中有多个电子文档（如源程序或图形等），则用WinRAR压缩成一个压缩包文档提交，压缩包文件名同实验报告文件名（见下条）。
  6. 提交的实验报告电子文档命名为：“年级（两位数字不要“级”字）专业（缩写：计算机科学与技术专业（计科）、网络工程专业（网络）、信息安全专业（信息）、物联网工程（物联网））班级（两位数字）学号（八位数字）姓名实验序号（一位数字）．doc。如学号为20115676、年级为2011级、专业为“计算机科学与技术”专业、班级为“02班”、姓名为“王宇”的学生，完成的第一次实验命名为： 11计科02班20115676王宇1．Doc，以后几次实验的报告名称以此类推。

4、教师（或助教）在评价学生实验时，应根据其提交的其他实验相关资料（例如源程序文件等）对实验报告进行仔细评价。评价后应完成的项目有:

1. 在“成绩”栏中填写实验成绩。每次实验成绩按照十分制方式评分，取整数分值。
2. 在“教师评价”栏中“评价教师签名”填写评价教师（或助教）姓名。
3. 课程实验环节结束后，任课教师将自己教学班的实验报告文件夹进行清理。在提交文件夹中，文件总数为实验次数×教学班学生人数（如，教学班人数为90人，实验项目为5，其文件数为：90×5=450）。任课教师一定要认真清理，总数相符，否则学生该实验项目不能得分。最后将学生提交的实验报告刻光盘连同实验成绩一起放入试卷袋存档。