电 子 科 技 大 学

UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA

计算机网络与通信技术
项目一总报告

指 导 教 师　　徐杰

作 者 姓 名　　刘文晨 郝绮瑞 邓召麒 崔晨奇

班 组 号　　3 组

2019 年 10 月 20 日

引言

项目一的目的是实现数据通信在网络中的传输。本项目通过 C 语言和物理层模拟软件 3 完成。

作为网络编程的入门，本项目根据计通网的大纲从 OSI 的七层模型入手，尝试去逐步理解网络分层模型的搭建以及网络分层的必要性，并且利用一个网元的模型设计和编程实现。

我们将模型浓缩为应用层，网络层和物理层。利用套接字库实现不同网络层次间的通信连接，并且利用端口号的匹配，实现了我设计开发的程序与电子科技大学网络工程系计算机网络课程组的物理层比特流传输模拟软件的对接。最后，本项目进行了不同通信情况下的测试，验证本项目能够在各种复杂的网络下进行数据通信。

关键词：帧定位 差错控制 流量控制 数据的封装和解封 交换机 单播 广播 路由寻址 分组转发

目录

一、 任务要求及小组分工

1.1. 主题

设计具有多层结构的网元，并将多个网元构成一个网络，实现信息、文件在多个网元之间的传递。

1.2. 目标和要求

利用项目提供的物理层模拟软件，在两个只能传输 0 和 1 数据的程序（物理层实体的仿真）之间通过增加各种控制功能，形成多层结构网元，网元之间形成多种拓扑结构，通过网元传输包含中文、英文、标点符号（大约 50 个字符）和图片文件等，并编程实现。

- 在模拟的两个物理层实体之间需要实现：

1） 信息编码、差错控制、成帧；

成帧（帧同步）功能：在 0 1 变化的比特流中，能够准确定位有效数据的起始和终止，从而将有效数据提取出来。

2） 差错检测和控制功能：

设计差错校验技术，能够判断（1）中的提取数据是否正确

设计差错控制协议，能够让发送方重传错误的帧

或者，设计纠错编码，在收端纠正错误的数据位

3） 信息表示功能：设计编码方案，将中文、英文、标点符号混杂在一起的 50 个字符左右的信息编制成（二进制）比特流发送到对方，对方能够正确解读。

- 在多个网元构成的复杂系统中还需要实现

4） 交换及路由功能

利用多个物理层实体组成的一个网络系统。该系统能实现任意两个点之间传输一段信息或图片文件。要求完成的功能包括，但不限于：

给各节点编址，这样在用户程序中输入目标地址和文件名，就能将指定图片文件传输。

每组至少有四个节点，至少有两个节点之间需要经过另外至少一个节点的中介。

节点启动以后，人工设定或自动建立路由表，完成数据路由和转发。最终从源端到达目的端。

5） 端到端的寻址、控制和差错恢复功能

能够穿越多个网元构成的网络，实现端到端数据、文件可靠传输。

1.3. 小组分工

姓名	学号	小组分工
刘文晨	2018080901006	阶段一报告撰写、阶段三报告撰写、竞争式媒体访问控制协议仿真实验、网元全过程编程功能实现、网元总设计规划、期末报告撰写
郝绮瑞	2018110601003	阶段一报告撰写、阶段二报告撰写、收集阶段一、二、三、四资料，参与讨论

邓召麒	2018160701016	收集阶段一、二、三、四资料，参与讨论
崔晨奇	2018080901003	查阅资料，参与讨论

二、 阶段一：网络协议体系

2.1 综述

首先我们先对网元的概念进行了明确。网元由一个或多个机盘或机框组成，能够独立完成一定的传输功能。简单的来说，网元就是网络中的元素、网络中的设备，它能够完成某些功能模块。

2.2 对网元模型的设计

根据项目目标，项目要搭建的网元要能够作为信源、信宿终端，也可作为路由器，或二层交换机，这些功能就对网元的层次结构提出了要求。

要作为信源、信宿终端，网元就要包含应用层。应用层主要是与用户进行直接交互的层次，传输的数据是基于用户认知范围的字符，主要体现在各种运行在操作系统上的应用软件或者命令行等等。作为网络分层模型的最高层，应用层会产生数据并发给下层，下层传递来的数据的终端也是应用层，因而只有具备应用层，网元才能成为信源、信宿终端。

要作为信源、信宿终端，网元还要包含网络层。网络层会协调数据在网络中的传输，将数据从源端设法经过若干个中间节点传送到目的端。其具体功能包括寻址、路由选择、建立连接等等，由于网络结构复杂，网路中传输的数据众多，因而需要网络层对数据的传输做出调控，通过选择合适的路径，建立适当的连接，来将数据正确的发送到目的端。路由，是指分组从源端到目的端时，决定端到端

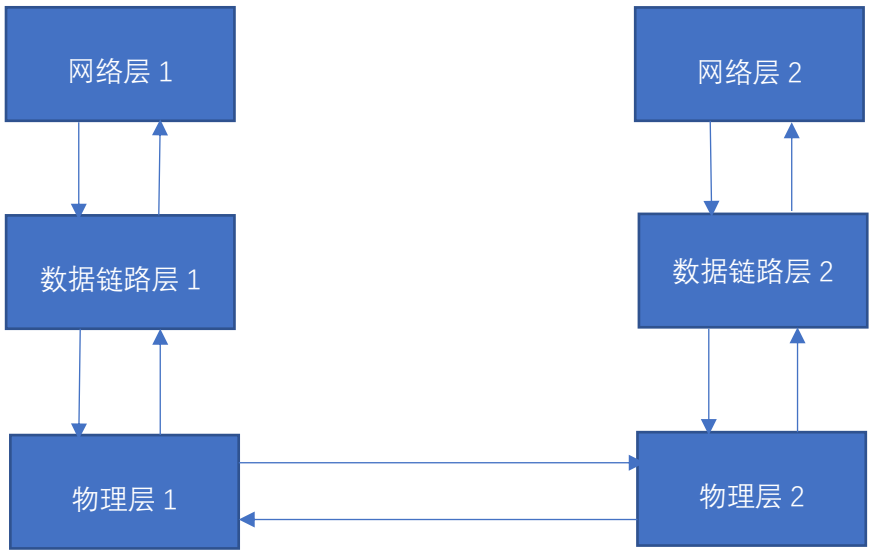
路径的网络范围的进程，也就是说，网元要具有路由器的功能的话，必须要有网络层。

同时，若此网元还要作为二层交换机，就必须具有数据链层。二层交换机是数据链层的设备，可以连接物理层和数据链层。二层交换技术可以识别数据包中的 MAC 地址信息，之后根据 MAC 地址转发数据包，并将这些 MAC 地址与对应的端口记录在二层交换机内部的地址表中。因此，网元要作二层交换机就还需具备数据链层。

最后，由于网元要基于 01 比特流信道传输数据，那就必须要具备物理层。物理层是网络模型中的最基础的一层，其作用就是传输 01 比特流数据。而本项目也是基于物理层模拟软件进行的，因此物理层也是网元中要包含的一层。

综上，本项目采用网络层-数据链路层-物理层的结构，用户在网络层输入信息，经数据链路层进行具体的处理之后传递给物理层之后发送给另一端。

项目一的模型层次如下所示：



2.3 网络层

我们设计的网络层事实上起的是应用层的作用，让用户可以选择三种状态，分别为接收信息状态，发送信息状态和广播状态，其中，在发送和广播状态，用户可以输入一个字符串或者是由文件或者图片转化成的二进制信息，然后根据 utf8 编码将这些信息全部转化为只包含 01 的比特流，之后发送给数据链路层，剩下的就由网元的其他部分完成。而选择了接收状态的网络层，就只需要等待对方网络层传输过来的信息即可。

网络层的基本框架为：

```
int main()
{
    初始化
    while(1)

    {
        选择状态：1.接收信息 2.发送信息 3.广播
        if 接收信息状态
        {
            告诉下面的数据链路层是接收信息状态
            等待接收数据链路层传过来的比特流
            将比特流转为字符串，输出
        }
        else if 发送信息状态
        {
            告诉下面的数据链路层是发送信息状态
            输入目的端口号
            告诉链路层该端口号
            输入要发的信息
            转为比特流后发给链路层
            等待链路层告知发送情况（对方已接收 or 对方接收失败）
        }
        else if 广播状态
        {
            告诉链路层是广播状态
            输入要发的信息
            转为比特流后发给链路层
            等待链路层告知发送情况（广播成功 or 广播失败）
        }
    }
    关闭套接字
}
```

2.4 数据链路层

这层受网络层支配，主要起一个信息交换编码解码校验的作用。

数据链路层的基本框架为：

```
int main()
{
    初始化
    while(1)
    {
        接收网络层的状态信息:1.接收信息状态 2.发送信息状态 3.广播状态
        if 接收信息状态:
        {
            while(1)
            {
                从物理层接收一帧的信息

                解帧
                if 出现错误
                {
                    发给物理层一个重传信息
                }
                else if 没出错
                {
                    发给物理层一个确认信息
```

```
                    if 为结束帧
                    {
                        按顺序拼接所有接收到的帧
                        break
                    }
                    else if 不为结束帧
                    {
                        continue
                    }
                }
            }

        }
        else if 发送信息状态
        {
            接收网络层的要传输的信息
            把信息存到帧结构里面去
            while(当前帧有效)
            {
                发送该帧
                接收物理层的信息(重传 or 确认)
                if 重传
                {
                    continue
                }
                else if 确认
                {
                    跳到下一帧
                }
            }
            告诉网络层对方已成功接收
        }
        else if 广播状态
        {
            接收网络层的要传输的信息
            把信息存到帧结构里面去

            while(当前帧有效)

            {
                发送该帧
                接收 n-1 次物理层的信息(n 为网元总数)
                if 有其中一个为重传
                {
                    continue
                }
                else if 所有都是确认信息
                {
                    跳到下一帧
                }
            }
            告诉网络层对方已成功接收
        }
    }
}
```

三、 阶段二：交互技术

3.1 帧同步

3.1.1 概念

在网络传输中，帧是最小的数据传输单位。数据包通过自上而下一层一层的处理和传递，当传输到数据链路层被处理之后，就叫做帧，之后数据就以一帧一帧的形式在信道上进行传输。

虽然理论上是这样的，但实际信道上传输的是 01 比特流，也就是说，是不间断的流的形式，因而数据块之间并没有间隔（除非使用块传输信道），所以我们就要从连续的数据中识别出一块一块的帧，这就需要帧同步了。

在发送端必须提供每帧的起始标记，在接收端检测并获取这一标志的过程称为帧同步。

帧同步就是发送端在发送时提供每帧的起始标记，然后接收端检测并获取这一标志，从而提取出一个个完整的帧，获得原本的数据包。

3.1.2 帧同步的方法

本项目的成帧过程是在数据段前面加上帧头，在帧头后面加上序列号，尾部加上帧尾，然后，再数据段和帧尾之间加上一个校验码，用于下一阶段的差错控制。

帧结构：

帧头	序号	数据	校验码	帧尾

我们设计的帧结构如上图所示，下面来详细介绍每一部分：

帧头、帧尾：8 位，均为“01111110”，用于帧同步；

序号：8 位，范围为 0-255，共 256 个，其中序号 0 代表结束帧，用于记录帧序号，防止乱序；

数据：32 位，即将信息通过 UTF-8 编码保存到这里；

校验码：8 位，用于差错检测（无纠错功能）。

具体的代码实现如下：

```
//帧封装
int frame(char *str)
{
    int target=FindTarget(str);
    printf("str'target is %d\n",target);
    char Str[10];
    memset(Str,0,sizeof(Str));
    if(target)
    {
        for(int i=0;i<=target;i++)
        {
            Str[i]=str[i];
        }
        Str[target+1]='0';
        if(target<7)
        {
            for(int i=target+1;i<=7;i++)
            {
                Str[i+1]=str[i];
            }
        }
    }
    else
    {
        strcpy(Str,str);
    }
    memset(Frame,0,sizeof(Frame));
    strcat(Frame,head);
    strcat(Frame,Str);
    strcat(Frame,Str);
    strcat(Frame,end);
}
```

由于通过物理信道后，传输的数据前后可能会加上一些干扰的 01 串，因此需要帧同步来找出帧的位置，这里我们用帧头帧尾来解决这个问题，帧头帧尾都为 01111110，我们可以保证帧内不包含和帧头帧尾一样的序列，而随机加上的干扰串出现和帧头帧尾相同序列的概率也非常小，因此我们只需要通过在帧的两头加上帧头和帧尾就可以实现帧同步。

3.2 差错控制

由于信道一般用模拟信号传输，且会有其他的一些噪音等的干扰存在，因此信道都会存在一定的误码率。

干扰有三种情况：

1） 将 0 干扰为 1；

2） 将 1 干扰为 0；

3） 某位的 0 或者直接被干扰而被丢弃。

这里我们不考虑最后一种情况，因此物理层的模拟软件也是没有比特丢弃这种可能的。而前面的两种情况我们判断传输过来的某个帧是否发生了误码，我们把除了帧头帧尾之外的所有序列都考虑进入校验码（不考虑帧头帧尾的原因是：

1） 如果帧头或者帧尾出错，那么大概率就找不到帧头或帧尾，也就无法识别出内容，那么这种情况显然要求重传;

2） 帧头帧尾序列都不变，加入到校验码也就等于没加入。

当另一端接收到信息后，按照对应的过程进行校验和解封，从而得到原始的信息，具体的代码实现如下：

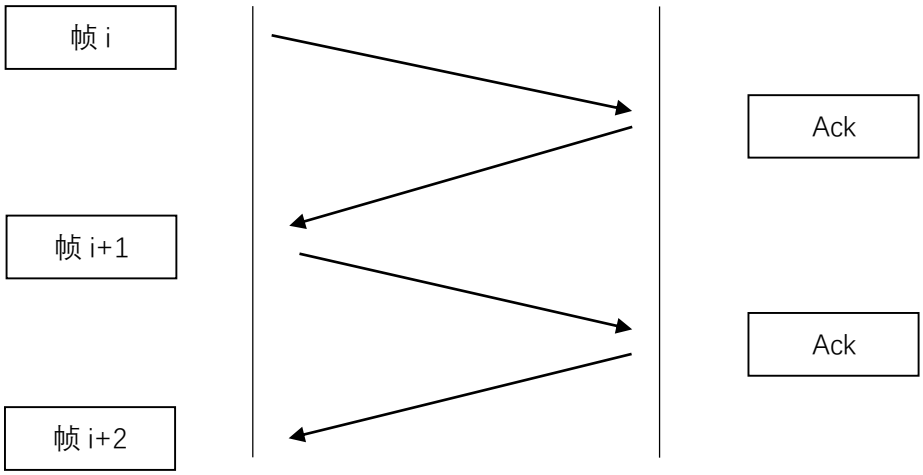
```
int FindFrame(char *str)
{
    int head=0;//找帧头
    while(str[head]=='1')
    {
        head++;
    }
    while(str[head++]=='0')
    {
        int one=0;
        while(str[head]=='1')
        {
            one++;
            head++;
        }
        if(one==6)
        {
            head++;
            break;
        }
    }
    int beginnum=head-8;
    int tail=head;//找帧尾
    while(str[tail]=='1')
    {
        tail++;
    }
    while(str[tail++]=='0')
    {
        int one=0;
        while(str[tail]=='1')
        {
            one++;
            tail++;
        }
        if(one==6)
        {
            tail-=8;
            break;
        }
    }
    int endnum=tail+8;
    int i;
    memset(RecFrame,0,sizeof(RecFrame));
    //将找出来的帧拷贝到接收帧中
    for(int i=beginnum;i<=endnum;i++)
    {
        RecFrame[i]=str[i];
    }
}
```

如果序号或者内容或者校验码的某位或者某几位出错了，明显会被校验出来，此时要求重传即可。

3.3 流量控制

流量控制的目的是为了更好的利用信道的资源，如果一次传输的数据过多，那么会有部分数据会被丢弃，这种情况是我们要避免的，但是如果数据过少，那么会造成信道资源的浪费，这种情况虽然有浪费，但是数据不会丢失，因此我们选择了后面这种方式，用停止等待协议来实现对流量的控制，具体操作为：每次只发一帧，等待对方传回来的确认应答信息我们再发下一帧。

具体的时序图如下所示：



通过这样的方式，我们实现了简单的流量控制。由于技术和时间等方面的问题，没有实现如滑动窗口协议等更优秀的流量控制。

四、 阶段三：共享技术

4.1 封装的数据结构

4.1.1 帧结构

帧头	源地址	序号	数据	目的地址	校验码	帧尾

我们设计的帧结构如上图所示：

帧头、帧尾、序号、数据、校验码的定义都和上述相同，在这里不赘述。

源地址、目的地址：24 位，事实上在项目一里，我们只是在一台电脑上面

做，因此这个源地址、目的地址事实上是源端口和目的端口，因为一般端口都是小于等于 5 位数字，因此我们将第一位数字映射为 8 位的比特位，其余四位映射位 4 位比特位，就构成了 24 位的地址，主要用于来源和目的的区分；

4.1.2 比特结构

用于保存由字符串转化为的比特流信息。

具体的代码实现如下：

```
struct Bits{
    char content[8+1];
    bool stats;
    void clear()
    {
        memset(content,0,sizeof(content));
        stats=false;
    }
};
```

4.1.3 应答结构

主要用于构建应答机制，即如果对方传过来的帧正确，则返回一个 ack1，表示正确，否则返回 ack0，表示数据有误，要求重传。

具体的代码实现如下：

```
struct Acks{
    char head[8+1];
    char src[24+1];
    char content[8+1];
    char dst[24+1];
    char checksum[8+1];
    char tail[8+1];
    bool stats;
    void clear()
    {
        stats=false;
        memset(src,0,sizeof(src));
        memset(content,0,sizeof(content));
        memset(dst,0,sizeof(dst));
        memset(checksum,0,sizeof(checksum));
    }
};
```

4.1.4 地址结构

传输一个地址信息，主要作用为让交换机知道要传的目的地址。

具体的代码实现如下：

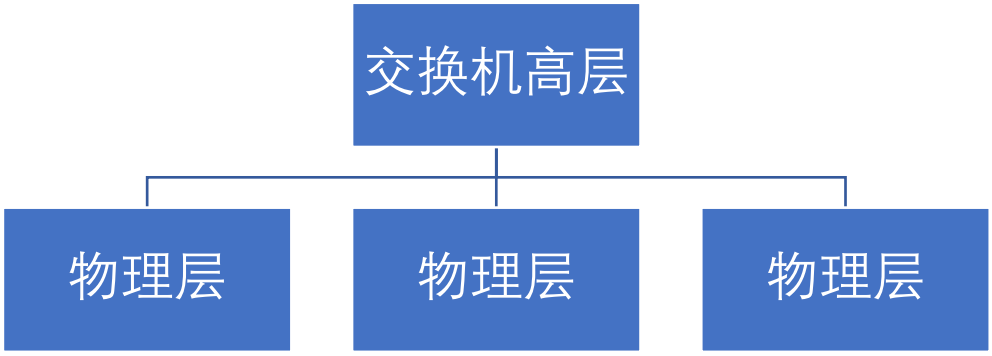
```
struct Address{
    char head[8+1];
    char content[24+1];
    char checksum[8+1];
    char tail[8+1];
    void clear()
    {
        memset(content,0,sizeof(content));
        memset(checksum,0,sizeof(checksum));
    }
};
```

这就是所有定义的数据结构，有了这些数据结构，也就有了编码和解码的规范。

4.2 交换机

我们一开始的想法是只用一个物理层来表示交换机，但是，如果这样的话，

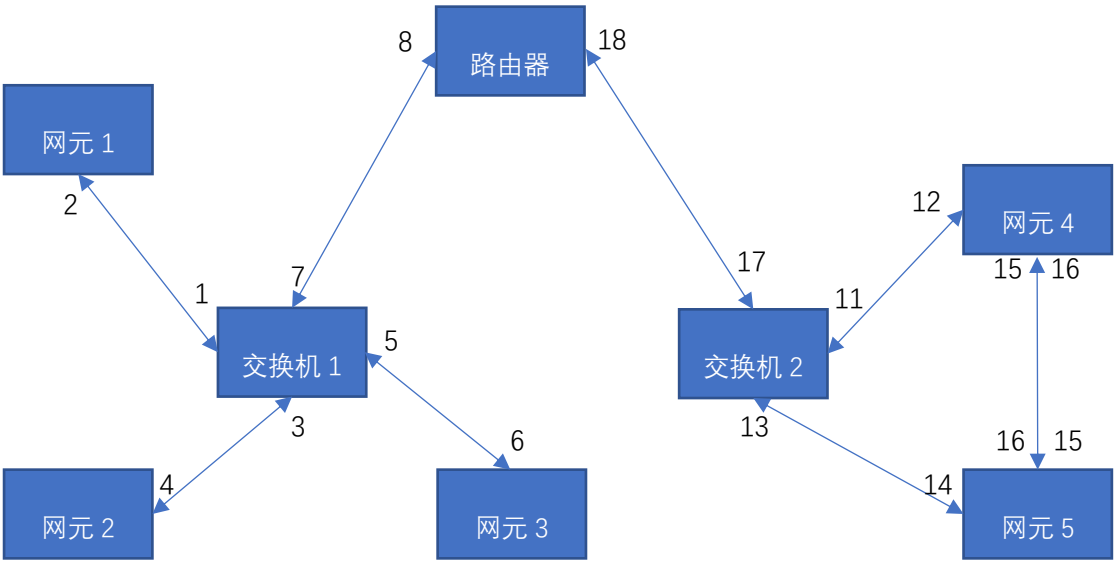
交换机无论接收到哪个发来的信息，就都会发给其他所有在交换机端口列表里面的端口，后来就想到可以用 n 个物理层加上一个上层来表示交换机，n 为交换机端口列表里面的端口数，因为我们这里采用的是 3 个网元通信的结构，所以这里 n 取 3，一个交换机模型如下图所示：



这样做的好处是，每一个物理层对接一个网元，使得物理层之间不会串发信息，而是将信息提到高层来，高层来决定接下来信息要传给哪个网元。

4.3 拓扑结构

本项目的拓扑结构如下所示：



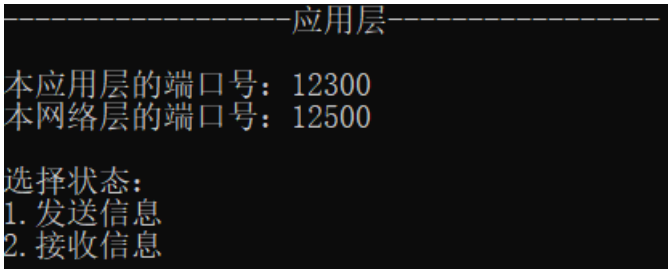
4.4 点播

点播，就是信息的点对点发送，在任一网元的网络层选择一个目的端口进行信息发送。

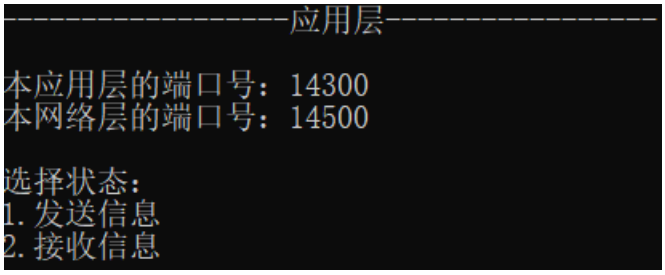
具体的过程如下所示：

初始状态：各个应用层初始化自己的应用层端口号和网络层端口号。各个网络层初始化自己的应用层端口号、网络层端口号和网络层端口号。各个交换机绑定自己的端口号。

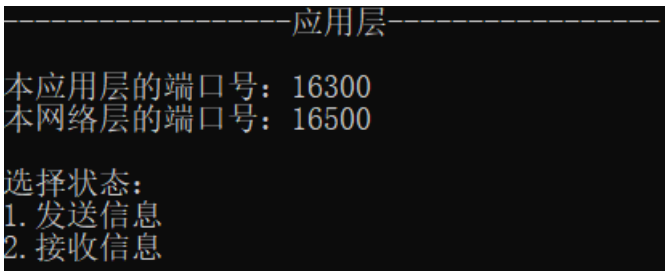
app1.1：



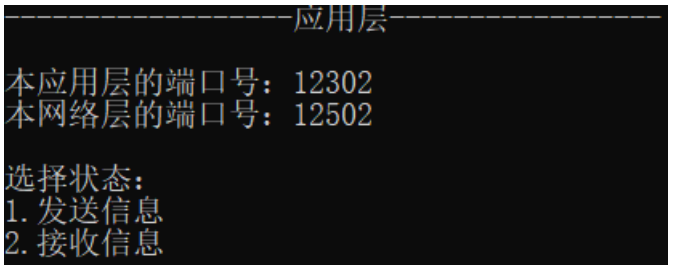
app1.2：



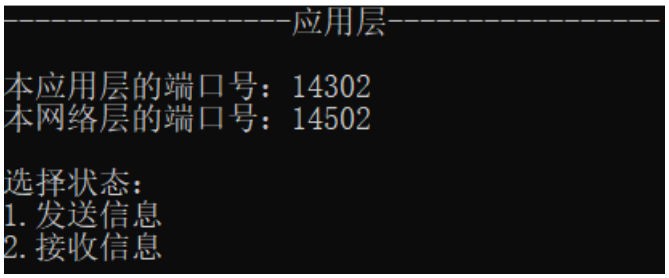
app1.3：



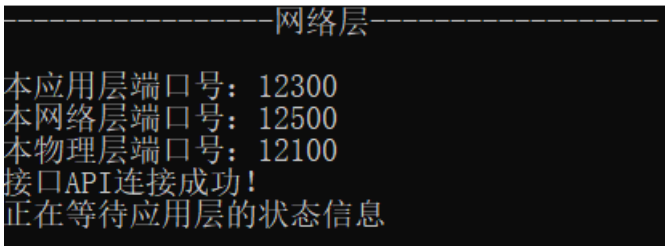
app2.1：



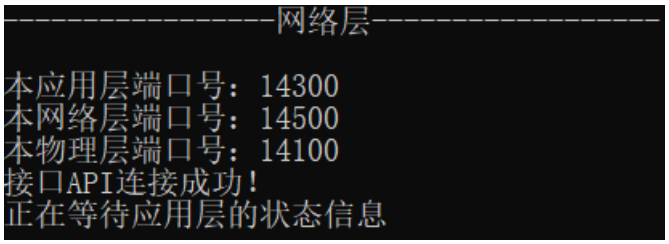
app2.2：



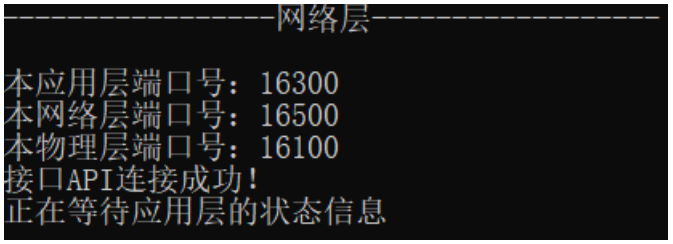
net1.1：



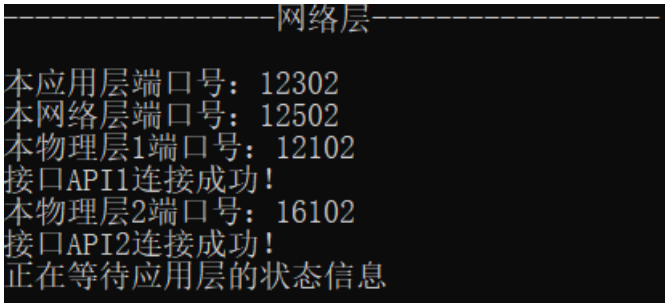
net1.2：



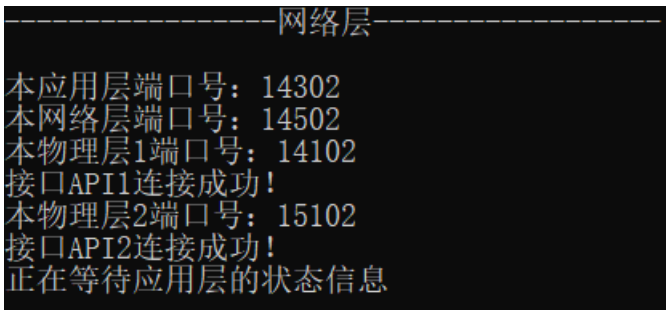
net1.3：



net2.1：

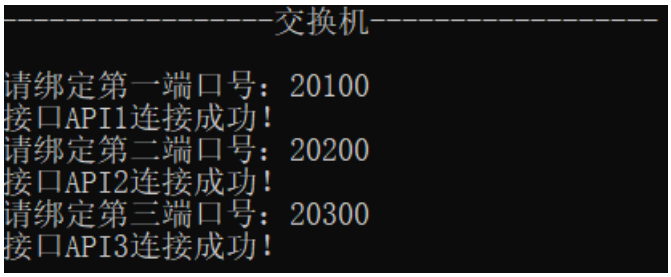


net2.2：



switches1：

switches2：



发送：app1.1 发送信息给 app1.2。

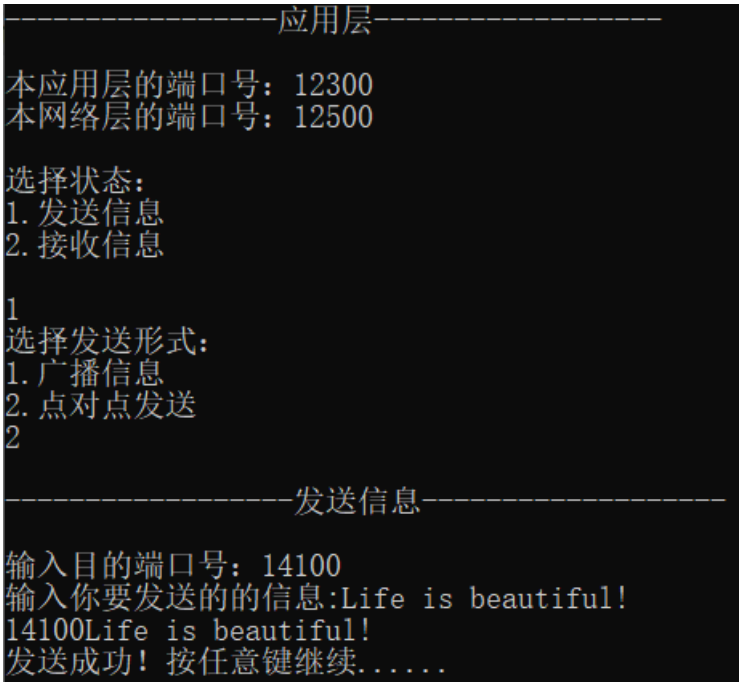
流程共四步：

1） 选择状态：1.发送信息；

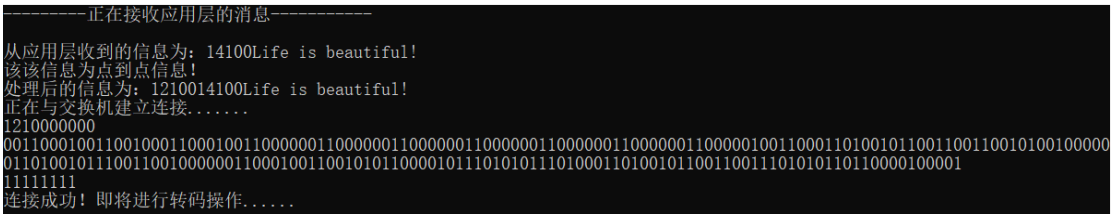
2） 选择发送方式：2.点对点发送；

3） 输入目的端口号；

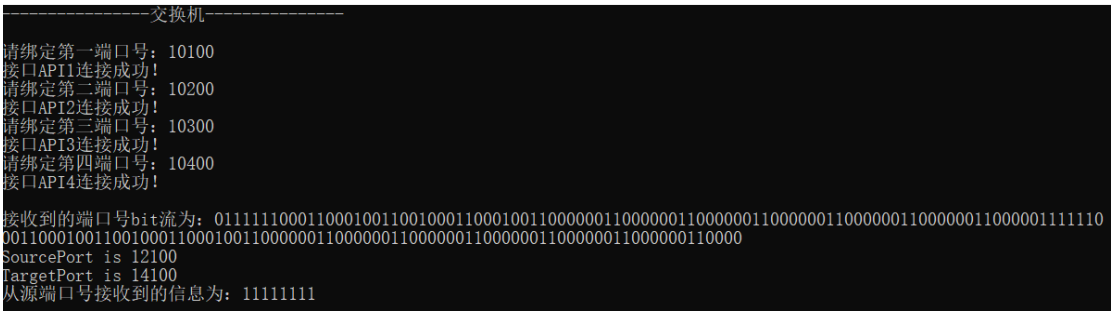
4） 输入发送信息。



net1.1:



交换机 switch1：



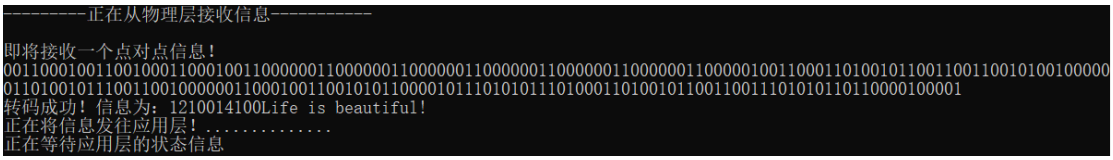
接收：app1.2 接收来自 app1.1 的信息。

流程共两步：

1） 选择状态：2.接收信息；

2） 接收信息。

net1.2：



4.5 广播

广播，就是在一个网元发送消息，在与交换机相连的其他网元都能接收到这个信息的一种方式。

具体过程如下所示：

初始状态：同点播。

发送：app1.1 发送信息给 app1.2 等其他 4 个网元。

流程共三步：

1） 选择状态：1.发送信息；

2） 选择发送方式：1.广播信息；

3） 输入发送信息。

```
------------------应用层-------------------

本应用层的端口号：12300
本网络层的端口号：12500

选择状态：
1.发送信息
2.接收信息

1
选择发送形式：
1.广播信息
2.点对点发送
1

------------------发送信息-------------------

输入你要发送的的信息:Life is beautiful!
00000Life is beautiful!
发送成功！按任意键继续.....
```

net1.1：

```
---------正在接收应用层的消息-----------

从应用层收到的信息为：00000Life is beautiful!
该信息为群发消息！
处理后的信息为：1210000000Life is beautiful!
正在与交换机建立连接.......
1210000000
00110001001100100011000100110000001100000011000000110000001100000011000000100110001101001011001100110010100100000
01101001011100110010000001100010011001010110000101110101011010001101001011001100111010101101100001000001
11111111
连接成功！即将进行转码操作.....

群发成功！
断开连接成功！
正在等待应用层的状态信息
```

交换机 switch1：

```
----------------交换机---------------

请绑定第一端口号：10100
接口API1连接成功！
请绑定第二端口号：10200
接口API2连接成功！
请绑定第三端口号：10300
接口API3连接成功！
请绑定第四端口号：10400
接口API4连接成功！

接收到的端口号bit流为：01111110001100010011001000110001001100000011000000110000001100000011000000110000001111110
00110001001100100011000100110000001100000011000000110000001100000011000000110000
SourcePort is 12100
收到了群发命令
从源端口号接收到的信息为：11111111
```

交换机 switch2：

```
----------------交换机----------------

请绑定第一端口号：20100
接口API1连接成功!
请绑定第二端口号：20200
接口API2连接成功!
请绑定第三端口号：20300
接口API3连接成功!

接收到的端口号bit流为：0111111000110001001100100011000100110000001100000011000000110000001100000011000000111111000110001001100100011000100110000001100000011000000110000001100000011000000110000
SourcePort is 12100
收到了群发命令
从源端口号接收到的信息为：11111111
```

接收：app1.2 等其他 4 个网元接收来自 app1.1 的信息。

流程共两步：

1） 选择状态：2.接收信息；

2） 接收信息。

app1.2：

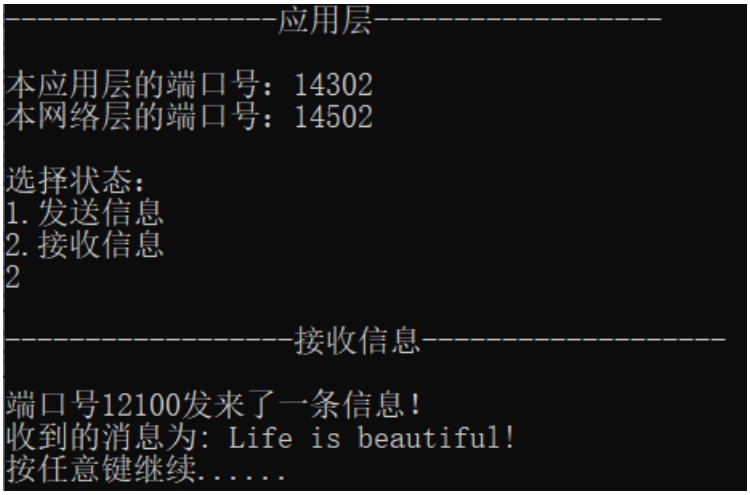
```
----------------应用层----------------

本应用层的端口号：14300
本网络层的端口号：14500

选择状态：
1.发送信息
2.接收信息
2

----------------接收信息----------------

端口号12100发来了一条信息!
收到的消息为: Life is beautiful!
按任意键继续.....
```

app1.3：

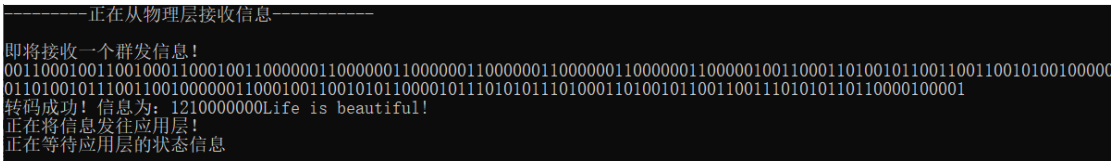
```
----------------应用层----------------

本应用层的端口号：16300
本网络层的端口号：16500

选择状态：
1.发送信息
2.接收信息
2

----------------接收信息----------------

端口号12100发来了一条信息!
收到的消息为: Life is beautiful!
按任意键继续.....
```

app2.1：

app2.2：



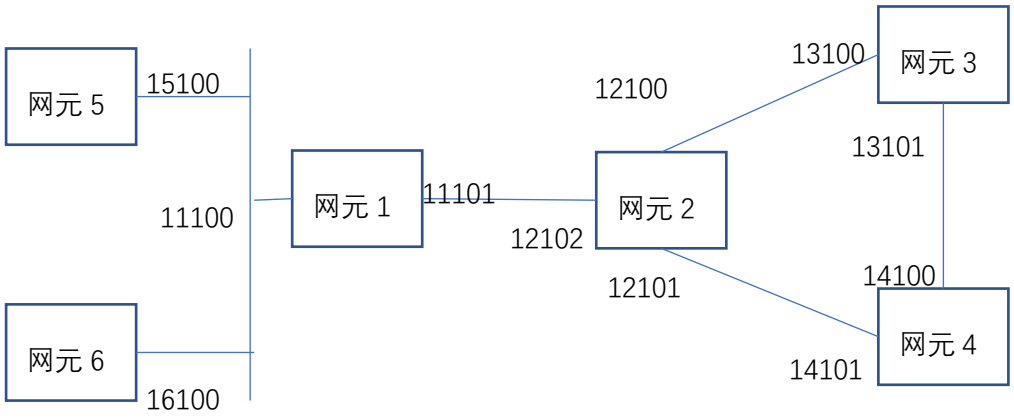
net1.2 等四个网络层：



五、 阶段四：交换与路由技术

5.1 配置文件拓扑图

本项目采用六个网元，混合组网的方式实现网元之间的相互通信。拓扑图如下所示：



其中，网元 1 是交换机，网元 2，3 和 4 是路由器，网元 5 和 6 是 PC。网元 5 和网元 6 通过共享信道与网元 1 进行连接。

基于六个网元混合组网的规划形式，能够较好地检测数据的传输结果和传输过程的完整性。

5.2路由器的四大功能

5.2.1 网关

路由器是工作在网络层，在网络层实现数据的转发等。路由器的每一个端口都有一个 IP 地址，连接子网的端口充当网关的作用。作用可以分为两点：

1）判断报文的接收方是否在一个子网中，如若在一个子网则不进行路由的转发。

2）接受子网的报文，充当门的作用，连接不同子网。

5.2.2 路径寻址转发

路由器还必须具有报文的转发功能，在路由器查询自己的路由表后，会进行指定端口的转发，将报文发送到网络中，实现两者之间的通信。因此，路由器除具备一般主机和交换机的功能外，还需要具备分组转发功能。

5.2.3 ARP 表

ARP 表是获取到的最近一段时间内使用过的 IP 地址与 MAC 地址的对应关系。ARP 表能方便报文的封装和数据的传输。

5.2.4 路由表

路由表存储着指向特定网络地址的路径。路由表中含有网络周边的拓扑信息。路由表建立的主要目标是为了实现路由协议和静态路由选择。路由表可以实现报文的分组转发，通过路由寻址技术实现报文的准确发送。报文在经过路由器后。路由器会在数据链路层修改源 MAC 地址，将其修改为自己的 MAC 地址，然后查询路由表，进行下一跳的转发。

5.3 PC、交换机、路由器的协议体系结构

网元 5 和网元 6 都为一般 PC，每个 PC 都有一个 IP 地址、一个 MAC 地址和一个物理层端口号。在 IP/TCP 五层模型中， 网络层实现数据的封装、解封和校验等功能。同时，网络层需要实现一个 ARP 表，该表实现了目的 IP 地址-目的 MAC 地址-源 MAC 地址的映射信息，每一次收回到 ACK 数据包都会更新该 ARP 映射表。IP 地址和 MAC 地址都是 2 个字节即 16 个 bit 的数字编码，存储的目的 IP 地址和目的 MAC 地址都为接收方的信息，ARP 表如下所示：

目的 IP 地址	目的 MAC 地址	源 MAC 地址
12300	12200	16200
13300	12200	16200

交换机在数据链路层实现数据的端口转发，通过报文的一次次转发，从而在交换机内部形成一个 MAC 映射表，实现从 MAC 地址到端口号的映射。MAC 地

址为转发的报文的目的 MAC 地址，端口号为数据转发的物理层端口。MAC 表如下所示：

MAC 地址	物理层端口
13200	12100
14200	12101
16200	12102

路由器需要实现数据的分组转发，路由寻址技术。所以，路由器具有一个路由表，路由表如下所示：

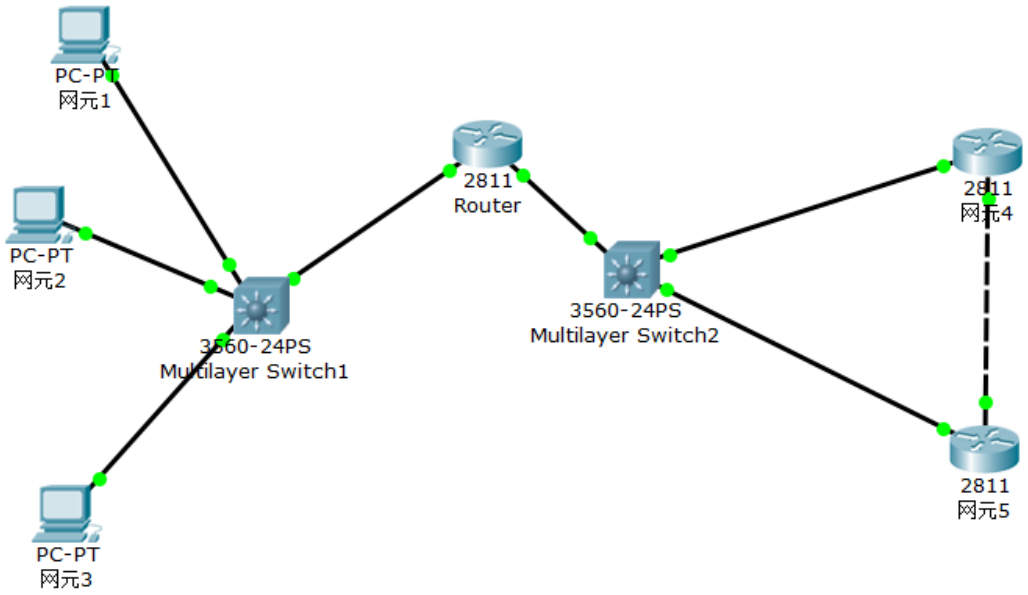
目的 IP 地址	MAC 地址	下一跳	级数 Metric
13200	12100	0	1
14200	0	12300	2

同时，路由器还具备一个 ARP 映射表，该表实现目的 IP 地址-目的 MAC 地址-源 MAC 地址的映射。ARP 表如下所示：

目的 IP 地址	目的 MAC 地址	源 MAC 地址
13300	15200	11200
15300	14201	11200

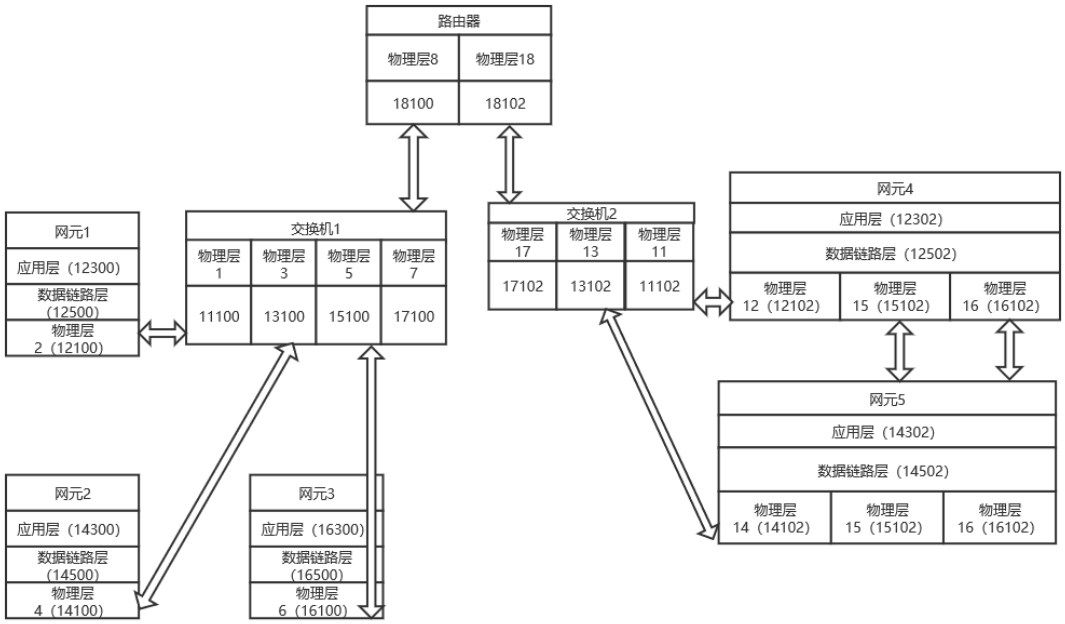
在混合子网中，路由器可以实现报文转发功能，路由器基于路由表实现三层到二层的寻址技术和路由分组转发功能。路由器的转发原理基于现代 IPv4 的通信原理，能够较好地实现在复杂网络中的数据通信功能。路由器的内部有两个表，路由表和 ARP 表。当路由器网元自身进行数据发送或接收时，首先使用 ARP 表，如若通过查询 ARP 表查询到目的 IP 地址的 MAC 地址，则封装进去并进行路由表的查询。查询到转发的 MAC 地址或下一跳后，进行报文的转发。当路由器仅实现数据转发时，则充当网关的作用，查询自己的路由表并实现转发。如若未找到对应的路径，则直接丢弃报文。三个路由器连接成环状，能够实现三层到二层寻址技术。

5.4 Packet Tracer 场景搭建



5.5 各网元协议体系

阶段四的网元架构是阶段三拓扑结构的补充，方案如下：



如图所示，网元 1，2，3，4，5 为 PC，所以具备一个 IP 地址和一个 MAC 地址。交换机有 2 个，工作在数据链路层，不具有 IP 地址，具有一个 MAC 地址。路由器具有一个 IP 地址，每一个端口都具有一个 MAC 地址和物理层端口号。

5.6 路由器的框架

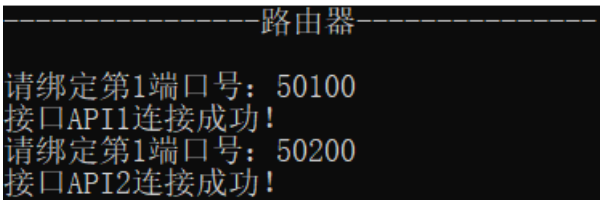
为了实现上述功能，路由器的框架如下所示：

```
int main()
{
    int RoutingList[][];
    //路由表,存储的是交换机端口号和交换机下的所有网元端口号
    初始化路由表

    循环绑定端口号
    {
        if 有接收
        {
            接口API关联成功!
        }
        else
        {
            接口API关联失败!
        }
    }
    while(1)
    {
        排队检测每个端口是否接收到了信息
        {
            if 超时
            {
                跳过本端口检查下一个端口
            }
            if 监测到有接收信息
            {
                if 该消息为群发消息
                {
                    循环将信息发送至每个端口
                    断开连接
                }
                提取有效信息
                解封
                查路由表寻找目的端口号的交换机序号
                if 查不到
                {
                    break;
                }
                else
                {
                    与目的交换机建立连接
                }
            }

        }
        return 0;
    }
```

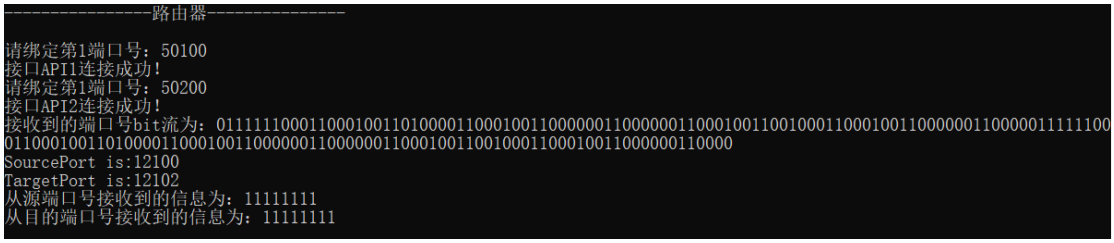
5.7 路由器在点播和广播的作用

5.7.1 点播

初始状态：

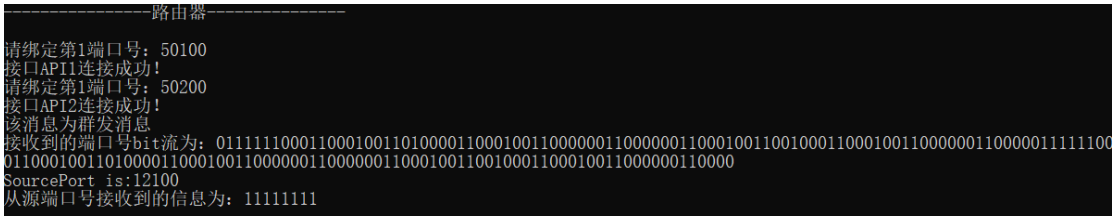


发送端为 app1.1，接收端为 app2.1。其过程需要经过两个交换机和一个路由器。



5.7.2 广播

发送端为 app1.1，接收端为 app1.2 等其他 4 个网元。其过程需要经过两个交换机和一个路由器。



六、 遇到的问题及解决方法

6.1 广播

在广播的时候考虑了差错控制，这就带来了一个问题：到底应该怎么差错控制呢，因为会出现这种情况：A 广播，B、C 接收，对于某一帧，如果 B 确认了，

返回给 A 一个确认的应答信息，但是 C 发现帧有错，那应该怎么做呢，一开始想的是对 B、C 分开考虑，也就是这时候给 B 发下一帧，给 C 重传，但是如果这样的话，会带来两个问题：

1) 需要根据 B、C 传过来的 Ack 应答信息的源端口来判断哪个是 B 的 Ack、哪个是 C 的 Ack，如果这时候发现某个 Ack 是错的怎么办，这个问题现在较难解决
2) 这是广播状态，A 发什么 B、C 都会接收到.因此，如果我这时候需要发一个上一个帧给 C，但是这个帧 B 已经接收过，那么也会造成浪费。

解决方法是：只要 B、C 有一个需要重传，那么就重传，否则发下一帧。

七、 可改进之处

1) 在网络层实现竞争协议等适应现代复杂网络情况的协议信息；
2) 无法实现交换机同时处理两个及以上信息传输，只能等待交换机一次传输完毕再进行下一次的信息传输；
3) 网络层及数据链路层的很多部分都很类似，写出来的代码较长，结构不清晰，可以考虑将某些重用多的地方封装成一个函数，甚至是一个类；
4) 可以增加相应的图片或者文件文档的编解码方式，而不是只能在网络层输入字符串来传输；
5) 在本项目的基础上，在网络层和传输层添加更多协议信息，实现更多的功能，构建出一个更完善的数据传输系统。

八、 总结反思与心得体会

本项目结合课堂知识和网络知识，以 TCP/IP 五层模型为基础，主要在网络层、数据链路层和物理层实现了数据的发送、传输和接收等一系列功能，包括帧定位，差错控制，流量控制，数据的封装和解封，交换机，单播，广播，路由寻址和分组转发……

本项目历时三个月有余，是《计算机网络与通信技术》课程最具挑战性的项目。在完成本项目的过程中，我们锻炼了我们的编程能力，查询资料的能力和小组合作能力。尽管本项目并非尽善尽美，但只要我们尽力做到最好，就无怨无悔。

感谢老师们的悉心教导！在学习《计算机网络与通信技术》课程的过程中，我们发现老师的谆谆教诲是我们不断学习的指路明灯，老师的尽职尽责是激励我们不断奋进的动力源泉。无论线上还是线下，老师们总能耐心地解答学生们的问题，亦师亦友的师生关系让我们的交流毫无障碍。

再次感谢老师们孜孜不倦的指导，师恩难忘！