






# 传输层

 Courses	 计算机网络
<input checked="" type="checkbox"/> Done	<input checked="" type="checkbox"/>
 Status	Done

## 传输层服务

- 多路分解和多路复用
  - 多路分解：将传输层报文段中的数据交付到正确的套接字的工作
  - 在源主机从不同套接字中收集数据块，并为每个数据块封装上**首部信息 (transport header)** 从而生成报文段，然后将报文段交付给网络层
  - UDP：来自不同主机/同一主机不同端口，可能会被定向到同一套接字
  - TCP：四元组标识，
- 编址
  - 套接字 `<HostIP, Port>`
- 面向连接
  - 通信双方在实际数据传输前需要建立一个逻辑连接，并在数据传输完成后需要释放该连接。
- 流控制
  - todo
- 可靠传输
  - 检验和：检测分组是否损坏
  - ACKs：接收方告诉发送方已经收到了分组
  - NACK：没有收到
  - 序列号：识别分组，按顺序交付

- 重传：重新发送分组
- 超时：何时重新发送分组

## 方法

- 分组损坏？ACK & NACK
- ACK丢失怎么办？序列号
- 分组丢失？TimeOut

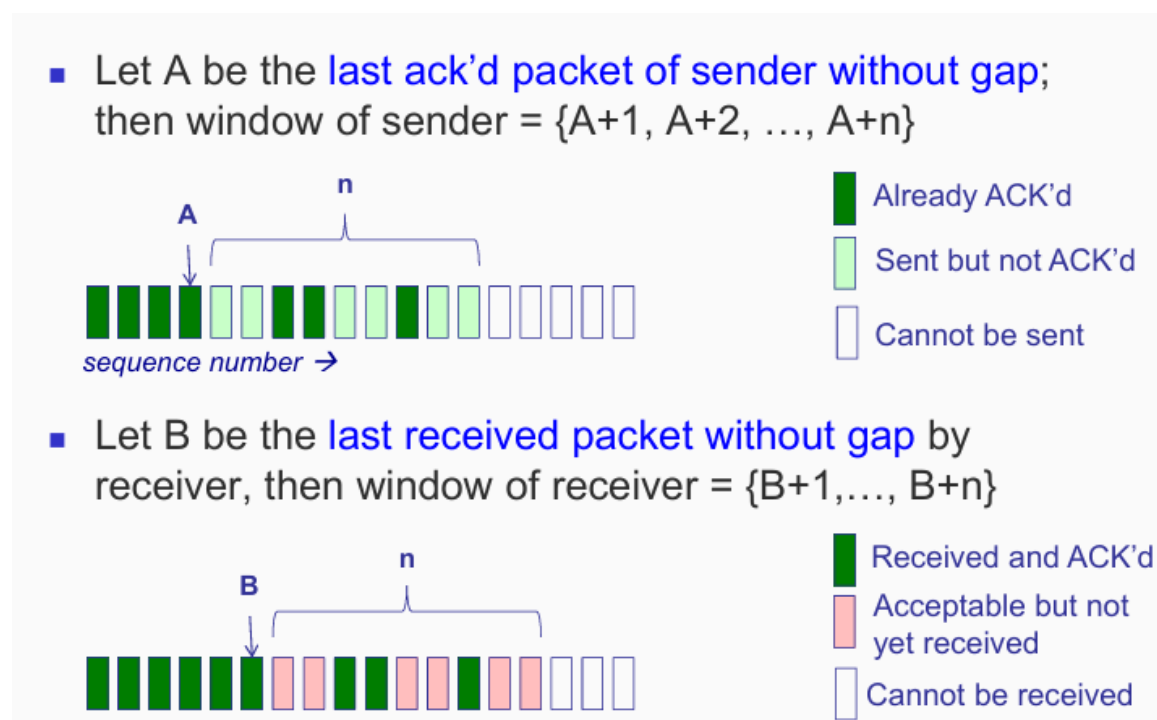
# 可靠传输协议的设计

## 等停

效率低下  $if\ RTT > D_{trans}$

## 流水线协议

- 滑动窗口



- ACK的形式
  - 累计ACK：携带期望下一个按顺序接受的序列号
  - 选择ACK：单独确认正确接收的数据包（更精确，但实现复杂）

## 滑动窗口协议

- 回退N步GBN

Receiver: 累计ACK

Sender: 为第一个未被ACK的分组维护计数器, 如果超时, 重传所有pkt (实验中所实现的)

- 选择重传SR

重传效率高, 但记录复杂, 需要为每个数据包维护一个计时器

- 二者比较

- 何时GBN更优? 错误率低时, 否则浪费带宽
- 何时SR更优? 错误率高时, 否则过于复杂

特性	Go-Back-N (GBN)	Selective Repeat (SR)
发送窗口	允许发送窗口大小内的多个数据包	允许发送窗口大小内的多个数据包
接收机制	只能按序接收数据包, 乱序到达的数据包被丢弃	可以按任意顺序接收数据包, 并缓存乱序数据包
确认机制	累计确认, 接收方只确认最后一个按序到达的数据包	单独确认, 每个数据包都有独立的确认
丢包处理	丢包后重传该数据包及其后的所有数据包	丢包后只重传丢失的数据包
实现复杂度	实现简单, 接收方只维护一个序号	实现复杂, 接收方需维护缓存和乱序处理
存储需求	较低, 接收方不需缓存乱序数据包	较高, 接收方需缓存所有未按序到达的数据包
适用场景	适用于低带宽-延迟产品环境	适用于高带宽-延迟产品环境

## 传输层协议

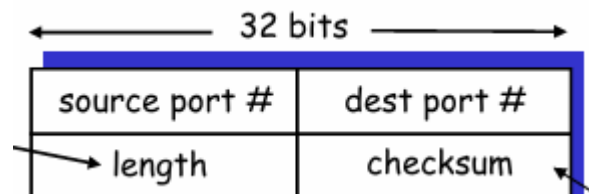
### UDP

尽力而为: 可能丢失、失序

无连接: 没有握手、每个报文段独立

使用场景: 流式多媒体app (容忍分组丢失, 对速率敏感)、DNS、SNMP

## UDP头部:



共8字节

## UDP检验和:

加法回卷, 检验: 全1正确, 方法: 16比特相加\*3=checksum

## TCP

### What TCP use?

检验和

序列号 (字节偏移)

发送方和接收方的滑动窗口

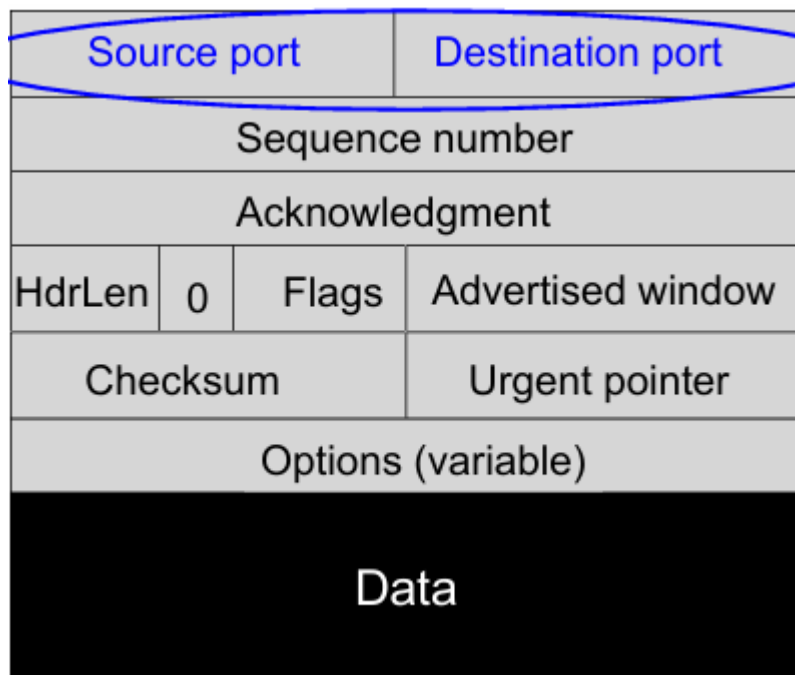
接收方发送累计ACK (发送方维护单个计时器)

接收方缓存无序分组

快速重传+超时估计算法

## TCP头部

20bytes (相比于UDP的8bytes)



- Sequence number使用的是字节偏移量，TCP 假定存在传入的**数据流**，并尝试将其传送到应用
- ACK使用的是**next expected byte** **【in order】**
  - Sender: seqno=X, length=B
  - Receiver: ACK=X+B
  - Sender: seqno=X+B, length=B
  - Receiver: ACK=X+2B
  - Sender: seqno=X+2B, length=B

## TCP重传

快速重传：在收到3个冗余ACK时触发，不用等待超时直接重传期待的分组

超时重传：如何选择timeout的值？

- RTT estimation:
  - Sample RTT: 实际的RTT (仅一次)
  - RTT均值:  $\text{Estimated RTT} = (1-a) \text{ Estimated RTT} + a * \text{sample RTT}$  其中a为0.125
    - 该方法称为**指数加权移动平均**

- RTT偏差:  $DevRTT$
- $DevRTT = (1 - \beta)DevRTT + \beta|SampleRTT - EstimatedRTT|$
- $\beta$ 推荐值0.25
- $RTO = EstimatedRTT + 4 \times DevRTT$

## 连接的建立

### 三次握手

用到Flag字段中的**SYN**

A→B

1. A: send无数据TCP、SYN=1、seq=random
2. B: send**SYNACK**、SYN=1、seq=random、ack=..+1
3. A: ack=..+1、SYN=0、可正常发送数据

**必要性:**

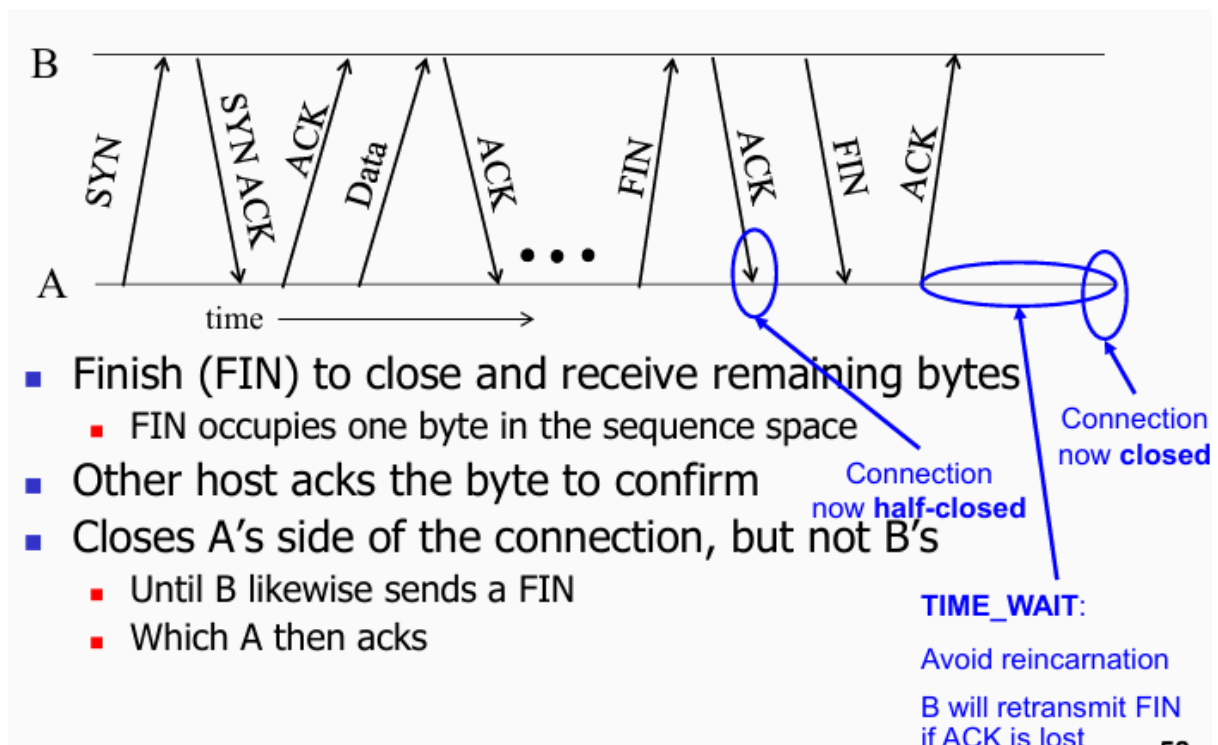
保证双方同步

**防止重复连接:** 网络中**滞留的旧SYN包**导致的连接混乱

确认双方的接收能力

## 连接的拆除

### 四次挥手



四次挥手后有一段TIME\_WAIT时间，以防ACK丢失

## TCP流量控制—信用量窗口

- 发送方
  - 维护RWND
  - $rwnd \leq bufferSize - [lbRcvd - lbRead]$
  - 跟踪 lastByteSend和lastByteAcked两个变量
  - 保证  $LBSend - LBAcked \leq rwnd$
- 接收方
  - 维护RecvBuffer
  - 跟踪两个变量：
    - lastByteRead：从buffer中读出的最后一个字节编号
    - lastByteRcvd：从网络中到达的已放入缓存的最后一个字节编号
    - **lbRcvd - lbRead** 即为 buffer中还存在的字节数
    - ACK包含：AN=i (ack number) , W=j (window size)
    - AN是期待的下一个八位字节

- 通知发送方：允许发送 $W=j$ 八位字节

## TCP拥塞控制

1. 发送方：如何限制发送流量速率？

cwnd：拥塞窗口—通过拥塞控制算法得出

发送方窗口  $\leq \min\{cwnd, rwnd\}$

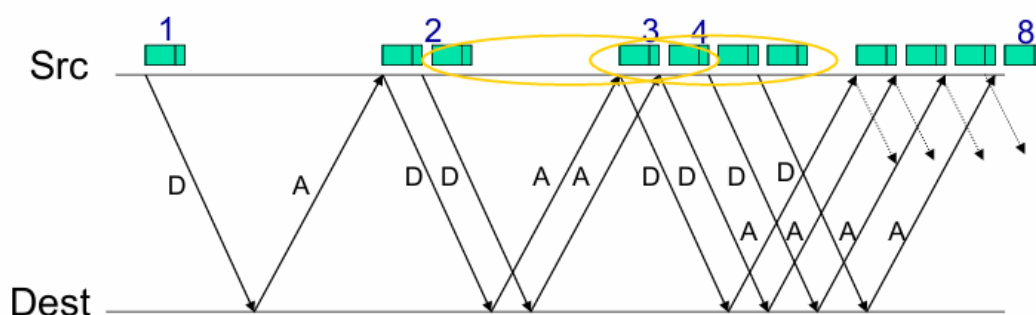
2. 发送方：如何感知到拥塞？

路由器在congestion时告知主机

## TCP拥塞控制算法

- 慢启动

- MSS：最大报文段长度=MTU-TCP&IP header，通常为1460
- 初始，cwnd=1，发送速率=MSS/RTT
- 如果没有损失，每个RTT的cwnd翻倍



何时停止？

引入慢启动阈值 `slow start threshold` : **ssthresh**

**if cwnd > ssthresh, 停止慢启动**

- AIMD：加性增、乘性减

- 当cwnd大于等于ssthresh时，每个ACK使cwnd的值增加一个MSS
- 3个冗余ACK：
  - $ssthresh = cwnd/2$
  - $cwnd = ssthresh$



- 进入拥塞避免 (cwnd一次增加1)
- timeout
  - ssthresh=cwnd/2
  - cwnd=1
  - 进入慢启动
- 快速恢复
 

当收到三个dupACK时: ssthresh=cwnd/2    **cwnd=ssthresh+3**

收到ACK时, **cwnd=ssthresh**

## 数据网络中的拥塞控制

- 抑制分组Choke Packet
  - 拥塞节点产生, 发送给源主机, 为每个丢弃的数据包发送ICMP
- 反压
  - 一跳一跳的Choke Packet
  - 传播时间大于传输时间。要求路径上的每一条减少传输
- 警告位 / ECN
  - 数据包头中的单个位
  - 作为拥塞的早期指标
- 随机早期丢弃RED
  - 路由器在缓冲区完全满之前随机丢弃数据包
  - RED算法
    - RED算法通过监控路由器队列的平均长度来判断网络的拥塞状态, 并在队列长度超过某个阈值时以一定概率丢弃进入队列的数据包。这样可以在拥塞发生前控制流量, 防止队列溢出。

## RED算法的步骤

1. **计算平均队列长度**: 使用指数加权移动平均法 (EWMA) 计算队列的平均长度:

$$\text{avg} = (1 - w_q) \cdot \text{avg} + w_q \cdot \text{current\_queue\_length}$$

其中,  $w_q$  是权重因子, 通常设置为一个较小的值。

2. **判断丢包条件**:

- 如果  $\text{avg} < \text{min\_th}$ , 不丢弃数据包。
- 如果  $\text{avg} > \text{max\_th}$ , 丢弃所有新到达的数据包。
- 如果  $\text{min\_th} \leq \text{avg} \leq \text{max\_th}$ , 以一定概率丢弃数据包。

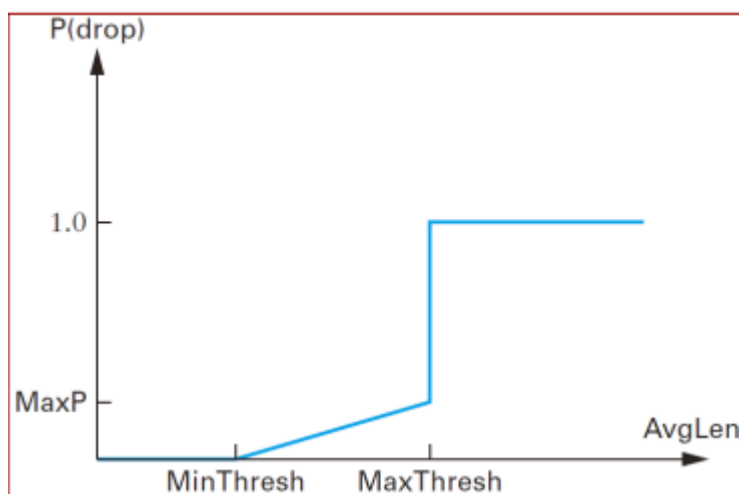
3. **计算丢包概率**:

- 丢包概率  $p$  随着  $\text{avg}$  的增加线性增加:

$$p = p_{\max} \cdot \frac{\text{avg} - \text{min\_th}}{\text{max\_th} - \text{min\_th}}$$

其中,  $p_{\max}$  是最大丢包概率。

4. **随机丢弃数据包**: 根据计算的丢包概率随机决定是否丢弃到达的数据包。

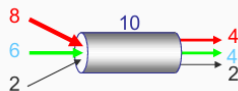


- 公平队列

- 路由器将数据包分类为“流”, 每个流都有自己的FIFO队列
- Max-Min Fairness

## Example

- $C = 10$ ;  $r_1 = 8$ ,  $r_2 = 6$ ,  $r_3 = 2$ ;  $N = 3$
- $C/3 = 3.33 \rightarrow$ 
  - $r_3$ 's need is only 2
    - » Can service all of  $r_3$
  - Remove  $r_3$  from the accounting:  $C = C - r_3 = 8$ ;  $N = 2$
- $C/2 = 4 \rightarrow$ 
  - Can't service all of  $r_1$  or  $r_2$
  - So hold them to the remaining fair share:  $f = 4$



$f = 4$ :  
 $\min(8, 4) = 4$   
 $\min(6, 4) = 4$   
 $\min(2, 4) = 2$

47

- FQ and FIFO? ? ?

## 网络服务质量

### 不同类型应用对QoS要求

Application	Data Loss (Reliability)	Throughput (Bandwidth)	Time Sensitive
File transfer	no loss	elastic	no
Email	no loss	elastic	no
Web documents	no loss	elastic	no
Real-time audio/video	loss-tolerant	audio: 5k~1Mbps video: 10k~5Mbps	100's msec
Stored audio/video	loss-tolerant	same as above	few secs
Interactive games	loss-tolerant	few kpbs up	100's msec
Instant messaging	no loss	elastic	nearly

- **弹性流量**：适应性强，对带宽、延迟、抖动和丢包的要求较低，能够根据网络条件调整传输速率。【如文件传输、email、一般的web浏览】
- **非弹性流量**：适应性差，对带宽、延迟、抖动和丢包有严格要求，无法根据网络条件灵活调整。【如VoIP、视频会议等】

## 综合服务体系ISA

- RSVP协议：在给定的QoS级别中为新的流预留资源

在 IP 网络中为单个应用会话提供 QOS 保证

需要专用的RSVP路由器

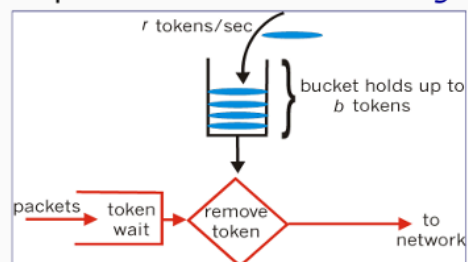
## 区分服务DS

- SLA：服务级别协议
  - 在使用DS之前在ISP和客户之间建立
  - 根据DS字段定义每跳行为

## 流量调度算法

- 漏桶
  - 通过平均数据速率将突发流量调整为固定速率流量
  - 如果存储桶已满，可能会丢弃数据包
  - 数据包**输出速率**是固定的
- 令牌桶
  - 使用令牌控制输出流量，允许改变输出速率
  - 令牌**生成速率**是固定的，当桶满时可能会丢弃令牌（不是数据包）
  - $rt+b$ : ( $b$ 是令牌容量,  $r$ 是令牌生成速率,  $t$ 是时间)

Limit input to specified *burst size* and *average rate*



- Bucket can hold  $b$  tokens
- Tokens generated at rate  $r$  token/sec unless bucket full
- *Over interval of length  $t$ : number of packets admitted less than or equal to  $(rt + b)$*