网络协议笔记3-传输层（初步笔记）

- 传输层的协议有两个，TCP和UDP
  - TCP和UDP的特点和区别：

UDP

数据格式

- UDP是无连接的，减少了建立和释放链接的开销
- UDP尽最大能力交付，不保证可靠交付
  - 所以不需要维护复杂的参数，首部只有8个字节 -UDP长度： 占16位，首部长度 + 数据长度

检验和

- 检验和的计算内容： 伪首部 + 首部 + 数据
  - 伪首部：仅在计算检验和时起作用，不传递到网络层 #### 端口
- UDP首部中端口占用2字节，取值范围0-65535
- 客户端的端口时随即开启的
- 防火墙可以设置开启/关闭某些端口提高安全性

TCP

- TCP首部示意图：

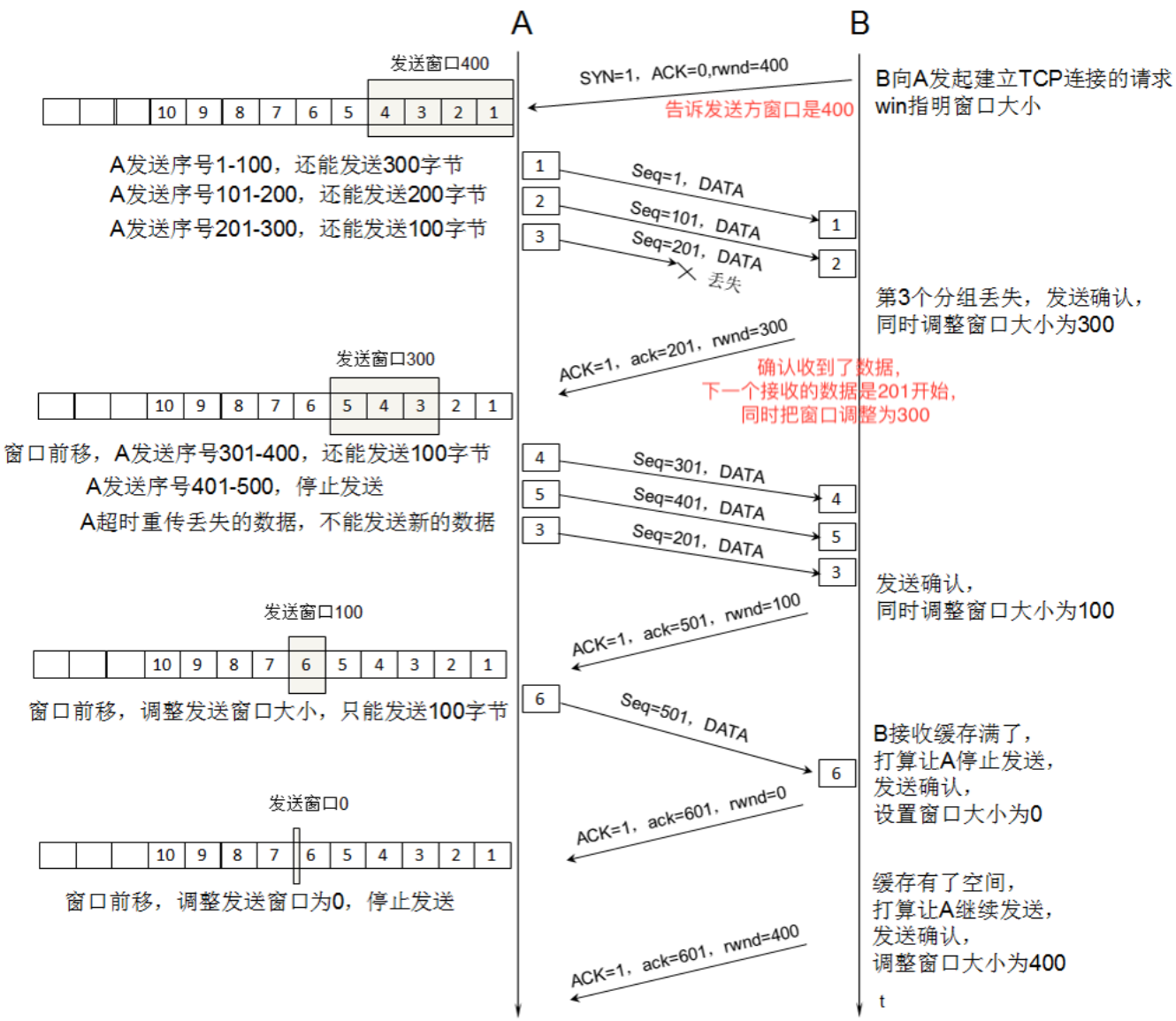
- 数据偏移：占4位，取值范围0x0101-0x1111
  - 首部长度需要 * 4 即 20-60字节
- 保留：占6位，目前全为0
- 检验和：伪首部 + 首部 + 数据
  - 伪首部：占12字节，仅在计算检验和时起作用，不传递到网络层（与UDP一样）
- 标志位：
  - URG（Urgent）：当URG = 1时生效，表明需要优先传输
  - ACK（Acknowledgment）：当ACK = 1时生效
  - PSH（Push）
  - RST（Reset）：当RST = 1时，表明链接出错，需要释放链接重新建立
  - SYN（Synchronization）：
    - 当SYN = 1，ACK = 0时，表明这是一个建立连接的请求
    - 若对方同意，则恢复SYN = 1，ACK = 1
  - FIN（Finish）：当FIN = 1时，表明数据发送完毕，需要释放
- 序号（Sequence Number）：
  - 占4字节
  - 首先，在传输过程的每一个字节都会有一个编号
  - 在建立连接后，序号代表这次传给对方的TCP数据第一个字节的编号
- 确认号（Acknowledgment Number）：
  - 占4字节
  - 建立连接后，确认号代表期望对方下次传过来的TCP数据部分的第一个字节编号

- 窗口（Window）
  - 两字节
  - 有流量控制功能，高速对方下一次允许发送的数据大小

TCP的特点

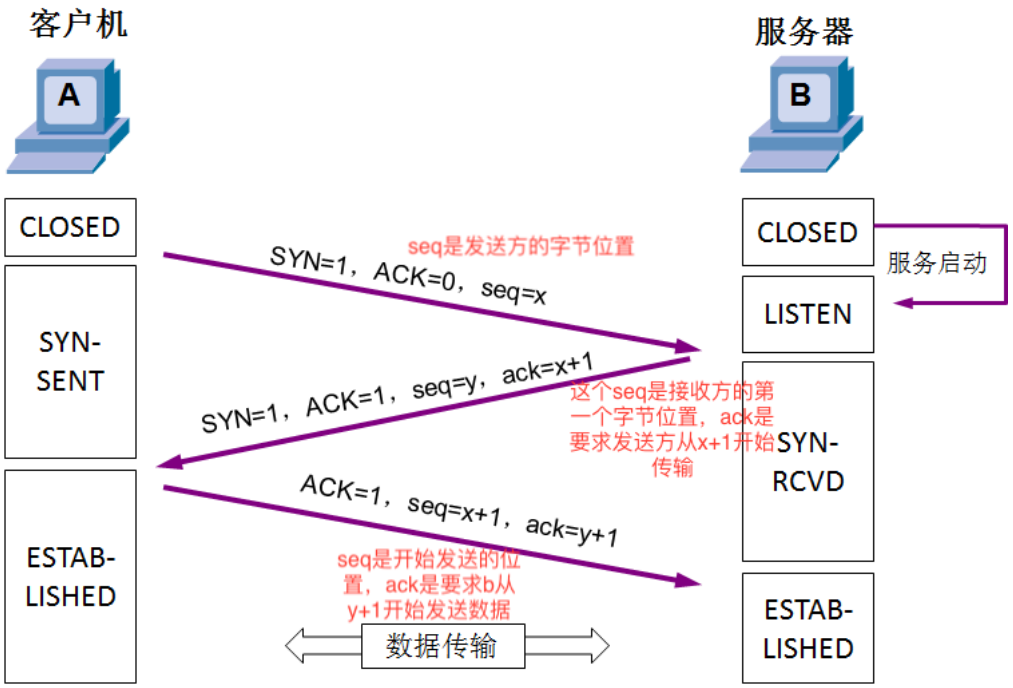
- 可靠传输 ####停止等待ARQ协议
  - 在服务端发送数据后，没有接收到客户端回复收到信息的报告的话，会重新发送该数据段，若后面再次收到之前发送过的数据段的回复，会丢弃掉该数据
  - 多次发送失败的话会发送Reset，断开连接 ####连续ARQ协议 + 滑动窗口协议
- 接收方通知发送方当前缓存区可以接受的大小，发送方将数据段分成n组同时发送，接收方收到后通知发送方下次发送开始的字节编号，再次分段发送。
  - 通信过程中，若中间某个数据包丢失（如1-5丢失了3）
  - 会通过重传最后确认的分组后续的分组（回复最后数据是2，重传345）
  - 为解决上述问题，有协议可以将45收到的消息返回，使发送方只发送3 SACK示意图：
  - kind：1字节，固定为5，表示是SACK
  - Length：1字节，比报名Sack一共占用多少字节
  - Left Edge：4字节，左边界
  - Right Edge：4字节，右边界
  - 其中左右边界存储接收方收到的数据段的左边的字节号和右边的字节号，表示中间位置已经收到
  - 该数据占用8个字节，由于TCP首部的选项最多40字节，所以最多携带4组信息，意味着有可能不能完全满足所有收到的数据段全部回复
  - SACK选项的最大占用字节数 = 4 * 8 + 2 = 34
- 流量控制
  - 当接收方缓存区满了后，控制发送方少发或不发的方法
  - 原理：通过确认报文中的wind字段来告诉发送方应该发送的数据大小，当发送方收到的窗口为0时会停止发送。
  - 特殊情况：接收方的窗口已满，调整接收缓存为0，当需要继续接收时通知发送方的包丢失了，会无法接收数据 ，解决方案是发送方收到窗口为0时定时发送测试报文询问接收方的窗口大小。

如图所示流量控制的步骤：



- 拥塞控制
  - 防止过多的数据注入到网络中，避免路由或链路过载。
  - 慢开始
  - 拥塞避免
  - 快速重传
  - 快速恢复
  - 名词：
    - MSS：每个段最大的数据部分大小，在链接时确定
    - cwnd：拥塞窗口
    - rwnd：接收窗口
    - swnd：发送窗口
    - ***发送窗口 = MIN（cwnd，rwnd）***
- 连接管理
  - 建立连接

建立连接图示



- 相关状态：
  - closed：关闭状态
  - Listen：接收方处于监听状态
  - Syn-Rcvd：接收方收到了syn报文，
  - Syn-Sent：发送方已经发送Syn报文，等待接收方的第二次握手
  - Established：连接已经建立
- 建立连接需要3次握手
  - 前两次握手的特点：syn都是1，数据部分长度为0，tcp头部长度一般是32字节包含20个固定头部和12个选项部分，在选项部分会包含MSS,SACK,window scale等信息

为什么要进行3次握手，两次的缺点是什么

- 两次握手可能出现client发送的第一个连接请求的报文段因为网络延迟，在连接释放后才到达server，本来是个失效的连接请求，但因为两次握手不需要client向server发送确认报文，所以连接就会被建立，浪费了server端的资源，三次握手会在client确认后建立，client不会相应这次的请求所以不会建立连接。

第三次握手失败，如何处理

- server在SYN-RCVD状态，如果等不到client的ACK，会重发SYN+ACK包
- 如果多次重发都等不到确认，会发送RST包强制关闭连接

- 发送数据
  - 发送方在建立连接请求的时候会将起始字节编号传递给接收方(seq),接收方响应时发送自己的起始字节编号(seq)并通过ack确认码告知发送方下一个包第一个需要从哪个字节发送，中间阶段超过了最大接受字节分段发送的逻辑也是如此
  - 示意图：

紫色标记客户端（发送数据方），蓝色标记服

①：TCP数据部分占0字节		
SYN=1，ACK=0	seq	ack
原生	s1	0
相对	0	0

②：TCP数据部分占0字节		
SYN=1，ACK=1	seq	ack
原生	s2	s1 + 1
相对	0	1

1，2，3是三次握手建立连接，4标记为发送数据情况

- 发送大数据分割的示意图

传输数据阶段，服务端将较大的

⑤：TCP数据部分占b1字节		
SYN=0 ACK=1	seq	ack
原生	s2 + 1	s1 + k + 1
相对	1	k + 1

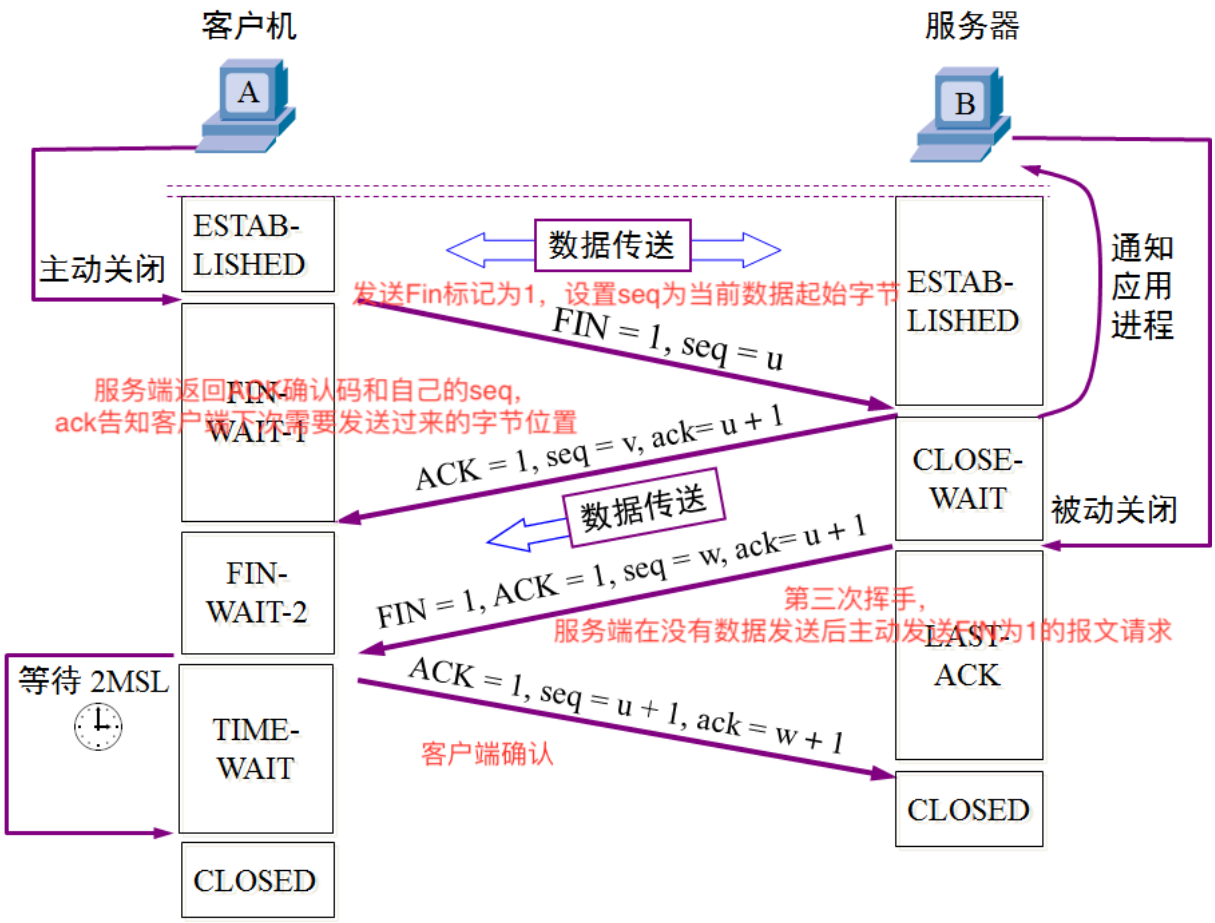
SYN=0 ACK=1	
原生	
相对	

⑥：TCP数据部分占b2字节		
SYN=0 ACK=1	seq	ack
原生	s2 + b1 + 1	s1 + k + 1
相对	b1 + 1	k + 1

SYN=0 ACK=1	
原生	
相对	

- 断开连接

断开连接四次挥手的图示：



- 相关状态：
  - FIN-WAIT-1:表示主动想要关闭连接：第一次挥手后进入该状态
  - CLOSE-WAIT：表示在等待关闭：收到第一次挥手请求后，进入该状态，在该状态下查询是否有数据发送给对方，如果没有发送FIN报文。
  - FIN-WAIT-2:主动方状态，收到对方ACK确认后进入该状态
  - CLOSING：双方同时发送FIN报文，没有收到ACK报文的状态，比较罕见
  - LAST-ACK：被动方状态：第三次挥手后进入，当收到主动方ACK报文后即可关闭
  - TIME-WAIT：表示收到了对方的FIN，并发出ACK，等待2msl即可关闭，如果在FIN-wait-1状态下收到同时带fin和ack标志的报文时会直接进入time-wait，无需经过fin-wait-2状态
  - CLOSED：关闭状态
- 细节：
  - 允许任何一方主动断开连接
  - 主动方发送ACK后，需要等待time-wait阶段才会关闭，一般为2 * 2分钟
  - 如果主动方发送ACK后马上释放，又因为网络原因，接收方没有收到，那么接收方会重发FIN，可能导致的情况：
    1. 主动方没有响应，多次发送FIN导致资源兰芳
    2. 主动方恰好分配了同一端口的应用，可能导致本不想断开连接的新应用断开连接
  - 有时断开连接只有3次挥手，是因为接收方没有数据返回，将第2、3次挥手合并成了一次 #### 为什么要进行4次挥手

■ 为什么释放连接的时候，要进行4次挥手?

□TCP是全双工模式

□第1次挥手：当主机1发出FIN报文段时

✓表示主机1告诉主机2，主机1已经没有数据要发送了，但是，

□第2次挥手：当主机2返回ACK报文段时

✓表示主机2已经知道主机1没有数据发送了，但是主机2还是可

□第3次挥手：当主机2也发送了FIN报文段时

✓表示主机2告诉主机1，主机2已经没有数据要发送了

□第4次挥手：当主机1返回ACK报文段时

✓表示主机1已经知道主机2没有数据发送了。随后正式断开整个