# 虚拟机网络 ip和路由设置：

## 虚拟网络：



1. 子网掩码一律为 255.255.255.0

2. 其中 R1 H1 同在以太网 VMnet2 中

3. R1 R2 在 VMnet3 中

4. R2 H2 在 VMnet4 中

## IP 设置

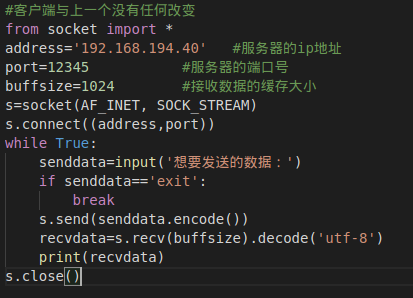
1. H1
   1. 在 VMnet2 中的地址为 192.168.21.30
2. R1
   1. 在VMnet2 中的地址为 192.168.21.10
   2. 在VMnet3 的地址为 192.168.136.10
3. R2
   1. 在VMnet3 的地址为 192.168.136.20
   2. 在VMnet4 的地址为 192.168.194.20
4. H2
   1. 在 VMnet4 的地址为 192.168.194.40

## 路由设置

同实验二的网络拓扑结构

# TCP 客户端和服务器设计和源程序

## 客户端



## 服务器

## 设计方案

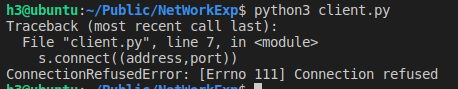
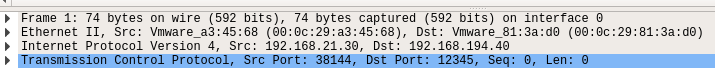
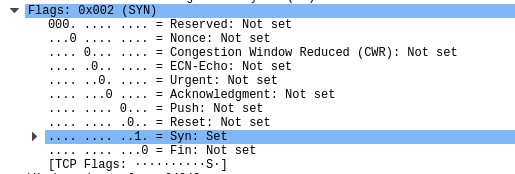
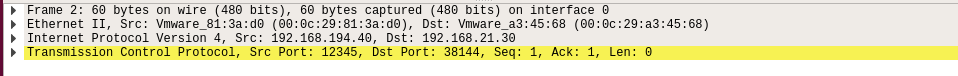
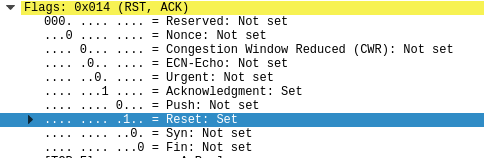
使用的 是python语言，通过python 提供的 socket库来进行tcp通信，此外服务器端还使用了python 的threading 包 来处理多客户端的情况，针对每一个来自不同ip 的客户端单独使用一个线程去进行响应操作。

1. Tcp 的使用是在 s = socket(AF\_INET,SOCK\_STREAM) 中识别的，SOCK\_STREAM 是流式传输，将默认使用 tcp协议

# TCP交互模拟

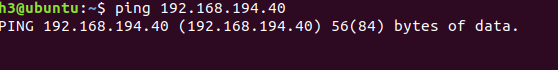
## 未启动服务器端

### H1（客户端）

1. 应用程序执行情况：  
   
   1. S.connect 方法 报错 —— 连接遭到拒绝
2. Tcp请求消息  
     
   
   1. Reset 位为0
   2. Syn 位位1 表示此帧位请求消息
3. Tcp响应消息  
     
   
   1. Rst 位即 reset为1，表示发生错误，拒绝打开一个连接
   2. Syn位为0

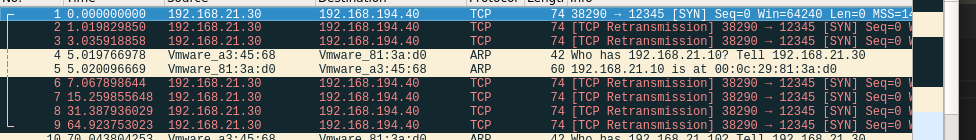
## 启动服务器打开R1防火墙

设置防火墙（主机名为r2但实际是r1）：  


设置防火墙之后 进行H1 ping H2的操作：  


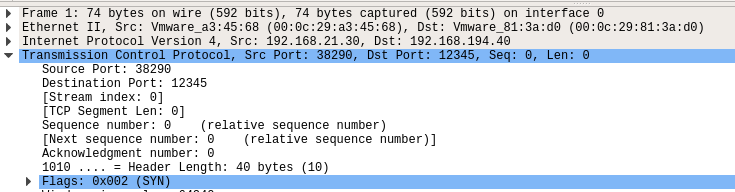
将会长时间得不到响应

### H1客户端



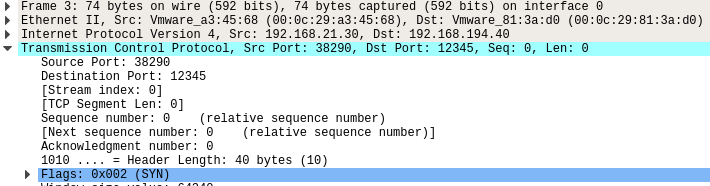
如上第一格tcp包为请求连接包，而由于包丢失得不到响应消息，所以之后 进行重发操作即如上图所示的黑色部分

#### 请求消息

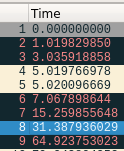


和之前没有区别

#### 重发的请求消息



内容和之前几乎一致，只有 timestamps（时间戳） identification（id） 和 checksum（校验和） 三个位置不同

1. Syn比特位： 都为1 表示建立连接请求
2. Ack比特位：都为 0
3. Seq 子段：皆为 0
4. 重发间隔时间：  
   通过wireshark 的time一列可以发现，重发的时间差基本保持 2的n-1次方 第一次重发在 2^0 =1 秒后 第二次重发在第一次重发后的 2^1 = 2秒后(即3秒时重发) ……

### H2 服务器端

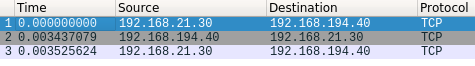
没有接收到任何的消息

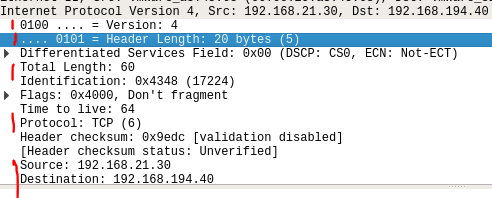
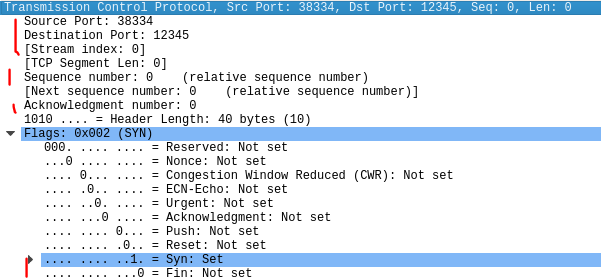
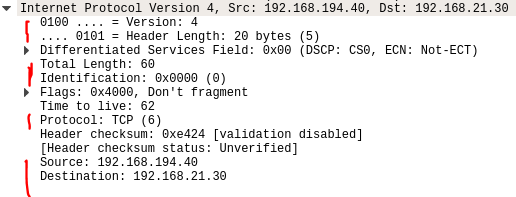
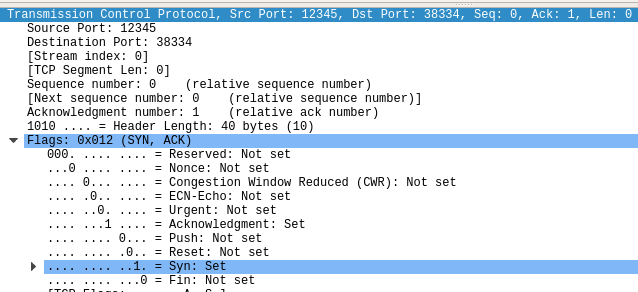
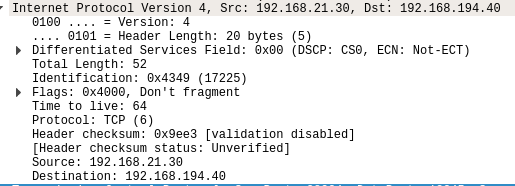
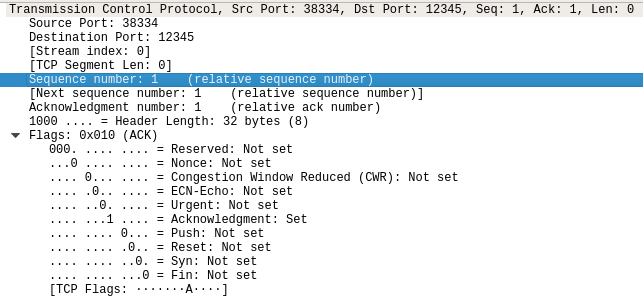
## 正常连接 三次握手

代码运行：

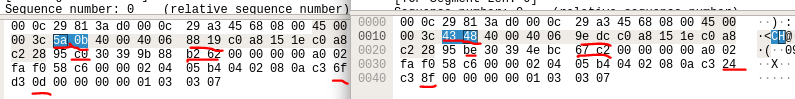
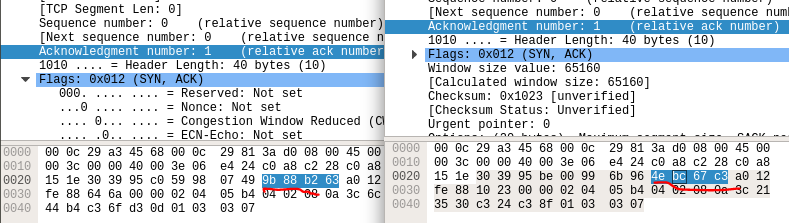


Wireshark 抓包：



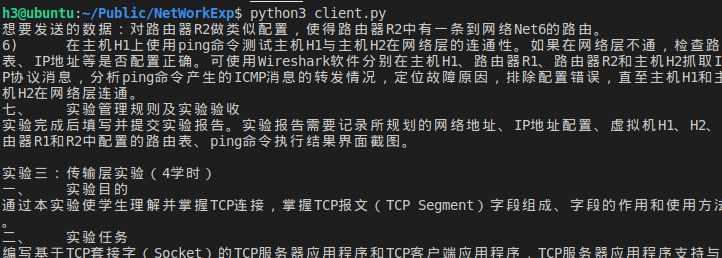
1. 连接请求包 （客户端到服务端）：
   1. IP 分组  
        
      从上到下解释如下
      1. 协议字段为 0100 表示 IPV4
      2. IHL 字段为0101 表示 5 即 包头长度为20bytes
      3. Total length 字段 为 60 表示整个 ip分组长度为 60byte
      4. Protocol 字段为6 表示使用的是TCP协议
      5. 源地址和目标地址：  
         
         1. 其中 源地址即为 H1客户端的ip地址 192.168.21.30
         2. 目标地址即为服务器端的地址：192.168.194.40
   2. TCP报文：  
      
      1. 源端口为 38334 ，目标端口为 12345
      2. 序列号字段为 0
      3. Ack 字段为 0
      4. Syn = 1 表示连接建立请求，fin = 0表示消息还未传输完
2. 服务器连接建立响应 （服务端到客户端）
   1. IP分组  
      
      1. 可以看到 源地址为 192.168.194.40即 身为服务器端的 H2 目标地址是客户端的 192.168.21.30
   2. Tcp 报文  
      
      1. 源端口为 12345 目标端口为38334
      2. Seq 为0 ，ack字段 = 1，表示希望下一次受到的字段的序列号为1
      3. syn = 1 ack=1 表示对建立连接的确定，syn = 1 默认这是一个 连接恳求或是毗邻接管报文，而 ack =1 则识别了其为 连接恳求报文，并且只有 ack位=1 确认字段才有效
3. 客户端的连接确认报文（客户端到服务端）
   1. Ip分组：  
      
      1. 信息基本和理解请求一致，除了 totallength 减低到了 52
   2. Tcp报文内容  
      
      1. 序列号字段变成了1 这和 服务器之前发送的帧的 ack字段相同。Ack 字段=1 同样表示希望接收到来自服务器的 seq =1 的帧
      2. Syn = 0表示不再是理解请求的帧。Ack 位 = 1 确认号(ack)字段有意义

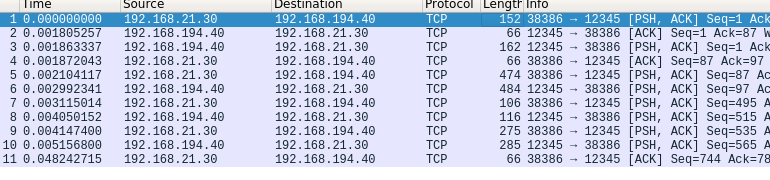
## 另一个客户端

1. 第一次握手（客户端到服务端）：
   1. 不同之处：  
        
      依次为：
      1. Id，因为这是唯一标识帧的 字段
      2. Header 校验和，校验和是根据所有的位来计算的，故不会相同
      3. 源端口，  
         
         1. 两个的端口分别位 38336 和 38334，这是因为38336 以及被前一个客户端占用，故需要使用另外的端口
      4. Sequence number，虽然seq 在 wireshark 中显示的是 0 ，但是0只是相对位，而真实的 seq 的值则为 二进制中描述的那样，不尽相同
      5. Option 下的 timestamp value，因为发送的时间不同 故 timestamp 时间戳的值不同
2. 第二次
   1. 不同之处
      1. 除了之前的不同外还有 ack字段也不同：  
           
         这是因为第一次握手中 ack 字段(ACK 位 为0)没有用，故全为0而在2，3握手的时候ack 有用。
3. 第三次 的不同之处同第二次

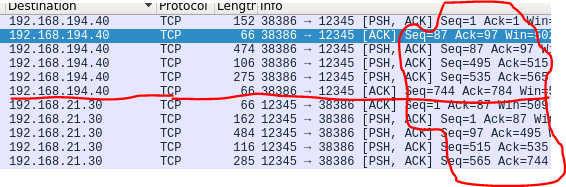
## TCP建立后数据量大于链路数据包最大长度

客户端向服务器发送实验指导书内容:



发送的包：  


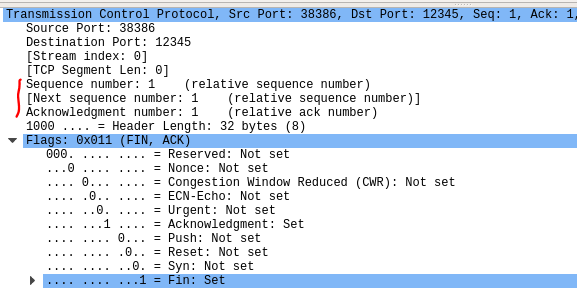
可以发现发送了很多tcp的包，所以可以判断 进行了分段操作



对应的 seq 和ack 字段的值如上， 是对ip 进行排序后的结果，上方为服务器ip下方为客户端ip，可以看到 服务器端和客户端发送的包的 seq 和 ack 分别和对端的ack 和seq 对应。

## 关闭TCP客户端和tcp服务端



1. 报文1：  
   
   1. ACK = 1 Fin =1，fin=1 标识请求断开连接
   2. Seq ack 字段都为1（相对位置）
2. 报文2（服务端到客户端）
   1. ACK = 1 Fin =1，fin=1 标识连接断开允许
   2. Seq=1 ack=2 标识希望接收客户端的响应信息
3. 报文3（客户端到服务端）
   1. ACK =1 FIN =0
   2. Seq ，ack =2