<!--

- @Title:
- @Description:
- @Author: SoulCompiler
- @Email: yangtw7@gmail.com
- @Blog: SoulCompiler.github.io
- @Creation Date: 2021-06-14 17:38:21
- @LastEditors: SoulCompiler
- @LastEditTime: 2021-06-15 18:16:36

-->

第 23 章 UDP、TCP 和 SCTP

一.端口（重要的知名端口）、套接字

端口

- 服务器使用全局端口号：它们称为熟知端口号（well-known port number）
- 一个端口号确定了主机上的一个进程
- 划分
    - 1~1023: 熟知端口
    - 1024~49151: 注册端口, IANA 不分配不控制, 但可以注册来防止重复
    - 49152~65535: 动态端口, 可以有任何进程使用, 不需注册

套接字

- 套接字地址: 由 IP 地址和端口号组成, 唯一确定了一台主机上的一个进程
- 一个IP地址和一个端口号结合起来称为套接字地址。
- 传输层协议需要一对套接字地址：客户套接字地址和服务器套接字地址。
- IP头部包含IP地址，而UDP或TCP头部包含端口号。

二. UDP 协议

> UDP: User Datagram Protocol, 用户数据报协议

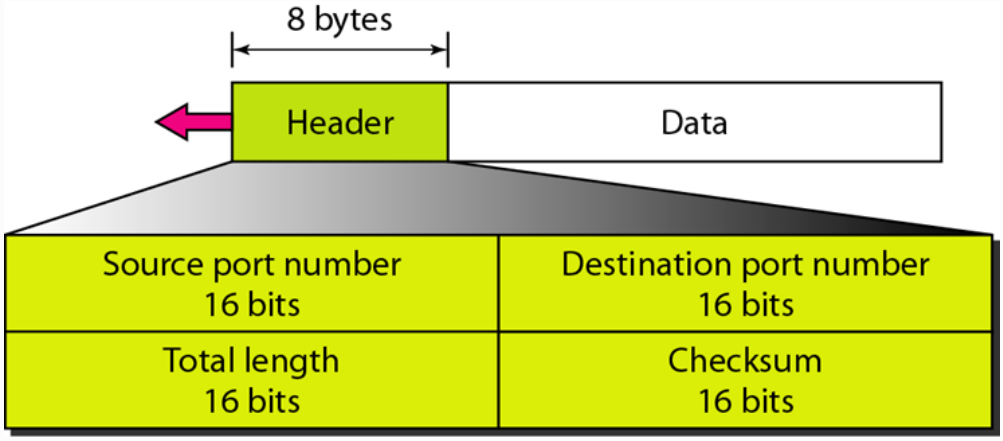
UDP 协议的概念、数据报结构、**校验和的计算**、UDP 的操作

UDP 协议的概念

- 无连接不可靠传输层协议。
- 提供进程到进程通信而不是主机到主机通信。
- 非常有限的差错检验。
-

Port	*Protocol*	*Description*
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
53	Nameserver	Domain Name Service
67	BOOTPs	Server port to download bootstrap information
68	BOOTPc	Client port to download bootstrap information
69	TFTP	Trivial File Transfer Protocol
111	RPC	Remote Procedure Call
123	NTP	Network Time Protocol
161	SNMP	Simple Network Management Protocol
162	SNMP	Simple Network Management Protocol (trap)

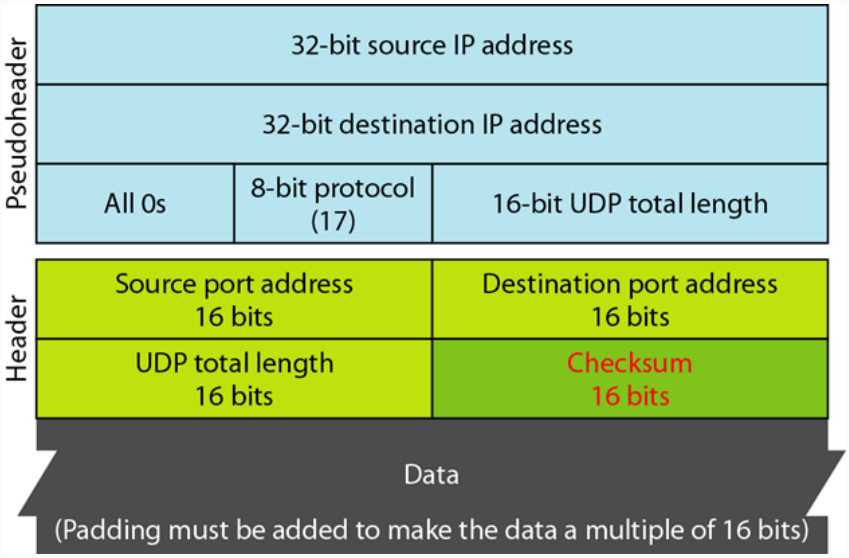
数据报结构

- 数据报结构图：
- 

- 头部固定 8 字节

- 源端口号: 如果是源主机是客户机, 则大概率是临时端口号; 服务器则大概率是熟知端口号
- 目的端口号: 类似上面, 不过反过来
- 总长度: 头部 (8 字节) + 数据长度
- 校验和
- 注意在真正的 UDP 头部中, 没有源地址和目的地址 (这是 IP 头部的东西), 但是校验和会用到它们

校验和的计算

- 校验包括三个部分：伪头部、UDP头部和从应用层来的数据。
- 

- 实际上是**伪头部**与头部共同的校验和
- 伪头部: 类似 IP 头部的最后 12 字节, 见上图
- 数据部分通过填充 0 补齐到 16 位的整数倍, 但**在发送时会被移除, 不进行填充**
- 在计算完校验和后, 伪头部和填充就被移除, 它们不会交给 IP
- 计算方法和之前一样, 16 位分组, 求和, 循环进位, 最后取反码
- 校验和是可选的, 若不计算则可在这个字段填充全 1 (因为真正的校验和不可能为全 1, 这说明和为 0)

UDP的操作

- 无连接服务：
    - UDP发送出去的每一个用户数据报都是一个独立的数据报。
    - 不同的用户数据报之间没有关系，即使它们都来自相同的源进程并发送到相同的目的程序。
    - 用户数据报不进行编号。
- 流量控制和差错控制：
    - 没有流量控制，因而也没有窗口机制。

    - 没有差错控制机制。
    - 使用UDP的进程必须要提供这些机制。
- 封装和拆封：
    - 将报文在IP数据报中进行封装和拆封。
- 排队：
    - 当一个进程想与多个进程通信时，它也只得到一个端口号，而最后也只有一个出队列和一个入队列。
    - 客户进程使用在请求中指明的源端口号将报文发送到出队列。UDP逐个将报文取出，加上UDP头部递交给IP。
    - 当报文到达客户端时，UDP要检查一下以确认对应于该用户数据报中目的端口号字段指明的端口号是否创建了入队列。如果有这样的入队列，UDP就将接收到的用户数据报放在该队列的末尾。如果没有这样的队列，UDP就丢弃该用户数据报，并请求ICMP协议向服务器发送端口不可达报文。当报文到达服务器时同理。

三. TCP：TCP 的特点与服务、段格式、三次握手（建立连接和拆除连接）、TCP 流量控制 （信贷滑窗协议）、TCP 差错控制、TCP 拥塞控制（慢启动过程、拥塞避免过程）

> TCP: Transmission Control Protocol, 连接控制协议

概念与特点

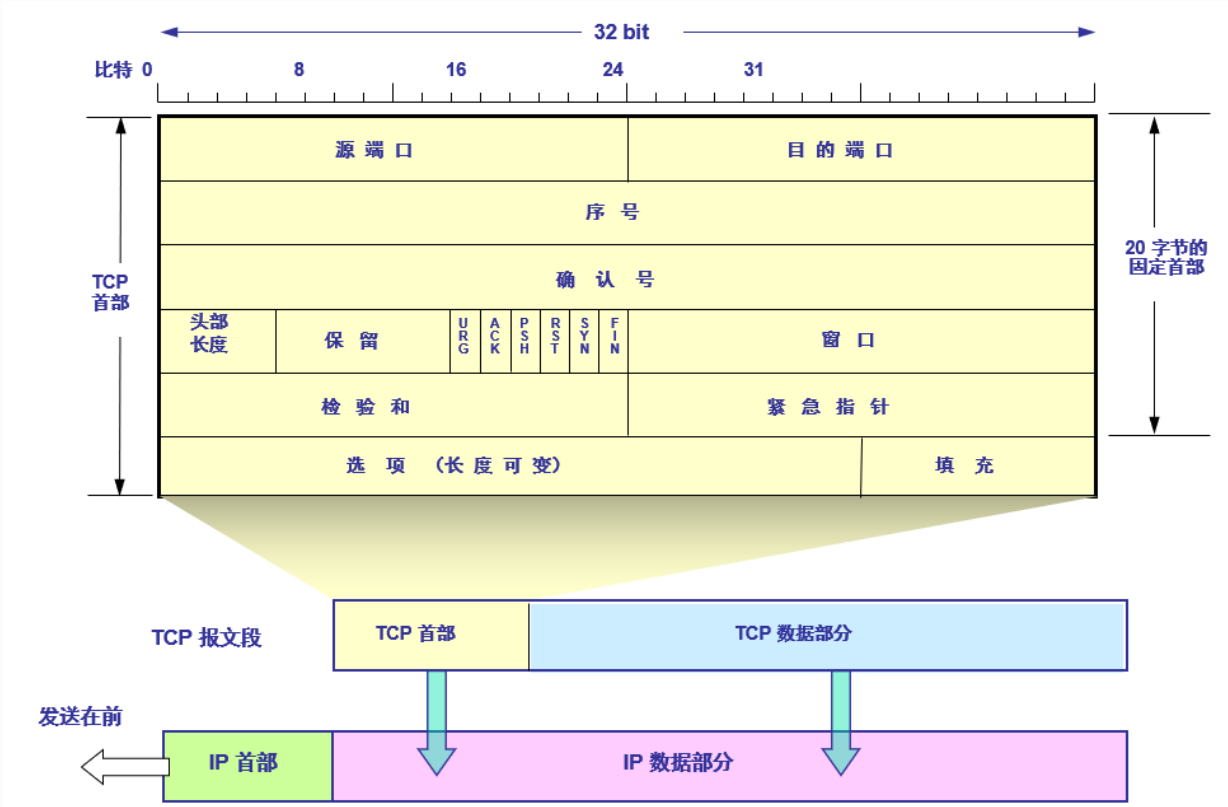
- 有连接 可靠 传输层协议
- 有流量控制和差错控制
- 服务:
    - 提供进程到进程的通信
    - 提供流传递服务, 类似于一个假想的管道, 存在缓冲区
    - 全双工通信
    - 面向连接
- 使用序列号 seq, 值为该段包含的第一个(数据)字节的序号
- 使用确认号 ack, 值为接收方预期接受的下一个字节的序号

TCP 的特点与服务

- TCP是面向连接的、可靠的传输协议。它为IP服务增加了面向连接和可靠性的特性。
- 服务：
    - 进程到进程的通信：
        - TCP用端口号提供进程到进程的通信。
    - 流传递服务：
        - TCP允许发送进程以字节流形式传递数据，并且接收进程也以字节流形式接收数据。
    - 发送和接收缓冲区：
        - 每一个方向都存在一个缓冲区：发送缓冲区和接收缓冲区。
        - 实现缓冲的一种方法是使用一字节存储单元的循环数组。
        - 缓冲实现图：
    - 全双工通信：
        - 数据可以在同一时间双向流动。
        - 每一方TCP都有发送和接收缓冲区，它们能在双向发送和接收段。
    - 面向连接的服务：
        - 具体过程如下：
            1. 在两个TCP之间建立一个连接；
            2. 在两个方向交换数据；
            3. 连接终止。
        - 这是一个虚连接，而不是一个物理连接。
    - 可靠的服务：
        - 使用确认机制来检查数据是否安全和完整地到达。
- 特点：
    - 序号系统：
        - TCP在段的头部采用称为序号和确认号的两个字段。
        - 在每个连接中传送的字节都由TCP编号，序号开始于一个随机产生的数。
        - 一个段的序号字段的值定义为该段包含的第一个字节的序号。

        - 段中确认字段的值定义了通信一方预期接收的下一个字节的编号。确认号是累加的。
    - 流量控制：
        - 数据的接收方控制发送方发送数据的数量，这样做是为了防止接收方数据溢出。序号系统允许使用面向字节的流量控制。
    - 差错控制：
        - 差错控制以段作为差错检测（丢失或损坏段）的数据单元。
    - 拥塞控制：
        - 发送方发送的数据量不仅由接收方控制（流量控制），而且还要由网络中的拥塞程度决定。

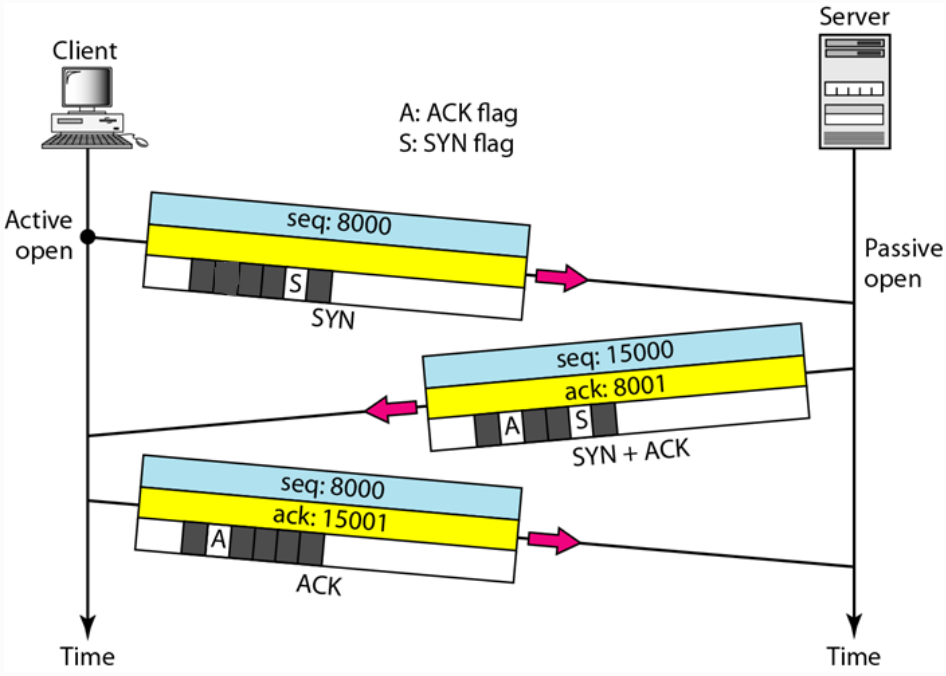
段格式

- 

- 源端口地址, 目的端口地址: 和 UDP 中的一样
- 序列号 seq: 段中数据的第一个字节的序号, 连接建立时双方各自通过随机数产生初始序列号 (ISN)
- 确认号 ack: 段的接收方期望接受的下一个字节号
- 头部长度: 单位为**4字节**, 也就是说该字段的值为头部长度除以 4
- 保留: 就是保留
- 六个控制位

    - URG: 紧急指针有效, 见后面
    - ACK: 该帧包含确认
    - PSH: 请求急迫/发送数据 (其实没啥用)
    - RST: 连接复位 (GFW经常就靠这个阻断你的连接)
    - SYN: 同步序列号
    - FIN: 终止连接
- 窗口: 定义了接收窗口 (rwnd, Receiver Window) 的大小, 见阻塞控制
- 校验和: 计算方法一样, 也有伪头部, 伪头部 + 头部 + 数据, 但是书上没说 TCP 的伪头部是啥
- 紧急指针: 指向紧急字节的最后一个字节, 紧急字节总是位于段的开始, 收到后会被直接送往进程, 无需等待

三次握手（建立连接和拆除连接）

- 建立连接三次握手过程图：
- 

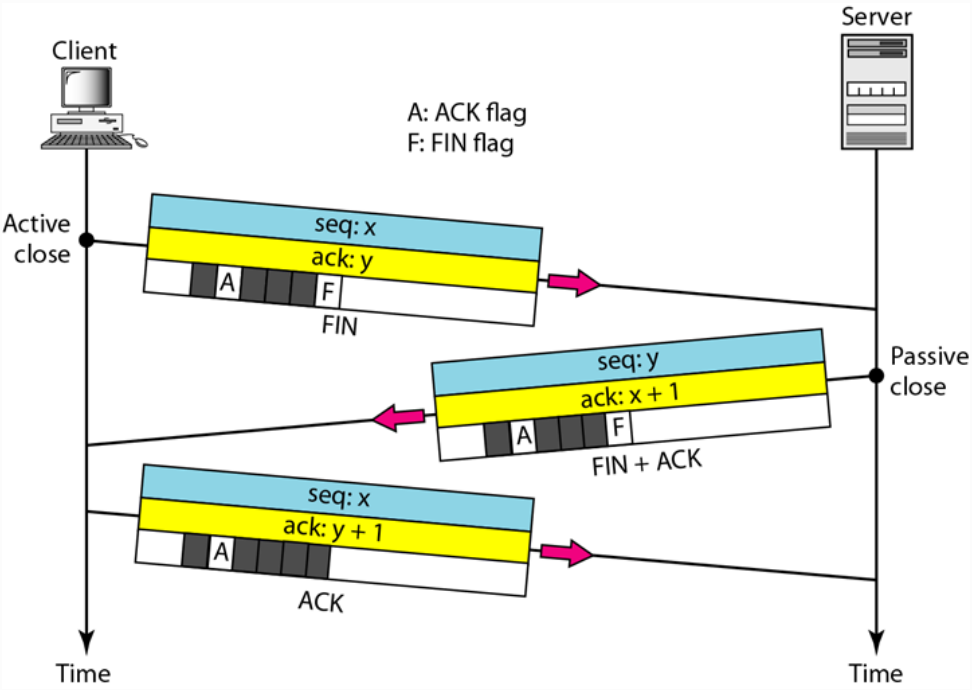
  采用三次握手

    - 客户端: SYN (仅包含己方 seq, SYN 段不携带数据, 但占用序列号)
    - 服务器: SYN + ACK (包含服务器的 seq 和 ack, 和单纯的 SYN 一样占用序列号)
    - 客户端: ACK (若 ACK 段不携带数据, 则不占用序列号)

  存在 SYN 洪泛攻击的问题

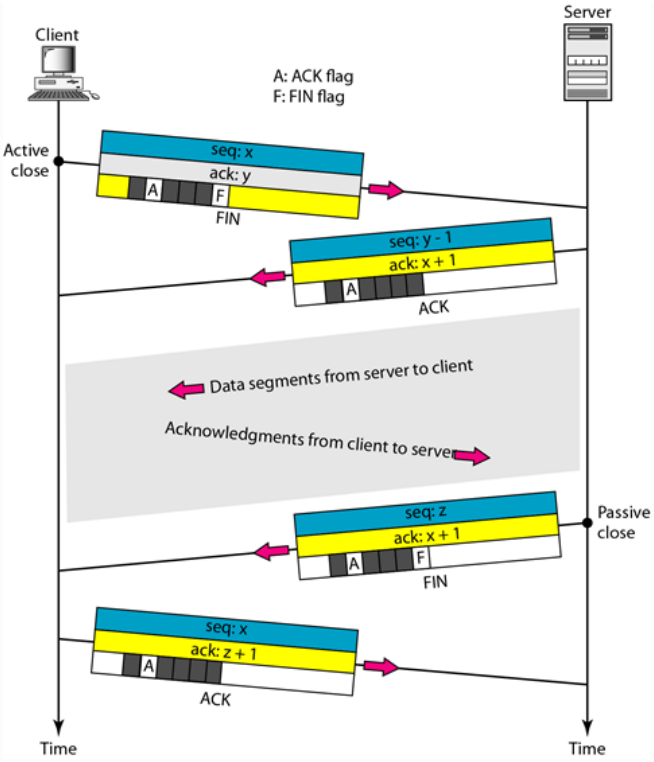
    - 攻击方伪造大量的 SYN, 伪装成正常的连接请求
    - 由于服务器需要为每一个请求分配资源, 因此很快被耗尽
    - 最终服务器无法接受新的正常用户的请求 -> 拒绝服务攻击 (DoS, Denial of Service)

> DDoS, Distributed Denial of Service, 就是分布式拒绝服务攻击

- 拆除连接三次握手过程图：
- 

- 拆除连接三个步骤：
    1. 从客户进程接收到一个关闭命令后，客户的TCP发送第一个段：FIN段， 即其中的FIN位置位。
        - 如果FIN段不携带数据，则该段占用一个序列号。
    2. 服务器TCP接收到FIN段后，通知它的进程，并发送第二个段：FIN + ACK段，证实它接收到来自客户端的FIN段，同时通告另一端连接关闭。
        - 如果FIN + ACK段没有携带数据，则该段仅占用一个序列号。
    3. 客户端的TCP发送最后一段：ACK段，证实它接收到来自服务器的FIN段。这个段包含确认号，它是接收到来自服务器FIN段的序号加1。
        - 这个段不携带数据也不占用序列号。
- 主动方: FIN, 可携带数据, 占用一个序列号
- 被动方: FIN+ACK, 可携带数据, 占用一个序列号
- 主动方: ACK, 不占用序列号

四次挥手



- 主动方: FIN
- 被动方: FIN+ACK
    - 此时进入**半关闭**
    - 主动方不能够再发送数据, 但可以接收数据
    - 被动方还能够继续发送数据
- 被动方: FIN
- 主动方: ACK

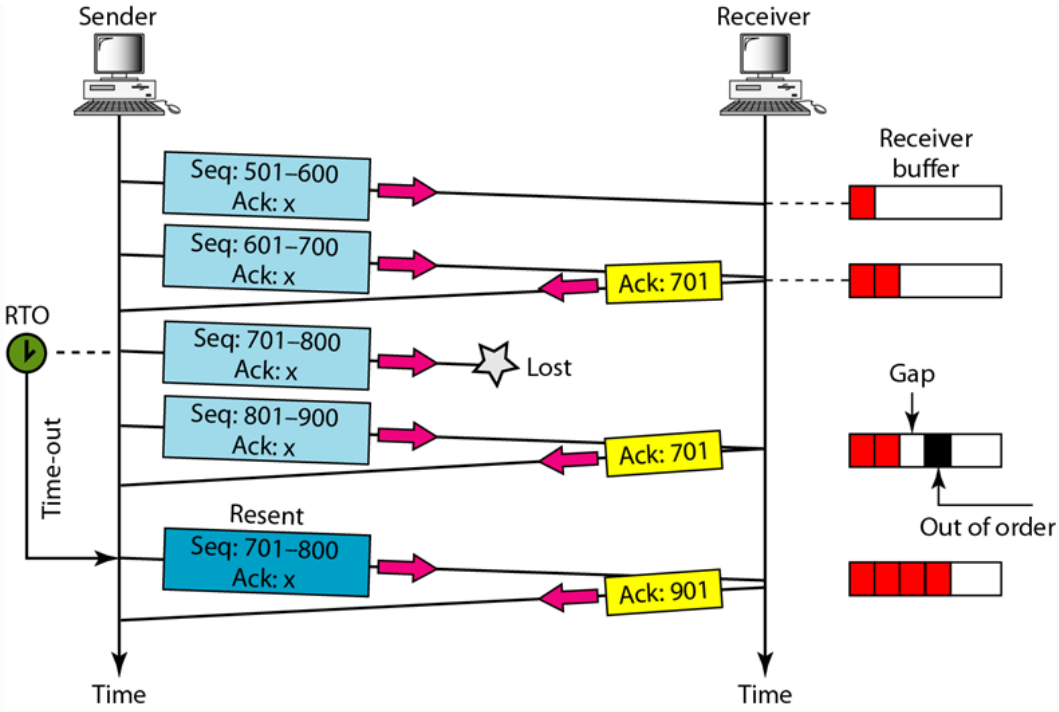
TCP 流量控制 （信贷滑窗协议）

- TCP使用的滑动窗口协议界于回退N帧与选择重发之间的滑动窗口。
- TCP的滑动窗口与数据链路所用的滑动窗口有两大点不同：
    - TCP的滑动窗口是面向字节的，而数据链路层讨论的滑动窗口是面向帧的。
    - TCP的滑动窗口是可变大小，而数据链路层讨论的滑动窗口是固定大小。
- TCP滑动窗口示意图：

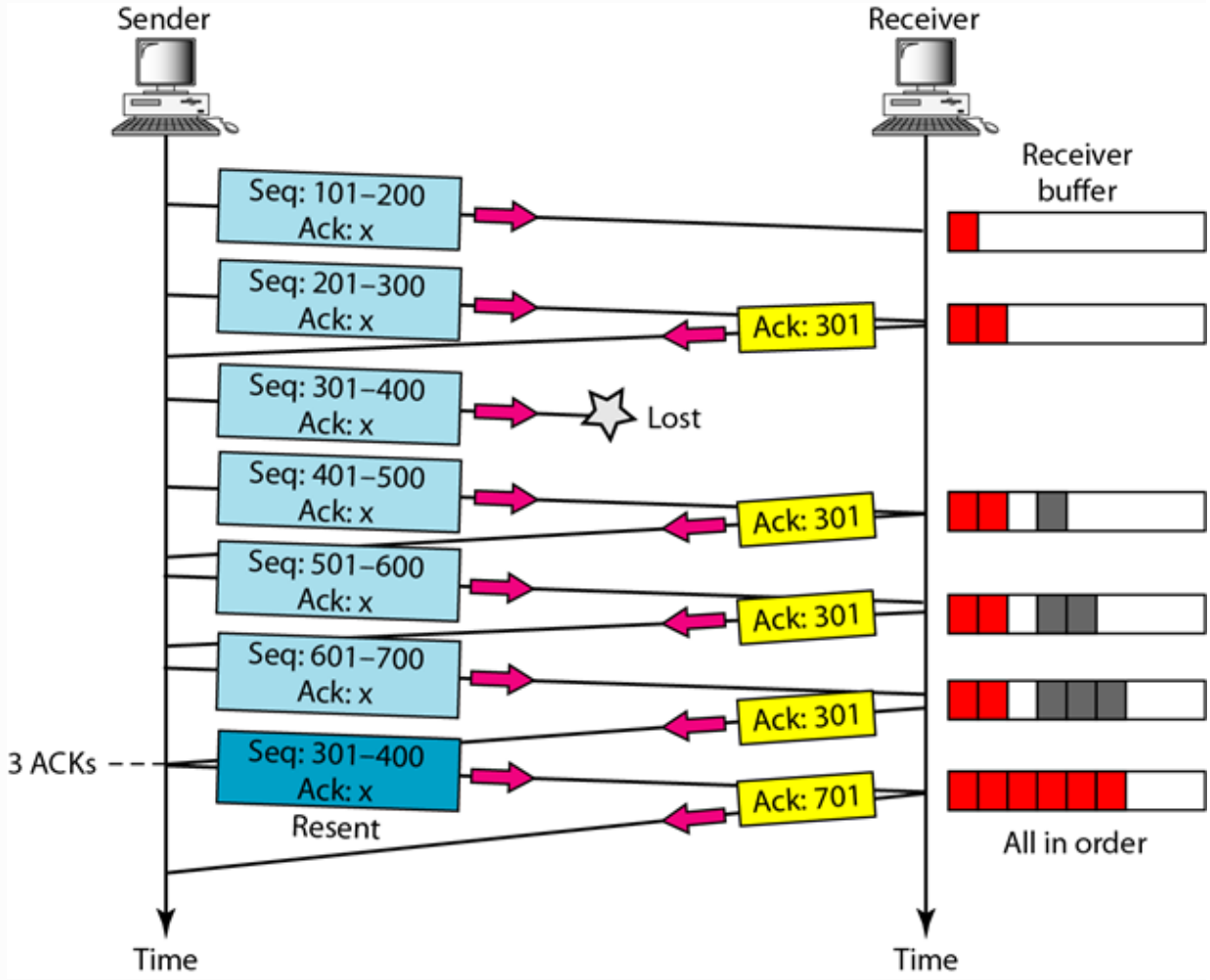
TCP 差错控制

- TCP中的差错检测和 纠正是通过三种简单工具来完成：校验和、确认和超时。
- 校验和：

    - 每个段都包括校验和字段，用来检查受到损坏的段。
    - 如果段被损坏，它将被目的端TCP丢弃，并认为是丢失了。
    - TCP在每段中强制使用一个16位的校验和。
- 确认：
    - 使用确认方法来证实收到了数据段。
    - 不携带数据但占用序列号的一些控制段也要确认，但ACK段不需要确认。
- 重传：
    - 出现损坏, 丢失或延迟时重传段
    - 在实现中, 分为重传计时器到时和连续收到 3 个 ACK
    - 重传计时器到时:
        - 已发送未确认的(占用序列号的)段有一个定时器
            - 称为 RTO 计时器 (Retransmission Time-Out)
            - 其值根据往返时间动态更新
            - 对 ACK 段不设置定时器
        - 定时器到时时重发最早的重要的段
    - 连续收到三个 ACK:
        - 触发**快速重传**
        - 立即发送缺少的段
    - 失序的段
        - 暂时保留, 直到缺少的那个段到达
        - TCP 保证提交给进程的数据是顺序正确的



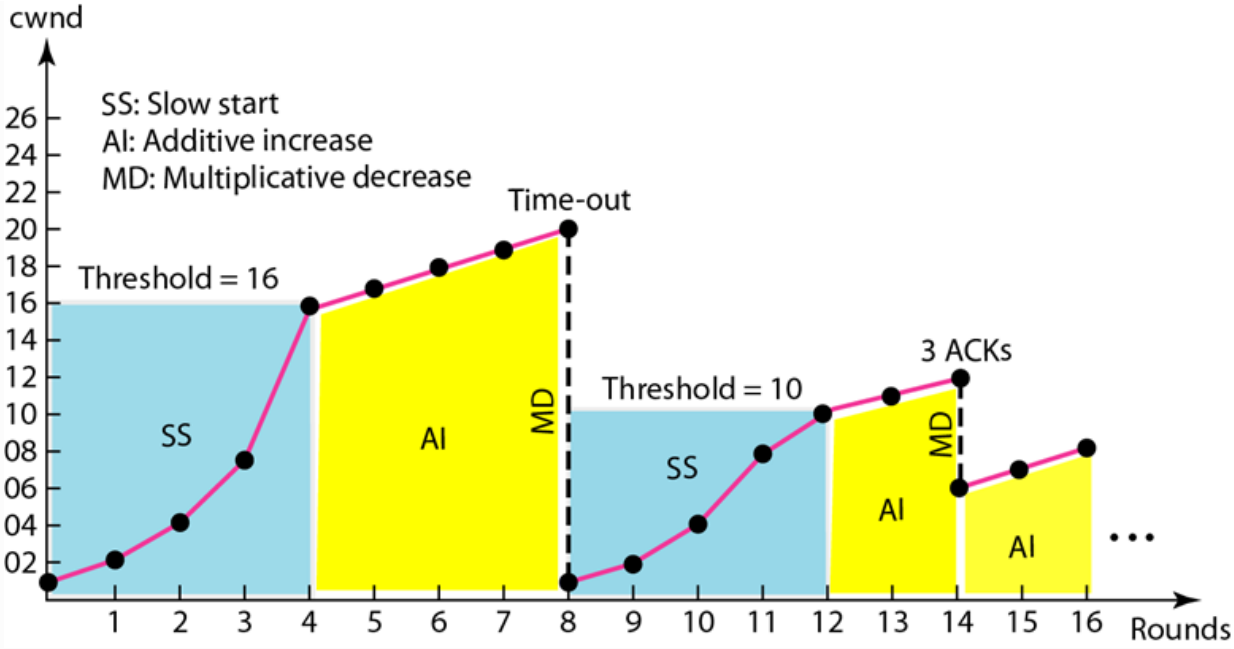
- 重传计时器超时重传



> 快速重传, 重传时 seq 301~400 的计时器还没到时, 但依然重传

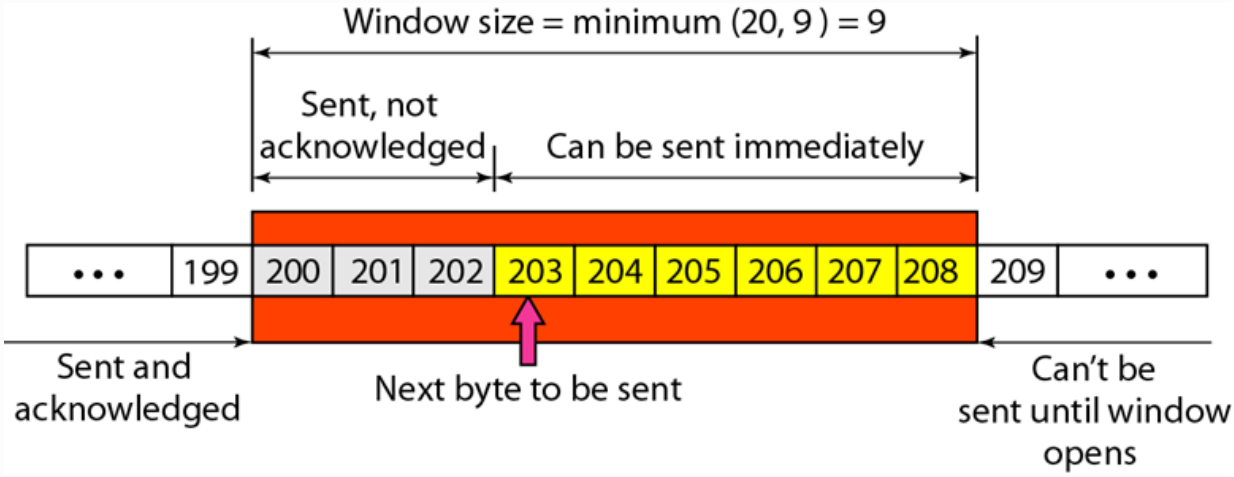
TCP 拥塞控制

- 在这里确定 cwnd 大小 (Congestion Window, 拥塞窗口)
- 采用三个阶段: 慢速启动, 拥塞避免, 拥塞检测
- 慢速启动: 指数增长
    - 开始时 cwnd = 1 MSS (最大段长度, 在连接建立时确定, 相当于这时窗口大小为一个段)
    - 每一个段的确认都会使 cwnd+1, **按照指数规律增长**
    - 到达**阈值**时, 结束慢速启动阶段
- 拥塞避免: 加性增加
    - 每次窗口中所有段均被确认后才有 cwnd+1
    - 直到检测到拥塞
- 拥塞检测: 乘性减少
    - 若重传计时器到时, 说明拥塞可能很严重: 段丢失且一直没收到相关信息
        - 强烈反应
            - 阈值设置为 cwnd 的一半
            - cwnd = 1 MSS
            - 开始慢速启动阶段
        - 相当于阈值变为实际上限的一半, 然后重启传输 (速度方面)
    - 若接收到三个 ACK, 说明可能有轻度拥塞, 一些段可能已经成功到达, 仅有部分缺失
        - 轻度反应
            - 阈值设置为 cwnd 的一半
            - cwnd = 阈值 (部分实现是 阈值+3)
            - 开始拥塞避免阶段 (就是加性增加)



> SS: 慢速启动 (指数增长)
>
> AI: 加性增加
>
> MD: 乘性减少
>
> Threshold: 阈值

TCP 流量控制 [IMP]



- 采用滑动窗口, 面向字节, 又称信贷滑窗协议
- 窗口大小为头部中的接收窗口 (rwnd) 与 cwnd 较小值
  - rwnd 表明了接收方当前可用的窗口大小
- 在窗口中, 左侧为已发送但未确认的, 右侧为可立刻发送的
- 在窗口外, 左侧为已发送且已确认的, 右侧为未发送且尚不可发送的
- 发送方接收到 ACK 后, 可以合拢窗口: 左侧向右移动
- 当 rwnd 或 cwnd 更新时, 可以张开窗口: 右侧向右移动, 但不建议收缩 (右侧向左移动)

- 窗口左边沿不能向左移动