[面试·网络] TCP/IP（四）：TCP 与 UDP 协议简介

从本章开始，我们开始介绍最重要的传输层。传输层位于 OSI 七层模型的第四层（由下往上）。顾名思义，传输层的主要作用是**实现应用程序之间的通信**。网络层主要是保证不同数据链路下数据的可达性，至于如何传输数据则是由传输层负责。

传输层协议简介

常见的传输层协议主要有 TCP 协议和 UDP 协议。TCP 协议是面向有连接的协议，也就是说在使用 TCP 协议传输数据之前一定要在发送方和接收方之间建立连接。一般情况下建立连接需要三步，关闭连接需要四步。

建立 TCP 连接后，由于有数据重传、流量控制等功能，TCP 协议能够正确处理丢包问题，保证接收方能够收到数据，与此同时还能够有效利用网络带宽。然而 TCP 协议中定义了很多复杂的规范，因此效率不如 UDP 协议，不适合实时的视频和音频传输。

UDP 协议是面向无连接的协议，它只会把数据传递给接收端，但是不会关注接收端是否真的收到了数据。但是这种特性反而适合多播，实时的视频和音频传输。因为个别数据包的丢失并不会影响视频和音频的整体效果。

IP 协议中的两大关键要素是源 IP 地址和目标 IP 地址。而刚刚我们说过，传输层的主要作用是**实现应用程序之间的通信**。因此传输层的协议中新增了三个要素：源端口号，目标端口号和协议号。通过这五个信息，可以唯一识别一个通信。

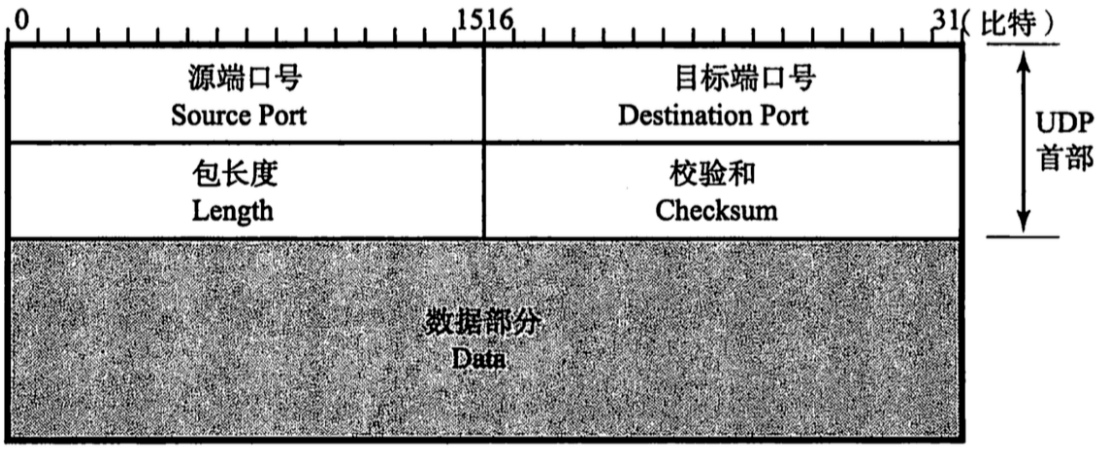
不同的端口用于区分同一台主机上不同的应用程序。假设你打开了两个浏览器，浏览器 A 发出的请求不会被浏览器 B 接收，这就是因为 A 和 B 具有不同的端口。

协议号用于区分使用的是 TCP 还是 UDP。因此相同两台主机上，相同的两个进程之间的通信，在分别使用 TCP 协议和 UDP 协议时也可以被正确的区分开来。

用一句话来概括就是：“源 IP 地址，目标 IP 地址，源端口号，目标端口号和协议号”这五个信息只要有一个不同，都被认为是不同的通信。

UDP 首部

UDP 协议最大的特点就是简单，它的首部如下图所示：



UPD 首部

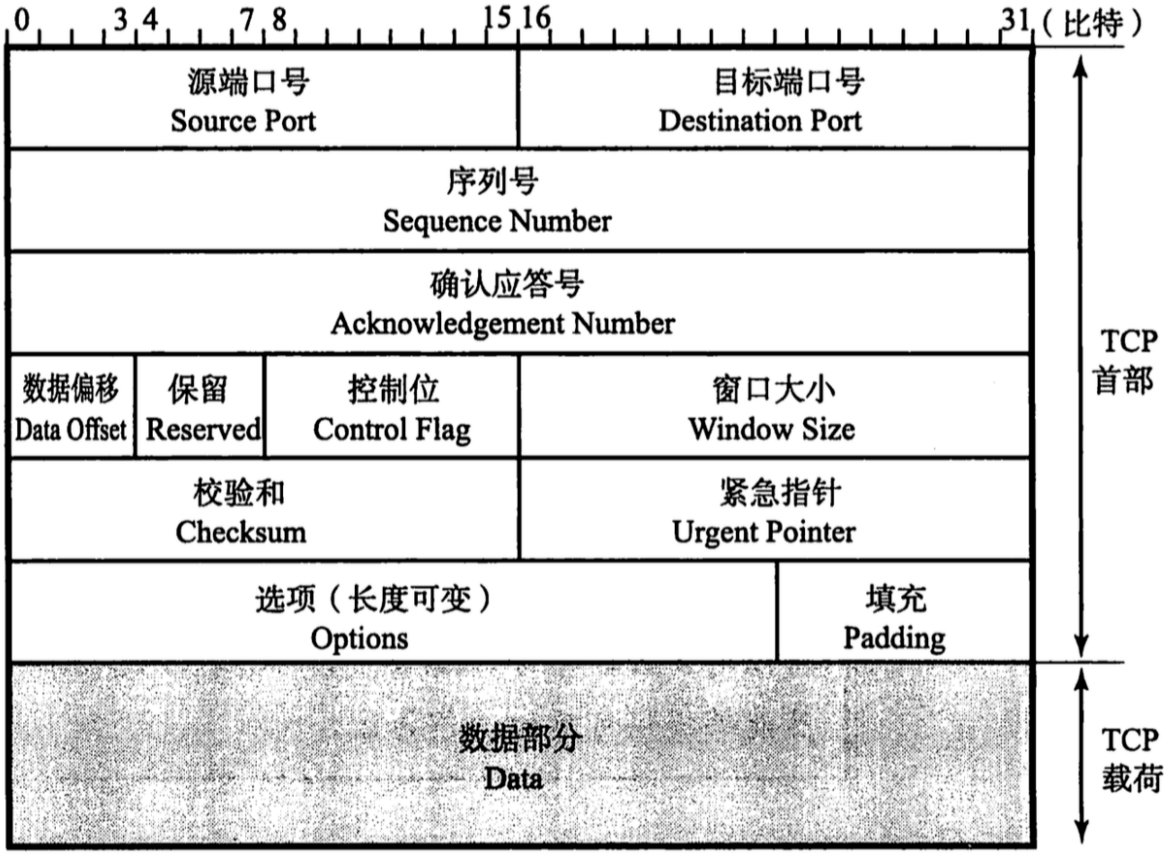
包长度表示 UDP 首部的长度和 UPD 数据长度之和。

校验和用来判断数据在传输过程中是否损坏。计算这个校验和的时候，不仅考虑源端口号和目标端口号，还要考虑 IP 首部中的源 IP 地址，目标 IP 地址和协议号（这些又称为 UDP 伪首部）。这是因为以上五个要素用于识别通信时缺一不可，如果校验和只考虑端口号，那么另外三个要素收到破坏时，应用就无法得知。这有可能导致不该收到包的应用收到了包，改收到包的应用反而没有收到。

这个概念同样适用于即将介绍的 TCP 首部。

TCP 首部

和 UDP 首部相比，TCP 首部要复杂得多。解析这个首部的时间也相应的会增加，这是导致 TCP 连接的效率低于 UDP 的原因之一。



TCP 首部

其中某些关键字段解释如下：

- 序列号：它表示发送数据的位置，假设当前的序列号为 s，发送数据长度为 l，则下次发送数据时的序列号为 s + l。在建立连接时通常由计算机生成一个随机数作为序列号的初始值。

- 确认应答号：它等于下一次应该接收到的数据的序列号。假设发送端的序列号为 s，发送数据的长度为 l，那么接收端返回的确认应答号也是 s + l。发送端接收到这个确认应答后，可以认为这个位置以前所有的数据都已被正常接收。

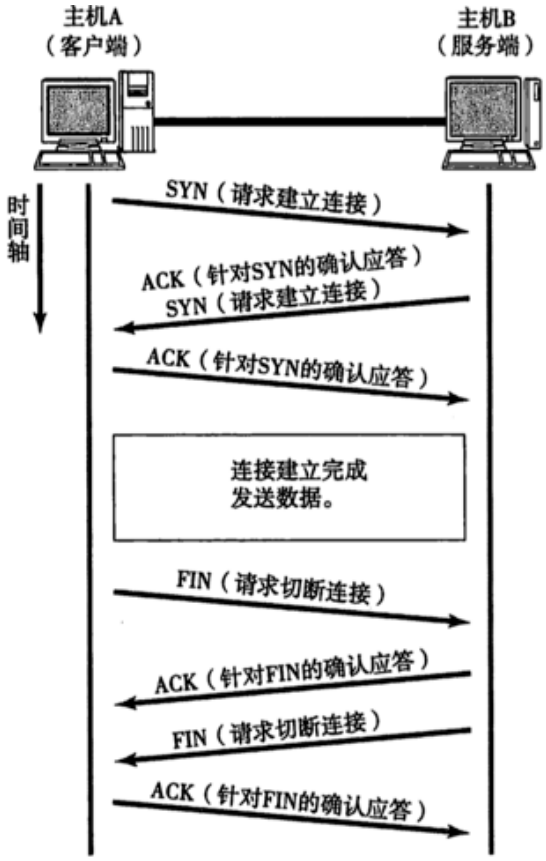
- 数据偏移：TCP 首部的长度，单位为 4 字节。如果没有可选字段，那么这里的值就是 5。表示 TCP 首部的长度为 20 字节。

- 控制位：改字段长度为 8 比特，分别有 8 个控制标志。依次是 CWR，ECE，URG，ACK，PSH，RST，SYN 和 FIN。在后续的文章中你会陆续接触到其中的某些控制位。

- 窗口大小：用于表示从应答号开始能够接受多少个 8 位字节。如果窗口大小为 0，可以发送窗口探测。
- 紧急指针：尽在 URG 控制位为 1 时有效。表示紧急数据的末尾在 TCP 数据部分中的位置。通常在暂时中断通信时使用（比如输入 Ctrl + C）。

TCP 握手

TCP 是面向有连接的协议，连接在每次通信前被建立，通信结束后被关闭。了解连接建立和关闭的过程通常是考察的重点。连接的建立和关闭过程可以用一张图来表示：



TCP 连接建立和关闭

通常情况下，我们认为客户端首先发起连接。

三次握手建立连接

这个过程可以用以下三句形象的对话表示：

1. （客户端）：我要建立连接了。
2. （服务端）：我知道你要建立连接了，我这边没有问题。
3. （客户端）：我知道你知道我要建立连接了，接下来我们就正式开始通信。

为什么是三次握手

根据一般的思路，我们可能会觉得只要两次握手就可以了，第三步确认看似是多余的。那么 TCP 协议为什么还要费力不讨好的加上这一次握手呢？

这是因为在网络请求中，我们应该时刻记住：“网络是不可靠的，数据包是可能丢失的”。假设没有第三次确认，客户端向服务端发送了 SYN，请求建立连接。由于延迟，服务端没有及时收到这个包。于是客户端重新发送一个 SYN 包。回忆一下介绍 TCP 首部时提到的序列号，这两个包的序列号显然是相同的。

假设服务端接收到了第二个 SYN 包，建立了通信，一段时间后通信结束，连接被关闭。这时候最初被发送的 SYN 包刚刚抵达服务端，服务端又会发送一次 ACK 确认。由于两次握手就建立了连接，此时的服务端就会建立一个新的连接，然而客户端觉得自己并没有请求建立连接，所以就不会向服务端发送数据。从而导致服务端建立了一个空的连接，白白浪费资源。

在三次握手的情况下，服务端直到收到客户端的应答后才会建立连接。因此在上述情况下，客户端会接受到一个相同的 ACK 包，这时候它会抛弃这个数据包，不会和服务端进行第三次握手，因此避免了服务端建立空的连接。

ACK 确认包丢失怎么办

三次握手其实解决了第二步的数据包丢失问题。那么第三步的 ACK 确认丢失后，TCP 协议是如何处理的呢？

按照 TCP 协议处理丢包的一般方法，服务端会重新向客户端发送数据包，直至收到 ACK 确认为止。但实际上这种做法有可能遭到 SYN 泛洪攻击。所谓的泛洪攻击，是指发送方伪造多个 IP 地址，模拟三次握手的过程。当服务器返回 ACK 后，攻击方故意不确认，从而使得服务器不断重发 ACK。由于服务器长时间处于半连接状态，最后消耗过

多的 CPU 和内存资源导致死机。

正确处理方法是服务端发送 RST 报文，进入 CLOSE 状态。这个 RST 数据包的 TCP 首部中，控制位中的 RST 位被设置为 1。这表示连接信息全部被初始化，原有的 TCP 通信不能继续进行。客户端如果还想重新建立 TCP 连接，就必须重新开始第一次握手。

四次握手关闭连接

这个过程可以用以下四句形象的对话表示：

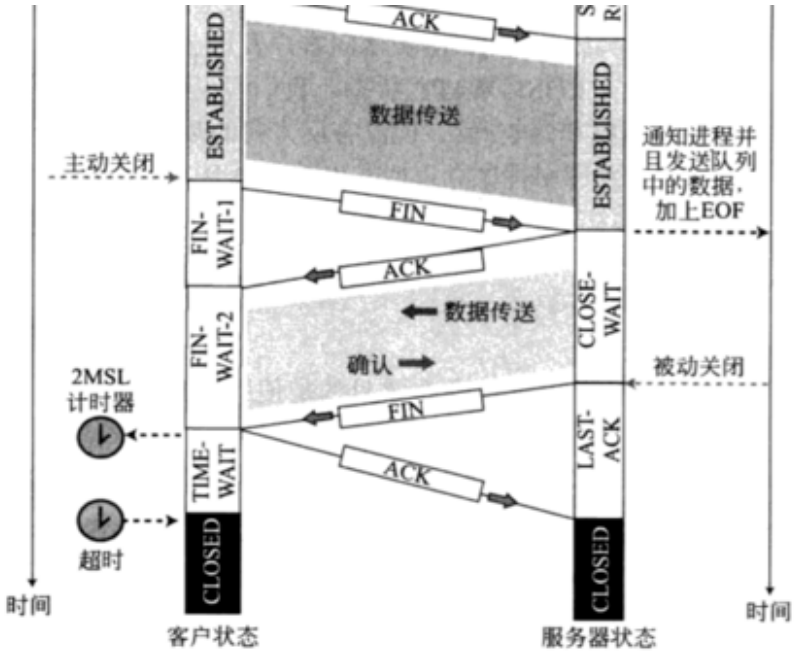
1. （客户端）：我要关闭连接了。
2. （服务端）：你那边的连接可以关闭了。
3. （服务端）：我这边也要关闭连接了。
4. （客户端）：你那边的连接可以关闭了。

由于连接是双向的，所以双方都要主动关闭自己这一侧的连接。

关闭连接的最后一个 ACK 丢失怎么办

实际上，在第三步中，客户端收到 FIN 包时，它会设置一个计时器，等待相当长的一段时间。如果客户端返回的 ACK 丢失，那么服务端还会重发 FIN 并重置计时器。假设在计时器失效前服务器重发的 FIN 包没有到达客户端，客户端就会进入 CLOSE 状态，从而导致服务端永远无法收到 ACK 确认，也就无法关闭连接。

示意图如下：


ACK
ESTABLISHED
数据传送
ESTABLISHED
主动关闭
通知进程并
且发送队列
中的数据，
加上EOF
FIN
FIN-
WAIT-1
ACK
数据传送
CLOSE-
WAIT
FIN-
WAIT-2
确认
被动关闭
2MSL
计时器
FIN
LAST-
ACK
TIME-
WAIT
ACK
超时
CLOSED
CLOSED
时间
客户状态
服务器状态
时间

TCP 关闭连接