"合格的开发者知道这么做,而优秀的开发者知道为什么那么做"

一. 延迟与带宽

分组从信息源发到的地所需的时间

逻辑或物理通信路径最大的吞吐量

延迟的构成

- 传播延迟 信号传播距离和速度的函数 =>信号传播距离/速度
- 传输延迟 把消息中比特转移到链路中的时间, 是消息长度和链路速率的函数 =>消息长度/链路速度
- 处理延迟 处理分组头部,检查为错误,以及确定分组目标需要的时间 =>路由处理速度
- 排队延迟 到来的分组排队等待处理的时间 =>网络拥堵状况/路由器处理速度

对用户来说超过200ms的延迟,就会感觉到拖拉,超过300ms就会说反应迟钝,超过1s,就会变得不可忍受!尽量把延迟控制在几百ms以内!

- CDN网络=>把内容放在世界各地,让用户从距离最近的地点下载数据,减少延迟

延迟中相当一部分花在了ISP到自己路由器的这段距离中 =>选择延迟最小的ISP

几个常量:

光在光纤中传播速度约 (20万公里/s),光速30万，折射率1.5；

1km 需要传播约 5ms

带宽

光纤的总带宽等于每个信道的数据传输速率乘以可以复用的信道数

网络边缘带宽(终端用户), 带宽很大程度取决于部署技术, 如拨号连接,DSL,各种无线技术,光纤到户,甚至局域网路由器性能等

用户可用带宽取决于客户端与目标服务器之间的最低容量连接(短板理论?)

"最后一公里问题"：用户接入点到运营商的主干路由器，FCC报告光纤入户的平均达到18ms

大多数网站性能的瓶颈是延迟，而不是带宽。

TCP构成

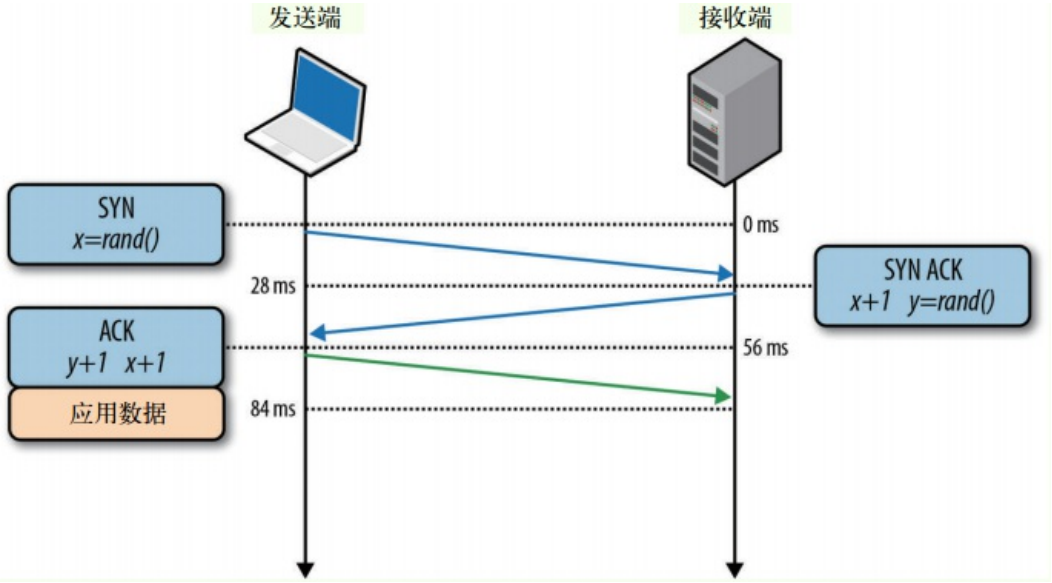
> 因特网有两个核心的协议：IP和TCP。

- IP，即`Internet Protoco`（因特网协议），负责联网主机之间的路由选择和寻址
- TCP，即`Transmission Control Protocol`（传输控制协议），负责在不可靠的传输信道之上提供可靠的抽象层。

TCP/IP也常被称为"因特网协议套件"（Internet Protocol Suite）

> 三次握手

所有的TCP连接一开始都要经过三次握手。客户端和服务器在交换应用数据之前，必须就起始分组序列号，以及其他一些连接相关的细节达成一致。处于安全考虑，序列号由两端随机生成。



SYN

客户端选择一个随机序列号x，并发送一个SYN分组，其中可能还包括其他TCP标志和选项。

SYN ACK

服务器给x加1，并选择自己的一个随机序列号y，追加自己的标志和选项，然后返回响应。

ACK

客户端给x和y加1并发送握手期间的最后一个ACK分组。

其中要记下来的是：

- 为了保证服务端能收接受到客户端的信息并能做出正确的应答而进行前两次(第一次和第二次)握手，
- 为了保证客户端能够接收到服务端的信息并能做出正确的应答而进行后两次(第二次和第三次)握手。

或者可以记忆下中的描述：

- 第一次握手：客户端发送syn包（syn=j）到服务器，并进入SYN_SEND状态，等待服务器确认。
- 第二次握手：服务器端收到syn包，必须确认客户的SYN（ack=j+1），同时自己也发送一个SYN包（syn=k）即SYN+ACK包，此时服务器进入SYN_RECV状态。
- 第三次握手：客户端收到服务器的SYN+ACK包，向服务器发送确认包ACK（ack=k+1），此包发送完毕，客户端和服务器端进入ESTABLISHED状态，完成三次握手。

三次握手完成后，客户端和接收端之间就可以通信了。在每次传输应用数据之前，都必须经历一次完整的往返。因此，**提高 TCP 应用性能的关键，在于想办法重用连接。**

> TCP快速打开
>
> 连接并不是想重用就可以重用。TFO(TCP Fast Open) 是可以减少新建TCP连接带来的性能损失。Linux 3.7 及以后的内核已支持在客户端和服务端支持 TFO，但 TFO 并不能解决所有问题，它虽然有助于减少三次握手的往返时间，但只能在某些情况下有效，比如只能发送某些类型的HTTP请求、只能用于重复的连接等。

拥塞预防机制

流量控制

预防发送端过多的向接收端发送数据的机制。为实现流量控制，TCP 的两端都要通告自己的接收窗口（rwnd）。如果一端跟不上数据传输，就会向发送端通告一个较小的窗口，因此接收的一端更可能成为性能瓶颈。

窗口缩放

> 最初的 TCP 规范分给通告窗口大小的字段是16位，相当于设定了发送端和接收端窗口的最大值为 65535 字节，在这个限制内通常无法获得最佳性能。为解决这个问题，RFC 1323 提供了 “TCP窗口缩放”（TCP Window Scaling）选项，可以将窗口大小提升到 1G 字节，这在三次握手期间完成，当中有一个值表示在将来的 ACK 中左移16位窗口字段的位数。
>
> TCP 窗口缩放机制在主要平台上都是默认弃用的，但中间节点和路由器可以重写或去掉这个选项。因此如果服务器或客户端连接不能完全利用现有的带宽，往往应该先查一查窗口大小。Linux 可通过下述命令查询：

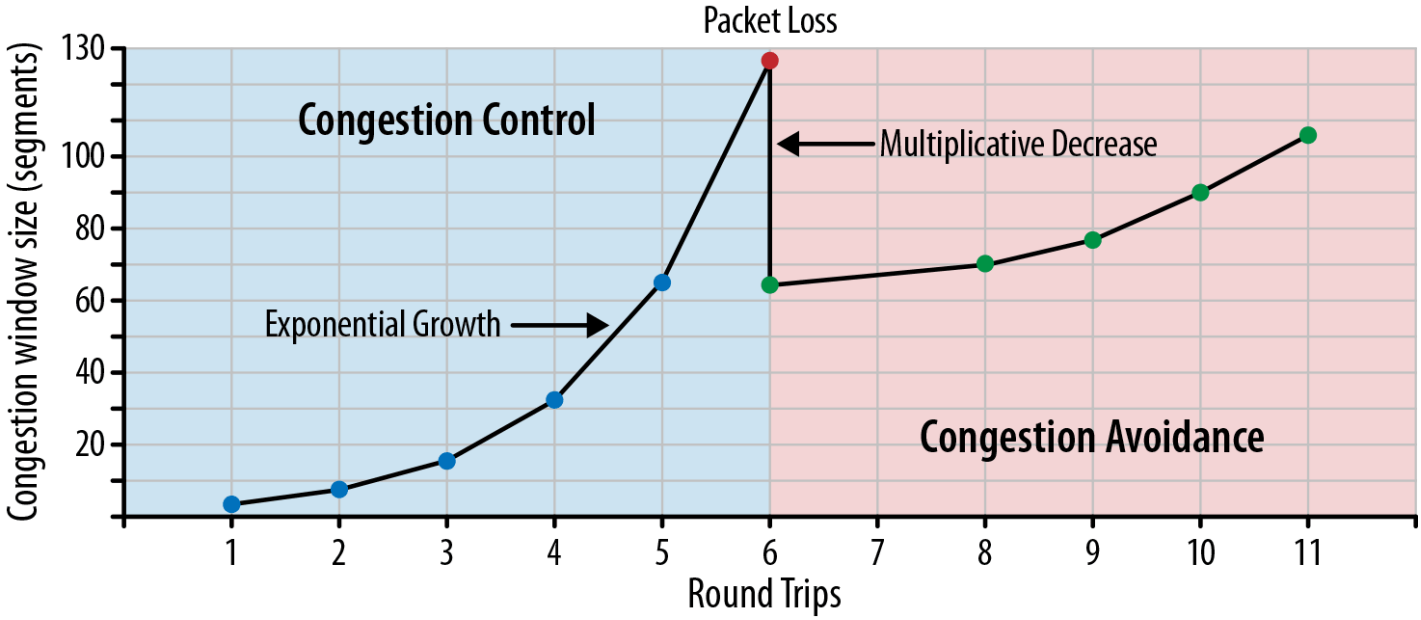
```
$> sysctl net.ipv4.tcp_window_scaling
$> sysctl -w net.ipv4.tcp_window_scaling=1
```

慢启动

流量控制可以防止发送端向接收端发送过多的数据，但在连接建立之初，无法知道可用宽带是多少，因此需要一个估算机制，且需要根据网络中不断变化的条件而动态改变速度。解决该问题的几种算法：**慢启动、拥塞预防、快速重发和快速恢复**。

慢启动的设计思路是根据交换数据来估算客户端与服务器之间的可用带宽。TCP 连接时初始化一个拥塞窗口（cwnd）变量，表示发送端从客户端接收确认（ACK）之前可以发送的数据量的限制。同时发送端不会通告 cwnd 变量，发送端只是维护这么一个私有变量。

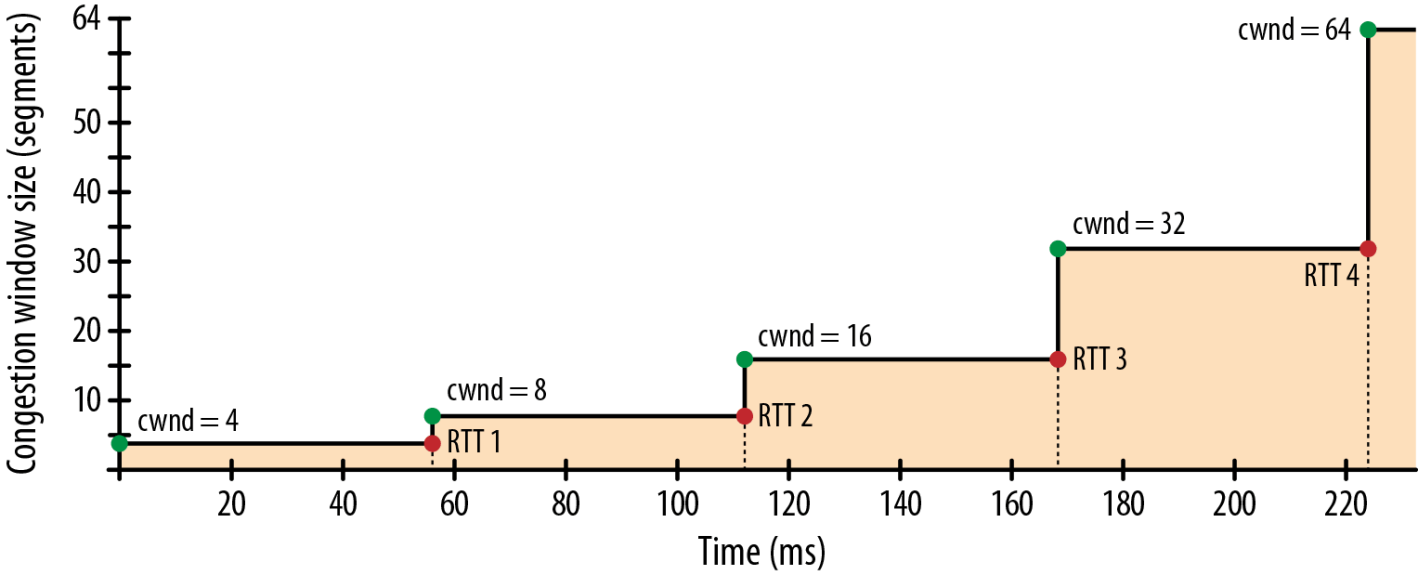
此时，客户端与服务端之间最大可以传输（未经ACK确认的）数据量取 rwnd 和 cwnd 中的最小值。在分组被确认之后，增大窗口大小（每收到一个 ACK，慢启动算法就告诉服务器可以将它的 cwnd 增加一个 TCP 段），慢慢的启动。因为每接收一个 ACK 就增加一个 TCP 段，所以是指数增长。



> TCP慢启动算法示意图

一个 TCP 端的大小是 1460字节 MTU 大小为1500，其中 ip 头为 20，tcp 头 20，所以 maximum segment size 为 1460 字节。1 字节为 8 位二进制。

无论宽带多大，每个 TCP 连接都必须经过慢启动阶段，因此不可能一上来就利用连接的最大带宽。例如接收窗口最大为 65535≈64kb≈45段TCP，则要达到这个窗口经过 4 次往返，这可能会带来上百毫秒的延迟。



> 拥塞窗口大小增长示意图，经4次往返才能达到最大接收窗口

慢启动限制了可用了的吞吐量，对于许多 HTTP 连接，特别是一些短暂、突发的连接，例如对小文件的传输来说非常不利。

慢启动重启 TCP 还实现了 SSR（Slow-Start Restart 慢启动重启）机制，这种机制会在连接空闲一段时间后重置连接的拥塞窗口。因为在连接空闲的同时，网络状况也可能发生了变化，为避免拥塞，理应将拥塞窗口重置回“安全的”默认值。SSR 对于会出现突发空闲的长周期 TCP 连接（如 HTTP 的 keep-alive 连接）有很大的影响，因此建议在服务器上禁用SSR，Linux平台检查和关闭的命令为：

```
$> sysctl net.ipv4.tcp_slow_start_after_idle
$> sysctl -w net.ipv4.tcp_slow_start_after_idle=0
```

队首阻塞

TCP 在不可靠的信道上实现了可靠的网络传输，虽然 TCP 很流行，但其并不是唯一的选择，在某些情况下也不是最佳选择，特别是按序交付和可靠交付有时候并不必要，反而会导致额外的延迟，对性能造成负面影响。

为了让分组按照顺序传送到接收端，如果中途有一个分组没能到达接收端，那么后续分组必须保存在接收端的 TCP 缓冲区，等待丢失的分组重发并到达接收端，这一切都发生在 TCP 层，应用程序对 TCP 的重发和分组缓存排队一无所知，需等到分组全部到达才能访问数据，在次之前应用程序只能在套接字读数据时感到延迟交付。这种效应称为 TCP 的队首（HOL，Head of Line）阻塞。

> 丢包就丢包 事实上丢包是让 TCP 达到最佳性能的关键，被删除的包正是一种反馈机制，能够让接收端和发送端各自调整速度，以避免网络拥堵，同时保持延迟最短。

TCP优化

> 把服务器内核升级到最新版本（Linux：3.2+）

TCP的最佳实践以及影响其性能的底层算法一直在与时俱进，而且大多数变化都只在最新内核中才有实现，一句，让你的服务器跟上时代是优化发送端和接收端TCP栈的首要措施，反正好就是了，如果有条件就升级。

> 确保cwnd和rwnd大小为10

Linux2.6.38默认`initcwnd`和`initrwnd`是3和10。它允许14.2KB初始值发送，等接收的返回后才开始缓慢增长。这比HTTP和SSL重要，因为它可以让你在头部初始设置更多空间的数据包。具体可以使用uname -a命令查看您自己的linux内核版本，如果小于3.2可以执行`ip route | while read p; do ip route change $p initcwnd 10 initrwnd 10; done`;更改。

> 禁用空闲后的慢启动

TCP实现了SSR慢启动重启机制。这种机制会在连接空闲一定时间后重置连接的拥塞窗口。道理很简单，在连接空闲的同时，网络状况也可能发生了变化，为了避免拥塞，理应将拥塞窗口重置回“安全的”默认值。

毫无疑问，SSR对于那些会出现突发空闲的长周期TCP连接（比如HTTP的keep-alive连接）有很大影响，因此，我们建议在服务器上禁用SSR。在

Linux平台，可以通过如下命令来检查和禁用SSR：

```
$> sysctl net.ipv4.tcp_slow_start_after_idle $> sysctl -w net.ipv4.tcp_slow_start_after_idle=0
```

确保启动窗口缩放（RFC1323）

启动窗口缩放可以增大最大接收窗口大小，可以让高延迟的连接达到更好的吞吐量，在Linux中，可以通过如下命令检查和启用窗口缩放选项：

```
$> sysctl net.ipv4.tcp_window_scaling $> sysctl -w net.ipv4.tcp_window_scaling=1
```

减少传输冗余数据

这个在没看TCP原理之前还没怎么感觉，看过之后感觉这个太重要了，举个例子TCP每次传输都需要三次握手也就是说每发送一次请求之前都要增加一次往返时间，可以想象每次请求的代价有多大，能减少请求尽量少请求，这个很重要！！

压缩要传输的数据

这个同上节省资源也很重要，多个文件压缩成一个还重用了已经建立的TCP连接

把服务器放到离用户近的地方以减少往返时间

减少传输时间，减少延迟

UDP

分组和数据报区别：

- 分组：任何格式化数据块
- 数据报：通过不可靠服务传输的分组

私有地址：

```
10.0.0.0~10.255.255.255

172.16.0.0~172.31.255.255

192.168.0.0~192.168.255.255
```

udp与nat设备冲突：

- 1.连接状态超时删除，tcp和udp都面临类似问题

解决办法：双向keepalive分组，周期性重置nat设备的计时器

- 2.对于p2p,voip,游戏和文件共享，需要端到端的双向通信，upd方式无法穿透nat设备

解决方案：STUN,TURN,ICE

谷歌实现的基础库：libjingle, 用于gtalk

TLS

- 1.功能：加密，完整性，身份验证
- 2.TLS握手：6次传输，3倍RTT，包括tcp握手，hello密钥套件协商，密钥生成和验证

对称加密和非对称加密的性能对比：

```
openssl speed rsa

openssl speed aes
```

- 3.SNI 机制，解决server 单ip支持多host https
- 4.TLS会话恢复机制，复用协商的对称密钥，减少一次rtt，实现机制包括：会话标示符（ID和协商的参数），保存在server端，存储管理瓶颈；会话记录单（服务器密钥加密过的数据），保存在客户端

优化建议：

- 1.tcp的全部优化办法
- 2.openssl最新库，性能明显优化
- 3.尽早完成握手，cdn与客户端建立tls
- 4.会话缓存和无状态恢复
- 5.tls记录大小，ipv4 20byte(ipv6 40字节)，tcp 20byte（tcp option 40字节），tls记录1420字节比较合适
- 6.tls压缩，禁用tls协商的压缩选项
- 7.证书链长度，减少链长度，避免慢启动阶段超过默认拥塞窗口大小，删除不必要证书
- 8.OCSP封套，证书链包括OCSP响应，让浏览器跳过在线查询
- 9.HTTP严格传输首部