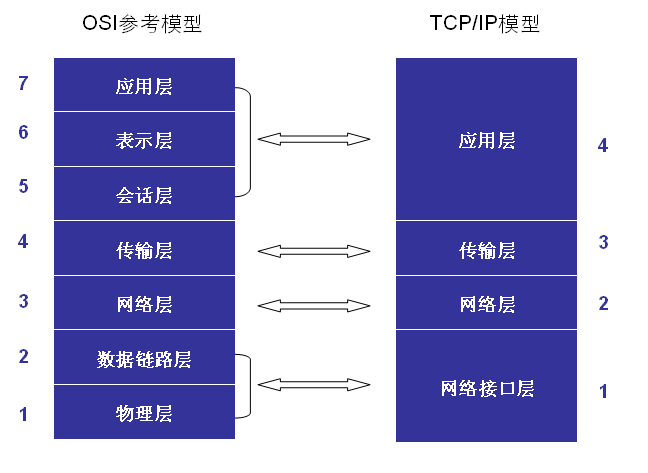
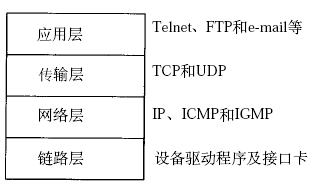
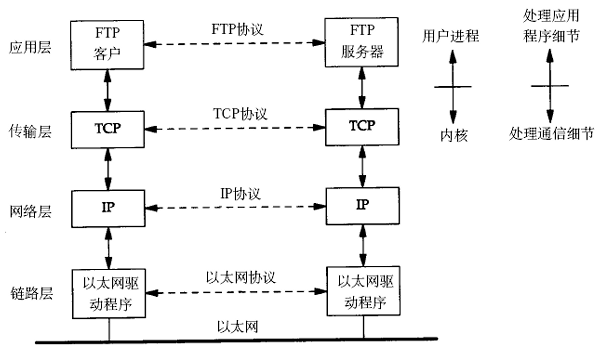
1. **OSI七层模型和TCP四层模型**





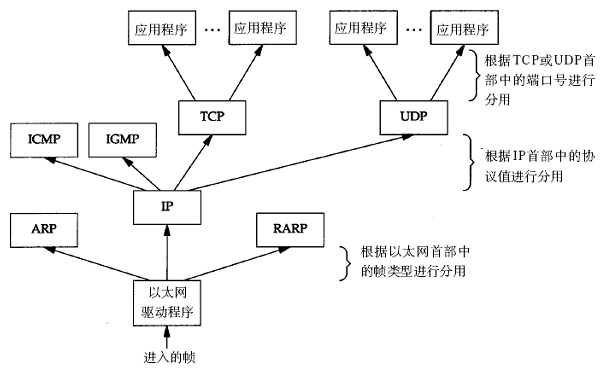
1. **TCP/IP通信过程**



1. **multiplexing过程**

虽然IP、ARP和RARP数据报都需要以太网驱动程序来封装成帧，但是从功能上划分，ARP和RARP属于链路层，IP属于网络层。

虽然ICMP、IGMP、TCP、UDP的数据都需要IP协议来封装成数据报，但是从功能上划分，ICMP、IGMP与IP同属于网络层，TCP和UDP属于传输层。



1. **MAC地址**

MAC地址也叫物理地址、硬件地址，由网络设备制造商生产时烧录在网卡，是全球唯一的。IP地址是32位，MAC地址是48位（如：40-8D-5C-9F-B2-BA）。

网络中数据帧从一个路由节点到下一跳路由节点，是依靠MAC地址而非IP地址进行传输的，因此数据传输时需要知道传输路径上的节点的MAC地址，通过ARP协议获取某IP主机的MAC地址。

1. **ARP协议**

地址解析协议，可以实现**通过IP地址获得对应主机的物理地址（MAC地址）**。

ARP协议要求通信的主机双方必须在**同一个物理网段**。

ARP协议具体通信过程：

①每台主机都会在自己的ARP高速缓冲区建立一个ARP列表，用于存储IP地址与MAC地址的对应关系。当源主机需要发数据包到目标主机时，先检查自己的ARP列表中是否存在该IP地址对应的MAC地址，如果存在则直接将数据包发送到该MAC地址。

②如果源主机没有目标主机的MAC地址时就需要向本地网段发一个ARP请求（ARPrequest）的广播包，用于查询目标主机对应的MAC地址。此ARP请求数据包里包括源主机的IP地址、NAC地址及目标地址的IP地址。

③网段中所有主机收到这个ARP请求后，看数据包中包含的IP地址是否与自己的IP地址一致，如果不一样就忽略；如果一致就把发送端的MAC地址和IP地址添加到自己的ARP列表中，如果该对应关系已经存在，则将其覆盖掉，然后给源主机发送一个ARP响应包(ARPreply)（单播），响应包包括自己的IP地址和MAC地址。

④源主机收到目标主机的响应后，将目标主机的IP地址的和MAC地址添加到自己的ARP列表中，并用此消息开始数据的传输。

⑤如果源主机一直没有收到ARP响应包，则提示ARP查询失败。

1. **RARP协议**

RARP协议**通过MAC地址获取IP地址**。

当设置一台新机器时，其RARP客户端需要向路由器上的RARP服务器请求相应的IP地址。假设在路由表中已经设置了一个记录，RARP服务器将返回IP地址给机器，此机器就会存储起来以便日后使用。

1. **ARP攻击**

ARP协议是建立在信任局域网内所有节点的基础上的，效率很高但是不安全。ARP协议是无状态的协议，它不会检查自己是否发过请求包，也不知道自己是否发过请求包，它也不管是否合法的应答，都会接受并缓存。

假设局域网中有A、B、C三台主机，主机C进行ARP欺骗。

**假冒ARPreply包**：A询问B的MAC地址，此时C冒充B对A进行非法回复，而且是大量的，所以A就会误信C是B，这样主机C就劫持了主机A发送给主机B的信息。

**假冒ARPrequest包**：C直接冒充网关，此时主机C就会不停的发送ARP欺骗广播，大声说我的IP是192.168.0.1，我的硬件地址是mac-c，此时局域网内的所有主机都被欺骗，更改自己的缓存表，此时C就会监听整个局域网发送给互联网的数据。

1. **ICMP协议**

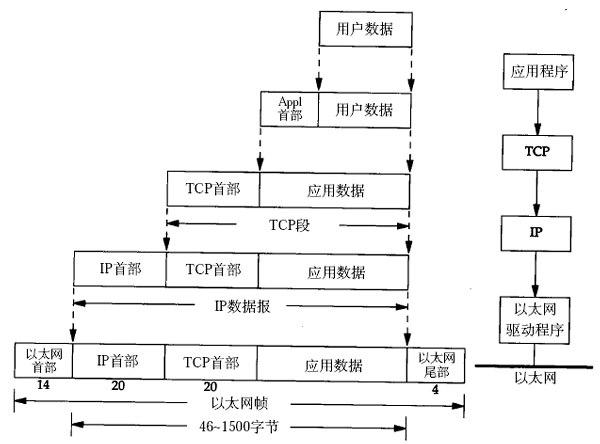
ICMP（Internet Control Message Protocol）（Internet控制报文协议）是一个网络层协议，一个新搭建好的网络往往需要先进行一个简单的测试，来验证网络是否畅通。但是IP协议并不提供可靠的传输，如果包丢了，IP协议并不能通知传输层是否丢包以及丢包原因。所以就需要ICMP来完成这样的功能。

ICMP协议功能：确认IP包是否成功到达目标地址；通知在发送过程中IP包被丢弃的原因。

ICMP是基于IP协议工作的，但是它并不是传输层的功能，仍然把它归结为网络层协议。

ping命令和traceroute命令都是基于ICMP协议的。

1. **数据包封装**

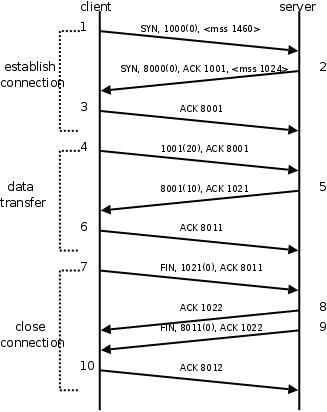


不同的协议对数据包有不同的称谓，在传输层叫做段（segment），在网络层叫数据报（datagram），在链路层叫做帧（frame）

以太网帧中的数据长度规定最小46字节，最大1500字节，ARP和RARP数据包的长度不够46字节，要在后面补填充位。最大值1500称为以太网的最大传输单元（MTU），不同的网络类型有不同的MTU，如果数据包长度大于1500则需要对数据包进行分片。

1500字节指的是有效载荷大小，不包括以太网首部和以太网尾部。

1. **TCP时序图**



**三次握手过程**：

1. 客户端发出段1，**SYN位标识连接请求**，**发送序号**是1000，这个序号在网络通信中用作临时的地址，每发一个数据字节，这个序号就要加1，这样在接收端就可以根据序号排出数据包的正确顺序，也可以发现丢包的情况，另外规定**SYN和FIN位也要占一个字节**，这次虽然没发数据，但是由于发了SYN位，因此下次再发送应该用序号1001。**mss表示最大段尺寸**，如果一个段太大，封装成帧后超过了链路层的最大帧长度，就必须在IP层分片，为了避免这种情况，客户端声明自己的最大段尺寸，建议服务器发来的段不要超过这个长度。
2. 服务器发出段2，也带有SYN位，同时置ACK位表示确认，确认序号是1001，表示“我接收到序号1000及其以前所有的段，请你下次发送序号为1001的段”，也就是应答了客户端的连接请求，同时也给客户端发出一个连接请求，同时声明最大段尺寸mms为1024。
3. 客户端发出段3，对服务器的连接请求进行应答，确认序号是8001。

在这个过程中，客户端和服务器分别给对方发了连接请求，也应答了对方的连接请求，其中服务器的请求和应答在一个段中发出，因此一共有三个段用于建立连接，称为三方握手（three-way-handshake）。在建立连接的同时，双方协商了一些信息，例如双方发送序号的初始值、最大段尺寸等。

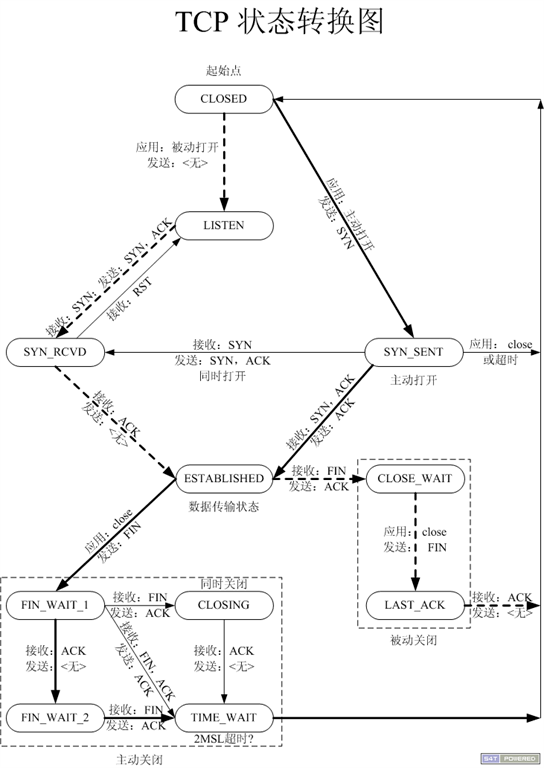
**数据传输的过程：**

1. 客户端发出段4，包含从序号1001开始的20个字节数据。
2. 服务器发出段5，确认序号为1021，对序号为1001-1020的数据表示确认收到，同时请求发送序号1021开始的数据，服务器在应答的同时也向客户端发送从序号8001开始的10个字节数据，这称为piggyback。
3. 客户端发出段6，对服务器发来的序号为8001-8010的数据表示确认收到，请求发送序号8011开始的数据。

**四次挥手过程：**

TCP连接是全双工的，因此每个方向都必须单独进行关闭，原则是当一方完成它的数据发送任务后就能发送一个FIN来终止这个方向的连接，此时这一方不能发送数据，但是收到FIN的一方还能发送数据，直到收到FIN的一方发送数据完毕，此时它也会发送一个FIN来终止自身的数据发送。首先关闭的一方将执行主动关闭，另一方将执行被动关闭。

1. 客户端发出段7，FIN位表示关闭连接的请求，用来关闭客户端到服务器的数据传送。
2. 服务器收到这个FIN，立即回发一个ACK，应答客户端的关闭连接请求。
3. 服务器发出段9，其中也包含FIN位，向客户端发送关闭连接请求。
4. 客户端收到服务器发出的FIN位，立即回发ACK报文，应答服务器的关闭连接请求。
5. **TCP状态转换图**



**CLOSED**: 这个没什么好说的了，表示初始状态。

**LISTEN**: 这个也是非常容易理解的一个状态，表示服务器端的某个SOCKET处于监听状态，可以接受连接了。（服务端执行了listen函数）

**SYN\_RCVD**: 这个状态表示接受到了SYN报文，在正常情况下，这个状态是服务器端的SOCKET在建立TCP连接时的三次

握手会话过程中的一个中间状态，很短暂，基本上用netstat你是很难看到这种状态的，除非你特意写了一个客户

端测试程序，故意将三次TCP握手过程中最后一个ACK报文不予发送。因此这种状态时，当收到客户端的ACK报文

后，它会进入到ESTABLISHED状态。

**SYN\_SENT**: 这个状态与SYN\_RCVD遥想呼应，当客户端SOCKET执行CONNECT连接时，它首先发送SYN报文，因此也随即

它会进入到了SYN\_SENT状态，并等待服务端的发送三次握手中的第2个报文。SYN\_SENT状态表示客户端已发送SYN

报文。

**ESTABLISHED**：这个容易理解了，表示连接已经建立了。

**FIN\_WAIT\_1**: 这个状态要好好解释一下，其实FIN\_WAIT\_1和FIN\_WAIT\_2状态的真正含义都是表示等待对方的FIN报

文。而这两种状态的区别是：FIN\_WAIT\_1状态实际上是当SOCKET在ESTABLISHED状态时，它想主动关闭连接，向

对方发送了FIN报文，此时该SOCKET即进入到FIN\_WAIT\_1状态。而当对方回应ACK报文后，则进入到FIN\_WAIT\_2状

态，当然在实际的正常情况下，无论对方何种情况下，都应该马上回应ACK报文，所以FIN\_WAIT\_1状态一般是比较

难见到的，而FIN\_WAIT\_2状态还有时常常可以用netstat看到。

**FIN\_WAIT\_2**：上面已经详细解释了这种状态，实际上FIN\_WAIT\_2状态下的SOCKET，表示半连接，也即有一方要求

close连接，但另外还告诉对方，我暂时还有点数据需要传送给你，稍后再关闭连接。

**CLOSING**: 这种状态比较特殊，实际情况中应该是很少见，属于一种比较罕见的例外状态。正常情况下，当你发送

FIN报文后，按理来说是应该先收到（或同时收到）对方的ACK报文，再收到对方的FIN报文。但是CLOSING状态表

示你发送FIN报文后，并没有收到对方的ACK报文，反而却也收到了对方的FIN报文。什么情况下会出现此种情况

呢？其实细想一下，也不难得出结论：那就是如果双方几乎在同时close一个SOCKET的话，那么就出现了双方同时

发送FIN报文的情况，也即会出现CLOSING状态，表示双方都正在关闭SOCKET连接。

**TIME\_WAIT:** 表示收到了对方的FIN报文，并发送出了ACK报文，就等2MSL后即可回到CLOSED可用状态了。如果

FIN\_WAIT\_1状态下，收到了对方同时带FIN标志和ACK标志的报文时，可以直接进入到TIME\_WAIT状态，而无须经过

FIN\_WAIT\_2状态。

**CLOSE\_WAIT**: 这种状态的含义其实是表示在等待关闭。怎么理解呢？当对方close一个SOCKET后发送FIN报文给自

己，你系统毫无疑问地会回应一个ACK报文给对方，此时则进入到CLOSE\_WAIT状态。接下来呢，实际上你真正需要

考虑的事情是察看你是否还有数据发送给对方，如果没有的话，那么你也就可以close这个SOCKET，发送FIN报文

给对方，也即关闭连接。所以你在CLOSE\_WAIT状态下，需要完成的事情是等待你去关闭连接。

**LAST\_ACK:** 这个状态还是比较容易好理解的，它是被动关闭一方在发送FIN报文后，最后等待对方的ACK报文。当收

到ACK报文后，也即可以进入到CLOSED可用状态了。

* TimeWait状态需要等待2MSL时间后才能回到CLOSED状态。

这是为了确保客户端最后一次发送的ACK成功到达对端。当最后一次发送的ACK在发送过程中丢失，对端等待接收ACK会超时，对端会重新发送FIN信令，此时如果客户端处于TimeWait的状态就能收到FIN信令并重新回复ACK。

其中MSL指的是Max Segment Lifetime(报文最大生存时间)，它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。

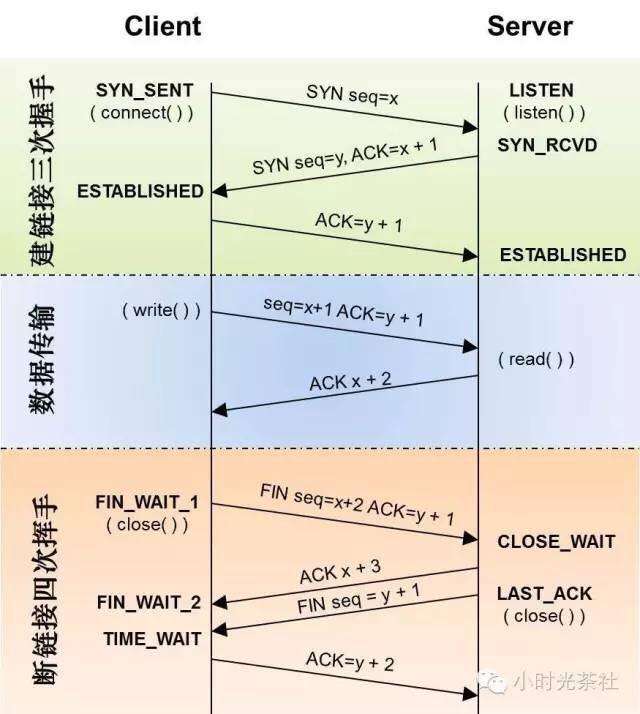
* 为什么需要三次握手。

这是为了防止已过期的连接再次传到被连接的主机。

如何采用两次握手，那么假设客户端为主机A，服务端为主机B，主机A向主机B发送SYN请求，但是网络超时未抵达，此时主机A重新向主机B发送SYN请求并递达，主机B回复ACK信令，连接建立成功，完成数据传输并关闭连接。

如果此时主机A第一次发送的SYN请求递达了主机B，那么主机B回复ACK并处于ESTABLISHED态，此时主机B会一直等待主机A发送过来数据，造成服务器资源的浪费。

下面的TCP流程图，标注出了每个流程的TCP状态。



1. **TCP流量控制（滑动窗口）**

接收端处理数据的速度是有限的，如果发送方的速度太快，接收端缓冲区满了，这个时候如果继续发送数据，就会导致丢包等一系列连锁反应（虽然TCP机制中丢包会重发，但是这样浪费了巨大的网络资源和系统资源）。

TCP发送方向接收方发送数据后，接收方会回复ACK，在ACK中携带了接收窗口大小字段win，表示接收方的接收缓存区还能存储多少数据，这样发送方就可以调整发送数据的大小，保证发送接收的速率匹配。

当接收方回复的ACK中，接收窗口大小字段win为0，表示接收方的接收数据缓存区已经满了，发送方就停止发送数据。那么发送方何时再重启数据发送呢，因为发送方不发送数据也就无法获得接收方回复的ACK报文，也就无法获得接收方的可用接收缓冲区的大小。解决的方法是发送方此时开启一个定时器，每隔一段时间就发个测试报文询问接收方，打听是否可以继续发送数据了，接收方会回复此时窗口大小，如果窗口大小大于0则可以继续发送数据，如果窗口大小还是0，则发送方再次刷新启动定时器。

1. **TCP半连接**

当TCP链接中A发送FIN请求关闭，另一段B回应ACK后，B没有立即发送FIN给A时，A方处在半链接状态，此时A可以接收B发送的数据，但是A已不能再向B发送数据。这就是半连接状态。半连接状态中，A处于FIN\_WAIT2状态，B处于CLOSE\_WAIT状态。