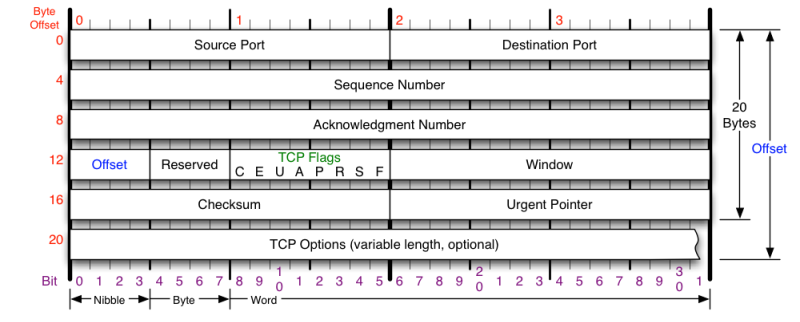
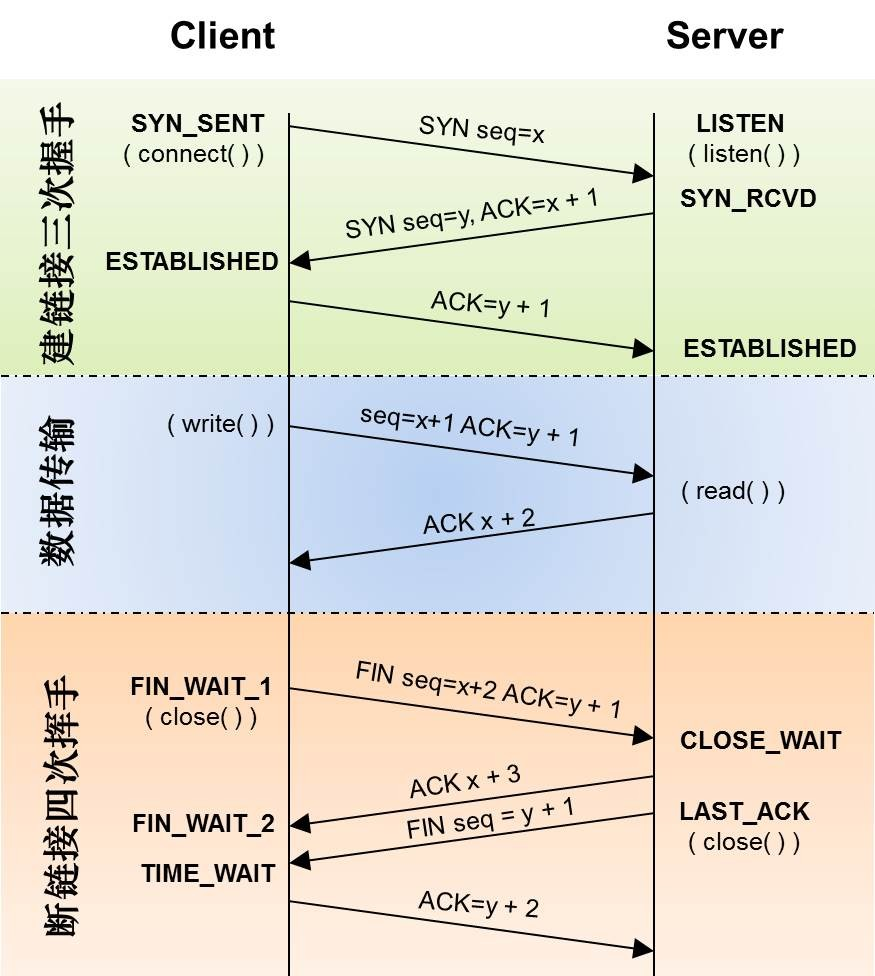
TCP和拥塞控制

1. TCP基础
2. Header



* Sequence Number 包的序号（Seq），增加数为包的长度
* Acknowledgement Number 确认序号（Ack）
* Window 窗口大小（接收缓冲区大小）
* TCP Flag 包的类型

1. 状态机



* 1. FLAGS:
     1. SYN：同步序列编号（Synchronize Sequence Numbers）
     2. ACK：确认序号有效
     3. FIN：释放一个连接（Finish）
  2. Entries & Parameters:
     1. MSL: Maximum Segment Lifetime
     2. SYN flood: SYN包攻击
     3. tcp\_tw\_reuse: TIME\_WAIT重用
     4. tcp\_tw\_recycle: TIME\_WAIT回收
     5. tcp\_max\_tw\_buckets: TIME\_WAIT最大数量

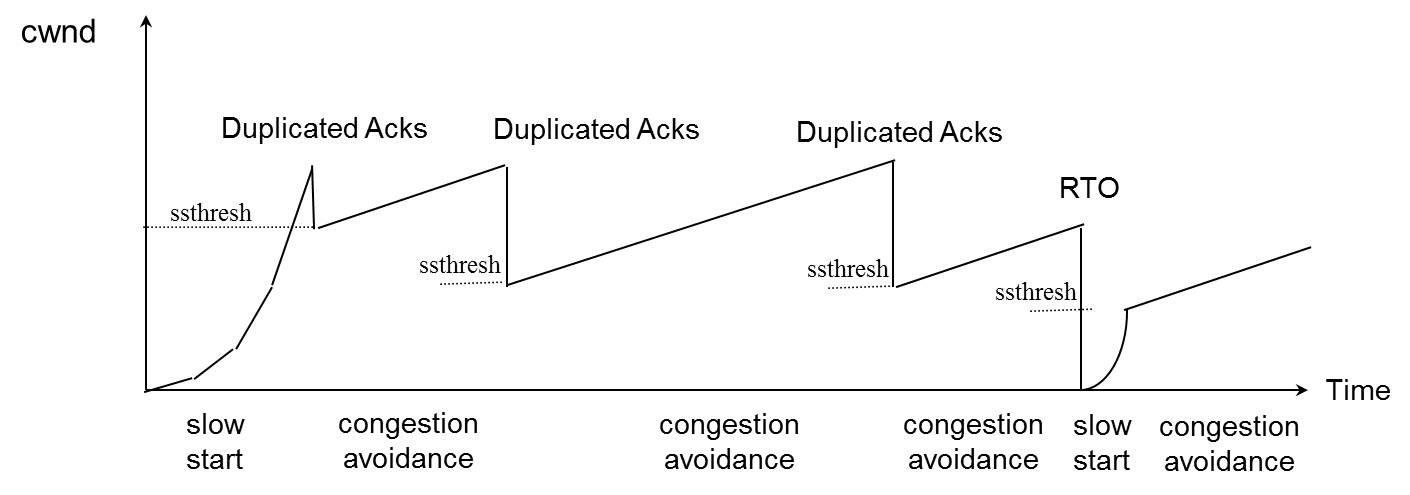
1. TCP重传
   1. 超时重传（Timeout Retransmit）
      1. sender死等ACK，超时重传
   2. 快速重传（Faster Retransmit）
      1. receiver发送最后一个ACK
      2. 收到3个重复的ACK，则重传该包
   3. SACK（Selective Acknowledgment）
      1. Header中增加SACK选项，记录接收端收到的数据包的SeqNum范围
      2. Parameters: tcp\_sack
   4. D-SACK（Duplicate Selective Acknowledgment）
      1. SACK中记录重复的SeqNum范围（ACK范围大于于SACK范围）
      2. Parameters: tcp\_dsack
2. TCP的RTT（Round Trip Time）
   1. 经典算法（RFC792）
      1. Smoothed RTT:

SRTT = (α \* 历史SRTT) + ((1-α) \* RTT)

* + 1. Retransmission TimeOut:

RTO = min [UBOUND, max [LBOUND, (β \* SRTT)]]

* + 1. α = [0.8, 0.9]
    2. β = [1.3, 2.0]
    3. UBOUND = 最大timeout时间
    4. LBOUND = 最小timeout时间
  1. Karn / Partridge 算法
     1. 忽略重传的RTT
     2. 发生重传时，对RTO翻倍
  2. Jacobson / Karels 算法
     1. SRTT = 历史SRTT + α\*(RTT – 历史SRTT)
     2. DevRTT = (1-β) \* DevRTT + β \* (|RTT-SRTT|)
     3. RTO= µ \* SRTT + ∂ \*DevRTT
     4. α = 0.125
     5. β = 0.25
     6. µ = 1
     7. ∂ = 4

1. TCP滑动窗口
   1. sender窗口（由receiver的ACK，窗口向右移动）
      1. #1 已发送，已收到ACK
      2. #2 已发送，未收到ACK
      3. #3 未发送，有空间
      4. #4 未发送，无空间
   2. zero window
      1. receiver缓冲区满，sender隔段时间发送ZWP（Zero Window Probe），让receiver发送ACK告知窗口大小
      2. Silly Window Syndrome
         1. 数据塞不满MTU（Maximum Transmission Unit），浪费带宽
         2. 解决方案：
            1. David D Clark算法：windowSize小于某值，receiver回复windowSize（由receiver引起的）
            2. Nagle算法：sender将多个小包合并成一个大包发送（由sender引起的）
2. TCP拥塞控制
3. 拥塞控制算法
   1. 慢启动（Slow Start）
      1. 初始cwnd（Congestion Window）=1
      2. 每收到一个ACK，cwnd++（线性增长）
      3. 每过一个RTT，cwnd=2\*cwnd（指数增长）
      4. cwnd>=ssthresh（slow start threshold，一般为65535 Byte）时，进入拥塞避免（Congestion Avoidance）
   2. 拥塞避免（Congestion Avoidance）
      1. 每收到一个ACK，cwnd=cwnd+1/cwnd
      2. 每过一个RTT，cwnd++
   3. 快速重传（Faster Retransmit）
      1. 发生快速重传（收到3个duplicated ACK）时，cwnd/=2，ssthresh=cwnd
      2. 进入快速恢复（Fast Recovery）
   4. 快速恢复（Fast Recovery）
      1. cwnd=ssthresh+3\*MSS（Maximum Segment Size）
      2. 重传丢失的包（根据Duplicate ACK）
      3. 再收到一个Duplicated ACK，cwnd++
      4. 收到新的ACK时，cwnd=ssthresh，进入拥塞避免
   5. TCP New Reno
      1. 没有SACK时，发生快速重传，进入Fast Recovery
      2. 在Fast Recovery阶段，根据部分 ACK（Partial Acknowledgment）机制，继续恢复丢失的包，直到所有丢失的包被确认，再进入拥塞避免
   6. FACK（Forward Acknowledgment）
      1. SND.UNA：未被确认的最小序列号（发送窗口左端）
      2. SND.NXT：下一个即将发送的序列号
      3. FACK：SACK 选项中 最高确认序列号
      4. 未确认数据量（inflight data）：

inflight=SND.NXT−FACK

* + 1. 丢包检测：

当inflight突然下降时，sender认为FACK之前的包丢失，重传丢失的包