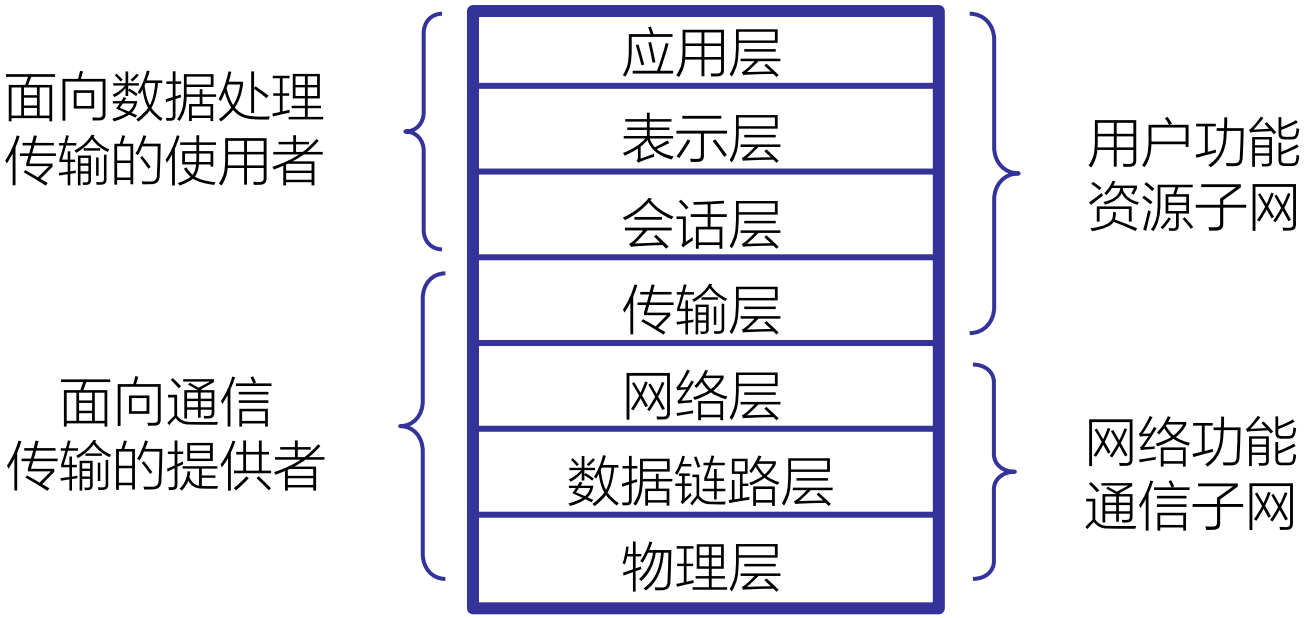
计算机网络

胡亮 等编著

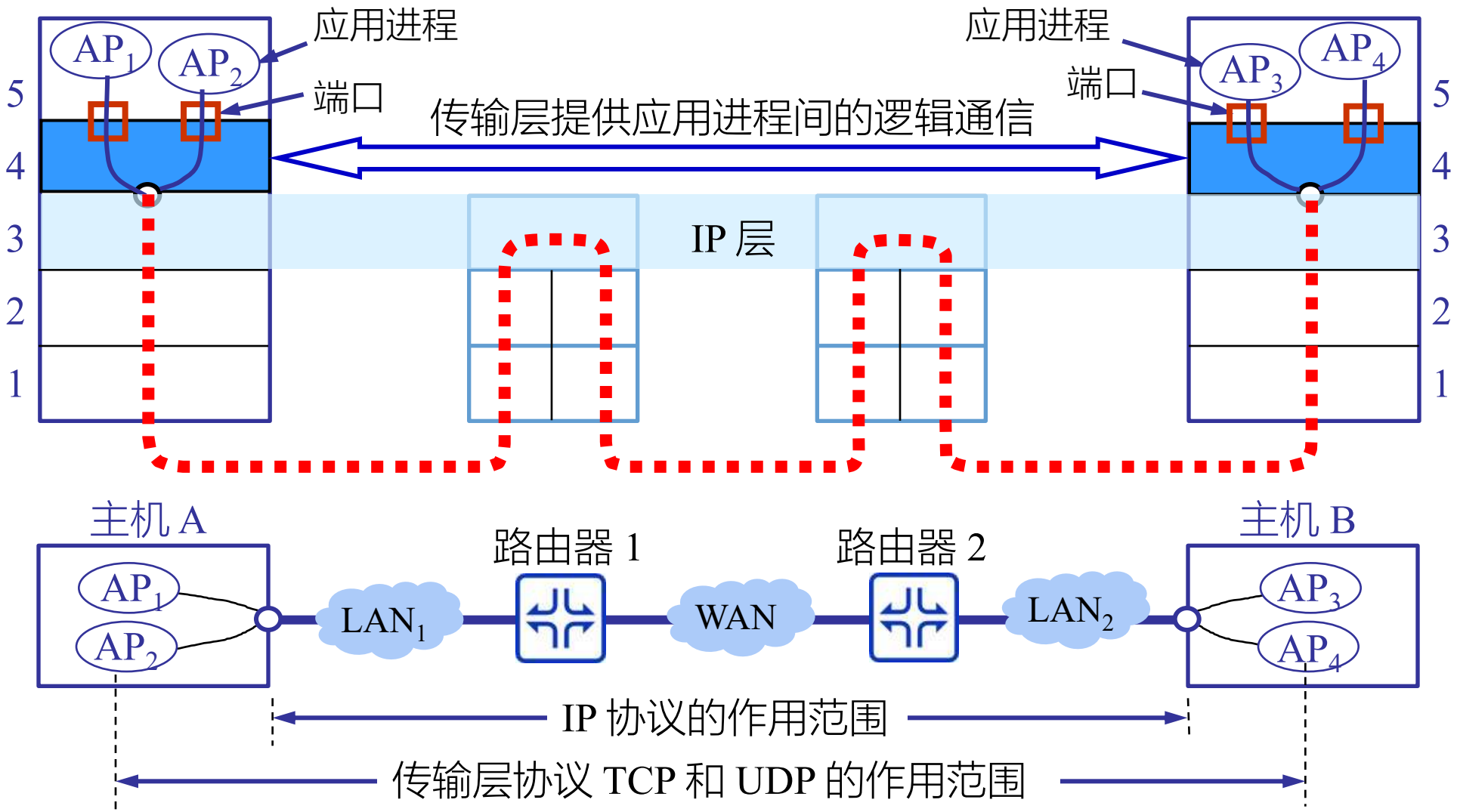
第六章 传输层

6.1传输层基本原理

- 在OSI 参考模型中，传输层位于通信子网和资源子网之间，是整个协议层次中最核心的一层
- 传输层为源主机上的进程和目的主机上的进程之间提供可靠的透明数据传送，使高层用户在相互通信时不必关心通信子网实现的细节

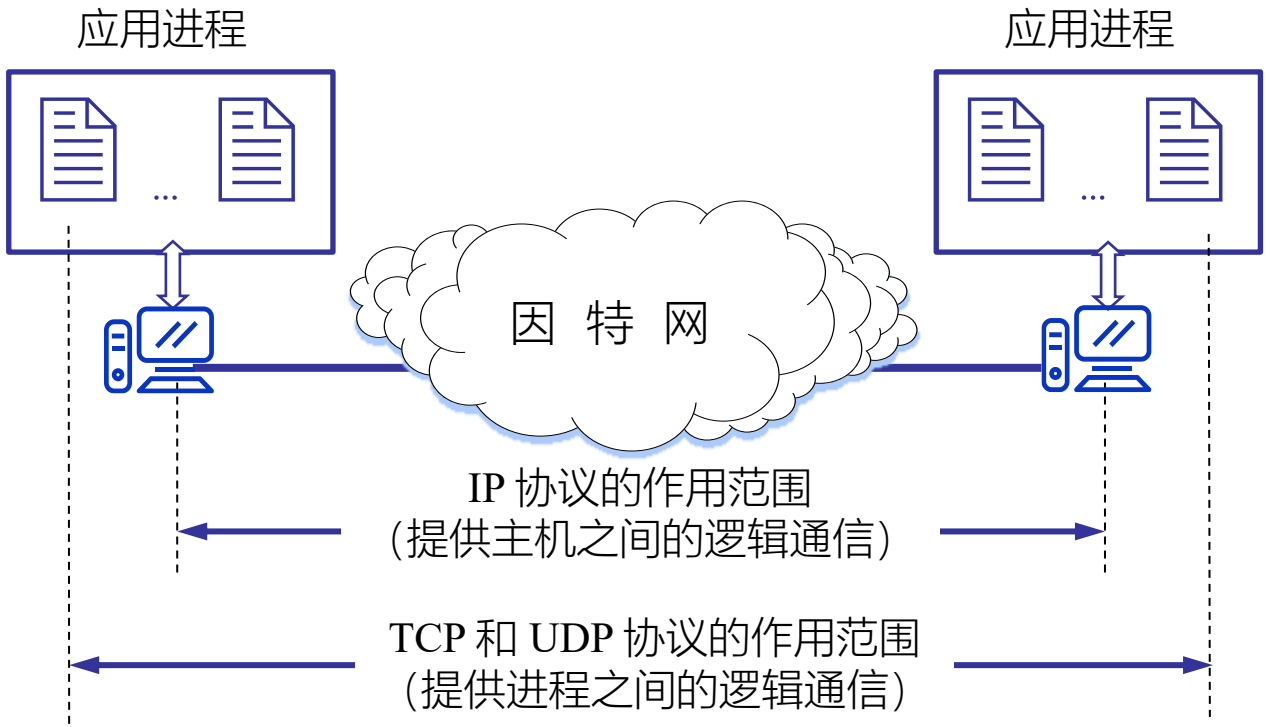


传输层提供逻辑通信


应用进程
端口
应用进程
端口
AP1
AP2
AP3
AP4
5
4
3
2
1
传输层提供应用进程间的逻辑通信
IP 层
主机 A
主机 B
路由器 1
路由器 2
LAN1
WAN
LAN2
IP 协议的作用范围
传输层协议 TCP 和 UDP 的作用范围

传输协议

- 传输服务是通过传输层实体间使用传输协议来实现的
- 通信不仅仅是发生在从源计算机到目的计算机，而且是从端应用程序到端应用程序



传输层和网络层协议的区别

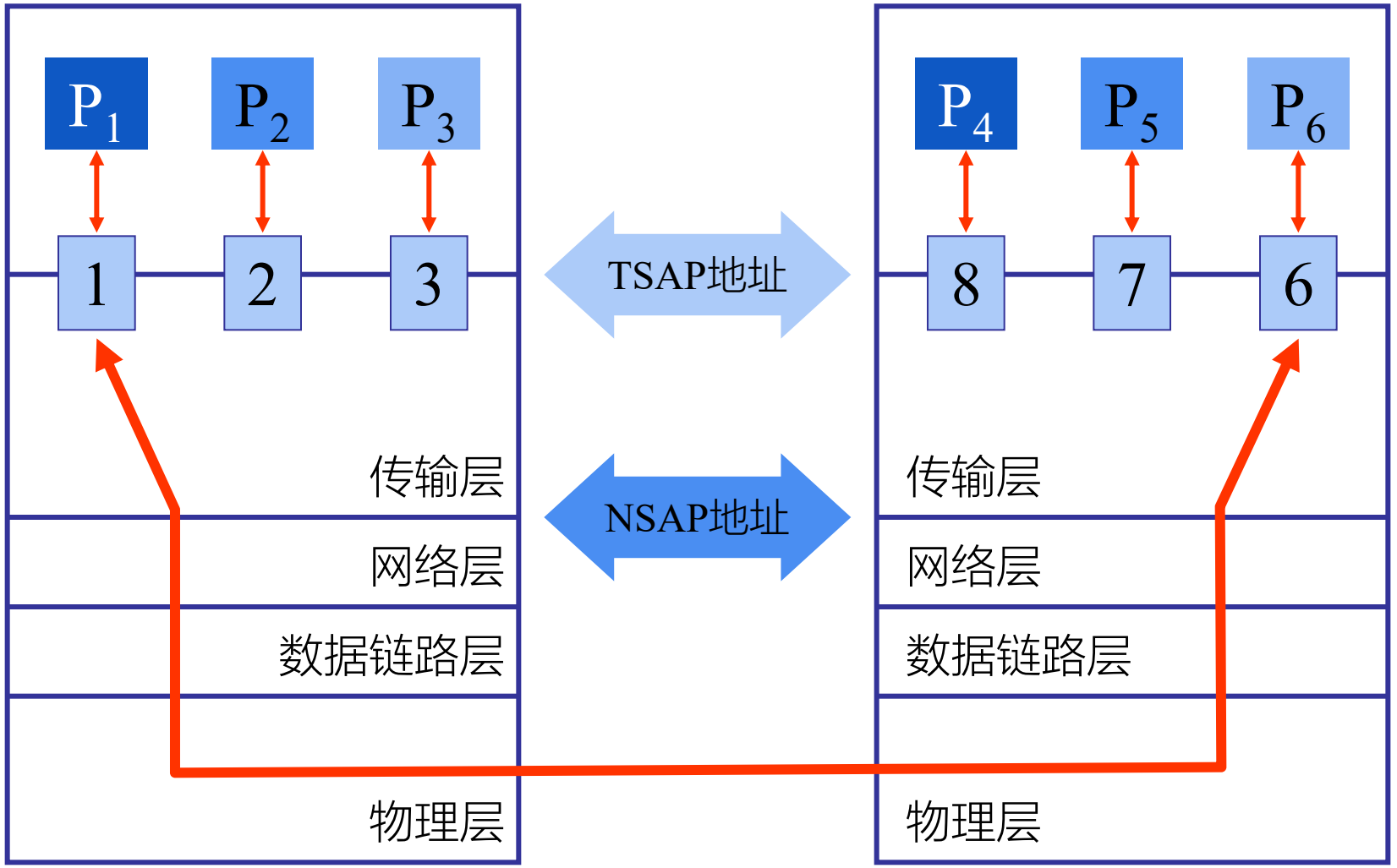
传输层功能

- 具体功能包括：
  - 端到端的报文传递
  - 服务点的寻址
  - 拆分和组装
  - 连接控制
- 传输层为上层提供两种类型服务：
  - 面向连接的传输服务
  - 无连接的传输服务

传输层地址

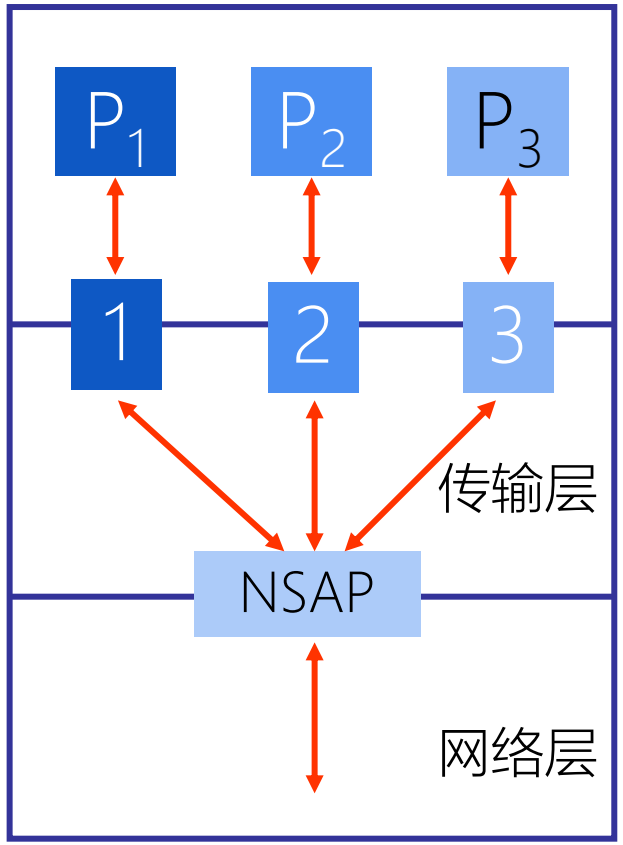
- 由一台计算机上的应用程序所产生的数据不仅必须被另外一台计算机所接收，而且必须被这台计算机上正确的应用程序所接收
- 一个应用程序同一个远程应用程序通信时，它必须知道两个地址：
  - TSAP地址(传输访问服务点，称为端口号)
  - NSAP地址(网络服务访问点)

访问点通信

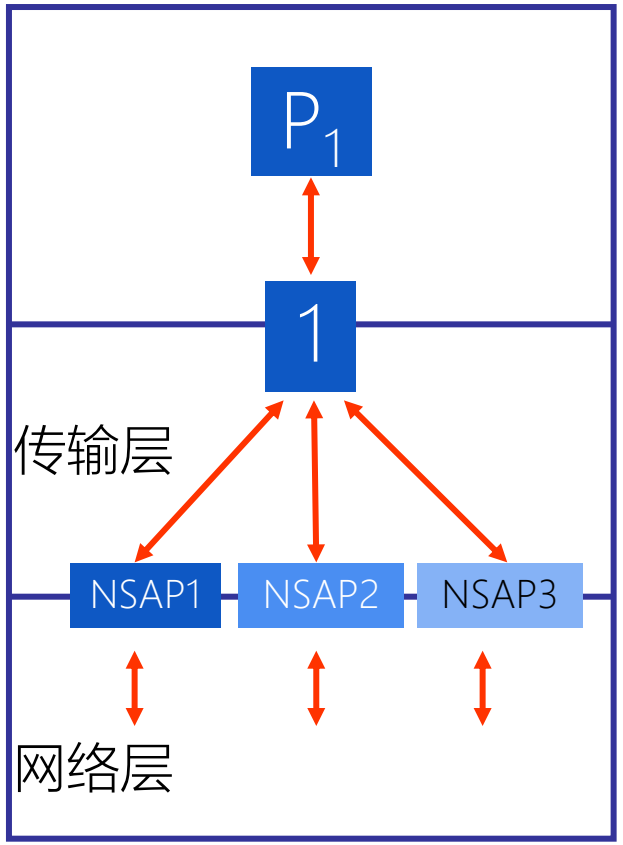


应用程序使用服务访问点通信

传输层复用—向上复用、向下复用



向上复用

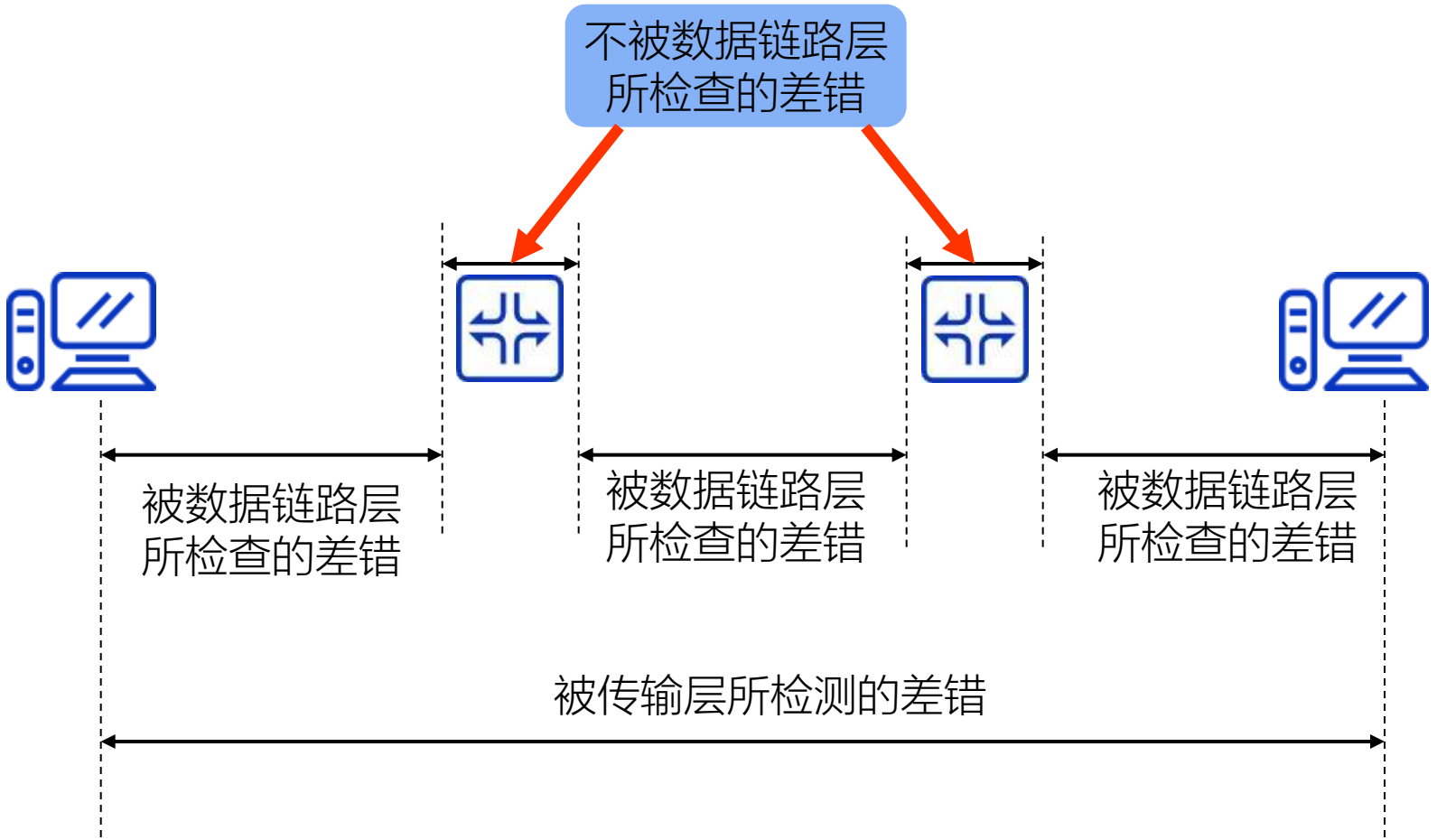


向下复用

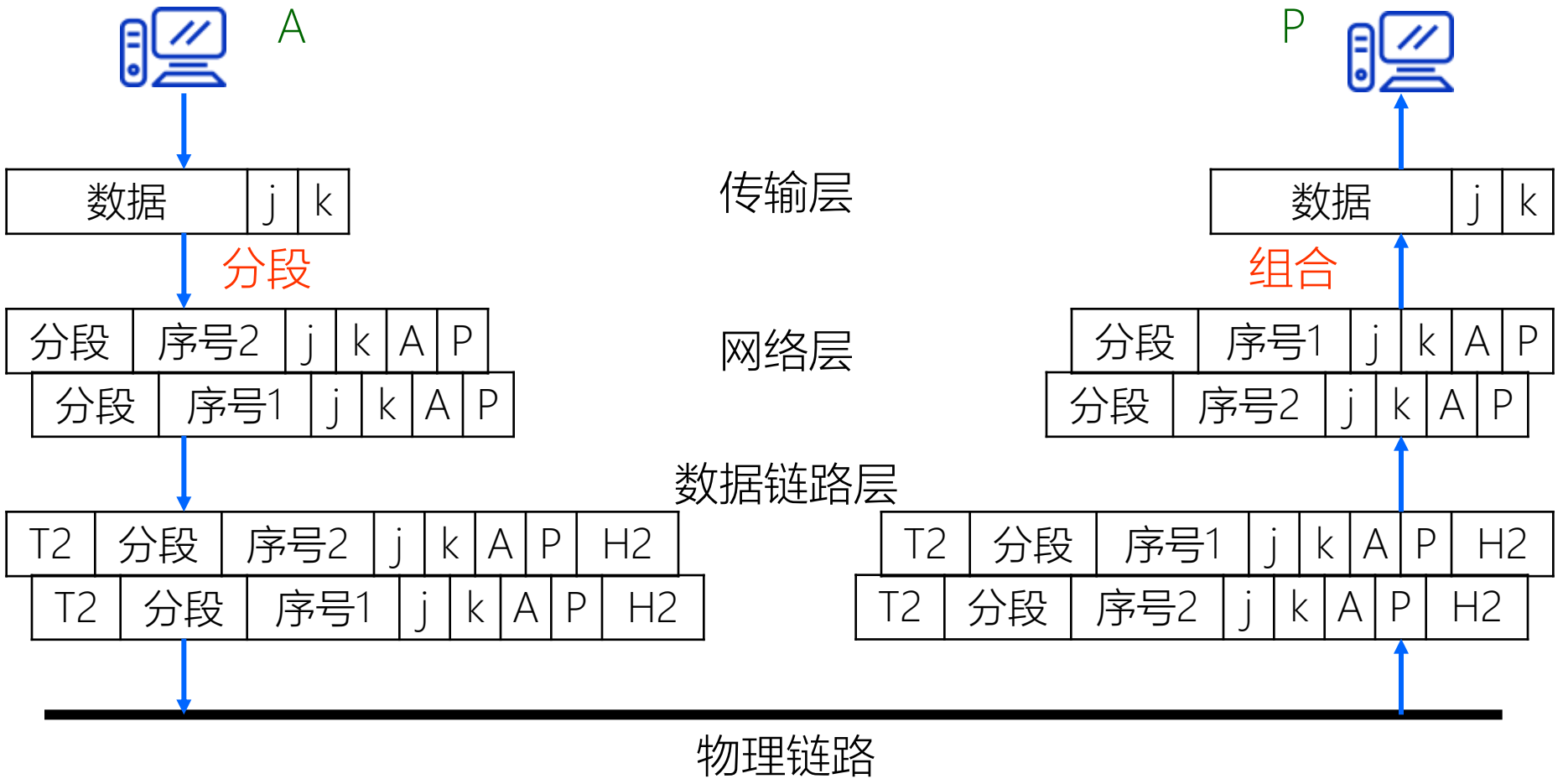
可靠传输

- 可靠传输包括以下4个方面：
  - 差错控制：保证可靠性
  - 次序控制：分段和拼接 、次序编号
  - 丢失控制：丢失重传
  - 重复控制：通过序列编号完成

差错控制

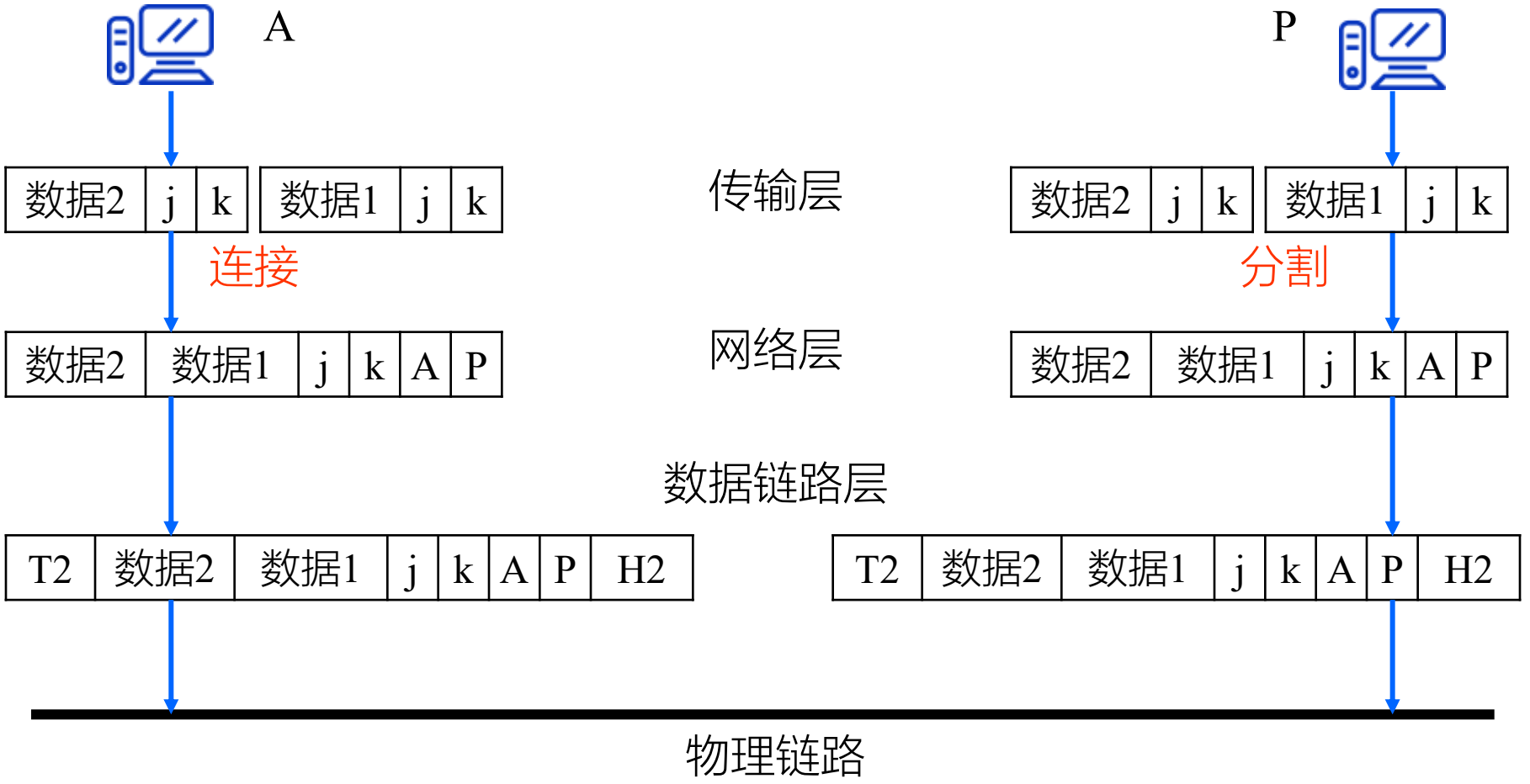

不被数据链路层
所检查的差错
被数据链路层
所检查的差错
被数据链路层
所检查的差错
被数据链路层
所检查的差错
被传输层所检测的差错

次序控制—分段和组合



H2、T2：数据链路层帧首尾， A、P：网络地址， j、k：端口号

次序控制—连接和分割



H2、T2：数据链路层帧首尾， A、P：网络地址， j、k：端口号

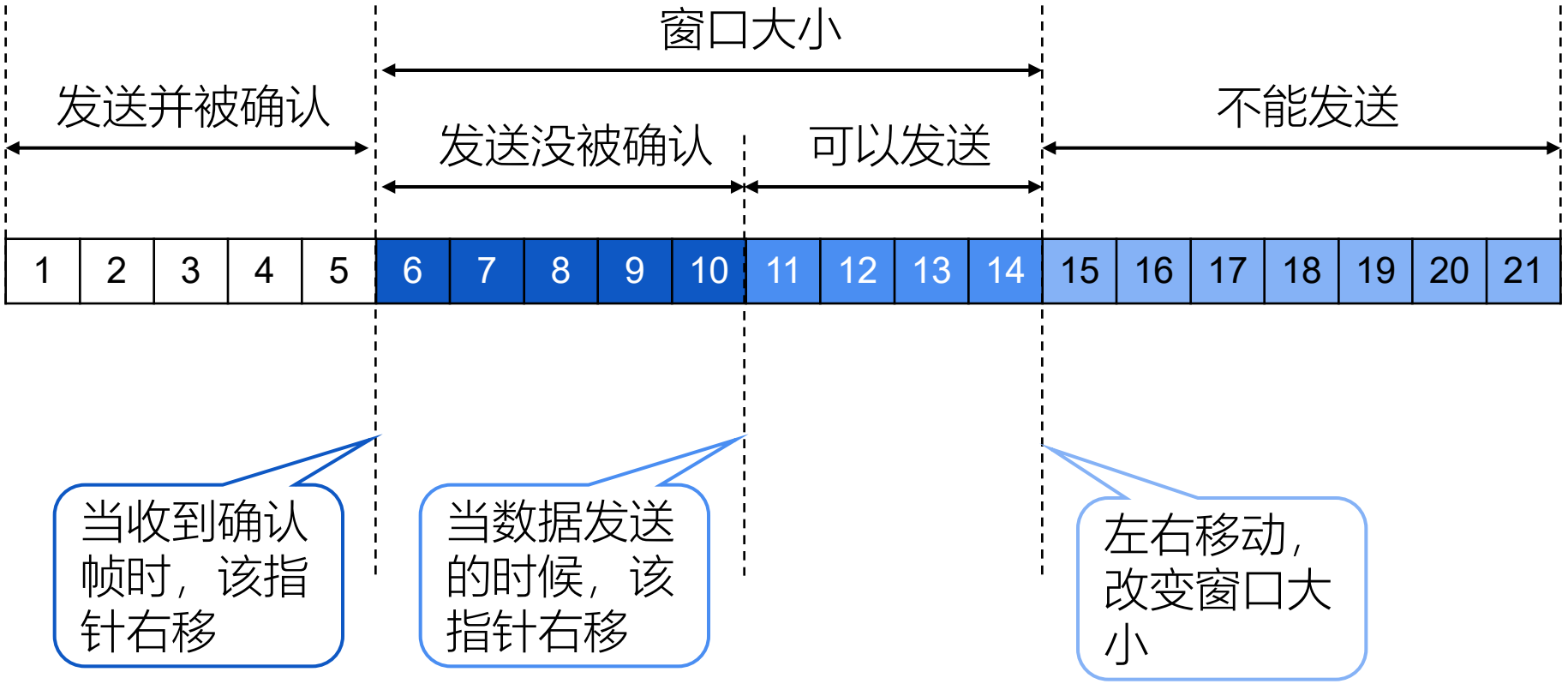
丢失控制与重复控制

- 传输层要确保一次传输的所有片段都到达目的地
- 如果发生了丢失要重传
- 使用序列编号保证接收方丢弃重复数据

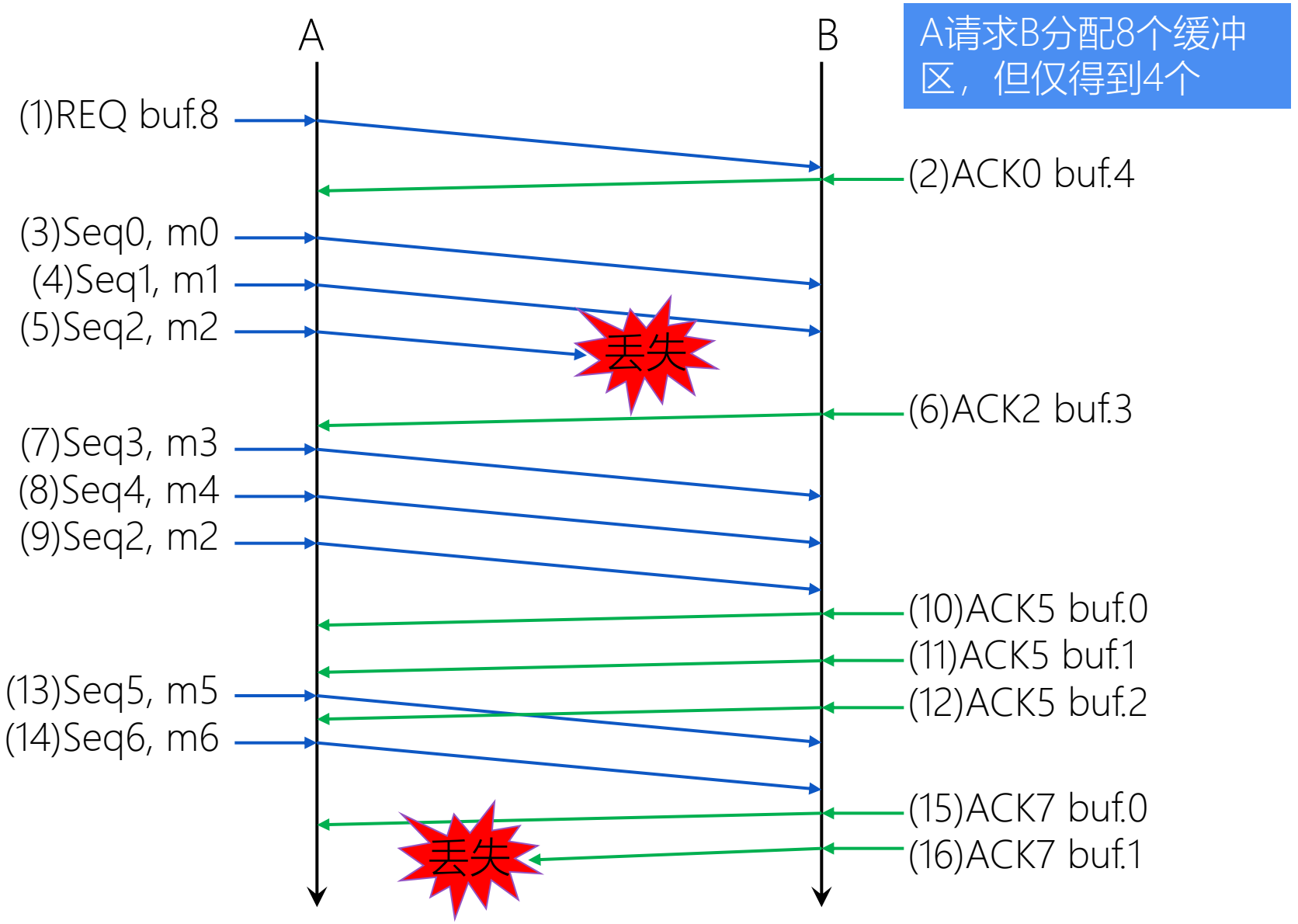
传输层流量控制

- 和数据链路层一样，传输层也负责流量控制。但是，传输层中的流量控制是作用在端到端上的，而不是作用在单条链路上的
- 传输层流量控制也使用滑动窗口协议，但是传输层中的窗口在大小上是可以变化的，以适应可使用的缓冲区的变化情况
- 使用3个指针识别缓冲区

滑动窗口


窗口大小
发送并被确认
发送没被确认
可以发送
不能发送
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
当收到确认帧时，该指针右移
当数据发送的时候，该指针右移
左右移动，改变窗口大小

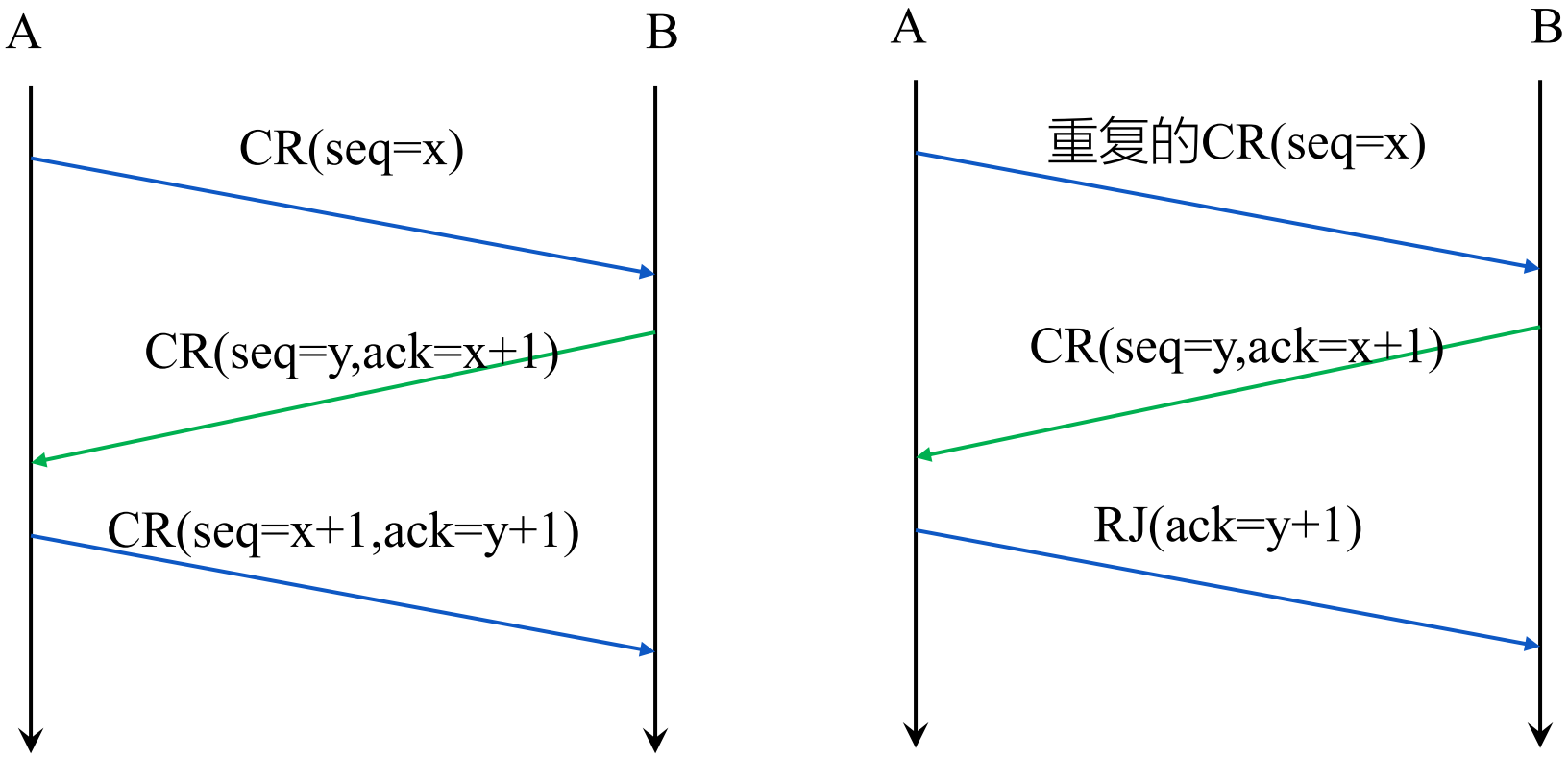
传输层的流量控制


A
B
A请求B分配8个缓冲区，但仅得到4个
(1)REQ buf.8
(2)ACK0 buf.4
(3)Seq0, m0
(4)Seq1, m1
(5)Seq2, m2
丢失
(6)ACK2 buf.3
(7)Seq3, m3
(8)Seq4, m4
(9)Seq2, m2
(10)ACK5 buf.0
(11)ACK5 buf.1
(12)ACK5 buf.2
(13)Seq5, m5
(14)Seq6, m6
(15)ACK7 buf.0
(16)ACK7 buf.1
丢失

传输连接

- 传输层端到端的传送可以采用两种模式来完成：
  - 面向连接
  - 无连接
- 面向连接传输有三个步骤：
  - 连接建立
  - 数据传输
  - 连接终止

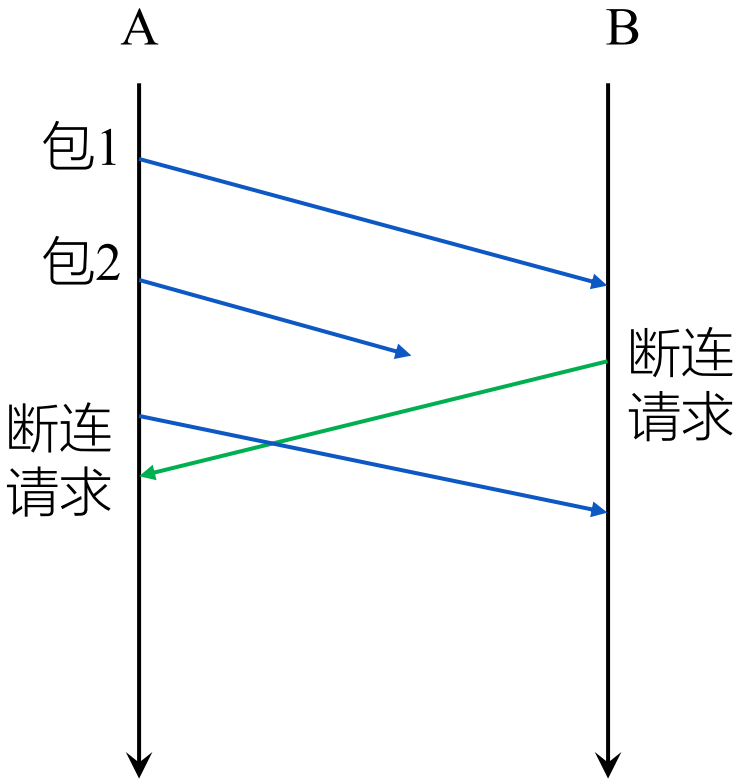
连接建立



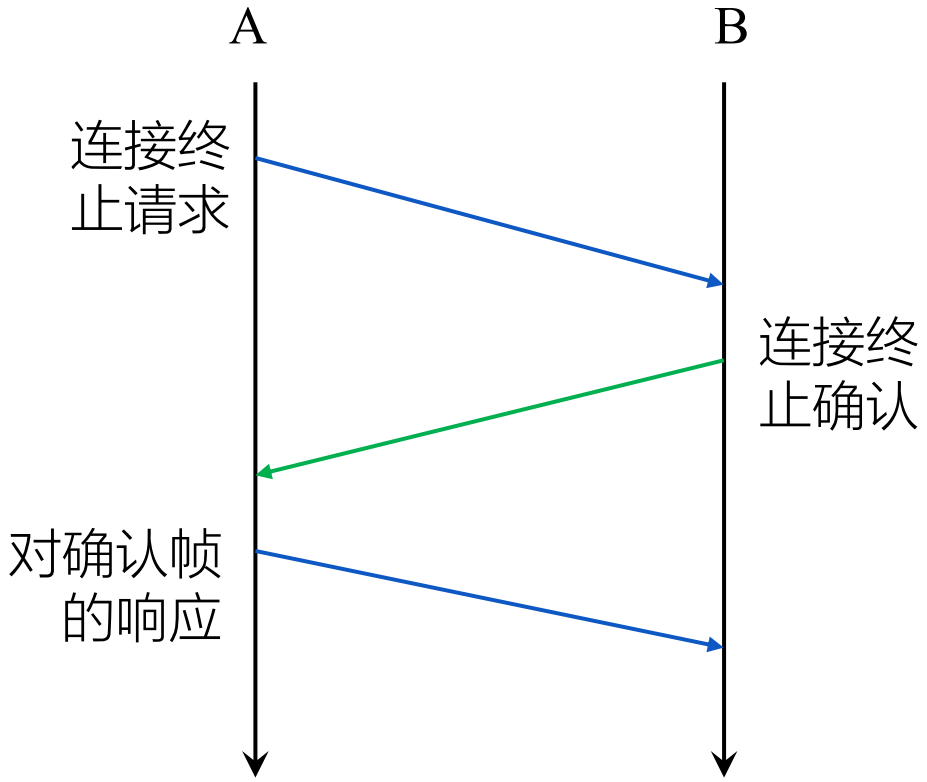
连接的建立　　拒绝重复的连接请求

三次握手方法建立连接

连接终止

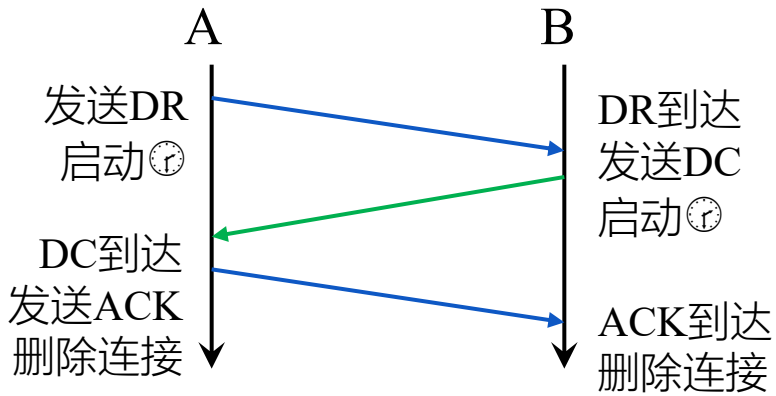


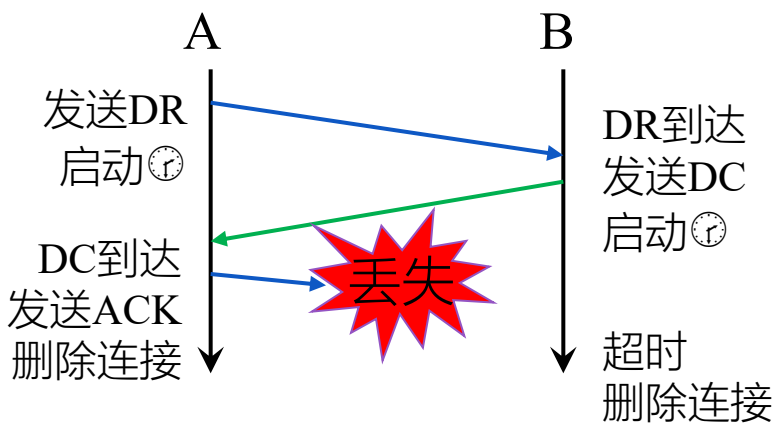
错误断连，数据丢失

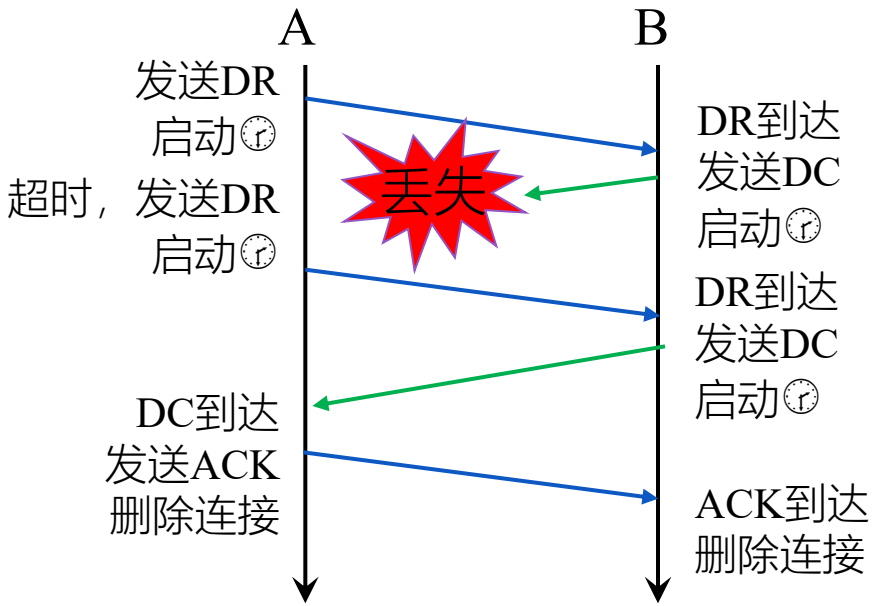


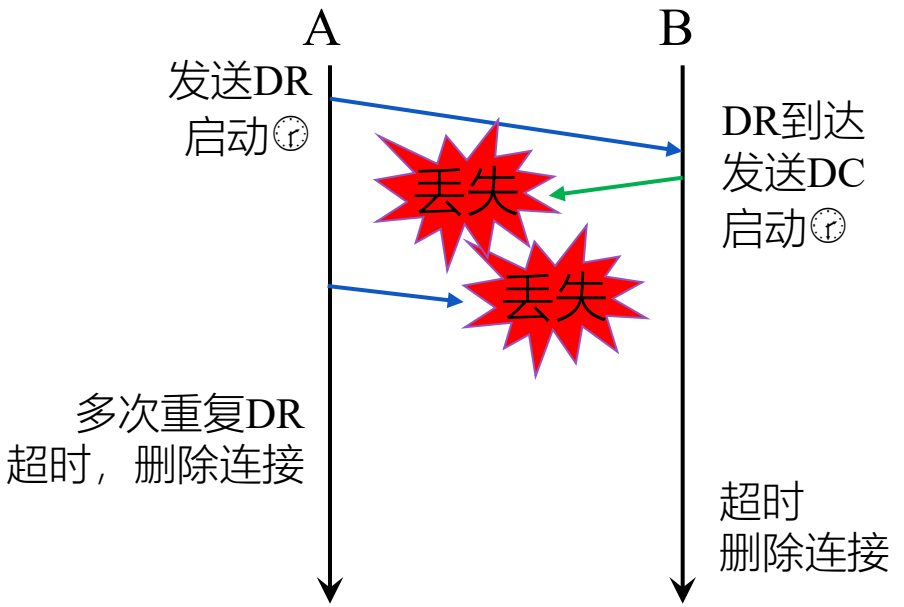
正确的断连方式

三次握手方法的连接释放









DR：断连请求 DC：断连证实 ACK：确认

6.2 用户数据报协议

- UDP采用非连接的方式提供网络应用层的事务处理
- UDP不提供可靠性，即UDP协议不提供端到端的确认和重传功能，它不保证数据包一定能到达目的地，因此称为不可靠协议
  - 不可靠：丢失、重复、延迟、乱序和损坏
- UDP必需在IP上运行。它的下层协议是以IP作为前提的

UDP特征

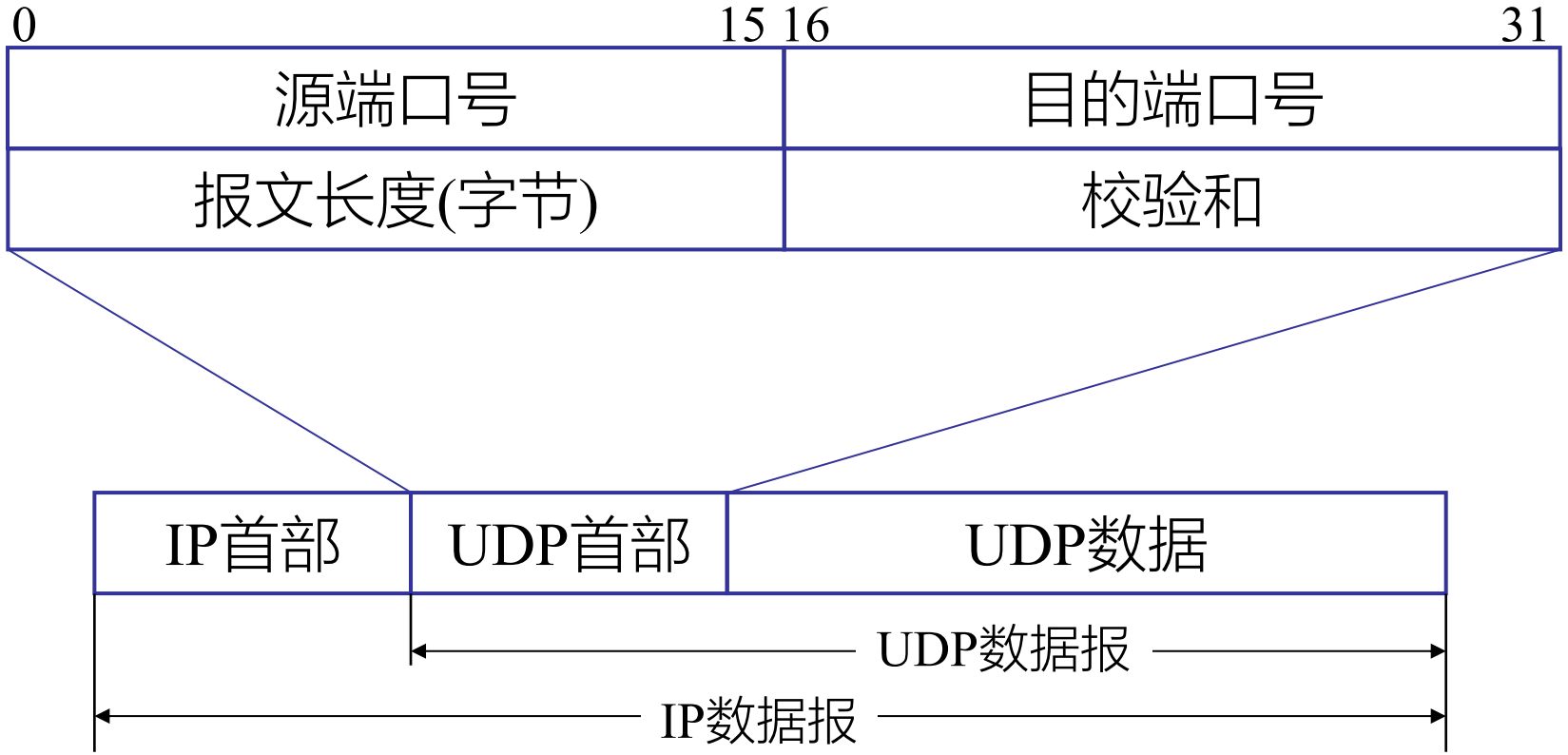
- UDP提供端到端服务，允许应用进程发送和接收单个报文，每个报文被装进单个数据报中进行传输
- UDP可以被表征为：
  - 端到端
  - 无连接
  - 面向报文
  - 尽力而为
  - 任意交互
  - 操作系统无关

采用的通信方式

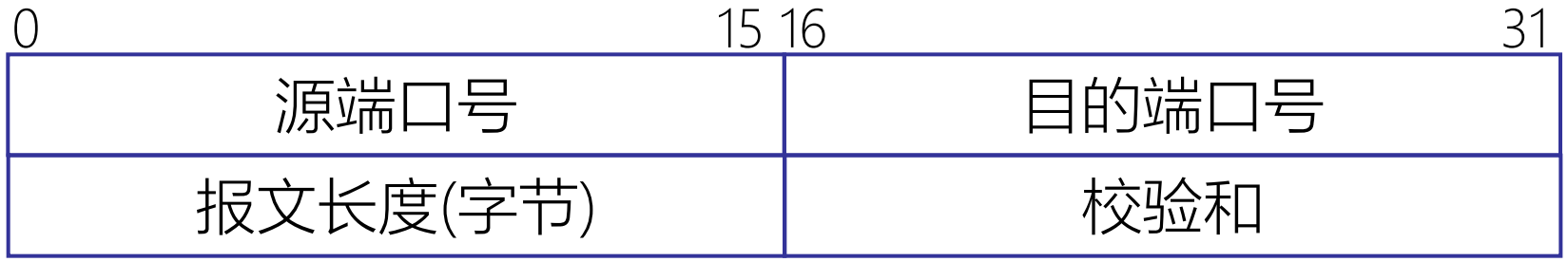
- UDP允许采用2种交互通信方式
  - 一对一
  - 一对多
- 使用协议端口号标识端点
  - 必须跨越异构计算机，UDP定义了一个标识符抽象集—协议端口号(protocol port number)
  - 在每台计算机内提供端口号与操作系统所用的程序标识符之间的映射关系

UDP报头格式

- UDP协议直接利用IP协议进行UDP数据报的传输
- UDP数据报封装在IP数据报中

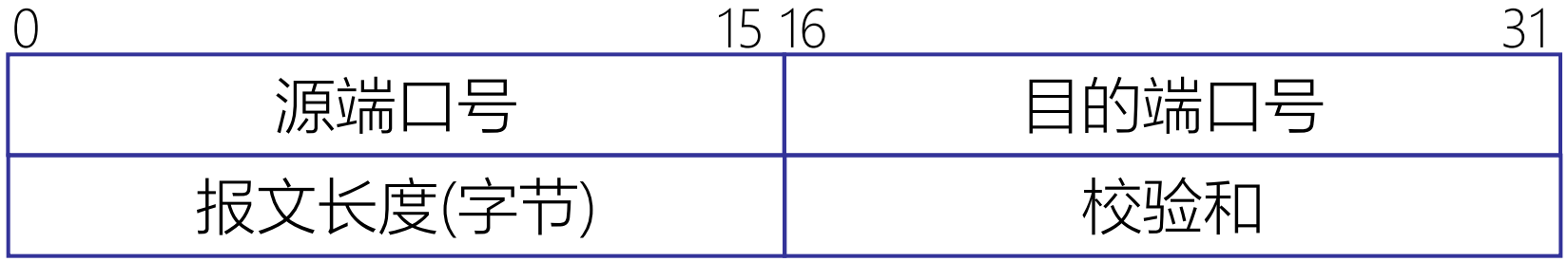


UDP报头格式



- 16位UDP源端口号(Source Port)：该端口号作为接收进程返回数据时的目的端口
  - 可选字段。若不选用，其值为0
- 16位UDP目的端口号(Destination Port)：该端口号是作为接收主机内与特定应用进程相关联的地址

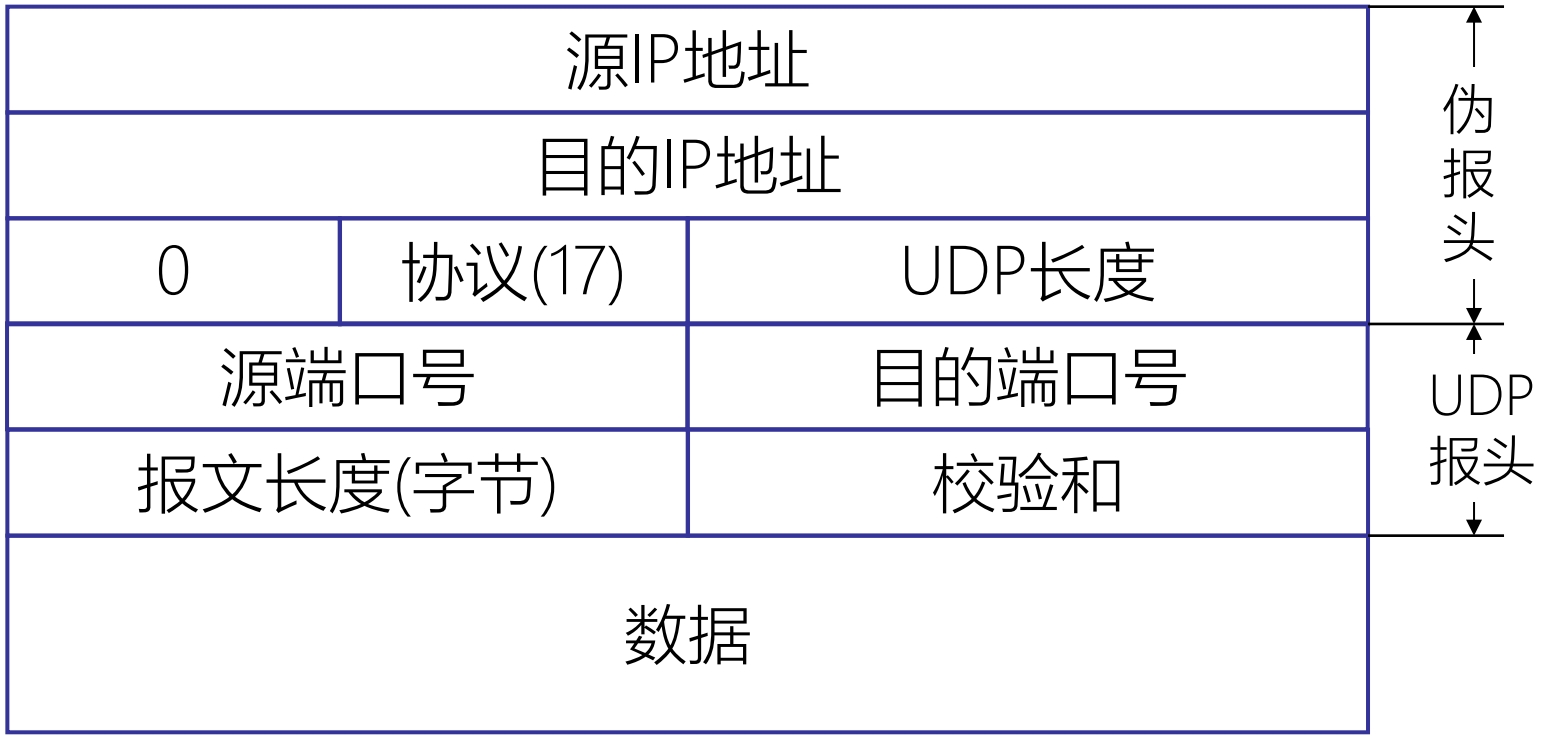
UDP报头格式



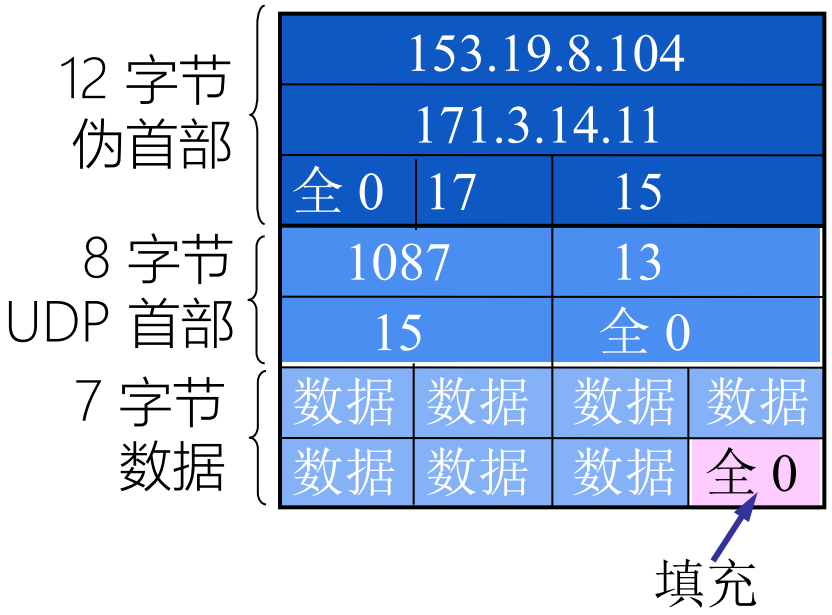
- UDP数据报的长度(Length)：UDP长度字段指的是UDP首部和UDP数据的字节长度。UDP数据报长度是IP全长减去IP首部的长度。最小值是8
- UDP检验和(Checksum)：可选字段，用来检验传输过程中是否出现了错误。UDP检验和覆盖伪首部、 UDP首部和UDP数据

UDP校验和

- UDP在数据报中包含一个12字节的伪报头以计算校验和。该伪报头包含IP报头的某些域，目的是让UDP 检测数据确已到达正确的目的端



计算 UDP 检验和的例子



10011001 00010011 → 153.19
00001000 01101000 → 8.104
10101011 00000011 → 171.3
00001110 00001011 → 14.11
00000000 00010001 → 0 和 17
00000000 00001111 → 15
00000100 00111111 → 1087
00000000 00001101 → 13
00000000 00001111 → 15
00000000 00000000 → 0（检验和）
01010100 01000101 → 数据
01010011 01010100 → 数据
01001001 01001110 → 数据
01000111 00000000 → 数据和 0（填充）

按二进制反码运算求和 10010110 11101101 → 求和得出的结果

将得出的结果求反码 01101001 00010010 → 检验和

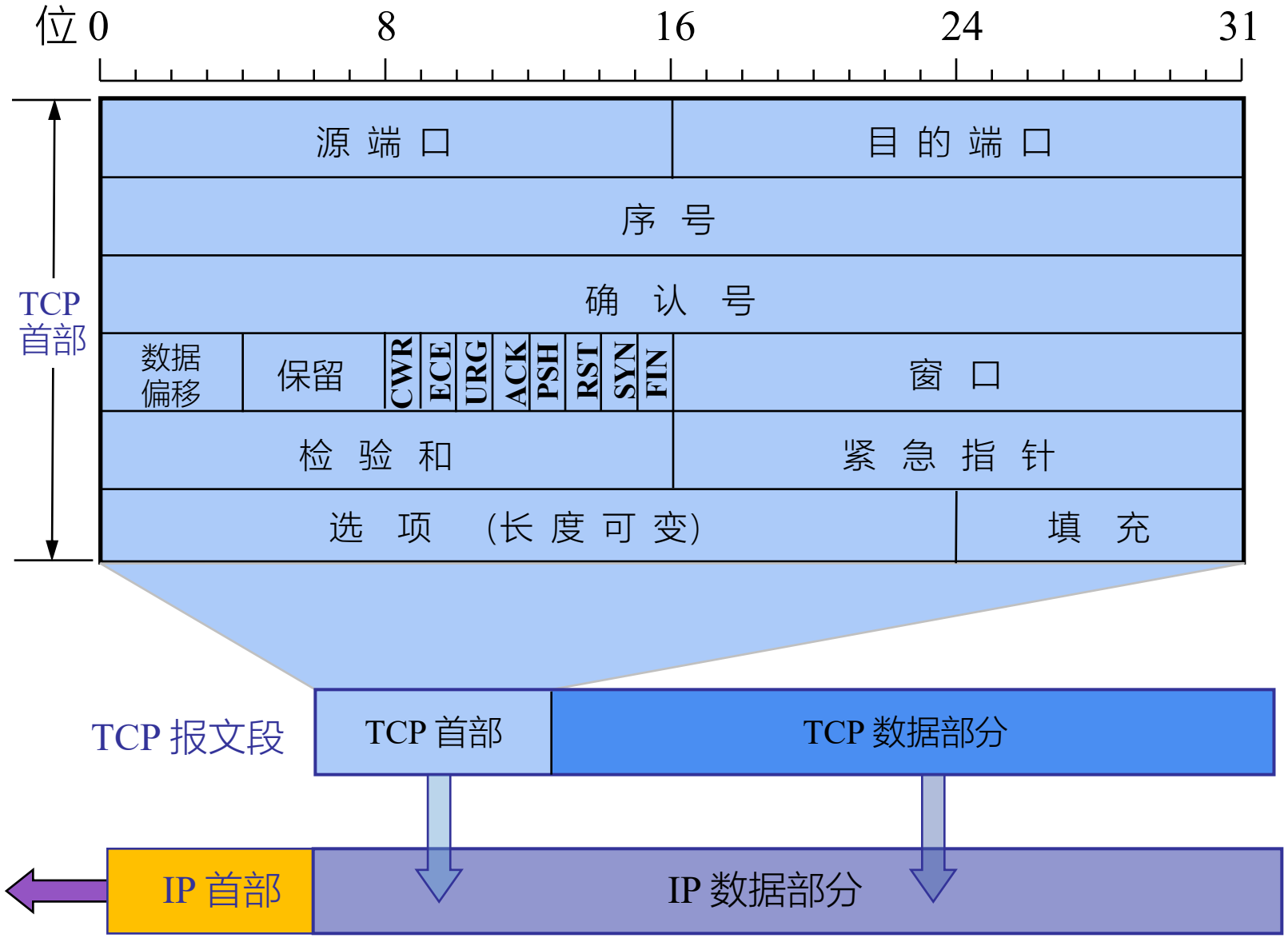
6.3 传输控制协议

- TCP提供了一种可靠的面向连接的字节流传输层服务，TCP提供端到端的流量控制，并计算和验证一个强制性的端到端检查和
- TCP提供的服务有以下主要特点：
  - 面向连接
  - 点对点
  - 完全的可靠性
  - 全双工通信
  - 流接口
  - 可靠的连接建立
  - 友好的连接关闭

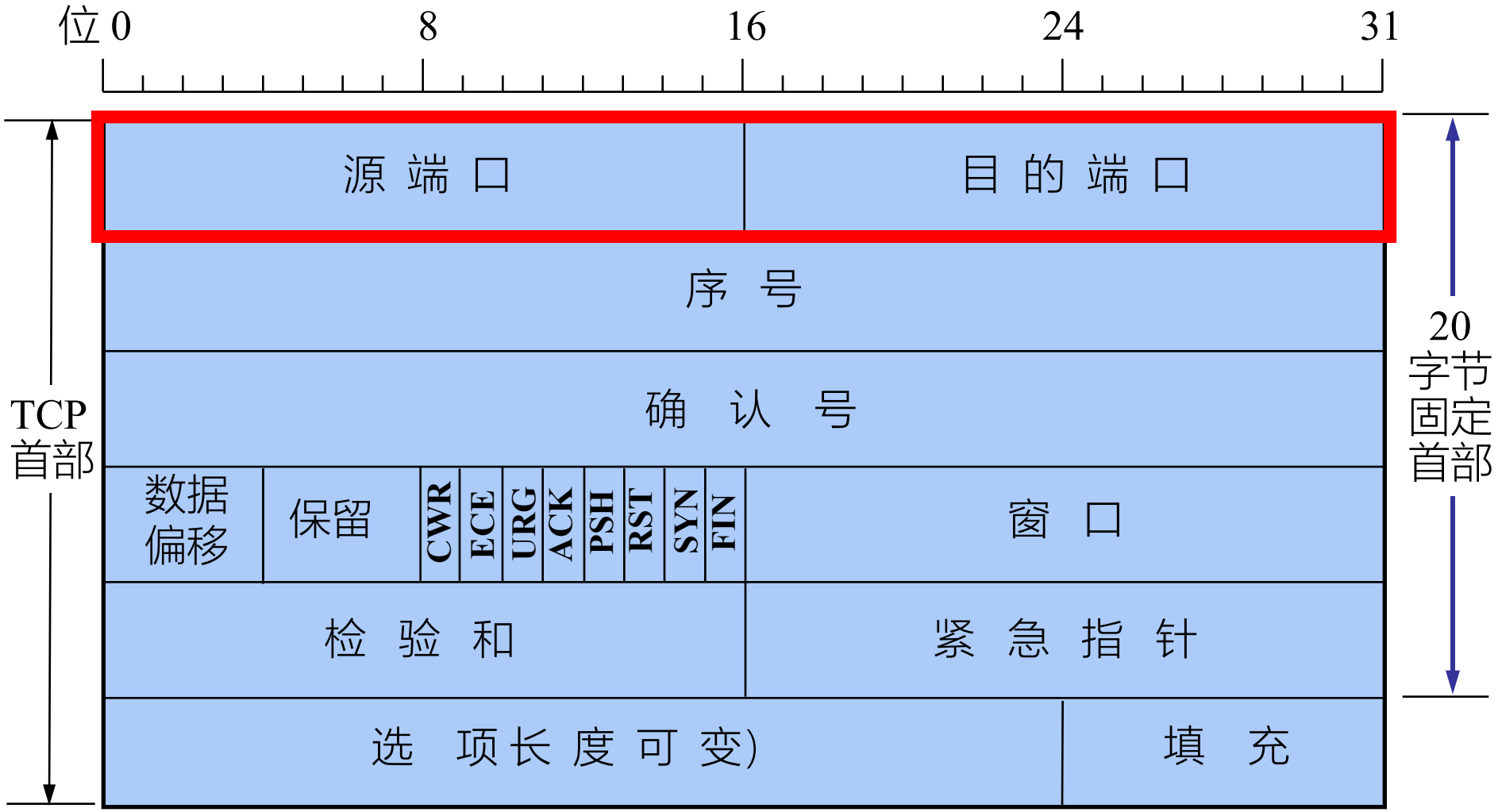
TCP提供的服务

- TCP在IP数据报中的封装
- 尽管TCP和UDP都使用相同的网络层（IP），TCP却向应用层提供与UDP完全不同的服务。TCP提供一种面向连接的、可靠的字节流服务
- TCP协议可以表述为一个没有选择确认或否认的滑动窗口协议，它的窗口大小是可变的

TCP的报头格式

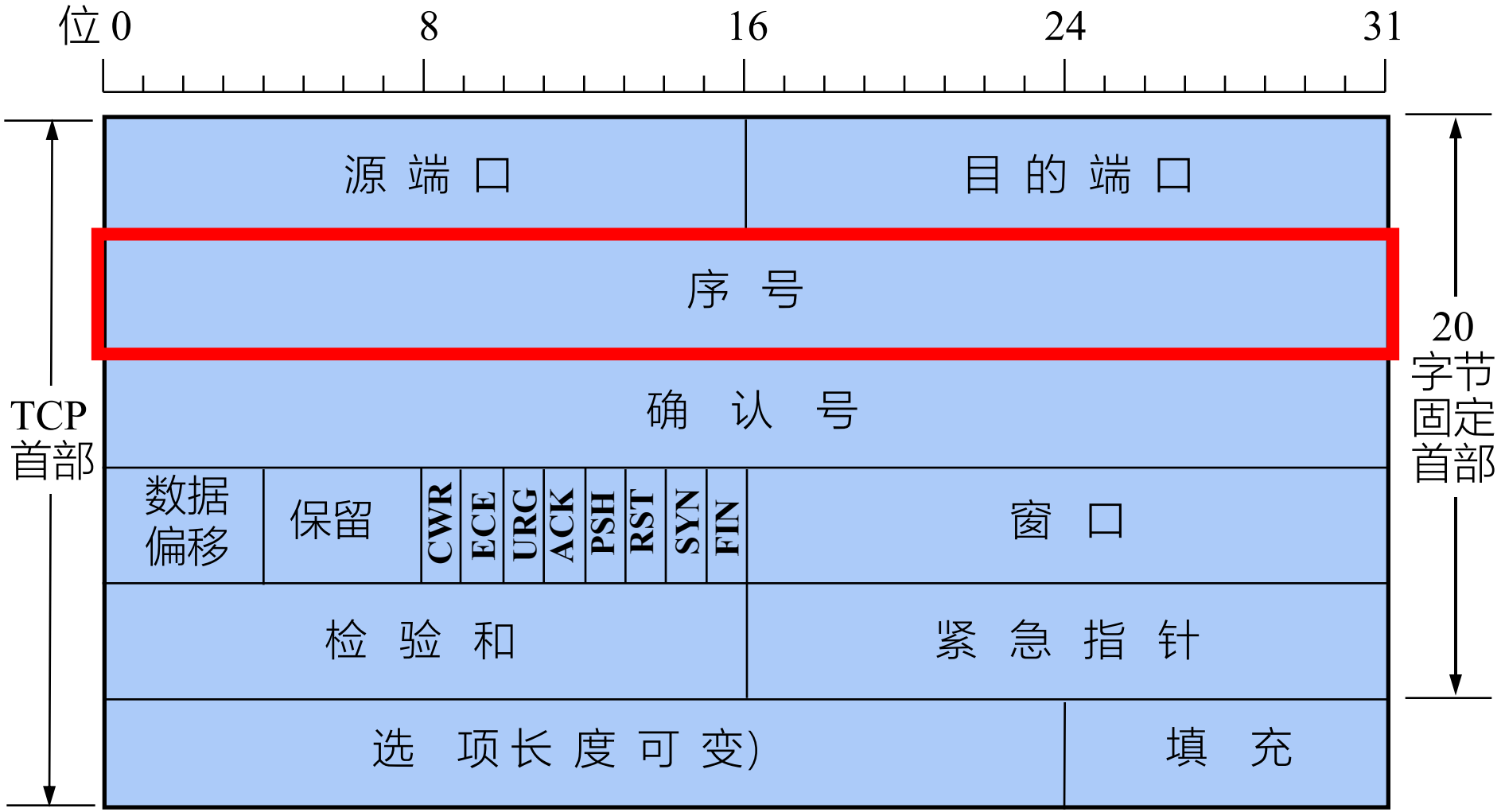

位 0
8
16
24
31
源 端 口
目 的 端 口
序 号
确 认 号
数据偏移
保留
CWR
ECE
URG
ACK
PSH
RST
SYN
FIN
窗 口
检 验 和
紧 急 指 针
选 项 （长 度 可 变）
填 充
TCP 首部
TCP 报文段
TCP 首部
TCP 数据部分
IP 首部
IP 数据部分

TCP的报头字段



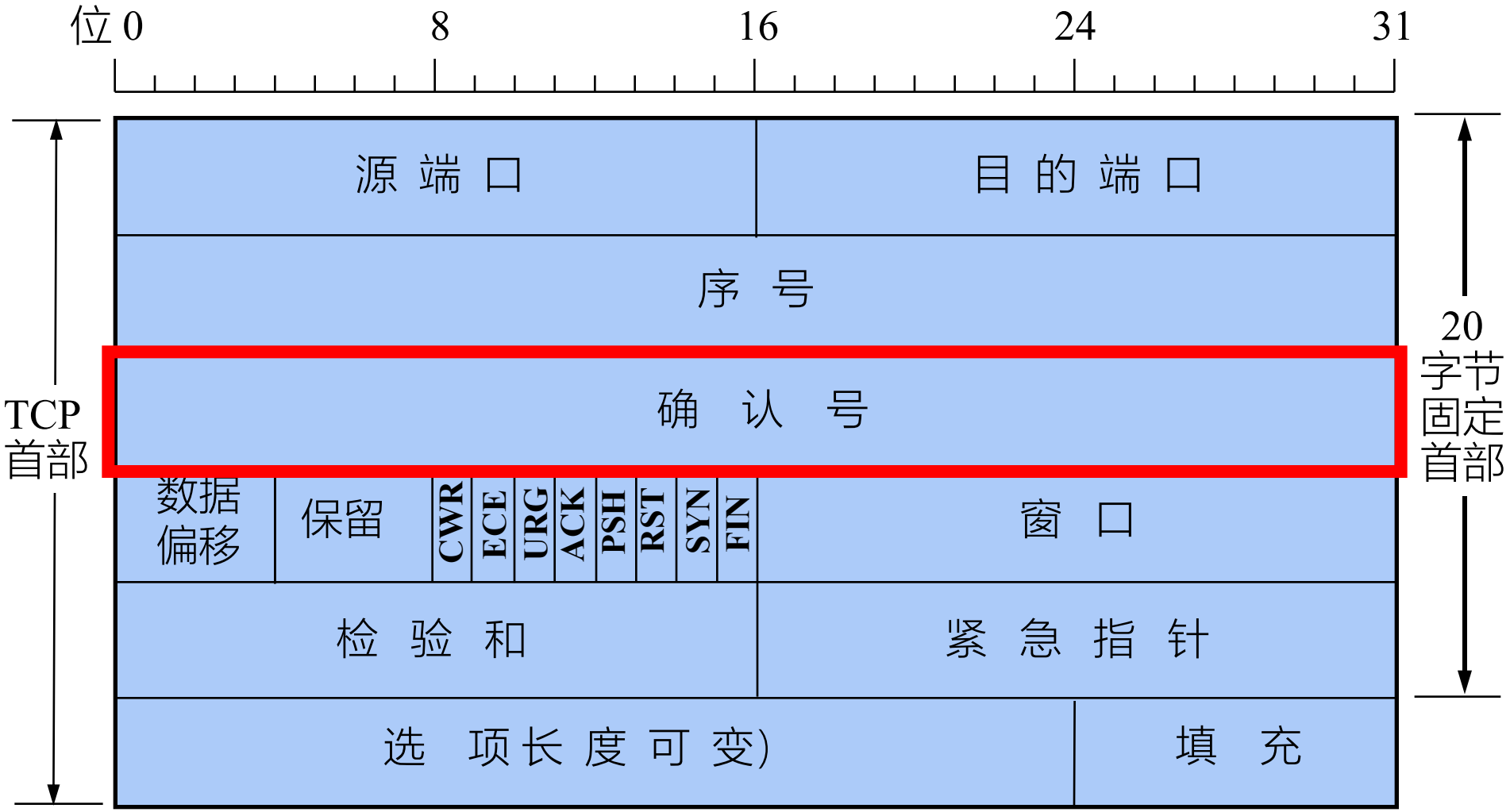
- 源端口、目的端口：各占 2 字节。端口是传输层与应用层的服务接口。传输层的复用和分用功能都要通过端口才能实现

TCP的报头字段



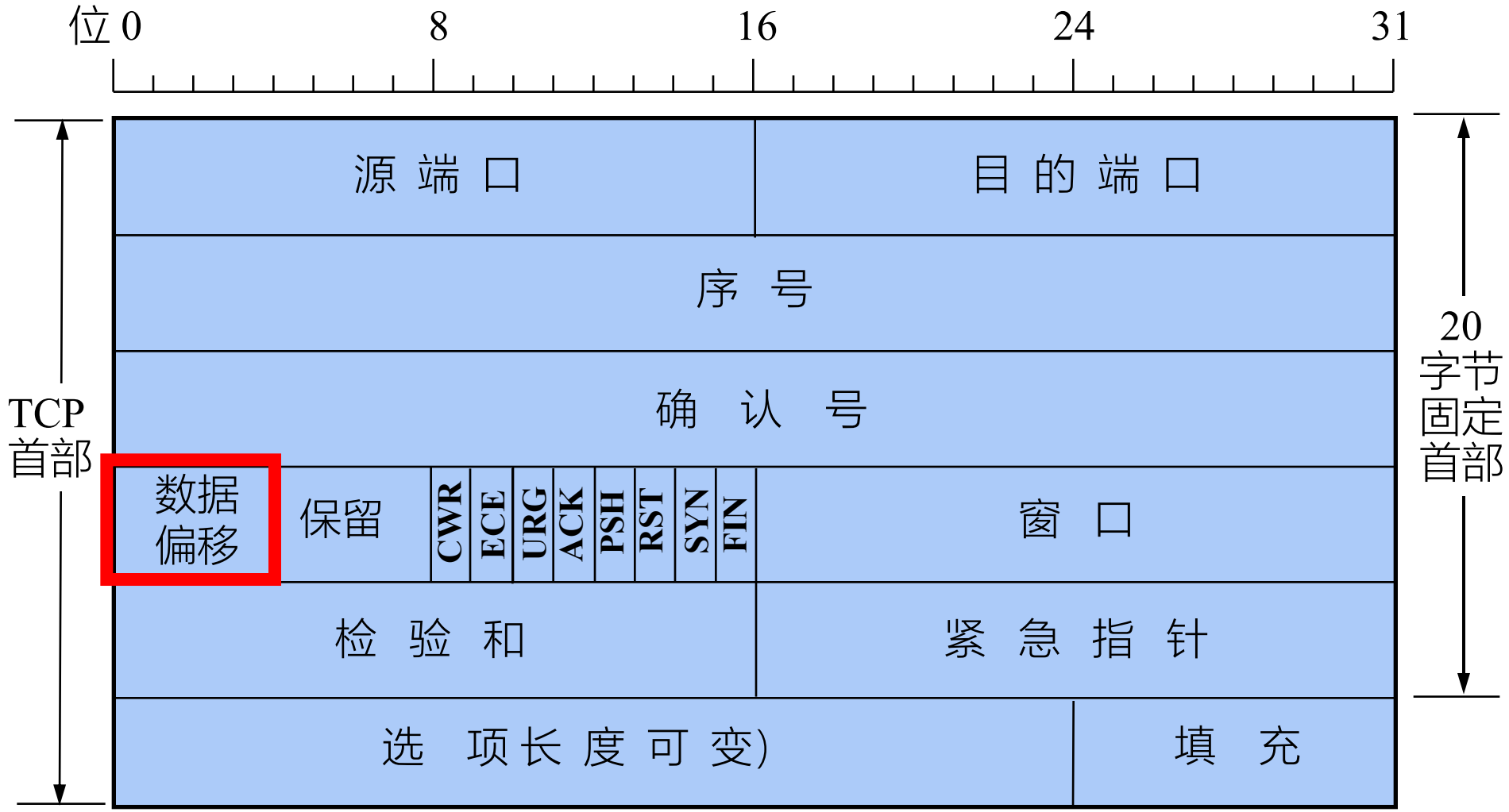
- 序号：占4字节。TCP 连接中传送的数据流中的每一个字节都编上一个序号。序号字段的值则指的是本报文段所发送的数据的第一个字节的序号

TCP的报头字段



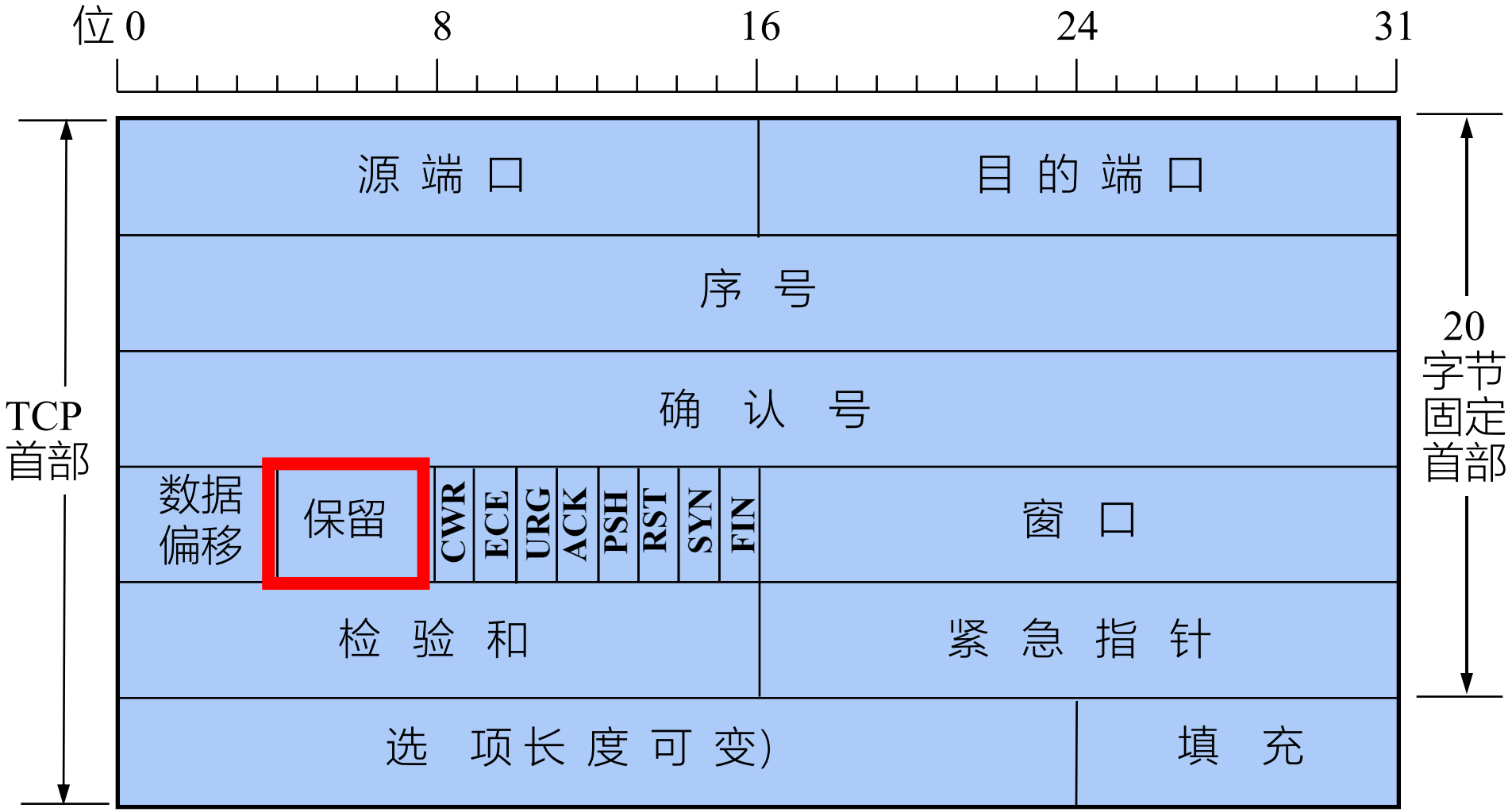
- 确认号：占4字节，是期望收到对方的下一个报文段的数据的第一个字节的序号

TCP的报头字段



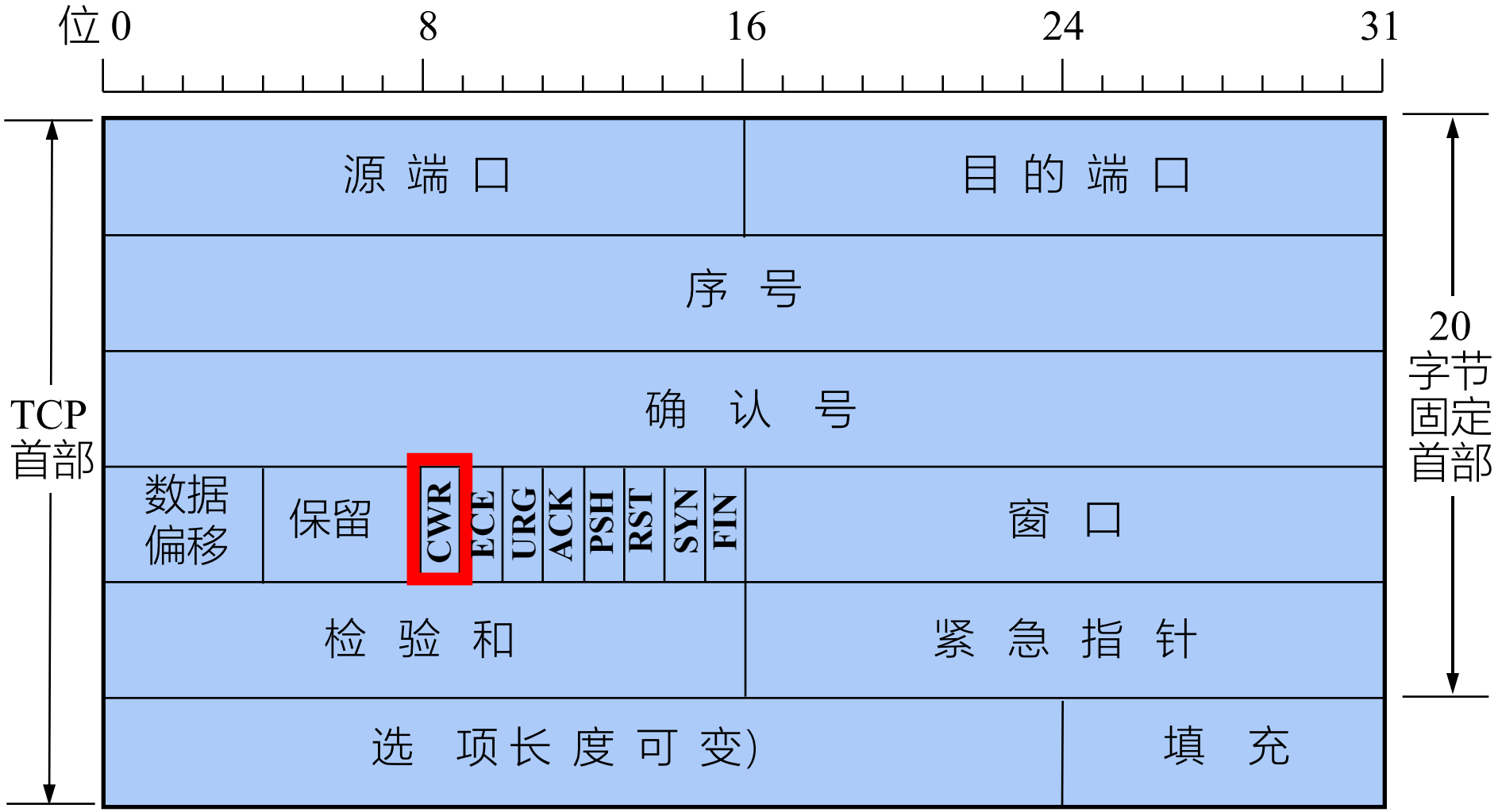
- 数据偏移(即首部长度)：占4位，它指出 TCP 报文段的数据起始处距离 TCP 报文段的起始处有多远。“数据偏移”的单位是 32 位字(以 4 字节为计算单位)

TCP的报头字段



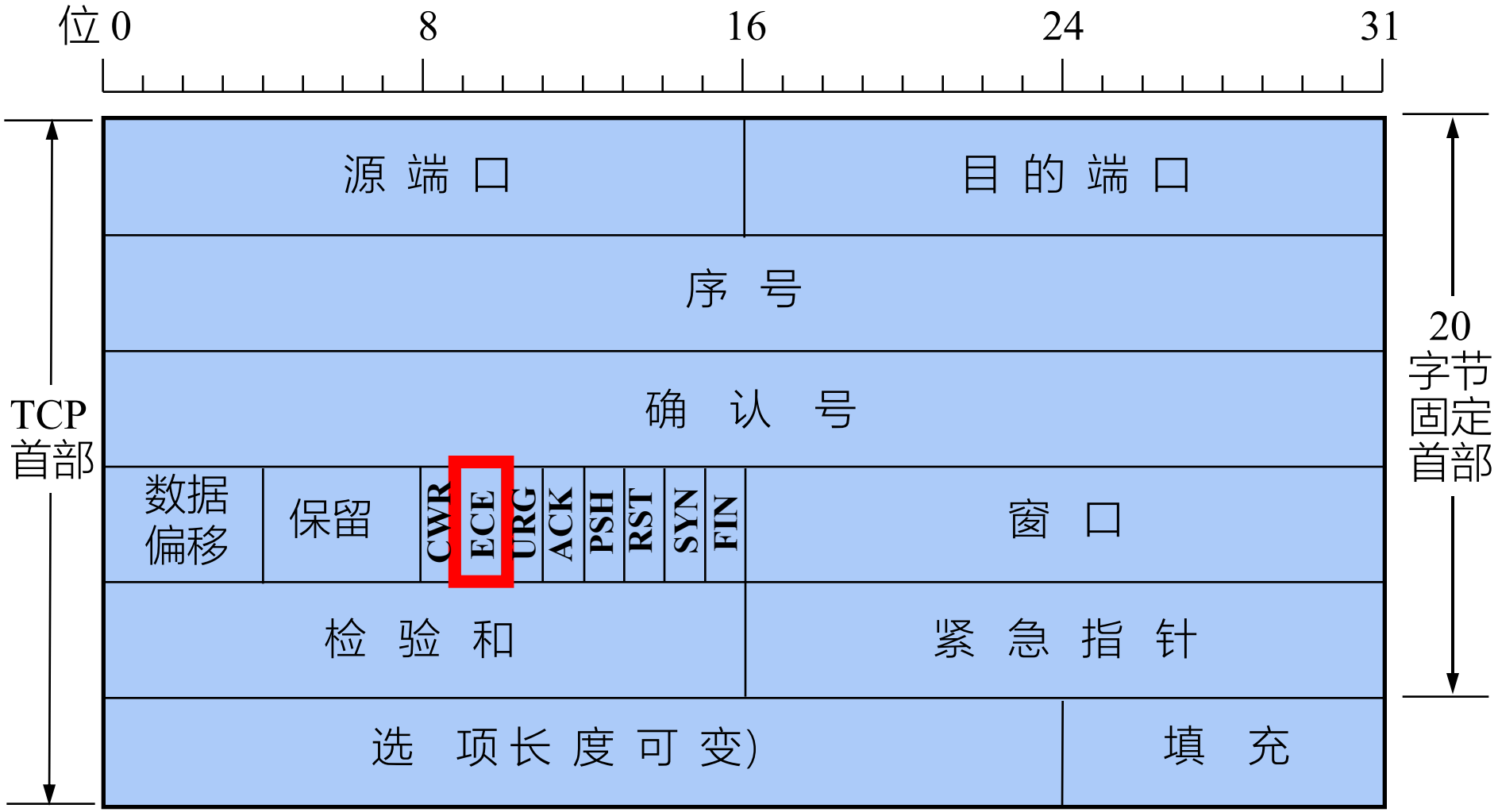
- 保留字段：占 4位，保留为今后使用，未用时应置为 0

TCP的报头字段



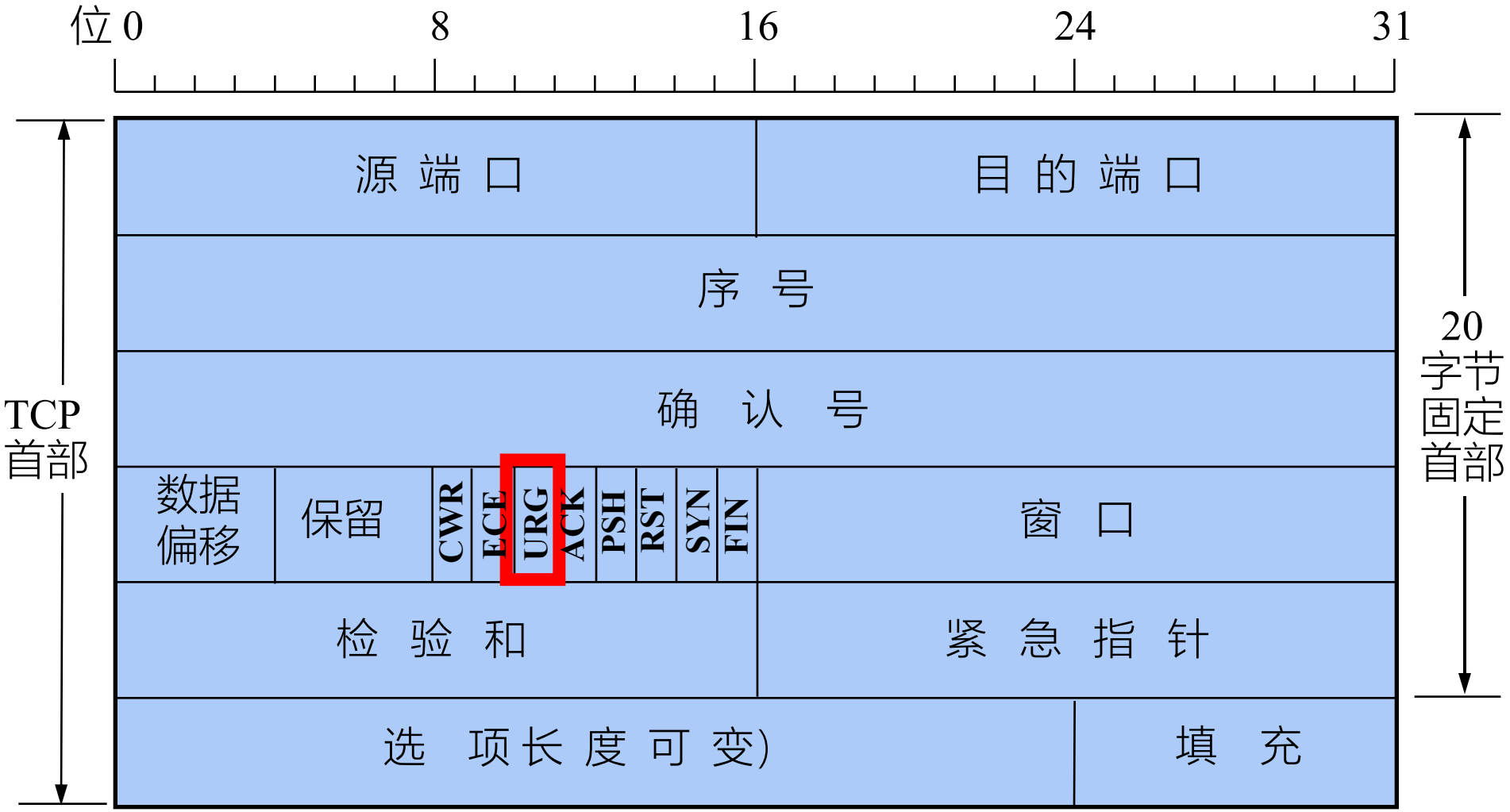
- 拥塞窗口减少CWR：用来表明它接收到了设置 ECE 标志的 TCP 包。并且，发送方收到消息之后，通过减小发送窗口的大小来降低发送速率

TCP的报头字段



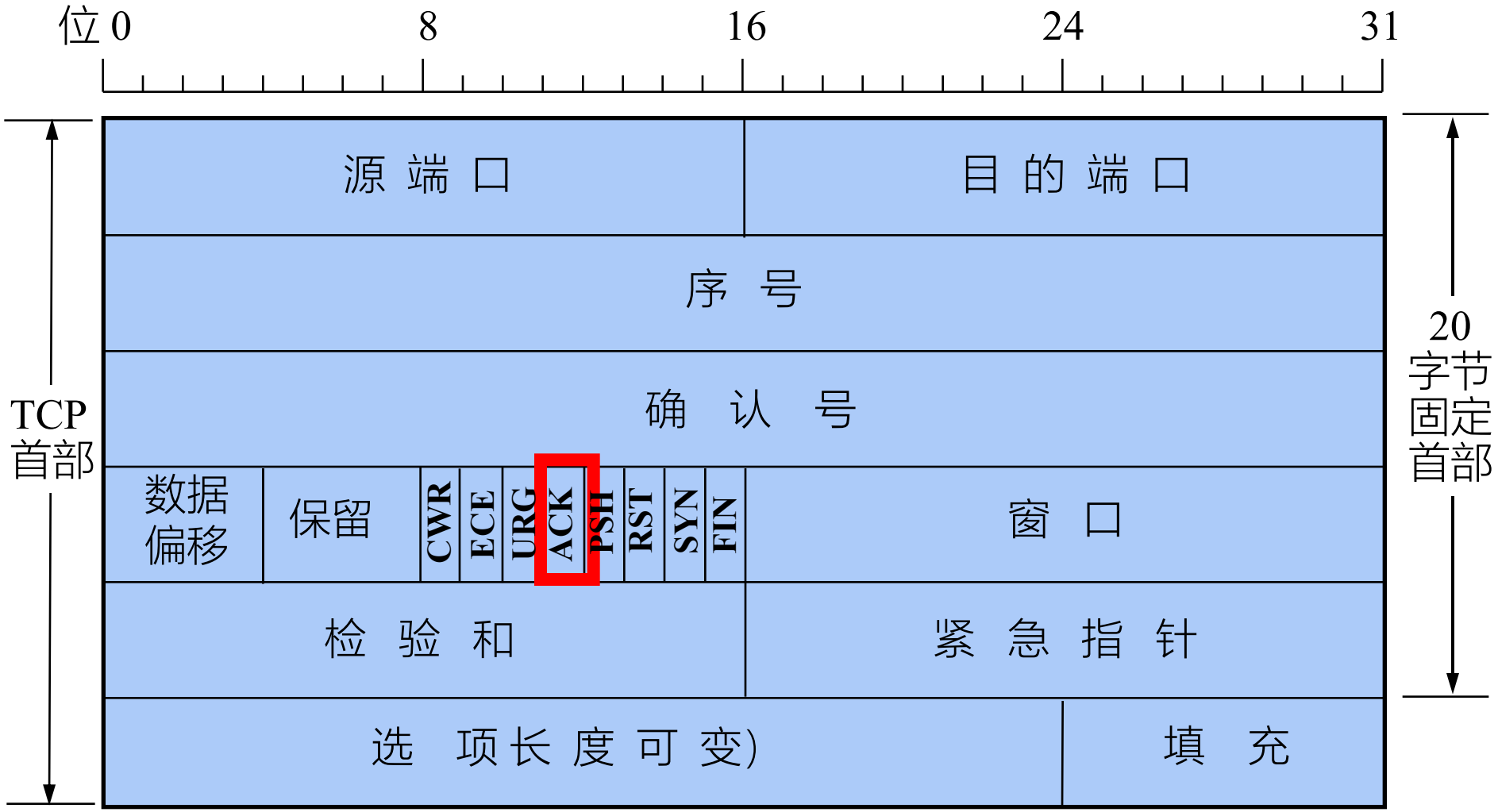
- ECN响应标志ECE：当ECE=1，表明接收到的 TCP 包的 IP 头部的 ECN 被设置为 1，即网络线路拥堵；在 TCP 三次握手时表明一个 TCP 端是具备 ECN 功能的

TCP的报头字段



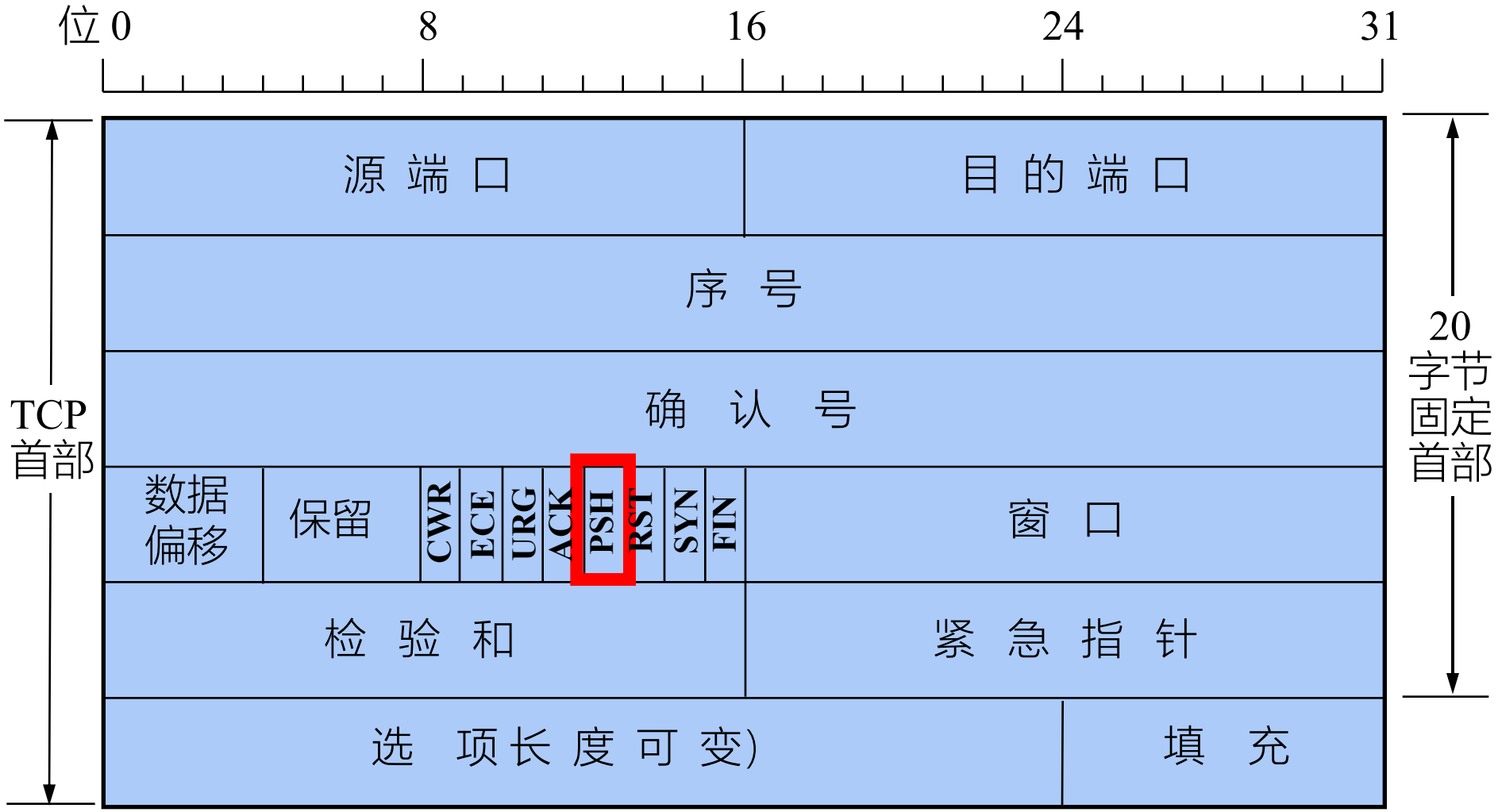
- 紧急URG：当URG =1时，表明紧急指针字段有效。此报文段中有紧急数据，系统应尽快传送(相当于高优先级的数据)

TCP的报头字段



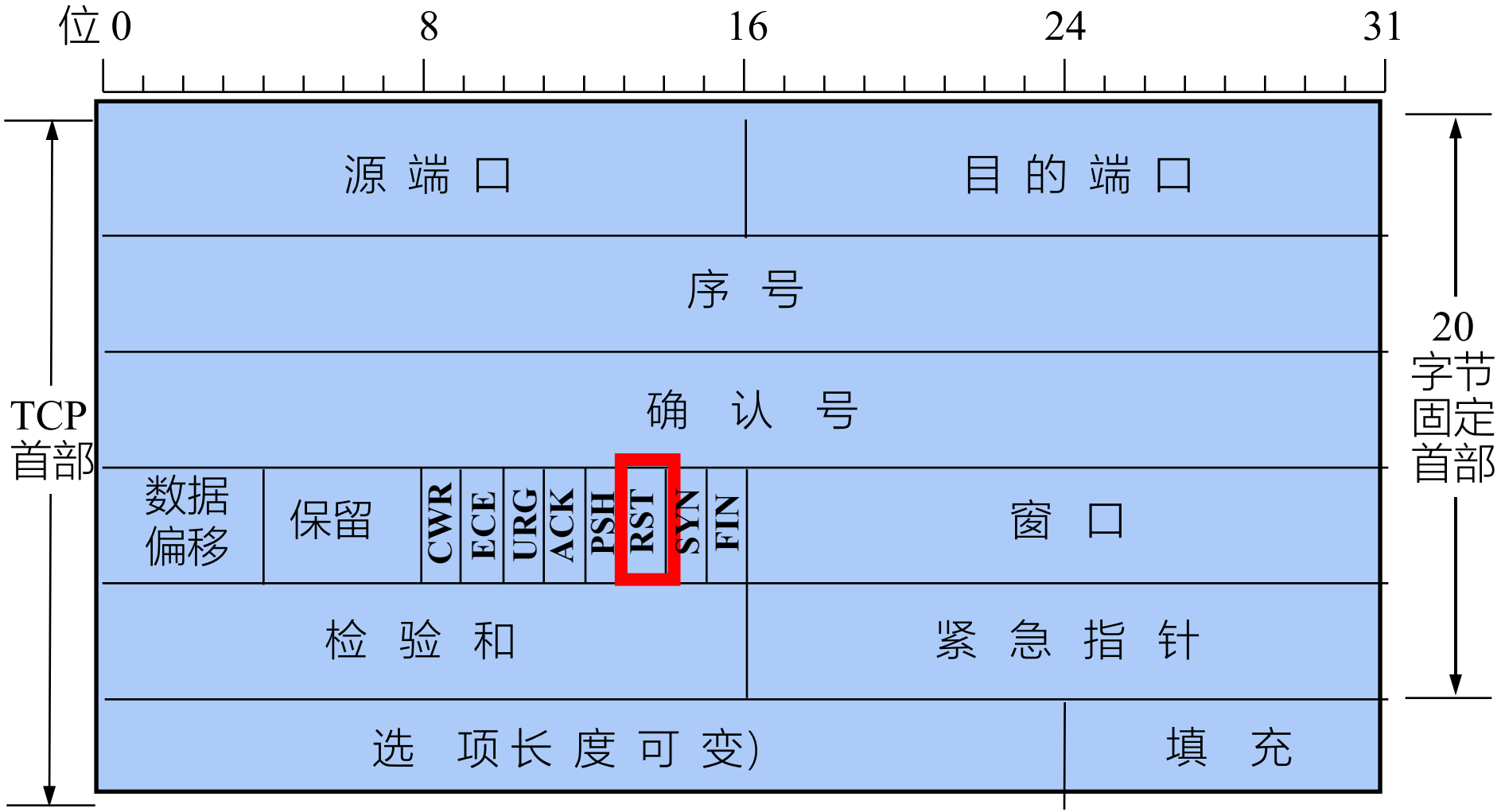
- 确认ACK：只有当 ACK=1时确认号字段才有效。当ACK=0 时，确认号无效

TCP的报头字段



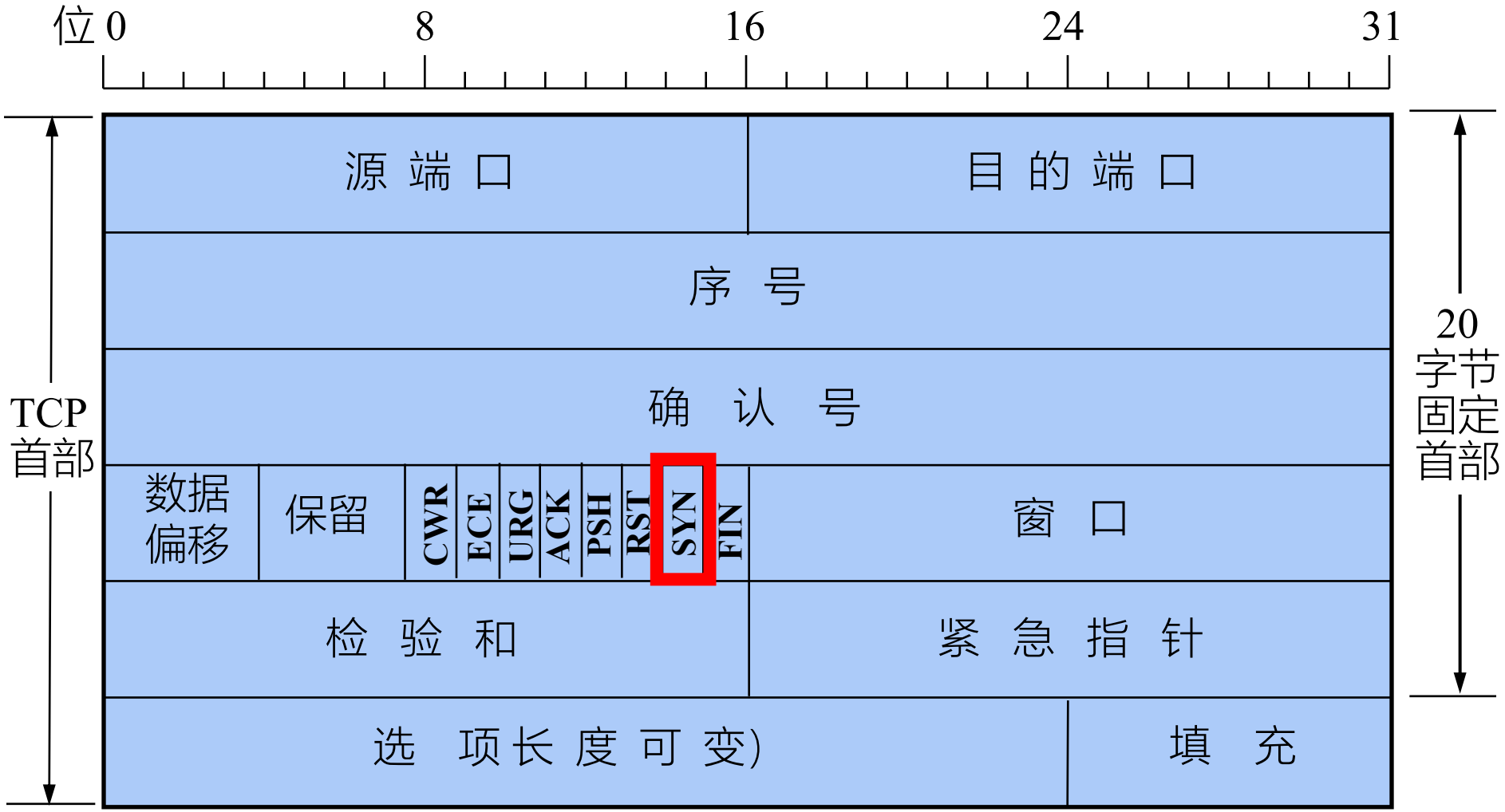
- 推送 PSH ： 接收TCP收到PSH=1的报文段，就尽快地交付接收应用进程，而不再等到整个缓存都填满后再向上交付

TCP的报头字段



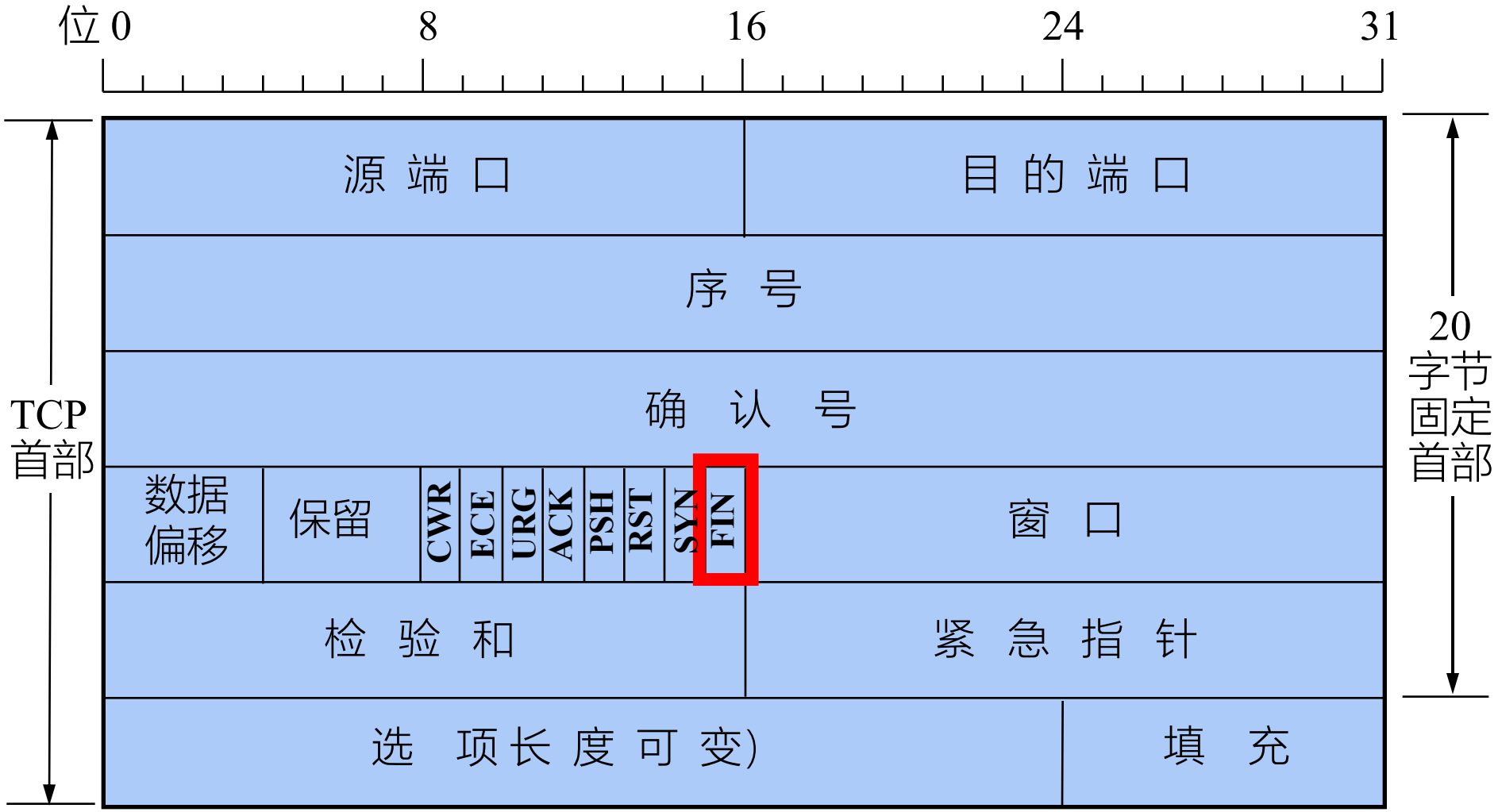
- 复位 RST ： 当RST=1时，表明TCP连接中出现严重差错(如由于主机崩溃或其他原因)，必须释放连接，然后再重新建立传输连接

TCP的报头字段



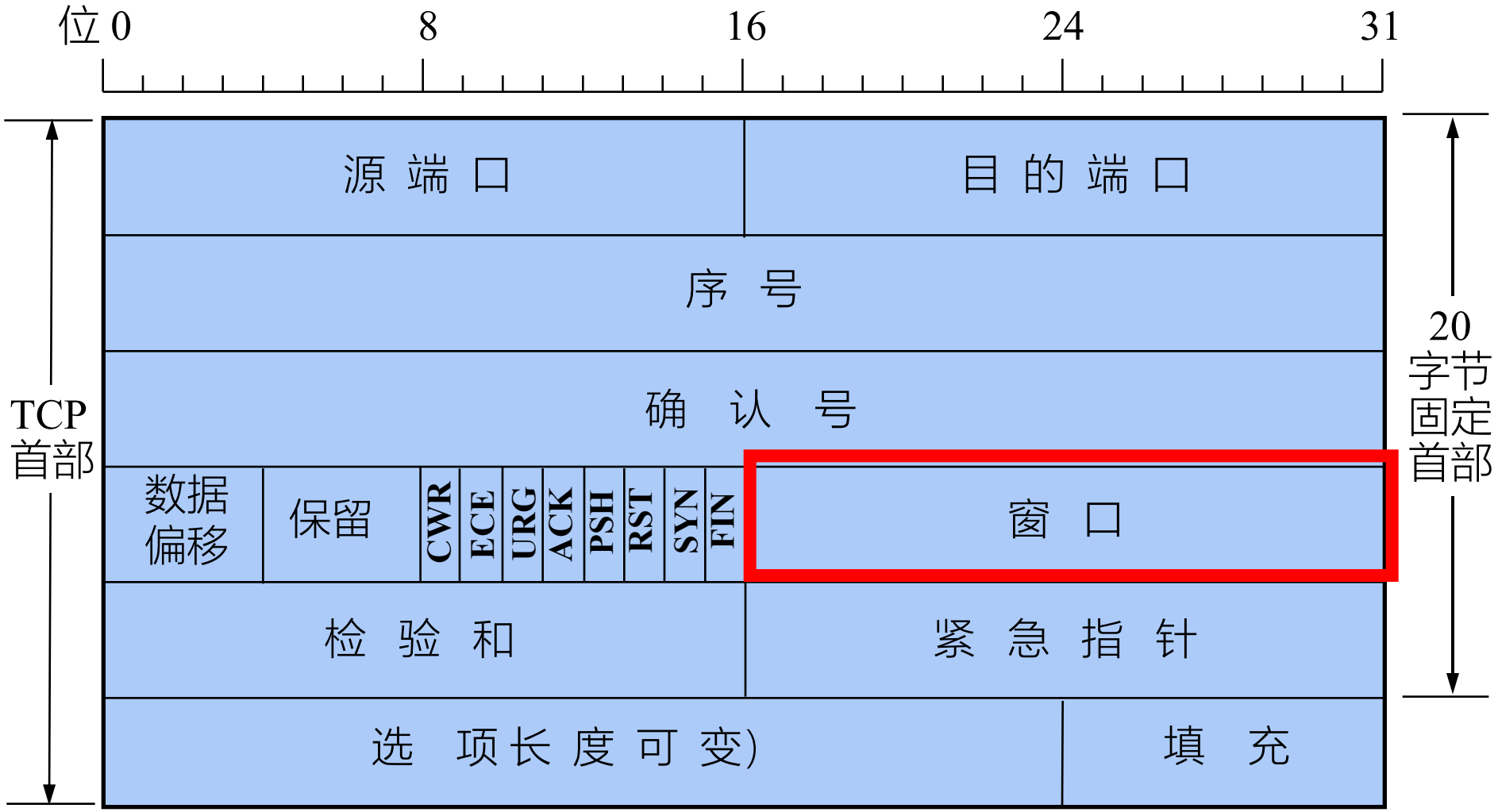
- 同步 SYN：同步SYN=1表示这是一个连接请求或连接接受报文

TCP的报头字段



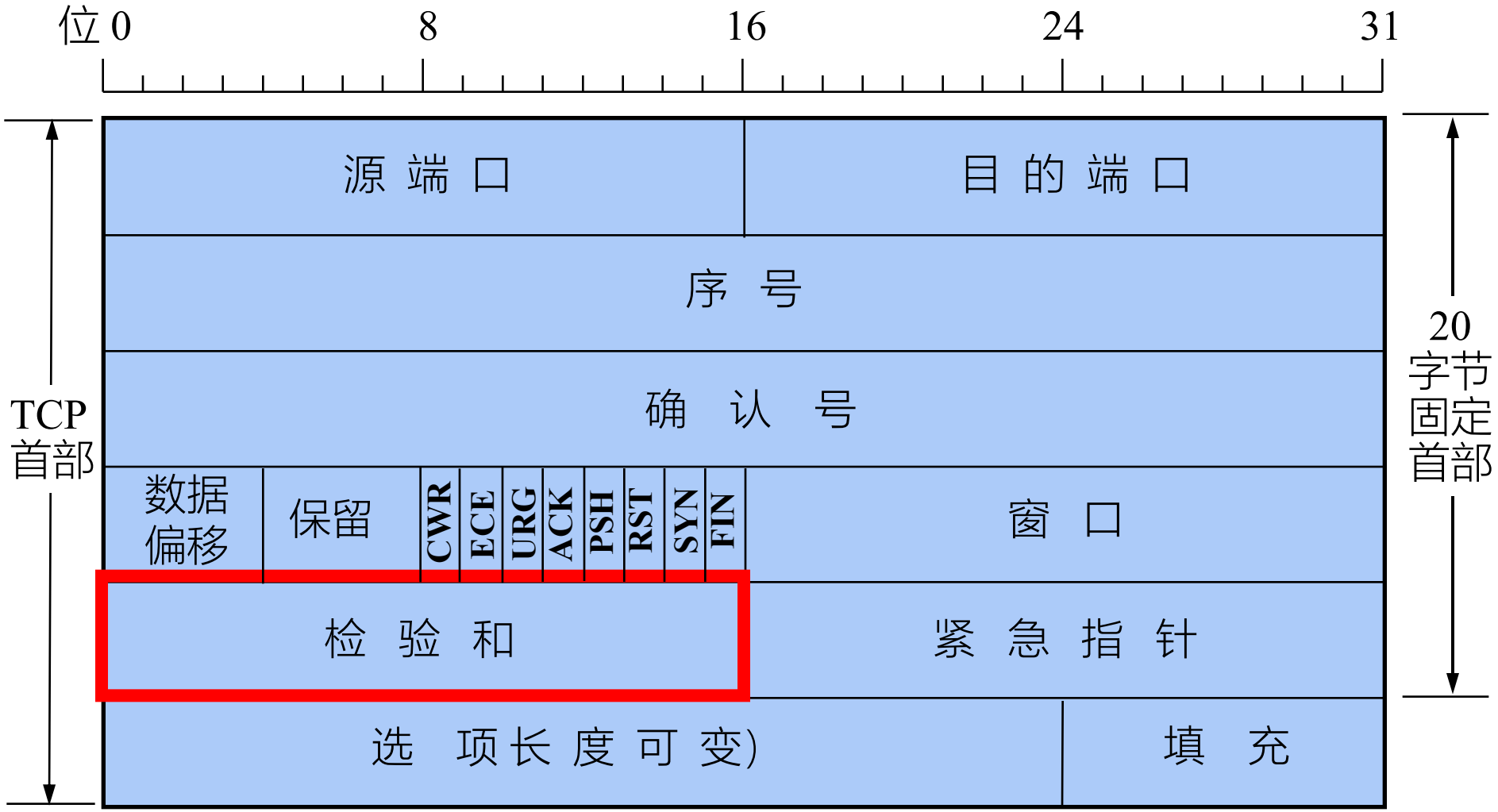
- 终止FIN：用来释放一个连接。FIN=1表明此报文段的发送端的数据已发送完毕，并要求释放连接

TCP的报头字段



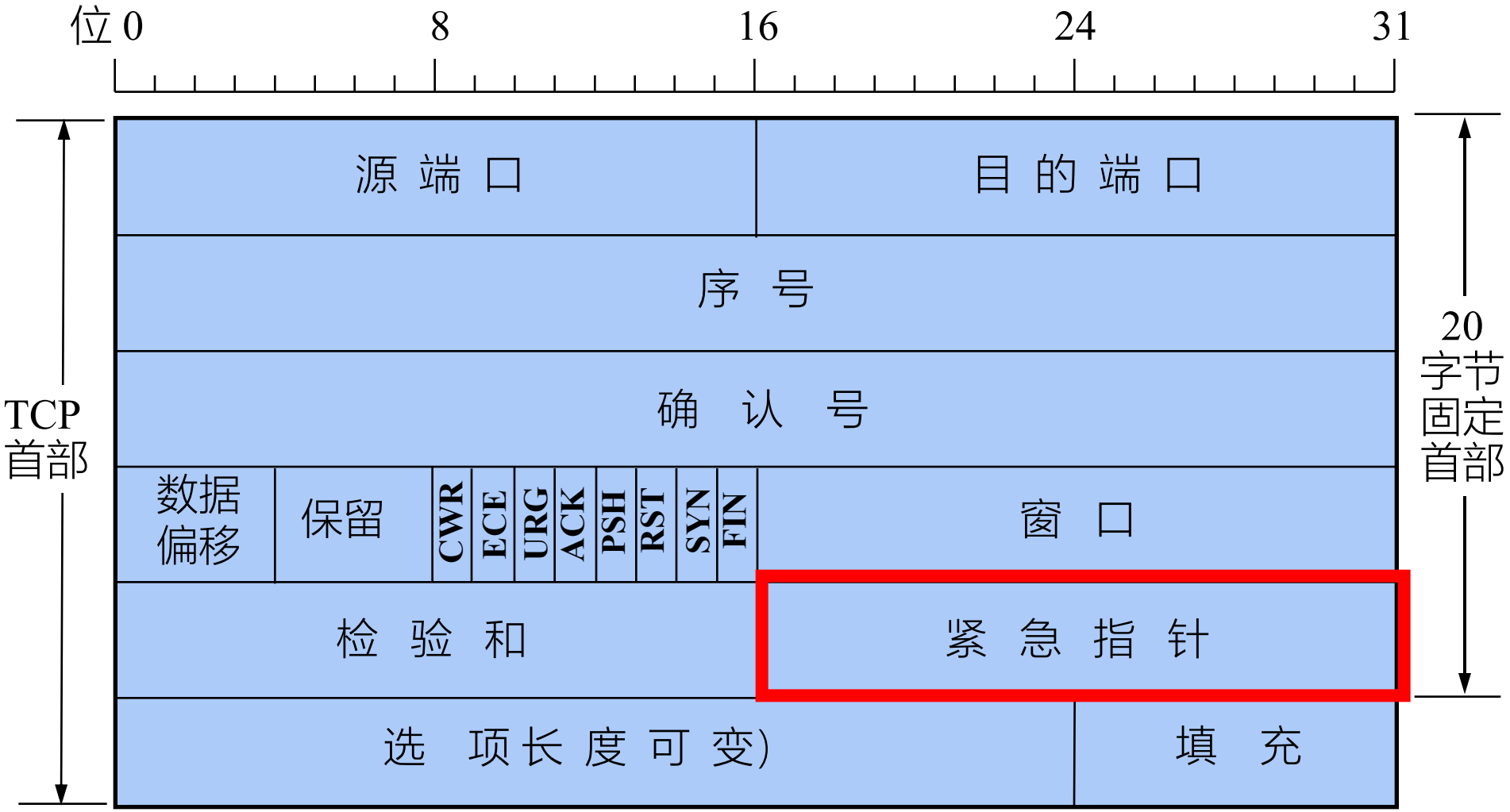
- 窗口：占2字节，用来让对方设置发送窗口的依据，单位为字节

TCP的报头字段



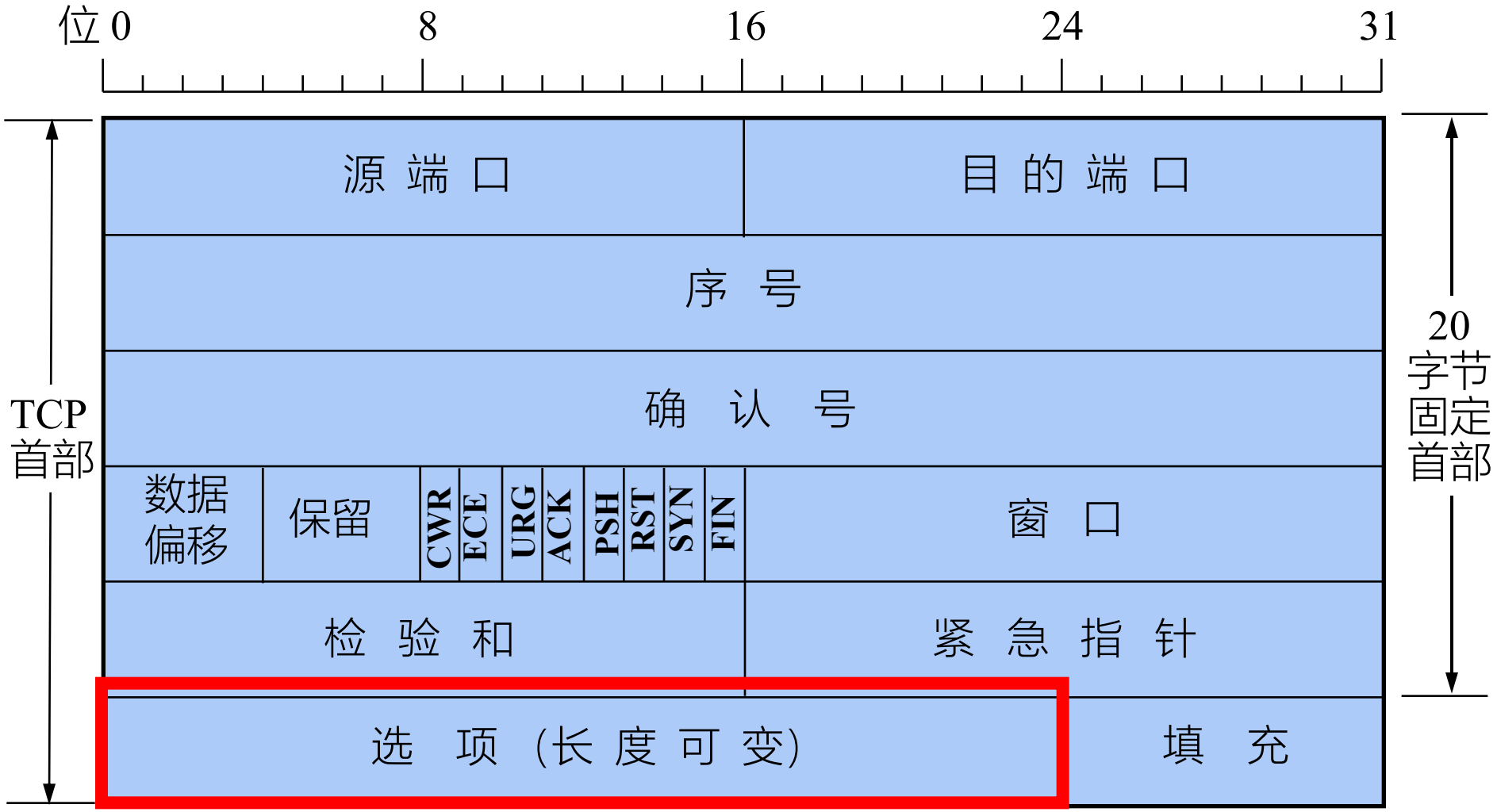
- 检验和：占2字节。检验和字段检验的范围包括首部和数据这两部分。在计算检验和时，要在TCP报文段的前面加上12字节的伪首部

TCP的报头字段



- 紧急指针：16位偏移量，指出在本报文段中紧急数据共有多少个字节(紧急数据放在本报文段数据的最前面)，和序号字段中的值相加表示紧急数据最后一个字节的序号

TCP的报头字段

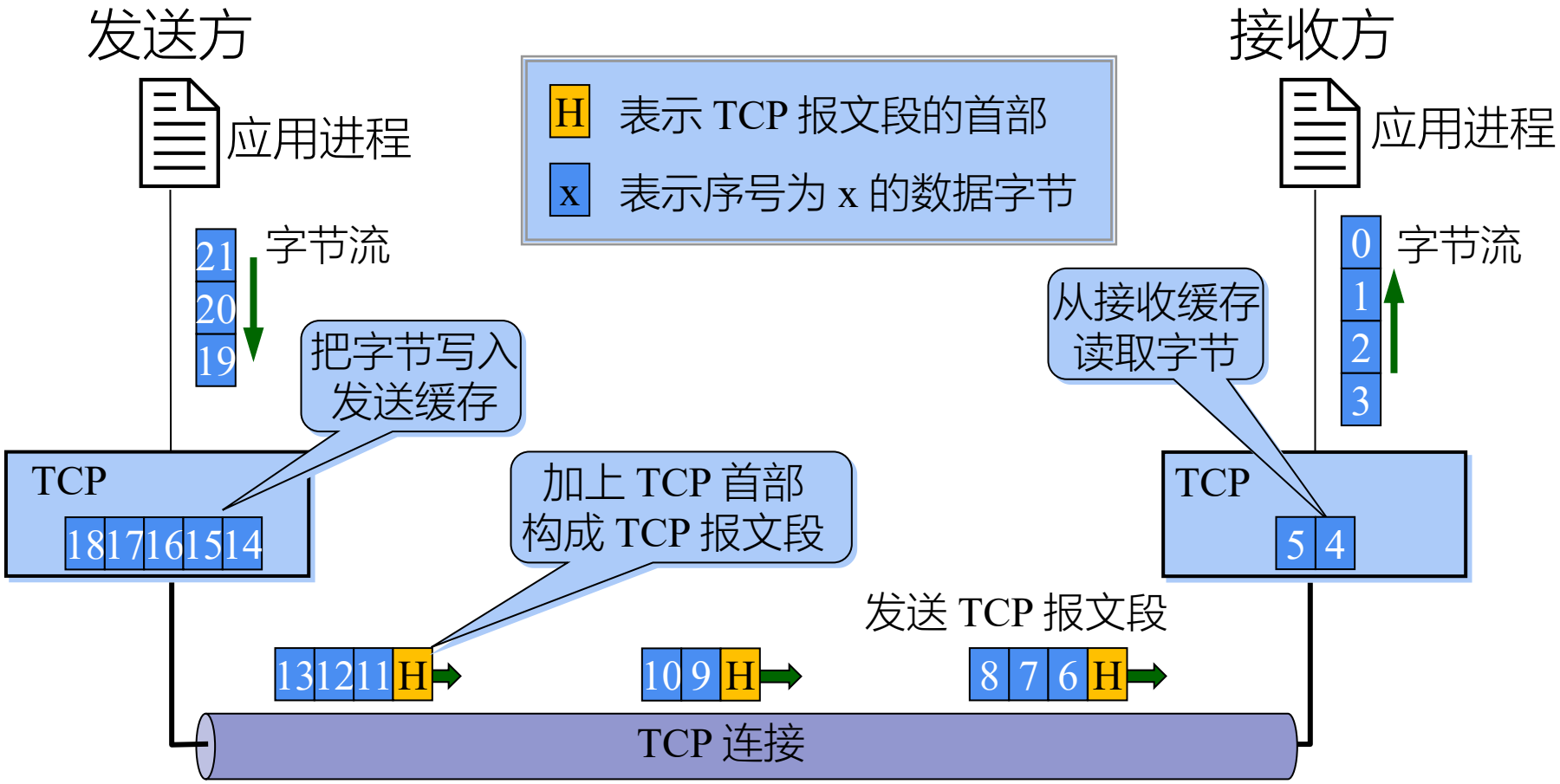


- 选项：长度可变。TCP最初只规定了一种选项，即最大报文段长度 MSS。MSS 告诉对方TCP："我的缓存所能接收的报文段的数据字段的最大长度是MSS个字节"

TCP特性

- 在TCP连接中每个传输的字节都被计数，确认序号包含发送确认的一端所期望收到的下一个序号。因此，确认序号应当是上次已经成功收到数据字节序号加1。只有ACK标志为1时，确认序号字段才有效
- 发送ACK无需任何代价，因为32bit的确认序号字段和ACK标志一样，总是TCP首部的一部分。因此，一旦一个 TCP连接建立起来，ACK标志总是被设置1

TCP面向流的概念


发送方
应用进程
字节流
21
20
19
把字节写入
发送缓存
TCP
18 17 16 15 14
H 表示 TCP 报文段的首部
x 表示序号为 x 的数据字节
加上 TCP 首部
构成 TCP 报文段
13 12 11 H
10 9 H
8 7 6 H
发送 TCP 报文段
TCP 连接
接收方
应用进程
字节流
0
1
2
3
从接收缓存
读取字节
TCP
5 4

TCP特性

- TCP 可以描述为一个没有选择确认或否定的滑动窗口协议。TCP 缺少选择确认是因为TCP首部中的确认序号表示发送方已成功收到字节，但还不包含确认序号所指的字节。当前还无法对数据流中选定的部分进行确认
- 例如，如果1 ~ 1024字节已经成功收到，下一个报文段中包含序号从2049 ~ 3072的字节，接收端并不能确认这个新的报文段，它所能做的就是发回一个确认序号为1025的ACK

TCP所采用的技术

- 对付分组重复和乱序传递的排序技术
  - 发送端为每个分组附加一个序号
- 对付分组丢失的重传技术
  - 使用带重传的正向确认机制
- 避免分组重复的技术
- 防止数据过荷的流量控制技术

超时重传时间的选择

- 重传机制是 TCP 中最重要和最复杂的问题之一
- TCP 每发送一个报文段，就对这个报文段设置一次计时器。只要计时器设置的重传时间到但还没有收到确认，就要重传这一报文段
- 往返时延的方差很大
  - 由于 TCP 的下层是一个互联网环境，IP 数据报所选择的路由变化很大。因而传输层的往返时间的方差也很大

加权平均往返时间

- TCP 保留了 RTT 的一个加权平均往返时间 RTT_S（又称为平滑的往返时间）
- 第一次测量到 RTT 样本时，RTT_S 值就取为所测量到的 RTT 样本值。以后每测量到一个新的 RTT 样本，按下式重新计算一次 RTT_S：

$$\text{新的 } RTT_S = (1-\alpha)\times(\text{旧的 } RTT_S) + \alpha \times (\text{新的 RTT样本})$$

式中，$0 \le \alpha < 1$。若α很接近于0，表示 RTT 值更新较慢。若选择α接近于 1，则表示 RTT 值更新较快

- RFC 2988 推荐的α值为1/8，即0.125

超时重传时间 RTO

- RTO 应略大于上面得出的加权平均往返时间 RTT_S
- RFC 2988 建议使用下式计算 RTO：

$$RTO = RTT_S + 4 \times RTT_D$$

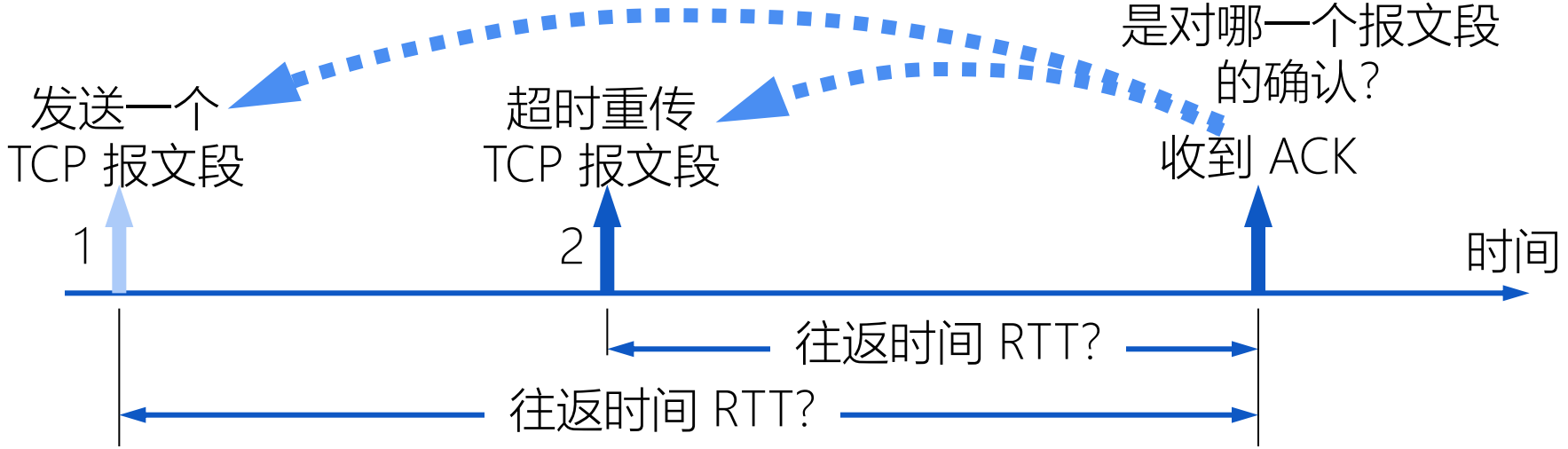
RTT_D 是 RTT 的偏差的加权平均值：

- RFC 2988 建议这样计算 RTT_D。第一次测量时，RTT_D 值取为测量到的 RTT 样本值的一半。在以后的测量中，则使用下式计算加权平均的 RTT_D：

$$新的RTT_D = (1 - \beta) \times (旧的RTT_D) + \beta \times |RTT_S - 新的\ RTT\ 样本|$$

β 是个小于 1 的系数，其推荐值是 1/4，即 0.25

Karn算法及其修正



- Karn：重传报文RTT不取样
  - 问题：超时重传时间无法更新
- 修正：报文段每重传一次，把 RTO 增大

$$\text{新的 RTO} = \gamma \times (\text{旧的 RTO})$$

系数γ 的典型值是 2 。不超时恢复计算方式

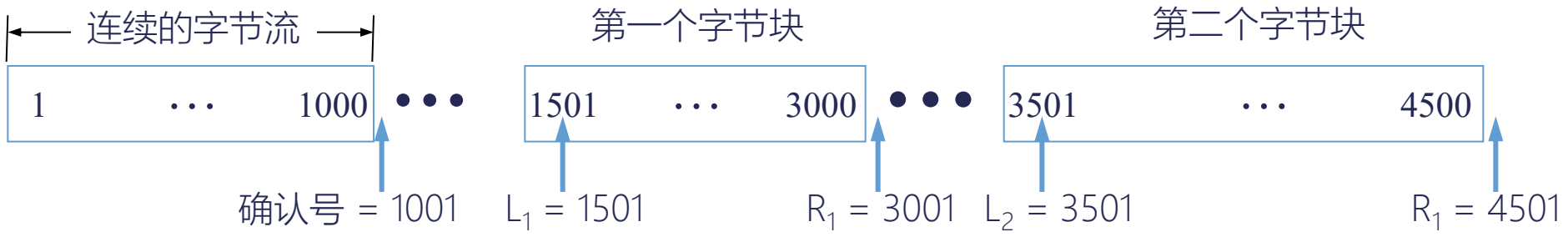
TCP计时器管理

- 重传计时器：数据包超时未被确认时，到时重传数据包
- 持久计时器：阻止死锁问题的发生，例如通告窗口包丢失
- 保活计时器：长连接场景，例如无数据传输时发送探测报文
- 关闭状态计时器：关闭连接时使用

TCP SACK

- TCP无法对一个报文段进行否认

  例如，如果收到包含1025~2048字节的报文段，但它的检查和有错误，TCP接收端所能做的就是发回一个确认序号为1025的ACK
- 解决办法：选择确认SACK (Selective ACK, RFC2018)



RFC 2018的一些规定

- 如果要使用选择确认，那么在建立 TCP 连接时，就要在 TCP 首部的选项中加上“允许 SACK”的选项，而双方必须都事先商定好。原来首部中的“确认号字段”的用法仍然不变
- 如果使用选择确认，那么需要2个字节用来指明该选项：
  - 1个字节用于SACK选项
  - 1个字节用于指明选项的长度
- 由于首部选项的长度最多只有 40 字节，而指明一个边界就要用掉 4 字节，因此在选项中最多只能指明 4 个字节块的边界信息

TCP何时发送数据

- 用不同的机制来控制 TCP 报文段的发送时机:
  - TCP 维持一个变量，它等于最大报文段长度 MSS。只要缓存中存放的数据达到 MSS 字节时，就组装成一个 TCP 报文段发送出去
  - 由发送方的应用进程指明要求发送报文段，即 TCP 支持的推送(push)操作
  - 发送方的一个计时器期限到了，这时就把当前已有的缓存数据装入报文段（但长度不能超过 MSS）发送出去
- 但不同的应用，发送时机仍然较为复杂

提高性能的措施

- Nagle算法：
  - 若发送进程把要发送的数据逐个字节送到TCP的发送缓存，则发送方把第1个字节发送出去，把后续的字节缓存起来
  - 当收到对第1个字节的确认后，在将缓存中的数据组装成一个报文段发送出去，同时继续对后续数据进行缓存
  - 当到达的数据已达到发送窗口大小的一半或者已达到最大报文段时，立即发送一个报文段

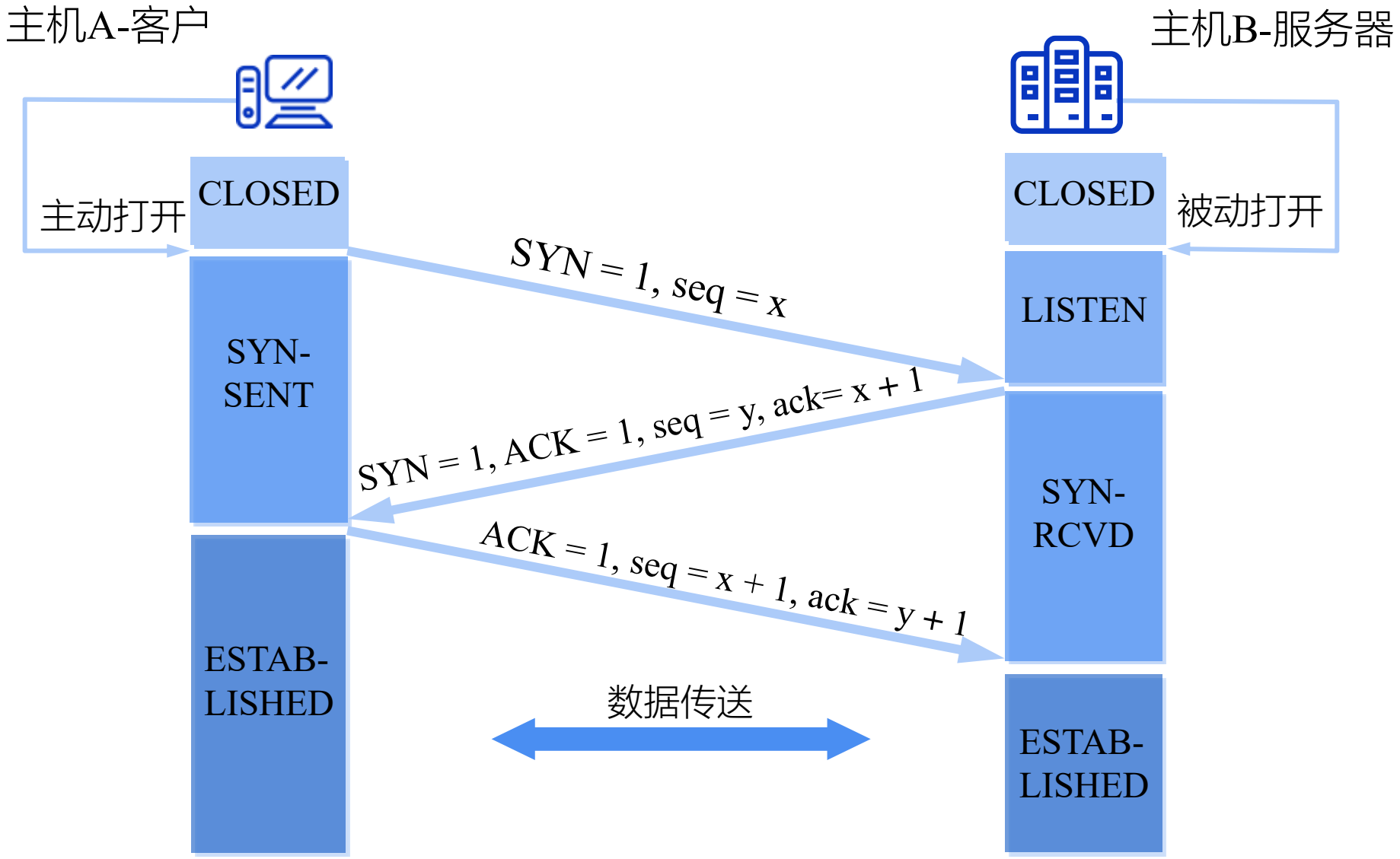
提高性能的措施

- 糊涂窗口综合症：TCP接收缓存已满，而应用进程一次只从缓存中读取一个字节
  - 让接收方等待一段时间，使得
    - 或者接收缓存可容纳一个MSS
    - 或者接受缓冲一半空间空闲
  - 发送方不要发送太小的报文

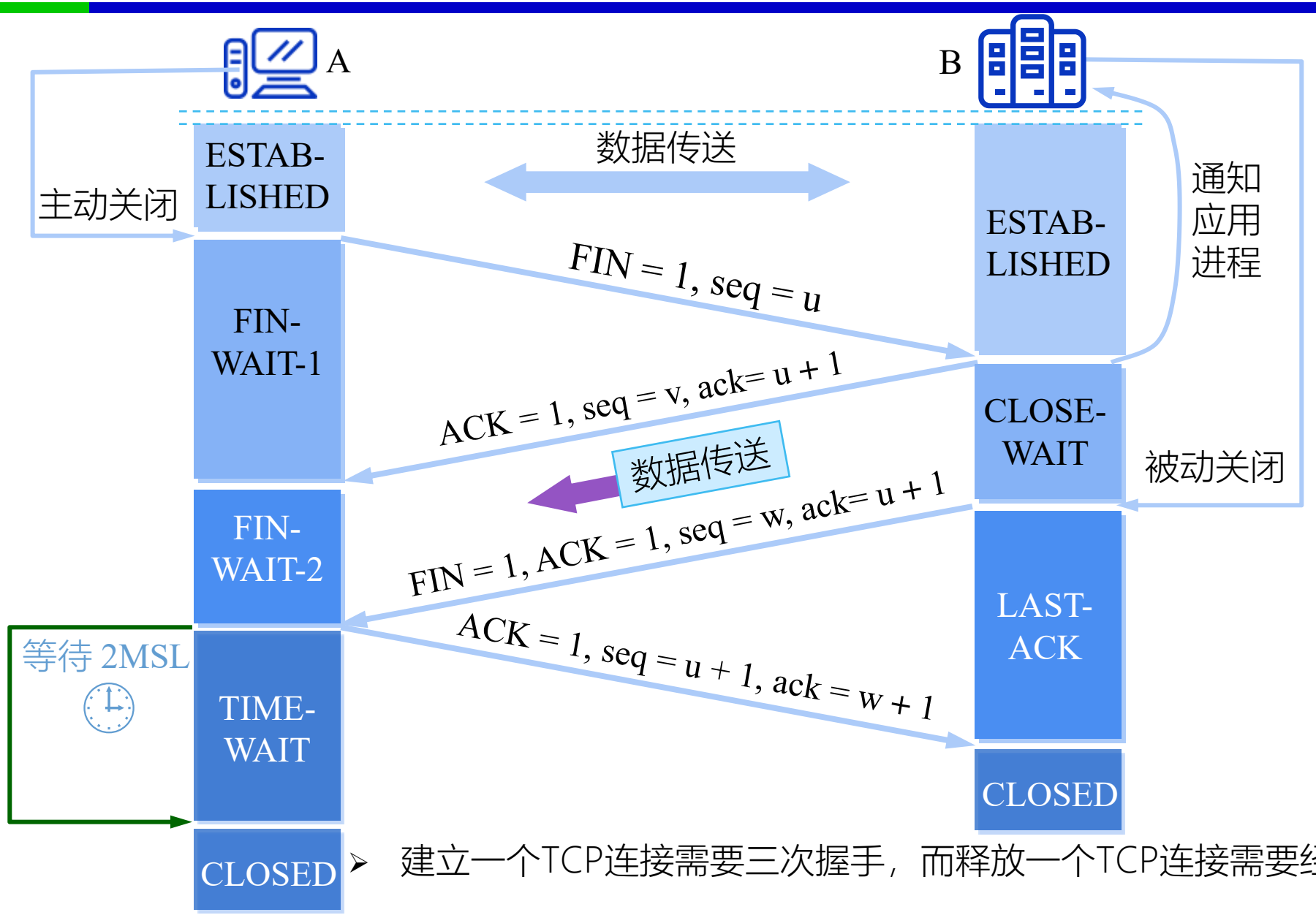
连接建立

- TCP是一个面向连接的协议，通信双方在发送数据之前都必须建立一个TCP连接
- 连接建立过程中要解决的问题：
  - 要使每一方能够确知对方的存在
  - 要允许双方协商一些参数（如最大报文段长度，最大窗口大小，服务质量等）
  - 能够对传输实体资源（如缓存大小，连接表中的项目等）进行分配
- TCP连接的建立过程称为三次握手(3-way handshake)
  - 避免已失效的连接请求报文

连接建立报文序列


主机A-客户
主机B-服务器
主动打开
CLOSED
SYN-SENT
ESTAB-LISHED
CLOSED
被动打开
LISTEN
SYN-RCVD
ESTAB-LISHED
SYN = 1, seq = x
SYN = 1, ACK = 1, seq = y, ack= x + 1
ACK = 1, seq = x + 1, ack = y + 1
数据传送

TCP连接的释放

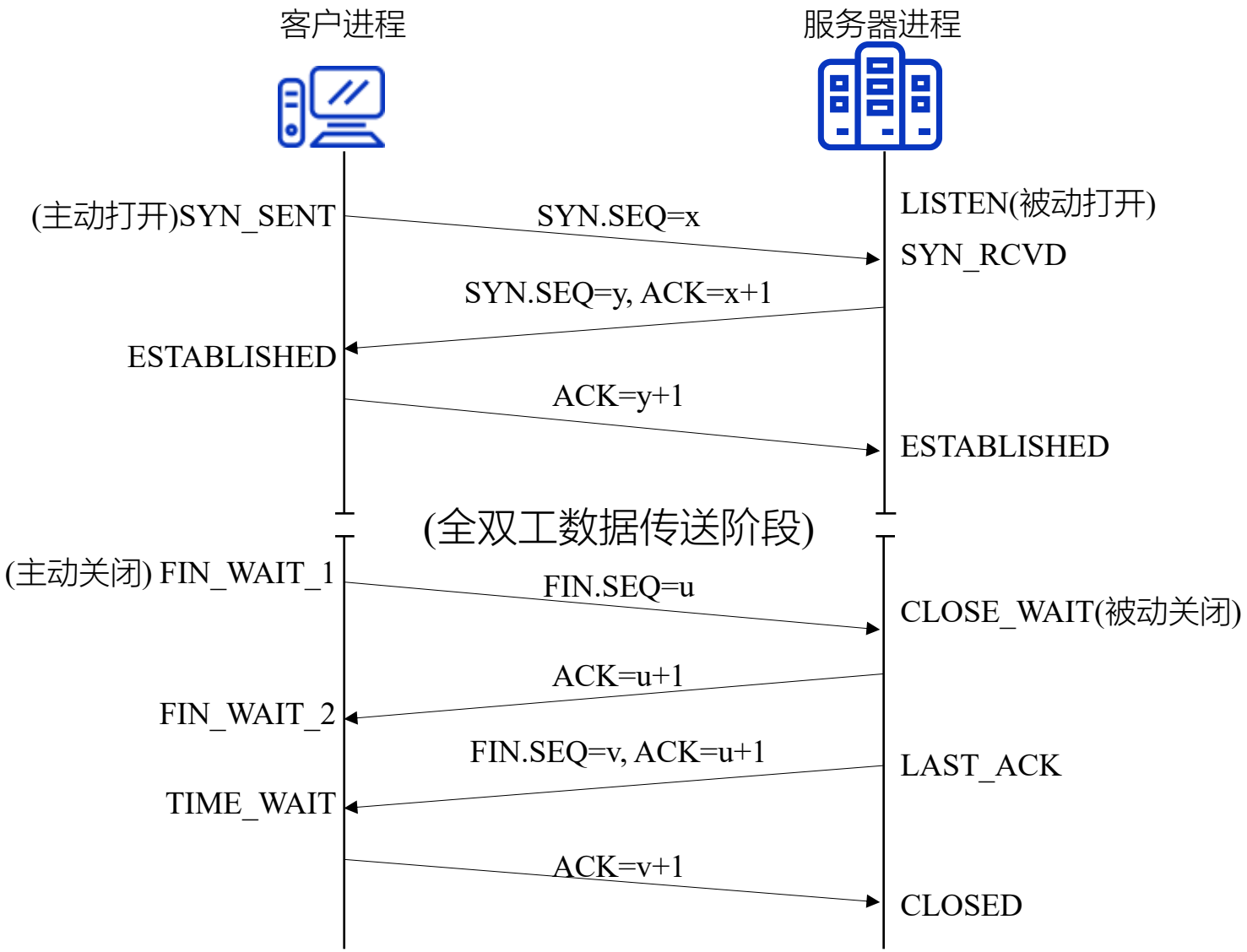


- 建立一个TCP连接需要三次握手，而释放一个TCP连接需要经过4次握手

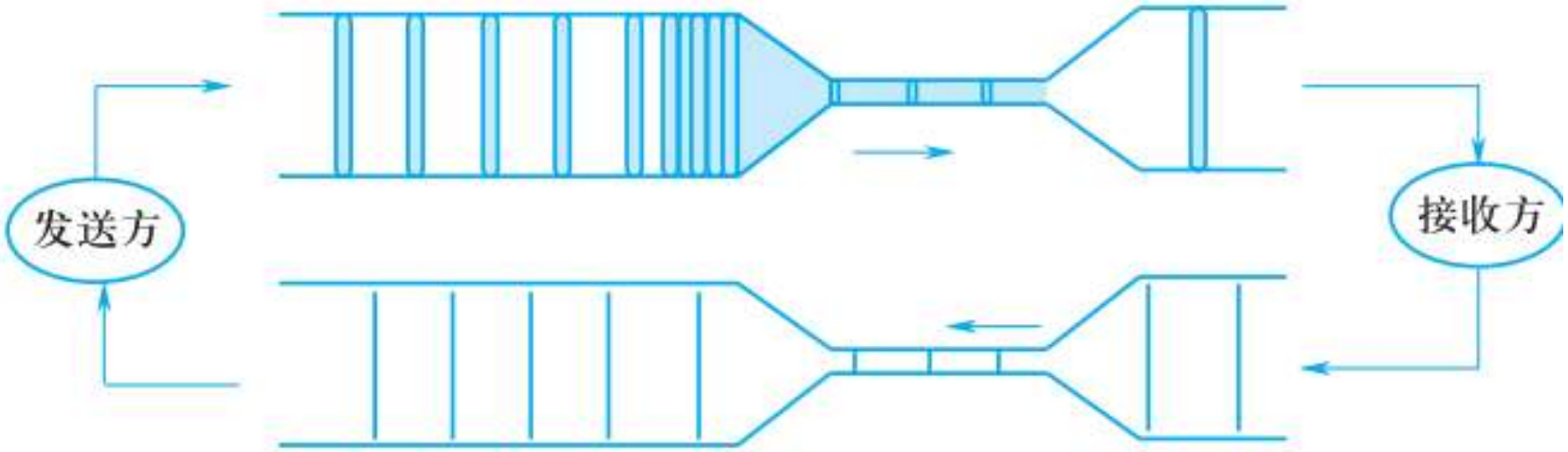
A必须等待2MSL的时间

- MSL：最长报文寿命Maximum Segment Lifetime
- 第一，为了保证 A 发送的最后一个 ACK 报文段能够到达 B，使B按正常步骤关闭
- 第二，防止 “已失效的连接请求报文段”出现在本连接中
  - A 在发送完最后一个 ACK 报文段后，再经过时间 2MSL，可使本连接持续的时间内所产生的所有报文段，都从网络中消失。这样就可以使下一个新的连接中不会出现这种旧的连接请求报文段

TCP正常的连接建立和关闭



TCP拥塞控制原理



- 报文段守恒原则
  - 每个用户都能确定网络中有多少可用容量，以便知道可以有效传输多少报文段
  - 防止过多的报文段注入网络，使网络中的路由器或者链路不至于过载
  - 在网络中发生拥塞时，减少向网络中发送数据的速率，以防止造成恶性循环
  - 在网络空闲时，提高发送数据的速率，以最大限度地利用网络资源

拥塞控制方式

- 端到端的拥塞控制
  - 端到端拥塞控制由发送方自己来判断是否拥塞，然后调整数据发送速率
- 网络辅助的拥塞控制
  - 由网络中的路由器来告诉发送方网络的拥塞情况
  - 通过网络层反馈的拥塞信息来实现拥塞控制的方法
    - 需要改造底层硬件，以得到网络设备的支持

传输层拥塞控制

- TCP拥塞控制有4种算法：
  - 慢启动
  - 拥塞避免
  - 快速重传
  - 快速恢复
- 几个术语
  - 拥塞窗口cwnd：TCP允许发送的最大数据量
  - 接收端窗口rwnd
  - 发送端最大数据段尺寸SMSS
  - 慢启动阀值ssthresh

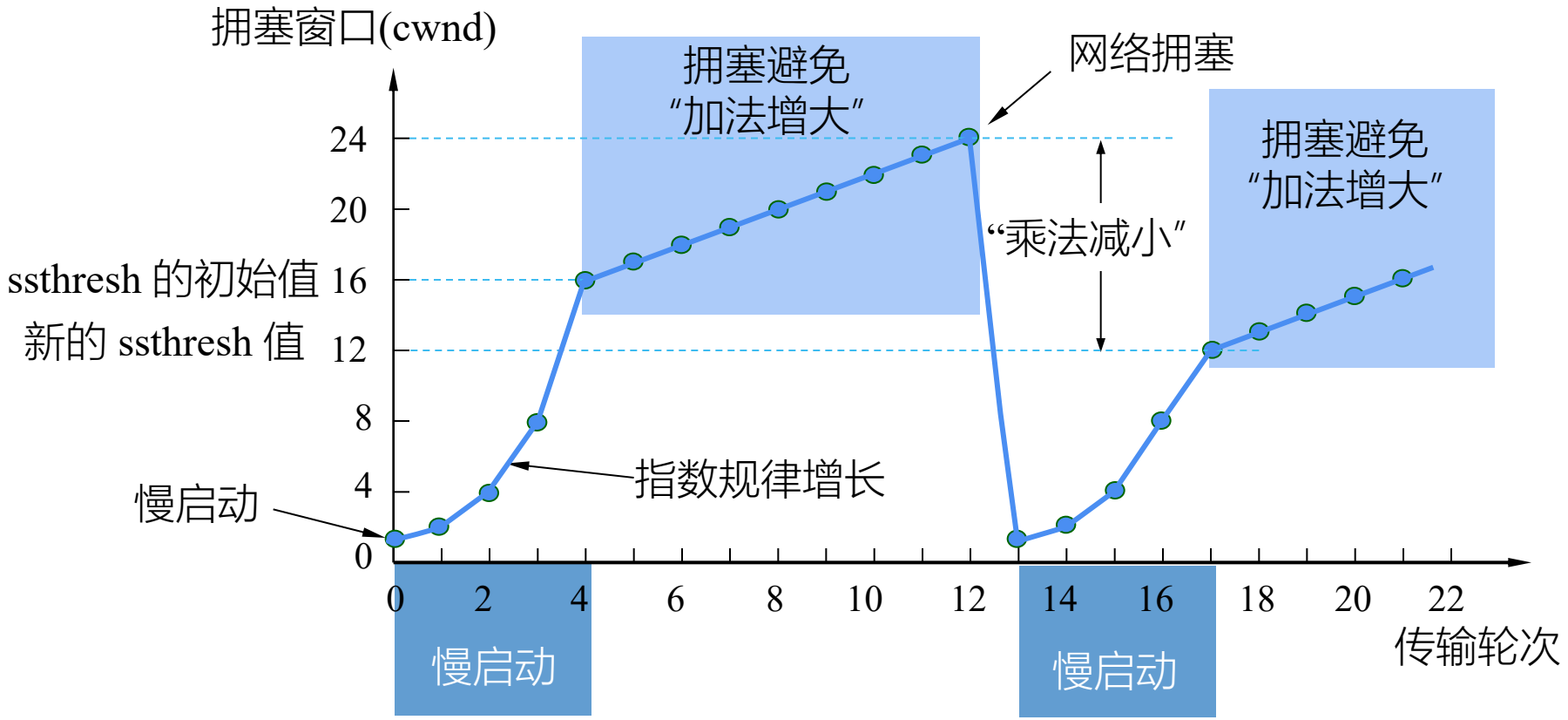
慢启动

- 在慢启动期间，发送方将初始的cwnd设置为1个SMSS字节
- 在第1个超时周期内没有丢失报文的情况下，cwnd设置为2个SMSS字节
- 在第2个超时周期内没有丢失报文的情况下，cwnd设置为4个SMSS字节
- 在第3个超时周期内没有丢失报文的情况下，cwnd设置为8个SMSS字节
- 依此方式，cwnd按指数方式增长，直到cwnd超过ssthresh

拥塞避免

- 当cwnd超过ssthresh或者当cwnd大小达到ssthresh的大小，进入拥塞避免期间
- 在拥塞避免期间，在没有丢失报文的情况下，cwnd按线性方式增长，即每收到一个ACK，cwnd的大小增加1个SMSS字节
- 当检测到数据段丢失时，则将ssthresh设置为当前cwnd的一半，并重新开始慢启动算法

慢启动和拥塞避免工作过程



乘法减小(multiplicative decrease)

- “乘法减小”是指不论在慢启动阶段还是拥塞避免阶段，只要出现一次超时（即出现一次网络拥塞），就把慢开始门限值 ssthresh 设置为当前的拥塞窗口值乘以1/2
- 当网络频繁出现拥塞时，ssthresh 值就下降得很快，以大大减少注入到网络中的分组数

加法增大(additive increase)

- “加法增大”是指执行拥塞避免算法后，在收到对所有报文段的确认后（即经过一个往返时间），就把拥塞窗口 cwnd增加一个 MSS 大小，使拥塞窗口缓慢增大，以防止网络过早出现拥塞

拥塞避免

- “拥塞避免”并非指完全能够避免了拥塞。利用以上的措施要完全避免网络拥塞还是不可能的
- “拥塞避免”在拥塞避免阶段把拥塞窗口控制为按线性规律增长，使网络比较不容易出现拥塞

例题

- 假定最大报文段长度是1KB，TCP拥塞窗口是16KB，并发生了超时事件。如果接着4个轮次传输都是成功的，那么该窗口将是多大？

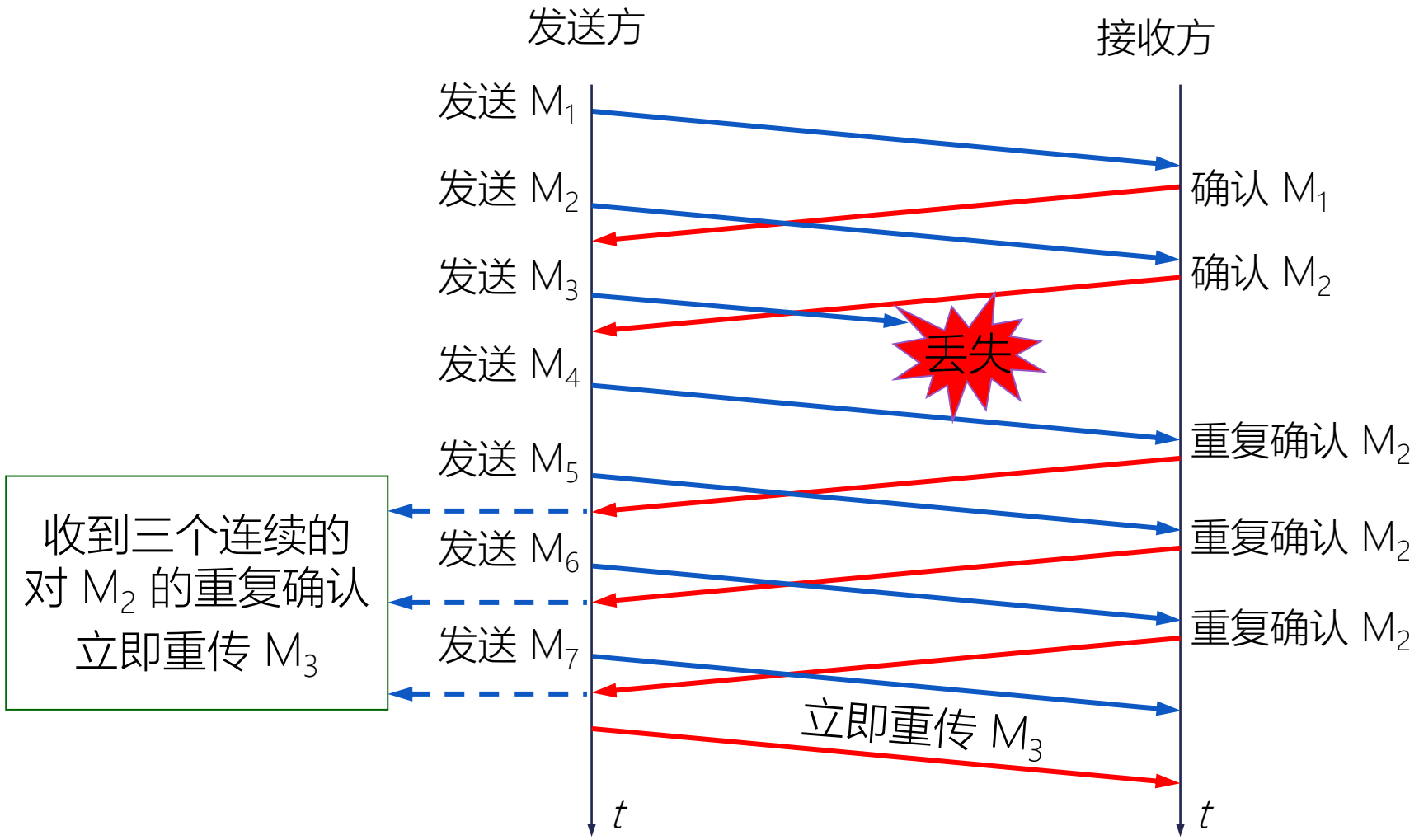
答：发生超时后，下一次传输的是1，接着是2、4、8个报文段。所以4个轮次后的拥塞窗口是9KB

如果接着6个轮次都成功，CWND的值应该是多大？

快速重传/快速恢复

- 快速重传算法首先要求接收方每收到一个失序的报文段后就立即发出重复确认。这样做可以让发送方及早知道有报文段没有到达接收方
- 发送方只要一连收到三个重复确认就应当立即重传对方尚未收到的报文段
- 不难看出，快速重传并非取消重传计时器，而是在某些情况下可更早地重传丢失的报文段

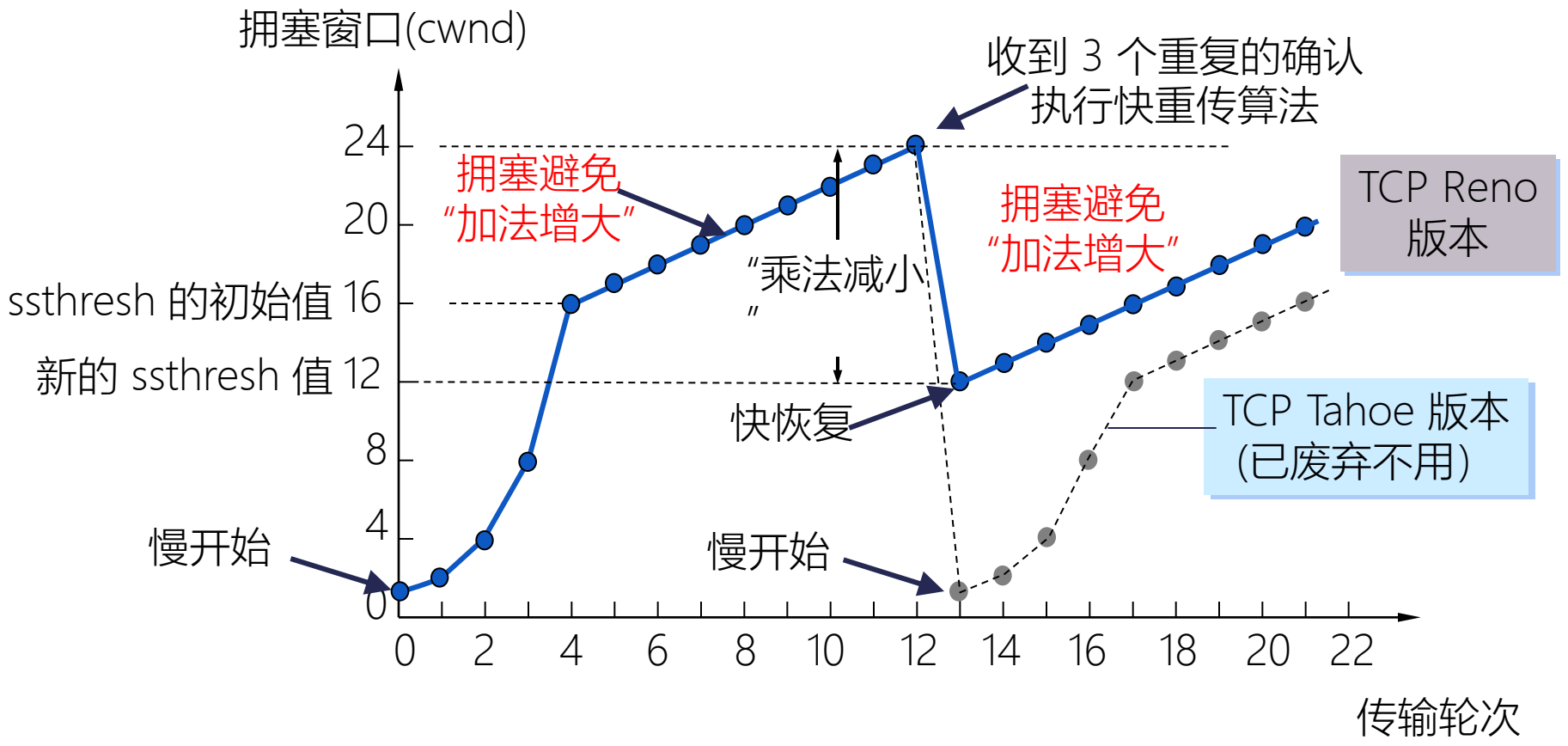
快速重传示例


发送方
接收方
发送 M_1
发送 M_2
发送 M_3
发送 M_4
发送 M_5
发送 M_6
发送 M_7
确认 M_1
确认 M_2
丢失
重复确认 M_2
重复确认 M_2
重复确认 M_2
收到三个连续的
对 M_2 的重复确认
立即重传 M_3
立即重传 M_3
t
t

快速恢复算法

- 当发送端收到连续三个重复的确认时，就执行“乘法减小”算法，把慢启动门限 ssthresh 减半。但接下去不执行慢启动算法
- 由于发送方现在认为网络很可能没有发生拥塞，因此现在不执行慢启动算法，即拥塞窗口 cwnd 现在不设置为 1，而是设置为慢启动门限 ssthresh 减半后的数值，然后开始执行拥塞避免算法（“加法增大”），使拥塞窗口缓慢地线性增大

连续收到三个重复的确认转入拥塞避免



发送窗口的上限值

- 发送方的发送窗口的上限值应当取为接收方窗口 rwnd 和拥塞窗口 cwnd 这两个变量中较小的一个，即应按以下公式确定：

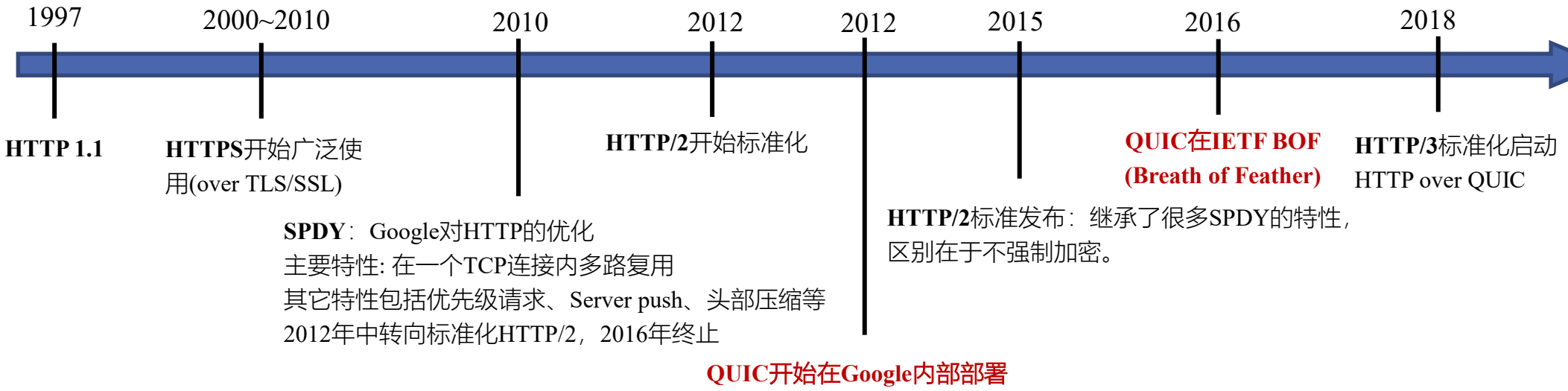
$$发送窗口的上限值=\min(rwnd,cwnd)$$

- 当 rwnd < cwnd 时，是接收方的接收能力限制发送窗口的最大值
- 当 cwnd < rwnd 时，则是网络的拥塞限制发送窗口的最大值

TCP选项

- kind=0，选项表结束（EOP）选项
- kind=1，空操作（NOP）选项
- kind=2，最大报文段长度（MSS）选项
- kind=3，窗口扩大因子选项
- kind=4，选择性确认（Selective Acknowledgment，SACK）选项
- kind=5，SACK实际工作的选项
- kind=8，时间戳选项

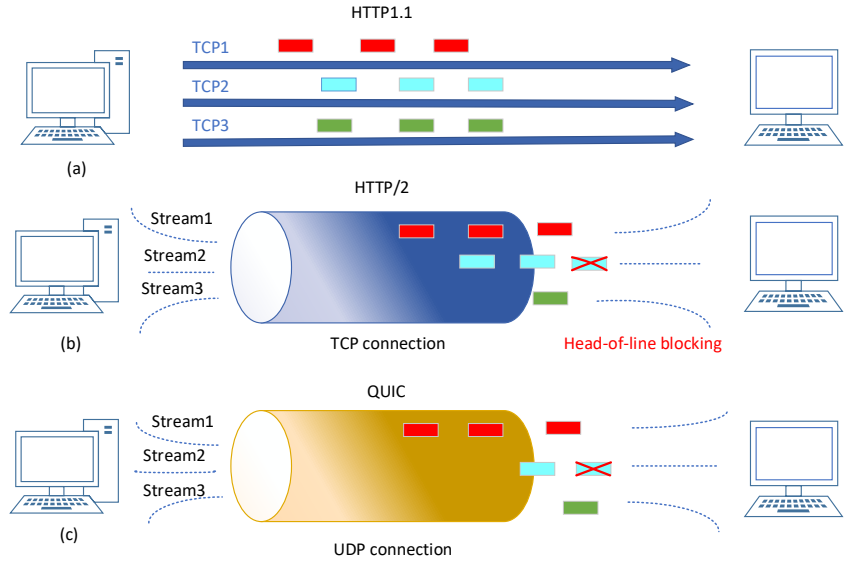
6.4 快速UDP网络连接

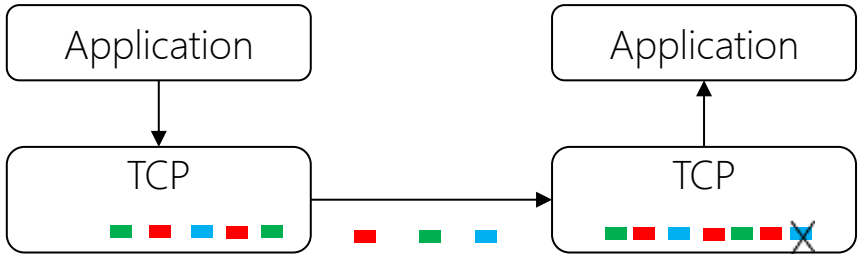


- QUIC：基于HTTP协议族而来，正转变为通用的传输层协议
  - Google研究QUIC，是为了解决HTTPS over TCP性能不佳的问题
  - 在协议栈中，主要替代HTTP2 + TCP + TLS的部分
  - 解决的核心问题：低时延建立连接、多路复用避免TCP头阻塞
  - 随着QUIC在IETF标准化的推动，当前的QUIC已经逐步变为通用的传输协议，不再局限于HTTP的场景

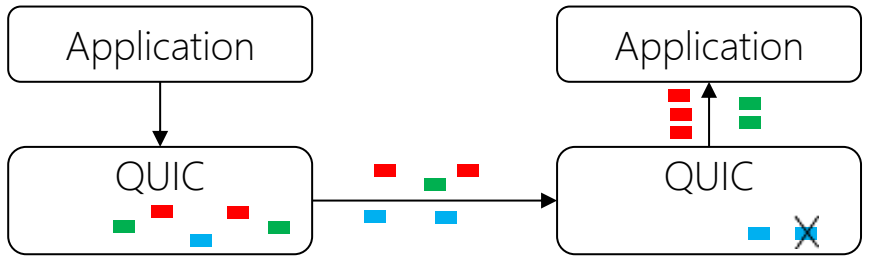
QUIC特性：多子流并发复用

- QUIC通过多子流复用一个连接，减少了建立连接的次数，并降低了服务器的开销
- HTTP的特性决定了业务(HTTP请求/响应)通常是并行的
  - 例如，打开一个网页，会同时加载很多元素
- HTTP/1.1中，每个HTTP消息使用独立的TCP连接，导致一个HTTP会话需建立大量的TCP连接，消耗服务器大量的连接资源；同时，TCP都需三次握手建立连接，增加时延
- HTTP/2中，多个HTTP消息并发复用一条TCP连接，一个HTTP会话仅需一条TCP连接。和HTTP/1.1相比，节省服务器资源、减少时延。但丢包时会面临TCP的头阻塞问题
- QUIC中，多个HTTP消息并发复用一条UDP连接。在接收端，基于子流Stream进行管理，避免头阻塞问题





TCP头阻塞：丢包后，后续报文都受影响



QUIC：仅丢包的子流需等待重传

QUIC报文格式

- QUIC包是由一个公共的首部后面跟着一个或多个帧组成的
- QUIC通过将流数据封装在一个或多个流帧中来实现复用
- 单个QUIC包可以携带来自多个流的帧
- QUIC有两种首部和多种帧类型

QUIC数据包格式

QUIC报文格式：首部

- QUIC的两种首部称为长首部和短首部
  - 长首部是在建立连接之前使用
  - 短首部是在建立第一个连接之后使用

长头

header form	fixed bit	long packet type	type specific bit	VID	DCID len	DCID	SCID len	SCID
1	1	2	4	32	8	0~60	8	0~160

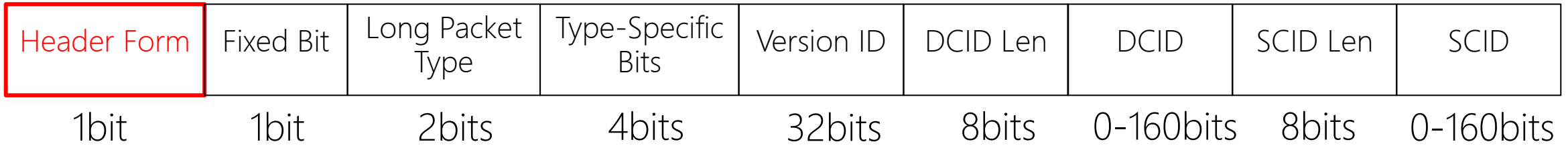
位

短头

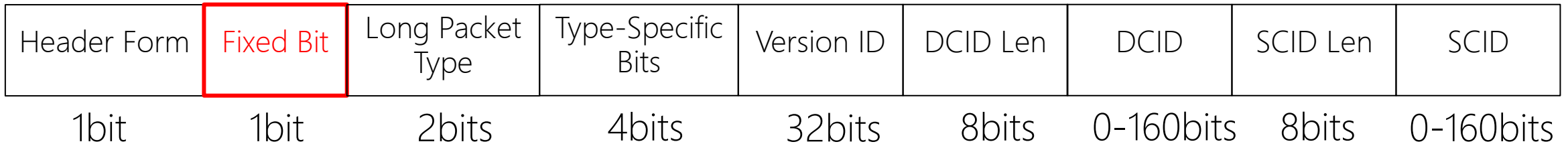
header form	fixed bit	spin bit	reserved	key phase	P	DCID	PKN	protected payload
1	1	1	2	1	2	160	P+8	

位

QUIC报文格式：长首部

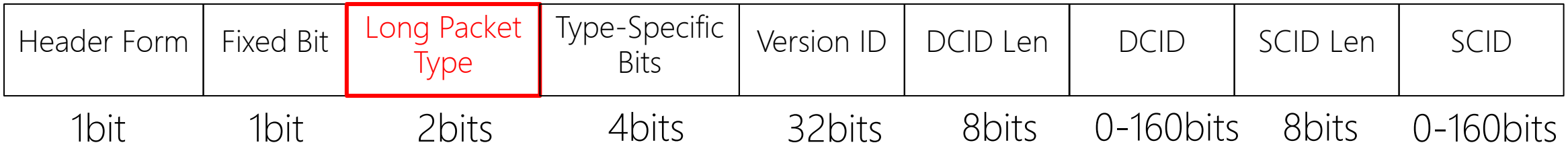


Header Form (HF)：标识头类型（1）



Fixed Bit (FB)：指示数据包是否有效。 如果设置为0，则数据包无效

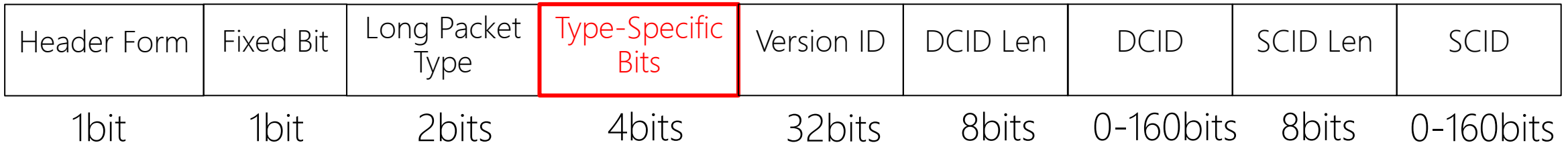
QUIC报文格式：长首部



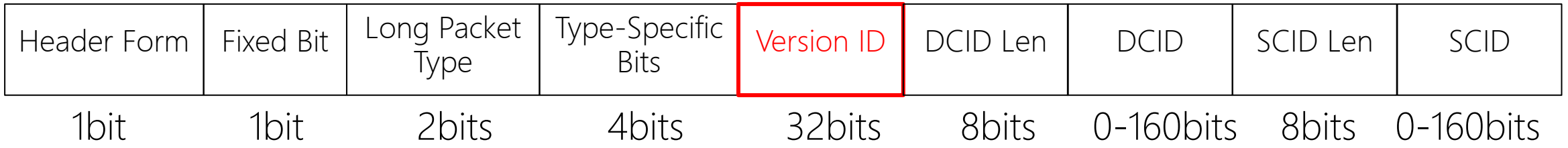
Long Packet Type (T)：指示长头包的类型

1. 0x00: Initial Packet
2. 0x01: 0-RTT
3. 0x02: handshake Packet
4. 0x03: Retry Packet

QUIC报文格式：长首部

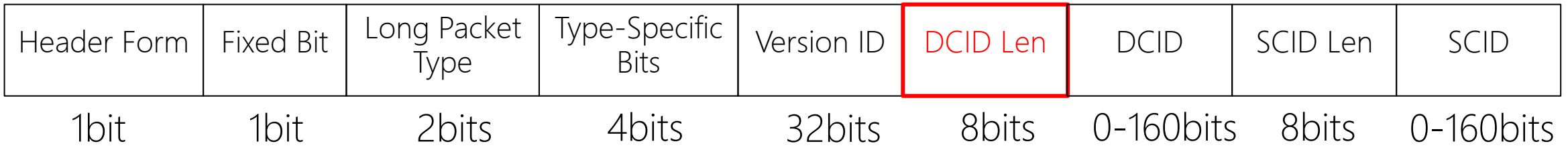


Type-Specific Bits (S)： 指定长头包类型的位

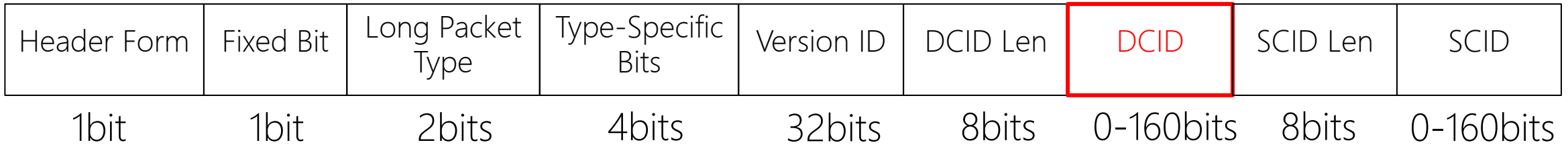


Version ID (VID)： 用于识别QUIC的版本

QUIC报文格式：长首部

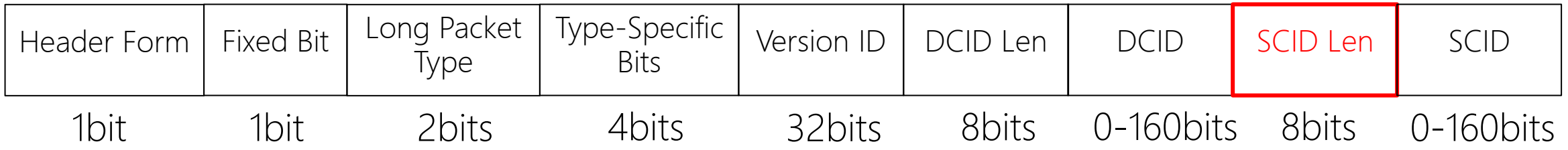


Destination Connection ID Length (DCID Len)： 目的连接ID长度

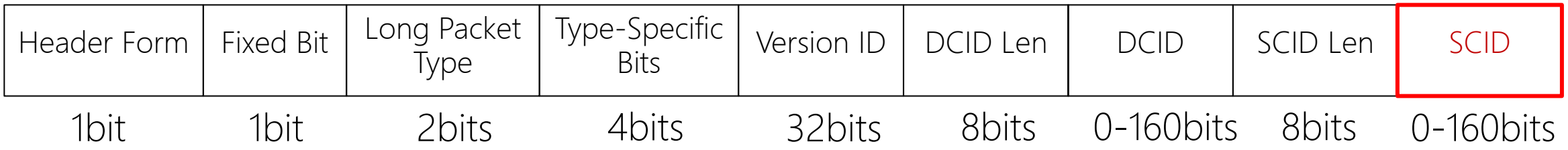


Destination Connection ID (DCID)： 目的连接ID

QUIC报文格式：长首部

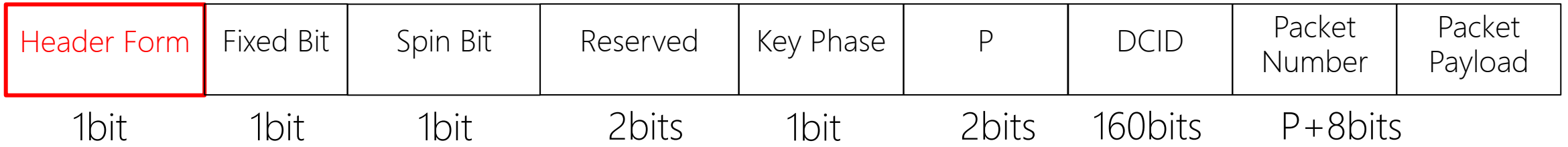


Source Connection ID Length (SCID Len)：源连接ID长度

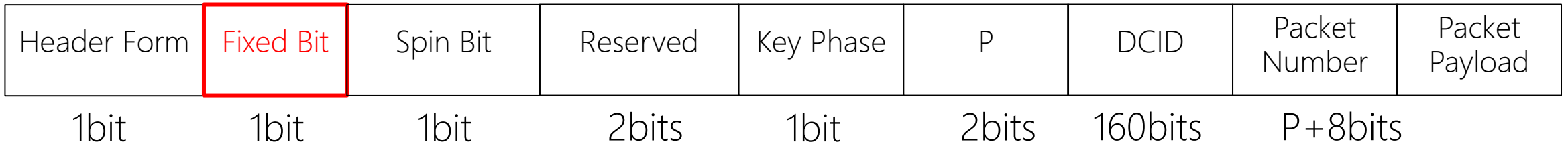


Source Connection ID (SCID)：源连接ID

QUIC报文格式：短首部

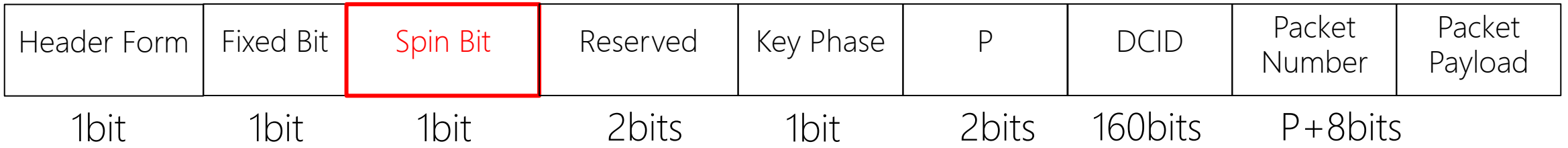


Header Form (HF)：标识头类型（0）

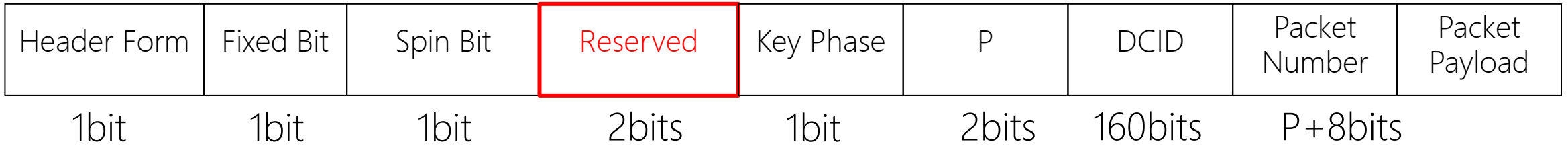


Fixed Bit (FB)：指示数据包是否有效。 如果设置为0，则数据包无效

QUIC报文格式：短首部

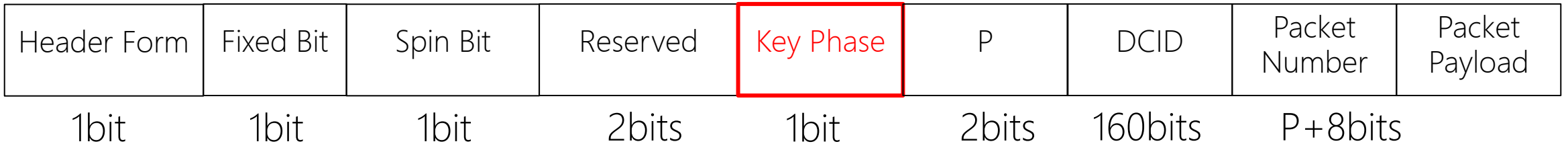


Spin Bit：时延旋转比特位

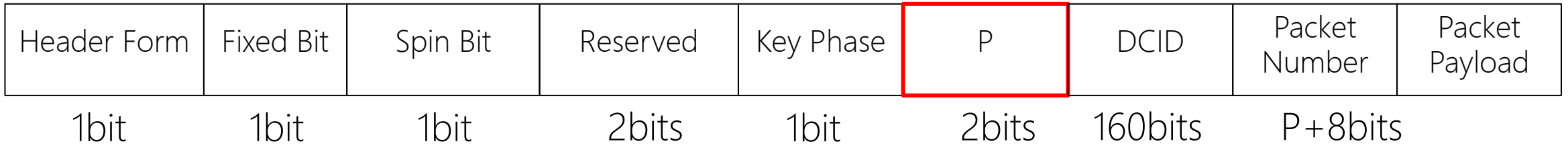


Reserved Bits：保留位

QUIC报文格式：短首部

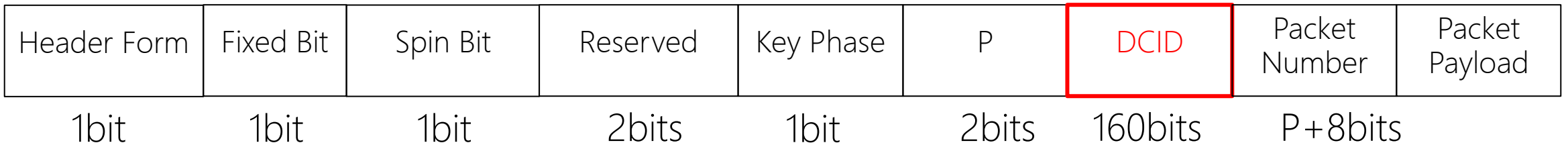


Key Phase： 标识数据包保护密钥

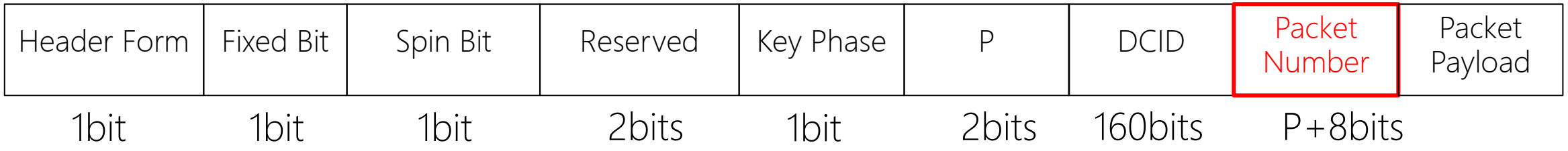


Packet Number Length (P)： 包数量长度

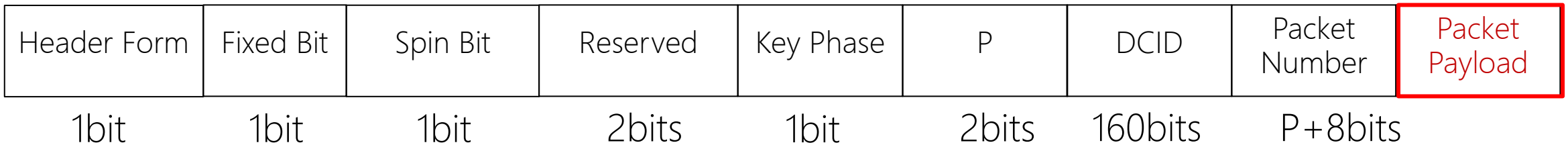
QUIC报文格式：短首部



Destination Connection ID (DCID)：目的连接ID

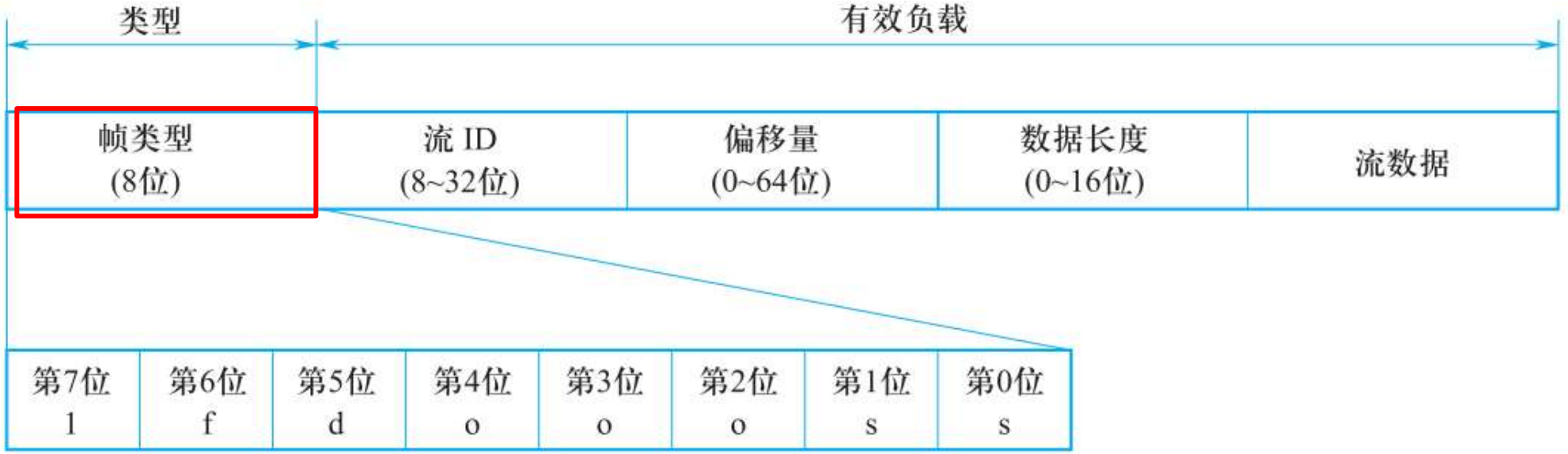


Packet Number：包数量



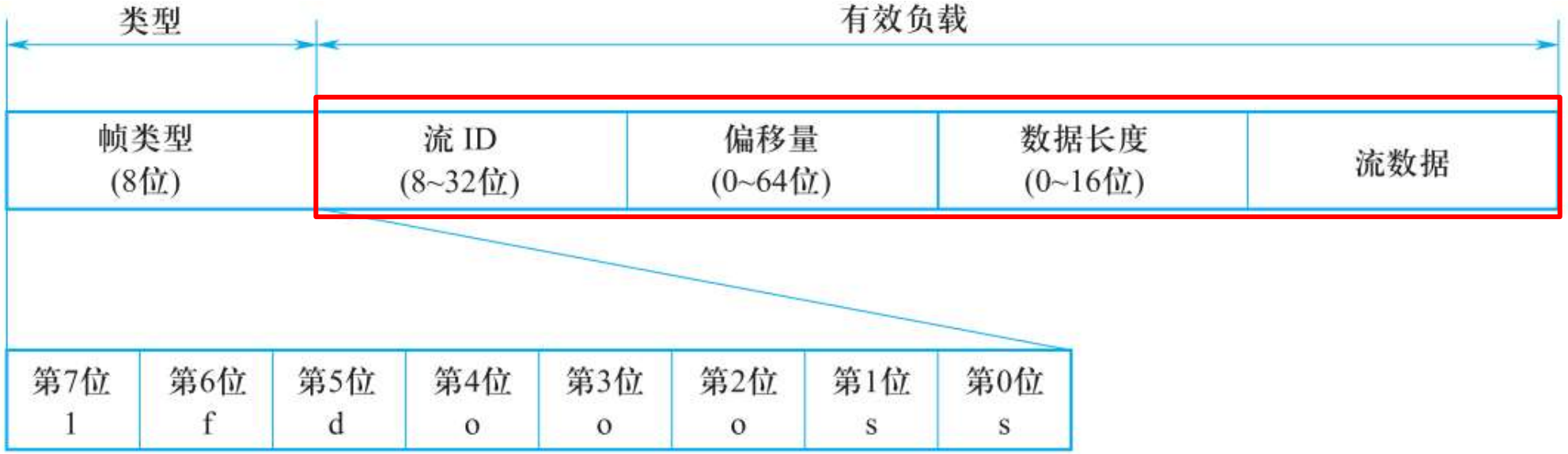
Packet Payload：包载荷

QUIC报文格式：QUIC帧类型



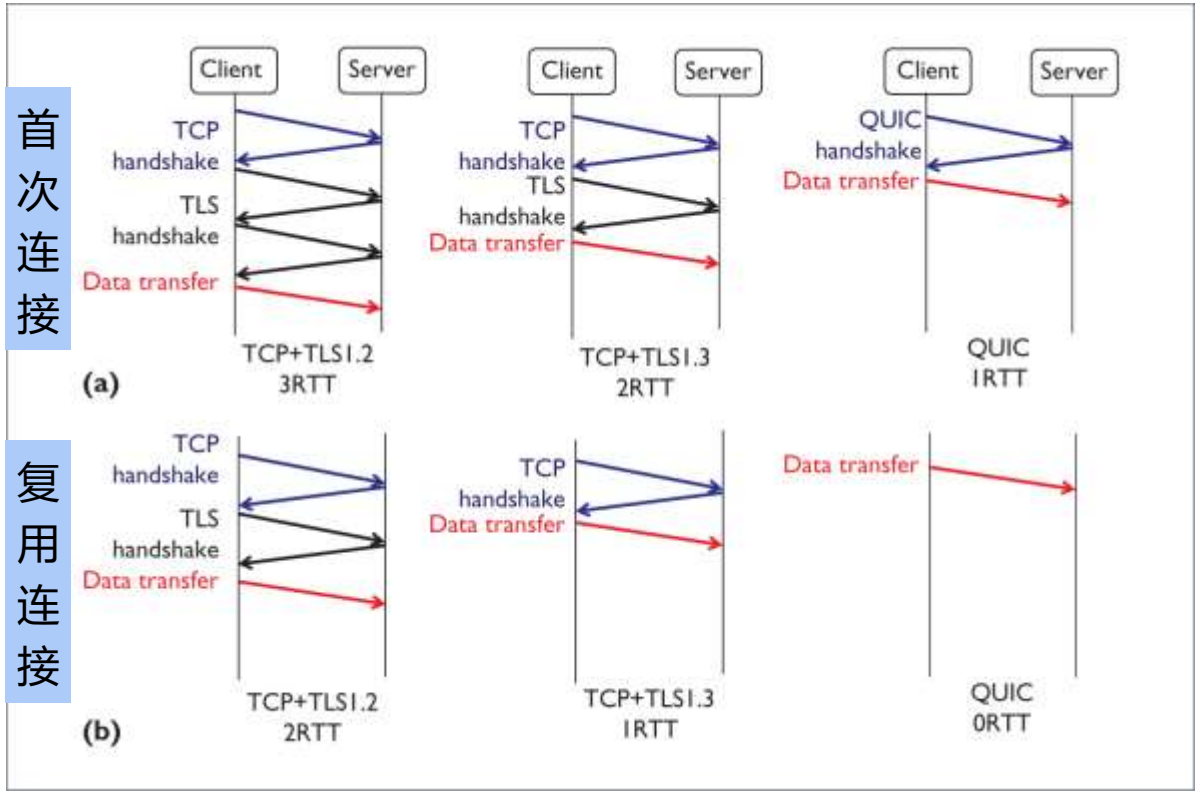
- Frame Type：1个字节，表示帧类型，包含可变的标志，其表现形式为1fdoooss。
  - 1: 1bit，必须设置为1，标志这是一个流类型帧
  - f: 1bit，Fin bit。如果设置为1，标识发送端完成了这条流上的发送并且希望进入半关闭状态
  - d: 1bit，标识当前流的首部中是否包含数据长度
    - 如果设置为0，标识无数据长度字段，这个流类型帧的大小为包的大小
    - 如果设置为1，标识有数据长度字段
  - o: 3bit，编码了首部中偏移量的长度，比如0，16，24，32，40，48，56，64（单位：bit）
  - s: 2bit，编码了首部中流ID的长度，比如8，16，24，32（单位：bit）

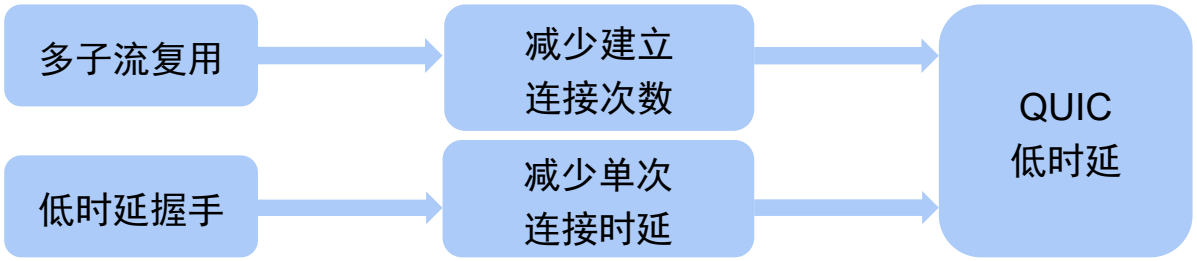
QUIC报文格式：QUIC帧类型



- Stream ID：用于标识当前数据流属于哪个资源请求
- Offset：标识当前数据包在当前Stream ID 中的字节偏移量
- Length：数据长度，表示实际应用数据的长度
- Stream Data：实际的应用数据

QUIC特性：低时延建立连接





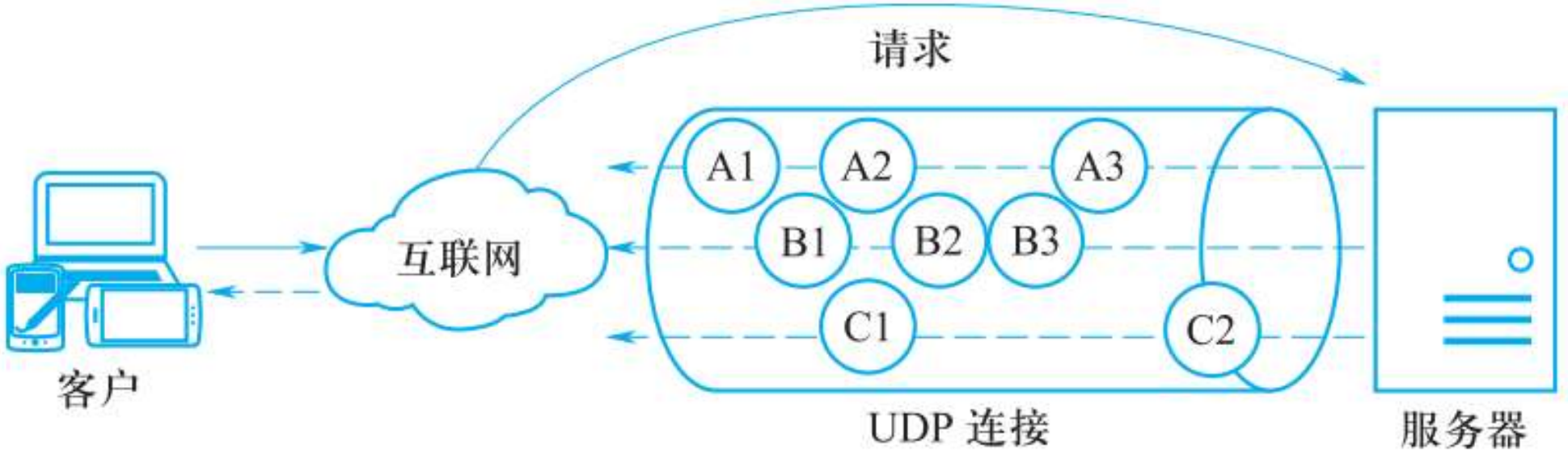
- QUIC通过优化握手过程，在建立加密连接时可以做到1RTT或0RTT，降低了握手时延
- 首次连接
  - QUIC将传输层握手和加密握手合二为一，在首次建立连接时仅需1RTT；
  - TCP/TLS的传输层握手和加密握手分开，根据TLS版本不同，分别需要2-3个RTT；
- 复用连接
  - QUIC连接建立后在两端缓存连接参数、密钥等，后续建立连接不需协商，0RTT即可发送数据
  - TCP/TLS后续连接中，加密握手次数可减少，但TCP握手不可省略

QUIC连接转移

- 连接有连接ID标识，不以4元组标识。例如
  - 客户端先使用 IP1 发送了 1 和 2 数据包
  - IP 变更为 IP2，发送了 3 和 4 数据包
  - 服务器根据数据包首部的 SCID 字段可以判断这 4 个包是来自于同一个客户端
- QUIC 能实现连接迁移的根本原因是底层使用 UDP 协议就是无连接的

QUIC无队头阻塞多路复用

- QUIC设计了连接（Connection）和流（Stream）的概念
  - 一个连接可以复用传输多个流，每个流之间都是独立的
  - 一个stream丢包并不会影响到其它Stream的处理
  - 仍然存在单条流上的队头阻塞

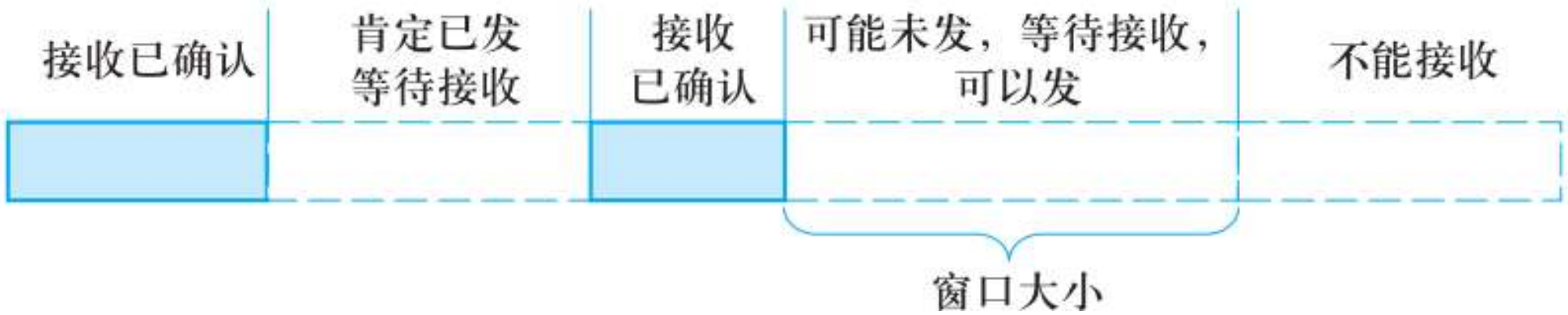


拥塞控制与流量控制

- QUIC 的拥塞控制不再依赖内核的拥塞控制算法，而是在应用层上实现
- QUIC 采用了一种基于限制（limit-based）的流量控制方案
  - 接收方通告了对在流（stream）和整个连接（connection）上准备接收的总字节数的限制
  - 发送方发送的数据不能超过这两个限制

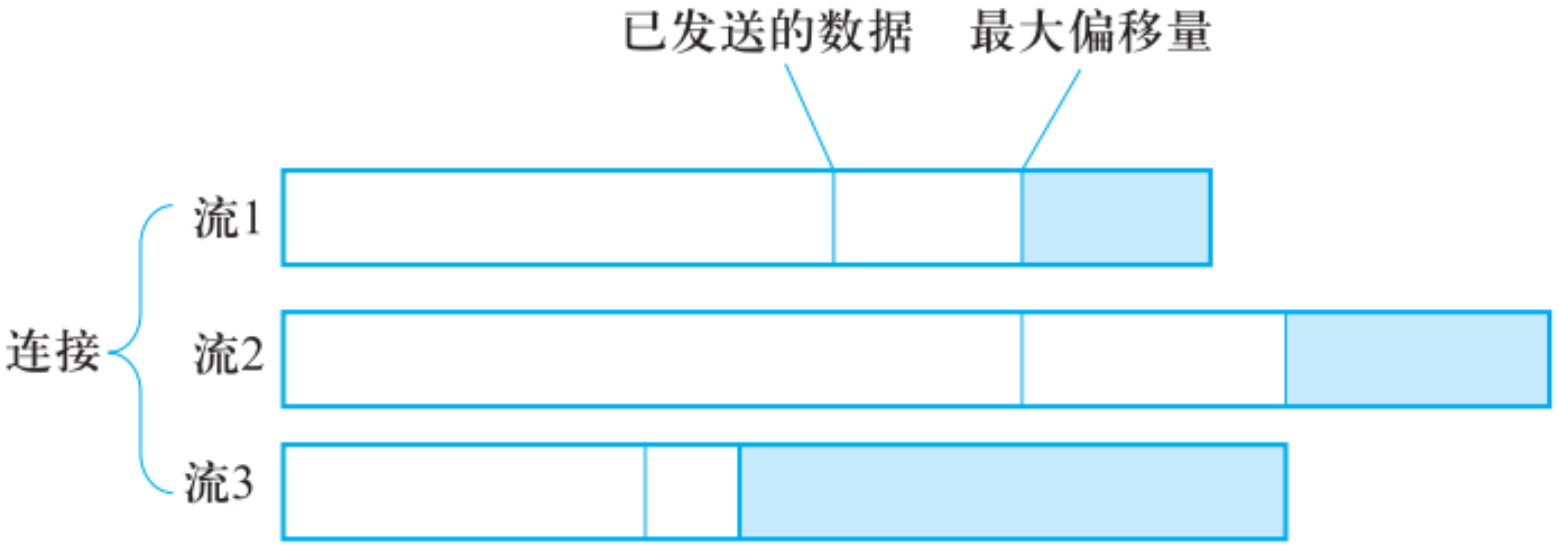
QUIC控制——流控制

- 通过限制流（Stream）发送的最大绝对字节偏移量，防止单个流消耗连接（Connection）的全部接收缓冲
- Stream流基于Stream ID+Offset进行包确认，流量控制需要保证所发送的所有包offset小于最大绝对字节偏移量
- QUIC利用ACK Frame来进行数据包的确认，来保证可靠传输



QUIC控制——连接控制

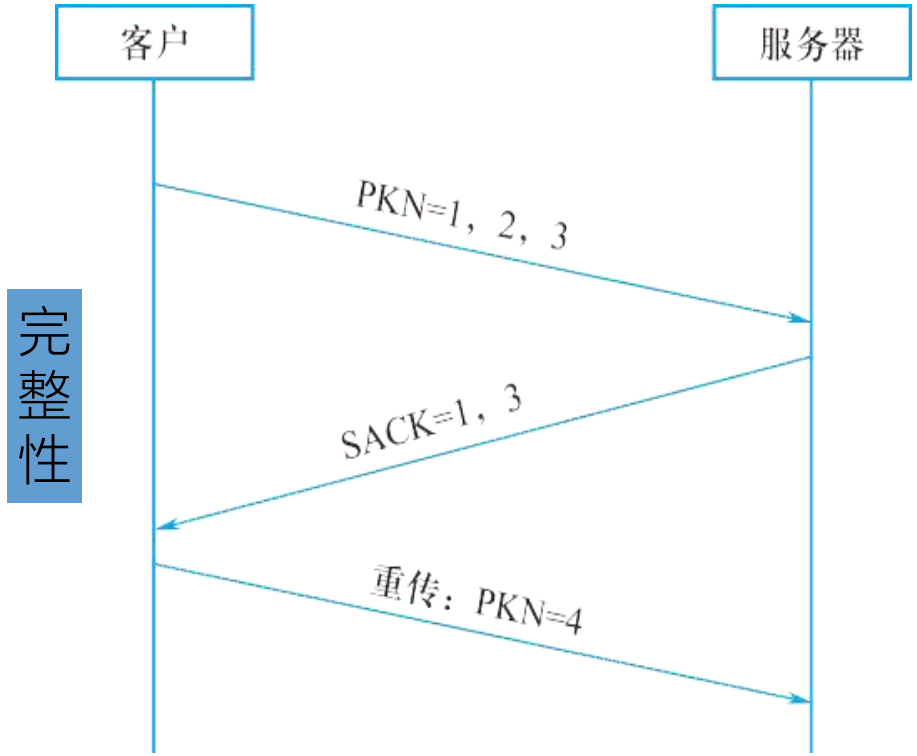
- 连接控制通过限制所有STREAM帧的数据总字节数，防止发送方超过接收方的连接缓冲容量
- 连接控制具有总的缓冲区大小限制，为每个Stream动态分配缓冲区大小，优先向速度更快的stream倾斜



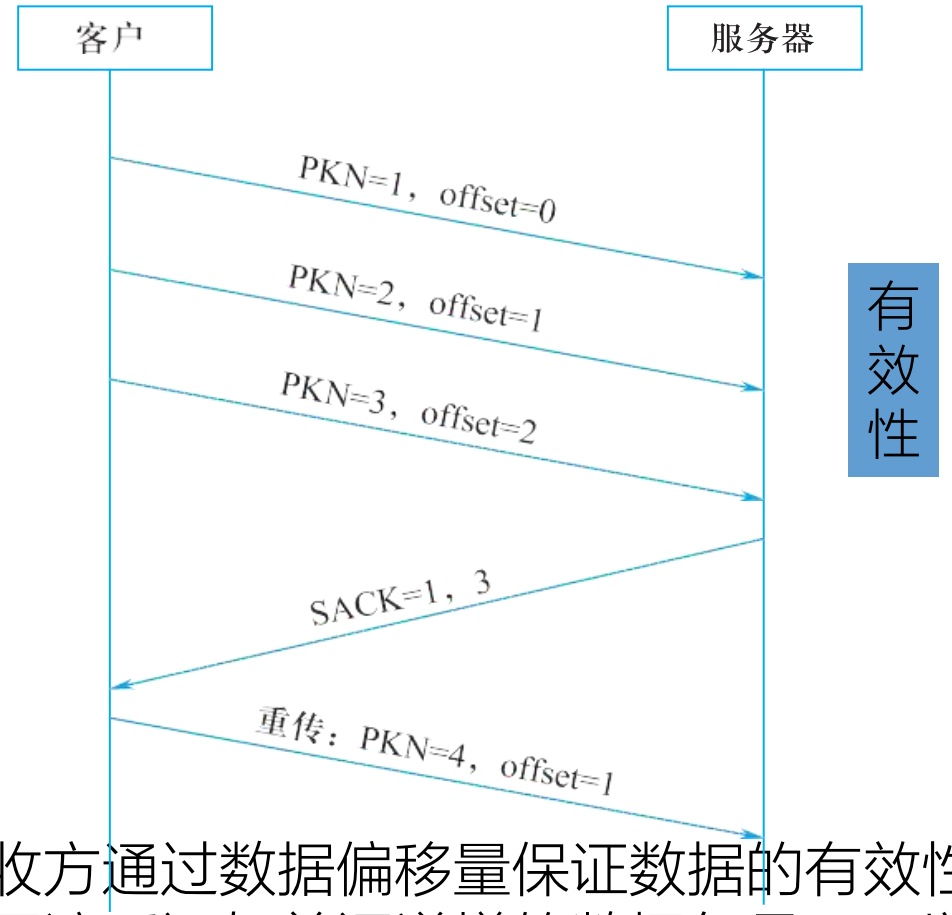
QUIC可靠传输

- 每一个流都有一个唯一标识——流 ID
- 流 ID 由客户 - 服务器进行静态分配，避免冲突
  - 客户主动发起的流的 ID 都是奇数
  - 服务器发起的流的 ID 都是偶数
- QUIC 数据包的大小，需要满足MTU限制，以避免被分段
  - 在 IPv6 下的最大长度为 1350 byte
  - 在 IPv4 下的最大长度为 1370 byte
- QUIC利用ACK Frame来进行数据包的确认，来保证可靠传输。两个重要特点
  - 完整性
  - 可靠性（有效性）

QUIC可靠传输—完整性、有效性



- 通过包号(PKN)和选择性确认(SACK)保证发送的数据都能被收到
- 数据包号单调递增数据包，数据一样，但数据包号不同

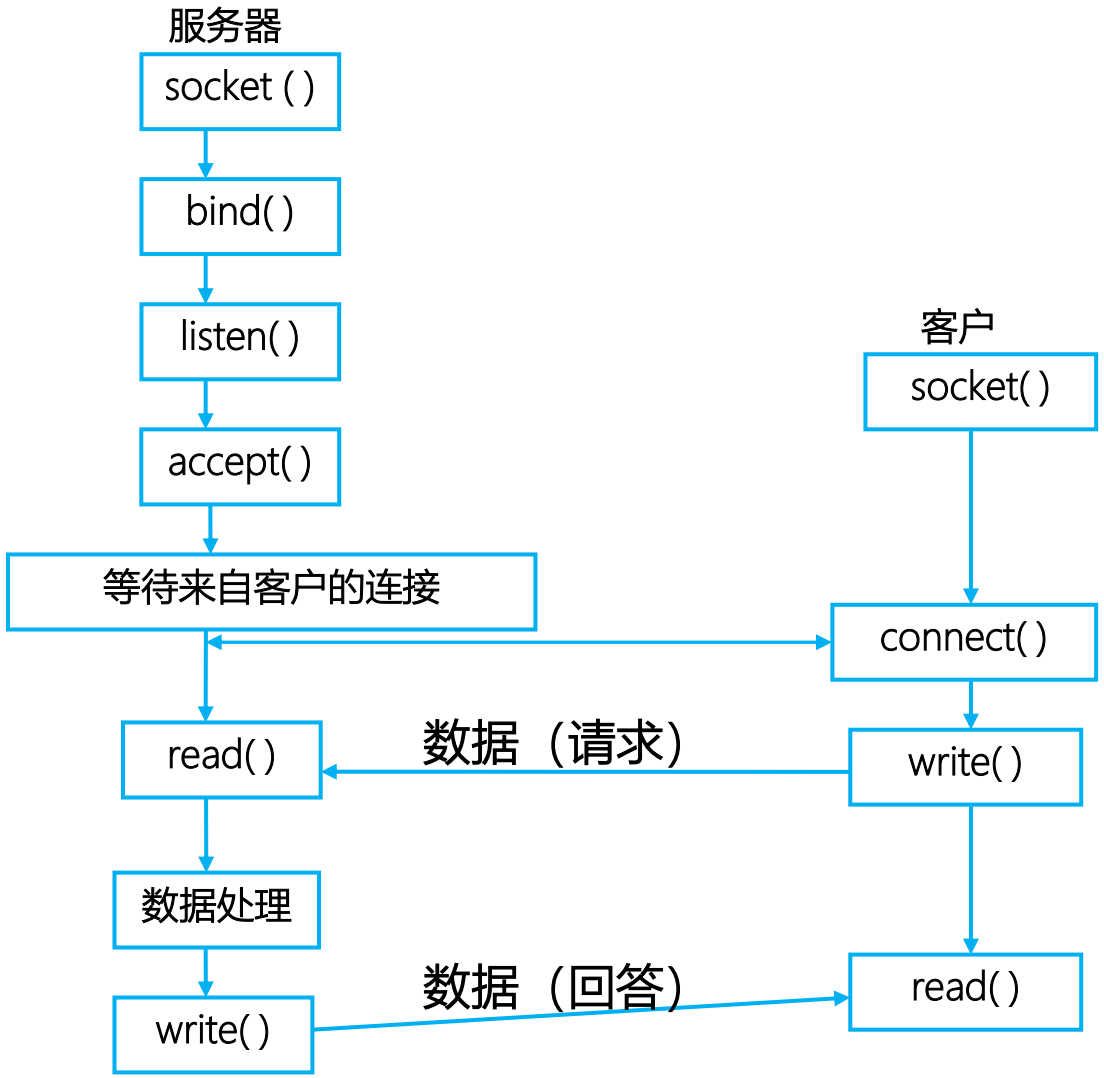


- 接收方通过数据偏移量保证数据的有效性
- 利用该延迟与单调递增的数据包号，可以精确地估算往返时间（RTT）有助于丢失检测，可以解决 TCP 的重传歧义问题

6.5socket程序设计

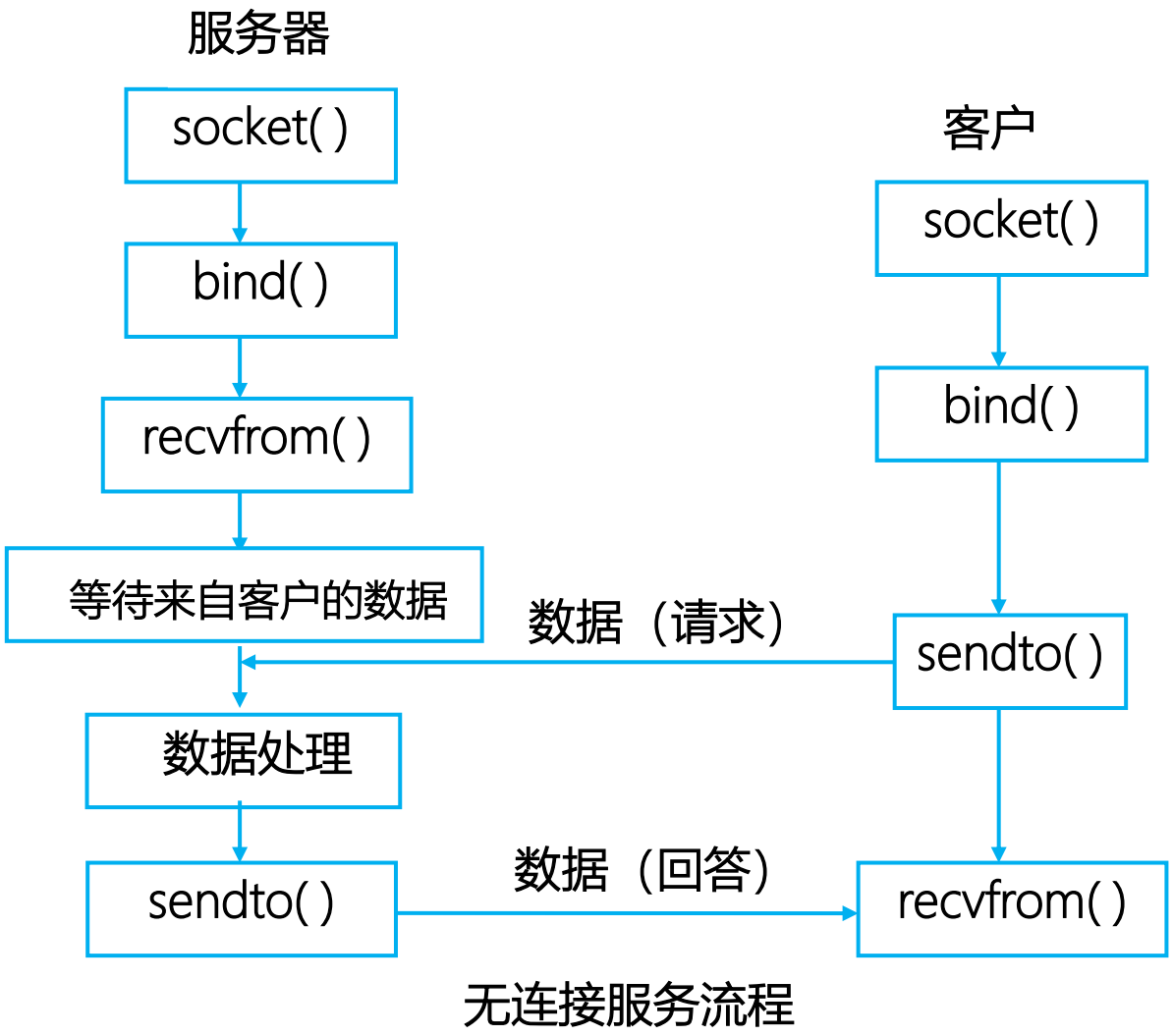
- socket 提供了简单的编程接口；
- 传输层提供两种服务
  - 面向连接的模型使用传输控制协议（TCP）
  - 无连接的模型使用用户数据报协议（UDP）
- 两种不同类型的服务具有不同类型的 socket

面向连接编程模型



面向连接服务流程

无连接编程模型



无连接服务流程

6.6 本章总结

- 传输层介于通信子网和资源子网之间
- 两个主要的传输协议UDP和TCP。UDP是一个无连接的协议，TCP协议提供了一个可靠的、双向的、拥塞可控的字节流服务
- TCP拥塞控制遵循AIMD控制法则，AIMD控制法则能将传输速率收敛到一个相对公平和有效的分配方案上
- 慢启动、拥塞避免、快速重传和快速恢复四个算法构成了TCP拥塞控制的基础
- QUIC运行在UDP之上，它的主要目标是让HTTP等应用层协议运行得更加快速