TCP/IP协议栈

弄清协议/机制产生的原因、背景/解决的问题，与弄清原理同样重要

1. 应用层

（1）应用层程序使用端口/协议总结

- 广播/组播——UDP
- DNS——UDP
- traceroute——ICMP（属于应用层直接调动传输层）
- ping——ICMP
- telnet——23/TCP
- SSH——22/TCP
- HTTP——80/TCP
- HTTPS——443/TCP
- SMTP——25/TCP
- FTP——21/TCP

（2）DNS——域名系统（domain name system）

- 实现**输入主机名，返回对应IP**的黑盒子。应用程序需要IP时，将主机名传递给当前主机上运行着的DNS客户端，DNS客户端通过查询返回对应IP给应用程序。
- DNS黑盒的内部运作很复杂。

  DNS解析详细过程1

  DNS解析详细过程2
- 区（Zone）——一个服务器所负责管辖(或有权限)的范围叫做区(zone)。

  采用上述的树状结构，每一个节点都采用一个域名服务器，这样会使得域名服务器的数量太多，使域名服务器系统的运行效率降低。
  各单位根据具体情况来划分自己管辖范围的区。但在一个区中的所有节点必须是能够连通的。每一个区设置相应的**权威（权限）域名服务器**，用来**保存该区中的所有主机到域名IP地址的映射**。顶级域名服务器也可以算作是权威域名服务器，只不过由于其特殊性，我们专门把它划分为一类。因此权威域名服务器通常是指顶级域名以下的管理二级、三级、四级等域名的服务器。

- 本地域名服务器

  电脑上网时IPv4或者IPv6设置中填写的那个DNS。这个有可能是手工指定的或者是DHCP自动分配的

这类服务器不属于上面的层次结构，当一个主机(个人电脑)发出DNS请求时，查询请求就被发送到本地域名服务器，本地域名服务器负责回答这个查询，或者代替主机向域名空间中不同层次的权威域名服务器查询，再把查询的结果返回给主机。如常用的114.114.114.114和8.8.8.8。

- DNS缓存

  本机上有一个DNS缓存——hosts文件。本地域名服务器、各级域名服务器上都有DNS缓存。本机首先在hosts上找目标主机的IP，找不到的话就向本地域名服务器发出请求，本地域名服务器也先查找DNS缓存，找不到的话再向根域名服务器发出查询，**把查询结果返回的同时也存入缓存备查。**各级域名服务器查询之前都先查找自身的DNS缓存。

- DNS的TTL——这个参数告诉本地DNS服务器，域名缓存的最长时间

  阿里云解析来举例，阿里云解析默认的TTL是10分钟，10分钟的含义是，本地DNS服务器对于域名的缓存时间是10分钟，10分钟之后，本地DNS服务器就会删除这条记录，删除之后，如果有用户访问这个域名，就要重复一遍上述复杂的流程。

- 递归查询和迭代查询

  localhost向本地域名服务器的查询是递归查询，即localhost将查询工作交给了本地域名服务器，本地域名服务器替localhost完成后续查询任务，返回结果给localhost，类似编程语言中的异常向调用处层层抛出。

  本地域名服务器向根域名服务器、顶级域名服务器、权威域名服务器的查询是迭代查询。根域名服务器不是亲自去查询，而是返回对应的顶级域名服务器的IP，由本地域名服务器去查询。

  例：主机m.xyz.com查询y.abc.com

  localhost：m.xyz.com ----------> 本地域名服务器dns.xyz.com ----------> root域名服务器

  <----------

  ---------- 对应的顶级域名服务器dns.com的IP

  本地域名服务器dns.xyz.com ----------> 顶级域名服务器dns.com

  <----------

  ---------- 权威域名服务器dns.abc.com的IP

  本地域名服务器dns.xyz.com ----------> 权威域名服务器dns.abc.com

  <----------

  ---------- 目标主机y.abc.com的IP

  目标主机y.abc.com的IP

  localhost：m.xyz.com <----------本地域名服务器dns.xyz.com

---

2. 网络层——建立主机host到主机的通信（TCP/IP核心）

IP数据报以**比特流bit**形式在物理连接上传输

（1）IP地址格式

- IP=net-id+host-id，给出了**网络部分**，给出计算机所处的子网；**主机部分**计算机的主机地址。

  IPv4——32bit

  IPv6——128bit

- 划分子网

  子网掩码——那我怎么知道网络部分和主机部分分别占几位呢？解决方案是引入**子网掩码**，他也是32位二进制数，并规定网络部分全为1，主机部分全为0。子网掩码实质是确定IP地址中网络号net-id和主机号host-id的位数

  有了子网掩码，就知道了哪些是网络部分，进而**只需要把两个IP地址网络部分的网络部分对比，就知道是否在同一个子网中**

  划分子网的**分片IP数据报** 路由转发机制 ——从分片IP数据报的首部获取目的地址，与当前路由所在网段的子网掩码AND，若与当前网段号相同，则转发到当前子网中的主机；若不同，则从当前路由的路由表依次将目的地址与子网掩码AND，若得到相同的网段则转发到相应的网关（路由）；否则，转发到默认路由或转发分组出错。

- 网段：主机号少2个，host-id全1表示广播地址（代表向所有主机发送），全0表示网段号。

（2）IP首部校验和

IP首部校验和计算方法

校验和计算时最高位产生进位时不能舍去，需要拆分后，将高位加到低位上

IP首部校验算法

- 计算：

> 将某个分片IP数据报的校验字段清零
>
> 对首部的20个字节，每2字节进行反码求和（求和后取反），若相加后最高位有进位，那么不能舍弃，一定要加到低位，才能是结果正确。如0x0319BB 拆分成0X03+0X19BB

- 校验：

> 将接收到的分片IP数据包的校验字段和同每2字节进行反码求和，若结果为**FFFF**则表明IP数据包无错误，否则将该IP数据包片丢弃

对如下十六进制数据求反码校验和：

0x45c0,0x0044,0x0009,0x0000,0x0159,0x0000（归零的校验和）,0x5900,0x0009,0xe000,0x0005

对以上数据直接相加得结果：0x180bd

按照2中规则，对此数据的处理应该是将16位数最高位的进位0x01与0x80bd相加，即得到中间结果：0x80be

按照3中规则对其取反即得校验和：0x7f41

(3) IPv4数据报（datagram）格式

- 报头最少20字节。**总长46B~1500B。**
- 标识16bit

  由同一数据报分片得到的`分片数据报`的标识

- 标志3bit

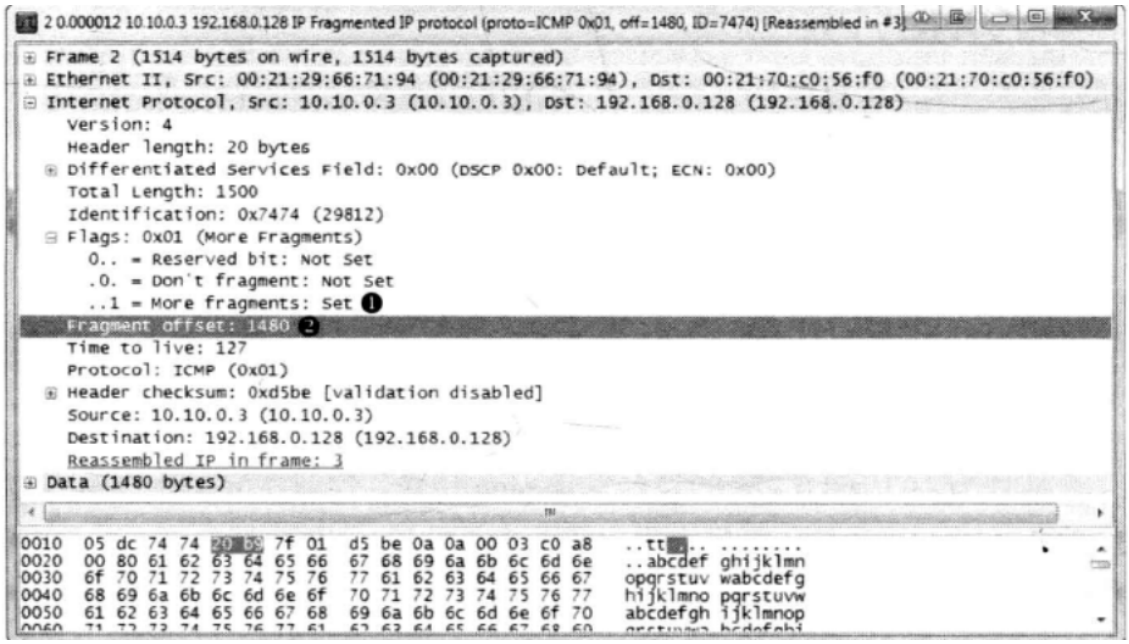
  > 未使用1bit、DF不分片1bit、MF片未完1bit

```
> Frame 36810: 1444 bytes on wire (11552 bits), 1444 bytes captured (11552 bits) on interface \Device\NI
> Ethernet II, Src: Charge-A_05:aa:08 (4c:bc:98:05:aa:08), Dst: Qualcomm_00:00:00 (00:a0:c6:00:00:00)
v Internet Protocol Version 4, Src: 183.251.24.56 (183.251.24.56), Dst: 192.168.0.100 (192.168.0.100)
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0xd4 (DSCP: Unknown, ECN: Not-ECT)
    Total Length: 1430
    Identification: 0x4c1d (19485)
  v Flags: 0x40, Don't fragment
      0... .... = Reserved bit: Not set
      .1.. .... = Don't fragment: Set
      ..0. .... = More fragments: Not set
    Fragment Offset: 0
    Time to Live: 51
    Protocol: UDP (17)
    Header Checksum: 0x6426 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 183.251.24.56 (183.251.24.56)
    Destination Address: 192.168.0.100 (192.168.0.100)
> User Datagram Protocol, Src Port: 30962 (30962), Dst Port: 7577 (7577)
> Data (1402 bytes)
```

- 片偏移13bit

  标明`分片数据区`在原数据报的`数据区`的位置。字节/8。

  下图中片偏移为1480，而IP数据区长度为1480B，说明这恰好是第二片。MF=1表明这不是最后一片。



- TTL（生存时间）

  一旦经过一个处理它的路由器，它的值就减1。当该字段为0时，数据报就丢弃，并发送ICMP报文通知源主机，因此可以防止进入一个循环回路时，数据报无休止地传输下去。

- 上层协议

  8bit，标明传输层处理该IP数据报应使用的协议（UDP/TCP）。

- **分片重组机制**

  分片——把一个IP数据报为了适合网络传输而分成多个数据报的过程称为分片，被分片后的各个**分片IP数据报**可能经过不同的路径到达目标主机。

  一个IP数据报在传输过程中可能被分片，也可能不被分片。如果被分片，分片后的IP数据报和原来没有分片的IP数据报结构是相同的，即也是由IP头部和IP数据区两个部分组成。

  分片后的IP数据报，**数据区是原IP数据报数据区的一个连续部分**，头部是原IP数据报头部的复制，但与原来未分片的IP数据报头部有两点主要不同：标志和片偏移。

  将原数据报分片后加入新的报头（报头是原数据报报头的复制），通过网络的不同路径传输到目标主机。提高了效率，当出现错误时的代价较小。

  重组——通过`标识`判断是否属于同一数据报；通过`标志`的`MF`判断是否是最后一片；根据`偏移量`确定在数据报中的位置。

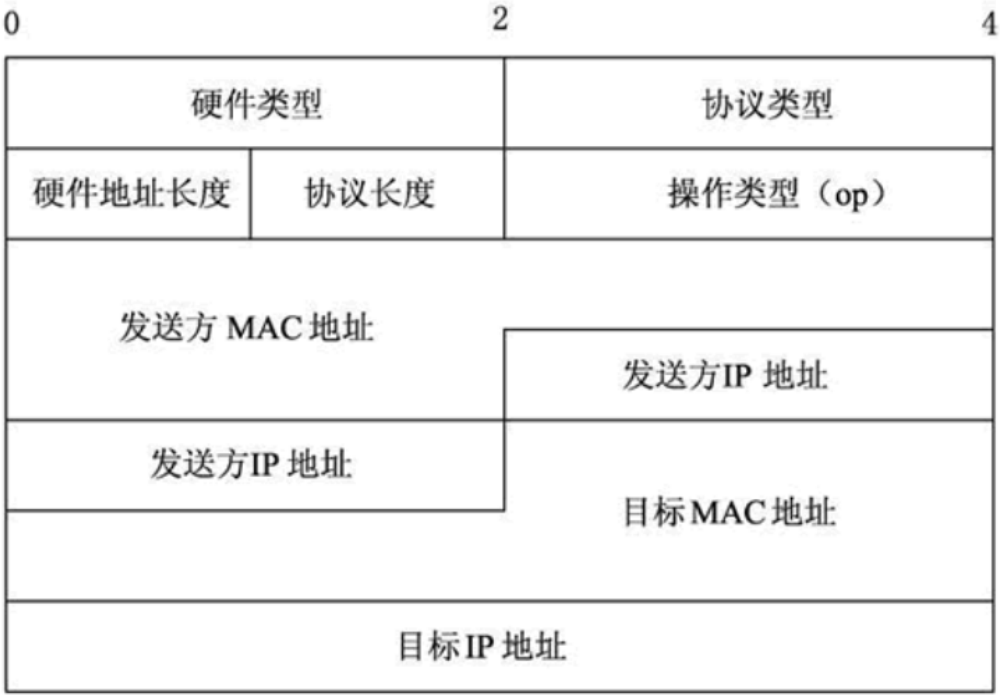
（4）路由转发分片IP数据报

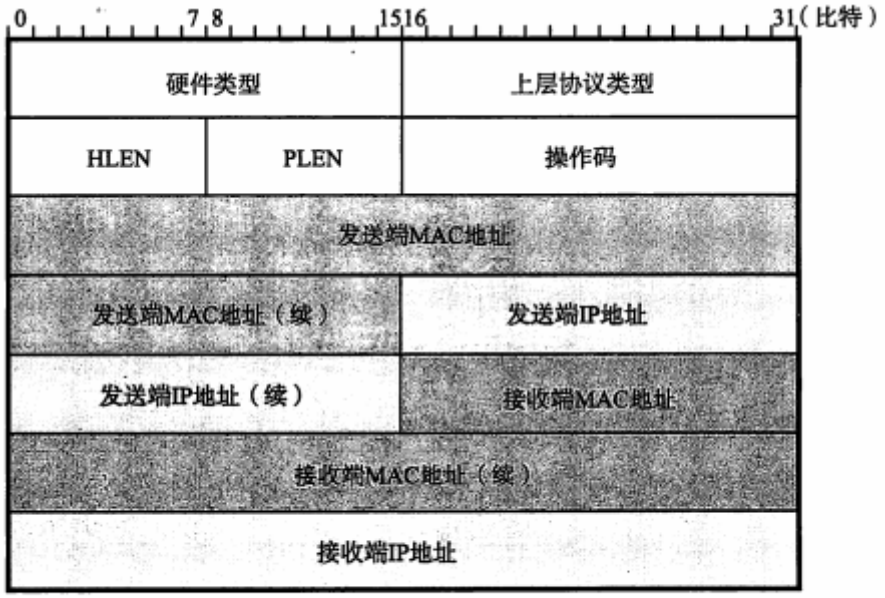
通过查找路由表，判断是否在当前局域网以及不在局域网时的转发路由是什么（默认路由）。关键是，路由可以同时处在两个不同的局域网中。

- 若目的主机和当前路由在同一个局域网中（网络号相同），则直接通过本路由的某个接口交付。
- 若目的主机和当前路由不在同一个局域网，则当前路由器通过路由表查找到转发路由，将数据报转发到转发路由。由转发路由再在目标网络中转发数据报。

（5）ARP——地址解析协议

- IP <==========> MAC
- 触发ARP的条件：目标IP地址对应的MAC地址在ARP的缓存表中不存在。当目的主机的MAC地址在源主机的ARP缓存中可以找到时，不触发ARP。
- 通常ARP包会被路由器隔离，只在一个局域网内有效，由交换机来完成。（但是采用代理ARP的路由器也可将ARP包发送到邻近网段，从而不在一个网段的节点可以像在同一个网段中进行通信）
- ARP请求报文是以**广播**的形式发送；ARP响应报文是以**单播**的形式回复。
- ARP报文格式





HLEN：MAC地址长度=6（字节）
PLEN：IP地址长度=4（字节）

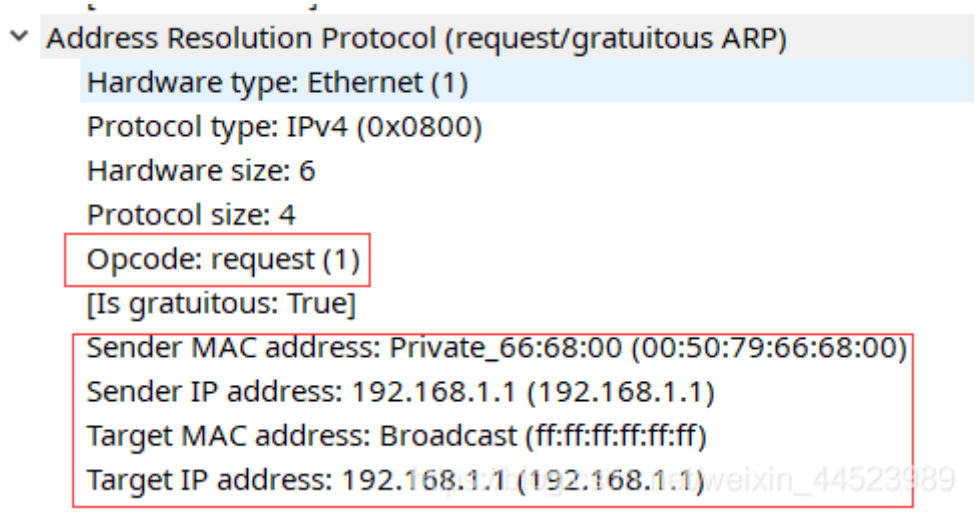
> ARP 报文总长度为 28 字节，MAC 地址长度为 6 字节，IP 地址长度为 4 字节。
>
> 其中，每个字段的含义如下。
>
> - 硬件类型：指明了发送方想知道的硬件接口类型，以太网的值为 1。
> - 协议类型：表示要映射的协议地址类型。它的值为 0x0800，表示 IP 地址。
> - 硬件地址长度HLEN和协议长度PLEN：分别指出MAC地址和IP地址的长度，以字节为单位。对于以太网上 IP 地址的ARP请求或应答来说，它们的值分别为 6 和 4。
> - **操作类型Opcode：用来表示这个报文的类型，ARP 请求为 1，ARP 响应为 2，RARP 请求为 3，RARP 响应为 4**
> - 发送方 MAC 地址：发送方设备的硬件地址。
> - 发送方 IP 地址：发送方设备的 IP 地址。
> - 目标 MAC 地址：接收方设备的硬件地址。
> - 目标 IP 地址：接收方设备的IP地址。
>
> ARP 数据包分为**请求包和响应包**，对应报文中的某些字段值也有所不同。

> - ARP 请求包报文的操作类型（op）字段的值为 request(1)，目标 MAC 地址字段的值为 Target 00：00：00_00：00：00(00：00：00：00：00：00)（广播地址）。
> - ARP 响应包报文中操作类型（op）字段的值为 reply(2)，目标 MAC 地址字段的值为目标主机的硬件地址。

- **在ARP缓存表中未找到对应MAC时ARP协议工作：首先生成ARP请求报文（操作类型字段为1）——将自身MAC地址作为源MAC地址，将目的MAC地址设为00-00-00-00-00-00（表示目的MAC未知），同时将源目IP填充进报文对应位。随后，将生成的ARP请求报文传到数据链路层封装成MAC帧，从源主机发出时的源MAC设为源主机MAC，若是从网关发出则设为对应网关的MAC，目的MAC设为FF-FF-FF-FF-FF-FF作为广播帧。**
- 免费/无偿ARP

  <1> 免费ARP即源主机主动发出的ARP广播请求，强制更新局域网中其他主机的ARP表。是一个目标IP为自己IP的ARP request。一般当源主机的IP发生改变时发送。

  特点：将源目IP都设为自己的IP



  <2> 除此以外，主机还可确定当前网络中是否有主机的IP和自己的IP是一样的。如果这个ARPrequest发出后，收到了ARP reply，说明当前LAN（或者VLAN）有主机用了和本机一样的IP。这个时候系统通常会提醒用户IP地址冲突。如果没有收到ARP reply，说明LAN（或者VLAN）中没有主机和自己IP地址一样。

  特点：目的MAC为00-00-00-00-00-00，源IP设为0.0.0.0

6. 地址冲突检测（ACD）

Gratuitous ARP发现地址冲突后什么都不能做，只能提示用户发生了地址冲突。所以RFC5227提出了一个新的机制：ACD（Address Conflict Detection）

ACD可以对地址冲突做出反应，ACD定义了ARP probe和ARP announcement两种ARP包（都是ARP request，只是填充内容不太一样）

ARP probe：用于检测IP地址冲突，Sender IP填充为0，填充为0是为了避免对其他主机的ARP cache造成污染（因为可能已经有主机正在使用候选IP地址了，不要影响别人的正常通行嘛），Target IP是候选IP地址（即本机想要使用的IP地址）

ARP announcement：用于昭示天下（LAN）本机要使用某个IP地址了，是一个SenderIP和Traget IP填充的都是本机IP地址的ARP request。这会让LAN(VLAN)中所有主机都会更新自己的ARP cache，将IP地址映射到发送者的MAC地址。

```
> Frame 4728: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface \Device\NPF_{DF3C274D-D0F1-4908-BCF4-6A7B51
v Ethernet II, Src: LAPTOP-Neo.local (00:a0:c6:00:00:00), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  > Destination: Broadcast (ff:ff:ff:ff:ff:ff)
  > Source: LAPTOP-Neo.local (00:a0:c6:00:00:00)
    Type: ARP (0x0806)
v Address Resolution Protocol (ARP Probe)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    [Is probe: True]
    Sender MAC address: LAPTOP-Neo.local (00:a0:c6:00:00:00)
    Sender IP address: 0.0.0.0 (0.0.0.0)
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: LAPTOP-Neo.local (192.168.0.100)
```

若已有主机使用了设置IP，则使用了IP的主机会返回一个ARP reply

```
v [Duplicate IP address detected for 192.168.1.1 (00:50:79:66:68:01) - also in use by 00:50:79:66:68:00 (frame
  > [Frame showing earlier use of IP address: 3]
    [Seconds since earlier frame seen: 195]
v Address Resolution Protocol (request/gratuitous ARP)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    [Is gratuitous: True]
    Sender MAC address: Private_66:68:01 (00:50:79:66:68:01)
    Sender IP address: 192.168.1.1 (192.168.1.1)
    Target MAC address: Broadcast (ff:ff:ff:ff:ff:ff)
    Target IP address: 192.168.1.1 (192.168.1.1)
```


- ARP命令

> arp -a 查看本机的ARP缓存表
>
> arp -d 删除ARP缓存中的条目（需要root权限）
>
> arp -s 网关ip 网关mac 向ARP缓存中添加条目（半永久，不会老化，但是系统重启时消失。如：arp -s 192.168.1.1 00-aa-00-62-c6-09）

- ARP缓存表——每块网卡都有ARP缓存表

```
接口: 192.168.0.100 --- 0x15
  Internet 地址         物理地址              类型
  192.168.0.1           4c-bc-98-05-aa-08     动态
  192.168.0.255         ff-ff-ff-ff-ff-ff     静态
  224.0.0.22            01-00-5e-00-00-16     静态
  224.0.0.251           01-00-5e-00-00-fb     静态
  224.0.0.252           01-00-5e-00-00-fc     静态
  239.255.255.250       01-00-5e-7f-ff-fa     静态
  255.255.255.255       ff-ff-ff-ff-ff-ff     静态
```

- ARP工作过程

> 1. 每个主机都会在自己的 ARP 缓冲区中建立一个 ARP 列表，以表示 IP 地址和 MAC 地址之间的对应关系。
> 2. 主机（网络接口）**新加入网络时**（也可能只是mac地址发生变化，接口重启等），会发送免费ARP报文把强制更新其他主机的ARP表。
> 3. 网络上的主机接收到免费ARP请求报文后，会更新自己的ARP缓冲区。将新的映射关系更新到自己的ARP表中。

> 4. 某个主机（192.168.1.2）需要发送报文时，首先检查 ARP 列表中是否有对应 IP 地址的目的主机的 MAC 地址，如果有，则直接发送数据；如果没有，当前主机就发送一个ARP请求包（该ARP request包括有：源主机 IP 地址，源主机 MAC 地址，目的主机的 IP 地址（192.168.1.3）、目的MAC（置为00-00-00-00-00-00，因为目的MAC未知）。ARP请求包到达数据链路层被封装为MAC帧，源MAC为源主机/网关的MAC，目的MAC为FF-FF-FF-FF-FF-FF作为广播帧。
> 5. 当本网络的所有主机收到该 ARP 数据包时：
>
> （A）首先检查**数据包中的 IP 地址是否是自己的 IP 地址**，如果**不是，则忽略该数据包。**
>
> （B）如果是，**则首先从数据包中取出源主机的 IP 和 MAC 地址写入到 ARP 列表中，如果已经存在，则覆盖。**
>
> （C）然后将自己的 MAC 地址写入 ARP 响应包中传给交换机，交换机**更新MAC表**中目的主机对应的表项，再将ARP响应包单播给源主机。
>
> 6. 源主机收到 ARP 响应包后。将目的主机的 IP 和 MAC 地址写入 自己的ARP缓存 列表，并利用此信息发送数据。如果源主机一直没有收到 ARP 响应数据包，表示 ARP 查询失败。

- 静态ARP/动态ARP

  ARP表项分为静态和动态

  `静态ARP`——需要管理员手工指定建立IP-MAC映射表，需要管理员手工建立和维护，项存在硬盘中，而不是缓存表，计算机重新启动后仍然存在。没有老化时间，不会刷新。

  `动态ARP`——通过报文去学习ARP表项，不需要管理员手工建立和维护，有老化时间，会刷新。

- **动态ARP的老化与刷新**

  linux中ARP表的老化机制

  `动态条目老化时间（Dynamic ARP cache timeout）`

  > 不同的系统对ARP表的老化时间设定不太一样，思科是 5分钟，华为是 20分钟，Windows2000/XP环境中是2分钟

  `动态条目最大数量（Dynamic ARP cache size）`

  > 有些系统会设置动态条目的最大条数，超过最大条数，会相应的让第一条条目被删除，而最新被存入的条目放在最后一条条目后面，比如，依次存入123456...30，当存入第31条时，超过最大条数，那么就会把第1条删除，把第31条放在第30条的后面，变成23456...31

  `ARP老化时间` 重置条件

  > 满足以下任一条件时，设备的ARP表项的老化时间定时器会重置：
  >
  > 1，设备相应的ARP表项更新时；
  >
  > 2.，设备调用（引用）ARP表项转发数据后

  `ARP表项更新` 条件

  https://blog.csdn.net/wenshifang/article/details/29265721

  > 在实际的环境中，只有同时满足以下两个条件时，设备的ARP表项才会更新：
  >
  > 1，设备收到来自某IP的ARP请求包或免费ARP包；
  >
  > 2，设备的现有ARP表项中已经存在该IP对应的ARP表项。

> 其他的非ARP报文不会对设备的ARP表项产生影响。

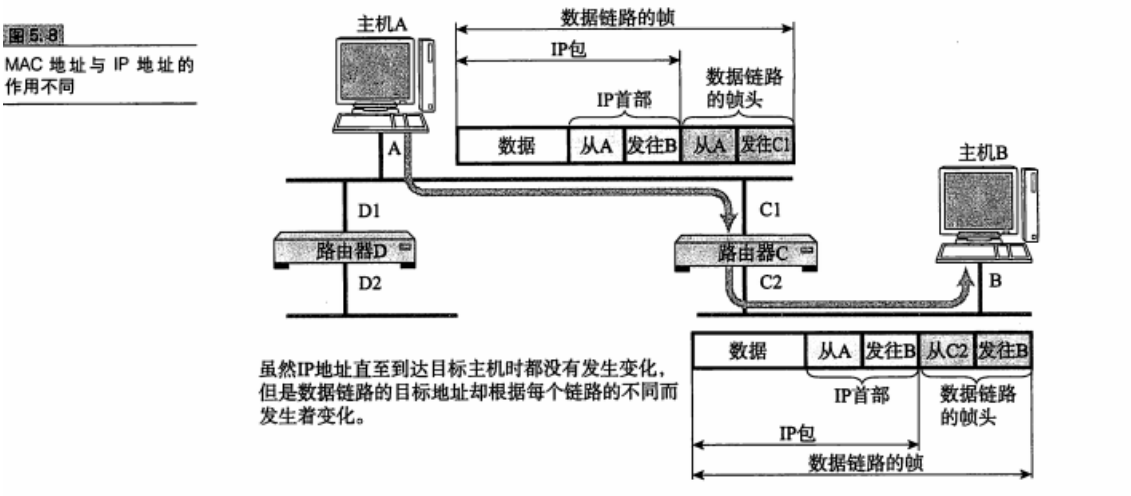
默认情况，WINDOWS系统的ARP缓存中的动态表项的老化时间为2分钟。一个ARP缓存表项若在2分钟内被用到（被设备引用获取相应MAC地址，或是表项被刷新时），**老化时间会重置为2分钟**，直到**最大生命期限10分钟**为止。若在老化时间2分钟内未用到，则此条目会从ARP缓存表中删除。超过10分钟的最大生命期限时，该表项同样会被移出ARP缓存表。并且通过另外一个ARP请求——ARP回应交换来获得新的对应关系。老化时间可设置更改。

- Q：只要知道了目的主机的MAC地址不就可以直接将数据传送到目的主机为什么还要IP?

  A：因为路由器将各个局域网分隔开来，数据包无法只借助MAC地址跨过路由器从一个局域网传送到另一个局域网。若是英特网是没有路由器连接的一个庞大的局域网的话，那是可以直接用MAC地址不用IP的。但是这样的话，交换机（网桥）将需要在未知该数据包的MAC地址需要往哪个接口发送时，向全世界的每台主机发送广播，这是不可想象的。并且还需要维护记录着全世界每台主机的MAC-接口的一张巨大的MAC表，这也是不可能的。所以，才需要将因特网通过IP地址划分成一个个小的局域网（事实上是先有的这些小的局域网，后来才有将这些局域网连接起来形成的因特网）。而局域网通过路由器互相连接。因此仅凭MAC地址是无法知道主机在因特网中的位置的。

- Q：为什么不直接在局域网链路上通过广播，将包发给目的主机，而要通过ARP解析获得目的主机的MAC地址呢?

  A：源主机每次发包都广播的话，链路上的非目的主机每次都会受到数据包，一是不安全，二是会增加其他主机的负担。并且，假如源主机在不同于目的主机的网段进行广播，会被源主机所处网段中所有路由器收到，正常该接收的路由会接收一份，其他的路由也会接收并且按照目的IP将包转发给该接收的路由，这样，该接收的路由会接收到好几份。但是事实上，路由器是不会转发MAC为广播地址的数据包的。而且，事实上，处于不同网段的主机也不会用源目MAC地址发送数据包，而是首先在网络层通过IP地址和路由表查找应发送的下一跳网关的IP。然后在链路层通过ARP查询下一条网关的对应接口的MAC，封装进数据包发送到下一跳网关的接口。这样就完成了IP数据报从一个网段到另一个网段，**数据包的传送过程中源IP和目标IP不会变，除非遇到NAT（SNAT或DNAT)，源MAC和目标MAC遇到网关会变。**



- ARP攻击

  https://blog.csdn.net/haodawei123/article/details/86480447

- RARP——从MAC获得IP

  一些嵌入式设备会遇到无法通过DHCP分配IP的情况，需要架设RARP服务器，将嵌入式设备连接服务器。设备向服务器发送自身MAC地址，由服务器返回分配给该设备的IP地址。

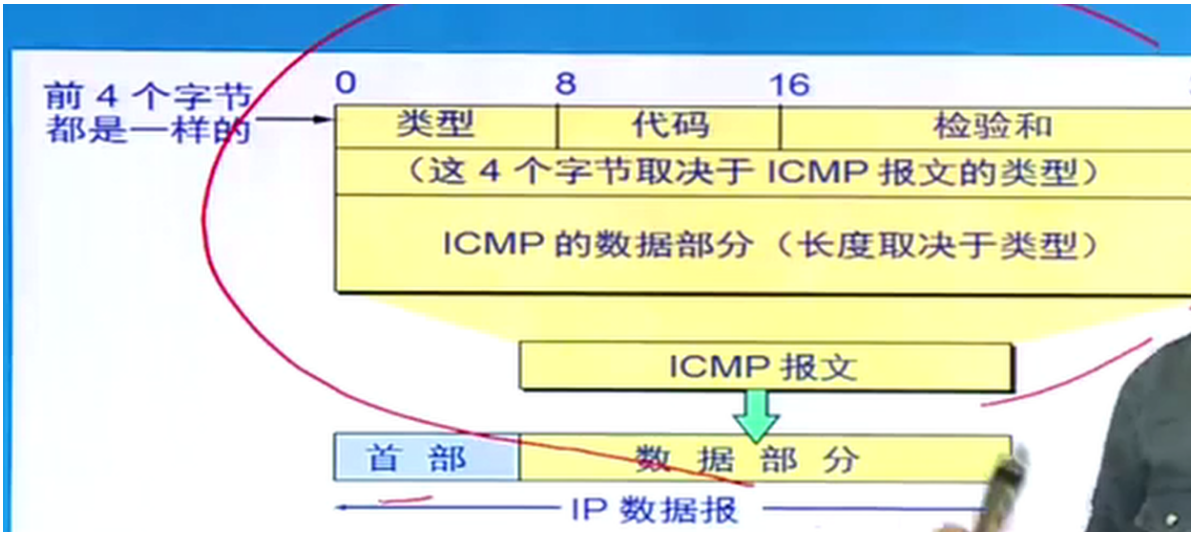
（6）ICMP协议报文

https://blog.csdn.net/chen1415886044/article/details/110356022

ICMP（Internet Control Message Protocol）因特网控制报文协议，ICMP报文封装在ip包里。它是一个对IP协议的补充协议。它是IPv4协议族中的一个子协议，用于IP主机、路由器之间传递**控制消息**。控制消息是在网络通不通、主机是否可达、路由是否可用等网络本身的消息。这些控制消息虽然不传输用户数据，但是对于用户数据的传递起着重要的作用。
ICMP协议与ARP协议不同，**ICMP靠IP协议来完成任务，所以ICMP报文中要封装IP头部。**它与传输层协议（如TCP和UDP）的目的不同，一般不用来在端系统之间传送数据，不被用户网络程序直接使用，除了Ping和Tracert这样的诊断程序。

- ICMP报文格式

  ICMP依赖于IP数据报，用的IP报头，ICMP数据区加上至少8B的报头，再加上至少20B的IP数据报报头封装成IP数据报。



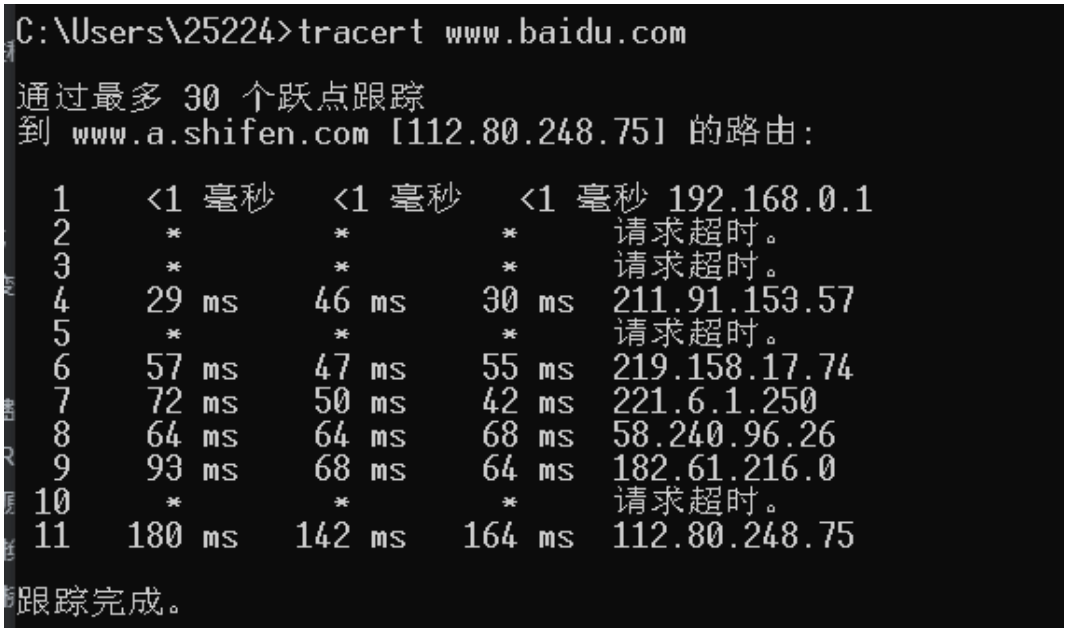
> Type：占8位
>
> Code：占8位
>
> Checksum：占16位
>
> Identifier：设置为ping 进程的进程ID。
>
> Sequence Number ： 每个发送出去的分组递增序列号。
>
> Type：8，Code：0：表示回显请求(ping请求)。
>
> Type：0，Code：0：表示回显应答(ping应答)
>
> 说明：**ICMP所有报文的前4个字节都是一样的**，但是剩下的其他字节则互不相同。

```
struct icmp {
        uint8_t icmp_type;
        uint8_t icmp_code;
        uint16_t icmp_cksum;
        uint16_t icmp_id;
        uint16_t icmp_seq;
    };
```

- 两类：ICMP**错误报告**报文、ICMP**查询**报文

- ICMP错误报告报文格式：从错误分片IP数据报取出报头，首位各加8字节作为数据区，再加上ICMP错误报告报头。

  `traceroute/tracert`

  > 可获取从源主机到目的主机所**经过的路由器**。应用程序将ICMP包中的IP报头中的TLL依次设为1,2,3，……，每经过一跳路由器时TTL减一，当TTL=0时路由器给源主机返回一个**ICMP超时报文**（类型11），这样源主机就可知TTL依次为1,2,3，……时的路由器的IP，并列出来。
  >
  > 
  > ```
  > C:\Users\25224>tracert www.baidu.com
  >
  > 通过最多 30 个跃点跟踪
  > 到 www.a.shifen.com [112.80.248.75] 的路由:
  >
  >   1    <1 毫秒    <1 毫秒    <1 毫秒 192.168.0.1
  >   2     *         *         *      请求超时。
  >   3     *         *         *      请求超时。
  >   4    29 ms     46 ms     30 ms  211.91.153.57
  >   5     *         *         *      请求超时。
  >   6    57 ms     47 ms     55 ms  219.158.17.74
  >   7    72 ms     50 ms     42 ms  221.6.1.250
  >   8    64 ms     64 ms     68 ms  58.240.96.26
  >   9    93 ms     68 ms     64 ms  182.61.216.0
  >  10     *         *         *      请求超时。
  >  11   180 ms    142 ms    164 ms  112.80.248.75
  >
  > 跟踪完成。
  > ```
  >
  > 序号指TTL
  >
  > 三个时间——每经过一个路有会都会发三个ICMP请求，这三个时间就是**三个请求分别响应的时间**
  >
  > *和请求超时——1、路由跳跃点禁PING。2、路由跳跃点不对TTL超时做响应处理，直接丢弃，不返回ICMP Echo Reply

- ICMP查询报文：

  > PING命令底层是ICMP，**回送请求**（**Echo Request**，类型8）和**回送应答**消息（**Echo Reply**，类型0）。
  >
  > 
  > ```
  > 正在 Ping www.a.shifen.com [112.80.248.75] 具有 32 字节的数据:
  > 来自 112.80.248.75 的回复: 字节=32 时间=40ms TTL=55
  > 来自 112.80.248.75 的回复: 字节=32 时间=57ms TTL=55
  > 来自 112.80.248.75 的回复: 字节=32 时间=46ms TTL=55
  > 来自 112.80.248.75 的回复: 字节=32 时间=41ms TTL=55
  >
  > 112.80.248.75 的 Ping 统计信息:
  >     数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),
  > 往返行程的估计时间(以毫秒为单位):
  >     最短 = 40ms, 最长 = 57ms, 平均 = 46ms
  > ```
  >
  > 三个时间是因为发送了很多ICMP Echo Request收到了很多ICMP Echo Reply，统计下来三个时间：【最短】【最长】【平均】

- ICMPv6

  IPv4中ICMP只是一个辅助支持作用，没有ICMPv4的话仍然可以IP通信。但是IPv6中若没有ICMPv6的话，就无法正常通信。

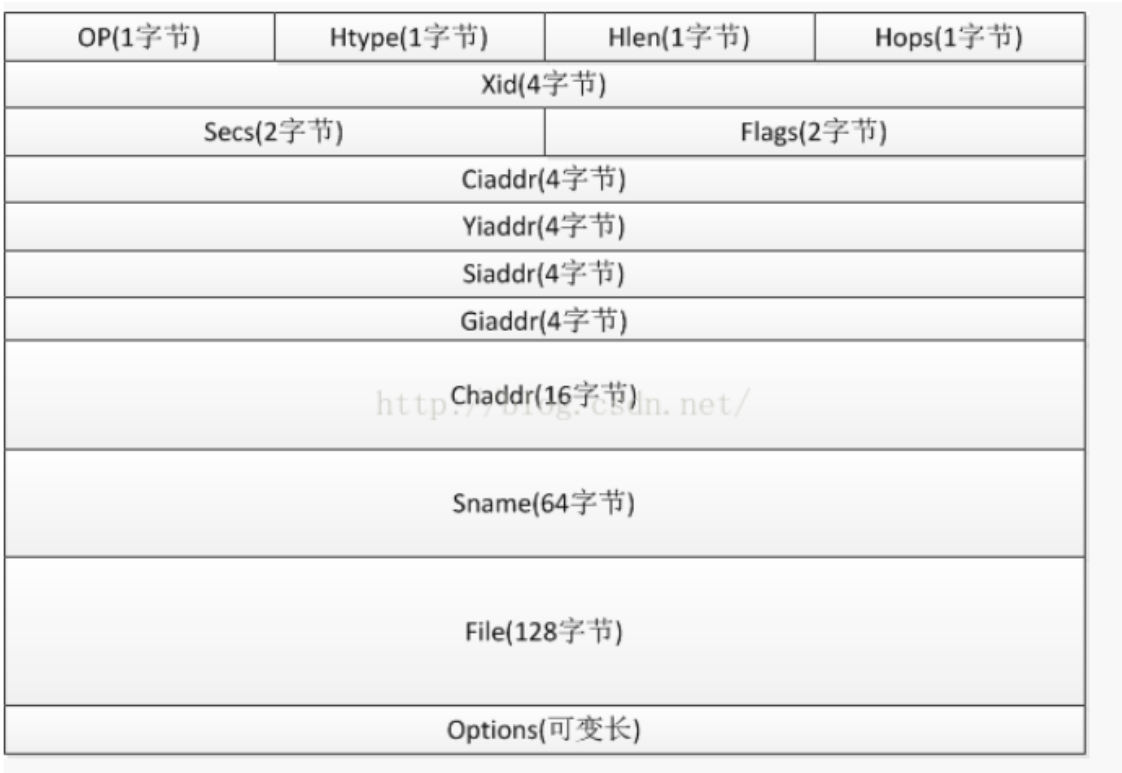
(7) DHCP——动态主机设置协议

- 连入局域网的设备都需要设置IP、子网掩码、默认网关（默认路由）、DNS，DHCP协议给接入链路的主机自动分配这些参数。
- 为了使设备即插即用，使用DHCP。在局域网内架设DHCP服务器，用来给新设备分配四个参数。新设备连入时可选择静态IP或DHCP。
- 网段中主机数较多时，为了避免DHCP服务器故障，一般在局域网中会假设两台及以上的DHCP服务器。但这样也会导致几个服务器同时给主机分配参数造成冲突。（可解决）主机较少时，一般就设置一台DHCP服务器，一般是由**路由器担任DHCP服务器**的角色。
- **为了检查分配的IP是否可用。DHCP服务器——分配IP前向分配的IP发送ICMP回送请求，确保没有ICMP回送应答。DHCP客户端——向分配的IP的主机发送ARP请求，确保没有ARP应答**
- 过程

  DHCP 协议是基于**UDP**（这也就是说在snort检测过程中，只能看UDP的包）之上的应用，dhcp使用udp携带报文，udp封装在ip数据包中发送。

  DHCP客户端对响应速度有较高要求，使用UDP，内部有保证可靠通信的机制。

- DHCP报文



  - DHCP一共有8种报文，分别为DHCP Discover、DHCP Offer、DHCP Request、DHCP ACK、DHCP NAK、DHCP Release、DHCP Decline、DHCP Inform。

    请求报文： DHCP Discover、DHCP Request、DHCP Release、DHCP Inform和DHCP Decline。
    应答报文： DHCP Offer、DHCP ACK和DHCP NAK。
  - DHCP报文格式基于BOOTP的报文格式
  - OP：报文的操作类型。分为请求报文和响应报文。1：请求报文，2：应答报文。即client送给server的封包，设为1，反之为2。

> Htype： DHCP客户端的MAC地址类型。 MAC地址类型其实是指明网络类型 ， Htype值为1时表示为最常见的以太网MAC地址类型。
> Hlen： DHCP客户端的MAC地址 长度。以太网MAC地址长度为6个字节，即以太网时Hlen值为6。
> Hops：DHCP报文经过的DHCP中继的数目，默认为0。DHCP请求报文每经过一个DHCP中继，该字段就会增加1。没有经过DHCP中继时值为0。(若数据包需经过router传送，每站加1，若在同一网内，为0。)
> Xid：客户端通过DHCP Discover报文发起一次IP地址请求时选择的随机数，相当于请求标识。用来标识一次IP地址请求过程。在一次请求中所有报文的Xid都是一样的。
> Secs：DHCP客户端从获取到IP地址或者续约过程开始到现在所消耗的时间，以秒为单位。在没有获得IP地址前该字段始终为0。(DHCP客户端开始DHCP请求后所经过的时间。目前尚未使用，固定为0。)
> Flags：标志位，只使用第0比特位，是广播应答标识位，用来标识DHCP服务器应答报文是采用单播还是广播发送，0表示采用单播发送方式，1表示采用广播发送方式。其余位 尚未使用 。(即 从0-15bits，最左1bit为1时表示server将以广播方式传送封包给client。)
> 【注意】在客户端正式分配了IP地址之前的第一次IP地址请求过程中，所有DHCP报文都是以广播方式发送的，包括客户端发送的DHCP Discover和DHCP Request报文，以及DHCP服务器发送的DHCP Offer、DHCP ACK和DHCP NAK报文。当然，如果是由DHCP中继器转的报文，则都是以单播方式发送的。另外，IP地址续约、IP地址释放的相关报文都是采用单播方式进行发送的。
> Ciaddr：DHCP客户端的IP地址。仅在DHCP服务器发送的ACK报文中显示，在其他报文中均显示0，因为在得到DHCP服务器确认前，DHCP客户端是还没有分配到IP地址的。只有客户端是Bound、Renew、Rebinding状态，并且能响应ARP请求时，才能被填充。
> Yiaddr：DHCP服务器分配给客户端的IP地址。仅在DHCP服务器发送的Offer和ACK报文中显示，其他报文中显示为0。
> Siaddr：下一个为DHCP客户端分配IP地址等信息的DHCP服务器IP地址。仅在DHCP Offer、DHCP ACK报文中显示，其他报文中显示为0。(用于bootstrap过程中的IP地址)
> Giaddr：DHCP客户端发出请求报文后经过的第一个DHCP中继的IP地址。如果没有经过DHCP中继， 则显示为0。(转发代理（网关）IP地址)
> Chaddr：DHCP客户端的MAC地址。在每个报文中都会显示对应DHCP客户端的MAC地址。
> Sname：为DHCP客户端分配IP地址的DHCP服务器名称（DNS域名格式）。在Offer和ACK报文中显示发送报文的DHCP服务器名称，其他报文显示为0。
> File：DHCP服务器为DHCP客户端指定的启动配置文件名称及路径信息。仅在DHCP Offer报文中显示，其他报文中显示为空。
> Options： 可选项字段，长度可变，格式为"代码+长度+数据"。

- DHCP中继代理——当一个局域网中的网段太多（比如学校、单位），给每个网段假设DHCP服务器将会是个繁重的任务。即使，路由器作为DHCP服务器，当网段过多时，对每个路由器维护可分配IP范围也是一件繁重的工作。因此，可只用一个DHCP服务器，给多个网段分配，实现多个网段的统一管理。

  在每个网段中不设置DHCP服务器，只设置一个DHCP中继代理，在多个网段上设置一个DHCP服务器即可。

  主机上的DHCP客户端通过广播发送DHCP请求包，被DHCP中继代理收到后，通过单播转发给DHCP服务器。DHCP服务器返回应答，通过单播发给DHCP中继代理，DHCP中继代理再通过单播发给主机。这样主机就知晓了DCHP服务器分配的IP。

（8）路径MTU

- 路径MTU——路径MTU是指一条因特网传输路径中，从源地址到目的地址所经过的“路径”上的所有IP跳的MTU的最小值（取最小值才能满足所有MTU）。或者从另外一个角度来看，就是无需进行分片处理就能穿过这条“路径”的MTU的最大值。
- 路径MTU发现方法——这是确定两个IP主机之间路径最大传输单元的技术，其目的就是为了避免IP分片。首先源地址将数据报的DF位置位（不论多长都不分片），在逐渐增大发送的数据报的大小——路径上任何需要将分组进行分片的设备都会将这种数据报丢弃并返回“数据报过大“的ICMP响应到源地址——这样源主机就”学习“到了无需分片就能通过这条路径的最大的最大传输单元。
- TraceRoute——打印出到达目的IP所经过的所有路由

  Traceroute是用来侦测主机到目的主机之间所经路由情况的重要工具。它的原理如下：它受到目的主机的IP后，首先给目的主机发送一个TTL=1的UDP数据包（每次送出的为3个40字节的包，包括源地址，目的地址和包发出的时间标签），而经过的第一个路由器收到这个数据包以后，就自动把TTL减1，而TTL变为0以后，路由器就把这个包给抛弃了，并同时产生 一个主机不可达的ICMP数据报给主机。主机收到这个数据报以后再发一个TTL=2的UDP数据报给目的主机，然后刺激第二个路由器给主机发ICMP数据报。如此往复直到到达目的主机。这样，traceroute就拿到了所有的路由器ip。
  Traceroute提取发送 ICMP TTL到期消息设备的IP地址并作域名解析。每次，Traceroute都打印出一系列数据,包括所经过的路由设备的域名及 IP地址,三个包每次来回所花时间。

（9）NAT和NAPT——网络地址转换协议

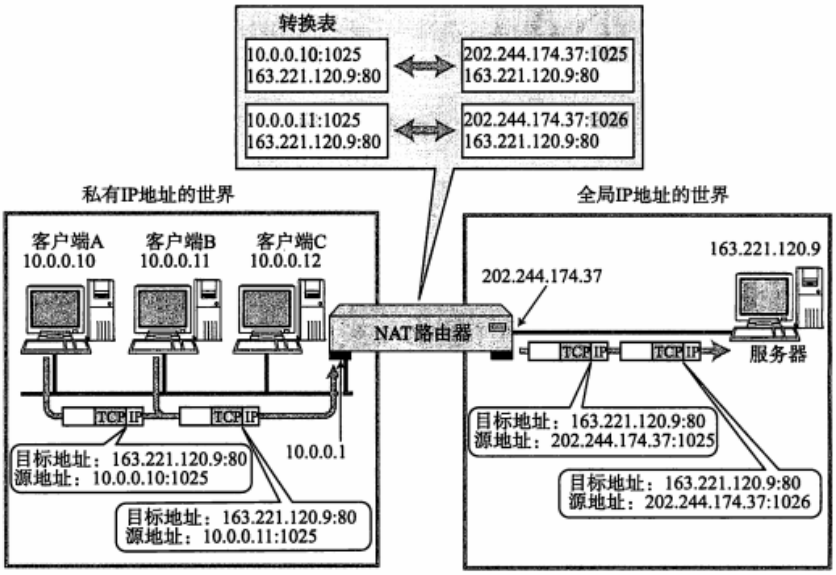
https://blog.csdn.net/q235990/article/details/88177202

- NAT——网络地址转换
- NAPT——网络地址端口转换

> 当“私有IP”世界中有多个主机同时想与“外部网络”通信，则通过NAT路由器就需要将这两个有着不同`私有IP`地址的主机对外分配成不同的`公网IP`。但是IP存在数量不够的情况，因此使用IP+端口号的方法标记不同的连接。
>
> 私有IP不同的主机，对外的共有IP相同，只是port号不同。
>
> TCP/IP和UCP/IP通信中，源目IP，源目port，协议号（TCP/UDP），这5个因素唯一标识了通信连接，任一因素不同都不是同一连接，由此区分了不同的主机。

（通俗的讲）它们都是地址转换，NAPT与NAT的区别在于 NAT是NAT是一对一转换，NAPT是多对一转换。通俗来说NAT是一个内部地址转换成一个外部地址进行通信的，而NAPT是多个内部地址使用同一地址不同端口转换成外部地址进行通信的。
简单来说：NAPT发送数据的时候会在源地址和目标地址上加上端口号（比如源地址：192.168.1.2:1010，目标地址：200.1.1.2:1020），回来的数据也是一样。

图 5.20
NAPT



*图中用"IP地址：端口号"标记。

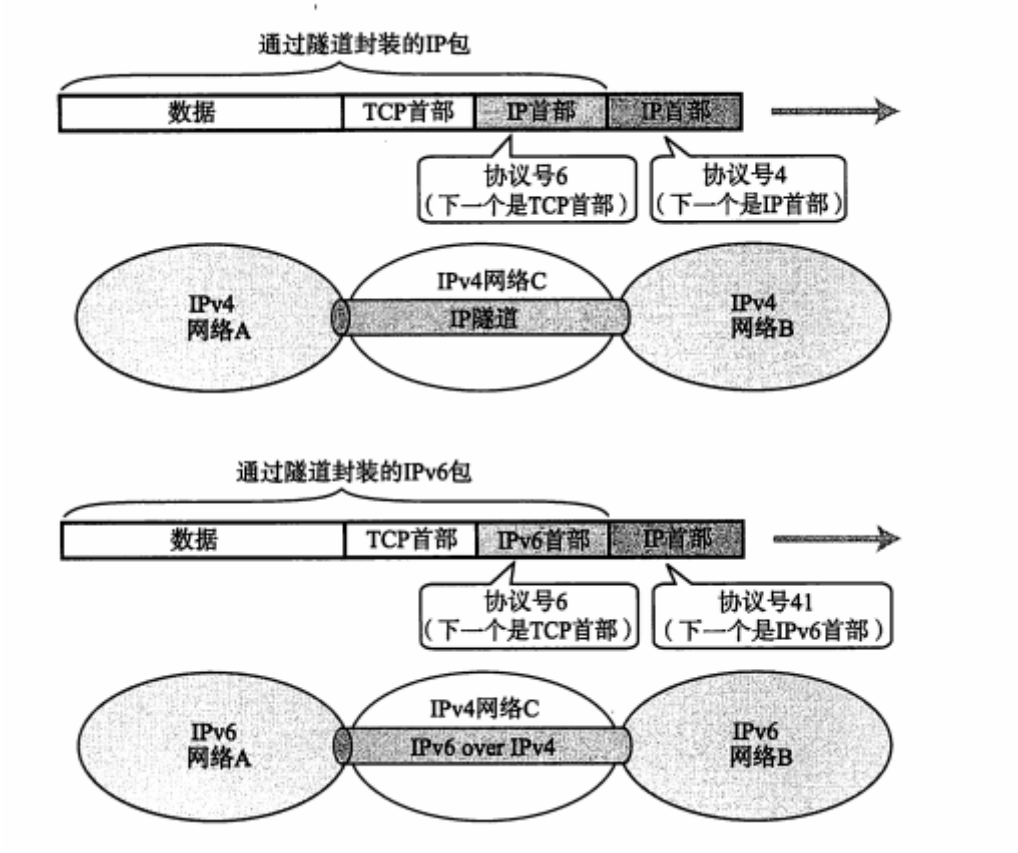
- NAT-PT（NAPT-PT）和IP隧道有什么不同?
- NAT穿越
- Mobile IP（MIP）

  允许移动节点（不限于手机）在不改变IP地址的情况下可以从一个子网移动到其他子网。是为满足移动节点在**移动中**保持其连接性而设计的网络服务，实现跨越不同网段的漫游功能。

（10）IP隧道

在IP报头后继续追加IP报头，以让IPv4数据报可以进入IPv6网络，IPv6进入IPv4的方法

> 构造一个既支持 IPv4 又支持 IPv6 的网络是一项极其庞大的工程。在这种网络环境中，由于其路由表的量有可能会涨到平常的两倍，所以会给网络管理员增加不小的负担，而在路由器进行两种协议都要支持的设置也是相当费劲的事情。骨干网上通常使用 IPv6 或 IPv4 进行传输。因此，那些不支持的路由器就可以采用 IP 隧道的技术转发数据包，而对应的 IP 地址也可以在一旁进行统一管理。这



（11）IP任播

将多个功能相同的服务器设成同一个IP，访问该IP时就近分配一个服务器提供连接。如DNS根服务器，不可能只有13台，而是将相同功能的作为同一个IP，且数量更多分布全球不同地点。

但是无法确保上一个包和下一个包发往同一个物理主机。对于无连接的UDP影响不大，但是对于需要建立连接的TCP，就会导致频繁的连接建立和断开。

（12）IP组播（IP多播）

基于UDP

---

3. 传输层——建立进程(端口)到进程（端口）的通信

在IP地址的基础上添加**端口**再加以封装

（1）端口（16bit）

- 分类

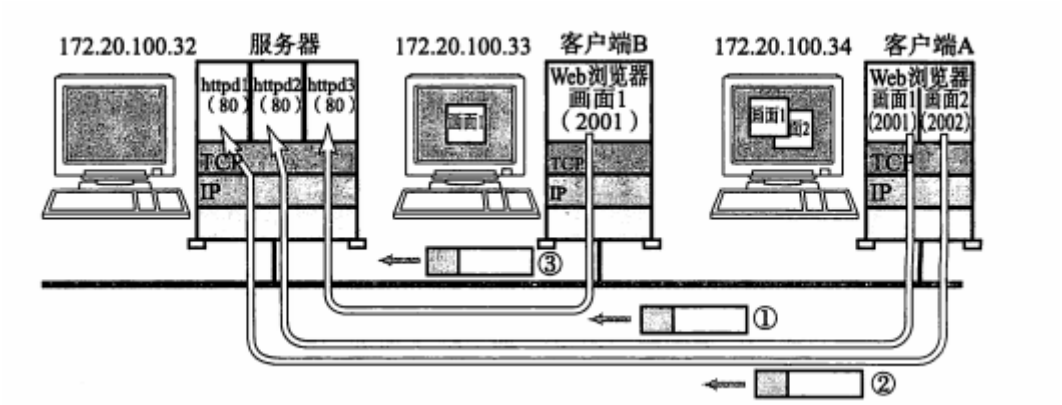
  `周知端口`（well-known-port），0~1023——只有“VIP应用程序”才能被分配的端口号，如DNS/TELNET/SSH等，这些应用程序的端口号一般固定，不允许被自行使用。

> 21端口：FTP 文件传输服务
> 22端口：SSH 远程连接服务
> 23端口：TELNET 终端仿真服务
> 25端口：SMTP 简单邮件传输服务
> 53端口：DNS 域名解析服务
> 80端口：HTTP 超文本传输服务
> 443端口：HTTPS 加密的超文本传输服务
>
> 3306端口：MYSQL数据库端口
> 5432端口：PostgreSQL数据库端口
> 6379端口：Redis数据库端口
> 8080端口：TCP服务端默认端口
> 8888端口：Nginx服务器的端口
> 9200端口：Elasticsearch服务器端口
> 27017端口：mongoDB数据库默认端口
> 22122端口：fastdfs服务器默认端口

注册端口，1024~49151——编写**服务器程序**时可使用的端口号。（***我们编程时候使用的端口***）

动态分配的端口，49152~65535——客户端进程运行时，由OS动态绑定的端口，不由用户操作。

- **通信连接由源目IP、源目端口、传输层协议5个参数共同决定**，有任一个参数不同都不是同一个连接。也就是说，对于目标端口来说，TCP的1024端口和UDP的1024端口，指定的是不同的进程。



①②中，客户端A的Web浏览器开了两个，***应用程序（可执行文件）虽然是同一个，但是开了两个，在系统中产生了两个进程***，OS给它们分配了两个不同的端口号2001和2002，因此目的端口虽然都是80，但是由于源端口不同，所以是不同的连接，从而通信不会混乱。

①③中，虽然源目端口相同，但是主机不同也即源IP不同，所以也不是同一个连接。

- 端口与进程/应用程序是一一对应的吗？？

> - 不是一一对应。一个进程可以绑定多个端口，但**一个端口只能对应一个进程**
> - 虽然理论上来说所有端口都是一样的，都可以使用。但是0~1023端口号作为周知端口/保留端口，约定俗成的对应着固定的应用程序。另外，如果随意使用端口号，容易与约定俗成冲突造成服务起不来
> - 所有的周知端口号和传输层协议是TCP或UDP无关，只要是同一个周知端口号，那么数据包就发送到应用层的同一个服务。但是有的服务并不是TCP和UDP都有的，比如80端口被用于HTTP，但是HTTP使用的是TCP，所以UDP的80端口未投入使用。

- 协议号与端口号的区别

协议号是在3层封装数据时，IP报头中用来指定数据包传到4层时使用TCP/UDP协议解析（传给TCP/UDP程序解析）。端口号是在4层封装数据时，在TCP/UDP报头中，用来指定数据包传到5层时使用什么应用层程序/协议解析数据。

对不同端口号的职能的处理是根据上层传输协议而进行的，不同的传输协议可使用相同的端口号，但是职能不同。

- 端口号的分配——两种方式

  > - 统一分配——静态分配方式，中央管理机构分配的端口号，是所有软件在设计时都要遵从的，是固定的、公认的，也即`周知端口`。TCP和UDP的周知端口号是各自独立的，同一个端口号，协议是TCP和协议是UDP时，端口号所指**服务**不同。
  > - 动态绑定——由主机上的操作系统负责分配。当应用程序运行时，产生某个进程，由OS负责分配一个可与其他进程区分开的端口号。进程终止时，端口号释放，可再由OS分配给其他进程。要想得知其他计算机的某个进程的端口号，需要发送请求报文询问，主机接收到报文才知道对端主机的某个进程的端口号。

(2) UDP

- 无连接（无逻辑连接），所以可以一对多，多对多，可承担广播、多播
- 不可靠（尽最大可能交付，接收端不返回确认报文）、
- 半双工（发送端和接收端交替发送）
- **面向数据报——不分段，不论应用层传下来的数据有多大，UDP都直接加上8B报头就送往发送缓存**
- 简单快速、耗费资源少。
- **适用于对丢包不敏感的领域：视频、直播、物联网等**
- UDP报文段格式

  伪首部12字节，首部8字节（2字节源端口、2字节目的端口、2字节UDP报文长度、2字节校验和checksum）

  > 1）源端口（2 字节）：发送方端口号
  >
  > 2）目的端口（2 字节 ）：接收方端口号
  >
  > 3）报文长度（2 字节）：UDP 用户数据报的总长度，以字节为单位。
  >
  > 4）校验和（2 字节）：检测 UDP 用户数据报在传输中是否有错，有错就丢弃。
  >
  > 5. 数据区：UDP 的数据部分如果不为偶数需要用 0 填补，就是说，如果数据长度为奇数，数据长度加“1”。

- 校验和

  同IP校验和只计算分片IP数据报报头不同，计算UDP报文段整个部分（**伪首部**、首部、数据区）

- 应用层传下来的数据到`发送缓存`，传输层从发送缓存取出数据，以UDP协议传输
- **socket=IP：port**

  端口号长度为2个字节，有效范围是0到65536。

  IP只指定主机，port指定进程。socket实质是进程间通信

  UDPsocket只有目的地址+目的端口

- UDP可在应用层通过应用程序自定义需要的功能，如：构建可靠连接

QUIC，*QUIC*（*Quic*k UDP Internet Connection）是谷歌制定的一种基于UDP的低时延的互联网传输层协议。比HTTP2.0更为先进。

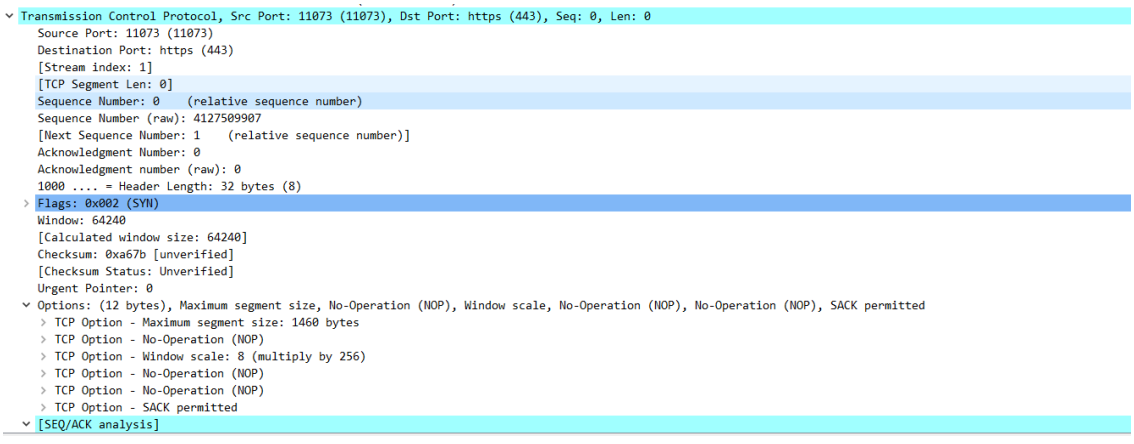
https://blog.csdn.net/mayifan_blog/article/details/86763626

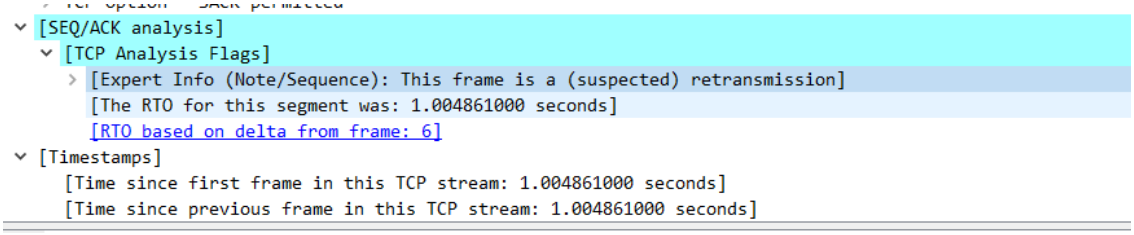
（3）TCP——最重要的是弄清TCP连接中主机的各个状态

- TCP报文段（segment ）格式

  最少20字节的首部

> 1. 源端口（16位）
>
> 2）目的端口（16位）
>
> 3） 序号seq（也叫序列号）（32位）
>
> 4）确认号ack（32位）
>
> 5） 数据偏移（首部长度）（4位）
>
> 6）保留位（6位）
>
> 7）控制位（6位）
>
> > SYN标志位——新建连接。仅在三次握手建立 TCP 连接时有效。
> >
> > FIN——释放连接
> >
> > ACK——确认标志
>
> 8）窗口（16位）
>
> 指发送本报文段的一方的接收窗口（而不是自己的发送窗口）。
>
> 此字段用来进行流量控制，**这个值是本机期望一次接收的字节数（告知发送方下次期望发送的数据的长度）**，即发送数据的窗口大小。告诉对方在不等待确认的情况下，可以发来多大的数据。这里表示的最大长度是2^16 - 1 = 65535，如需要使用更大的窗口大小，需要使用选项中的窗口扩大因子选项。
>
> 9）校验和（16位）
>
> 源主机和目的主机计算TCP报文段整个部分（伪首部、首部、数据区）计算校验和。
>
> 10）紧急指针（16位）
>
> 11）选项、填充字段
>
> 12）数据区（长度可变）
>
> 数据区MSS（最大报文段长度，不包括段头）：1460B（IP数据报最大长度1500B-IP数据报报头20B-TCP报文段段头最小20B）





上图是一个TCP重传，[TCP Analysis]中说明了该包是重传包，RTO（重传超时设定值）是1.004861000s，RTO值根据6号包设定因为是6号包的重传。

- TCP socket需要：源地址+源端口、目的地址+目的端口
- 有连接（虚拟的逻辑连接），一对一
- 全双工
- 面向**字节流**——怎么理解这个流？为了不让TCP segment到传输层时被IP分片，以及充分使用MTU。若应用层传下来的数据包过小时，则等待，等到几个包一起，达到长度**MSS**（TCP的数据区长度，最理想的MSS是——确保加上TCP报头、加上IP报头后形成的IP数据报长度不超过链路上的MTU的最小值。TCP MSS=1460B），再一起发送，这时候接收端可能一次收到好几个应用层的数据包。（反之，若应用层传下来的数据包过大，则会被TCP分成几段分别组成TCP segment传送到网络层。就是为了避免这种情况。）
- MSS大小确定——TCP连接建立时，客户端主机和服务器主机都会向对端发送请求，并在SYN报文的报头中写入自己接口适应的MSS大小。TCP通信时，选择两个较小的MSS。
- 可靠传输——超时重传机制和等待确认机制

  - **超时重传机制**——超时时间确定

    TCP重传也是以**MSS为单位**进行重传的

  - 超时时间——略大于RTT+误差时间。RTT——TCP报文段的往返时间。由于数据包经过不同的链路传送，而且网络环境不同造成RTT产生较大幅度摇摆，所以需要加上误差时间。
  - **等待确认**——序号seq（**通信双方协商，发送端随机产生（这样可避免黑客很容易猜到序列号而发起攻击）一个初始序号ISN（Initial Sequence Number），放入SYN包的序列号字段，将SYN包发给接收端，一旦ISN设置好，后续传输的每个TCP报文段都会根据数据载荷增加seq。在三次握手和四次挥手时，发送的SYN、FIN包作为一个字节在seq的基础上增加1。传递数据时的包，则seq每次增加转发过去的字节数**、确认号ack（确认应答超时、确认应答丢失）。

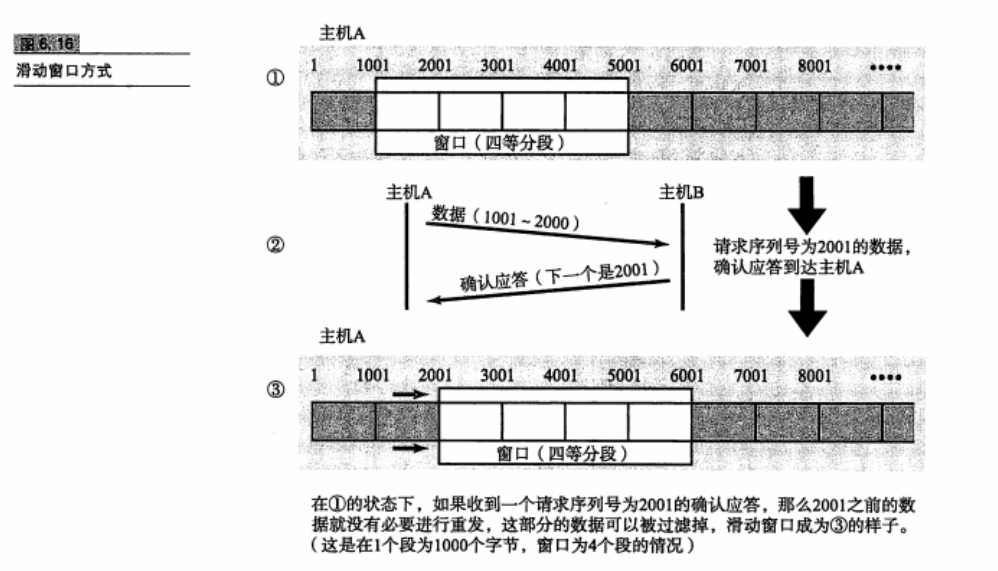
- `滑动窗口`，包括`发送端窗口`、`接收端窗口rwnd`。在实现上即`缓存`，是内存中的一块空间，**发送端**以3个指针维护四个区域（已确认、已发送未确认、未发送）。

  连续ARQ协议（***连续发送、累计确认***）——在内存中开辟空间作为“窗口”，用来缓存那些发送端已经发出但是尚未收到确认的包，当收到确认时就将对应的包从缓存中移除。

  - 为什么要用连续发送、累计确认机制？

    因为对超时重传机制和等待确认机制的改进。若是发一个包就等待确认，这样包的往返时间越长，通信性能就越低。所以，发送端就不停下等待确认，而是一直连续发送。接收端只对接受的连续包的最后一个包做确认。

    另外，对于发送端来说，就只需要按照seq顺序发送包即可，不论接收端是用什么顺序接收到包。哪怕是丢包了，发送端也会从接收到的ack号开始顺序发送，因为窗口里缓存了没接收到ack号所指定范围的包。



确认数据收到的是接收端，接收端发送一个ack=2001，那么接收端就是确认好了1~2000的包已经收到了。**接收端接收到的包由于网络情况（丢包重发、超时、重复接收等）接收到的包也不是按顺序的，接收缓存会接收seq不连续的很多包，但是只将迄今接收到的seq形成连续的包的序号+1作为ack发回。**如：接收到1000、2000、4000、5000、6000、8000的包，只会返回ack=2001。但是若下次收到了3000，就会直接返回ack=6001。若下次接受到的不是3000，则仍会返回ack=2001，当发送端收到同一个ack三次后，就会直接发送对应ack的包（快重传算法）。

接收端也不用针对收到的每个包都给发送端发送一个ACK，这样当发生“接收端接收到包，但ACK丢失”时，发送端可继续发送，只要发送端接收到后续的ACK就可以将之前的包都标记为“已收到”而从窗口中删除。

- **怎么实现连续发送、累计确认？——滑动窗口**

  滑动窗口是概念，实际实现是在发送端和接收端内存中开辟空间，以指针维护大小

  在确认应答策略中，对每一个发送的数据段，都要给一个ACK确认应答，收到ACK后再发送下一个数据段，这样做有一个比较大的缺点，就是性能比较差，尤其是数据往返的时间长的时候。

  （1）接收端将自己可以接收的缓冲区大小放入TCP首部中的“窗口大小”字段，通过ACK来通知发送端

  （2）窗口大小字段越大，说明网络的吞吐率越高

  （3）窗口大小指的是无需等待确认应答而可以继续发送数据的最大值，即就是说不需要接收端的应答，可以一次连续的发送数据

  （4）操作系统内核为了维护滑动窗口，需要开辟发送缓冲区，来记录当前还有那些数据没有应答，只有确认应答过的数据，才能从缓冲区删掉

  ps：发送缓冲区如果太大，就会有空间开销

  （5）接收端一旦发现自己的缓冲区快满了，就会将窗口大小设置成一个更小的值通知给发送端，发送端收到这个值后，就会减慢自己的发送速度

  （6）如果接收端发现自己的缓冲区满了，就会将窗口的大小设置为0，此时发送端将不再发送数据，但是需要定期发送一个窗口探测数据段，使接收端把窗口大小告诉发送端

  （7）在TCP的首部中，有一个16为窗口字段，此字段就是用来存放窗口大小信息的

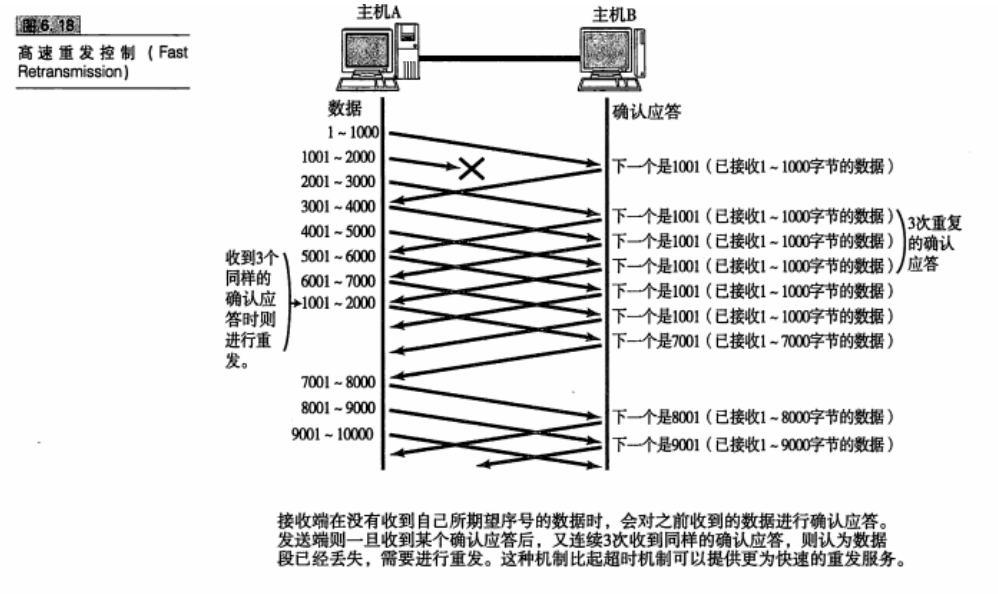
  原文链接：https://blog.csdn.net/borderhz/article/details/117284318

- 流量控制——使用滑动窗口还可以让接收端按照自身的`接收窗口rwnd`大小，通知发送端，**改变发送端的窗口大小**

  接收端在发给发送端的TCP报文中设置`窗口`字段，通知发送端改变窗口大小，从而发送端负责控制流量

- 拥塞控制——发送端维护`拥塞窗口cwnd`，发送端取rwnd和cwnd较小者作为发送窗口大小。**慢开始、快速重传、快速恢复**算法设置cwnd长度（字节）。

  > - **慢开始算法**（slow-start ）/机制——当主机开始发送数据时，如果立即把大量数据字节注入到网络，那么就有可能引起网络拥塞，因为现在并不清楚网络的负荷情况。因此，较好的方法是 先探测一下，即由小到大逐渐增大发送窗口，也就是说，由小到大逐渐增大拥塞窗口数值。通常在刚刚开始发送报文段时，先把拥塞窗口 cwnd 设置为1。而在每收到一个对新的报文段的确认后，把拥塞窗口增加至原来的**2倍**。用这样的方法逐步增大发送方的拥塞窗口 cwnd ，可以使分组注入到网络的速率更加合理。慢开始的“慢”并不是指cwnd的增长速率慢，而是指在TCP开始发送报文段时先设置cwnd=1进行“试探”。
  > - **拥塞避免算法**（congestion avoidance）/机制——了防止拥塞窗口cwnd增长过大引起网络拥塞，还需要设置一个慢开始门限ssthresh状态变量（如何设置ssthresh）。慢开始门限ssthresh的用法如下：
  >
  >   当 cwnd < ssthresh 时，使用上述的慢开始算法。
  >   当 cwnd > ssthresh 时，停止使用慢开始算法而改用拥塞避免算法。
  >   当 cwnd = ssthresh 时，既可使用慢开始算法，也可使用拥塞控制避免算法。
  >
  >   拥塞避免算法/机制：让拥塞窗口cwnd缓慢地增大，即每经过一个往返时间RTT就把发送方的拥塞窗口cwnd加1，而不是加倍。这样拥塞窗口cwnd按线性规律缓慢增长，比慢开始算法的拥塞窗口增长速率缓慢得多。
  > - 无论在慢开始阶段还是在拥塞避免阶段，只要发送方判断网络出现拥塞（其根据就是没有收到确认），就要把慢开始门限**ssthresh设置为出现拥塞时的发送方窗口值的一半**（但不能小于2）。**然后重新把拥塞窗口cwnd重新设置为1，执行慢开始算法。**这样做的目的就是要迅速减少主机发送到网络中的分组数，使得发生 拥塞的路由器有足够时间把队列中积压的分组处理完毕。
  > - **快重传算法/机制**
  >
  >   TCP 的超时重传，但是超时重传往往会带来许多微妙的问题，比如说：
  >
  >   当一个报文段丢失时，会等待一定的超时周期然后才重传分组，增加了端到端的时延。
  >   当一个报文段丢失时，在其等待超时的过程中，可能会出现这种情况：其后的报文段已经被接收端接收但却迟迟得不到确认，发送端会认为也丢失了，从而引起不必要的重传，既浪费资源也浪费时间。
  >   幸运的是，由于TCP采用的是累计确认机制，即当接收端收到比期望序号大的报文段时，便会重复发送最近一次确认的报文段的确认信号，我们称之为冗余ACK（duplicate ACK）。



> 不使用快重传的情况——如果发送方设置的超时计时器时限已到但还没有收到确认，那么很可能是网络出现了拥塞，致使报文段在网络中的某处被丢弃。这时，TCP马上把拥塞窗口 cwnd 减小到1，并执行慢开始算法，同时把慢开始门限值ssthresh减半。
>
> 使用快重传：
>
> - **快恢复**——与快重传同时使用
>
>   当发送方接收到连续的3个确认时，除了执行快重传立即发送接收端期待的包。因为发生了丢包的情况，所以推测发生了拥塞。所以要将cwnd改变，此时不使用慢开始算法（不将cwnd设为1），而是使用快恢复算法：**直接将cwnd设为ssthresh的一半**，然后执行加1增长。

- 总结拥塞控制

  慢开始和拥塞避免，就是cwnd从1开始，发生拥塞就将ssthresh减半然后重新开始。

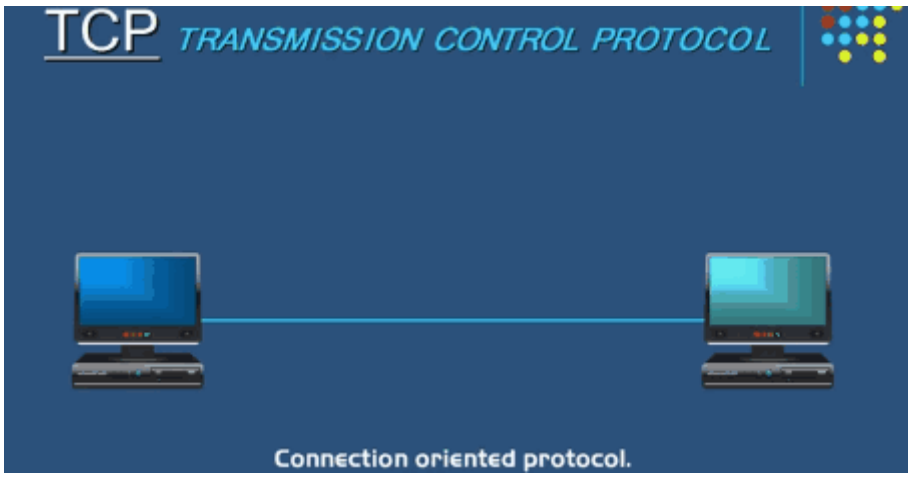
  快重传和快回复，就是发生接收包乱序时，不等超时计时结束立刻发送接收端期望的包。同时，cwnd不从1开始，而是从一半ssthresh开始。

- 现在，发送端窗口大小，由接收端窗口rwnd大小和拥塞窗口cwnd共同决定。发送窗口设置的比Min{rwnd, cwnd}略小

  > 发送方窗口的上限值 = Min [rwnd, cwnd]
  > 当rwnd < cwnd 时，是接收方的接收能力限制发送方窗口的最大值。
  > 当cwnd < rwnd 时，则是网络的拥塞限制发送方窗口的最大值。

- 连接建立与断开——三次握手、四次挥手

  https://zhuanlan.zhihu.com/p/78244069



> ### 为什么需要三次握手?
>
> TCP的连接因为是全双工的，也就是Client和Server两端，发送消息两个方向的连接都要建立成功。如果要保证双向连接都成功的话，三次通信是最少的次数了。大于三次的话，后面的次数通信就没有必要了，是在浪费资源。
>
> 二次的话，会怎么样，可不可以呢？答案是不可以，我们来看下，下面的场景。
>
> 在谈论这个之前，我们先要知道TCP是基于IP协议的，而IP协议是有路由的，IP协议不能够保证先发送的数据先到达，这当中依赖于IP协议底层的网络质量，以及Client与Server之间的路由跳数。
>
> Client在发送完Syn消息1，这里称作Syn1之后，假设因为网络原因，Syn1并没有到达Server端，这个时候Client端已经超时，Client之后重新发起SYN消息，这里称作Syn2。结果由于网络原因Syn2先到答Server，Server于是与Client基于Syn2建立了连接，结果没过多久Syn1又到达了Server,Server于是关掉了Syn2建立的那条连接，又重新建立了一条连接。对于Client来说新建立的这条连接是早就过时的，所以Client不会在这条连接上发送任何数据，这就导致了Server端长时间收不到数据，Client新的连接被断掉了。
>
> ### 三次握手失败了会怎么处理?
>
> 这里要看是在那个阶段失败的，Client在发送SYN之后没有收到ACK消息，Client会进行重传，第一次重传时间5.5-6s之间，第二次重传会是24s，不成功还会继续尝试，伯克利系统在超过75s之后，如果还是不成功，会放弃尝试连接。（备注：这里面的重传时间设置，与底层的定时器设置有过关系，可以参考TCP/IP详解卷1，这里不做详谈。）
>
> 如果Server没有收到最后的一次Ack消息，同样的原理，Server也会进行重传第二步的Syn+Ack消息。
>
> 四次挥手：MSL是Maximum Segment Lifetime英文的缩写，中文可以译为“报文最大生存时间”，他是任何报文在网络上存在的最长时间，超过这个时间报文将被丢弃
>
> MSL、TTL和RTT简介

- **关于 TIME_WAIT 状态的说明**

  客户端最后一次发送 ACK包后进入 TIME_WAIT 状态，而不是直接进入 CLOSED 状态关闭连接，这是为什么呢?

  TCP 是面向连接的传输方式，必须保证数据能够正确到达目标机器，不能丢失或出错，而网络是不稳定的，随时可能会毁坏数据，所以机器A每次向机器B发送数据包后，都要求机器B”确认“，回传ACK包，告诉机器A我收到了，这样机器A才能知道数据传送成功了。如果机器B没有回传ACK包，机器A会重新发送，直到机器B回传ACK包。

客户端最后一次向服务器回传ACK包时，有可能会因为网络问题导致服务器收不到，服务器会再次发送 FIN 包，如果这时客户端完全关闭了连接，那么服务器无论如何也收不到ACK包了，所以客户端需要等待片刻、确认对方收到ACK包后才能进入CLOSED状态。那么，要等待多久呢?

数据包在网络中是有生存时间的，超过这个时间还未到达目标主机就会被丢弃，并通知源主机。这称为报文最大生存时间（MSL，Maximum Segment Lifetime）。TIME_WAIT 要等待 2MSL 才会进入 CLOSED 状态。ACK 包到达服务器需要 MSL 时间，服务器重传 FIN 包也需要 MSL 时间，2MSL 是数据包往返的最大时间，如果 2MSL 后还未收到服务器重传的 FIN 包，就说明服务器已经收到了 ACK 包。

（4）UDP-Lite

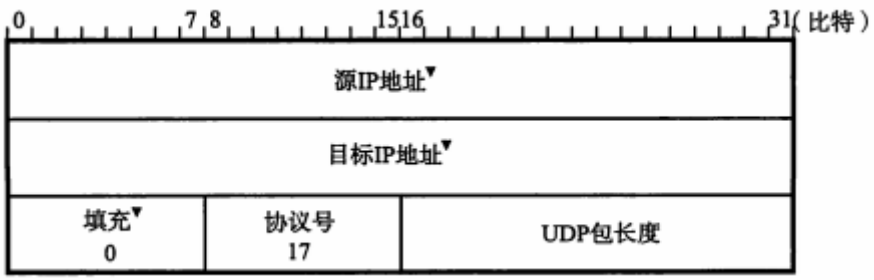
对UDP的改进。UDP若校验和出现错误，所有的包都会被丢弃。但是包中有些部分被破坏，影响并不大。所以UDP-Lite可以用ChecksumCoverage字段设置包中进行校验的范围，从而设定只对会引起严重后果的部分进行校验，如：源目端口、数据区中的源目IP。从而对发生错误不严重的包，不丢弃。

（5）伪首部

伪首部(pseudo header)，通常有TCP伪首部和UDP伪首部。在UDP/TCP伪首部中，包含32位源IP地址，32位目的IP地址，8位填充0，8位协议，16位TCP/UDP长度。通过伪首部的校验，UDP可以确定该数据报是不是发给本机的，通过首部协议字段，UDP可以确认有没有误传。

伪首部是一个虚拟的数据结构，其中的信息是从数据报所在IP分组头的分组头中提取的，**既不向下传送也不向上递交，而仅仅是为计算UDP和TCP的校验和，放在首部前，校验结束后清除。**

最终目的端根据伪报头和数据单元计算校验和以验证通信数据在传输过程中没有改变而且到达了正确的目的地址。




```
//伪头部:用于TCP/UDP计算CheckSum
//填充字段值来自IP层
typedef struct tag_pseudo_header
{
u_int32_t source_address;  //源IP地址
u_int32_t dest_address;  //目的IP地址
u_int8_t  placeholder;  //必须置0,用于填充对齐
u_int8_t  protocol;  //8为协议号(IPPROTO_TCP=6,IPPROTO_UDP=17)
u_int16_t tcplength;  //UDP/TCP头长度(不包含数据部分)
}PseudoHeader_S;
```

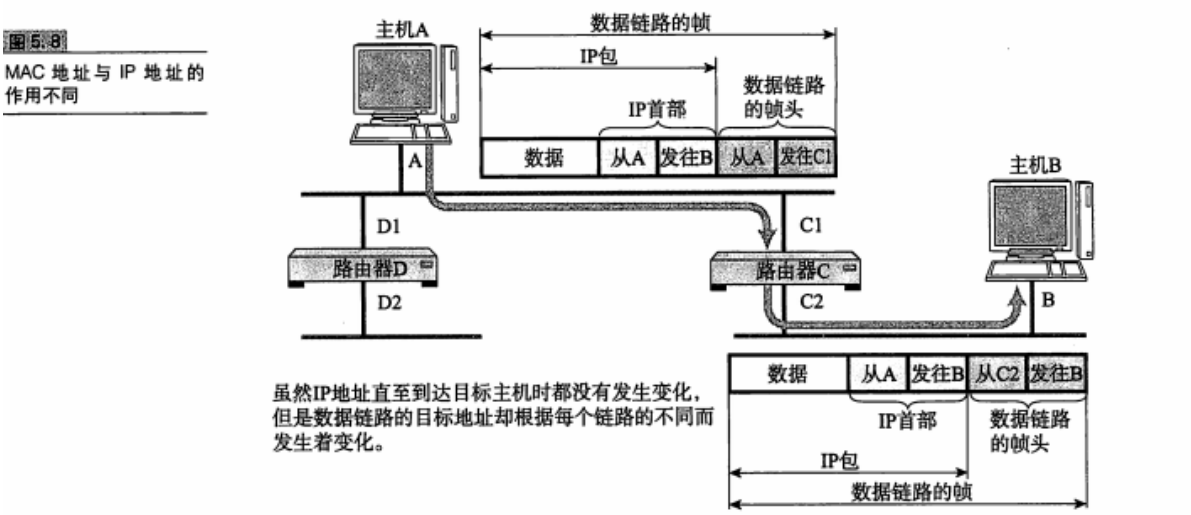
4. 数据链路层

（1） MAC帧格式

- 帧头16B：源MAC地址6B，目的MAC地址6B

（2）源目IP是不变的，但是源目MAC跨过网关时一直在变化

IP数据报转发时，源目IP始终不变，这样在跨过路由经过一个个局域网时，可以标记最后在哪个网段停下。但是IP数据报外面封装的源目MAC在跨过网关（路由器）时，路由器会把MAC帧拆开，将源MAC地址修改为自身的MAC地址，将目的MAC地址改写成通过ARP获取的下一跳网关（还没到目的IP指定的网段时）的MAC地址，封装成新的MAC帧发送到下一跳网关。到达最终网段时，通过目的IP和ARP将MAC帧发送给目的主机。



5. 总结

总结

（1） 详解一次完整的数据包传输过程 -- 层层递进

> 传输层封装**源端口、目的端口**
>
> 网络层封装**源IP、目的IP**
>
> 数据链路层封装**源MAC、目的MAC**
>
> 层层封装与解封的过程类似于将信件（数据）放入层层信封，信件就是从应用层传下来的数据区，每一层拆开信封后拿出内容都重填“寄件人”（源MAC地址）和“收件人”（目的MAC地址）。某一层的设备只解封到相应层，不会 拆开上一层对应的“内部”的信封，拆开后从信封封面读取该层所需信息。重新封装也是拿一个新的当前层对应的信封，将手里当前的信封装入，并在信封上写上新的发件人、收件人等信息。

(2) MAC表、ARP表、路由表

路由表、ARP表、MAC表2

> 路由表——网络层，选择最短路径，判断下一跳网关是哪个。（用来在**局域网间**寻找下一跳要传送到的局域网的网关）。分为静态路由表和动态路由表，前者由网络管理员设置好，后者是路由器基于路由协议学习。
>
> ARP表——数据链路层，IP→MAC（得到下一跳网关的MAC），将目的MAC地址封装进MAC数据帧。（**局域网内部**起作用，获取目标主机的MAC地址）
>
> MAC表——用于交换机，MAC地址→接口，用于交换机确定目的MAC对应的主机要通过那个接口发送（接口与MAC绑定），实现单播，由交换机负责将数据包发送到某一个设备，而不是将数据包在局域网内广播到所有设备。这样连接在同一个交换机上的设备之间是无法截取发送给另一个设备的数据包的。交换机动态学习的MAC地址默认只有300S的有效期，如果300S内记录的MAC地址没有通信，则会删除此记录。

(3) 路由器工作原理

(4) 路由器和交换机的区别

> 集线器工作在第一层物理层，路由器工作在第三层网络层，交换机工作在第二层数据链路层
>
> 路由器和交换机的主要工作如下：
> **路由器：寻址，转发（依靠 IP 地址）**
> **交换机：过滤，转发（依靠 MAC 地址）**
>
> **每一个路由器与其之下连接的设备，其实构成一个局域网**
> 交换机工作在路由器之下，就是也就是**交换机工作在局域网内**
> 交换机用于**局域网内网的数据转发**
> 路由器用于**连接局域网和外网**
>
> 路由器发送数据——广播
>
> 交换机——单播：一个接口接一个host，若MAC表中有目的MAC对应的交换机接口则直接通过相应接口单播转发到目的host；若MAC表中没有目的MAC对应的接口，则通过所有**非接收接口**进行广播，目的主机接收后返回响应，交换机会将目的主机和接口的对应关系记录到MAC表中，下次发送时就直接向对应接口发送实现单播。
>
> 路由器内集成了交换机的功能，主机与路由器相连也可以实现数据转发，但是不足之处是：
> 可扩展的接口不如交换机多
> 交换机通常由硬件加速转发，路由器主要靠软件寻址，速度慢。

(5) 集线器、网桥、交换机

(6) 各层封装

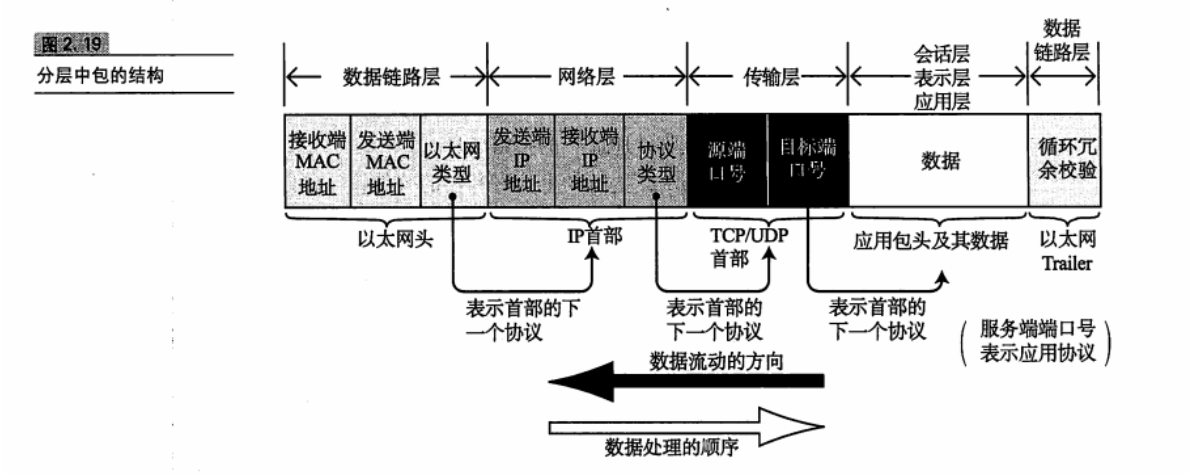
- 协议数据单元在应用层、表示层和会话层被称做**数据**(Data)，在传输层被称做**分段**(Segment)，在网络层被称做**包**(Packet)，在数据链路层被称做**帧**(Frame)，在物理层被称做**比特**(Bit)

  TCP segment——应用层数据+端口号（源目端口）

  IP datagram——TCP segment+IP地址（源目IP）

  MAC frame——IP datagram+MAC地址（目的MAC地址）

- 数据包的封装



以太网类型——2B，协议类型——1B，MAC帧循环冗余校验——4B

- 应用层协议端口号固定，常用协议端口号如下：

  常用服务 协议　端口号
  POP3　TCP 110
  IMAP　TCP 143
  SMTP　TCP 25
  Telnet TCP 23
  终端服务 TCP 3389
  PPTP　TCP 1723
  HTTP　TCP 80
  FTP(控制)　TCP 21
  FTP(数据)　TCP 20
  HTTPS　TCP 443
  NTP UDP　123
  RADIUS　UDP　1645
  DHCP　UDP　67
  DNS　UDP　53
  DNS　TCP 53
  SNMP UDP　161
  ipsec　UDP　500
  TFTP　UDP　69
  L2TP　UDP　1701

（7）IP分片与TCP分组

最大传输单元（MTU——每层协议数据区最大长度）。数据大小 > MTU，就需要被分片。

我们的数据经过七层协议的过程中就像包粽子一样，每过一层就需要增加数据的大小，ip层的上层是传输层（tcp/udp ，tcp 的头部为20Byte，udp头部字节是8B）ip层自己的头部需要占20字节，ip层的MTU = 1500Byte - 20 =1480B。ip层传输的数据包最大是1480B, 超过1480Byte的数据，都需要被ip层分片，在达到目的前会自己重组。

又因为，tcp是可靠传输协议，通过超时与重传机制，来保证收到的数据是完整的。因为tcp是可靠传输协议，如果要传输的数据大于 1480 - 20(tcp头部) =1460Byte时，在ip层被分片，而ip层分片会导致，**如果其中的某一个分片丢失，因为tcp层不知道哪个ip数据片丢失，所以就需要重传整个数据段**，这样就造成了很大空间和时间资源的浪费。为了解决这个问题，就需要避免TCP报文被IP分片。所以在传输层就分组，即将TCP报文段分组，数据部分超过MSS（**最大报文长度——TCP数据区最大长度**）的就要分

组，这样`分组TCP报文段`到达IP时就不会被分片了。就需要确保到达IP时的数据区长度最长为MTU=1500B-20B（数据链路层的MTU=1500B，所以IP数据报最长为1500B）。所以TCP MSS=MTU-20B-20B=1460B。

由于udp是不可靠传输的，所以ip分片主要是为了upd服务的，所以就有了网上的1500 - 20(ip头部) - 8(udp头部) > 1472 的说法，把1472作为ip分片的标准。
