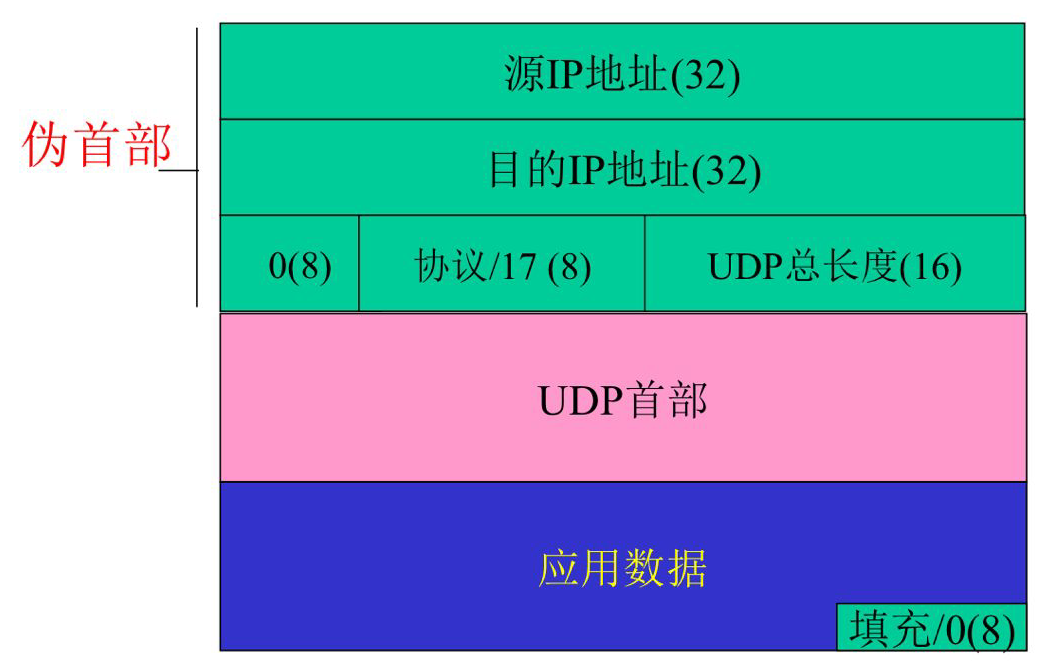
# 4.计算机网络-传输层

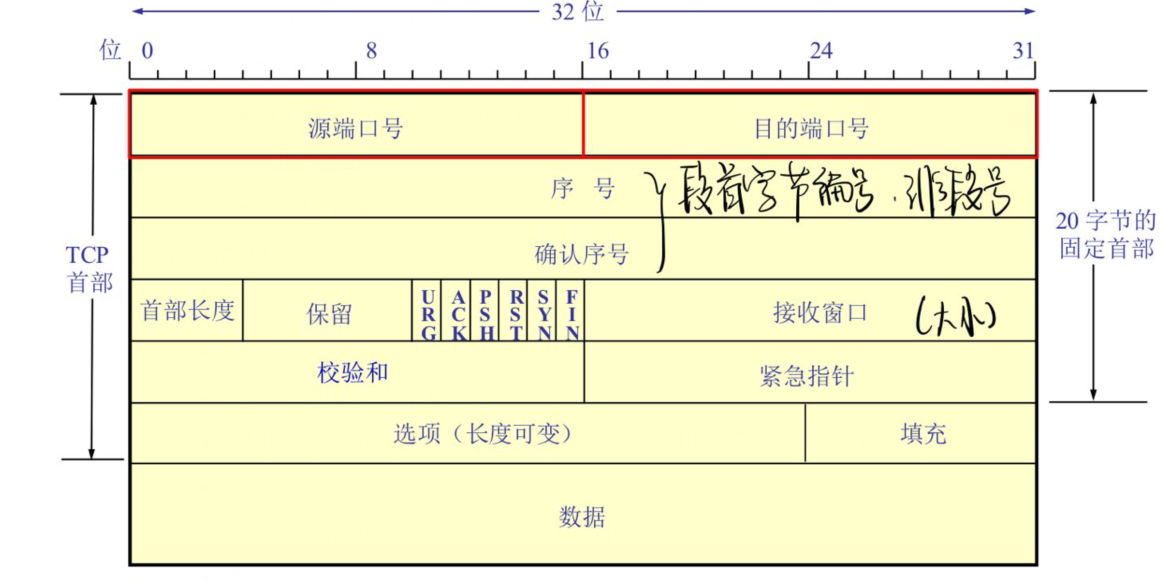
1. 传输层服务
   1. 传输层服务和协议
      1. 传输层协议为运行在不同Host上的**应用进程**提供了一种逻辑通信机制，提供端到端传输服务
      2. 端系统运行传输层协议
         1. 发送方：将应用递交的消息（报文）分成一个或多个的Segment，并向下交给网络层
         2. 接收方：将接收到的Segment组装成消息并向上提交给应用层
      3. 传输层可以为应用提供多种协议：UDP、TCP
   2. Internet传输层协议：
      1. 可靠、按序的交付服务（TCP）：流量控制、拥塞控制、连接建立；不丢失，不出错
      2. 不可靠的交付服务（UDP）：基于“尽力而为”的网络层，没有可靠性扩展
      3. 两种服务均无法保证延迟和带宽
   3. 端口号：应用进程在传输层的通信标识
      1. 服务器端口号分为熟知端口号（0~1023）和登记端口号（1024~49151）
      2. 客户机端口号（短暂端口号）：49152-65535
      3. 常见的应用的熟知端口号：
         1. FTP：21（控制连接）；20（数据连接的PORT模式）
         2. TELNET：23
         3. SMTP：25
         4. DNS：53
         5. TFTP：69
         6. HTTP：80
         7. POP3：110
         8. SNMP：161
2. 传输层多路复用/分用
   1. 多路复用/分用：某层的一个协议/实体直接为上层的多个协议/实体提供服务
   2. 发送端多路复用：从多个Socket接收数据，为每块数据封装上头部信息，生成Segment并交给网络层
   3. 接收端多路分用：传输层依据头部信息将收到的Segment交给正确的Socket，即不同的进程
   4. 传输层分用工作过程
      1. 主机接到IP数据报
         1. 每个IP数据报携带源IP地址、目的IP地址
         2. 每个IP数据报携带一个传输层的段，每个段携带源端口号和目的端口号
      2. 主机收到数据段之后，传输层协议提取IP地址和端口号信息，将数据段导向相应的Socket
         1. TCP会做更多处理
   5. 传输层无连接分用
      1. 利用端口号创建Socket，UDP的Socket用二元组（目的IP地址，目的端口号）标识
      2. 主机收到UDP段后，检查段中的目的端口号，将UDP段导向绑定在该端口号的Socket
      3. 来自不同源IP地址和/或源端口号的IP数据包，只要目的IP地址和目的端口号一样，就会被导向同一个Socket
   6. 传输层面向连接的分用
      1. TCP的Socket用四元组标识：（源IP地址，源端口号，目的IP地址，目的端口号）
      2. 接收端利用所有的四个值将Segment导向合适的Socket
      3. 同一个目的端口号可能指向同一个进程的不同Socket，故需要四元组标识不同的连接，考虑一个进程可以创建多线程
      4. TCP的进程通信是一对一连接，一个客户机进程对应一个服务机进程
      5. 服务器可能同时支持多个TCP Socket，每个TCP Socket用自己的四元组标识
      6. Web服务器为每个客户端开不同的Socket
3. UDP协议
   1. UDP：User Datagram Protocol
      1. 基于Internet IP协议，支持复用/分用，有简单的错误校验，无错误恢复
      2. Best Effort服务，UDP段可能丢失，也可能非按序到达
      3. 无连接，UDP发送方和接收方之间不需要握手，每个UDP段的处理独立于其他段
      4. 延迟少；实现简单；头部开销少；没有拥塞控制，应用更好控制发送时间和速率



* 1. 常用于流媒体应用（长/短视频）：容忍一定丢失，对速率敏感；还用于DNS等
  2. 在UDP上实现可靠数据传输：
     1. 在应用层增加可靠性机制
     2. 应用特定的错误恢复机制（如停等协议，滑动窗口协议等）
  3. UDP校验和：检测UDP段在传输中是否发生错误（如位翻转）
     1. 发送方：
        1. 将参与计算的所有内容视为16bit整数序列
        2. 计算（二进制）整数序列的和，最高位进位也要加在和的后面（回卷），将和求反码得到校验和
        3. 将校验和放入相应字段
     2. 接收方：
        1. 针对收到的UDP报文段（包括检验和字段），按照相同的方法计算校验和，得到的校验和为全1则无错，否则有错
     3. 三部分：
        1. 伪首部：包括源、目的IP地址、协议、总长度等
        2. UDP首部
        3. 应用数据



1. TCP协议
   1. TCP概述
      1. 点对点：一个接收方，一个发送方
      2. 传输可靠的、按序的字节流
      3. 提供流水线机制：TCP拥塞控制和流量控制机制设置窗口尺寸
      4. 发送方和接收方均有缓存
      5. 全双工通信机制：同一连接可以同时传输双向数据流
      6. 面向连接：
         1. 通信双方在发送数据前必须建立连接
         2. 连接状态只在连接的两端维护，沿途结点不维护状态
         3. TCP连接包括：两台主机上的缓存、连接状态变量、socket等
      7. 提供流量控制机制、拥塞控制机制
   2. TCP段结构



* + 1. 源/目的端口号：各16位，用于多路复用/分用
    2. 序号字段和确认序号字段：各32位，
    3. 序号字段和确认序号字段：各32位，对每个应用层数据的每个字节进行编号
       1. 确认序号是期望从对方接收数据的字节序号
       2. 累计确认机制，确认序号前的所有数据均被正确接收
    4. 首部长度：4位，以4字节为单位（与IP格式相同）
       1. **固定首部占20B**
       2. 还包含选项和填充
    5. 保留字段：6位，没用
    6. 标志位：
       1. URG=1：紧急指针字段有效（通常不使用）
       2. ACK=1：确认序号字段有效
       3. PSH=1：希望尽快将段中数据交付给应用进程（通常不使用）
       4. RST=1：重新建立TCP连接
       5. SYN=1：该TCP段是一个建立新连接的请求控制段
       6. FIN=1：请求释放TCP连接
    7. 接收窗口：16位，用于流量控制
    8. 校验和：16位，包括TCP伪首部、TCP首部和应用层数据
    9. 紧急指针字段：16位，URG=1时才有效，指出紧急数据最后一个字节在数据中的位置
    10. 选项：长度可变，最大段长度MSS（不包括TCP首部，只包含数据部分），可用于时间戳
    11. 填充：长度0-3字节，取值全0
  1. TCP：序列号和ACK
     1. 序列号：段中第一个字节的编号，而不是段的连续编号；建立TCP连接时，双方随机选择序列号
     2. ACK：期望收到的下一个字节的序列号，前面的字节都已经正确接收，应用累计确认机制
     3. 乱序到达的段如何处理？TCP规范中并未规定，由TCP实现者自己决策
  2. TCP可靠数据传输：TCP在IP层提供的不可靠服务基础上实现可靠数据传输服务
     1. 流水线机制、累计确认机制、使用单一重传定时器
     2. 触发重传的事件：超时/收到重复ACK
     3. 如何设置定时器超时时间？
        1. 大于RTT？但RTT是动态数值
        2. 估计RTT，测量多个SampleRTT，取平均值形成RTT的估计值EstimatedRTT，然后计算加权移动
        3. 加权移动：EstimatedRTT=(1-α)EstimatedRTT+αSampleRTT
           1. α的典型值：0.125
        4. 得到的EstimatedRTT是一个安全边界
        5. 测量RTT的变化值：SampleRTT与EstimatedRTT的差值DevRTT
           1. DevRTT= (1-β) DevRTT + β |SampleRTT - EstimatedRTT |
           2. 通常β=0.25
        6. 定时器超时时间设置为：TimeOutInterval = EstimatedRTT + 4DevRTT
     4. TCP发送方事件：
        1. 若超时，重传引起超时的最古老的段，重启定时器
        2. 收到ACK
           1. 若是确认之前未确认的段，更新sendbase；若窗口中还有未被确认的分组，重启定时器
     5. TCP ACK生成：TCP接收方的动作
        1. 若期望的报文段按序到达，且期望序号前的数据已经被确认
           1. 延迟的ACK：对另一个按序报文段的到达最多等待500ms，若下一个报文段在这个时间未到达，则发送一个ACK
        2. 若期望的报文段按序到达，另一个按序报文段等待发送ACK
           1. 立即发送单个累计ACK确认两个按序报文段
        3. 若比期望序号大的报文段乱序到达
           1. 发送重复ACK，指明下一个期待字节的序号
        4. 能部分或者完全填充数据间隔的报文段到达
           1. 若该报文段起始于间隔低端，立即发送ACK
     6. 快速重传机制
        1. TCP实现中，超时则会重新设置超时时间间隔使其加倍，等待时间会变得很长
        2. 可以通过检测重复ACK实现快速重传，即在定时器超时之前重传
        3. 若发送方收到对同一序号的3个重复ACK，我们认为该数据之后的段丢失，直接重传
  3. TCP流量控制：通过告知接收窗口尺寸实现的速度匹配机制
     1. 接收方为TCP连接分配缓存，通过TCP的头部字段（接收窗口字段）将RcvWindow告诉发送方
     2. 发送方限制自己已经发送但还未收到ACK的数据不超过接收方分RcvWindow尺寸
     3. 如果接收方告知RcvWindow=0，发送方仍然会发送一个很小的段——需要继续获取响应
  4. TCP连接管理
     1. 面向连接，在传输数据前需要建立连接，
     2. **三次握手：建立TCP连接**
        1. step1：client向server发送TCP SYN控制segment
           1. 初始化client序列号，不携带数据
           2. SYN=1，seq=x
        2. step2：server收到SYN，向client恢复TCP SYN、ACK控制segment
           1. server分配缓存，初始化server序列号，不携带数据
           2. SYN=ACK=1，seq=y，ack\_seq=x+1
        3. step3：client收到SYN、ACK ，向server回复ACK，可能携带数据
           1. ACK=1，seq=x+1，ack\_seq=y+1
     3. **四次挥手：关闭TCP连接**
        1. step1：client向server发送TCP FIN 控制segment
        2. step2：server收到FIN，回复ACK，关闭连接，发送FIN
        3. step3：client收到FIN，回复ACK
           1. 进入等待状态，若收到FIN，会重新回复ACK（确保关闭）
        4. step4：server收到ACK，连接关闭完成
  5. TCP拥塞控制：发送方限制发送速率
     1. 拥塞窗口CongWin：动态调整以改变发送速率，反应感知到的网络拥塞
     2. 如何感知网络拥塞：丢失事件=timeout或3个重复ACK
     3. 发生丢失事件后调整发送速率：AIMD、SS
        1. 加性增-乘性减：AIMD
           1. 逐渐增加发送速率，谨慎探测可用带宽，直到发生丢失事件
           2. 加性增：每个RTT将CongWin增大一个MSS——拥塞避免阶段
           3. 乘性减：发生丢失事件后将CongWin减半
           4. 问题：未发生拥塞时增长太慢，效率低
        2. 慢启动：SS
           1. TCP连接建立时CongWin=1（个MSS）
           2. 可用带宽可能远高于初始速率，我们希望能够快速增长
           3. 连接开始时，每个RTT内发送的段数呈指数增长，每个RTT将CongWin翻倍——慢启动阶段
           4. 何时将指数性增长切换为线性增长（拥塞避免）？CongWin达到丢失事件前值的1/2时

增设一个阈值变量Threshold

丢失事件发生时，Threshold被设为丢失事件前CongWin值的1/2，CongWin指数增长达到Threshold时进入拥塞避免阶段，变为线性增长

* + - * 1. 遇到3个重复ACK（丢包，但没有堵塞，还能够传输一些段）

阈值Threshold设置为CongWin的一半，CongWin设为Threshold，然后线性增长

* + - * 1. TimeOut事件（网络发生严重拥塞）

CongWin直接设置为1个MSS，重新开始指数增长，达到Threshold后再线性增长

* 1. TCP吞吐率：
     1. 给定拥塞窗口大小和RTT，TCP的平均吞吐率（忽略SS）
        1. 假定发生超时时CongWin的大小（以TCP段数计）是W，吞吐率为W/RTT
        2. 超时后，CongWin变为W/2，吞吐率为W/2RTT
        3. 平均吞吐率（以TCP段数计）为0.75W/RTT
     2. 吞吐率与丢包率：想要获得高吞吐率，要求丢包率特别低，对硬件要求太苛刻了
  2. TCP拥塞控制的改进
     1. 快速恢复：出现3个重复ACK的时候可以，可以将CongWin加3——网络起码还有这些能力
     2. TCP CUBIC、Delay-based TCP congestion control、ECN（略）
  3. TCP的公平性：K个TCP session 共享相同的瓶颈带宽R，每个session的平均速率为R/K
     1. TCP具有公平性，多个TCP会话的平均速率相同
     2. 多媒体应用通常不使用TCP以免被拥塞控制机制限制速度，UDP可以容忍丢失，这种情境下使用TCP会带来不公平
     3. 同样，有些应用，如Web浏览器会打开多个并行连接，这也导致了应用间不公平