

计算机网络与通信技术

最小网元设计总报告

段班 7 组制

2019 年 12 月 20 日

目录

一、小组分工.....	4
二、设计：分层方案.....	4
2.1 层次划分.....	4
2.2 功能设计.....	4
2.3 端口分配方案.....	5
三、交互.....	5
3.1 帧定位.....	5
3.1.1 定义.....	5
3.1.2 方案.....	6
3.1.2 函数实现.....	6
3.1.3 测试.....	6
3.2 差错检测.....	6
3.2.1 定义.....	6
3.2.2 方案.....	7
3.2.3 函数实现.....	7
3.2.4 测试.....	7
3.3 差错控制.....	7
3.3.1 定义.....	8
3.3.2 方案.....	8
3.3.3 函数实现.....	8
3.3.4 测试.....	9
3.4 流量控制.....	9
3.5 函数封装.....	9
四、共享.....	10
4.1 按目的转发.....	10
4.1.1 定义.....	10
4.1.2 方案.....	10
4.1.3 代码实现.....	10
4.1.4 测试.....	11
4.2 反向地址学习.....	11
4.2.1 定义.....	11
4.2.2 方案.....	11
4.2.3 函数实现.....	12
4.2.4 测试.....	12
4.3 未知广播.....	12
4.3.1 定义.....	12
4.3.2 方案.....	12
4.3.3 代码实现.....	12
4.3.4 测试.....	13
五、路由.....	13
5.1 路由表.....	13
5.1.1 定义.....	13

5.1.2 方案.....	13
5.1.3 代码实现.....	14
5.1.4 测试.....	14
5.2 基于静态路由的寻址与转发.....	14
5.2.1 定义.....	14
5.2.2 方案.....	14
5.2.3 函数实现.....	15
5.2.4 测试.....	15
六、应用.....	15
6.1 能传信息.....	15
6.1.1 随机比特流.....	15
6.1.2 字符.....	16
6.1.3 图片.....	16
6.2 多种拓扑结构适应性.....	17
6.2.1 环状.....	17
6.2.2 混合组网.....	18
七、混合组网.....	18
7.1 概述.....	18
7.2 层次化设计.....	18
7.3 拓扑图.....	19
7.4 路由表分配.....	19
7.5 测试.....	20
八、附件说明.....	21
附件一：源码.....	21
附件二：项目日志.....	21
附件三：交换机成环 Demo.....	21
附件四：图片传输 Demo.....	21
附件五：混合组网 Demo.....	21

一、小组分工

于汇洋 (2018150801020): 分层方案、帧定位、按目的转发、路由表、基于静态路由的寻址与转发、测试阶段四代码、撰写相应报告。

张鸿飞 (2018010801027): 分层方案、差错检测、未知广播、字符编码方案、图片编码方案、测试阶段二代码、撰写相应报告。

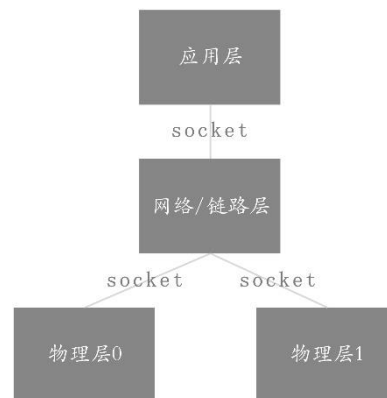
张国胜 (2018080901024): 分层方案、差错控制、反向地址学习、交换机成环、混合组网、代码框架、总报告撰写。

二、设计：分层方案

2.1 层次划分

通过对OSI模型以及TCP/IP模型的分析，项目指导书的阅览以及 对相关协议体系文献的阅览,我们针对于本项目(最小网元设计)，将模型划分为三层：应用层、网络层（链路层）和物理层。模型结构如右图所示。

应用层和网络层我们将继续编写相应的软件进行实现,物理层我们将使用项目所提供的物理层模拟软件进行模拟。(在本项目中,我们将网络层和链路层功能集于一体,具体区分如下述)



2.2 功能设计

a. 应用层

应用层首先实现和用户的交互功能，既能够接受用户输入的指令，又能够将收到的信息反馈给用户；其次，应用层还应实现和下层的通信，既可以将用户输入的信息传递给下层，又能够接受下层传入的信息。

具体功能如下：

- 允许自动发送随机生成的比特流（便于测试）
- 字符编码/译码方案
- 图片编码/译码方案
- 正确处理确认帧（基于停止等待协议）

b. 网络层（链路层）

本层首先也要实现和上层（应用层）的通信功能，既可以接受来自应用层的信息，又可以将处理过的信息反馈给应用层。（在具体实现上，网络层和链路层均具备本层所有功能，针对交换机和路由器，不同工作模式下，我们定义

了全局变量进行选择)

具体功能如下:

- 帧定位
- 差错检测
- 差错控制: 基于停止等待协议
- 简单的流量控制
- 交换机: 按目的转发、反向地址学习、未知广播、环状拓扑适应
- 路由器: 路由表、寻址与转发(静态路由)

c. 物理层

物理层主要负责比特流数据的传输。即将从网络层接收到的比特流数据传输给指定的对等层, 同时接收对等层传来的比特流数据。(所有数据我们均使用字节模式处理)

2.3 端口分配方案

我们采用课程组建议的方案:

端口号的五位数字有以下格式: 第一位 1 是常量, 第二位 1 表示设备号为 1, 第 3 位 3 表示第 3 层——应用层, 00 在该应用层的实体编号, 如果网元在这一层有多个实体, 比如本例中每个网元在物理层就有两个实体, 所以这两个实体的本地端口号分别位 11100 和 11101。

例如若有网元 1, 其具有一个应用层 11400、一个网络层 11300、一个链路层 11200 以及两个物理层 11100 和 11101。

三、交互

01111110	0	0101	0100	0011	10101011	11101010	01111110
帧头	ACK 标识	源地址	目的地址	序号	数据	CRC	帧尾

3.1 帧定位

3.1.1 定义

成帧技术是一种用来在一个比特流内分配或标记信道的技术。

3.1.2 方案

我们采用在帧头和帧尾各加 8 位标识的比特流 01111110。但是，仅仅如此的话会造成对帧头帧尾的误判，我们考虑如果在数据中遇到连续的 6 个 1，那么将最后一个 1 替换为 0，但是仔细考虑，这样仍然会造成误判，因此我们采用：如果在发送端数据中遇到连续 5 个 1，那么在最后一个 1 后面增添 1 个 0，例如：Data:11111100 -> 111110100。在接受端遇到 111110100 这种情况若判断出比特流为 9 位时，可去除连续 5 个 1 后的 0，得到：11111100。

3.1.2 函数实现

发送机制：

```
char* get_frame(char* s); //成帧函数
bool define_fiveone(char* p, int index); //确定是否存在连续的 5 个 1
```

接收机制：

```
char* locate_frame(char* s); //定位帧头帧尾，提取帧
char* de_frame(char* s); //提取 8 位比特流
```

3.1.3 测试

发送机制：

```
成功接收应用层数据：11110010
-----成帧-----
成帧成功：011111101111001001111110
```

接收机制：

```
成功从物理层接收数据：0101111100101011011111101111001010110011111101111100110000011
-----提取帧-----
成功提取帧：01111110111100101011011001111110
-----提取比特流-----
成功提取比特流：11111001
```

3.2 差错检测

我们采用循环冗余校验来检验误码。

3.2.1 定义

CRC 是一种根据网络数据包或计算机文件等数据产生简短固定位数校验码的一种信道编码技术，主要用来检测或校验数据传输或者保存后可能出现的错误。它是利用除法及余数的原理来作错误侦测的。

3.2.2 方案

- a. 编写函数计算 CRC 校验码，并添加至 8 位比特流后。
接收方通过重新计算 Q 来判断是否误码。若出现误码，则令 ERROR_FLAG=true (ERROR_FLAG 是初始化为 false 的误帧标志) (需要注意的是，这里我们并不是用 FCS 作为校验码，二而是 T/P 得到的 Q 作为校验码，这样，接收方可以重新计算 Q 来进行比对，以发现是否出错。
- b. 此外，我们为每个帧加入序号，可以通过序号差来判断是否有帧丢失，重复帧的情况发生。

3.2.3 函数实现

发送机制：

```
char* get_crc(char* s); //计算 CRC
char* add_crc(char* p, char* crc); //将 CRC 插入 Data 后
int de_serial_number(char* p); //将 10 进制帧序号转为 2 进制
char* add_serial_number(char* p, char* serial); //插入帧序号
```

接收机制：

```
char* de_crc(char* s); //提取 CRC
int two2ten(char* p); //将 2 进制帧序号转为 10 进制
int de_serial_number(char* p); //提取帧序号
```

3.2.4 测试

发送机制：

```
-----差错控制-----
计算得校验码：00110101
```

接收机制：

```
-----差错检测-----
成功提取校验码：10110110
重算校验码：10110110
```

```
-----提取序列号-----
成功提取序列号：4
```

3.3 差错控制

对于出现误码的帧，我们采用停止等待协议进行差错的控制。

3.3.1 定义

停止等待协议用于通信系统中，两个相连的设备相互发送信息时使用，以确保信息不因丢包或包乱序而丢失，是最简单的自动重传请求方法。

3.3.2 方案

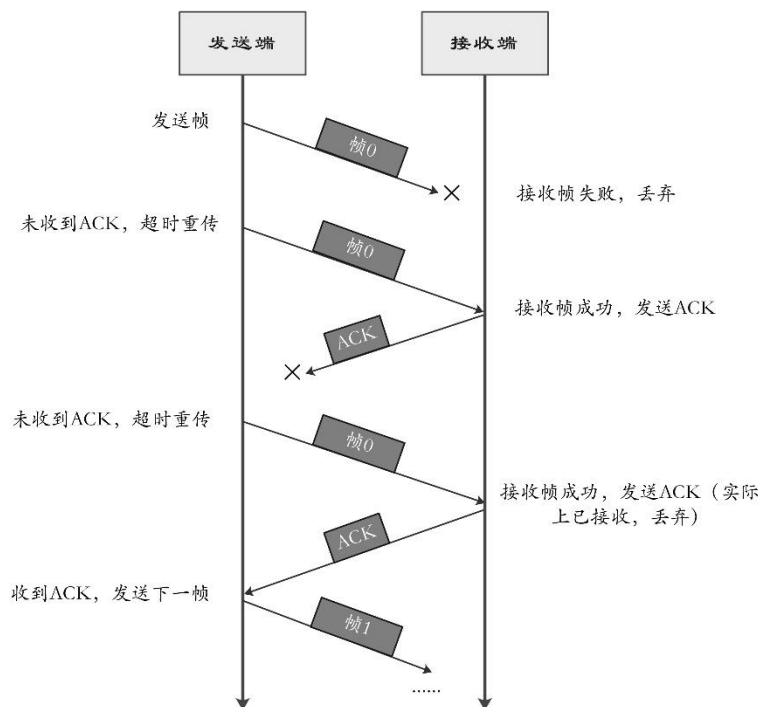
遵循以下原则：

- 收方对收到的帧进行差错检测以及帧序号差（与本地）的判断
- 收方只对成功接收的帧发送确认帧（ACK），其余一律丢弃
- 发送方每发一帧必须暂存此帧并停止等待确认帧，超时则重发

关于确认帧：

格式：将源、目的地址颠倒，ACK 标识置为 1，数据位为 11111111

在这里，考虑到反馈比特流可能会出现误码，我们规定，只要 8 位比特流内 1 的个数大于等于 4，则认为不需要重传，8 位比特流内 1 的个数小于 4，则需要重传。这样就巧妙的解决了反馈信息误码造成的判断错误，毕竟误码率 p_4 近似等于 0 了，几乎是不可能事件。



3.3.3 函数实现

发送机制：

```

char* add_dirty(char* p); // 加入 ACK 标识位
bool if_ack(char* s); // 判断是否为确认帧
bool get_dirty(char* p); // 获得重传标志
  
```

接收机制：

```
char* get_retrans_frame(char* p); //获得确认帧
```

3.3.4 测试

发送机制：

```
-----加序列号-----  
成功添加序列号：011111100000011010000000101001111110
```

```
-----反馈机制：差错控制-----  
等待连接...  
此帧发送成功!!!
```

```
发送数据：01100101  
接收反馈数据：11111111  
发送数据：01101100  
接收反馈数据：11111111  
发送数据：01101100  
reflect_retval:0  
失帧!! 重传：01101100
```

接收机制：

```
-----提取序列号-----  
成功提取序列号：0
```

```
解封：01101100  
发送确认帧：0111111010100010100001111101110000000001111110  
确认帧丢失!!
```

3.4 流量控制

起初并没有发现有什么地方需要进行流量控制的，但当发送多组数据时，接收方并不能全部接收到，每隔一组数据都会漏接一个，这时候我意识到发送方向物理层灌输数据过快，一些数据被丢弃，于是我在网络层代码中，每次向物理层发送完数据后加 `Sleep(10)`，实现了简单的流量控制。

3.5 函数封装

为了便于后续阶段的进行，我们将交互阶段大部分功能封装在一个函数中，并且注释了不必要的打印信息。

发送机制：

```
char* encapsulate_frame(char* revData); //封装函数，获得最终向物理层发送的帧
```

接收机制：

```
char* decapsulate_frame(char* recbits); //解封函数，获得数据，并进行差错检测
```

测试：

```
封装：011111100010101000100011011111000001101111110
```

```
解封：01101111
```

四、共享

4.1 按目的转发

4.1.1 定义

二层交换机从某个端口收到帧后，通过查看到此帧的目的 MAC 地址，然后凭借 MAC 地址表向相应的端口转发。

4.1.2 方案

在交互设计基础上为每个帧添加源地址和目的地址

为网络中每个设备编址（根据设备号编址，例如设备 6 其 MAC 地址为 6），为交换机编写代码实现 MAC 地址表（利用结构数组），每接收一帧，查看其目的 MAC，先判断是否为自身，若是则解封处理递交上层，若不是则查表获得端口号转发，若查无此地址则广播到其他端口（直通转发）。

4.1.3 代码实现

```
struct Switch_Table //建立 MAC 表
{
    int addr;
    int port;
};
```

```
Switch_Table* st = (Switch_Table*)malloc(15 *
sizeof(Switch_Table)); //实例化

char* add_code_source_node(char* p); //插入目的地址
int get_decode_source_node(char* s); //获得目的地址
void print_switch_table(); //打印 MAC 表
int find_st_port(int addr); //根据目的地址查找端口号
```

4.1.4 测试

发方将源地址、目的地址插入帧中发送

```
接收数据: 01101100
01101100计算得校验码: 10000000
封装: 011111100010100100010011011001000000001111110
```

交换机 MAC 表

```
-----MAC Table-----
addr: 5   port: 0
addr: 2   port: 1
-----
```

查看目的地址，按目的查表转发

```
目的设备号: 5
从接口1接收数据: 1011111010010010100001111101110000000011111101
将接口1数据 48 比特转发给接口0
```

4.2 反向地址学习

4.2.1 定义

在交换机初始化的时候，其 MAC 地址表是没有任何 MAC 地址和端口的映射条目的，通过查看帧源地址进行地址的学习，以此得到 MAC 表。

4.2.2 方案

在接收帧后，获取其源地址，查表，若表中无对应关系，则进行该地址的学习。

特别说明的是，反向地址学习一般流程是要先进行 ARP 广播对其所有端口所连设备进行反向地址学习的，我们对此问题进行讨论认为，在规模较大，通信复杂的情况下，确实应该先进行广播学习，但若是规模小的通信系统，通信的设备不多的情况下，如若仍然先学习所有设备地址，则会造成信道资源的浪费，在本项目最终成果中，我们让交换机只对所接收帧内的源地址进行学习（按需学习），对于没有参与通信的设备地址不进行学习，我们认为这是很节约信道资源的方案。

4.2.3 函数实现

```
char* add_code_node(char* p); //插入源地址
int get_decode_node(char* s); //获得源地址
void reverse_addr_learn(int addr, int port); //反向地址学习
```

4.2.4 测试

初始 MAC 表为空

```
-----MAC Table-----
|
|
|
```

反向地址学习得到新的 MAC 表

```
更新MAC表.....
-----MAC Table-----
addr: 5    port: 0
addr: 3    port: 1
-----
```

4.3 未知广播

4.3.1 定义

交换机对未知目的地址的帧会广播到其他所有端口。

问题：在由交换机组成的环状拓扑中，对于未知广播会引起广播风暴（广播数据充斥网络无法处理，并占用大量网络带宽，导致正常业务不能运行，甚至彻底瘫痪）。

4.3.2 方案

对于未知广播的转发处理代码实现较简单，不再赘述，我们主要对其产生的广播风暴问题进行探讨：为交换机的每个端口设置使能标志，当出现广播风暴是，交换机会进行判决，使得其中一个端口逻辑上断开（即简单的生成树协议）。

4.3.3 代码实现

```
bool disabled_0 = false; //定义使能标志：true 为断开，false 为正常连接
if (D_n > Max_Device && D_n != 15) //根据具体代码的端口断开条件
{
    .....
    disabled_0 = true; // 接口 0 断开
    printf("\n 未知广播:%s\n 设备 %d 物理层接口 0 断
```

```
开! \n", my_initial_frame, Source_Device_number);  
    continue;  
}  
//接口失效的具体实现方式  
    if (disabled_0)  
    {  
        continue;  
    }
```

4.3.4 测试

1 号网元向设备 5 发了信息，但在三个网元成环的测试案例中并不存在 5 号设备

```
当前设备号为 1，请输入目的设备号: 5  
Please input your words:anybody ?
```

与 1 号网元所连交换机为了避免广播风暴，做出断开接口的决策

```
未知广播:011111100000101010000011000010000100101111110  
设备 2 物理层接口 0 断开!
```

五、路由

*以下内容均基于最终成果《混合组网》编写

5.1 路由表

5.1.1 定义

路由表或称路由择域信息库，是一个存储在路由器或者联网计算机中的电子表格（文件）或类数据库。路由表存储着指向特定网络地址的路径（在有些情况下，还记录有路径的路由度量值）。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。

5.1.2 方案

为每个路由器配置路由表，记录目的地址、下一跳、端口、距离四个信息项。
例如：

Router 2 - Routing Table					
Destination	1	3	4	5	6
Gataway	1	3	3	1	1
Port	2	0	0	2	2
Metric	1	1	3	2	2

5.1.3 代码实现

```
struct Router_Table//建立路由表
{
    int destination;//目的地址
    int gataway;//下一跳
    int port;//端口
    int metric;//距离
};
Router_Table* rt = (Router_Table*)malloc(15 *
sizeof(Router_Table));//实例化
void print_router_table();//打印路由表
```

5.1.4 测试

图为《混合组网》2号路由器路由表

```
-----Routing Table-----
Destination: 1  Gataway: 1  Port: 2  Metric:1
Destination: 3  Gataway: 3  Port: 0  Metric:1
Destination: 4  Gataway: 3  Port: 0  Metric:3
Destination: 5  Gataway: 1  Port: 2  Metric:2
Destination: 6  Gataway: 1  Port: 2  Metric:2
-----
```

5.2 基于静态路由的寻址与转发

5.2.1 定义

一种路由的方式，路由项由手动配置，而非动态决定。与动态路由不同，静态路由是固定的，不会改变，即使网络状况已经改变或是重新被组态。一般来说，静态路由是由网络管理员逐项加入路由表。

5.2.2 方案

为每个路由器预先配置好路由表，每当收到帧是，查看其目的地址，查表，获得对应端口并转发。

值得一提的是，由于最终的混合组网拓扑结构过于简单，直接使用无法体现路由器对最优路径的选择，因此，我们为每条路径都规定了距离，如下：例如从2号到4号距离为4

Device	5<->1	6<->1	1<->2	2<->3	3<->4	2<->4
Metric	1	1	1	1	2	4

为了更真实的模拟，我们根据距离使用 Sleep() 函数，在每个端口发送信息前暂停 $n \times 10\text{ms}$ ，其中 n 为对应路径距离，例如从 2 号到 4 号将暂停 $4 \times 10\text{ms}$ 。

5.2.3 函数实现

```
int find_rt_port(int destination); //查表，取最优路径，获得端口号
```

5.2.4 测试

以下测试为在《混合组网》中实现，从 5 号主机发送数据到 4 号路由器经过 2 号路由器

```
目的设备号: 4
从接口2接收数据: 110111111000101010000000110100000001010011111101111
将接口2数据 51 比特转发给接口0
```

经过 3 号路由器

```
目的设备号: 4
从接口0接收数据: 111011111100010101000000011010000000101001111110111111
将接口0数据 55 比特转发给接口1
```

4 号路由器进行地址比对，接收此帧

```
目的设备号: 4
从接口 0 收到数据: 1111101111100010101000000011010000000101001111110111111
-----提取帧-----
成功提取帧: 0111111000101010000000011010000000101001111110
```

六、应用

6.1 能传信息

6.1.1 随机比特流

a. 方案

编写函数实现比特流的随机生成，便于项目进行中的调测。

b. 函数实现

```
char* get_eight_bites(); //随机生成 8 位比特流
```

c. 测试

发送机制

```
1 发送数据: 10100001
```

接收机制

```
接收数据: 00100100
```

6.1.2 字符

a. 方案

采用 ASCII 码进行对英文字母，标点，常用符号的编码及其解码

b. 函数实现

```
char* ascii_code(char ch); //将字符编码为 8 位比特流
char ascii_decode(char* p); //将 8 位比特流解码为字符
char* ten2two(int s); //ASCII 编码过程中用到的 10 进制转 2 进制函数
int two2ten(char* p); // ASCII 解码过程中用到的 2 进制转 10 进制函数
```

c. 测试

发送机制

```
当前设备号为 5，请输入目的设备号: 4
Please input your words:hello
向下层传递参数: 0100
发送数据: 01101000
```

接收机制

```
接收数据: 01101111
译码: hello
共发送 0 位, 0 次, 发生 0 次错误; 共接收
```

6.1.3 图片

a. 方案

采用 base64 对图片进行编解码，发方先将图片编码为字符串，再将字符串编码为比特流，收方先将比特流译码为字符串，再用 base64 解码为图片并保存。

b. 函数实现

```
const char* base64char =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+
/"; //base64 编解码用到的字符
char* code_image(char* file_path); //图片编码
void decode_image(char* imageBase64, unsigned int imageSize); //
图片解码
```

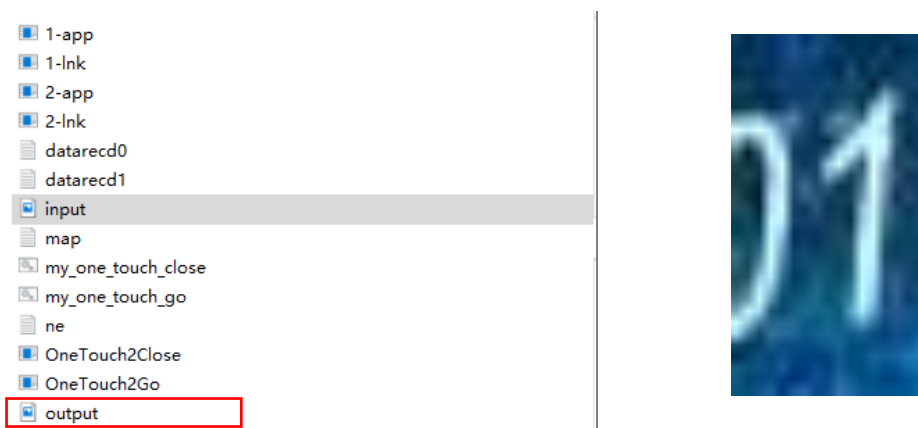
```
char* base64_encode(const unsigned char* bindata, char* base64,
int binlength); //图片编码过程中使用的 base64 编码函数
int base64_decode(const char* base64, unsigned char*
bindata) ; //图片解码过程中使用的 base64 解码函数
unsigned int get_imagesize(char* file_path); //获取图片大小（基于
base64）
```

c. 测试

发方检索用户输入的图片文件路径: "input.jpg", 进行编码发送

[illegible]

收方将收到的图片命名为"output.jpg"进行保存



*这种编码方式效率还是比较低的，本例中图片分辨率 17×32 ，共发送 27264 帧，时长约 270s.

*为了便于使用，用户可以通过修改 ne 文件中 workMode 参数进行不同类型信息的发送，具体分配如下：

11: 自动发比特流 (20 组)

10: 自动发字符串 (“hello”)

1:手动发字符串

0: 发图片

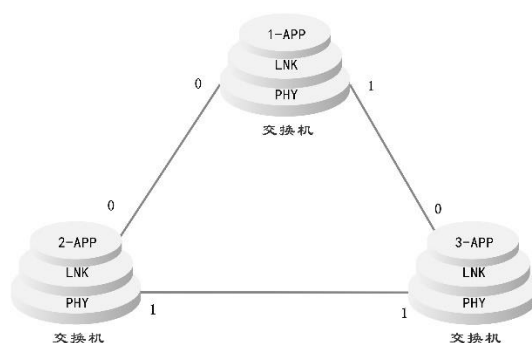
6.2 多种拓扑结构适应性

6.2.1 环状

a. 概述

本例为仿真三个相连成环状的网元，各网元均为二层交换机。本例中使用了共享中所述简单的生成树协议，避免了广播风暴的发生。

b. 拓扑图

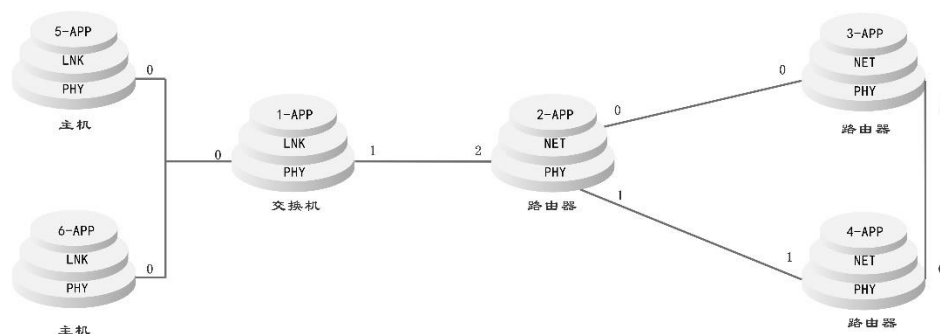


6.2.2 混合组网

a. 概述

本例为综合整个项目内容搭建的具有 6 个网元的混合组网，其中包括了 2 个主机，1 个交换机，3 个路由器，并且每个网元都可是一个主机。具体细节见下一章节。

b. 拓扑图



七、混合组网

7.1 概述

本例为仿真六个网元，网元 1 为交换机，网元 5、6 是主机，与交换机 1 的一个接口在共享信道上，交换机 1 的另一个端口与网元 2 连接，网元 2、3、4 为路由器连接成环状，整个拓扑需要交换与路由器的配合，形成复杂网络结构。

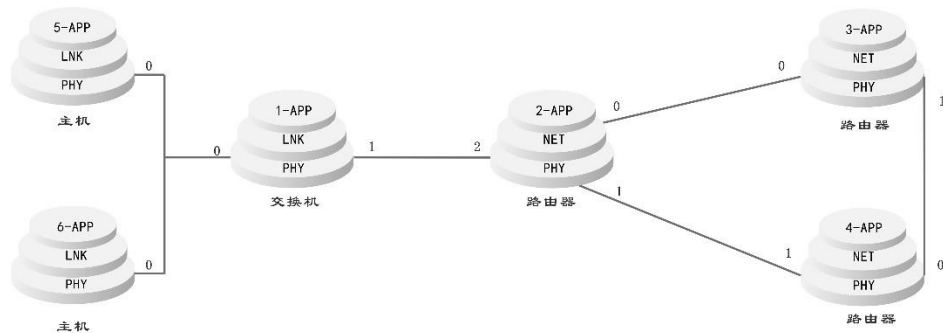
7.2 层次化设计

相比较于项目所要求层次设计，我们作了相应的简化，即将链路层与网络层功能集于一体，但在端口分配时仍区分开来。如下图【“***”表示此层不存在】

最小网元设计

设备	物理层			链路层		网络层		应用层	
1	模拟软件0	本地端口号	11100	端口号	11200	端口号	***	端口号	11400
		对方端口号	15100/16100						
	模拟软件1	本地端口号	11101						
		对方端口号	12102						
2	模拟软件0	本地端口号	12100	端口号	***	端口号	12300	端口号	12400
		对方端口号	13100						
	模拟软件1	本地端口号	12101						
		对方端口号	14101						
	模拟软件2	本地端口号	12102						
		对方端口号	11101						
3	模拟软件0	本地端口号	13100	端口号	***	端口号	13300	端口号	13400
		对方端口号	12100						
	模拟软件1	本地端口号	13101						
		对方端口号	14100						
		对方端口号	14100						
4	模拟软件0	本地端口号	14100	端口号	***	端口号	14300	端口号	14400
		对方端口号	13101						
	模拟软件1	本地端口号	14101						
		对方端口号	12101						
		对方端口号	12101						
5	模拟软件0	本地端口号	15100	端口号	15200	端口号	***	端口号	15400
		对方端口号	11100						
6	模拟软件0	本地端口号	16100	端口号	16200	端口号	***	端口号	16400
		对方端口号	11100						

7.3 拓扑图



7.4 路由表分配

Routing Table						
Router 2	Destination	1	3	4	5	6
	Gataway	1	3	3	1	1
	Port	2	0	0	2	2
	Metric	1	1	3	2	2
Router 3	Destination	1	2	4	5	6
	Gataway	2	2	4	2	2
	Port	0	0	1	0	0
	Metric	2	1	2	3	3
Router 4	Destination	1	2	3	5	6
	Gataway	3	3	3	3	3
	Port	0	0	0	0	0
	Metric	4	3	2	5	6

*由于最终的混合组网拓扑结构过于简单，直接使用无法体现路由器对最优路径的选择，因此，我们为每条路径都规定了距离，如下：例如从 Router 2 到 Router 4 距离为 4。

Device	5<->1	6<->1	1<->2	2<->3	3<->4	2<->4
Metric	1	1	1	1	2	4

7.5 测试

本例为实现 5 号主机与 4 号路由器之间通信
依照上述内容，帧传递路径为 5->1->2->3->4

5 号主机将用户输入信息” Hello,Router 4 !” 编码发送；

```
-----收发窗口-----
当前设备号为 5，请输入目的设备号：4
Please input your words:Hello,Router 4 !
向下层传递参数：0100
发送数据：01001000
接收反馈数据：11111111
```

1 号交换机收到未知目的地址的帧，转发至其他所有端口（端口 1），并进行反向地址学习；

```
-----MAC Table-----
-----收发窗口-----
更新MAC表.....
-----MAC Table-----
addr: 5   port: 0
-----
目的设备号：4
从接口0接收数据：101111110001010100000000100100010001010011111101
将接口0数据 47 比特转发给接口1
```

2 号路由器收到帧，解析其目的地址并查表转发；

```
-----Routing Table-----
Destination: 1  Gataway: 1  Port: 2  Metric:1
Destination: 3  Gataway: 3  Port: 0  Metric:1
Destination: 4  Gataway: 3  Port: 0  Metric:3
Destination: 5  Gataway: 1  Port: 2  Metric:2
Destination: 6  Gataway: 1  Port: 2  Metric:2
-----
-----收发窗口-----
目的设备号：4
从接口2接收数据：111011111100010101000000001001000100010100111111011
将接口2数据 50 比特转发给接口0
```

3 号路由器收到帧，解析其目的地址并查表转发；

```
-----Routing Table-----
Destination: 1  Gataway: 2  Port: 0  Metric:2
Destination: 2  Gataway: 2  Port: 0  Metric:1
Destination: 4  Gataway: 4  Port: 1  Metric:2
Destination: 5  Gataway: 2  Port: 0  Metric:3
Destination: 6  Gataway: 2  Port: 0  Metric:3
-----
```

收发窗口

目的设备号: 4

从接口0接收数据: 11110111111000101010000000100100010001010011111011111

将接口0数据 54 比特转发给接口1

4 号路由器（主机）收到帧，进行目的地址比对，传递给应用层，应用层进行译码

共发送 0 位,0 次,发生 0 次错误; 共接收 120 位,15 次

接收数据: 00100001

译码: Hello,Router 4 !

共发送 0 位,0 次,发生 0 次错误; 共接收 128 位,16 次

八、附件说明

附件一：源码

该文件为《混合组网》所有源码，实际上大同小异，每个网元都可以是主机、交换机或路由器，只需查看 2-app.cpp 与 2-net.cpp 即可。

附件二：项目日志

该文件较详细的记录了该项目的进展情况，以及每次调测记录，问题记录，版本记录等。具有很强的实用性与纪念意义。

附件三：交换机成环 Demo

该文件用于测试共享阶段所有功能，尤其是交换机对广播风暴的解决。由于文件较多，为了方便调测，我们结合课程组所给的一键启动/关闭文件编写了 my_one_touch_go.bat 与 my_one_touch_close.bat 批处理文件（提前关闭防火墙）。

附件四：图片传输 Demo

该文件用于测试应用阶段所述的传输图片功能，为了测试方便，我们将误码设为 0。同样，我们编写了一键启动/关闭文件便于调测。my_one_touch_go.bat 与 my_one_touch_close.bat

附件五：混合组网 Demo

该文件用于测试《混合组网》，同样，我们编写了一键启动/关闭文件便于调测。my_one_touch_go.bat 与 my_one_touch_close.bat