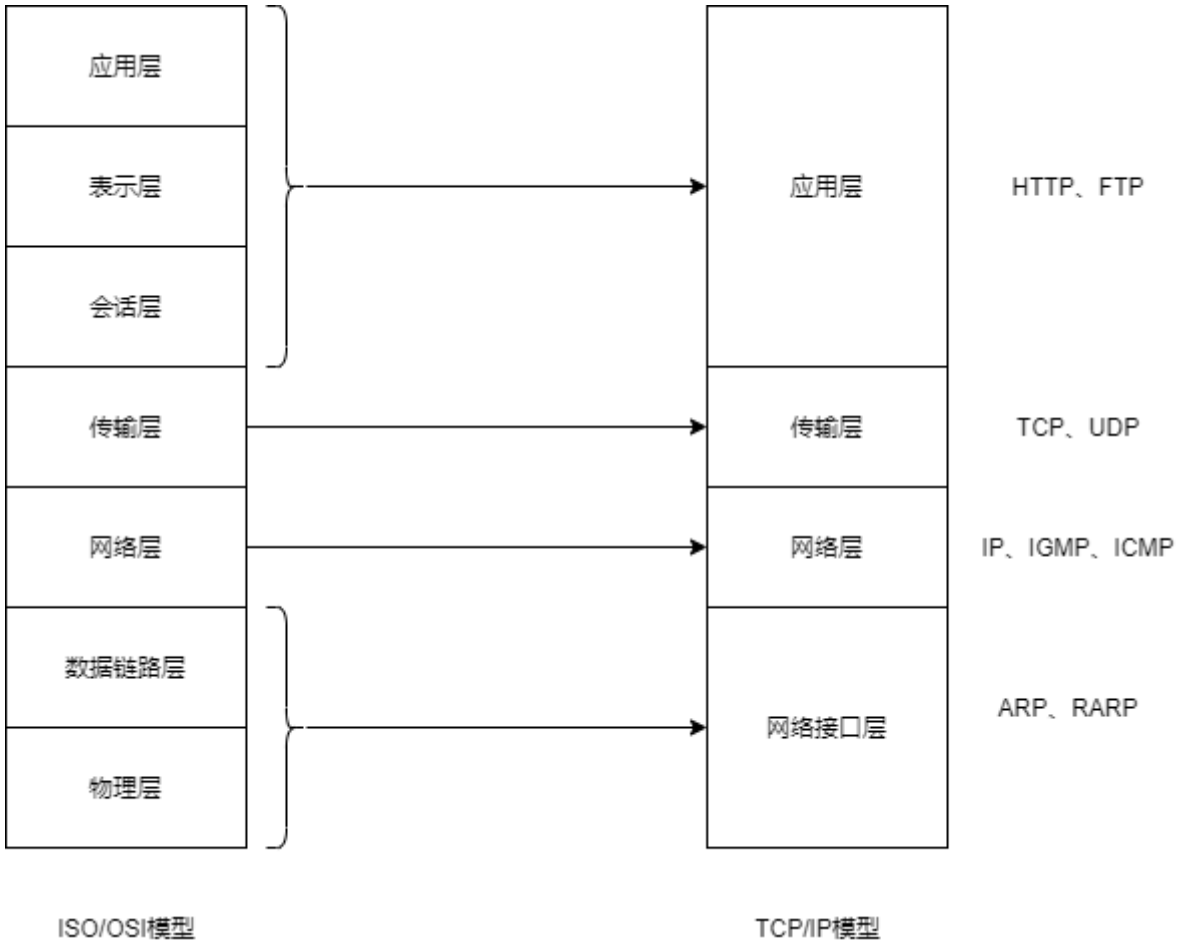
9 网络和网络编程

9.1 网络协议和网络模型

如果需要书写跨越网络的代码，首先需要对网络通信的基本概念有一些认识。在使用网络通信时，可以认为是由一端程序发起了请求而由另一端回复了一个响应。发起请求的那一端称作**客户端**，响应的那一端称作**服务端**。在不同的网络应用，客户端和服务端的组织结构会有一定的变化，有些应用可能会有固定的一个服务端来响应大量的客户端请求（绝大多数网络应用就是如此设计的），有些的应用可以让程序既充当客户端也充当服务端（比如P2P下载软件）。

两个不同的人进行沟通，首先需要保证彼此之间理解对方的语言，网络通信的也是如此，**网络协议规定了通信两端数据交互的规则**。由于现实世界的网络通信纷繁复杂，要关注的侧面更多，比如硬件如何设计、数据如何选择传播方向、如何提供可靠性等等，网络设计不得不将其分解多个层次，每个层次都处理独立的任务。层次之间存在着上下层关系，上层依赖于下层，它需要调用下层提供的接口才能工作。

在不同的规范体系中，层次结构会有所不同，其中OSI网络模型有七层模型：分别是应用层、表示层、会话层、传输层、网络层、数据链路层和物理层，但是目前广泛使用的是**TCP/IP协议族**网络模型，它是**四层模型**：从上往下依次是应用层、传输层、网络层和网络接口层。不过7层模型影响也是相当深远的，即便在4层模型的体系中，许多应用层协议在设计的时候会考虑提供会话（比如建立和断开通信功能）和表示的功能（比如多语言交互功能）；此外，我们也经常将网络接口层分解成物理层(L1)和数据链路层(L2)。



对于用户而言，可以认为每一台主机上的每一层都是一个单独的进程。当数据从一端传输到另一端时，从逻辑上，它需要传输到目标机器上对等的层级，所以网络协议是针对**同一层**之间的通信生效的，但是单次数据的传送实际上却只能在发生上下层之间或者是网络线缆和网络接口层之间，上层的数据需要调用下层提供的接口将数据转移到下层，然后通过网络线缆转移到目标机器上，再从底层逐步向上传递，直到访问到同一层为止。

在网络编程的学习过程中，最重要的参考资源是RFC文档。RFC文档是一系列草案的集合，所有的互联网协议和其他相关的内容都位于其中。访问rpc-editor网站(www.rfc-editor.org)可以比较方便地查阅所有的RFC文档。

9.2 TCP/IP协议族概览

9.2.1 四层模型的各层实体

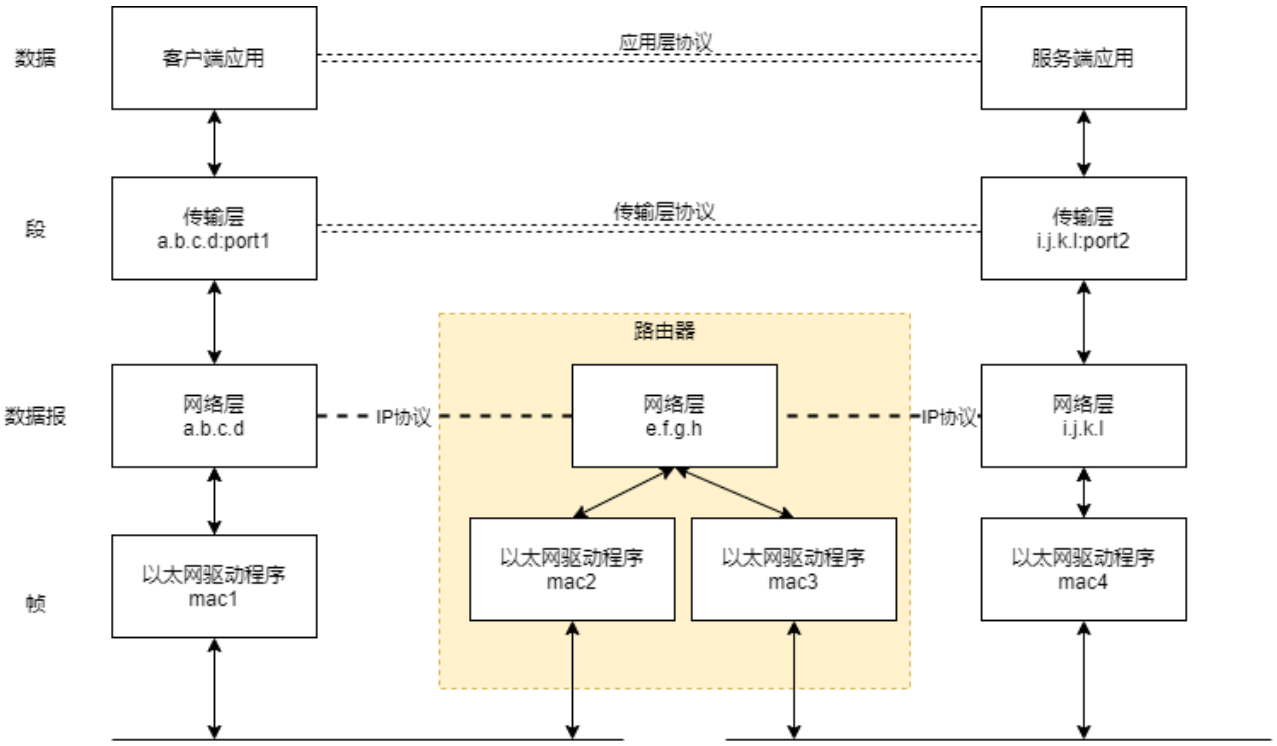
在四层模型当中，完成每一层功能的实体如下：

- 网络接口层之物理层：通常指实际传输的物理设备，比如线缆、调制解调器等设备，数据在这些设备中以比特流的形式就行传播。

- 网络接口层之数据链路层：通常是指网卡的驱动程序。该层将比特流转换成**数据帧**(frame)，**MAC地址**是用来描述不同的数据链路层设备的地址。
- 网络层：通常指不同的主机。网络层负责将数据帧封装成**IP数据报**(packet/datagram)。使用IP地址来描述主机在网络层的位置。网络层之间采用**逐机跳转**的通信方式。
- 传输层：负责将IP数据报封装成TCP**段**（segment）或UDP报文。该层可以为两台主机的应用程序提供**端到端**的通信。传输层只关心通信的起始端和目的端，不关心数据的中转过程。
- 应用层：负责处理应用程序的逻辑。

下面的图片简单说明了客户端通过路由器与服务端通信过程的数据流向。可以发现，协议是在同等的层次上生效的。每个层次能够处理的数据单位称作**协议数据单元**（PDU），不同层次之间的协议数据单元也是不同的，数据在上层下层之间传递时，需要使用一定的手段转换协议数据单元。

在网络中，除了有能完整处理4层协议的主机以外，还有一些设备只支持部分协议，比如路由器就是只能支持最高到网络层的协议处理的设备。使用路由器可以协助转发网络层数据报到更远的网络位置中。而交换机设备只能工作在网络接口层，它能让数据帧传播到更远的链路上。

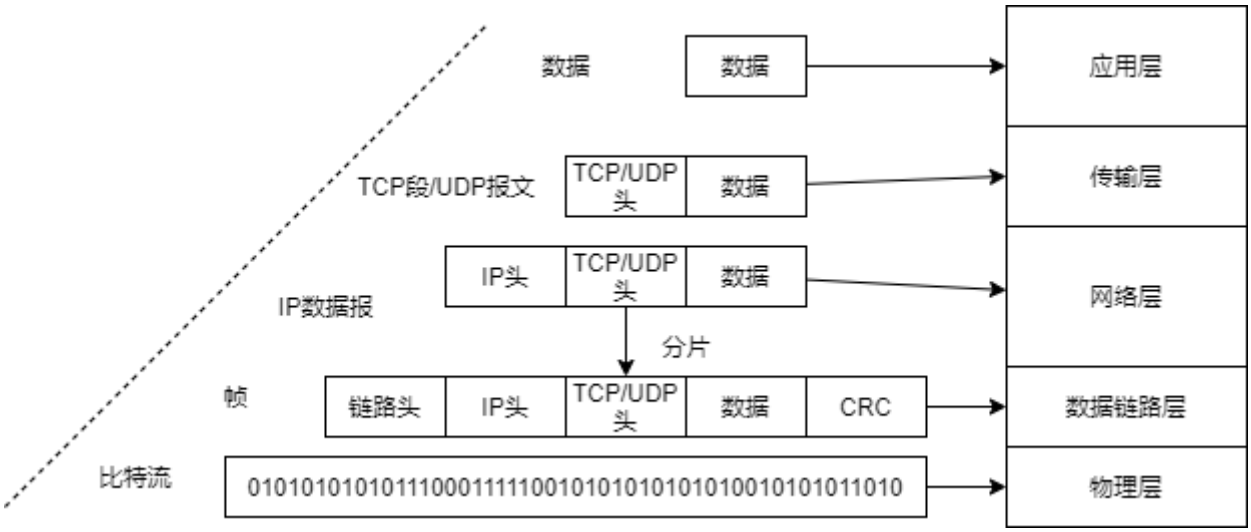


9.2.2 协议数据单元的转换

当数据从上层转移到下层时，下层的数据单元会把**上层数据单元的所有内容包含进来**，作为下层协议数据单元的有效载荷，除此还需要在头部或者是尾部增加下层的控制信息。其中TCP段/UDP报文会在数据中增加端口等信息作为头部。IP数据报会在TCP段/UDP报文的内容头部的添加IP地址等控制信息。

数据链路层帧的处理会相对复杂一点。在数据链路层存在一个**MTU**参数。它的数值决定了**数据帧有效载荷的最大长度**（单位是字节），长度超过MTU的数据帧将无法传递到网络层。常用的MTU值有1500、576等，而IP数据报总长度上限为65535个字节。所以当数据在网络层和数据链路层之间转换时，网络层应用需要将IP数据报按照下层的MTU大小进行分片，每个分片都是一个IP数据报，其大小足够小从而能够放入帧中。在到达目的主机以后，网络层应用还需要将分片进行重组。除此以外，在成帧的时候，除了需要增加一个链路头，还需要在末尾补充CRC校验和，检测传输的过程中是否出现硬件错误导致的比特跳变。

在四层模型中，协议数据单元的转换可以参考下图。



9.2.3 常见协议以及分层

一些常见的网络协议及其所属的分层如下：

协议名	层	协议功能	占用端口号
HTTP 超文本传输协议	应用层	传输Web网站网页和其他资源	80
ftp 文件传输协议	应用层	文件上传和下载	20/21
telnet 远程连接协议	应用层	远程登录（无加密）	23

协议名	层	协议功能	占用端口号
ssh 安全外壳协议	应用层	远程登录（加密）	22
SMTP 简单邮件传输协议	应用层	电子邮件（邮件在服务器之间提交和传送）	25
POP3 邮局协议版本3	应用层	电子邮件（邮件最终交付协议）	110
TCP	传输层	可靠的面向连接的传输层协议	
UDP	传输层	不可靠的基于数据报的传输层协议	
IP	网络层	网际互联协议	
ICMP	网络层	网络控制消息协议	
IGMP	网络层	网络组管理协议	
ARP	数据链路层	地址转换协议	
RARP	数据链路层	反向地址转换协议	

虽然说我们可以将每一层都看成是一个独立的进程，但是在实际的Linux系统中，数据链路层、网络层和传输层的各种协议的相关代码已经由内核开发者写好，并且以内核线程的形式运行，我们称为**内核协议栈**。这样的话，网络应用开发者只需要关注应用层协议的处理，并且调用内核协议栈提供的接口实现功能。如果我们需要使用一些**知名**的应用层协议（**默认端口号一般小于1024**），就需要自己按协议实现或者是安装已经写好的应用程序。比如浏览器就是HTTP协议的客户端，sshd是ssh协议的服务端。另外一些应用层协议是不知名的，是由开发者自己设计并且不公开的，这些就叫做**私有协议**，比如许多网络游戏（比如《英雄联盟》、《王者荣耀》之类）的协议。

9.2.4 ifconfig

ipconfig命令（Windows版本是ipconfig）可以展示本地网卡的信息，其中ens33（有些系统是使用eth0）是表示以太网网卡，而lo表示loopback本地回环设备。

```
root@ubuntu:~# ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.135.132  netmask 255.255.255.0  broadcast
192.168.135.255
        inet6 fe80::9b93:d1a1:1bd7:31a0  prefixlen 64  scopeid
0x20<link>
```

```
        ether 00:0c:29:bf:f5:c8  txqueuelen 1000  (Ethernet)
        RX packets 1015  bytes 410518 (410.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 944  bytes 136458 (136.4 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 228  bytes 19666 (19.6 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 228  bytes 19666 (19.6 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

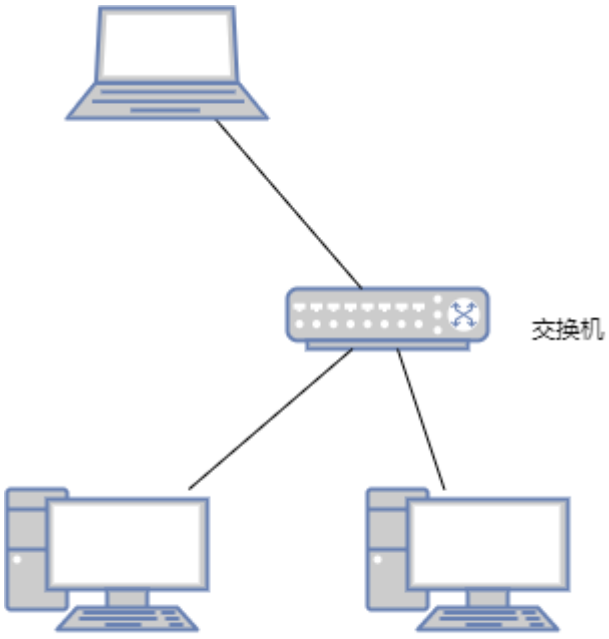
9.2.5 本地环回设备

在有些情况，通信的服务端和客户端会位于同一台主机之上，操作系统提供了一种**环回网络接口**来实现同一主机上两端通信，环回设备是一种虚拟的网络设备，通常来说环回数据会通过内核协议栈的部分层次，一般是指传输层和网络层，但是不会走到物理层设备（网卡）上。通常来说环回设备的通信不会受到MTU的限制。

9.3 以太网

9.3.1 以太网和交换机

以太网是最常用的数据链路层标准之一，它于1980年由DEC、Intel和Xerox首次发布，并随后略加修订被IEEE采纳为802.3标准。早期以太网的结构通常是共享的：多个主机共享一个网络介质传播数据，如果多个主机同时发送数据，则需要采用一些算法来避免碰撞。随着网络速度不断提升，共享以太网逐渐被一个星形拓扑接口的网络取代了：主机通过一条独享的网线和交换机连接，由交换机负责在为每个主机提供全双工通信。

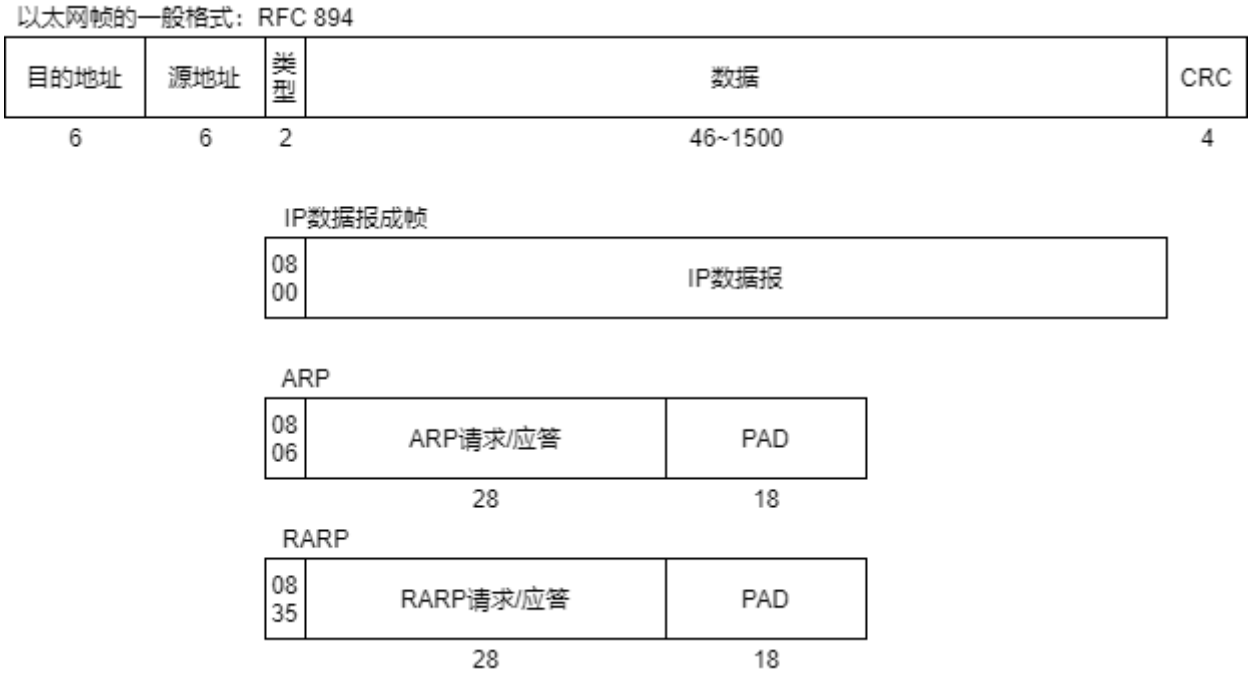


以太网的星形拓扑结构

网桥和交换机（就是高性能版网桥）是工作在**数据链路层(L2)**上的设备，它们可以用来将多个链路层网络连接起来，从而构成一个拓展的局域网。交换机拥有自己的MAC地址，并且有许多对外连接的接口，通过一段时间的“学习”，交换机可以构建一个列表，里面存储了目的MAC地址和接口编号的对应关系。

9.3.2 以太网帧

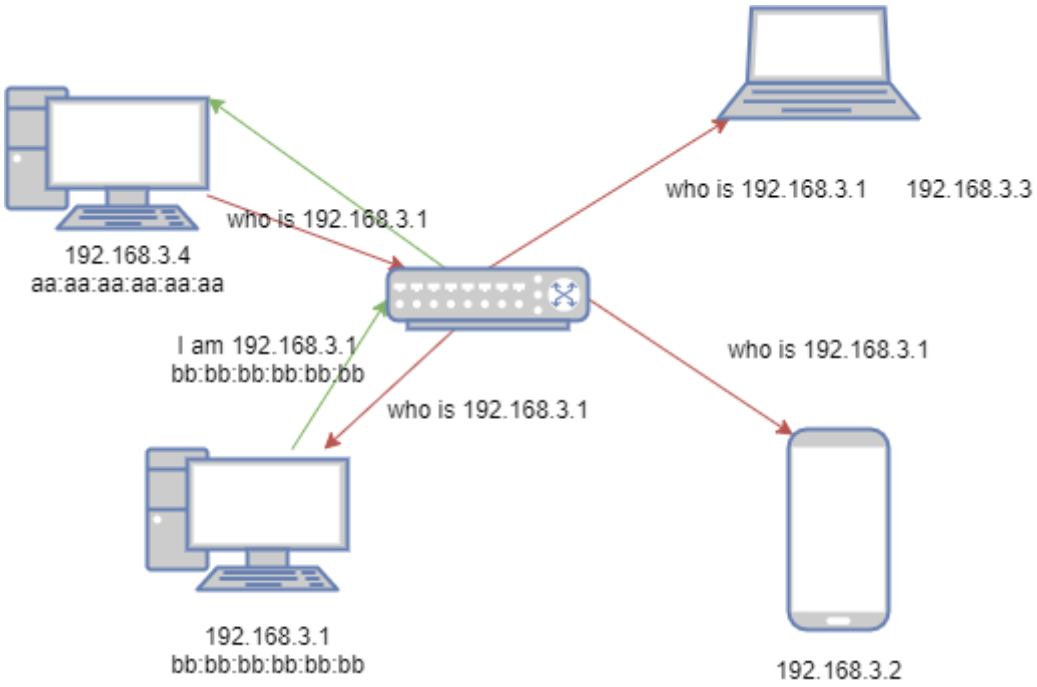
以太网中处理的协议数据单元是**以太网帧**，下面帧结构做一个简短的介绍。一个典型以太网帧如下图所示：由目的地址（6字节）、源地址（6字节）、类型（2字节）、变长的数据（46~1500字节，所以**以太网的MTU是1500**）和CRC（4个字节）。在目的地址和源地址字段当中填写的是设备的**硬件地址（MAC地址）**，而不是主机的IP地址。类型字段中说明以太网帧载荷的类型，如果载荷类型是IPv4数据报，那么类型的内容就是16进制的0800。数据字段的长度范围是46字节到1500字节，如果IP数据报的内容不足46个字节，则会在末尾填充（padding）上0。CRC部分是一种校验码，用于校验数据传递过程是否出现**比特跳变**



9.3.3 ARP协议

在网络传输中，我们使用IP地址来定位主机的位置，通常来说IP地址的选择是任意的，但是网络设备在链路层上的MAC地址是由生产厂商分配是固定的。在构建以太网帧时，网络层的IP地址不能够直接填入帧头中，故我们需要使用一种协议来执行**地址解析**，即**IP地址和MAC地址之间的映射转换——**这就是**ARP协议**的作用，ARP协议可以将32位的IPv4地址映射成48位的MAC地址，并且会随着时间动态调整以和实际的网络拓扑结构相符合。

当一台主机接入网络之后，假如它需要向自己所在局域网的设备（假设IP是192.168.3.1）发送数据，那么它就需要知道邻居IP地址和MAC地址之间的映射关系。主机会发送一个**ARP请求**，ARP请求会向在一个共享的链路层网段上所有主机发送（广播，此时以太网帧的目的MAC每一位都填1），并且获取这个问题的答案“谁拥有IP192.168.3.1，请告知自己的MAC地址？”。IP地址不是192.168.3.1的设备会忽略此请求，但是拥有IP地址192.168.3.1的设备会直接回复一个**ARP应答**给源主机，在接收到应答之后，源主机再向192.168.3.1发送数据就不再需要广播而是直接交付了。



ARP请求和应答

在Linux和windows上，我们可以使用`arp`命令显示自己本机上的arp缓存。

```
root@ubuntu:~# arp -an
? (192.168.135.1) at 00:50:56:c0:00:08 [ether] on ens33
? (192.168.135.2) at 00:50:56:f5:2f:4e [ether] on ens33
```

使用wireshark工具可以抓取网络报文。

```
这是一个典型的arp请求
Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: Giga-Byt_e2:1f:dd (40:8d:5c:e2:1f:dd)
    Sender IP address: 192.168.3.103
    Target MAC address: CloudNet_3b:b2:35 (28:3a:4d:3b:b2:35)
    Target IP address: 192.168.3.19
这是一个典型的arp响应
Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
```

```
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: CloudNet_3b:b2:35 (28:3a:4d:3b:b2:35)
    Sender IP address: 192.168.3.19
    Target MAC address: Giga-Byt_e2:1f:dd (40:8d:5c:e2:1f:dd)
    Target IP address: 192.168.3.103
```

从抓取的报文内容可以得知，arp请求和应答消息的有效部分为28个字节，不足46字节的部分由18个字节的0进行填充。有效部分依次由硬件类型、协议类型、硬件长度、协议长度、Opcode（请求为1，应答为2）、来源MAC地址、来源IP地址、目的MAC地址和目的IP地址。在arp请求中，以太网帧头部的目的MAC地址是广播地址，即全部为1，arp应答的目的MAC地址是其对应请求的MAC地址。

9.4 IP协议

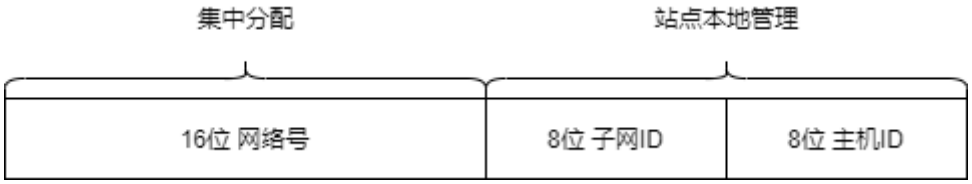
IP协议是四层模型中的核心协议，所有TCP、UDP、ICMP数据都通过IP数据报传输。IP提供一种尽力而为（就是**不可靠**）、无连接的数据报交付服务。不可靠意味着如果传递过程中出现差错，IP层可以选择丢弃数据，并且不会主动重传；无连接意味着IP协议不会记录传递过程中的路径，那同样的两端发生的不同数据报可能会走不同的路径，并且有可能不按顺序到达。

9.4.1 IP地址以及其分类

IP地址用来区分不同的主机在网络层中的位置。IPv4的地址长度为32位，为了方便描述，通常将其按8位一组分隔，并用.号隔开，这种就是点分十进制，比如192.168.1.1。IPv6的地址长度是128位，一般用8个4位十六进制描述，每个十六进制数描述一个段。IPv6的应用可以解决IP地址缺乏的问题，但是随着NAT技术的广泛使用，IPv4目前实际上还是占据了大部分市场。

在早期，每个IP地址会分为两部分，高位是网络号，低位是主机号，并且根据网络号的前缀，将其分为5类地址，A、B、C、D（这是用于组播的）和E（这是保留的地址）类，这种分类方式除了规定了网络号的前缀，还是划定了网络号和主机号的长度。随着Internet逐渐发展，这种死板的分类方式已经不适应人们的需求。一种相对自由的划分方式就是采用子网机制：把主机号的前缀作为子网ID，剩余部分作为主机ID，主机ID的长度由本地网络管理员自行划定。

子网掩码可以用来描述主机ID的长度，其长度和IP地址一样是32，其中高位部分全为1，剩余的部分全为0。前缀部分的长度说明网络号和子网ID的长度，剩余部分自然就是主机ID了。子网掩码可以用在局域网内为路由器的路由决策作出参考，而局域网外的路由决策则只和网络号部分有关。我们可以用点分十进制来描述子网掩码（比如255.255.255.0），或者在IP地址的后缀中说明网络号和子网ID的总长度（比如192.168.3.0/24）



在每个IPv4子网中，主机部分全为1的IP地址被保留为本地广播地址，比如子网为128.32.1.0/24的子网广播地址是128.32.1.255。除此以外，特殊地址255.255.255.255被保留为本地广播地址，它不会被路由器转发，通常配合UDP/IP和ICMP等协议使用。

随着CIDR技术的引入，传统的5类分类方式被废弃了，IP地址不再按照固定的长度进行划分，在分配IP地址时，除了要指定分配的网络号，还需要说明网络号的长度，可以采用类似子网掩码的方式进行描述。

下面是一些常见的用于特殊用途的IP地址：

前缀	用途
0.0.0.0/8	作为源地址时表示本地主机||作为目的地址时，表示任意IP地址
10.0.0.0/8	局域网IP地址
172.16.0.0/12	局域网IP地址
192.168.0.0/16	局域网IP地址
127.0.0.0/8	回环地址
169.254.0.0/16	链路本地地址，通常出现在DHCP自动分配IP未完成时
255.255.255.255/32	本地网络广播地址

9.4.2 IP数据报的结构

正常的IPv4头部大小为20个字节（在很少情况会拥有选项，此时头部不只20个字节），IP头部的传输是按照大端法进行的，对于一个32位值而言，首先传输高位8位，然后次高8位，以此类推。因为TCP/IP协议中所有协议的头部都采用大端法进行传输，所以大端法也称作**网络字节序**。由于大部分PC使用的是小端法，所以在构造完头部之后传输之前需要对其执行大小端转换才行。

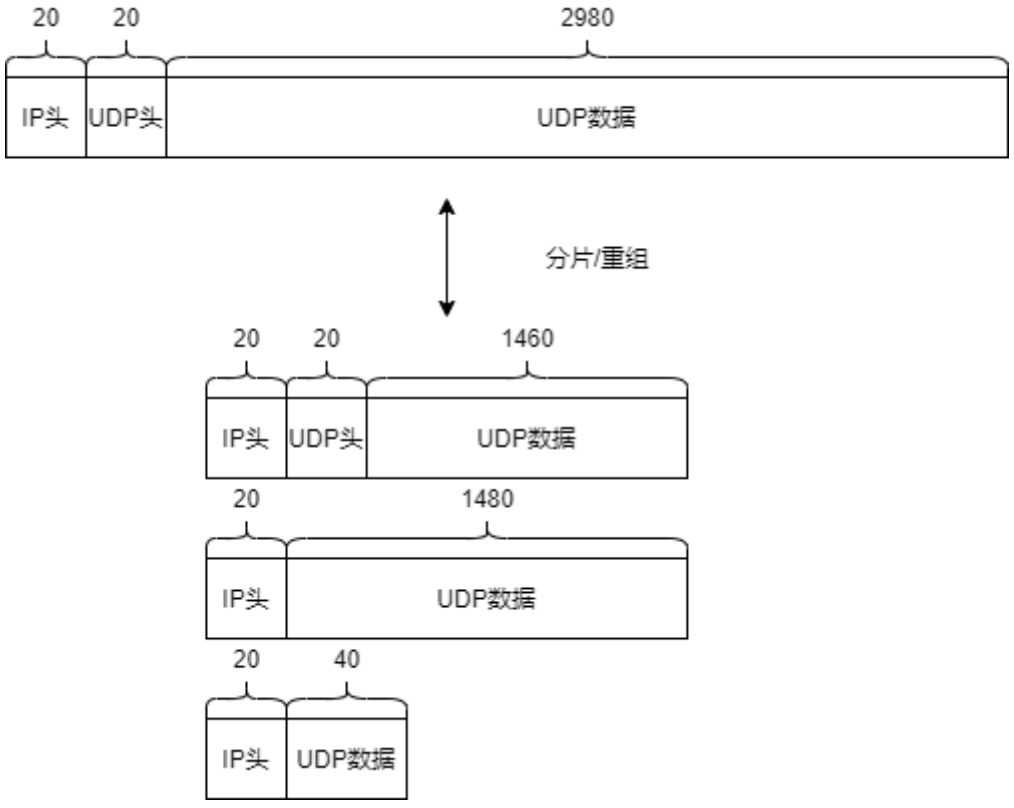
下图当中描述IPv4中IP数据报的格式。

- 版本：4位，数值4指IPv4，6指IPv6。
- 头部字段：4位，用来描述IP数据报头部的长度为多少32位。因此IP数据报头部最多只有60个字节。
- 服务类型：8位，描述服务质量和拥塞情况
- 总长度：16位，描述IP数据报的总长度（包括头部）为多少字节。这个字段有助于从带填充的以太网帧取出IP数据报的有效部分（可能这个IP数据报不足46个字节，在以太网帧当中填充了0）。
- 标识：16位，描述IP数据报的分片编号，同一个大IP数据报分解出来的多个分片拥有相同的标识。
- 标志：3位，描述是否发生分片，以及是否后续有更多的分片。
- 片偏移：13位，描述该分片在重组后的大IP数据报当中的位置，以8字节为单位。

- 生存期（TTL）：8位，描述一个数据报可以经过路由器的上限，每次路由器转发时该数值会减一。这个属性可以避免在环形路由情况下，数据报在网络中永远循环。
- 协议：8位，描述上层协议的类型，最常见的是1(ICMP)、17(UDP)和6(TCP)
- 首部校验和：16位，IP数据报头部的校验和，注意并不检查载荷部分内容，所以需要上层协议自己检查。
- 源IP地址和目的IP地址：各有32位，描述IP数据报的发送者和接收者的IP地址。

9.4.3 分片和重组

由于IP数据报的总长度限制为65535字节，这远远超过了部分链路层标准的MTU，当数据从网络层准备转移到数据链路层时，网络层会将IP数据报进行分片操作，将分解成若干个独立的IP数据报（分解之后IP数据的总长度字段改变了），并且在网络之间独立传输。一旦到达终点的目的主机之后（中间不会重组），目的主机的网络层会将分片重组成一个大IP数据报。由于重组的过程十分复杂，所以协议设计者应该尽可能**避免出现让IP数据报超过MTU**的情况。比如DNS、DHCP等就规定UDP报文长度为512字节。



9.4.4 路由器

链路层的数据是采用多跳的方式进行通信，也就是一个主机实际上只能决定数据报下一跳能转发到哪一个邻居，而不能直接和最终的目的地建立连接。在这种逐跳通信的网络之中寻找来源和目的之间一条合适的通路过程以及最终生成的通路称为**路由**。路由器是一种拥有两个以上的接口的网络设备，它们工作在网络层(L3)，路由器在经过一定的路由决策之后，可以将数据从一个接口转发到另一个接口。路由器可以充当一个端到端通信系统当中的中间结点。部分主机也支持路由功能。

路由器的主要功能就是IP转发，它需要维护一个数据结构，称作路由表，表的每一行会拥有如下信息：

- 目的地：一个32位字段，和掩码合在一起工作，用以描述目的IP地址。0.0.0.0表示所有目的地，完整的IP地址表示一个直接相连的主机。
- 掩码：和目的地址合在一起使用，用来表示目标主机或者目标网段
- 下一跳：数据报真正会被转发到的地址。0.0.0.0表示在同一个链路上，可以直接在数据链路层进行传输。

使用`route`可以查看主机的路由表，添加和删除路由表的表项。

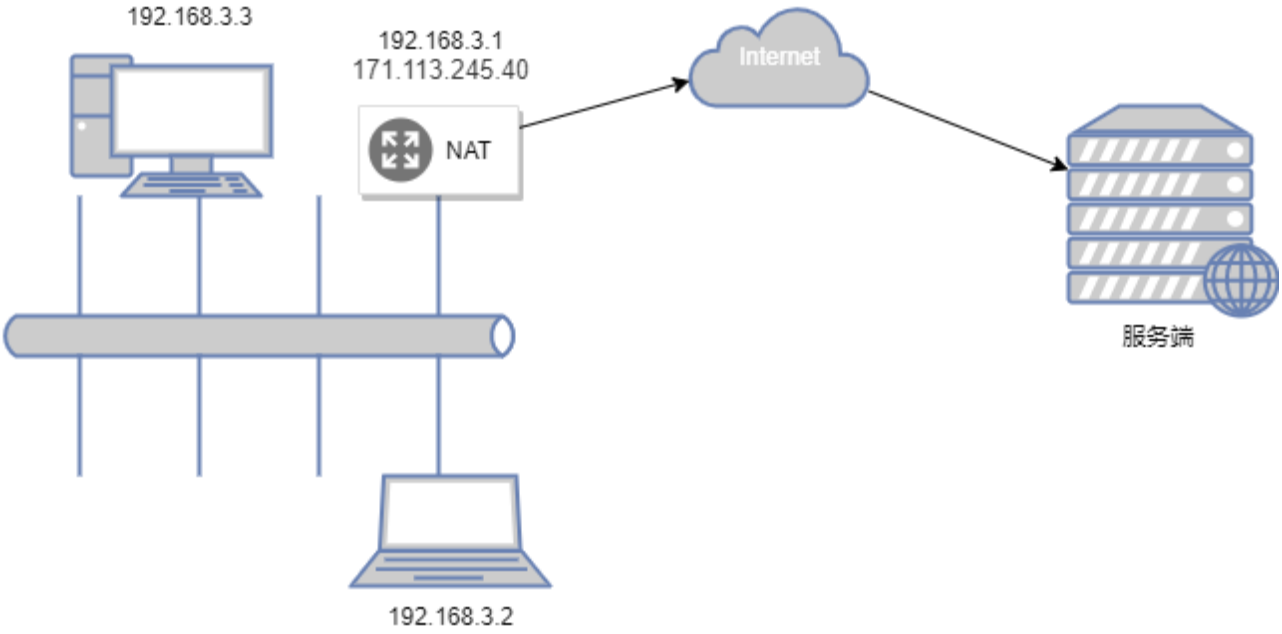
```
root@ubuntu:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref
Use Iface
0.0.0.0         192.168.135.2   0.0.0.0         UG    100    0
   0 ens33
169.254.0.0     0.0.0.0         255.255.0.0     U     1000   0
   0 ens33
192.168.135.0   0.0.0.0         255.255.255.0   U     100    0
   0 ens33
删除路由表
route del -net 0.0.0.0 gw 192.168.3.1 dev ens33
```

从路由表的信息可以看到，路由器和主机不包含到任何目的地址的完整转发路径，除非目的主机直接和其相连。

如果目的地是直接相连的主机，那么数据报就是直接通过链路层传输到目的主机，否则的话，数据报会发送给路由器（网关），再由路由器进行转发，以间接交付的形式最终到达目的地。

9.4.5 NAT简述

随着IPv4的IP地址急速地接近告罄，网络地址转换（NAT）被设计出现缓解这个问题。NAT的本质就是在不同的地方使用重复的IP地址。我们可以把网络分解成Internet和内部网络，内部网络当中只有少量的结点直接和Internet连接，并且被分配了IP地址。NAT最初被认为是解决IP地址耗尽的权宜之计，但是因为太好用了，已经成为了IPv6推广过程中的拦路虎。



NAT的原理就是重写通过路由器的数据报的识别信息。具体而言，就是在离开内部网络的时候重写数据报的源地址为公网地址，进入内部网络的时候将目的地址从公网地址改成内网地址。应用NAT之后，内部网络之外的所有主机将无法从外网主动发起对内部结点的连接，也就是内部结点无法充当服务端。

9.4.6 ping和ICMP

Internet报文控制协议（ICMP）和IP结合使用，以便提供与IP协议层配置和IP数据包处置相关的诊断和控制信息。ICMP一般认为是网络层的一部分，但是它的实现却需要基于IP，ICMP数据包是放置在IP数据报之中，所以可以认为ICMP是一种位于网络层和传输层之间的协议。

ICMP报文是封装在IP数据报之中，其结构如下：

IP头 20~60bytes	ICMP头部 4bytes	ICMP数据

使用ICMP的最经典的应用就是`ping`和`traceroute`（Windows下为`tracert`），下面是抓包内容：

```
ping的抓包
Internet Protocol Version 4, Src: 192.168.135.132, Dst:
14.215.177.38
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 84
    Identification: 0x2920 (10528)
    Flags: 0x4000, Don't fragment
    Time to live: 64
    Protocol: ICMP (1)
    Header checksum: 0x095f [validation disabled]
    [Header checksum status: Unverified]
    Source: 192.168.135.132
    Destination: 14.215.177.38
Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x82d4 [correct]
    [Checksum Status: Good]
    Identifier (BE): 10388 (0x2894)
    Identifier (LE): 37928 (0x9428)
    Sequence number (BE): 1 (0x0001)
    Sequence number (LE): 256 (0x0100)
    [Response frame: 135]
    Timestamp from icmp data: Jan 16, 2022 23:14:47.000000000 中国标
准时间
    [Timestamp from icmp data (relative): 40.674858000 seconds]
    Data (48 bytes)
```

9.5 TCP协议

传输控制协议（TCP）是整个四层模型当中最重要的协议，它工作在传输层，其目标是在不可靠的逐跳传输的网络层之上，构建一个**可靠的、面向连接的、全双工的端到端**协议。

9.5.1 TCP的基本特性

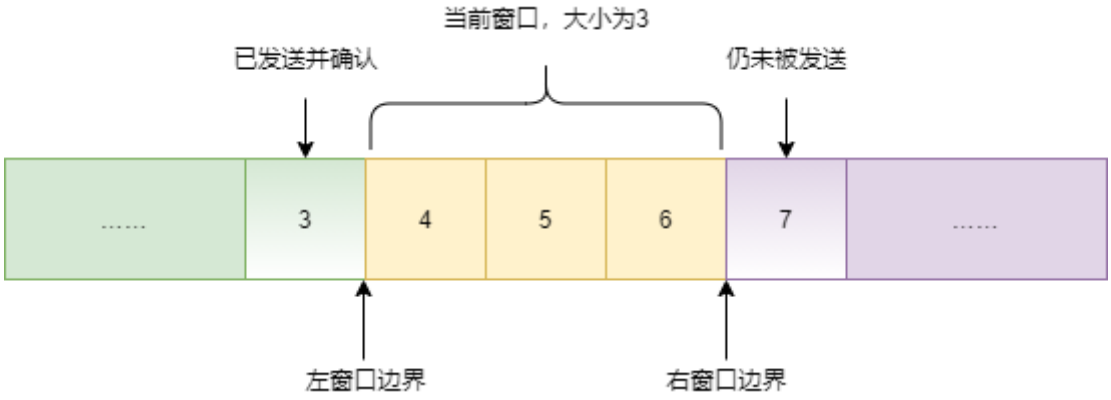
由于通信媒介可能会丢失或者改变信息，所以TCP采用了一种**自动重复请求**的策略来提供可靠性。当发送方向接收方发送分组时，发送会一直重复发送分组直到分组确认被接收，所以每次接收方收到分组的时候，都需要回复一个确认信息（ACK）给发送方。在收到ACK之前，发送方需要占用存储空间保存所有的未确认分组，并且需要维持一个计时器，以确定何时重传这些分组。

采用重传的策略之后，就可能会出现这种情况：一个分组的ACK返回到发送方之前，定时器超时了，此时重复分组就再次发送给了接收方。为了让接收方有能力区分不同的入境分组，每个分组必须添加一个**序号**，对应的ACK信息中也必须包含序号的信息。

为了增加网络的吞吐量，我们可以采用流水线的方式发送网络分组，也就发送一个分组之后，不等到这个分组被确认就继续发送下一个分组。同时处理多个分组会导致一些复杂的情况：发送方的发送速率如果远大于接收方的处理速率怎么办？发送方收到了乱序的分组如何将其整理成有序的？

滑动窗口机制有助于解决上述问题，分组窗口是已经被发送方注入但是还没有确认的分组的集合，该集合中分组的数量称为窗口大小，尽管通信的过程中会涉及到大量的分组，但是用户只需要关心窗口内数据的传输和窗口的变化即可。

下图是发送方窗口的一个示例，3号分组刚刚被确认，所以它从窗口中移除，并且发送端可以释放掉它的内存，7号分组还未发送，一旦发送则7号就要加入到窗口。如此重复上述行为，则整个窗口就不断地右移。



接收端也会维持一个窗口，它记录着那些分组已经被接收和确认，那些分组是下一步期望的，以及哪些分组即便被接收也将会因内存限制而丢弃。

滑动窗口机制有助于解决这两个问题：流量控制和拥塞控制。使用滑动窗口进行流量控制的时候，需要采用窗口通告的机制，也就是说接收方通知发送方使用多大的窗口。通常窗口的更新和ACK报文一起回复到发送方。流量控制可以避免发送接收双方处理速率不一致的问题，但是如果不是接收方的限制而是中间网络的原因限制了网络的传输，这时候（比如接收方重复发送了多个ACK时）就需要拥塞控制了，拥塞控制的相关过程和理论可以查阅相关RFC文档资料，此处不予说明。

我们把TCP的分组称作TCP（报文）段，当TCP发送一个报文段的时候，发送方会为每个窗口设置一个重传定时器，另外，TCP采取了累积确认的机制，如果一个ACK丢失了，但是后续的ACK到达了的话，可以认为之前的所有报文段都被确认了。

9.5.2 面向连接的协议

3次握手

TCP是一个面向连接的协议，在通信双方真正交换数据之前，必须先先相互联系建立一个TCP连接，这个就类似于电话的开头的“喂”的效果。TCP是一个全双工协议，双方都需要对连接状态进行管理。每一个独特的TCP都由一个**四元组**唯一标识，组内包括通信双方的IP地址和端口号，建立连接的过程通常被称作是**3次握手**。虽然是全双工的通信，但是有一次建立连接和确认行为可以合并在一起，所以只需要传输3次报文段即可。下面是其步骤：

- 客户端发起连接，发送一个SYN报文给服务端，然后说明自己连接的端口和客户端初始序列号seq1。
- 服务端收到SYN报文，也需要发起反方向的连接，所以发送一个SYN报文给服务端，说明自己连接的端口和服务端初始序列号seq2，除此以外，这个报文还可以携带一个确认信息，所以把seq1+1作为ACK返回。
- 客户端收到服务端的SYN之后，需要确认，所以把seq2+1作为ACK返回给服务端。

3次握手最主要的目的是为了建立连接，并且交换**初始序列号**。在TCP连接过程中，如果存在一个老旧的报文（上一次连接时发送的）到达服务端，服务端可以根据其序列号是否合法来决定是否丢弃。有些情况可能出现双方同时发起连接的情况，这个时候就需要4个报文段来建立连接了。

> *使用2次握手可不可行？*

> *答案是否定的，因为服务端发起的SYN未确认。一种典型的场景就是客户端发起SYN，第一个SYN超时并重传，第二个SYN到达并建立连接，之后再完成连接并关闭，倘若关闭之后，第一个SYN到达服务端，此时服务端就会认为对方建立连接，并回复SYN+ACK，由于没有确认，所以服务端并不知道客户端的状态，此时客户端完全可能已经关闭，那服务端就会陷入永久等待了。*

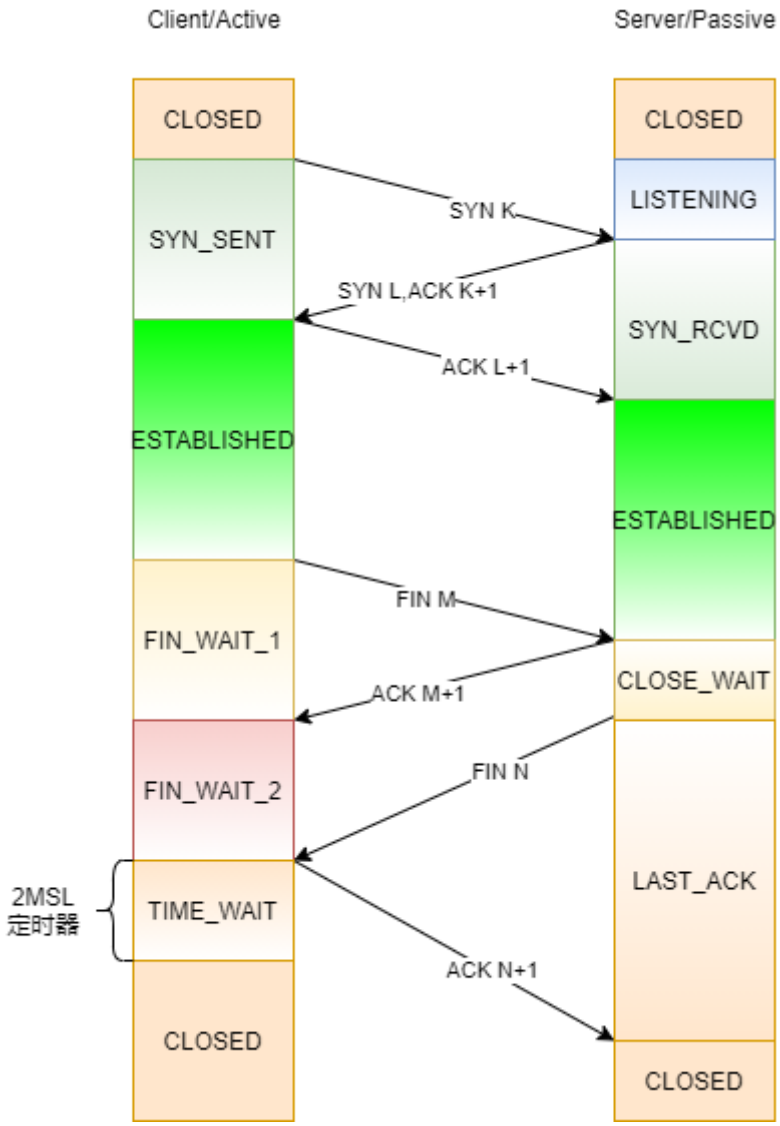
4次挥手

断开连接的过程要更加复杂一些，通信双方谁都可以主动断开连接，但是由于TCP是全双工连接，所以一个方向断开并不意味着反向的数据已经传输完成，所以每个方向的断开是相对独立的，这样的话两个方向各有一次断开和确认，总共需要4次报文段的传输，即**4次挥手**。下面是其具体流程：

- 主动关闭方发送一个FIN段表示希望断开连接。
- 被动关闭方收到FIN段，并且回复一个确认信息。其上层应用会收到一个EOF，被动关闭方继续传输剩余的数据。
- 被动关闭方发送完数据了，发送一个FIN段。
- 主动关闭方回复一个确认，并且等待一段时间（2MSL，MSL指单个报文在网络中的最长生存时间）。

在第2次挥手之后，TCP连接此时处于一种半关闭的状态。可以任为现在是一个单工通信方式（被动关闭方-->主动关闭方）。

9.5.3 TCP的状态转换



上图说明了在TCP连接建立、通信和关闭的过程中，通信双方状态的一个状态转化。需要特别注意的就是TIME_WAIT状态。TIME_WAIT状态的存在意义有两个原因：

- 可靠地实现TCP全双工连接的终止。如果被动方没收到主动方发送的ACK，那么被动方会重传FIN，主动需要等待一段时间（先一次ACK，再一次FIN合计2MSL）就是为了处理这种情况的发生。
- 允许老的分段在网络中消逝。有可能两端建立了不只一条连接，在新的连接建立之前，老的连接中正在传递的分组就已经在网络中消失了。

如果不加特殊的处理，在处于TIME_WAIT状态下，用户无法对同样的四元组建立新的连接。我们可以使用命令来查看TIME_WAIT的时长。

```
$ cat /proc/sys/net/ipv4/tcp_fin_timeout
60
```

9.5.4 TCP报文头部

上图中描述了TCP报文首部的各个字段，具体含义如下：

- 序号：即SEQ的值
- 确认需要：即ACK的值，描述预期接收的下一个序列号。注意发送一个ACK和发送一个普通报文的消耗是一样的。
- 首部长度：首部的长度是可变的，以32位为单位。首部长度最短为20字节，最长为60字节。
- URG：紧急。
- ACK：确认号字段有效。连接建立以后通常一直有效。
- PSH：推送。
- RST：重置连接，出现在连接出错时。
- SYN：发起连接。
- FIN：发起关闭。
- 窗口大小：通告一个窗口大小以限制流量。
- 校验和：校验传输中的比特跳变错误。
- 紧急指针：向对端提供一种特殊标识。
- 选项：最常见的选项是**最大段大小（MSS）**，用来描述后续希望接收到的报文段的最大值，这个数值通常受限于MTU，比如MTU为1500，IP数据报头部为20字节，TCP头部为20字节，则MSS是1460。

9.6 UDP协议

UDP是一种保留消息边界的简单的面向数据报的传输层协议。它不提供差错纠正、流量控制和拥塞管理等功能，只提供差错校验，但是一旦发现错误也只是简单地丢弃报文，不会通知对端，更不会有重传。由于功能特别简单，所以UDP的实现和运行消耗特别地小，故UDP协议可以配合一些应用层协议实现在一些低质量网络信道上的高效传输。许多早期的聊天软件或者客户端游戏都采用了基于UDP的应用层协议，这样能最好地利用性能，同时在比较差的网络状态下提供更良好的服务。

9.6.1 UDP报文头部

UDP的报文结构非常简单：

- 长度：指UDP报文的总长度（包括UDP头部），实际上这个长度是冗余的，报文长度可以根据IP报文长度计算而来。
- 校验和：用于最终目的方校验，出错的报文会直接丢弃。