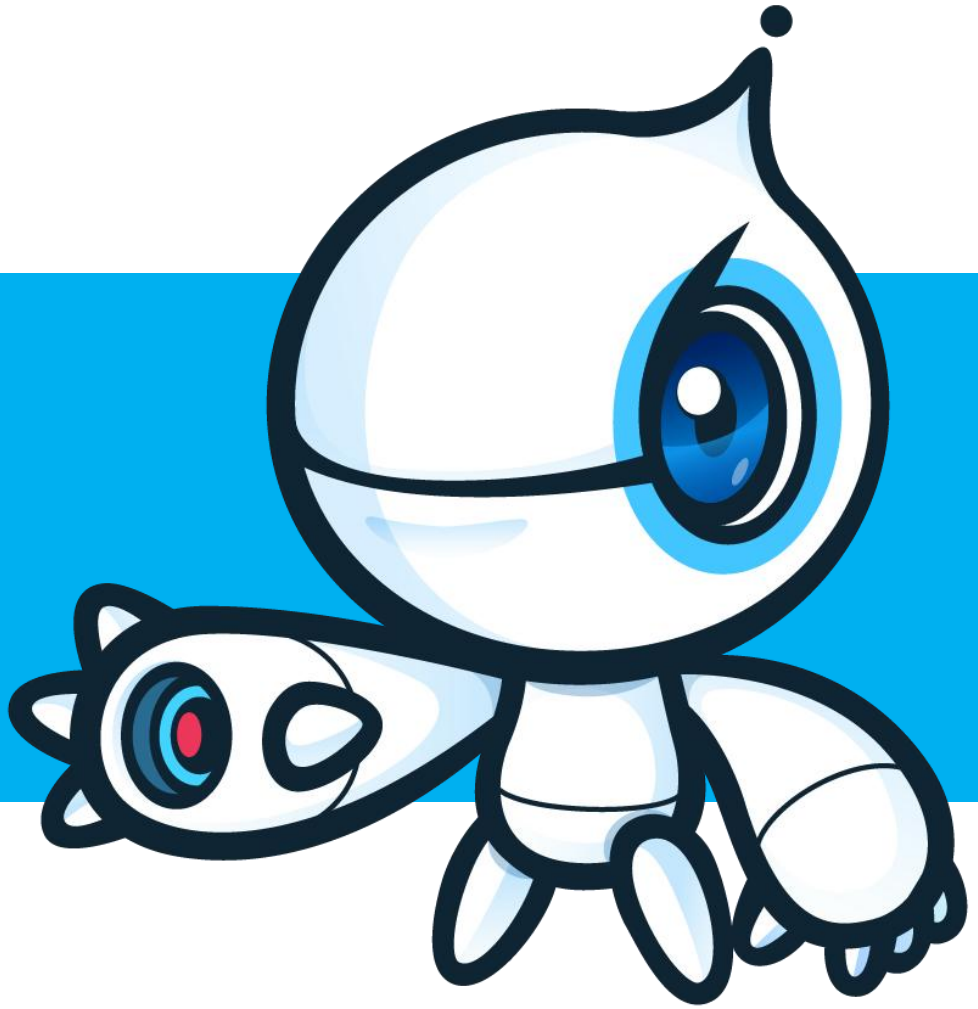
车轮互联
WWW.CHELUN.COM

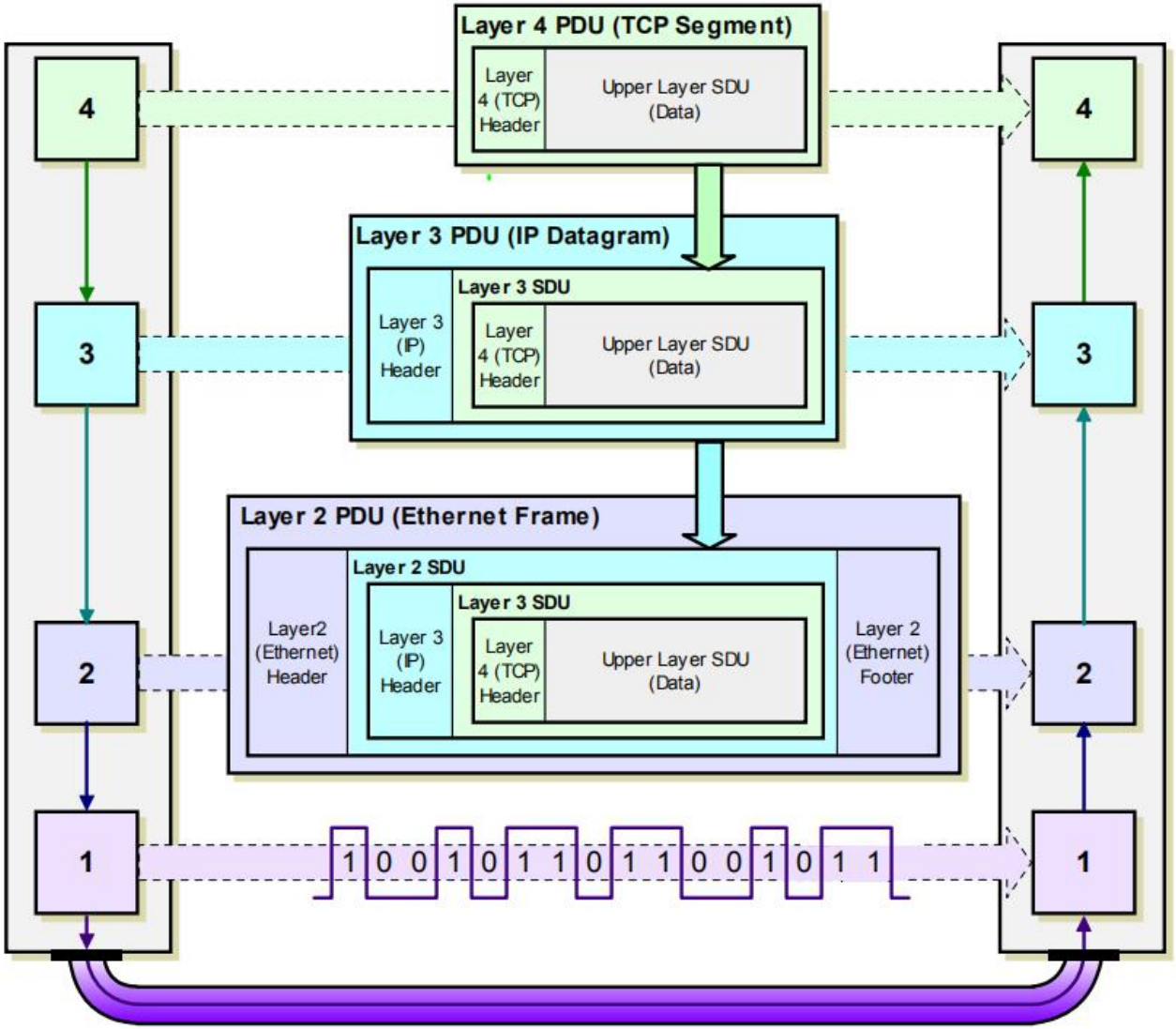
TCP那些事



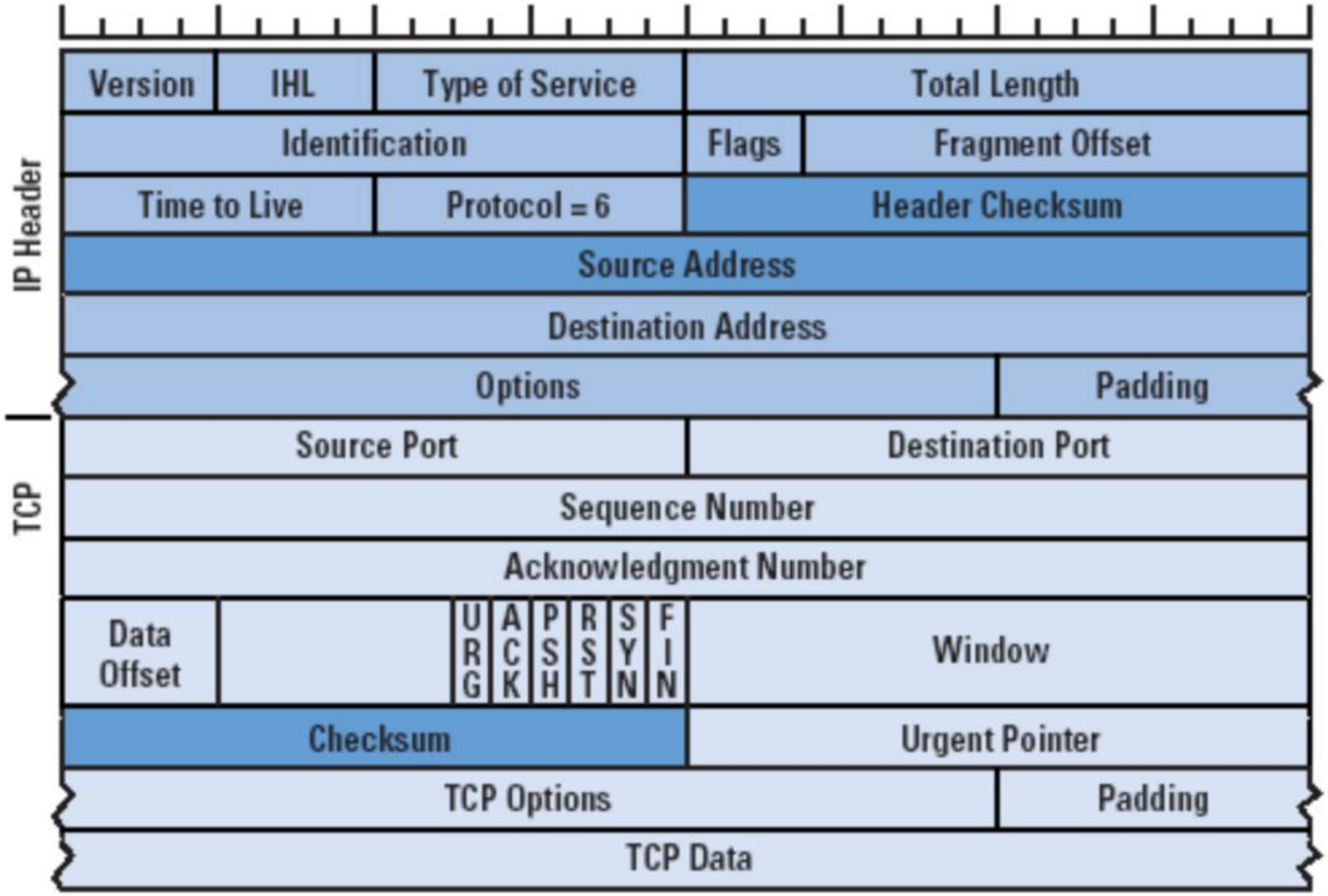
技术研发部——饶超勋

目录

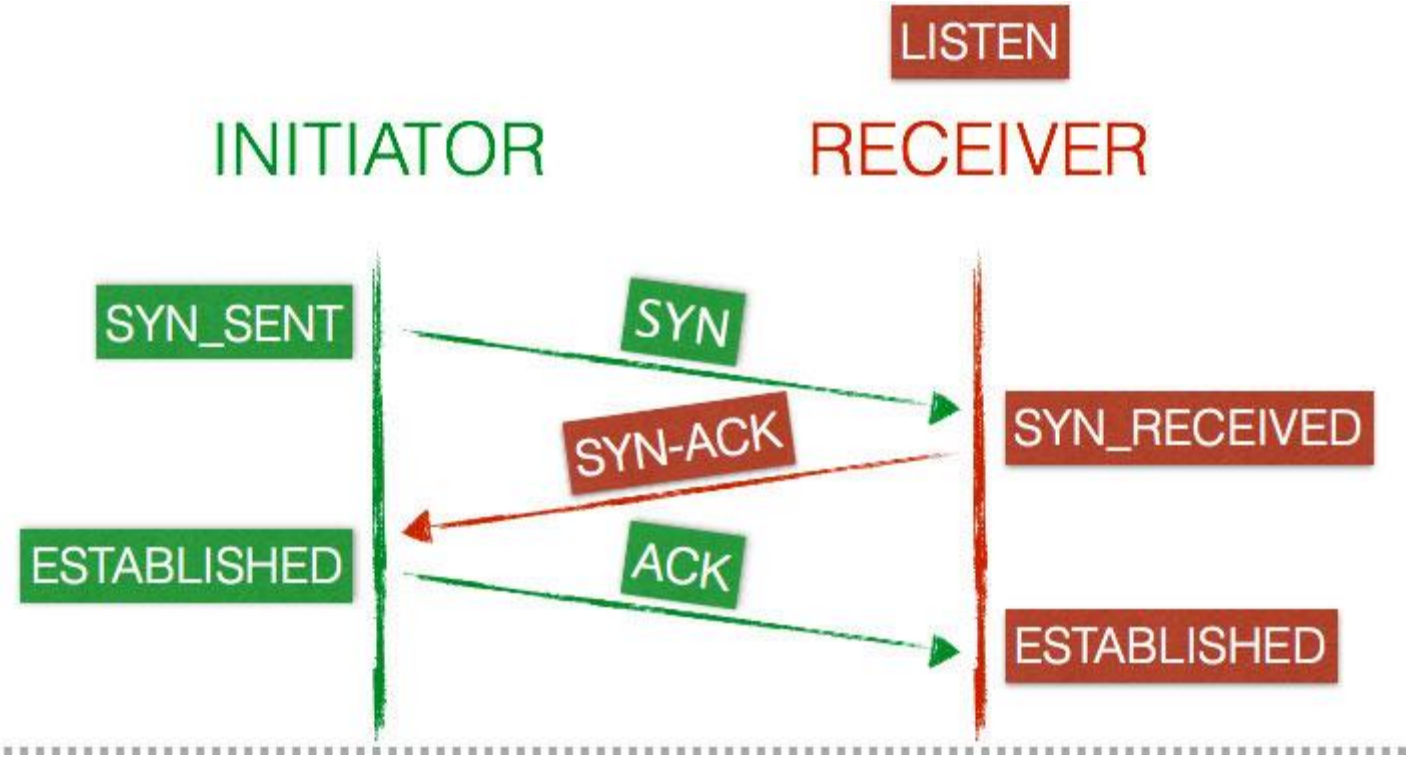
TCP概念——OSI模型



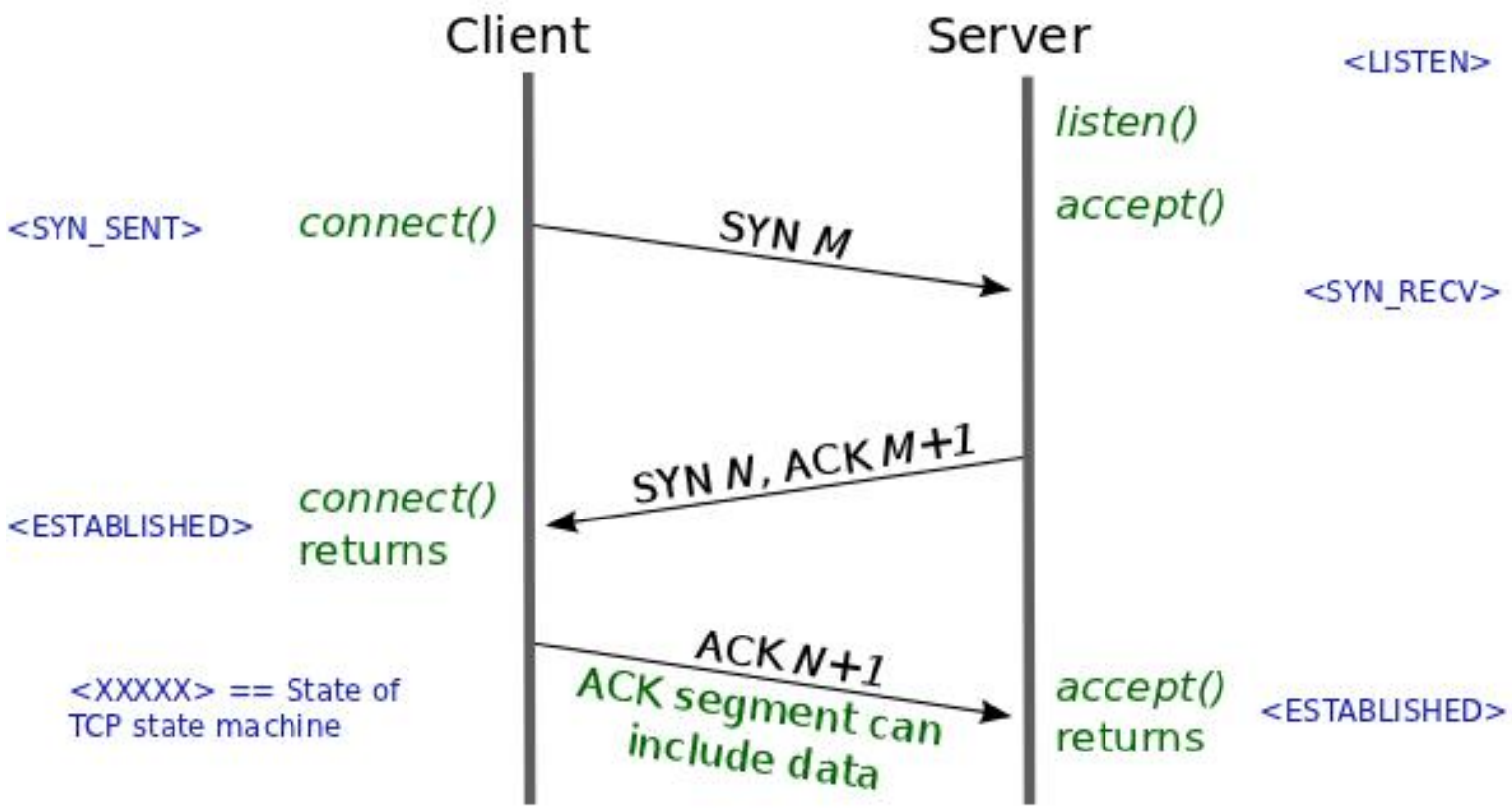
TCP概念——TCP包头



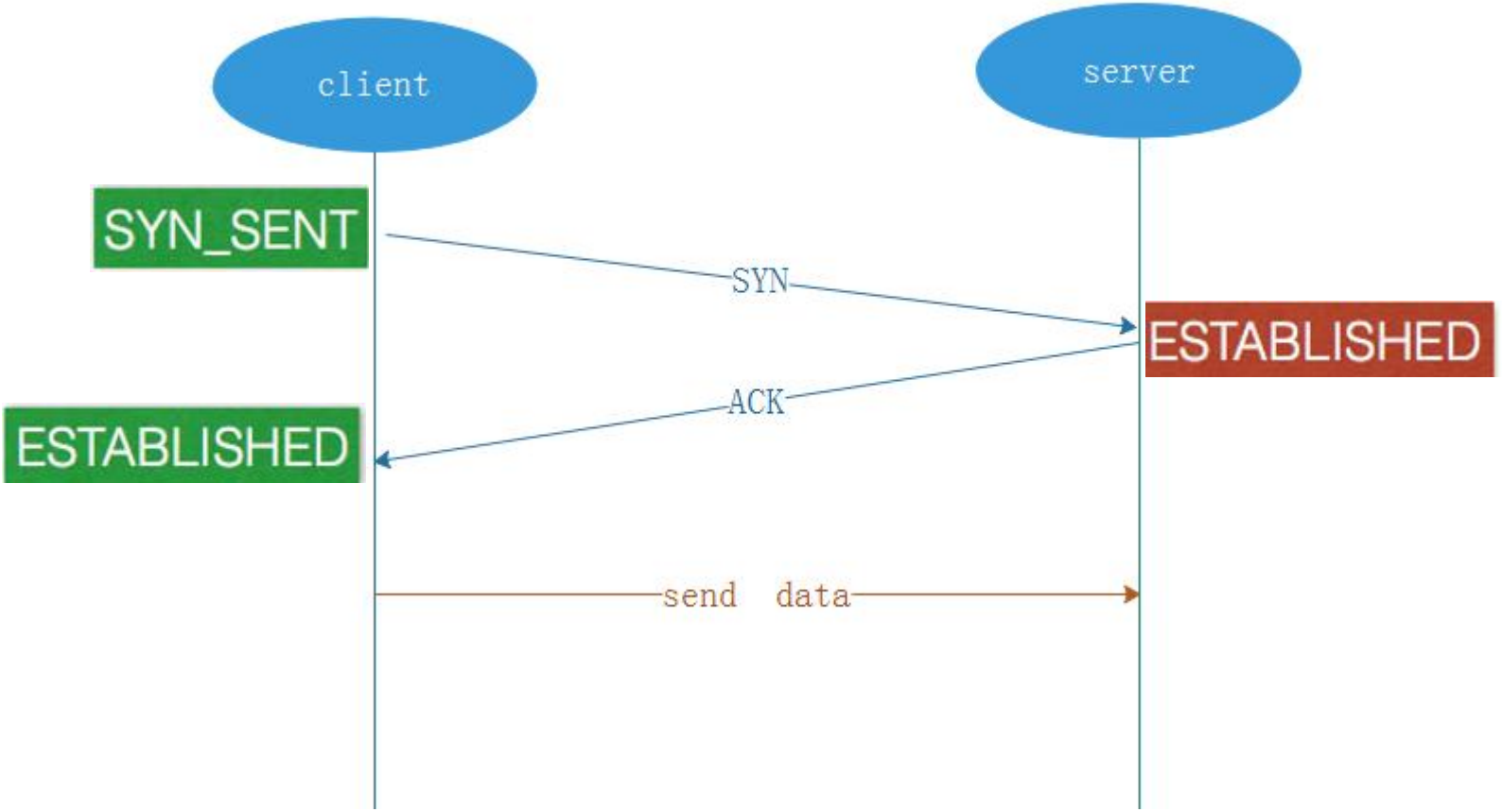
连接阶段——三次握手



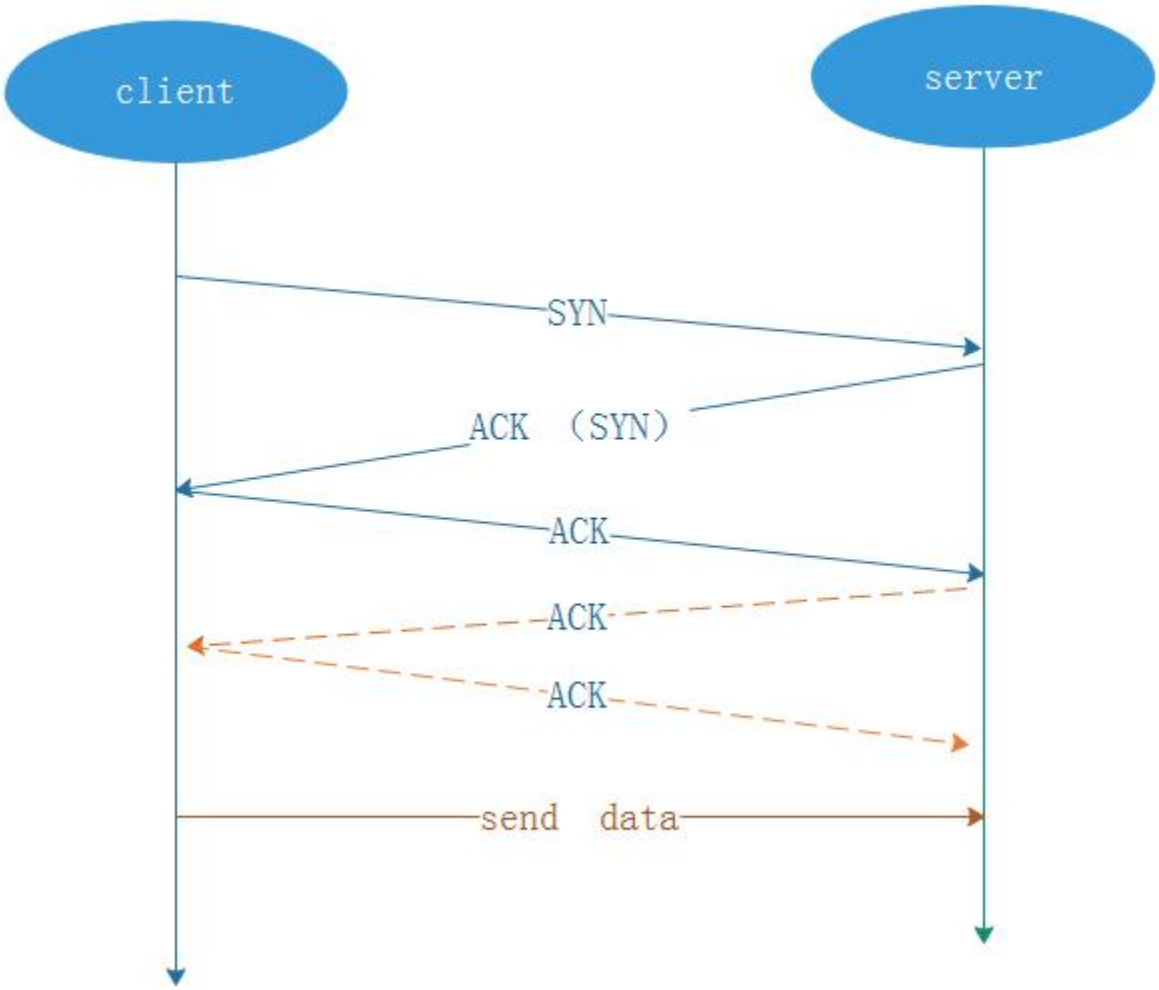
连接阶段——三次握手



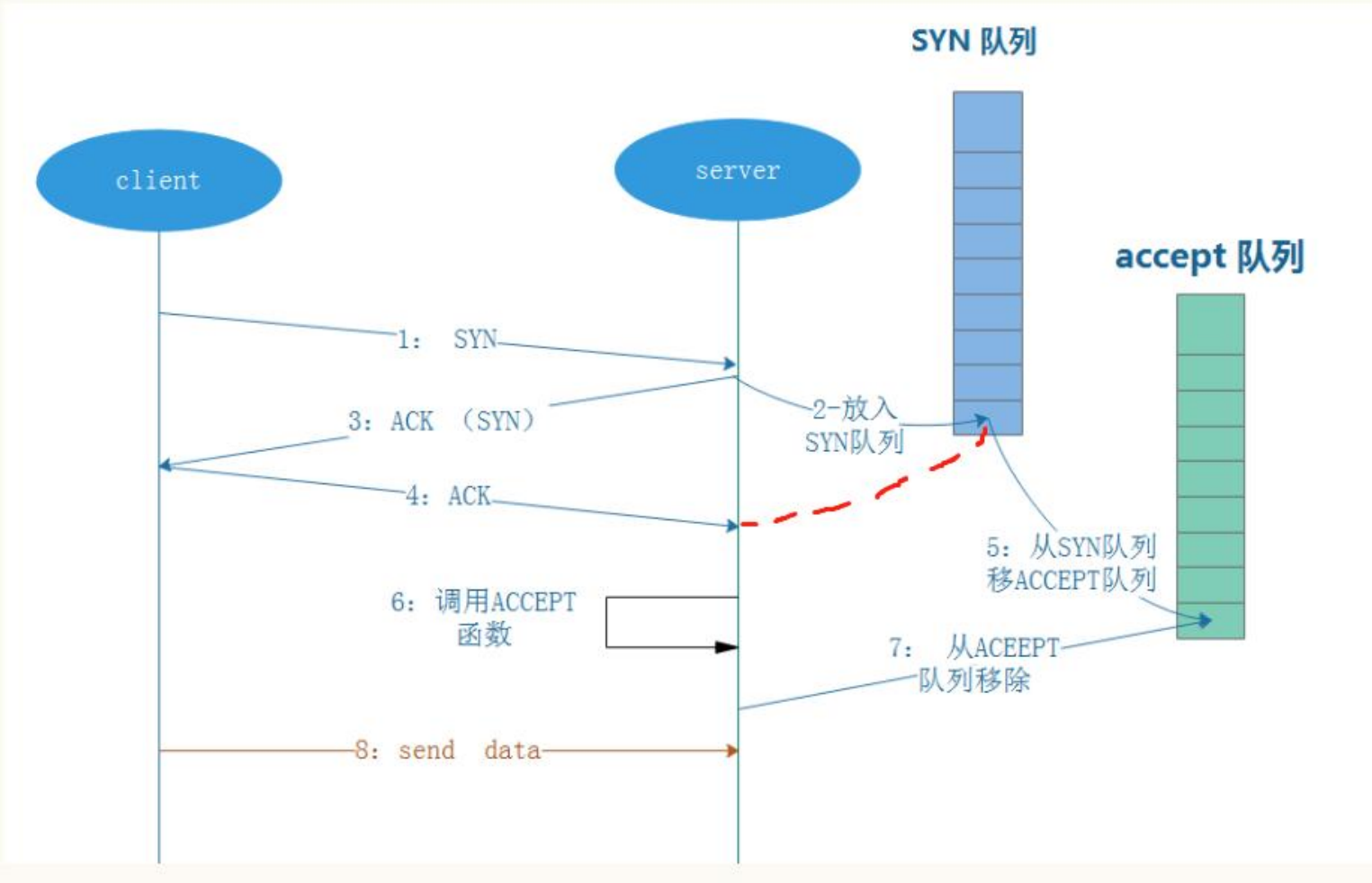
连接阶段——二次握手？



更多次

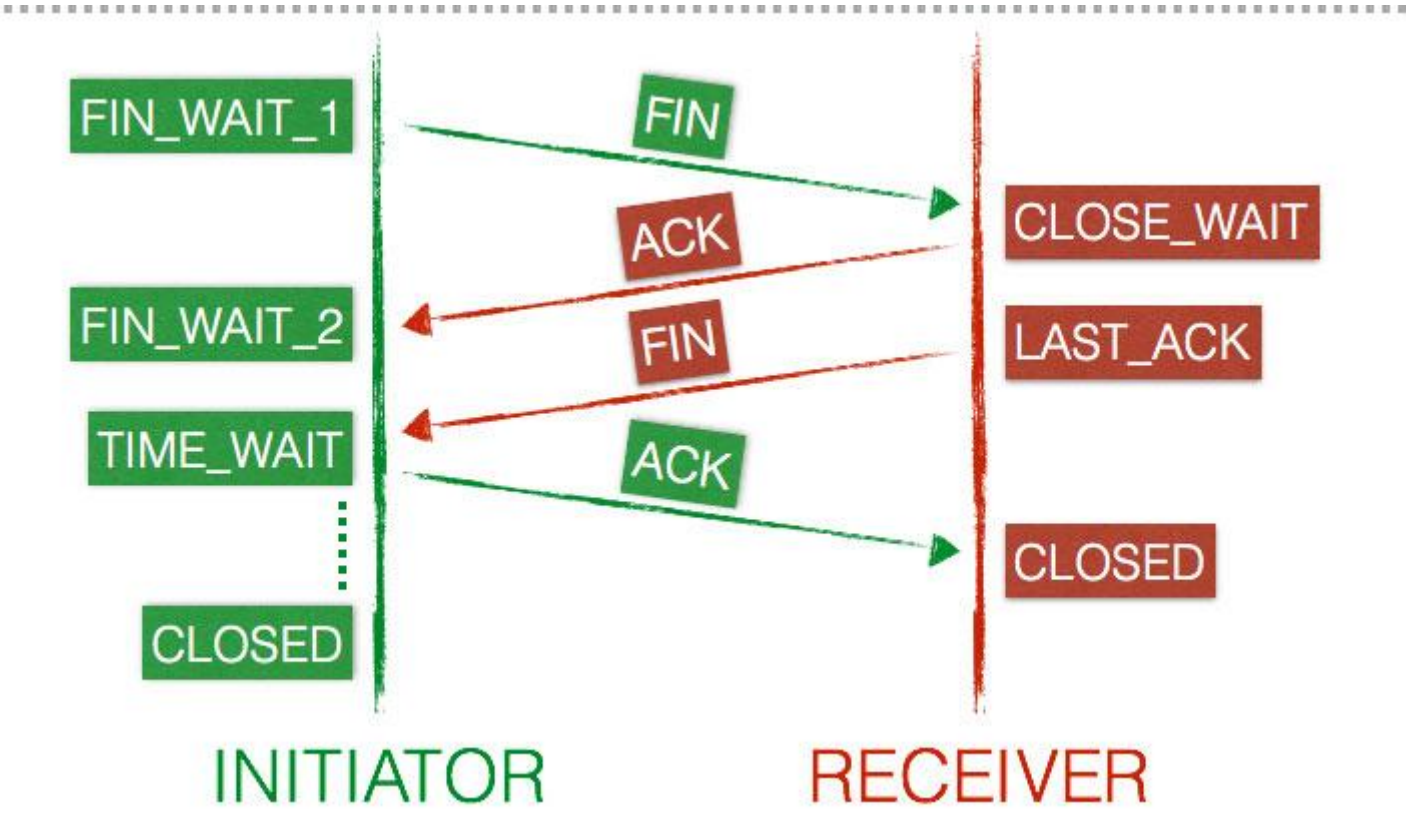


连接阶段——连接队列&半连接

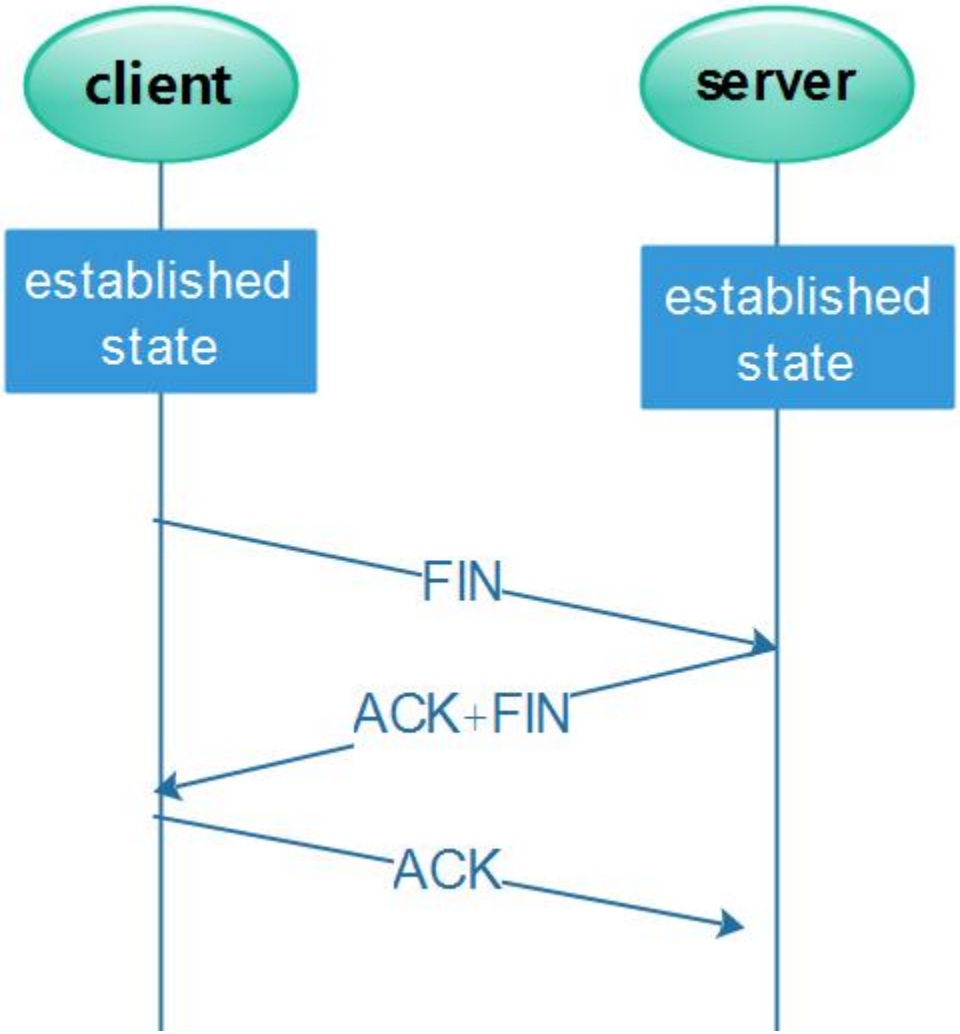


目录

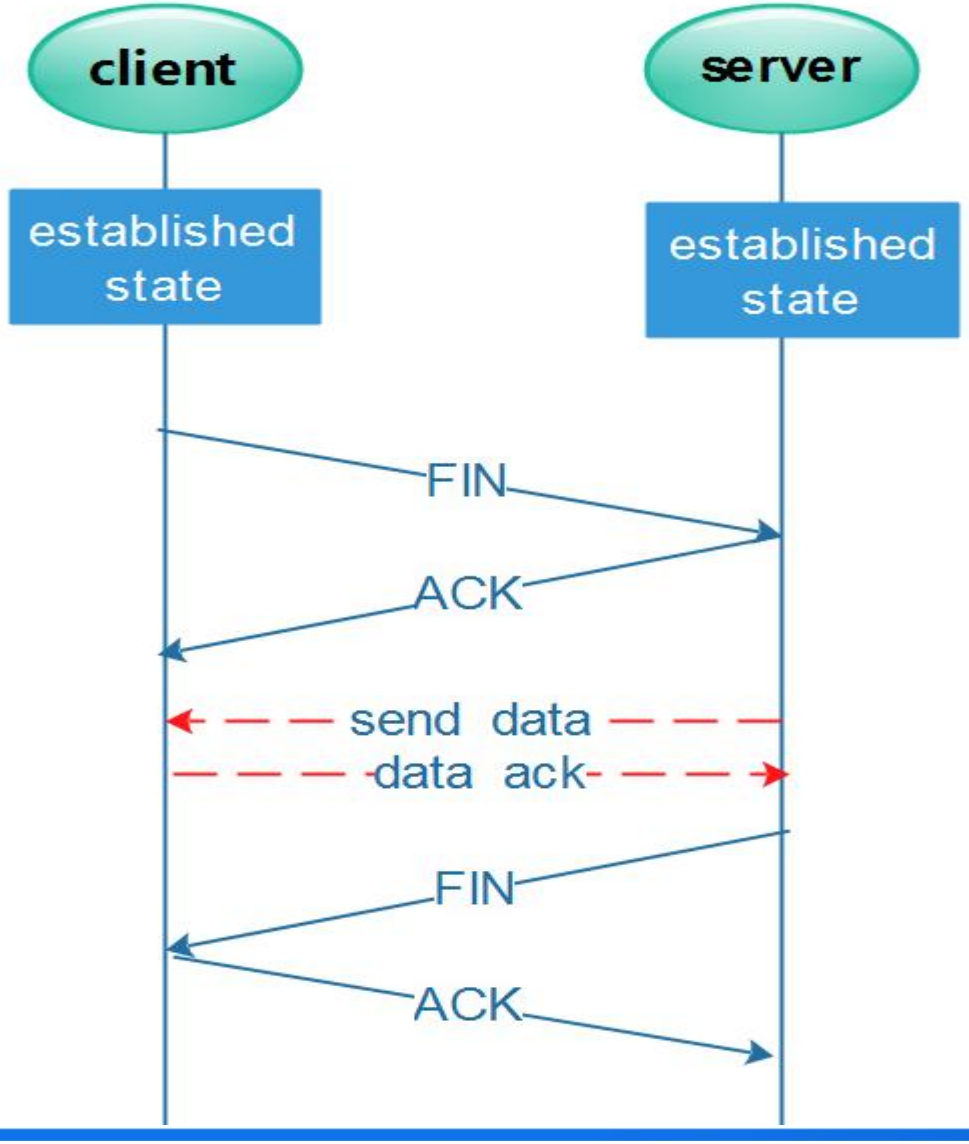
关闭阶段——四次挥手



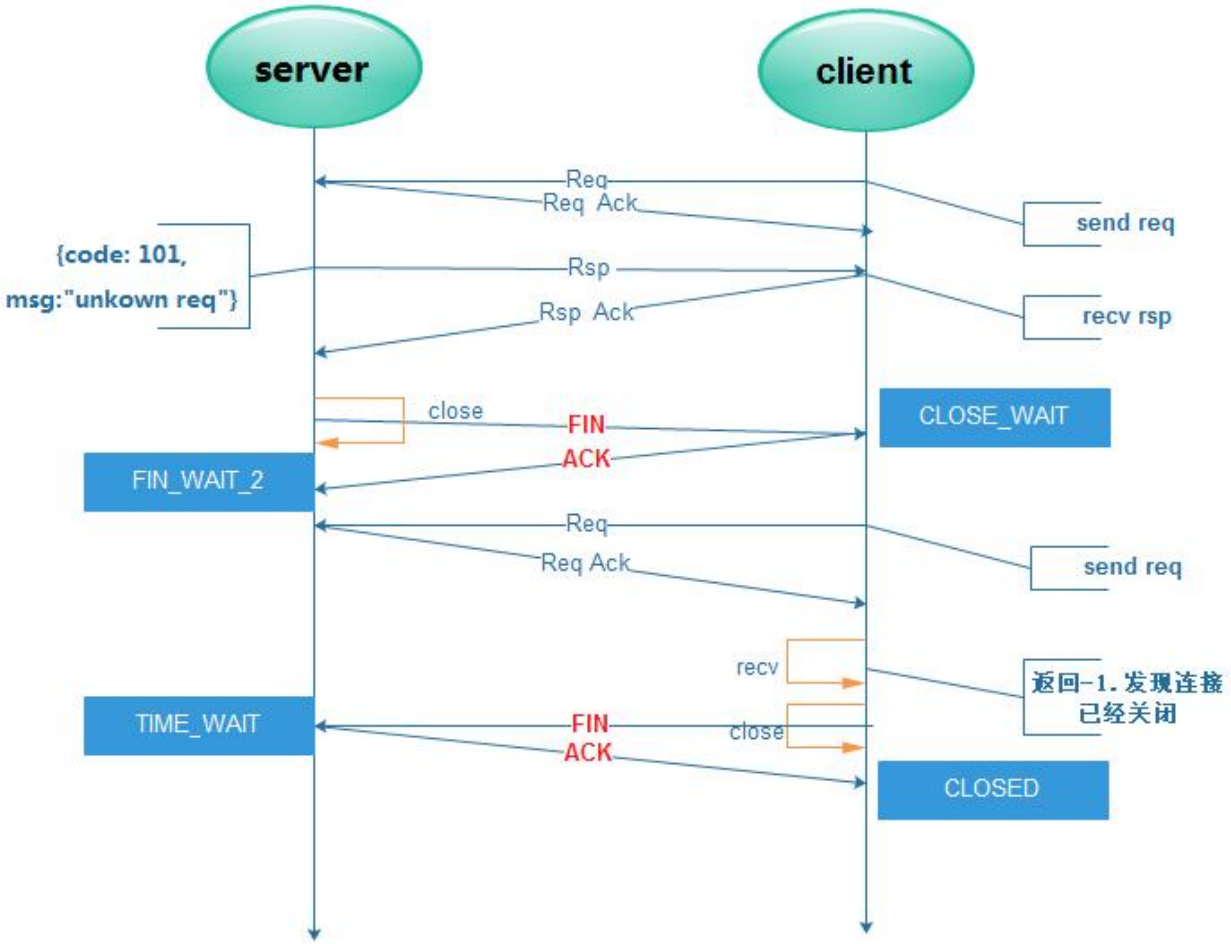
关闭阶段——三次挥手?



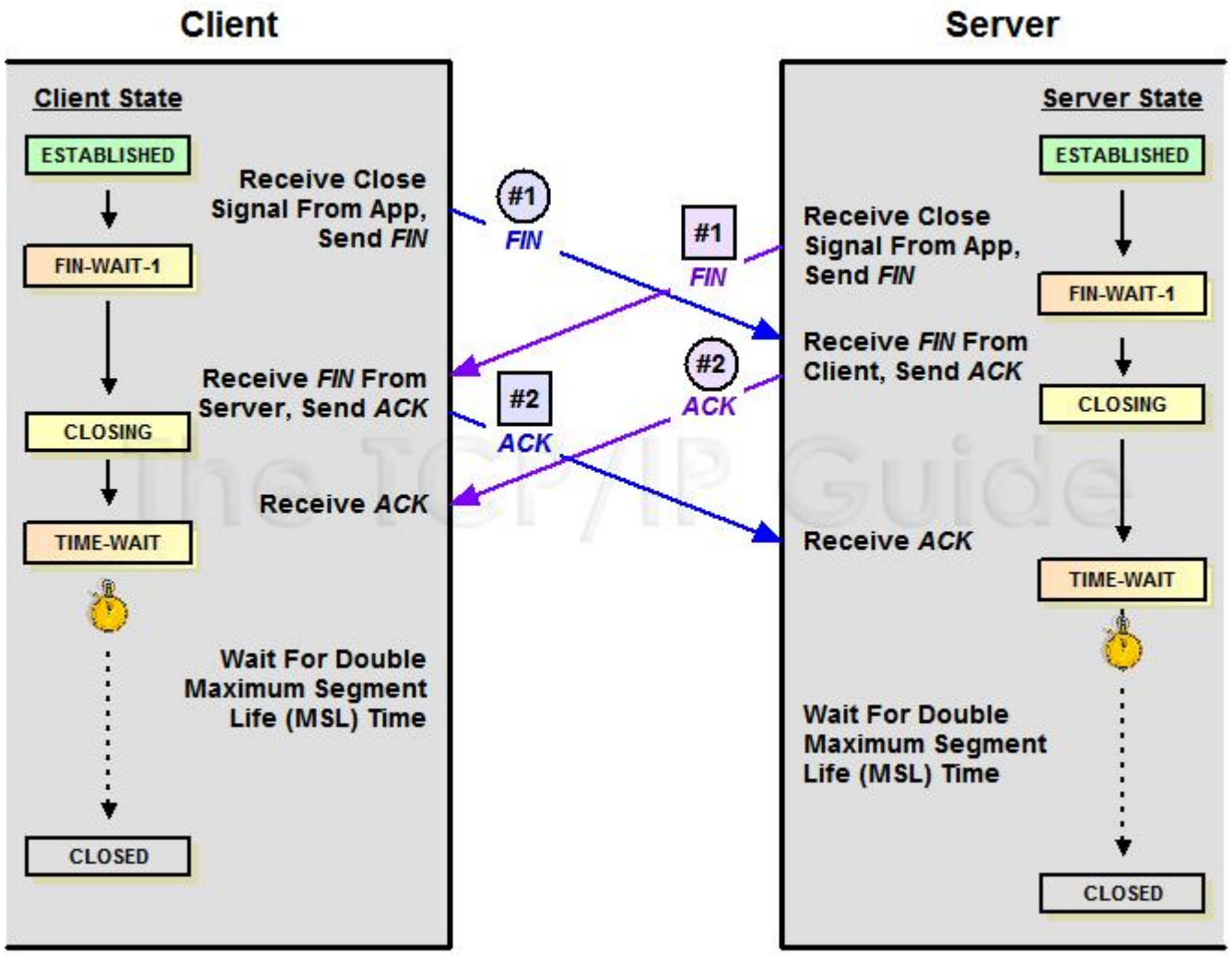
关闭阶段——半关闭



关闭阶段——半关闭引发问题



关闭阶段——同时关闭


Client
Server
Client State
Server State
ESTABLISHED
ESTABLISHED
Receive Close Signal From App, Send FIN
#1
FIN
#1
FIN
Receive Close Signal From App, Send FIN
FIN-WAIT-1
FIN-WAIT-1
Receive FIN From Client, Send ACK
#2
ACK
#2
ACK
Receive FIN From Server, Send ACK
CLOSING
CLOSING
Receive ACK
Receive ACK
TIME-WAIT
TIME-WAIT
Wait For Double Maximum Segment Life (MSL) Time
Wait For Double Maximum Segment Life (MSL) Time
CLOSED
CLOSED

关闭阶段——CLOSE_WAIT&TIME_WAIT

CLOSE_WAIT 和 TIME_WAIT 危害都很大：

1. CLOSE_WAIT: 连接资源没有释放，**占用句柄**和**很多内存**，**端口，而且没有超时机制。**

2. TIME_WAIT：**占用端口**时间长 2*MSL。基本上不占用内存，句柄资源都释放了。

CLOSE_WAIT 相对于TIME_WAIT，**危害更大**，但是因为其发生的概率非常低。因此日常开发中很少碰到，即使发生也不会出现大量，可以忽略掉。

TIME_WAIT 问题却非常普遍，因为在服务后台开发CGI机器，存在**大量并发连接的短连接。调用方都是主动关闭**。

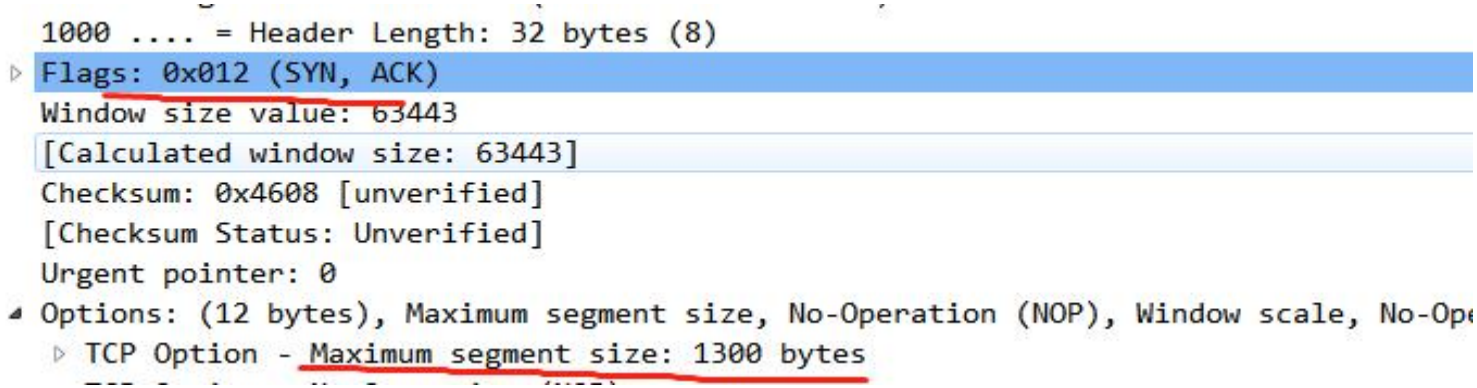
目录

传输阶段——分包问题

MTU：Maximum Transmission Unit最大传输单元。L2链路层的限制。 以太网最大包长为1500. 减去L3层 IP协议头 20字节。减去L4层 TCP包头20. 所以对TCP来说 MTP 就是 max = 1500 - 20 -20 = 1460 。

MSS：Maximum Segment Size 。L4层的限制。TCP层发送最大尺寸。不是固定的。有发送方和接收方一起协商。从MTP和MSS定义可以看出他们关系 MSS 永远要小于MTU。

业务层是不关心发送包的大小的。但是由于MSS限制，所以到了TCP层。会被拆分成MSS包大小一个个发出。



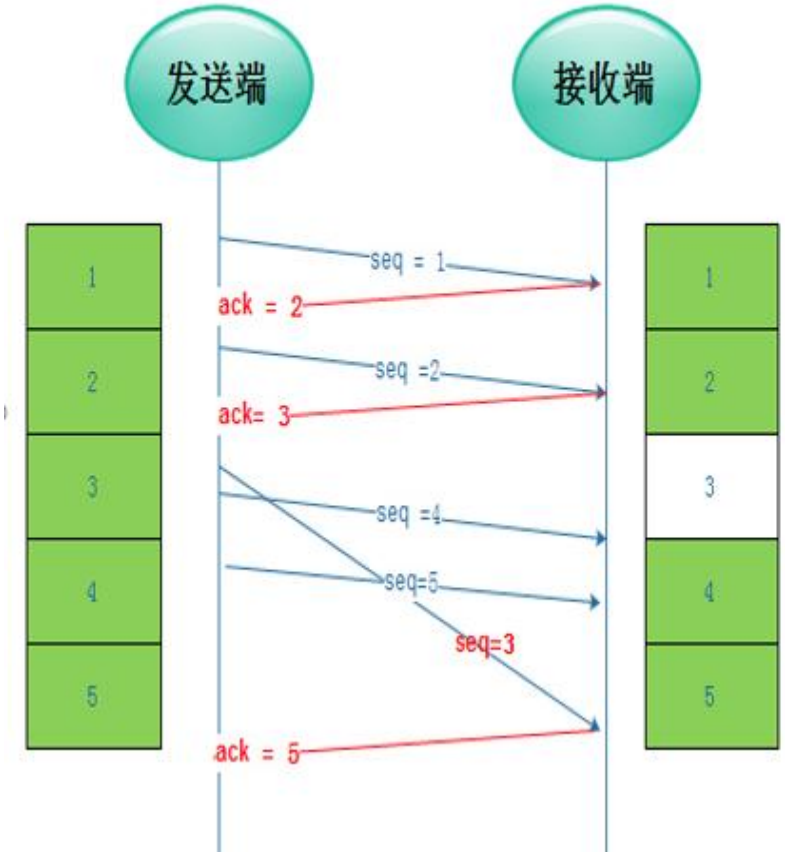
传输阶段——延迟问题/乱序重组

由于网络路由问题。先发的包可能会比后发的包后到达。这种延迟导致一个问题——乱序。

TCP是如何解决乱序问题呢。通过seq，ack机制。

seq（Sequence number）：发送数据包的编号，用来保证数据的顺序，数据包可能会先发送的，但是后到达。给每个数据包在发送时按顺序编上号码，接收者按照seq 重新排序。就保证了正确性。

ack（Acknowledgment number）：期望接收下一个数据包的编号。 用来保证包不丢失。发送者通过这个确认对方到底是否收到包，决定是否丢失，进行重传



传输阶段——并包/粘包问题

Nagle算法(纳格算法)： 为了避免发送大量的小包，防止小包泛滥于网络。把多次发送的小包合并成一次发送。提高传输效率。

TCP_CORK 选项：禁止小报文发送，合并成一个大报文(<=MSS)发送出去

两者相似: 都是会合并小报文为一个大报文，一次发出去；

两者区别: TCP_CORK 更激进，效率更高。

Nagle vs **TCP_CORK**

（1）如果包长度达到MSS，则允许发送；

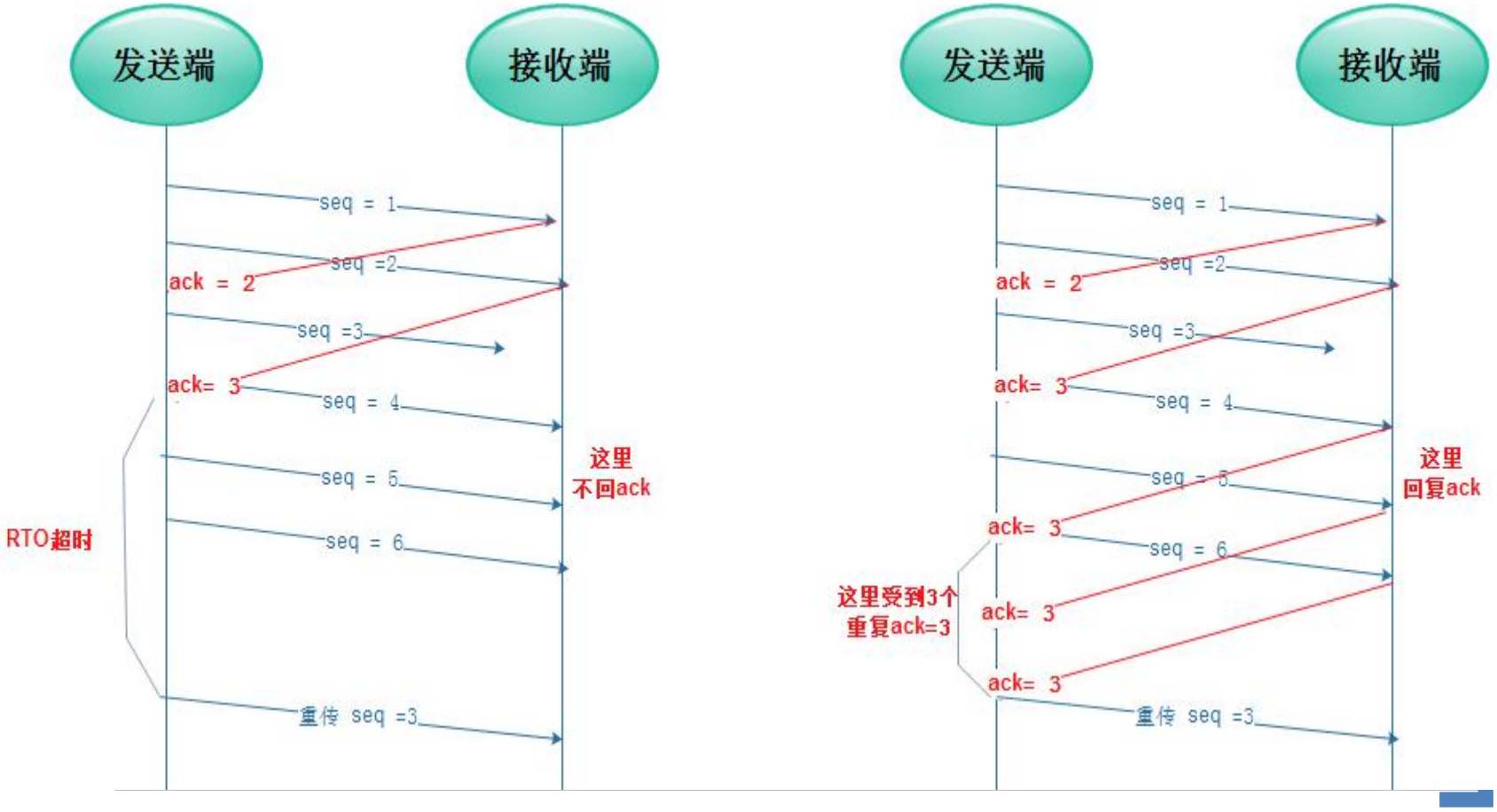
（2）如果该包含有FIN，则允许发送；

（3）设置了TCP_NODELAY选项，则允许发送；

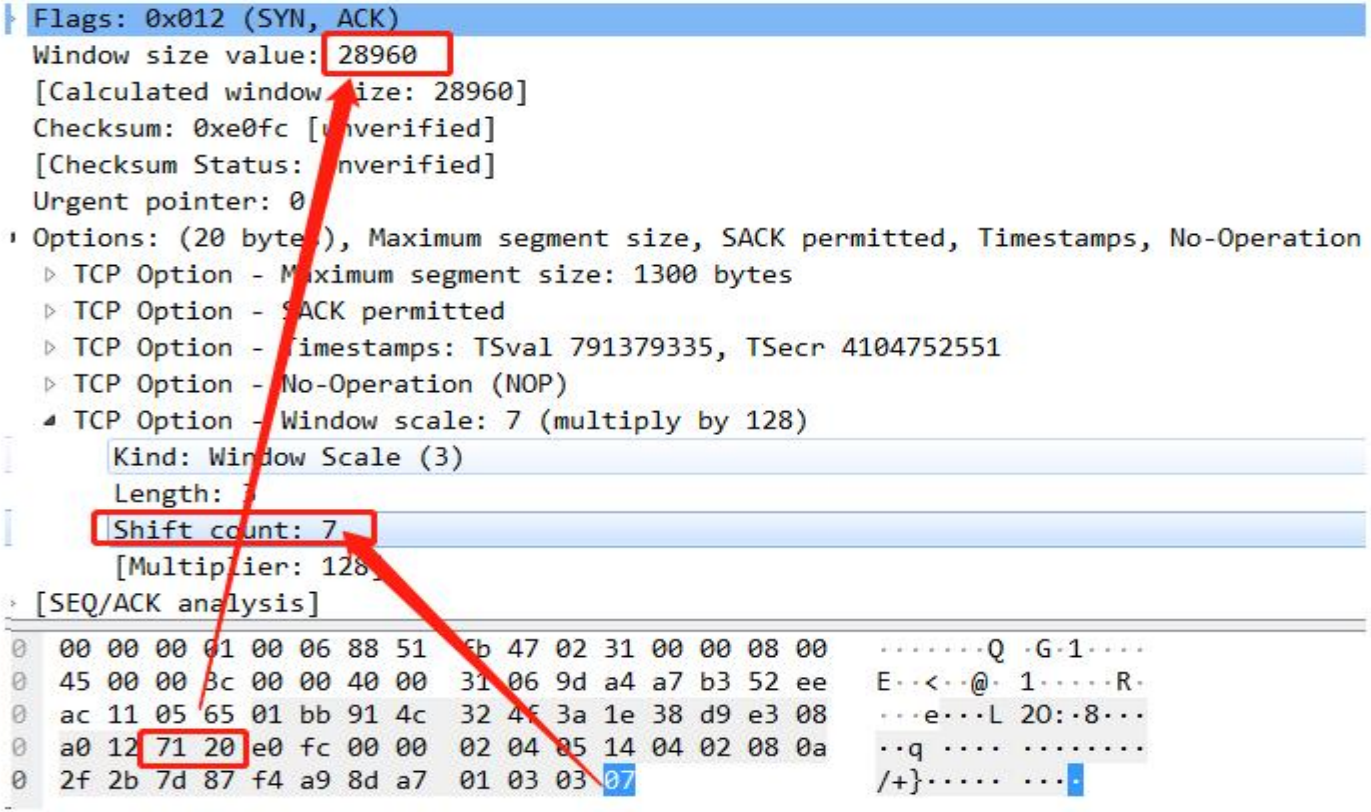
（4）若所有发出去的包均被确认，则允许发送；

（5）上述条件都未满足，但发生了超时（200ms），则立即发送。

传输阶段——丢包问题/重传



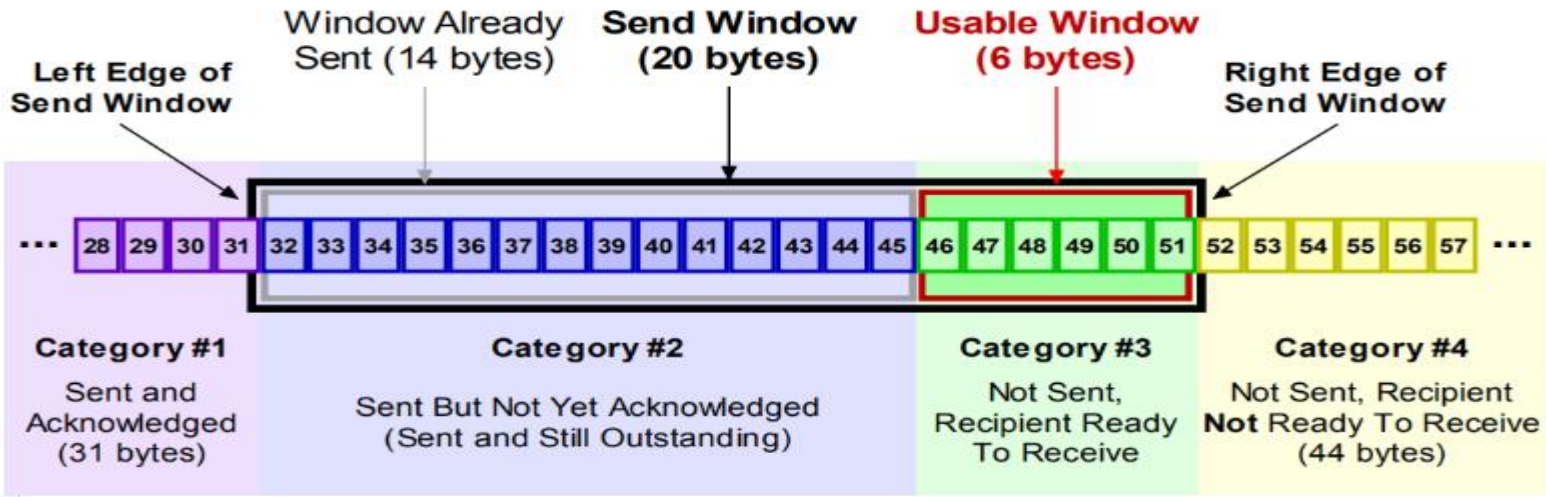
传输阶段——流量控制/RWND



滑动窗口大小计算: window_size = window_size * (2 ^ Window_scale) . 这里为 28960*(2^7) = 144800 字节

传输阶段——流量控制/滑动窗口

发送方有个发送缓冲区，发送缓冲区的布局如下：

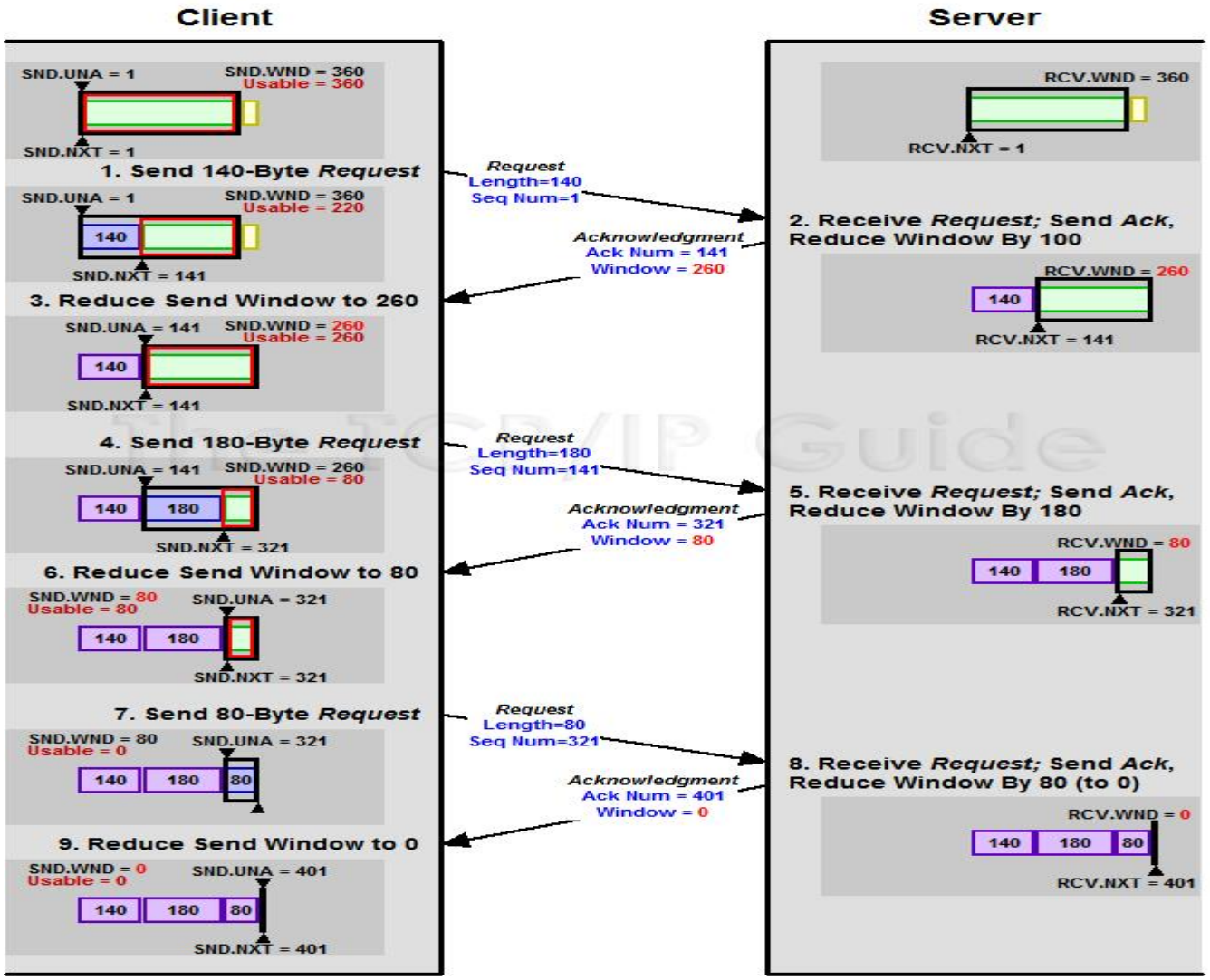


图七.png

图七中各段含义

- Category#1已收到ack确认的数据。
- Category#2发还没收到ack的。
- Category#3在窗口中还没有发出的（接收方还有空间）。
- Category#4窗口以外的数据（接收方没空间）

传输阶段——流量控制/滑动窗口



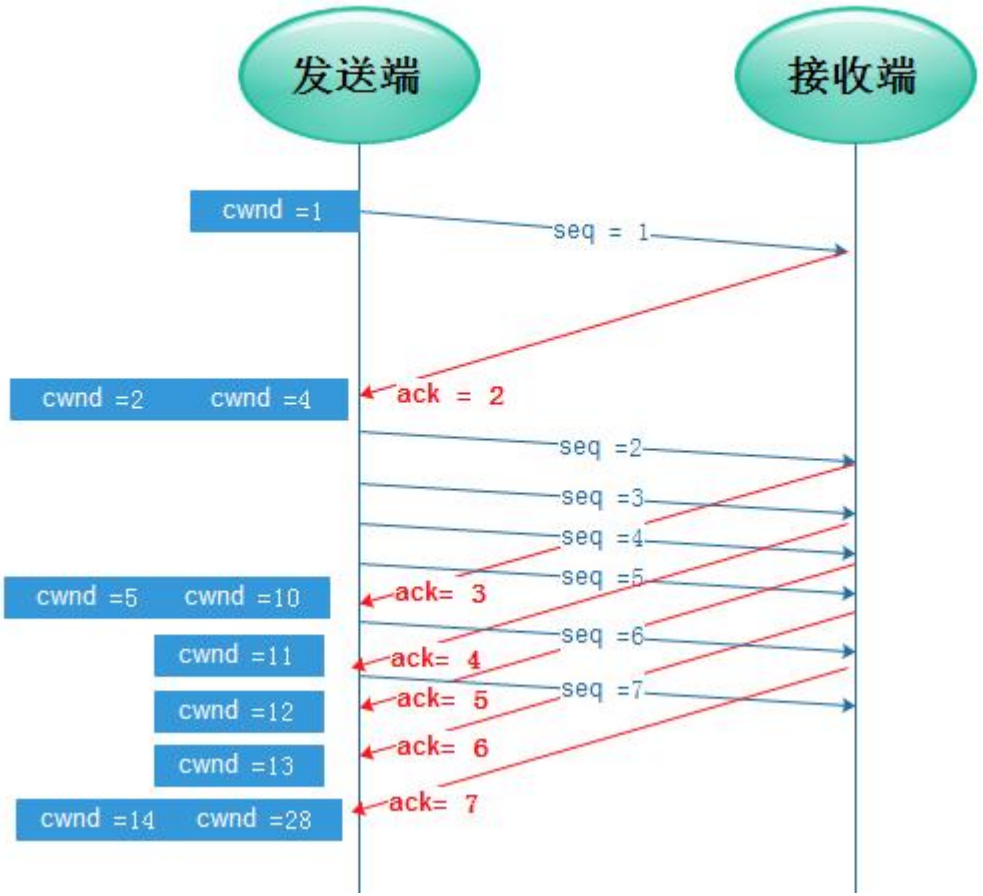
传输阶段——拥塞控制

和流量控制目的一样，都是为了提高网络传输效率。但是两者有些不同。

流量控制是依赖接**接收端**的接收能力进行发送速率调节。

拥塞控制是根据**网络状态的反馈**进行调节。因为网络状态不好直接体现。所以拥塞控制算法非常多，并且这几十年来一直在演变改进。主要算法有: **慢启动、拥塞避免、快重传、快恢复**。其中快速重传前面讲过。

传输阶段——拥塞控制/慢启动



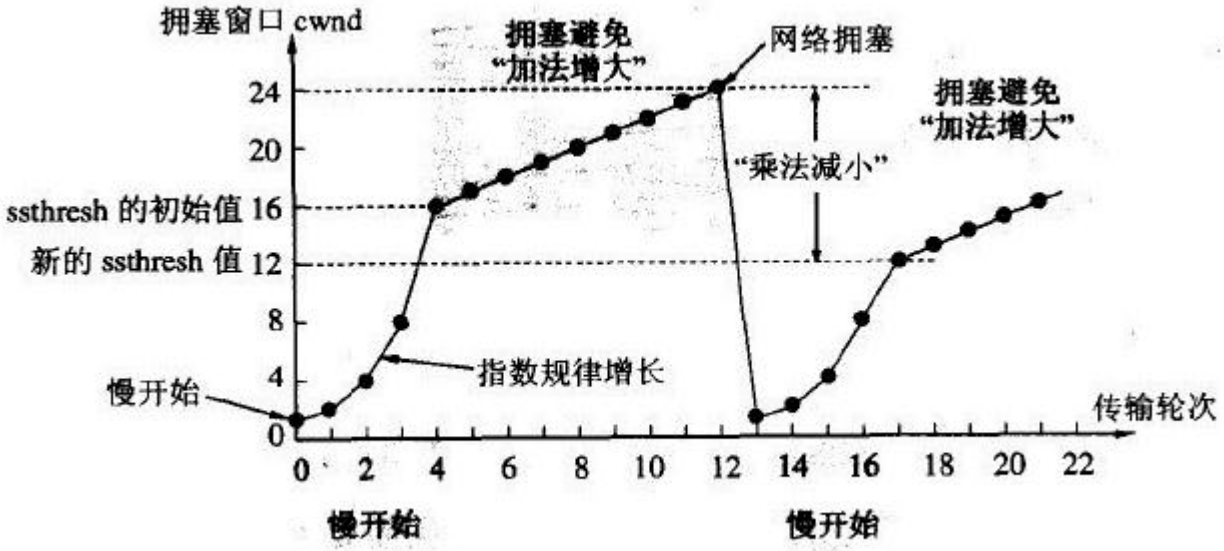
传输阶段——拥塞控制/拥塞避免

拥塞避免(Congestion Avoidance)算法：可以简单归纳成一句话：“加法增加， 乘法减少”

1、收到一个ACK时，cwnd = cwnd + 1 (加法增加)

2、当每过一个RTT时，cwnd = cwnd + 1 (加法增加)

3、当ACK超时重传时，sshthresh = cwnd /2 （乘法减少），cwnd =1 降到最低点重新慢启动流程



传输阶段——拥塞控制/快速恢复算法

当出现ACK重传时候，并且启用了快速重传机制的话。则采用快速恢复算法 (Fast Recovery)

1、收到一个ACK时，cwnd = cwnd + 1 (加法增加)

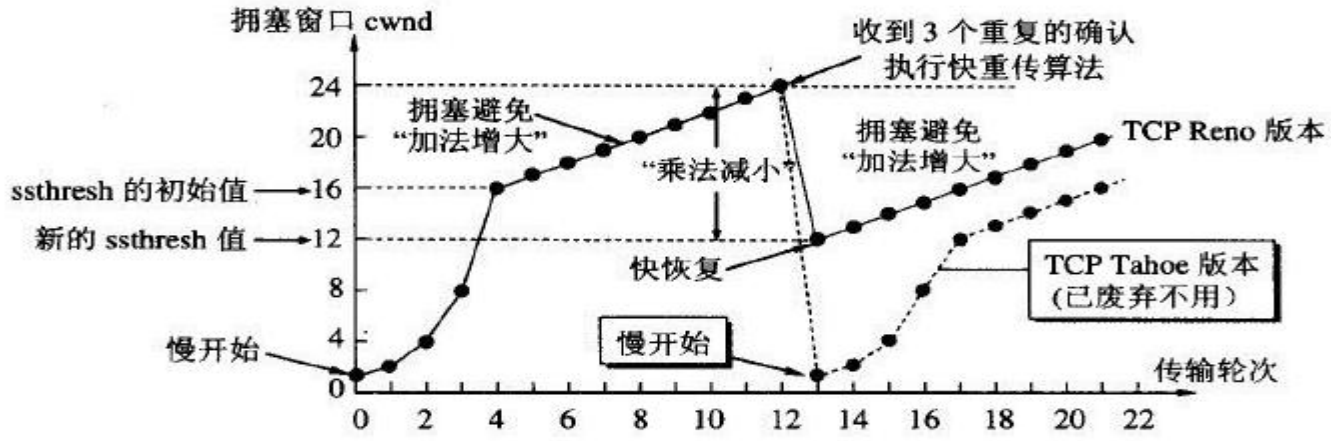
2、当每过一个RTT时，cwnd = cwnd + 1 (加法增加)

当丢包或延迟出现连续三个重复ACK时候:

3、sshthresh = sshthresh /2 （乘法减少）

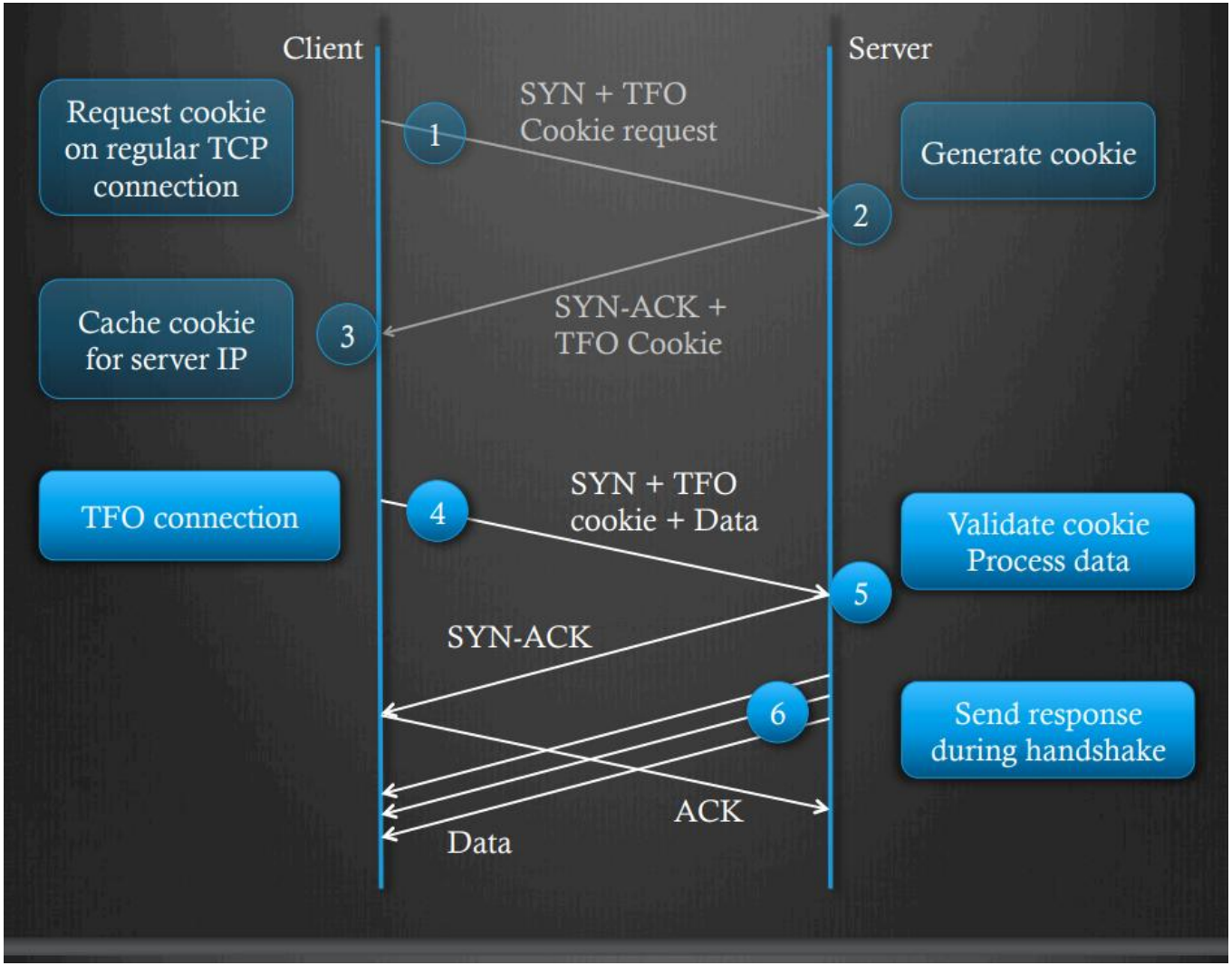
4、cwnd = sshthresh

5、进入加法增加，乘法减少的 拥塞避免算法



目录

TCP优化——快速启动TFO



TCP加速——贪婪算法

1、“加塞”：加大重传频率，一个包重复发送几次。1K包消耗3K流量

2、“永不减速”：不管网络拥塞的反馈，不减少发送窗口大小。

损人利己，不可持续发展

TCP加速——BBR

传统算法:

1、基于丢包率决定发送窗口大小

2、慢启动过程，探测极限带宽。填充路由器缓冲，增大延迟

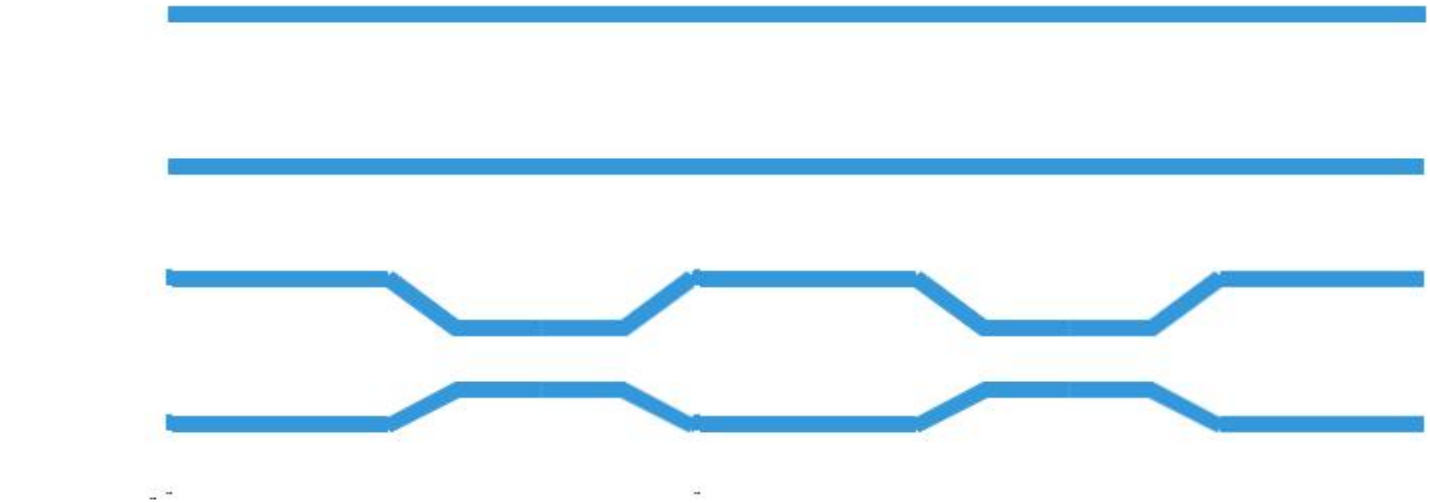
3、乘法减小：对丢包敏感，带宽利用率低。

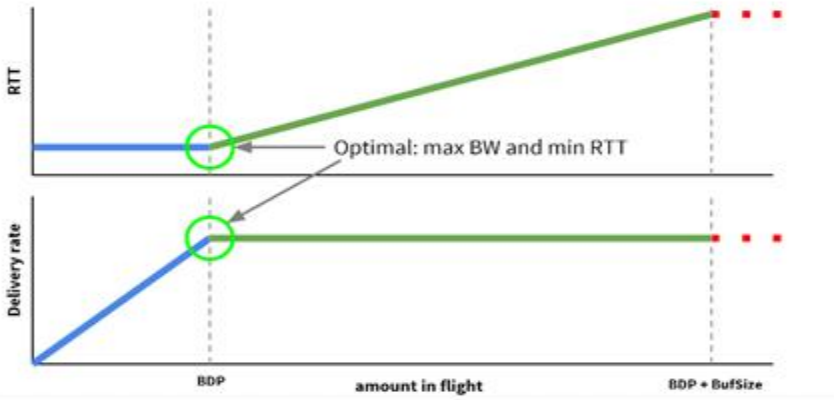
BBR算法:

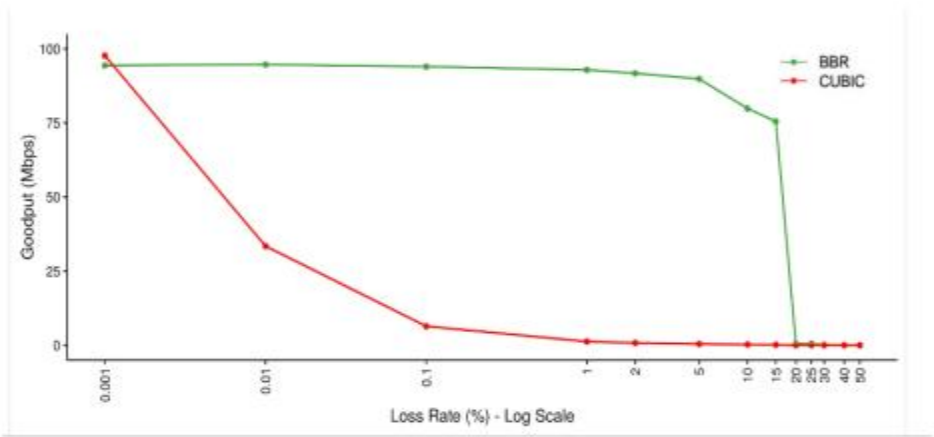
1、基于 最小延迟 和极大带宽计算发送带宽大小。

2、减少路由器上的缓冲占用，降低延迟。

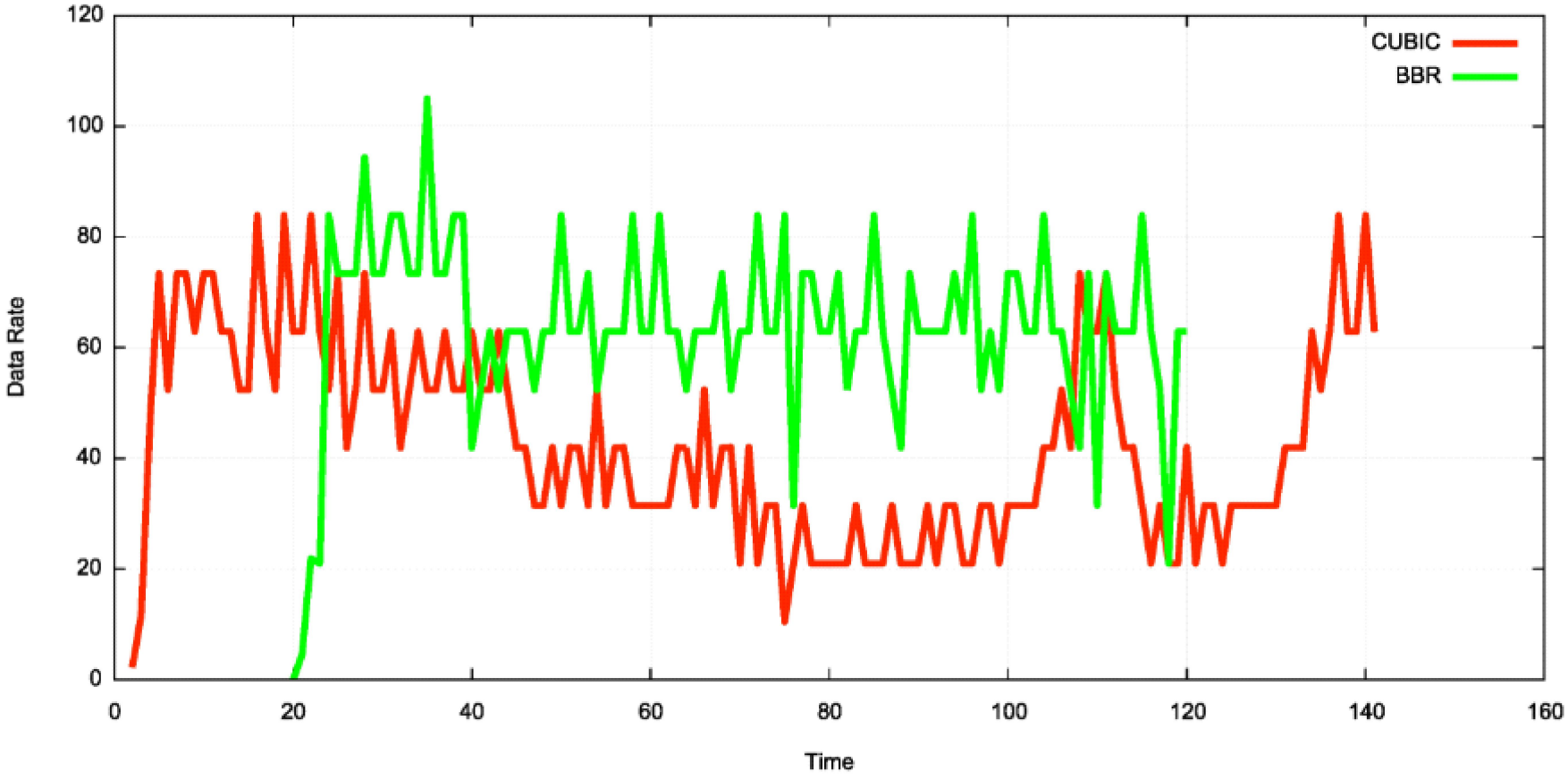
TCP加速——BBR







TCP加速——BBR



TCP那些事

谢谢！

THANKS