- 对端传输的应答也可能出现丢失或超时的情况 。那么超过定时器时间 A 端照样会重传报文 。这时候 B 端收到相同序号的报文会丢弃该报文并重传应答， 直到 A 端发送下一个序号的报文 。
- 在超时的情况下也可能出现应答很迟到达， 这时 A 端会判断该序号是否已经接收过， 如果接收过只需要丢弃应答即可。
- 从上面的描述中大家肯定可以发现这肯定不是一个高效的方式 。假设在良好的网络环境中，每次发送数据都需要等待片刻肯定是不能接受的 。那么既然我们不能接受这个不那么高效的协议，就来继续学习相对高效的协议吧。

连续 ARQ

> 在连续 ARQ 中，发送端拥有一个发送窗口， 可以在没有收到应答的情况下持续发送窗口内的数据， 这样相比停止等待 ARQ 协议来说减少了等待时间，提高了效率。

累计确认

> 连续 ARQ 中，接收端会持续不断收到报文 。如果和停止等待 ARQ 中接收一个报文就发送一个应答一样，就太浪费资源了 。通过累计确认， 可以在收到多个报文以后统一回复一个应答报文 。报文中的 ACK 标志位可以用来告诉发送端这个序号之前的数据已经全部接收到了，下次请发送这个序号后的数据。

> 但是累计确认也有一个弊端 。在连续接收报文时， 可能会遇到接收到序号 5 的报文后， 并未接收到序号 6 的报文 ，然而序号 7 以后的报文已经接收 。遇到这种情况时，ACK 只能回复 6， 这样就会造成发送端重复发送数据的情况

4. 滑动窗口

- 上面小节中讲到了发送窗口 。在 TCP 中，两端其实都维护着窗口 ： 分别为发送端窗口和接收端窗口 。
- 发送端窗口包含已发送但未收到应答的数据和可以发送但是未发送的数据。

- 发送端窗口是由接收窗口剩余大小决定的 。接收方会把当前接收窗口的剩余大小写入应答报文 ， 发送端收到应答后根据该值和当前网络拥塞情况设置发送窗口的大小 ， 所以发送窗口的大小是不断变化的。
- 当发送端接收到应答报文后，会随之将窗口进行滑动

> 滑动窗口是一个很重要的概念， 它帮助 TCb 实现了流量控制的功能 。接收方通过报文告知发送方还可以发送多少数据，从而保证接收方能够来得及接收数据， 防止出现接收方带宽已满，但是发送方还一直发送数据的情况

Zero 窗口

> 在发送报文的过程中， 可能会遇到对端出现零窗口的情况 。在该情况下， 发送端会停止发送数据， 并启动 `persistent timer` 。该定时器会定时发送请求给对端，让对端告知窗口大小 。在重试次数超过一定次数后， 可能会中断 `TCP` 链接

5. 拥塞处理

- 拥塞处理和流量控制不同，后者是作用于接收方 ，保证接收方来得及接受数据 。而前者是作用于网络， 防止过多的数据拥塞网络， 避免出现网络负载过大的情况。
- 拥塞处理包括了四个算法，分别为：慢开始，拥塞避免，快速重传，快速恢复

慢开始算法

> 慢开始算法，顾名思义，就是在传输开始时将发送窗口慢慢指数级扩大，从而避免一开始就传输大量数据导致网络拥塞 。想必大家都下载过资源，每当我们开始下载的时候都会发现下载速度是慢慢提升的， 而不是一蹴而就直接拉满带宽

慢开始算法步骤具体如下

- 连接初始设置拥塞窗口（Congestion Window） 为 `1 MSS` （一个分段的最大数据量）
- 每过一个 `RTT` 就将窗口大小乘二
- 指数级增长肯定不能没有限制的，所以有一个阈值限制， 当窗口大小大于阈值时就会启动拥塞避免算法。

拥塞避免算法

- 拥塞避免算法相比简单点，每过一个 `RTT` 窗口大小只加一， 这样能够避免指数级增长导致网络拥塞，慢慢将大小调整到最佳值。
- 在传输过程中可能定时器超时的情况， 这时候 `TCP` 会认为网络拥塞了，会马上进行以下步骤：

1. 将阈值设为当前拥塞窗口的一半
2. 将拥塞窗口设为 `1 MSS`
3. 启动拥塞避免算法

快速重传

> 快速重传一般和快恢复一起出现 。一旦接收端收到的报文出现失序的情况，接收端只会回复最后一个顺序正确的报文序号 。如果发送端收到三个重复的 `ACK` ， 无需等待定时器超时而是直接启动快速重传算法 。具体算法分为两种：

TCP Taho 实现如下

- 将阈值设为当前拥塞窗口的一半
- 将拥塞窗口设为 1 MSS
- 重新开始慢开始算法
- `TCP Reno` 实现如下

拥塞窗口减半

- 将阈值设为当前拥塞窗口
- 进入快恢复阶段（重发对端需要的包，一旦收到一个新的 ACK 答复就退出该阶段)， 这种方式在丢失多个包的情况下就不那么好了
- 使用拥塞避免算法

TCP New Ren 改进后的快恢复

- TCP New Reno 算法改进了之前 TCP Reno 算法的缺陷 。在之前，快恢复中只要收到一个新的 ACK 包，就会退出快恢复。
- 在 TCP New Reno 中，TCP 发送方先记下三个重复 ACK 的分段的最大序号。

> 假如我有一个分段数据是 `1 ~ 10` 这十个序号的报文， 其中丢失了序号为 3 和 7 的报文，那么该分段的最大序号就是 10。发送端只会收到 ACK 序号为 3 的应答 。这时候重发序号为 3 的报文，接收方顺利接收的话就会发送 ACK 序号为 7 的应答 。这时候 TCP 知道对端是有多个包未收到，会继续发送序号为 7 的报文，接收方顺利接收并会发送 ACK 序号为 11 的应答， 这时发送端认为这个分段接收端已经顺利接收，接下来会退出快恢复阶段。

32 HTTP/TLS

32.1 HTTP 请求中的内容

> HTTP 请求由三部分构成，分别为：

- 请求行
- 首部
- 实体

> - 请求行大概长这样 GET /images/logo.gif HTTP/1.1 ， 基本由请求方法、 URL 、协议版本组成， 这其中值得一说的就是请求方法了。
> - 请求方法分为很多种， 最常用的也就是 Get 和 Post 了 。虽然请求方法有很多，但是更多的是传达一个语义， 而不是说 Post 能做的事情 Get 就不能做了 。如果你愿意，都使用 Get 请求或者 Post 请求都是可以的

> 常考面试题：Post 和 Get 的区别?

- 首先先引入副作用和幂等的概念。
- 副作用指对服务器上的资源做改变， 搜索是无副作用的， 注册是副作用的。
- 幂等指发送 M 和 N 次请求（两者不相同且都大于 1)， 服务器上资源的状态一致， 比如注册 10 个和 11 个帐号是不幂等的，对文章进行更改 10 次和 11 次是幂等的 。因为前者是多了一个账号（资源)， 后者只是更新同一个资源。
- 在规范的应用场景上说， Get 多用于无副作用，幂等的场景，例如搜索关键字 。Post 多用于副作用，不幂等的场景，例如注册。

> - Get 请求能缓存， Post 不能
> - Post 相对 Get 安全一点点， 因为 Get 请求都包含在 URL 里（当然你想写到 body 里也是可以的)，且会被浏览器保存历史纪录 。 Post 不会，但是在抓包的情况下都是一样的。
> - URL 有长度限制，会影响 Get 请求，但是这个长度限制是浏览器规定的，不是 RFC 规定的
> - Post 支持更多的编码类型且不对数据类型限制

1. 首部

> 首部分为请求首部和响应首部， 并且部分首部两种通用， 接下来我们就 来学习一部分的常用首部。

1.1 通用首部

通用字段	作用
Cache-Control	控制缓存的行为
Connection	浏览器想要优先使用的连接类型， 比如 keep- alive
Date	创建报文时间
Pragma	报文指令
Via	代理服务器相关信息
Transfer-Encoding	传输编码方式
Upgrade	要求客户端升级协议
Warning	在内容中可能存在错误

1.2 请求首部

请求首部	作用
Accept	能正确接收的媒体类型
Accept-Charset	能正确接收的字符集
Accept-Encoding	能正确接收的编码格式列表
Accept-Language	能正确接收的语言列表
Expect	期待服务端的指定行为
From	请求方邮箱地址
Host	服务器的域名
If-Match	两端资源标记比较
If-Modified-Since	本地资源未修改返回 304 （比较时间）
If-None-Match	本地资源未修改返回 304 （比较标记）

请求首部	作用
User-Agent	客户端信息
Max-Forwards	限制可被代理及网关转发的次数
Proxy- Authorization	向代理服务器发送验证信息
Range	请求某个内容的一部分
Referer	表示浏览器所访问的前一个页面
TE	传输编码方式

1.3 响应首部

响应首部	作用
Accept-Ranges	是否支持某些种类的范围
Age	资源在代理缓存中存在的时间
ETag	资源标识
Location	客户端重定向到某个 URL
Proxy-Authenticate	向代理服务器发送验证信息
Server	服务器名字
WWW-Authenticate	获取资源需要的验证信息

1.4 实体首部

实体首部	作用
Allow	资源的正确请求方式
Content-Encoding	内容的编码格式
Content-Language	内容使用的语言
Content-Length	request body 长度
Content-Location	返回数据的备用地址
Content-MD5	Base64 加密格式的内容 MD5 检验值

实体首部	作用
Content-Range	内容的位置范围
Content-Type	内容的媒体类型
Expires	内容的过期时间
Last_modified	内容的最后修改时间

2. 常见状态码

> 状态码表示了响应的一个状态， 可以让我们清晰的了解到这一次请求是成功还是失败， 如果失败的话， 是什么原因导致的， 当然状态码也是用于传达语义的 。如果胡乱使用状态码，那么它存在的意义就没有了

2XX 成功

- 200 OK ，表示从客户端发来的请求在服务器端被正确处理
- 204 No content ，表示请求成功，但响应报文不含实体的主体部分
- 205 Reset Content ，表示请求成功，但响应报文不含实体的主体部分，但是与 204 响应不同在于要求请求方重置内容
- 206 Partial Content ， 进行范围请求

3XX 重定向

- 301 moved permanently ，永久性重定向，表示资源已被分配了新的 URL
- 302 found ， 临时性重定向，表示资源临时被分配了新的 URL
- 303 see other ，表示资源存在着另一个 URL ，应使用 GET 方法获取资源
- 304 not modified ，表示服务器允许访问资源，但因发生请求未满足条件的情况
- 307 temporary redirect ， 临时重定向，和 302 含义类似，但是期望客户端保持请求方法不变向新的地址发出请求

4XX 客户端错误

- 400 bad request ，请求报文存在语法错误
- 401 unauthorized ，表示发送的请求需要有通过 HTTP 认证的认证信息
- 403 forbidden ，表示对请求资源的访问被服务器拒绝
- 404 not found ，表示在服务器上没有找到请求的资源

5XX 服务器错误

- `500 internal sever error` ，表示服务器端在执行请求时发生了错误
- `501 Not Implemented` ，表示服务器不支持当前请求所需要的某个功能
- `503 service unavailable` ，表明服务器暂时处于超负载或正在停机维护， 无法处理请求

32.2 TLS

- `HTTPS` 还是通过了 `HTTP` 来传输信息，但是信息通过 `TLS` 协议进行了加密。
- `TLS` 协议位于传输层之上， 应用层之下 。首次进行 `TLS` 协议传输需要两个 `RTT` ，接下来可以通过 `Session Resumption` 减少到一个 `RTT` 。
- 在 `TLS` 中使用了两种加密技术，分别为：对称加密和非对称加密。

对称加密

- 对称加密就是两边拥有相同的秘钥，两边都知道如何将密文加密解密。
- 这种加密方式固然很好，但是问题就在于如何让双方知道秘钥 。因为传输数据都是走的网络， 如果将秘钥通过网络的方式传递的话，一 旦秘钥被截获就没有加密的意义的。

非对称加密

- 有公钥私钥之分， 公钥所有人都可以知道， 可以将数据用公钥加密，但是将数据解密必须使用私钥解密， 私钥只有分发公钥的一方才知道。
- 这种加密方式就可以完美解决对称加密存在的问题 。假设现在两端需要使用对称加密，那么在这之前， 可以先使用非对称加密交换秘钥。

> 简单流程如下： 首先服务端将公钥公布出去，那么客户端也就知道公钥了 。接下来客户端创建一个秘钥，然后通过公钥加密并发送给服务端， 服务端接收到密文以后通过私钥解密出正确的秘钥， 这时候两端就都知道秘钥是什么了。

TLS 握手过程如下图：

- 客户端发送一个随机值以及需要的协议和加密方式。
- 服务端收到客户端的随机值， 自己也产生一个随机值， 并根据客户端需求的协议和加密方式来使用对应的方式， 并且发送自己的证书（如果需要验证客户端证书需要说明)
- 客户端收到服务端的证书并验证是否有效，验证通过会再生成一个随机值， 通过服务端证书的公钥去加密这个随机值并发送给服务端， 如果服务端需要验证客户端证书的话会附带证书