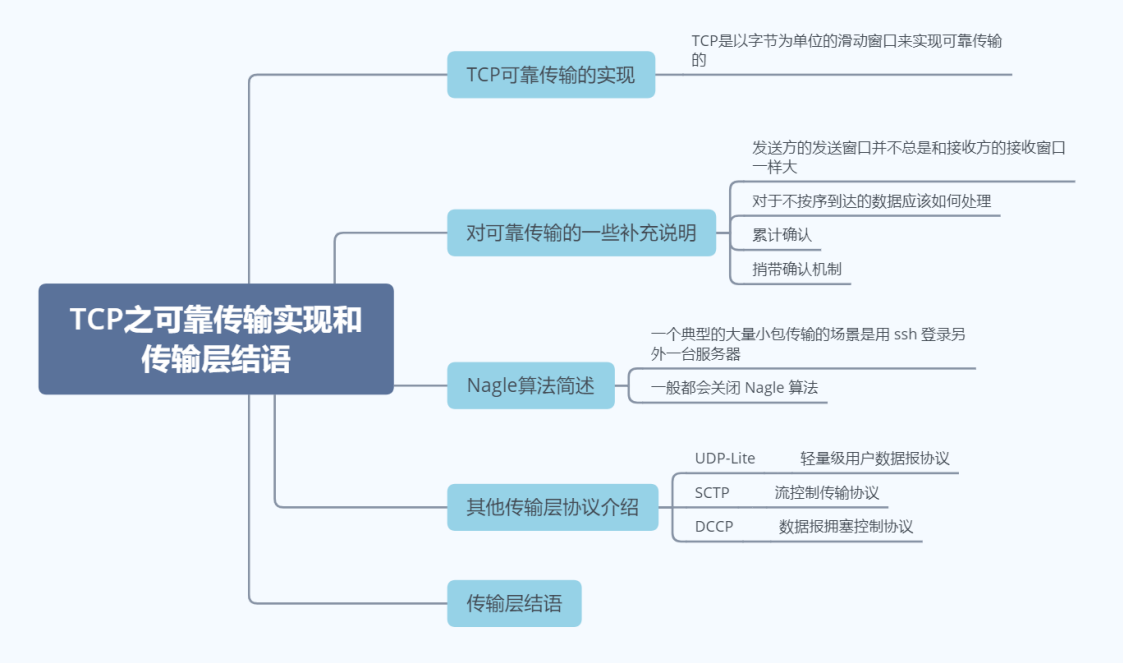
本篇文章作为正序中传输层篇的最后一篇文章，对可靠传输机制进行一个总结，其次介绍下为了提高传输效率而产生的：累计确认、捎带确认以及Nagle算法，最后的最后捎带介绍下其他三种传输层的协议和基本思想。

本篇文章作为结束篇，是对之前内容的一个整体补充和再次强调，且容我继续絮叨絮叨，整体内容如下：



一、TCP可靠传输的实现

我们一开始学习传输层的时候就知道：**TCP是可靠的传输协议**。

经过以上一系列的学习，我们知道，TCP是以字节为单位的滑动窗口来实现可靠传输的，我们也在《**45 | 传输层篇：TCP之流量控制**》中学习了滑动窗口的概念和工作机制。

关于如何做到可靠，联系到现实生活，我们夸赞一个人可靠会说：**事事有回应，件件有着落**。

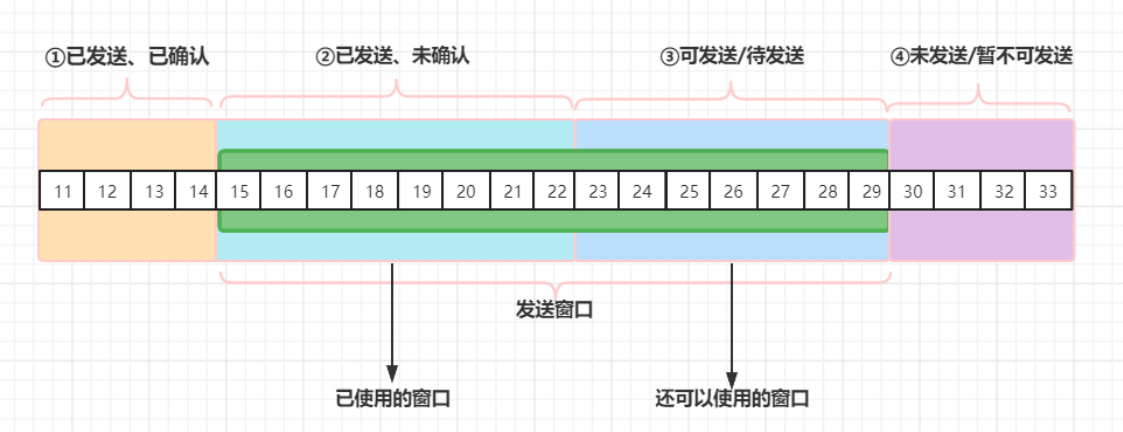
回到TCP中也是一样，发送方每发送的一个包，接收方都应该有个回复，如果接收方超过一定的时间没有回复，发送方就会重新发送这个包，直到有回复。

如何做呢，**自然就是对每个包进行编号**，发送方发送一个包，接收方应当对收到的包进行确认，如果一个包一个包地确认，没有必要，还增加了网络负担，于是出现了累计确认或者累计应答（cumulative acknowledgment）。

累计确认意思是，比如接收方回复ACK(10)，表示0-9号包已全部接收了，我下面需要接收的是10号包。

发送方收到这个ACK(10)之后也就知道前面的0-9数据已经安全到达，发送窗口往后移动，前面的数据包可以从发送缓存中删除了。

自然，发送方和接收方双方都需要记录所发送的包和接收的包，以发送方为例说明，发送窗口中存储的就是包的序号，将数据包分为了：已发送并已确认、已发送未确认、未发送等几类数据包，具体如图：



二、对TCP可靠传输的补充说明

虽然发送方的发送窗口是根据接收方的接收窗口设置的，但在同一时刻，发送方的发送窗口并不总是和接收方的接收窗口一样大，原因是：

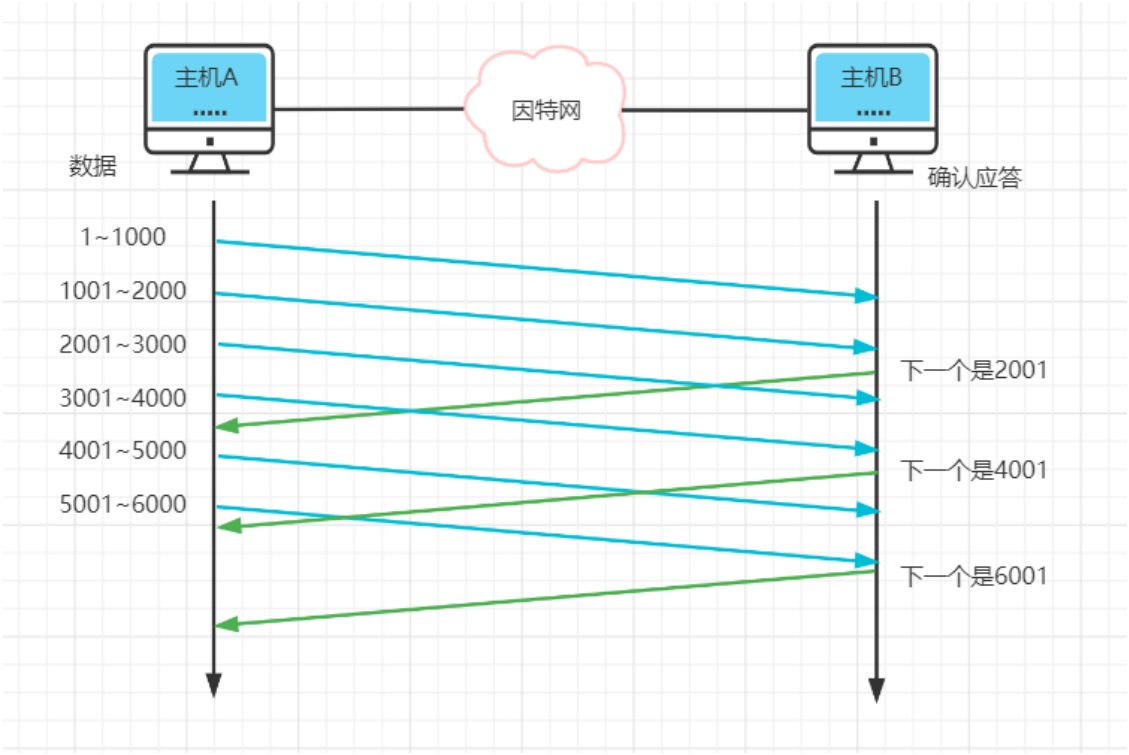
- 网络传送窗口值需要经历一定的时间滞后，并且这个时间是不确定的
- 发送方还可能根据网络当时的拥塞情况适当减少自己的发送窗口尺寸

对于不按序到达的数据应该如何处理，TCP并无明确规定：

- 如果接收方把不按序到达的数据一律丢弃，那么接收窗口的管理将会比较简单，但这样做对网络资源的利用不利，因为发送方会重复传送较多的数据
- **TCP通常对不按序到达的数据是先临时存放在接收窗口中，等到字节流中所缺少的字节收到后，再按序交付给上层应用进程**

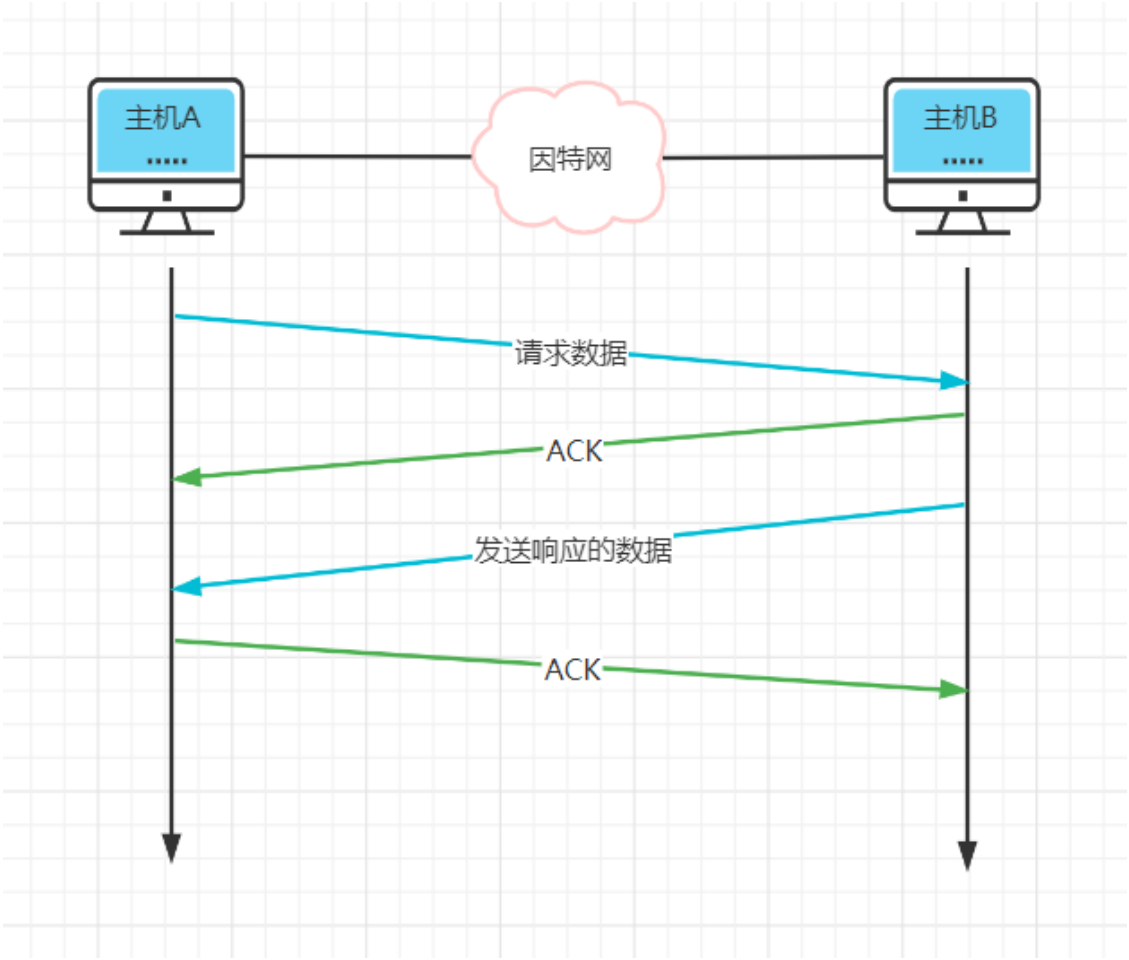
TCP要求接收方必须有累计确认和捎带确认机制，这样可以减少传输开销，接收方可以在合适的时候发送确认，也可以在自己有数据要发送时把确认信息顺便捎带上。

- 累计确认：也有书中称之为延迟确认应答，学习了滑动窗口机制，我们知道大可不必为每个数据段都进行一次确认应答，因此通常少应答一些也无妨，**绝大多数是每两个数据段返回一次确认应答**。

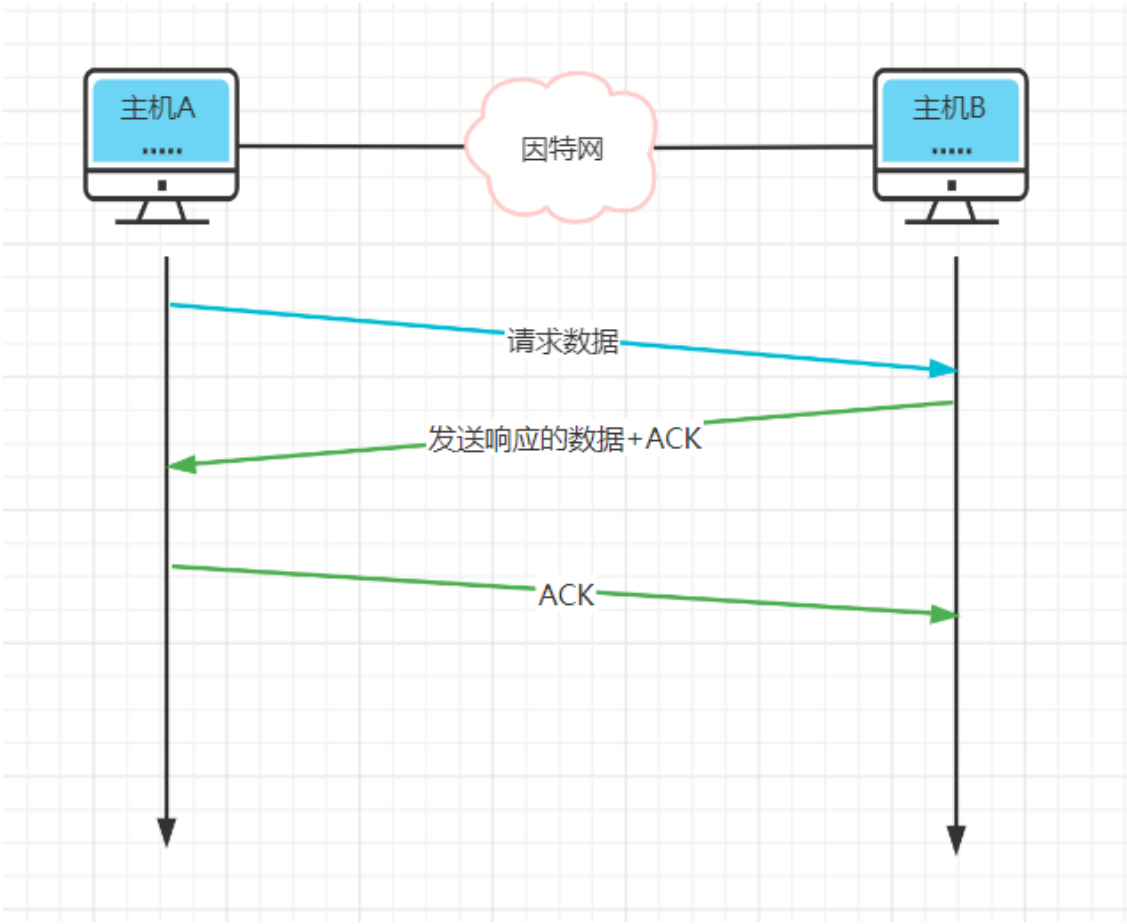


- 捎带确认机制：是指在同一个TCP包中既发送数据又发送确认应答的一种机制，由此提高网络利用率，降低计算机的负载；另外，接收方接收数据后如果立刻返回确认应答，就无法实现捎带确认，也就是说，如果没有启用延迟确认应答就无法实现捎带确认。

捎带确认机制很容易理解，比如不带捎带确认机制时，客户端与服务端是一问一答：



加上捎带确认机制后变成：



此外，需要额外说明的点是：

- 接收方不应过分推迟发送确认，否则会导致发送方不必要的超时重传，这反而浪费了网络资源。TCP标准（RFC 1122）规定，确认推迟的时间不应超过0.5秒，若收到一连串具有最大长度（MSS）的报文段，则必须每隔一个报文就发送一个确认，即**每两个数据段返回一次确认应答**。（这个需要按照实际情况来，有的操作系统不管报文段大小，只要收到两个包就立刻返回确认应答）
- 捎带确认实际上并不经常发生，因为大多数应用进程很少同时在两个方向上发送数据。

TCP的通信是全双工通信，通信中的每一方都在发送和接收报文段，因此，每一方都有自己的发送窗口和接收窗口，当谈到这些窗口时，要弄清楚是哪一方的窗口。

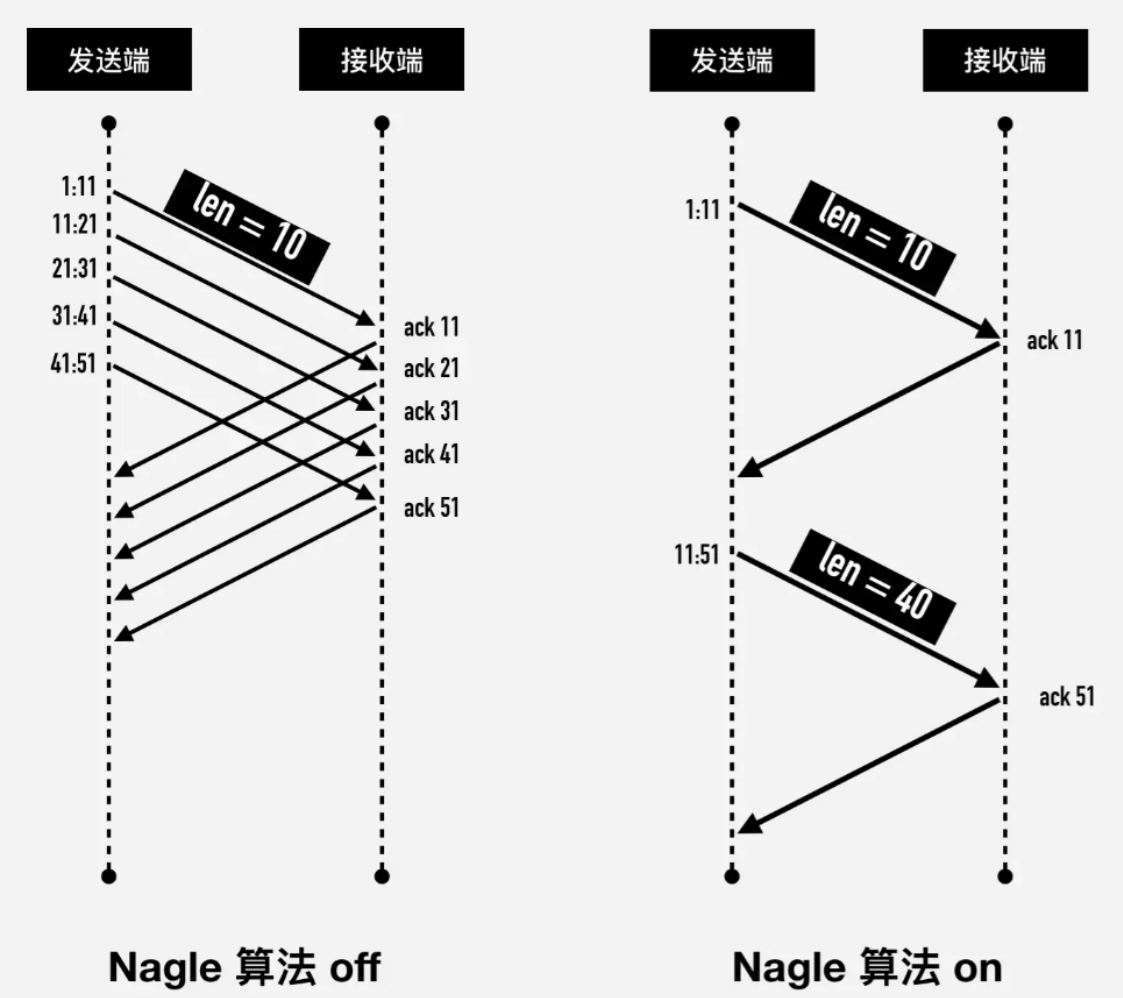
上面提到了累计确认、捎带确认，都是为了提高网络利用率而生，另外其实还有一个Nagle算法。

三、Nagle算法简述

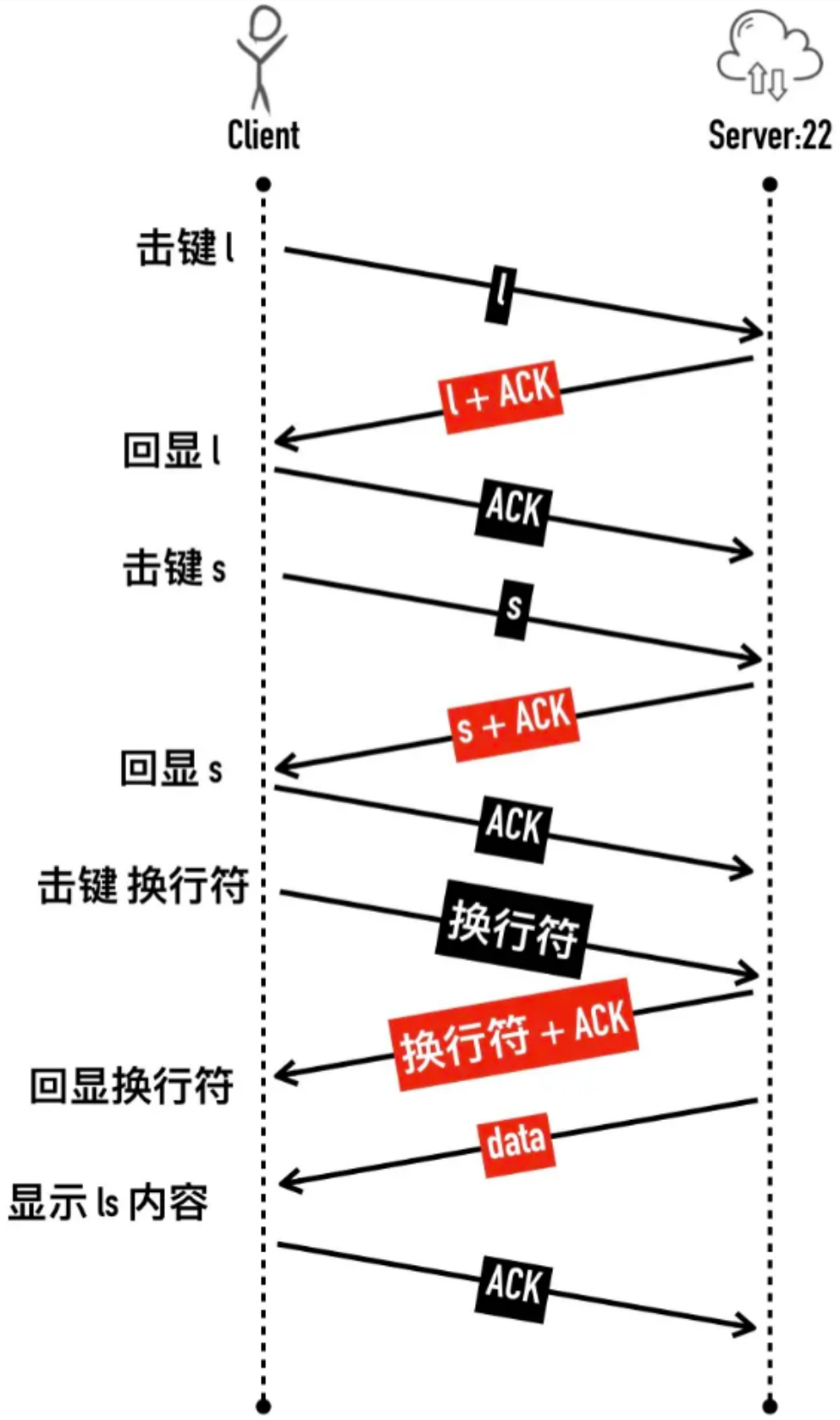
简单来讲nagle算法讲的是减少发送端频繁的发送小包给对方。

Nagle 算法要求，当发送方即使还有应该发送的数据，但如果这部分数据很少的话，则进行延迟发送。

具体来讲，当一个 TCP 连接中有在传数据（已经发出但还未确认的数据）时，小于 MSS 的报文段就不能被发送，直到所有的在传数据都收到了 ACK。同时收到 ACK 后，TCP 还不会马上就发送数据，会收集小包合并一起发送。



一个典型的大量小包传输的场景是用 ssh 登录另外一台服务器，每输入一个字符，服务端也随即进行回应，客户端收到了以后才会把输入的字符和响应的内容显示在自己这边。比如登录服务器后输入ls然后换行，中间包交互的过程如下图：



- 客户端输入l，字符 l 被加密后传输给服务器
- 服务器收到l包，回复被加密的 l 及 ACK
- 客户端输入s，字符 s 被加密后传输给服务器
- 服务器收到s包，回复被加密的 s 及 ACK

- 客户端输入 enter 换行符，换行符被加密后传输给服务器
- 服务器收到换行符，回复被加密的换行符及 ACK
- 服务端返回执行 ls 的结果
- 客户端回复 ACK

Nagle 算法是时代的产物：Nagle 算法出现的时候网络带宽都很小，当有大量小包传输时，很容易将带宽占满，出现丢包重传等现象。因此对 ssh 这种交互式的应用场景，选择开启 Nagle 算法可以使得不再那么频繁的发送小包，而是合并到一起，**代价是稍微有一些延迟**。现在的 ssh 客户端已经默认关闭了 Nagle 算法。

在如今的网络环境下，网络时延都很低，一般都会关闭 Nagle 算法。

四、其他传输层协议介绍

这里插入一个对其他传输层协议的介绍，作为一个扩展点说一说，并不是重点。

在传输层，UDP和TCP犹如两颗双子星般交相辉映，璀璨于夜空，不过随着时间的推移，又出现了一些协议，并逐步将走出实验室进入实用阶段。

- **UDP-Lite（轻量级用户数据报协议）**

UDP-Lite全称是Lightweight User Datagram Protocol，轻量级用户数据报协议，是扩展UDP机能的一种协议。

UDP协议中如果出现校验和错误，则直接丢弃该包，但是有时候我们并不想丢弃。

那直接关闭UDP的校验和机制呢？也不推荐，因为出错很可能是UDP首部或是IP首部被破坏（比如接收方端口号或者接收方IP这些关键信息），就会产生严重后果，因此不建议关闭校验和。

UDP-Lite于是出现了，提供与UDP几乎相同的功能，不过计算校验和的范围可以由应用自行决定。我们可以只针对不允许发生错误的部分进行校验和检查，对于其他数据即使发生错误也忽略不计，继而这个数据可以顺利交付给上层应用处理。

- **SCTP（流控制传输协议）**

SCTP全称是Stream Control Transmission Protocol，流控制传输协议，与TCP一样都是提供可靠传输的传输层协议。

SCTP主要用于进行通信的应用之间发送众多较小消息的情况，它有一个特色是：SCTP支持多重宿主以及设定多个IP地址。

多重宿主是指同一台主机具有多个NIC（网卡），比如笔记本电脑一般同时具有以太网卡和无线网卡。

多个IP就很好理解了，同时使用以太网和无线网时，各自的NIC就会获取到不同的IP地址。

如果是TCP协议，如果一开始使用的是以太网，后来切换为WLAN，那么连接就会断开，因为从SYN到FIN包必须使用同一个IP地址。

但是在SCTP中，由于可以管理多个IP地址使其同时进行通信，因此即使出现通信过程中以太网和无线网之间的切换，也能保持通信不中断。

- **DCCP（数据报拥塞控制协议）**

DCCP全称是Datagram Congestion Control Protocol，数据报拥塞控制协议。

是一个辅助UDP的新的传输层协议，由于UDP没有拥塞控制机制，当发送大量数据时很容易出问题，因此，即使使用UDP也应该配备拥塞控制机制，而很难将这个机制融入UDP中，于是出现了DCCP，具有如下特点：

- 与UDP一样，不提供可靠传输
- 面向连接，在建立连接和断开连接这块处理上有可靠性
- **能够根据网络拥堵情况进行拥塞控制**
- 为了进行拥塞控制，接收端收到包后返回ACK确认应答，该确认应答将被用于判断是否重发

五、传输层结语

关于传输层，尤其是TCP，我们主要讨论了最重要的基础知识，很多方面还未深入探讨，希望后续有机会，结合操作系统层面深入学习TCP原理。