计算机网络

1、OSI七层协议和TCP/IP四层协议

OSI七层协议：

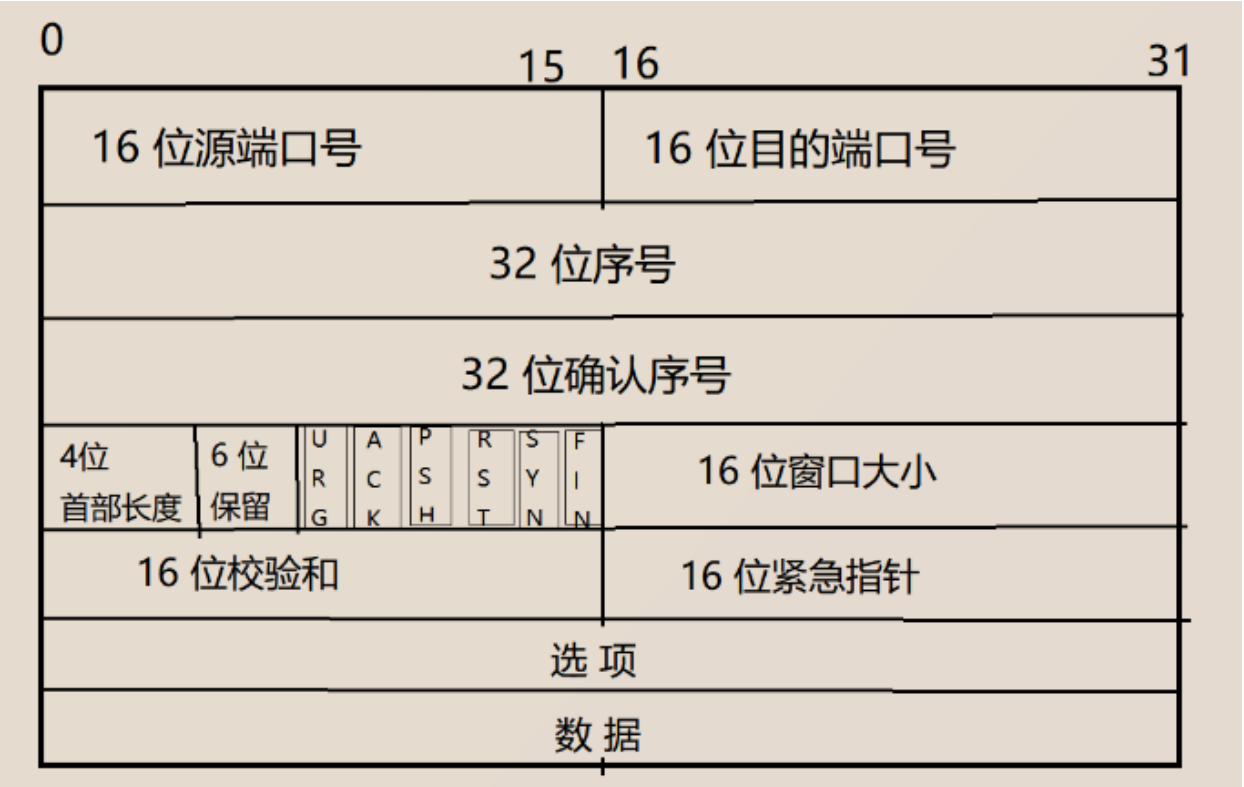
- 物理层：设备间通过媒介比特流传输，确定机械和电气规范 IEEE.802.11
- 数据链路层：将比特封装成帧传输和点到点传输 PPP VLAN
- 网络层：负责数据包从源到宿传输和网际互连 IP ARP ICMP
- 传输层：提供端到端的可靠报文传递和错误恢复 TCP、UDP
- 会话层：建立、管理和终止会话。 NFS 、RPC
- 表示层：对数据进行翻译，解密和压缩 JEPG ASCII
- 应用层：允许访问OSI环境的手段 WWW、HTTP、FTP 、DNS

TCP/IP四层协议：

- **应用层**:应用程序间沟通的层。
- **运输层**:节点间的数据传送，应用程序之间的通信服务，主要功能是数据格式化、数据确认和丢失重传等。
- **网际层**:负责提供基本的数据封包传送功能，让每一块数据包都能够到达目的主机。
- **网络接口层**:定义如何使用实际网络来传送数据。

2、TCP协议报文

TCP协议全称: 传输控制协议, 顾名思义, 就是要对数据的传输进行一定的控制.



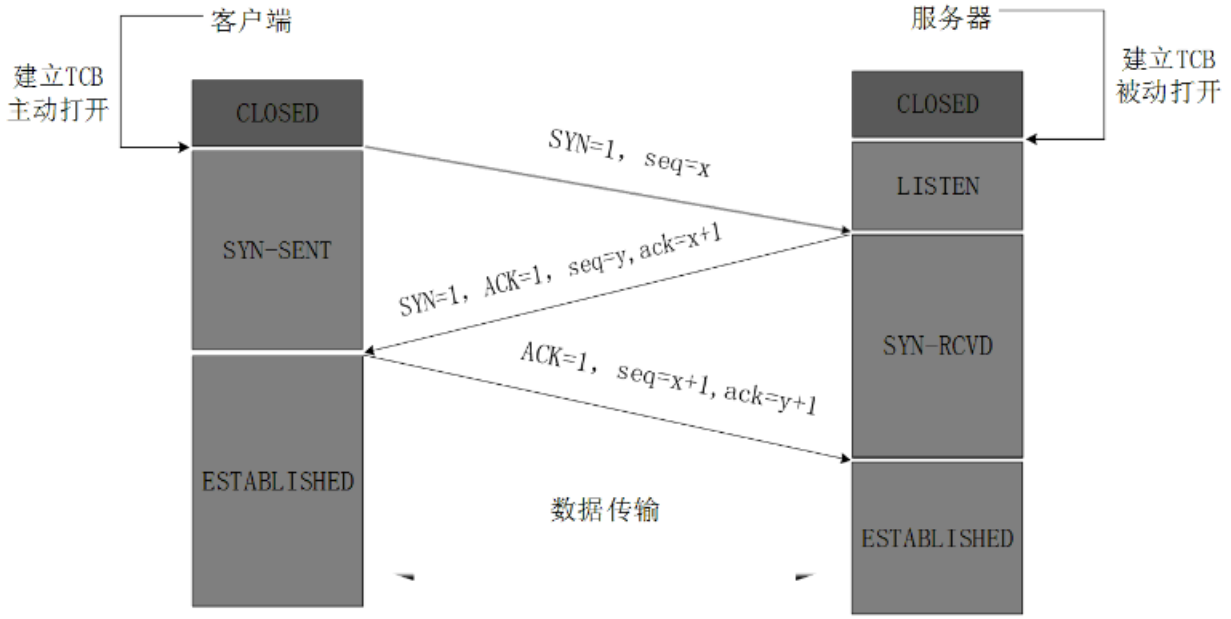
每部分的含义和作用

源端口号/目的端口号: 表示数据从哪个进程来, 到哪个进程去.
32位序号:
4位首部长度: 表示该tcp报头有多少个4字节(32个bit)
6位保留: 顾名思义, 先保留着, 以防万一
6位标志位
URG: 标识紧急指针是否有效
ACK: 标识确认序号是否有效
PSH: 用来提示接收端应用程序立刻将数据从tcp缓冲区读走
RST: 要求重新建立连接. 我们把含有RST标识的报文称为复位报文段
SYN: 请求建立连接. 我们把含有SYN标识的报文称为同步报文段
FIN: 通知对端, 本端即将关闭. 我们把含有FIN标识的报文称为结束报文段

16位窗口大小:
16位检验和: 由发送端填充, 检验形式有CRC校验等. 如果接收端校验不通过, 则认为数据有问题. 此处的校验和不光包含TCP首部, 也包含TCP数据部分.
16位紧急指针: 用来标识哪部分数据是紧急数据.

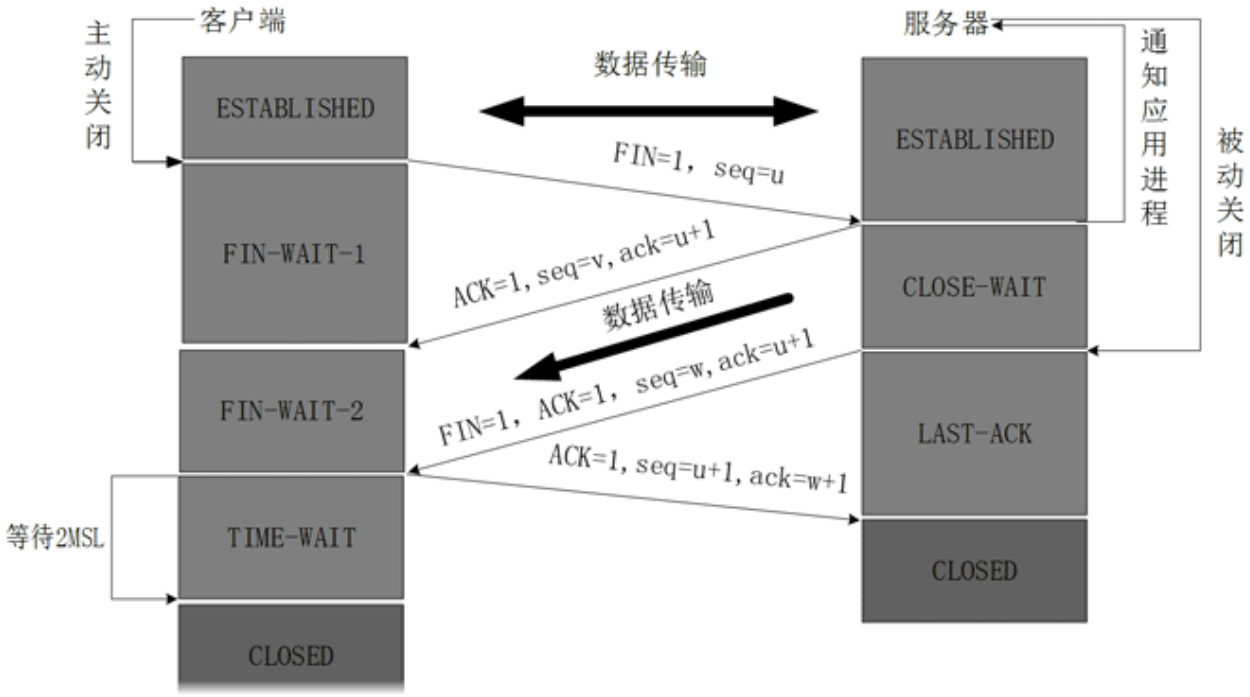
3、TCP三次握手，四次挥手

三次握手



流程:

四次挥手



流程：

3、TCP建立连接可以是两次吗?

不可以，建立连接三次握手主要是解决已失效的连接请求报文又传送到了服务器，从而产生的错误。

如果使用两次握手建立连接。假设客户端发送第一个请求连接并且没有丢失时，只是因为在网络中滞留时间太长，TCP的客户端一直没有接收到确认报文，认为服务器没有接收到。重新向服务器发送报文，服务器和客户端经两次握手建立连接，传输数据，关闭连接。上次滞留的请求，网络通畅后到达了服务器，这段报文本应该是失效的，经过两次握手会重新建立连接，造成了不必要的错误和资源浪费。

4、为什么建立连接三次握手，关闭连接四次握手?

- 建立连接时，服务器在LISTEN状态，收到建立连接请求的SYN报文后，把ACK+SYN放在一个报文里发送给客户端。
- 关闭连接时，服务器收到对方的FIN报文时，仅仅表示对方不在发送数据了，但是还能够接受数据，而自己也未必将数据全部发送给对方，会先发送ACK报文告诉对方收到了断开连接请求，等数据发送完成后，再将ACK+FIN发送给客户端关闭连接。所以导致多了一次。

5、TIME_WAIT 2MSL(最长报文段寿命)作用，常见问题解决办法?

MSL（Maximum Segment Lifetime）,TCP允许不同的实现可以设置不同的MSL值。

debian默认60秒

cat /proc/sys/net/ipv4/tcp_fin_timeout

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

作用：

- 保证客户端的最后一个ACK报文能够到达服务器，因为这个ACK报文可能会丢失，站在服务器角度来看，服务器已经发送FIN+ACK报文请求断开了，客户端没回应，应该是没有收到，服务器会重新发送一次，而客户端就能在这个2MSL时间段内收到这个重传的报文，并给出回应，重启2MSL计时器.
- 避免已失效的报文重新出现在本连接中。客户端发送完最后的确认报文后，在2MSL时间中，就可以使本次连接中所产生的报文从网络中小时。这样新的连接中就不会有就连接的请求报文了。

问题：

- 如果进程中的某个TCP连接处于TIME_WAIT等待状态，因为这个等待时间比较长，在这期间该连接使用的端口将一直被占用。
- 如果一个服务端进程（绑定了某个端口）退出（正常退出或异常退出）后，立即启动一个新的该进程，端口的释放不及时，导致这个端口还没有被释放，不能被再次绑定，从而导致新进程绑定端口失败。

解决：

我们在网络编程中经常设置的SO_REUSEADDR选项就可以解决这个问题，

```
int flag = 1;
setsockopt(socket, SOL_SOCKET, SO_REUSEADDR, reinterpret_cast<const char*>(&flag), sizeof(flag));
```

SO_REUSEADDR提供如下四个功能：

- SO_REUSEADDR允许启动一个监听服务器并捆绑其众所周知端口，即使以前建立的将此端口用做他们的本地端口的连接仍存在。这通常是重启监听服务器时出现，若不设置此选项，则bind时将出错。
- SO_REUSEADDR允许在同一端口上启动同一服务器的多个实例，只要每个实例捆绑一个不同的本地IP地址即可。对于TCP，我们根本不可能启动捆绑相同IP地址和相同端口号的多个服务器。
- SO_REUSEADDR允许单个进程捆绑同一端口到多个套接口上，只要每个捆绑指定不同的本地IP地址即可。这一般不用于TCP服务器。
- SO_REUSEADDR允许完全重复的捆绑：当一个IP地址和端口绑定到某个套接口上时，还允许此IP地址和端口捆绑到另一个套接口上。一般来说，这个特性仅在支持多播的系统上才有，而且只对UDP套接口而言（TCP不支持多播）。

6、保证可靠性的机制

- **校验和：** TCP将保持它首部和数据的校验和。这是一个端到端的检验和，目的是检测数据在传输过程中的任何变化。如果收到的段的检验和有差错，TCP将丢弃这个报文段和不确认收到此报文段。

- **确认应答：**每一个ACK都带有对应的确认序列号, 意思是告诉发送者, 我已经收到了哪些数据; 下一次你要从哪里开始发.
- **超时重传：** 当TCP发出一个段后，它启动一个定时器等待接收端确认收到这个报文段。如果不能即使收到一个确认，将重发这段报文。
- **流量控制：**TCP连接的每一方都有固定大小的缓冲空间，TCP的接收端只允许发送端发送接收端缓冲区能容纳的数据。当接收方来不及处理发送方的数据，能提示发送方降低发送速率，防止包丢失。TCP使用的流量控制协议是可变大小的滑动窗口协议。
- **拥塞控制：**当网络拥塞时，减少数据的发送。

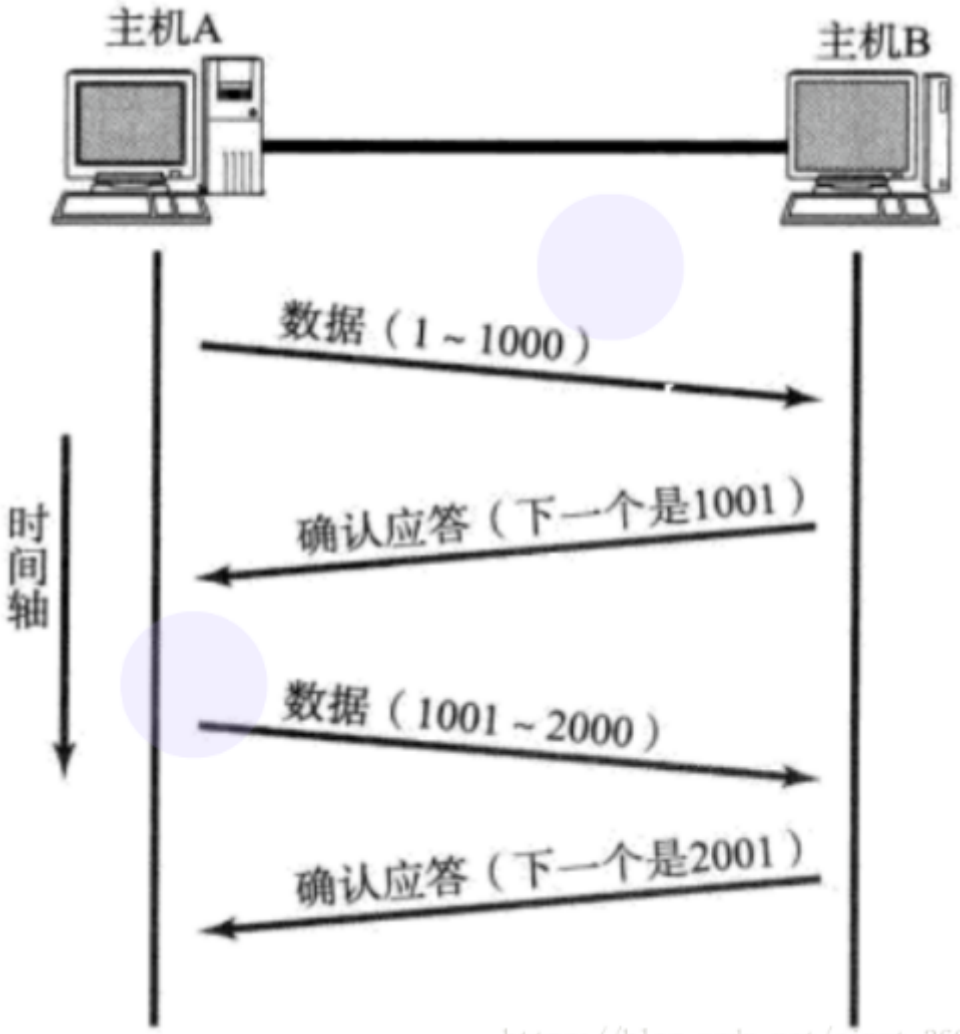
7、提高性能的机制

- **滑动窗口**
- **快速重传**
- **延迟应答**
- **捎带应答**

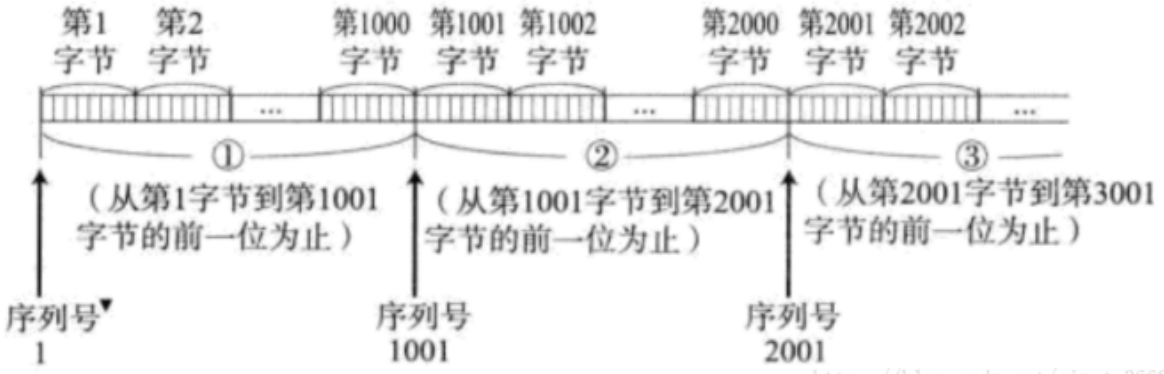
8、定时器

- 超时重传定时器
- 保活定时器
- TIME_WAIT定时器

确认应答机制(ACK机制)



TCP将每个字节的数据都进行了编号, 即为序列号.



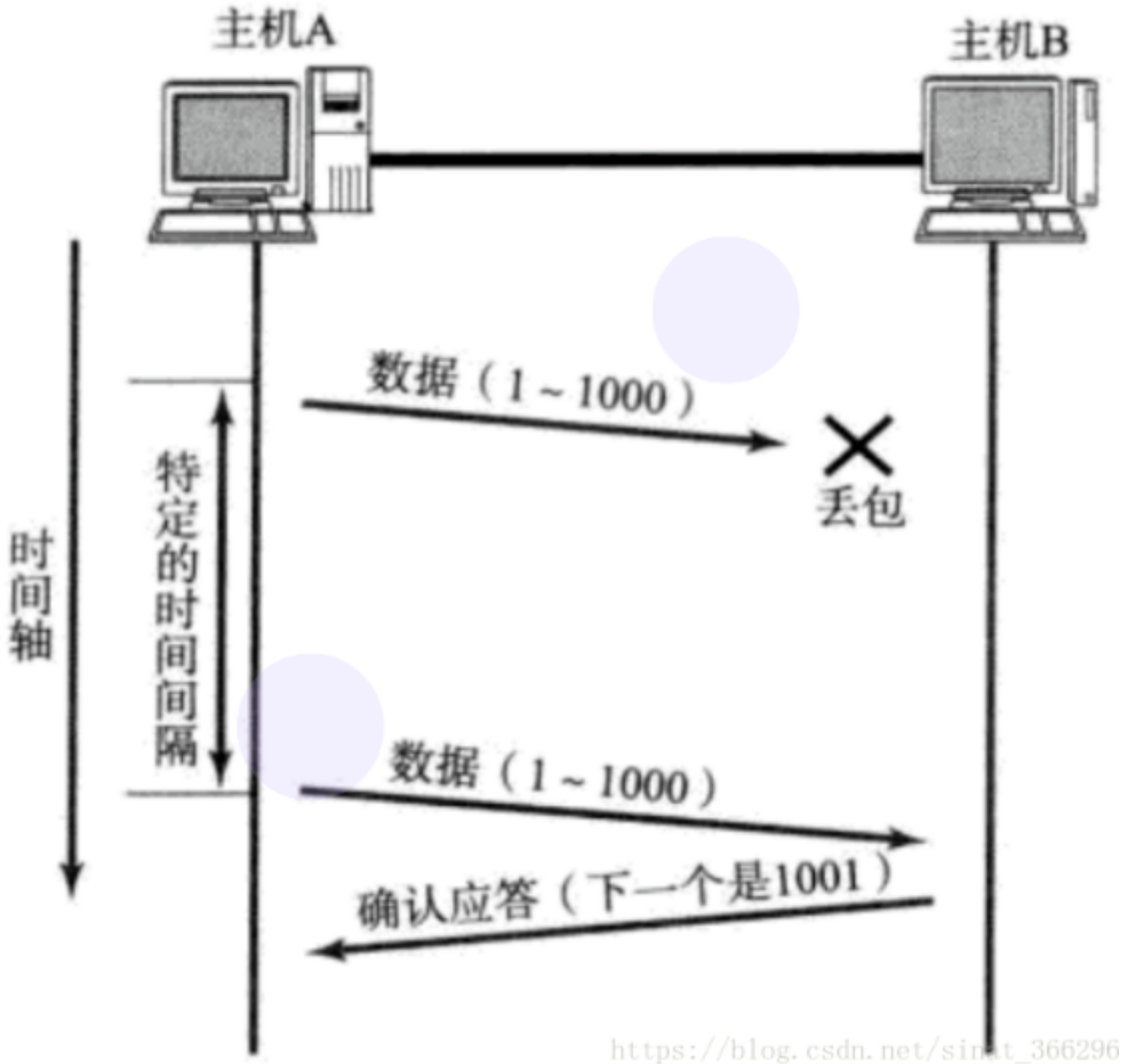
每一个ACK都带有对应的确认序列号, 意思是告诉发送者, 我已经收到了哪些数据; 下一次你要从哪里开始发.

比如, 客户端向服务器发送了1005字节的数据, 服务器返回给客户端的确认序号是1003, 那么说明服务器只收到了1-1002的数据.

1003, 1004, 1005都没收到.

此时客户端就会从1003开始重发.

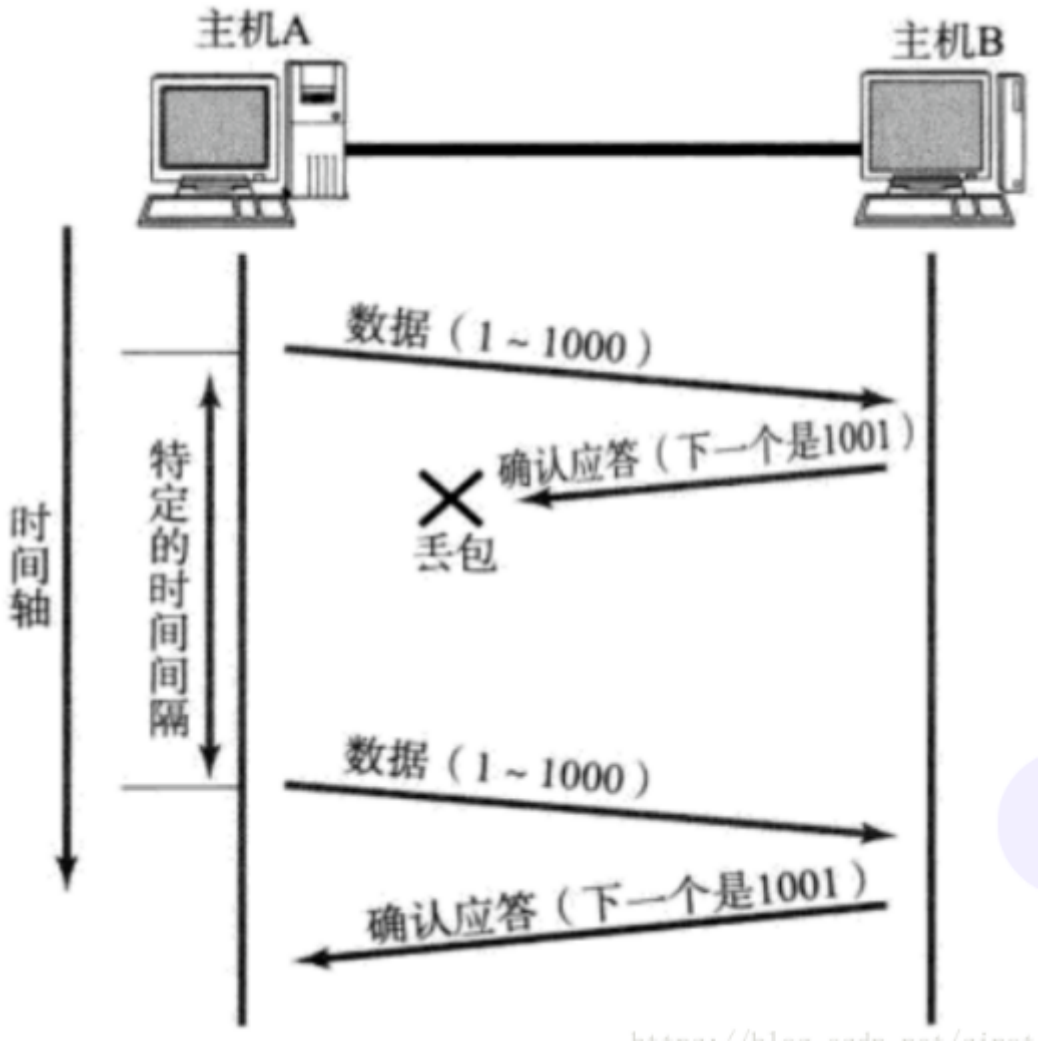
超时重传机制



主机A发送数据给B之后, 可能因为网络拥堵等原因, 数据无法到达主机B

如果主机A在一个特定时间间隔内没有收到B发来的确认应答, 就会进行重发

但是主机A没收到确认应答也可能是ACK丢失了.



这种情况下, 主机B会收到很多重复数据.

那么TCP协议需要识别出哪些包是重复的, 并且把重复的丢弃.

这时候利用前面提到的序列号, 就可以很容易做到去重.

超时时间如何确定?

最理想的情况下, 找到一个最小的时间, 保证 “确认应答一定能在这个时间内返回”.

但是这个时间的长短, 随着网络环境的不同, 是有差异的.

如果超时时间设的太长, 会影响整体的重传效率; 如果超时时间设的太短, 有可能会频繁发送重复的包.

TCP为了保证任何环境下都能保持较高性能的通信, 因此会动态计算这个最大超时时间.

Linux中(BSD Unix和Windows也是如此), 超时以500ms为一个单位进行控制, 每次判定超时重发的超时时间都是500ms的整数倍.

如果重发一次之后, 仍然得不到应答, 等待 2*500ms 后再进行重传. 如果仍然得不到应答, 等待 4*500ms 进行重传.

依次类推, 以指数形式递增. 累计到一定的重传次数, TCP认为网络异常或者对端主机出现异常, 强制关闭连接.

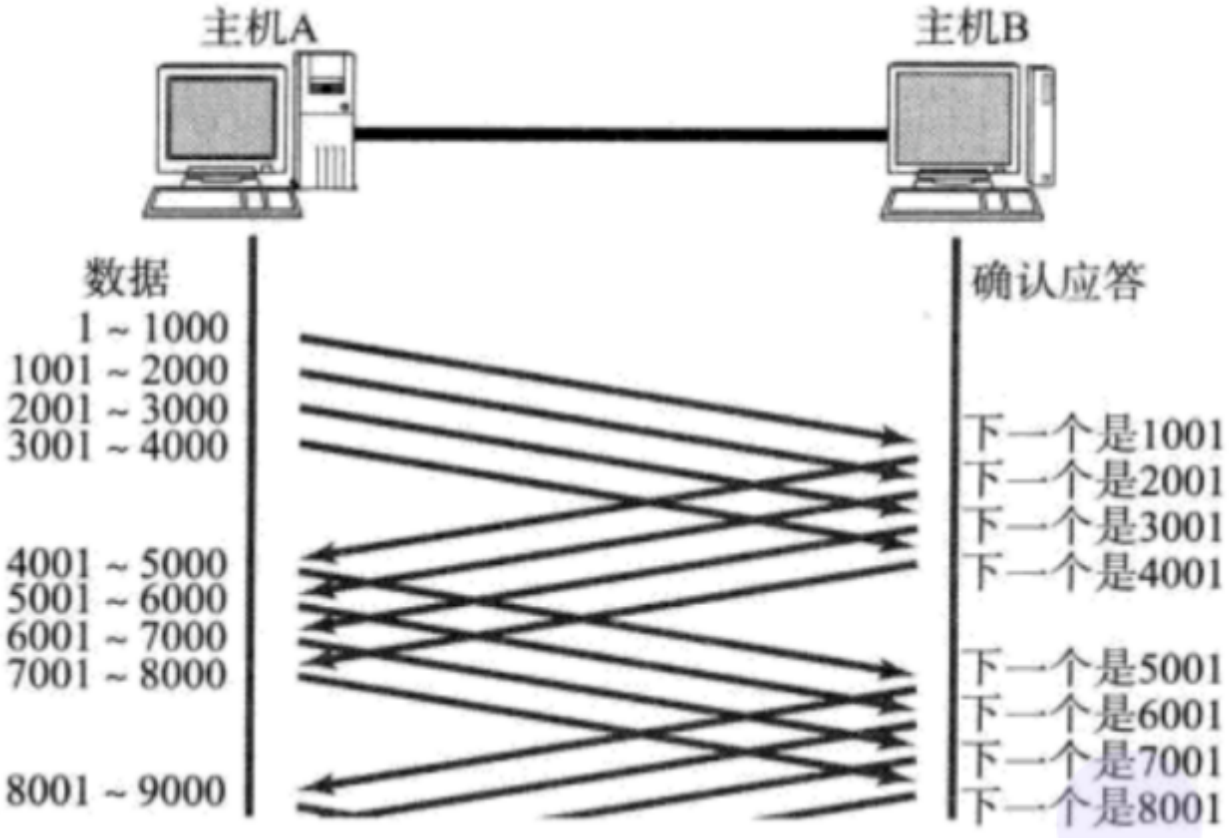
滑动窗口

刚才我们讨论了确认应答机制, 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, 就是性能较差. 尤其是数据往返时间较长的时候.

那么我们可不可以一次发送多个数据段呢?

例如这样:



一个概念: **窗口**

窗口大小指的是无需等待确认应答就可以继续发送数据的最大值.

上图的窗口大小就是4000个字节 (四个段).

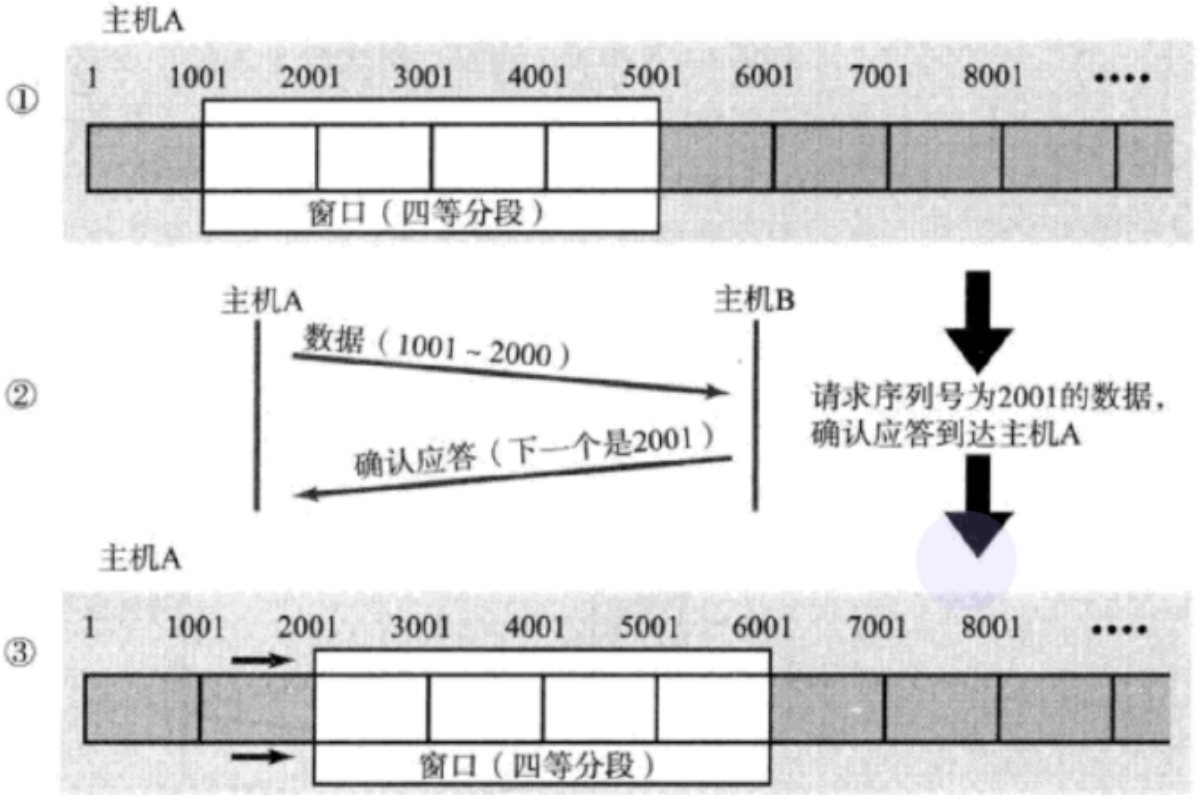
发送前四个段的时候, 不需要等待任何ACK, 直接发送

收到第一个ACK确认应答后, 窗口向后移动, 继续发送第五六七八段的数据...

因为这个窗口不断向后滑动, 所以叫做滑动窗口.

操作系统内核为了维护这个滑动窗口, 需要开辟发送缓冲区来记录当前还有哪些数据没有应答

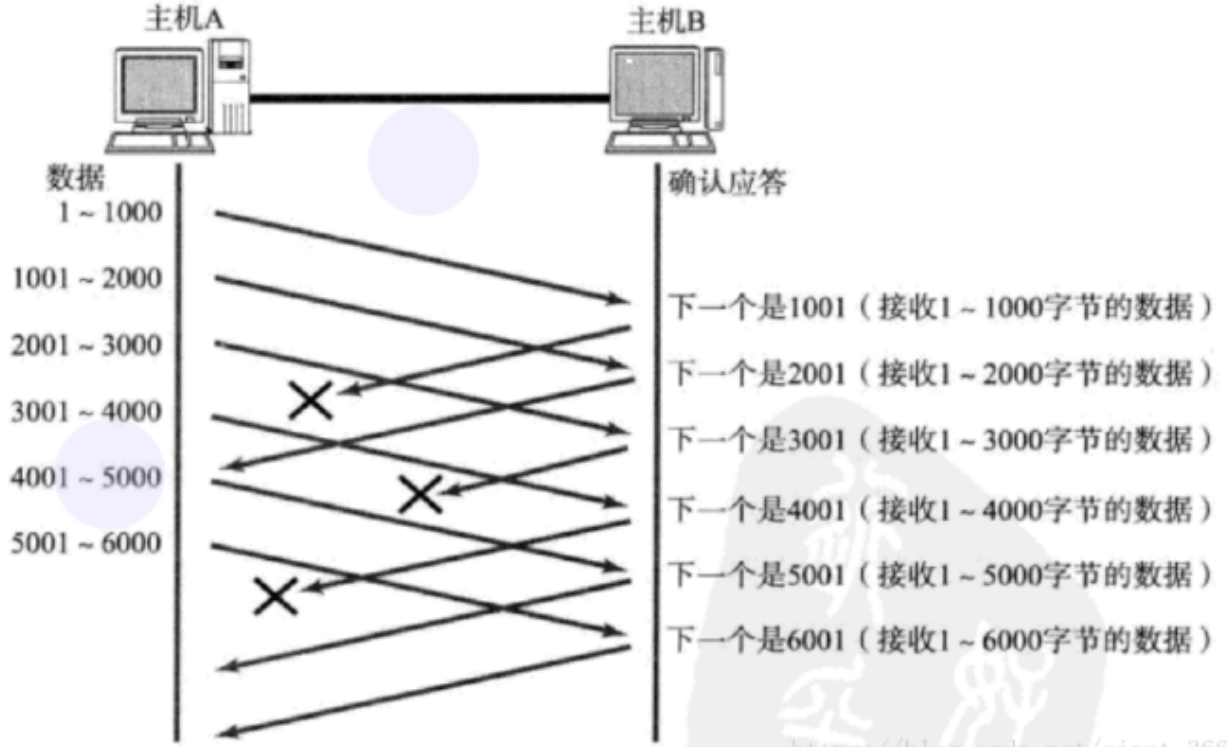
只有ACK确认应答过的数据, 才能从缓冲区删掉.



如果出现了丢包, 那么该如何进行重传呢?

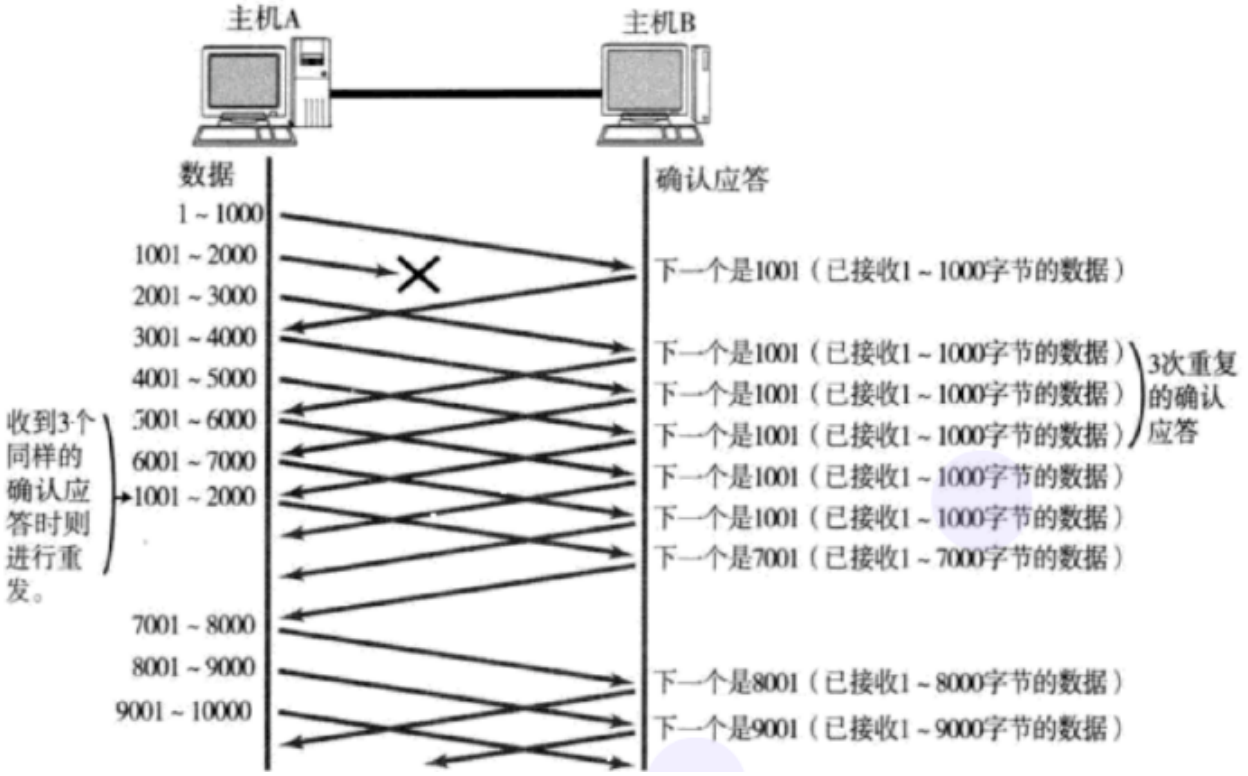
此时分两种情况讨论:

1, 数据包已经收到, 但确认应答ACK丢了.



这种情况下, 部分ACK丢失并无大碍, 因为还可以通过后续的ACK来确认对方已经收到了哪些数据包.

2, 数据包丢失



当某一段报文丢失之后, 发送端会一直收到 1001 这样的ACK, 就像是在提醒发送端 “我想要的是 1001”。如果发送端主机连续三次收到了同样一个 “1001” 这样的应答, 就会将对应的数据 1001 - 2000 重新发送。这个时候接收端收到了 1001 之后, 再次返回的ACK就是7001了。因为2001 - 7000接收端其实之前就已经收到了, 被放到了接收端操作系统内核的接收缓冲区中.

这种机制被称为 “**高速重发控制**”（也叫 “**快重传**”）

流量控制

接收端处理数据的速度是有限的. 如果发送端发的太快, 导致接收端的缓冲区被填满, 这个时候如果发送端继续发送, 就会造成丢包, 进而引起丢包重传等一系列连锁反应.

因此TCP支持根据接收端的处理能力, 来决定发送端的发送速度.

这个机制就叫做 流量控制(Flow Control)

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通知发送端;

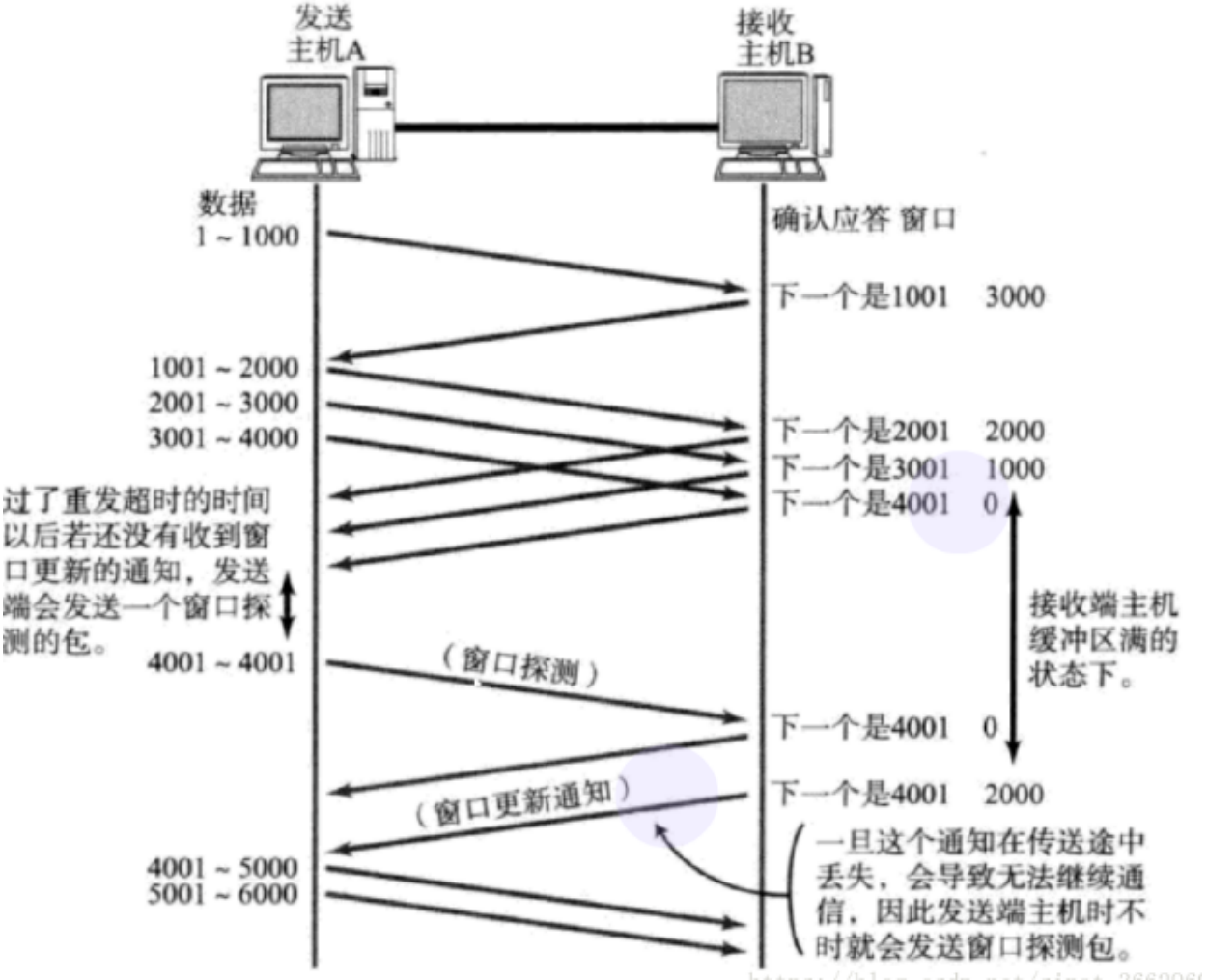
窗口大小越大, 说明网络的吞吐量越高;

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;

发送端接受到这个窗口大小的通知之后, 就会减慢自己的发送速度;

如果接收端缓冲区满了, 就会将窗口置为0;

这时发送方不再发送数据, 但是需要定期发送一个窗口探测数据段, 让接收端把窗口大小再告诉发送端.



那么接收端如何把窗口大小告诉发送端呢?

我们的TCP首部中, 有一个16位窗口大小字段, 就存放了窗口大小的信息;

16位数字最大表示65536, 那么TCP窗口最大就是65536字节么?

实际上, TCP首部40字节选项中还包含了一个窗口扩大因子M, 实际窗口大小是窗口字段的值左移 M 位(左移一位相当于乘以2).

拥塞控制

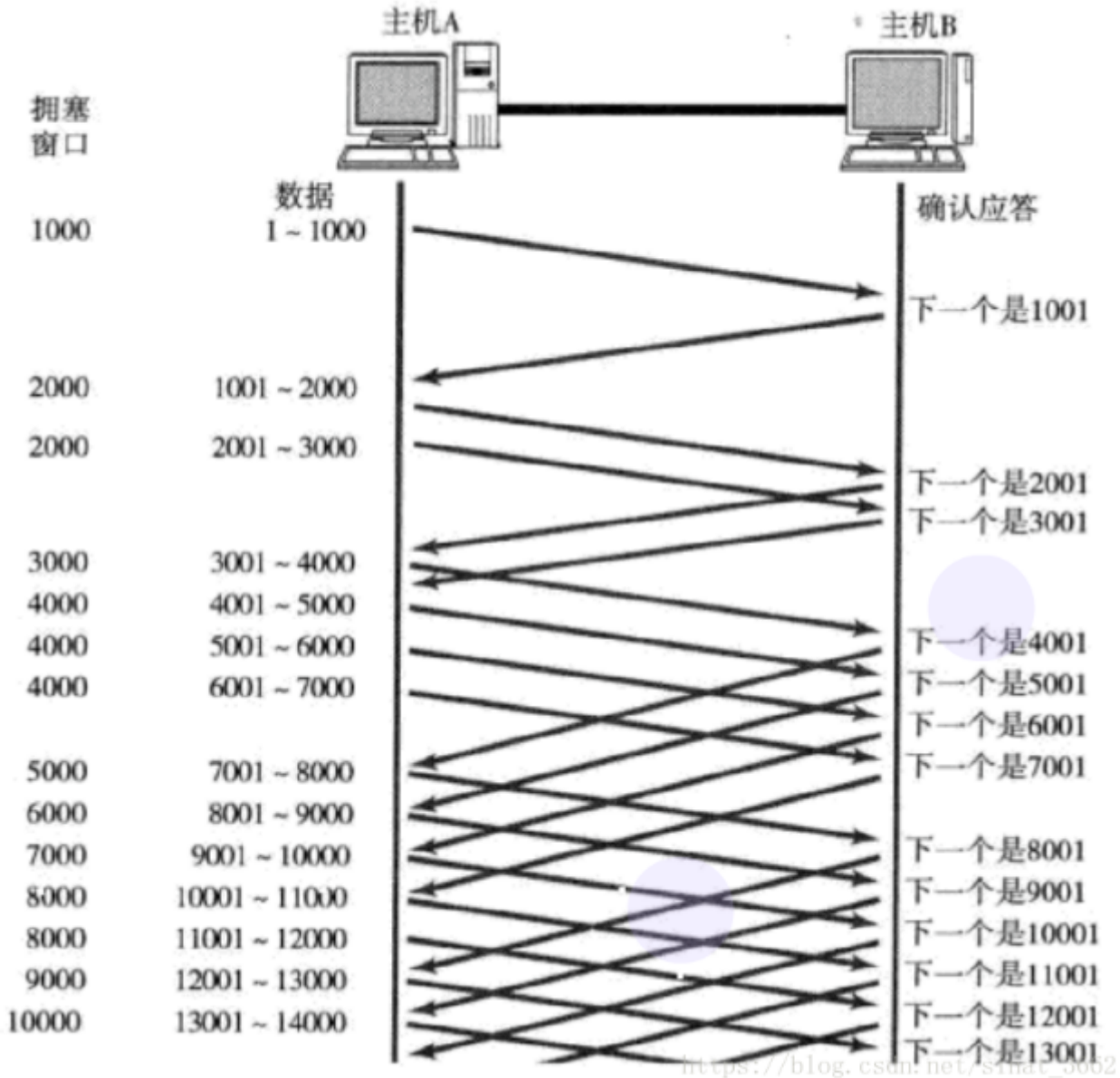
虽然TCP有了滑动窗口这个大杀器, 能够高效可靠地发送大量数据.

但是如果在刚开始就发送大量的数据, 仍然可能引发一些问题.

因为网络上有很多计算机, 可能当前的网络状态已经比较拥堵.

在不清楚当前网络状态的情况下, 贸然发送大量数据, 很有可能雪上加霜.

因此, TCP引入 慢启动 机制, 先发少量的数据, 探探路, 摸清当前的网络拥堵状态以后, 再决定按照多大的速度传输数据.



在此引入一个概念 **拥塞窗口**

发送开始的时候, 定义拥塞窗口大小为1;

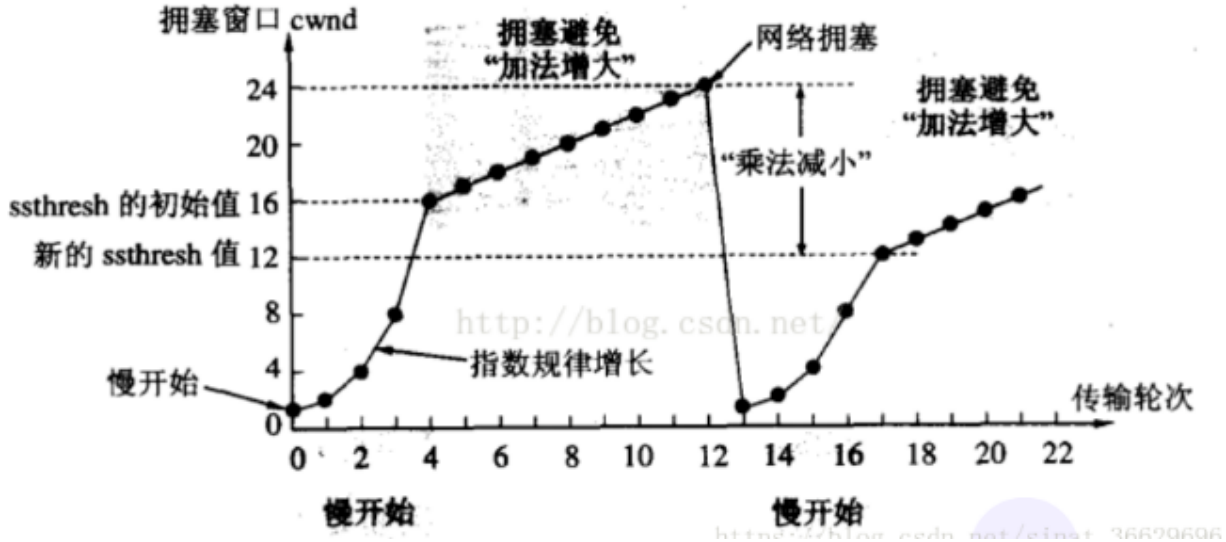
每次收到一个ACK应答, 拥塞窗口加1;

每次发送数据包的时候, 将拥塞窗口和接收端主机反馈的窗口大小做比较, 取较小的值作为实际发送的窗口

像上面这样的拥塞窗口增长速度, 是指数级别的.

“慢启动” 只是指初使时慢, 但是增长速度非常快.

为了不增长得那么快, 此处引入一个名词叫做慢启动的阈值, 当拥塞窗口的大小超过这个阈值的时候, 不再按照指数方式增长, 而是按照线性方式增长.



- 当TCP开始启动的时候, 慢启动阈值等于窗口最大值
- 在每次超时重发的时候, 慢启动阈值会变成原来的一半, 同时拥塞窗口置回1

少量的丢包, 我们仅仅是触发超时重传;

大量的丢包, 我们就认为是网络拥塞;

当TCP通信开始后, 网络吞吐量会逐渐上升;

随着网络发生拥堵, 吞吐量会立刻下降.

拥塞控制, 归根结底是TCP协议想尽可能快的把数据传输给对方, 但是又要避免给网络造成太大压力的折中方案.

延迟应答

如果接收数据的主机立刻返回ACK应答, 这时候返回的窗口可能比较小.

假设接收端缓冲区为1M. 一次收到了500K的数据;

如果立刻应答, 返回的窗口大小就是500K;

但实际上可能处理端处理的速度很快, 10ms之内就把500K数据从缓冲区消费掉了; 在这种情况下, 接收端处理还远没有达到自己的极限, 即使窗口再放大一些, 也能处理过来;

如果接收端稍微等一会儿再应答, 比如等待200ms再应答, 那么这个时候返回的窗口大小就是1M

窗口越大, 网络吞吐量就越大, 传输效率就越高.

TCP的目标是在保证网络不拥堵的情况下尽量提高传输效率;

那么所有的数据包都可以延迟应答么?

肯定也不是

有两个限制

- 数量限制: 每隔N个包就应答一次
- 时间限制: 超过最大延迟时间就应答一次

具体的数量N和最大延迟时间, 依操作系统不同也有差异

一般 N 取2, 最大延迟时间取200ms

捎带应答

在延迟应答的基础上, 我们发现, 很多情况下
客户端和服务器在应用层也是 “一发一收” 的
意味着客户端给服务器说了 “How are you”
服务器也会给客户端回一个 “Fine, thank you”
那么这个时候ACK就可以搭顺风车, 和服务器回应的 “Fine, thank you” 一起发送给客户端

面向字节流

创建一个TCP的socket, 同时在内核中创建一个 发送缓冲区 和一个 接收缓冲区;
调用write时, 数据会先写入发送缓冲区中;
如果发送的字节数太大, 会被拆分成多个TCP的数据包发出;
如果发送的字节数太小, 就会先在缓冲区里等待, 等到缓冲区大小差不多了, 或者到了其他合适的时机再发送出去;
接收数据的时候, 数据也是从网卡驱动程序到达内核的接收缓冲区;
然后应用程序可以调用read从接收缓冲区拿数据;
另一方面, TCP的一个连接, 既有发送缓冲区, 也有接收缓冲区,
那么对于这一个连接, 既可以读数据, 也可以写数据, 这个概念叫做 全双工

由于缓冲区的存在, 所以TCP程序的读和写不需要一一匹配
例如:
写100个字节的数据, 可以调用一次write写100个字节, 也可以调用100次write, 每次写一个字节;
读100个字节数据时, 也完全不需要考虑写的时候是怎么写的, 既可以一次read 100个字节, 也可以一次read一个字节, 重复100次;

9、TCP和UDP区别

1、TCP面向连接的，可靠性高，UDP面向无连接的，可靠性低；

2、TCP面向字节流，UDP面向报文的；

3、TCP只能实现点到点，UPD支持一对一，一对多，多对一，多对多的交互通信。

4、由于TCP是连接通信，需要三次握手、重新确认等连接过程，会有延时、时间差，过程复　杂，容易被攻击；UDP没有建立连接的过程，实时性较强，也比较安全。

5、在传输相同大小的数据时。TCP在IP协议的基础上添加了序号机制、确认机制、超时重传机制等，保证了传输的可靠性，不会出现丢包或乱序，而UDP有丢包情况，故TCP开销大，UDP开销小。

10、HTTP和HTTPS区别

1）HTTPS协议需要到CA申请证书，一般免费的证书很少，因此需要花费一定的费用。

2）HTTP是超文本传输协议，信息是明文传输，HTTPS则是具有安全性的SSL加密传输协议。

3）HTTP和HTTPS使用的是完全不同的连接方式，用的端口也不一样HTTP 端口80 HTTPS端口443.

4）HTTP的连接很简单，是无状态的。HTTPS的协议是由SSL+HTTP协议构建的可进行加密传输、身份验证的网络协议，比HTTP更加安全。（无状态的意思对其数据包的发送传输和接收是相互独立的。无连接的意思是指通信双发都不长久的维持对方的任何信息）。

11、常用的端口

- ssh服务器：22端口
- ftp服务器：21端口
- http服务器：80端口
- telnet服务器：23端口
- https服务器：443端口
- MYSQL服务器：3306端口

12、select poll epoll

区别：

select和poll每次循环调用时，都需要将描述符和事件从用户空间拷贝到内核空间，开销比较大；epoll通过内核和用户空间共享一块内存来实现。

select 底层使用数组保存fd，有数目限制　poll使用链表没有数目限制，与系统内存有关。epoll底层是红黑树实现。**select　cat　/proc/sys/fs/file-max　　ulimit -a　open files　默认1024　都需要修改**

select和poll内核通过轮询实现的，时间复杂度是O(n),epoll是在每一个描述符上设置回调函数，时间复杂度是O(1)。1G内存的机器上可以打开10万左右的连接。

场景：

epoll ： 连接数较多，并且活跃数较少的时候。epoll_wait 不断的轮询就绪链表，链表中有就绪的fd时，会唤醒在epoll_walt中的进程。epoll只需要判断就绪链表是否为空。

select和poll ： 连接数较少，并且都很活跃的情况下。（不断轮询整个fd集合）

epoll两种触发模式：

ET模式是高速模式，叫做边缘触发模式，**LT**模式是默认模式，叫做水平触发模式；

区别：

LT模式：如果对一个描述符上的数据没有读取完成，下次当epoll_wait返回时会继续触发，可以继续获取这个描述符，从而能够接着读。

ET模式：如果一个描述符数据到达，然后读取这个描述符上的数据，如果没有将数据读取完，当下次epoll_wait返回的时候，这个描述符中的数据不能在被读取到。

实现方式：

当一个socket描述符的中断事件发生，内核会将数据从网卡复制到内核，同时将1socket描述符插入到rddlist中，如果此时调用了epoll_wait会把rdlist中就绪的socket描述符复制到用户空间,然后清理掉这个rdlist中的数据，最后epoll_wait还会再次检查这些socket描述符。如果是工作在LT模式下，并且这些socket描述符上还有数据没有读取完，那么LT就会再次把没有读完的socket描述符放入到rdlist中，所以再次调用epoll_wait的时候会再次触发。

epoll为什么要有EPOLL ET触发模式?

如果采用EPOLL LT模式的话，系统中一旦有大量你不需要读写的就绪文件描述符，它们每次调用epoll_wait都会返回，这样会大大降低处理程序检索自己关心的就绪文件描述符的效率.。而采用EPOLL ET这种边沿触发模式的话，当被监控的文件描述符上有可读写事件发生时，epoll_wait()会通知处理程序去读写。如果这次没有把数据全部读写完(如读写缓冲区太小)，那么下次调用epoll_wait()时，它不会通知你，也就是它只会通知你一次，直到该文件描述符上出现第二次可读写事件才会通知你！这种模式比水平触发效率高，系统不会充斥大量你不关心的就绪文件描述符。

epoll优点：

1、没有最大连接数限制，能打开的fd上线远大于1024（1G内存上能监听约10万个端口）；

2、采用回调的方式，不会随着FD数目的增多而导致效率降低，只有活跃的fd才会调用callback函数。

3、内存拷贝，利用mmap()文件映射内存加速与内核空间的消息传递，减少复制的开销。

epoll函数介绍

epoll_create创建一个epoll句柄；epoll_ctl注册监听的事件类型；epoll_wait等待事件的产生。

每次注册新的事件到epoll句柄时，会把所有的fd拷贝到内核，而不是在epoll_wait的时候重复拷贝，保证了每个fd在整个过程中只会拷贝一次。

epoll_ctl时会为每个fd指定一个回调函数，当设备就绪时，会调用这个回调函数将就绪的fd加入到就绪链表。

epoll_wait查看就绪链表中有没有就绪的fd

```
struct eventpoll{
    ....
    /*红黑树的根节点，这颗树中存储着所有添加到epoll中的需要监控的事件*/
    struct rb_root  rbr;
    /*双链表中则存放着将要通过epoll_wait返回给用户的满足条件的事件*/
    struct list_head rdlist;
    ....
};
```

每一个epoll对象都有一个独立的eventpoll结构体，用于存放通过epoll_ctl方法向epoll对象中添加进来的事件。这些事件都会挂载在红黑树中，如此，重复添加的事件就可以通过红黑树而高效的识别出来

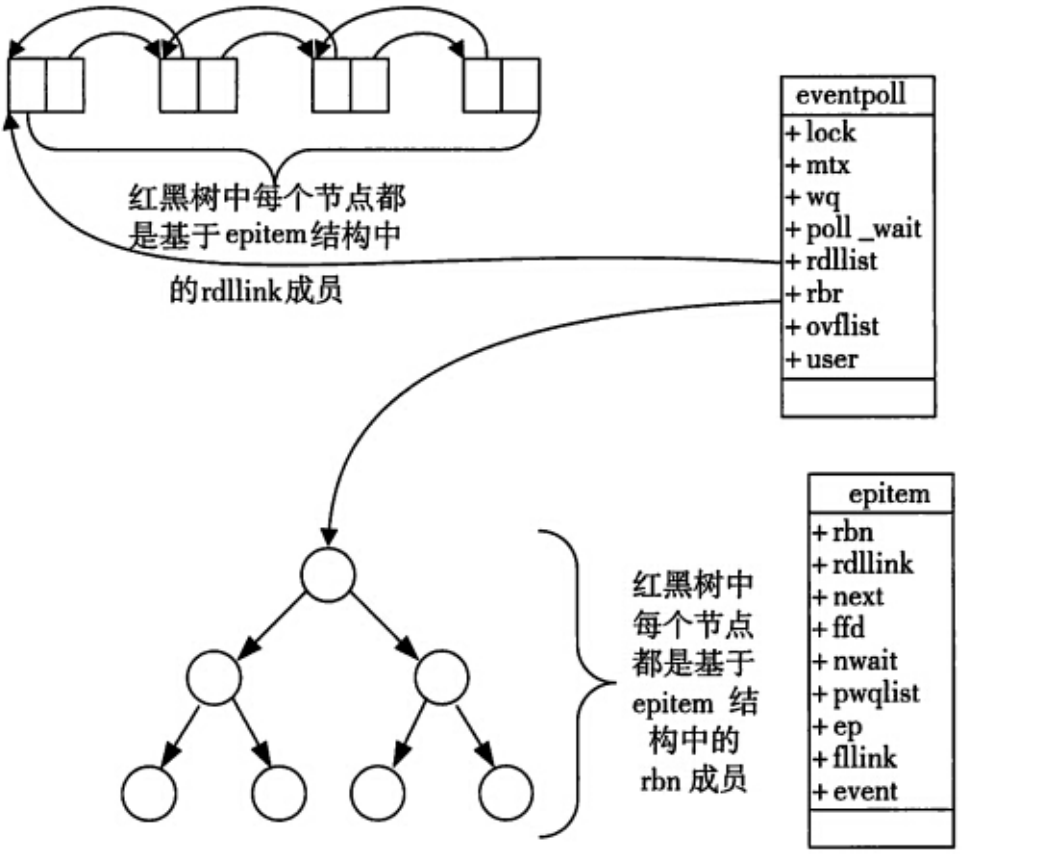
（n是所有节点的数量，高度为5的满二叉树，节点数量是31，查找这个二叉树的叶子结点花费的时间是log2^31 = 5）

而所有添加到epoll中的事件都会与设备(网卡)驱动程序建立回调关系，也就是说，当相应的事件发生时会调用这个回调方法。这个回调方法在内核中叫ep_poll_callback,它会将发生的事件添加到rdlist双链表中。

在epoll中，对于每一个事件，都会建立一个epitem结构体，如下所示：

```
struct epitem{
    struct rb_node  rbn;//红黑树节点
    struct list_head    rdllink;//双向链表节点
    struct epoll_filefd  ffd;  //事件句柄信息
    struct eventpoll *ep;    //指向其所属的eventpoll对象
    struct epoll_event event; //期待发生的事件类型
}
```

当调用epoll_wait检查是否有事件发生时，只需要检查eventpoll对象中的rdlist双链表中是否有epitem元素即可。如果rdlist不为空，则把发生的事件复制到用户态，同时将事件数量返回给用户。



epoll数据结构示意图

从上面的讲解可知：通过红黑树和双链表数据结构，并结合回调机制，造就了epoll的高效。

OK，讲解完了Epoll的机理，我们便能很容易掌握epoll的用法了。一句话描述就是：三步曲。

第一步：epoll_create()系统调用。此调用返回一个句柄，之后所有的使用都依靠这个句柄来标识。

第二步：epoll_ctl()系统调用。通过此调用向epoll对象中添加、删除、修改感兴趣的事件，返回0标识成功，返回-1表示失败。

第三步：epoll_wait()系统调用。通过此调用收集收集在epoll监控中已经发生的事件。

13、tcp 粘包半包问题怎么处理?

粘包：连续给对端发送两个或者两个以上的数据包，对端一次收取中可能收到的数据包大于1个，几个和某个包的部分或者几个包一起。

半包：对端一次收取中只能接收到一个包的部分。

解决办法：

1）固定包长的数据包：每个协议包的长度都是固定的。包收满后进行解析，如果未收满则先存起来。

2）以指定字符（串）为包的结束标志：字节流中遇到特殊的符号值时就认为到一个包的末尾了。例如：ftp '/r/n'包结束标志

3）包头+包体格式：包头是固定大小的，且包头中必须含有一个字段来说明接下来的包体有多大。对端先收取到包头大小字节数目，然后解析包头，根据包头中指定的包体大小收取包体，等包体收取够了。就组装成完整的包来处理。

14、tcpdump抓包，如何分析数据包?

`tcpdump tcp -i ens32 -t -s 0 -c 3 and dst port ! 22 and src net 192.168.218.0/24 -w ./target.cap`

(1)tcp: ip icmp arp rarp 和 tcp、udp、icmp这些选项等都要放到第一个参数的位置，用来过滤数据报的类型

(2)-i eth1 : 只抓经过接口eth1的包

(3)-t : 不显示时间戳

(4)-s 0 : 抓取数据包时默认抓取长度为68字节。加上-S 0 后可以抓到完整的数据包

(5)-c 100 : 只抓取100个数据包

(6)dst port ! 22 : 不抓取目标端口是22的数据包

(7)src net 192.168.218.0/24 : 数据包的源网络地址为192.168.218.0/24

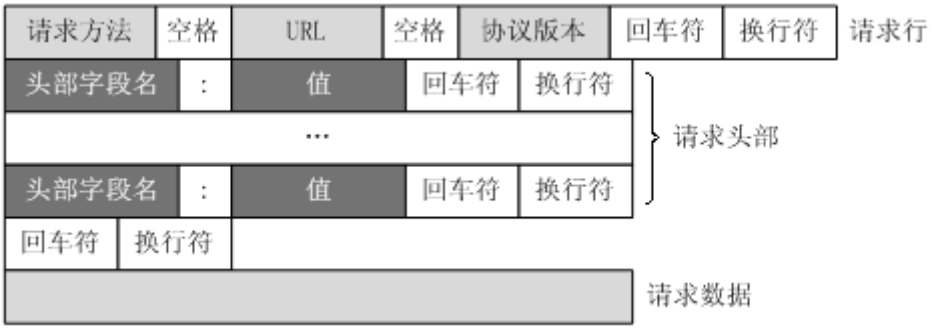
(8)-w ./target.cap : 保存成cap文件，方便用ethereal(即wireshark)分析

分析：

https://blog.csdn.net/daniel_ustc/article/details/18451203?utm_medium=distribute.pc_relevant_t0.none-task-blog-BlogCommendFromMachineLearnPai2-1.control&dist_request_id=56edd76b-2ee1-453c-96b2-27736521b121&depth_1-utm_source=distribute.pc_relevant_t0.none-task-blog-BlogCommendFromMachineLearnPai2-1.control

15、http报文格式

组成：请求行、请求头部、空行和请求数据。



< request-line >

< headers >

< blank line >

[< request-body >

16、http1.1与http1.0区别，http2.0特性，http3了解吗?

http1.1与http1.0区别

长连接

HTTP 1.0需要使用keep-alive参数来告知服务器端要建立一个长连接，而HTTP1.1默认支持长连接。

HTTP是基于TCP/IP协议的，创建一个TCP连接是需要经过三次握手的,有一定的开销，如果每次通讯都要重新建立连接的话，对性能有影响。因此最好能维持一个长连接，可以用个长连接来发多个请求。

节约带宽

HTTP 1.1支持只发送header信息(不带任何body信息)，如果服务器认为客户端有权限请求服务器，则返回100，否则返回401。客户端如果接受到100，才开始把请求body发送到服务器。

这样当服务器返回401的时候，客户端就可以不用发送请求body了，节约了带宽。

另外HTTP还支持传送内容的一部分。这样当客户端已经有一部分的资源后，只需要跟服务器请求另外的部分资源即可。这是支持文件断点续传的基础。

HOST域

现在可以web server例如tomat，设置虚拟站点是非常常见的，也即是说，web server上的多个虚拟站点可以共享同一个ip和端口。

HTTP1.0是没有host域的，HTTP1.1才支持这个参数。

17、HTTP1.1 HTTP 2.0主要区别和HTTP2.0特性

多路复用HTTP2.0使用了(类似epoll)多路复的技术，做到同一个连接并发处理多个请求，而且并发请求的数量比HTTP1.1大了好几个数量级。

当然HTTP1.1也可以多建立几个TCP连接，来支持处理更多并发的请求，但是创建TCP连接本身也是有开销的。

TCP连接有一个预热和保护的过程，先检查数据是否传送成功，一旦成功过，则慢慢加大传输速度。因此对应瞬时并发的连接，服务器的响应就会变慢。所以最好能使用一个建立好的连接，并且这个连接可以支持瞬时并发的请求。

数据压缩

HTTP1.1不支持header数据的压缩，HTTP2.0使用HPACK算法对header的数据进行压缩，这样数据体积小了，在网络上传输就会更快。

服务器推送

意思是说，当我们对支持HTTP2.0的web server请求数据的时候，服务器会顺便把一些客户端需要的资源一起推送到客户端，免得客户端再次创建连接发送请求到服务器端获取。这种方式非常合适加载静态资源。

服务器端推送的这些资源其实存在客户端的某处地方，客户端直接从本地加载这些资源就可以了，不用走网络，速度自然是快很多的。

HTTP3 ID 在切换网络环境时不会重新进行连接

在 HTTP/3 中，将弃用 TCP 协议，改为使用基于 UDP 的 QUIC 协议实现。

QUIC（快速UDP网络连接）是一种实验性的网络传输协议，由Google开发，该协议旨在使网页传输更快。

18、http1.1长连接时，发送一个请求阻塞了，返回什么状态码?

导致请求超时返回408

200：请求被正常处理

204：请求被受理但没有资源可以返回

206：客户端只是请求资源的一部分，服务器只对请求的部分资源执行GET方法，相应报文中通过Content-Range指定范围的资源。

301：永久性重定向

302：临时重定向

303：与302状态码有相似功能，只是它希望客户端在请求一个URI的时候，能通过GET方法重定向到另一个URI上

304：发送附带条件的请求时，条件不满足时返回，与重定向无关

307：临时重定向，与302类似，只是强制要求使用POST方法

400：请求报文语法有误，服务器无法识别

401：请求需要认证

403：请求的对应资源禁止被访问

404：服务器无法找到对应资源

500：服务器内部错误

503：服务器正忙

http 状态码

常见状态码：

1xx：请求已被接收，需要继续处理，是一个临时状态码

2xx：请求成功

3xx：重定向

4xx：客户端错误

5xx：服务器内部错误

19 get和post区别?

1）传送方式：get通过地址栏传输，post通过报文传输。

2）传送长度：get参数有长度限制（受限于URL长度），而post无限制。

3）get产生一个TCP数据包；POST产生两个数据包。

对于GET方式的请求，浏览器会把http header和data一并发送出去，服务器响应200（返回数据）；

而对于POST，浏览器先发送header，服务器响应100 continue，浏览器再发送data，服务器响应200 ok（返回数据）。

注：Firefox就只发送一次。

4）get的安全性比post差，传输的数据包含需要保密的信息最好使用post(登录等)，

做数据查询时可以用get,增删改时尽量使用post。get中URL内会包含需要传输的数据，而post会将请求数据封存在数据身体中，用户不可见。

20、udp调用connect有什么作用?

1）UDP调用connect内核仅仅把对端的ip&port记录下来。

2）UDP可以多次调用connect,主要用途：指定新的ip&port连接；断开和之前的ip&port的连接。 指定新连结,直接设置connect第二个参数即可.断开连结,需要将connect第二个参数中的sin_family设置成 AF_UNSPEC即可.

3）UDP使用connect能够提高效率。普通的UDP发送两个报文内核做了如下:#1:建立连结#2:发送报文#3:断开连结#4:建立连结#5:发送报文#6:断开连结。

采用connect方式的UDP发送两个报文内核如下处理:#1:建立连结#2:发送报文#3:发送报文另外一点，每次发送报文内核都由可能要做路由查询.

4）使用connect的UDP发送接受报文可以调用send、write、recv、read等操作，.调用recvfrom,recv,read系统调用只能获取到先前connect的ip&port发送的报文.

IO模型

21、ARP作用

根据目标机器的IP地址，查询目标设备的MAC地址

- 第一步：根据主机A上的ARP缓存内容，IP确定用于访问B主机的转发地址是192.168.1.2。然后A主机在自己本地的ARP缓存中检查主机B的匹配MAC地址。
- 第二步：如果主机A在ARP缓存中没有找到MAC映射，它将询问192.168.1.2的硬件地址，从而将ARP请求帧广播到本地网络的所有主机。源主机A的IP地址和MAC地址以及目标主机IP都包含在ARP请求中。本地网络上的每台主机在接收到ARP请求后，检查自己的IP是否能够匹配上，如果不匹配将丢弃ARP请求。
- 第三步：目标主机B在ARP请求中匹配到了IP，会将主机A的IP和MAC地址添加到自己的本地ARP缓存中
- 第四步：主机B将包含自己MAC地址的ARP回复消息直接发送给主机A.
- 第五步：当主机A收到B的ARP回复消息后，会将B的IP和MAC地址映射更新到本地的ARP缓存中。本地缓存是有生存期的，生存期结束后，将再次重复上面的过程。主机B的MAC地址确定后，A主机就能够向B主机发送IP通信了。

22、ICMP作用

网际控制报文协议（Internet Control Message Protocol）。它是TCP/IP协议族的一个子协议。是一种面向无连接的协议，用于传输出错报告的控制信息。用于在IP主机、路由器之间传递控制信息。ICMP差错报文和ICMP询问报文两种类型。ICMP常用于探测网络联通性。

23、ping过程

- ping 通过发送网际控制报文，验证与另一台计算机的联通性。
- A主机ping B主机时，先在A主机上构建一个ICMP的请求数据包，ICMP协议会将数据包和B主机的IP等信息交到IP层协议。
- IP层收到数据后，会将自己的IP和目标IP、控制信息构建成IP数据包。还需要通过ARP映射表找到B主机的MAC地址。拿到B主机的MAC地址和本机的MAC地址后，一并将数据交给数据链路层，组装成帧，依据以太网的介质访问规则将它们传送出去。
- B主机收到这个数据帧后，检查发送过来的MAC地址是不是本机的，如果是，将数据帧中的IP数据包取出来，交给本机的IP层协议，IP层协议检查完，再将ICMP数据包取出来交给ICMP协议处理，完成后构建一个ICMP应答数据包，回发给主机A，在一定时间内，A收到了应答包，说明A和B之间网络可达。如果没收到，则说明网络不可达。

24、IP分类

- A类地址：以0开头，第一个字节范围：0~127（1.0.0.1-126.255.255.254）
- B类地址：以10开头，第一个字节范围：128~191（128.0.0.1-191.255.255.254）
- C类地址：以110开头，第一个字节范围：192~223（192.0.0.1-223.255.255.254）
- 10.0.0.0-10.255.255.255，172.16.0.0-172.31.255.255，192.168.0.0-192.168.255.255（internet上保留地址用于内部）

25、 在浏览器中输入url地址 ->> 显示主页的过程（面试常客）

过程：

1、客户端浏览器通过DNS解析到www.baidu.com的IP地址202.108.22.5，通过这个IP地址找到客户端到服务器的路径。客户端浏览器发起一个HTTP会话到202.108.22.5，然后通过TCP进行封装数据包，输入到网络层。

2、在客户端的传输层，把HTTP会话请求分成报文段，添加源和目的端口，如服务器使用80端口监听客户端的请求，客户端由系统随机选择一个端口如5000，与服务器进行交换，服务器把相应的请求返回给客户端的5000端口。然后使用IP层的IP地址查找目的端。

3、客户端的网络层不用关心应用层或者传输层的东西，主要做的是通过查找路由表确定如何到达服务器，期间可能经过多个路由器，这些都是由路由器来完成的工作，我不作过多的描述，无非就是通过查找路由表决定通过那个路径到达服务器。

4、客户端的链路层，包通过链路层发送到路由器，通过邻居协议查找给定IP地址的MAC地址，然后发送ARP请求查找目的地址，如果得到回应后就可以使用ARP的请求应答交换的IP数据包现在就可以传输了，然后发送IP数据包到达服务器的地址。

事件顺序

1. 浏览器获取输入的域名www.baidu.com
2. 浏览器向DNS请求解析www.baidu.com的IP地址
3. 域名系统DNS解析出百度服务器的IP地址
4. 浏览器与该服务器建立TCP连接(默认端口号80)
5. 浏览器发出HTTP请求，请求百度首页
6. 服务器通过HTTP响应把首页文件发送给浏览器
7. TCP连接释放
8. 浏览器将首页文件进行解析，并将Web页显示给用户。

涉及到的协议

1. 应用层：HTTP(WWW访问协议)，DNS(域名解析服务)，DNS解析域名为目的IP，通过IP找到服务器路径，客户端向服务器发起HTTP会话，然后通过运输层TCP协

议封装数据包，在TCP协议基础上进行传输

2. 传输层：TCP(为HTTP提供可靠的数据传输)，UDP(DNS使用UDP传输)，HTTP会话会被分成报文段，添加源、目的端口；TCP协议进行主要工作

3. 网络层：IP(IP数据数据包传输和路由选择)，ICMP(提供网络传输过程中的差错检测)，ARP(将本机的默认网关IP地址映射成物理MAC地址)

为数据包选择路由，IP协议进行主要工作，相邻结点的可靠传输，ARP协议将IP地址转成MAC地址。

什么是 MTU

- MTU：Maximum Transfer Unit (最大传输单元)，即该链路层所能允许通过最大数据大小

输入 url 到网页显示的过程

DNS地址解析

TCP三次握手

HTTP请求

HTTP应答

浏览器渲染

TCP四次挥手

ip 和 mac 地址的区别？他们各自有什么用

IP地址划分时基于地理区域，换了不同地方，即便是同一台硬件设备，IP地址一定不一样，可以理解为和地理位置有关；而MAC地址不依赖于地理区域，换了不同地方，只要还是同一台硬件设备，MAC地址就不会变，它只和硬件设备有关。IP地址标识一台计算机接入网络的标识，mac地址唯一标识一个硬件.

数据在数据链路层怎么传输？交换机的作用是什么

在数据链路层，数据是以帧的形式传输的。帧由帧首部、数据部分、帧尾部这三个部分组成。

交换机作用：

建立MAC地址表，交换机能够在MAC地址和端口之间建立映射

转发数据帧

消除冗余回路

连接不同的网络，交换机能够连接不同类型的网络

划分局域网，交换机可以局域网分为多个冲突域，每个冲突域都有独立的网络带宽，因此提高了局域网的网络带宽。

对称加密、非对称加密

- 对称加密就是通信双方约定一个共同的密钥，每次通信都对通信内容用这密钥进行处理 (加密和解密)；非对称加密就是通信双方有各自的公钥和私钥，通信时，使用公钥进行加密，使用私钥解密.

TCP 怎么保证可靠传输

校验和

连接管理

超时重传

流量控制

拥塞控制：慢开始、拥塞避免、快重传、快恢复

UDP 怎么实现可靠传输

由于在传输层UDP已经是不可靠的连接，那就要在应用层自己实现一些保障可靠传输的机制

简单来讲，要使用UDP来构建可靠的面向连接的数据传输，就要实现类似于TCP协议的 超时重传、有序接收、确认应答、滑动窗口流量控制等

目前已经有一些实现UDP可靠传输的机制，比如UDT (基于UDP的数据传输协议)，UDT的主要目的是支持高速广域网上的海量数据传输.