

# Chapter 5 Transport layer

- [1. 运输层概述](#)

- [1.1. 进程间基于网络的通信](#)
- [1.2. TCP/IP运输层中的两个重要协议](#)
- [1.3. 运输层端口号](#)
- [1.4. 发送方的复用和接收方的分用](#)
  - [1.4.1. TCP/IP体系结构应用层常用协议所使用的运输层协议和熟知端口号](#)
- [1.5. 运输层端口号应用举例](#)

- [2. UDP和TCP的对比](#)

- [2.1. 无连接的UDP和面向连接的TCP](#)
- [2.2. 对单播、多播和广播的支持情况](#)
- [2.3. 对应用层报文的处理](#)
- [2.4. 对数据传输可靠性的支持情况](#)
- [2.5. 首部对比](#)

- [3. 传输控制协议](#)

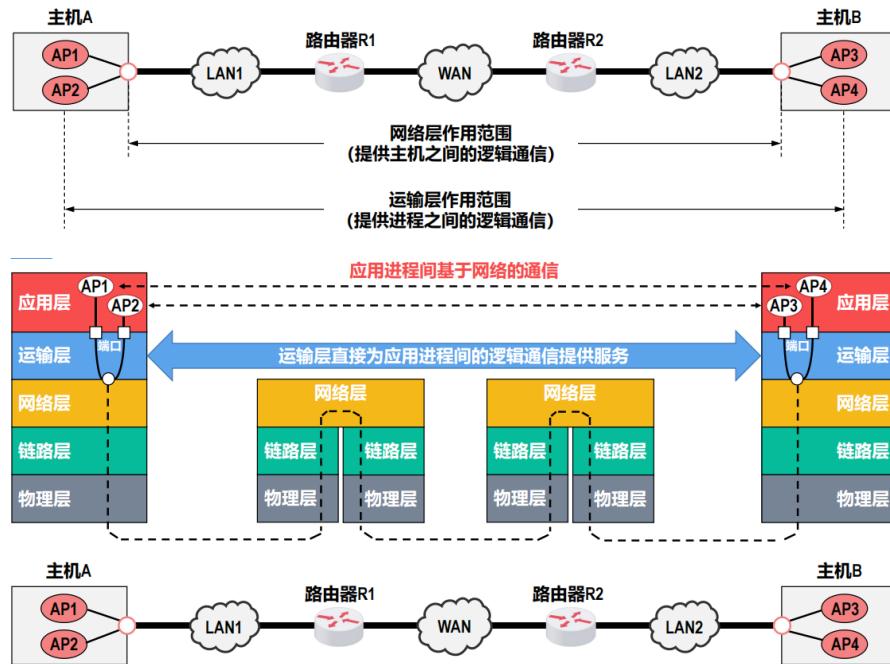
- [3.1. TCP报文段的首部格式](#)
  - [3.1.1. 序号、确认号、确认标志位ACK](#)
  - [3.1.2. 数据偏移](#)
  - [3.1.3. 保留](#)
  - [3.1.4. 窗口](#)
  - [3.1.5. 检验和](#)
  - [3.1.6. 同步标志位SYN](#)
  - [3.1.7. 终止标志位FIN](#)
  - [3.1.8. 复位标志位RST](#)
  - [3.1.9. 推送标志位PSH](#)
  - [3.1.10. 紧急标志位URG、紧急指针](#)
  - [3.1.11. 选项（长度可变、最大40字节）](#)
  - [3.1.12. 填充](#)
- [3.2. TCP的运输层连接管理](#)
  - [3.2.1. “三报文握手”建立连接](#)
  - [3.2.2. “四报文挥手”释放连接](#)
- [3.3. TCP的流量控制](#)
  - [3.3.1. 基本概念](#)
  - [3.3.2. 控制方法](#)
- [3.4. TCP的拥塞控制](#)
  - [3.4.1. 基本概念](#)
  - [3.4.2. 基本方法](#)
    - [3.4.2.1. 对比流量控制](#)
    - [3.4.2.2. 开环控制与闭环控制](#)
    - [3.4.2.3. 衡量网络拥塞的指标](#)
    - [3.4.2.4. 闭环拥塞控制算法](#)

- [3.4.2.5. 代价](#)
- [3.4.3. TCP的四种拥塞控制方法](#)
  - [3.4.3.1. 慢开始算法和拥塞避免算法](#)
  - [3.4.3.2. 快重传算法和快恢复算法（改进TCP性能，1990年Reno版本）](#)
  - [3.4.3.3. 四种控制方法拥塞窗口变化图](#)
  - [3.4.3.4. TCP拥塞控制流程图](#)
- [3.4.4. TCP拥塞控制与网际层拥塞控制的关系](#)
- [3.5. TCP可靠传输的实现](#)
- [3.6. TCP超时重传时间的选择](#)
- [3.7. TCP的选择确认](#)
- [4. 题目](#)
  - [4.1. TCP序号、确认号](#)
    - [4.1.1. 【2009 38】](#)
    - [4.1.2. 【2013 39】](#)
    - [4.1.3. 【2011 40】](#)
  - [4.2. 建立TCP连接](#)
    - [4.2.1. 【2011 39】](#)
    - [4.2.2. 【2019 39】](#)
  - [4.3. 释放TCP连接](#)
    - [4.3.1. 【2020 39】](#)
  - [4.4. TCP流量控制](#)
    - [4.4.1. 【2010 39】](#)
  - [4.5. TCP拥塞控制](#)
    - [4.5.1. 【2009 39】](#)
    - [4.5.2. 【2014 38】](#)
    - [4.5.3. 【2015 39】](#)
    - [4.5.4. 【2017 39】](#)
    - [4.5.5. 【2019 38】](#)
    - [4.5.6. 【2020 38】](#)
  - [4.6. TCP超时重传的时间选择](#)

## 1. 运输层概述

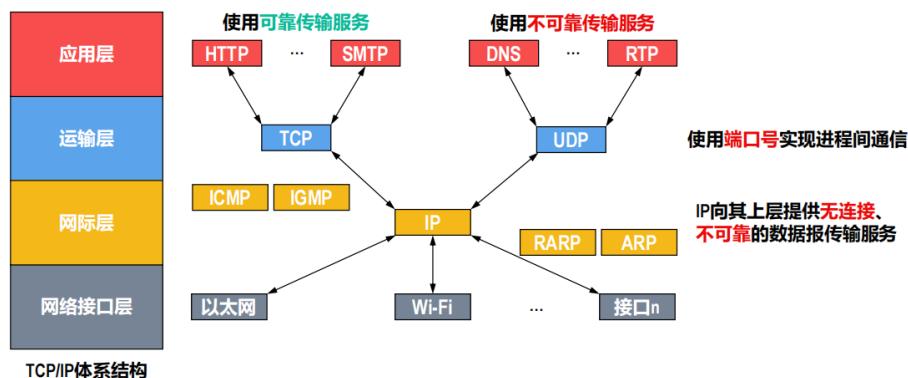
### 1.1. 进程间基于网络的通信

- 计算机网络体系结构中的物理层、数据链路层和网络层，它们共同解决了将主机通过异构网络互联起来所面临的问题，实现了主机到主机的通信。
- 计算机网络中实际进行通信的真正实体，是位于通信两端主机中的进程。
- 运输层的主要任务：为运行在不同主机上的应用进程提供直接的逻辑通信服务
- 运输层协议又称为端到端协议。



- 运输层向应用层实体屏蔽了下面网络核心的细节（例如网络拓扑、所采用的路由选择协议等），它使应用进程看见的好像是在两个运输层实体之间有一条端到端的逻辑通信信道。
- 根据应用需求的不同，因特网的运输层为应用层提供了两种不同的运输层协议，即面向连接的TCP和无连接的UDP，这两种协议就是本章要讨论的主要内容。

## 1.2. TCP/IP运输层中的两个重要协议



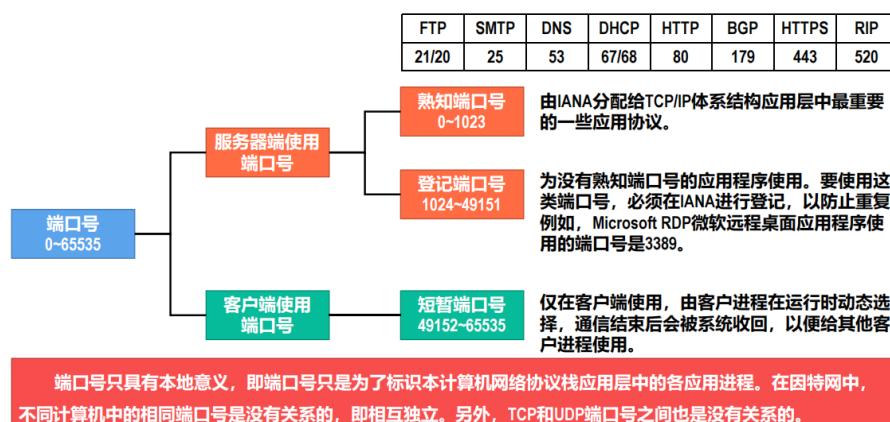
| TCP                                                                                                                                                                                                                                                                                                                                                              | UDP                                                                                                                                                                                                                                                                                      |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>● <b>传输控制协议 (Transmission Control Protocol, TCP)</b> 为其上层提供的是<b>面向连接的可靠</b>的数据传输服务。</li> <li>● 使用TCP通信的双方，在传送数据之前必须首先建立<b>TCP连接</b>（逻辑连接，而非物理连接）。数据传输结束后<b>必须要释放TCP连接</b>。</li> <li>● TCP为了实现可靠传输，就必须使用很多措施，例如<b>TCP连接管理、确认机制、超时重传、流量控制以及拥塞控制</b>等。</li> <li>● TCP的<b>实现复杂</b>，TCP报文段的<b>首部比较大，占用处理机资源比较多</b>。</li> </ul> | <ul style="list-style-type: none"> <li>● <b>用户数据报协议 (User Datagram Protocol, UDP)</b> 为其上层提供的是<b>无连接的不可靠</b>的数据传输服务。</li> <li>● 使用UDP通信的双方，在传送数据之前<b>不需要建立连接</b>。</li> <li>● UDP<b>不需要实现可靠传输</b>，因此<b>不需要使用实现可靠传输的各种机制</b>。</li> <li>● UDP的<b>实现简单</b>，UDP用户数据报的<b>首部比较小</b>。</li> </ul> |

因特网中的一些典型应用所使用的TCP/IP应用层协议和相应的运输层协议

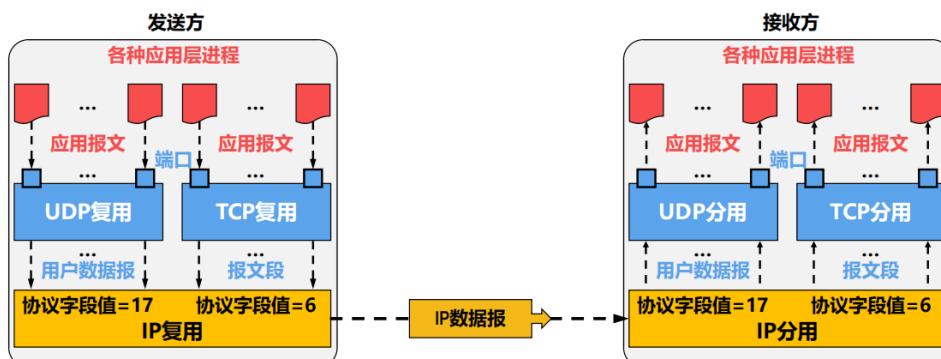
| 因特网应用   | TCP/IP应用层协议  | TCP/IP运输层协议 |
|---------|--------------|-------------|
| 域名解析    | 域名系统DNS      | UDP         |
| 文件传送    | 简单文件传送协议TFTP | UDP         |
| 路由选择    | 路由信息协议RIP    | UDP         |
| 网络参数配置  | 动态主机配置协议DHCP | UDP         |
| 网络管理    | 简单网络管理协议SNMP | UDP         |
| 远程文件服务器 | 网络文件系统NFS    | UDP         |
| IP电话    | 专用协议         | UDP         |
| 流媒体通信   | 专用协议         | UDP         |
| IP多播    | 网际组管理协议IGMP  | UDP         |
| 电子邮件    | 简单邮件传送协议SMTP | TCP         |
| 远程终端接入  | 电传机网络TELNET  | TCP         |
| 万维网     | 超文本传送协议HTTP  | TCP         |
| 文件传送    | 文件传送协议FTP    | TCP         |

### 1.3. 运输层端口号

- 运行在计算机上的进程是使用进程标识符（Process Identification, PID）来标识的。
  - 不同操作系统（Windows、Linux、MacOS）又使用不同格式的进程标识符。
  - 为了使运行不同操作系统的计算机的应用进程之间能够基于网络进行通信，就必须使用统一的方法对TCP/IP体系的应用进程进行标识。
- TCP/IP体系结构的运输层使用端口号来标识和区分应用层的不同应用进程。
- 端口号的长度为16比特，取值范围是0~65535。

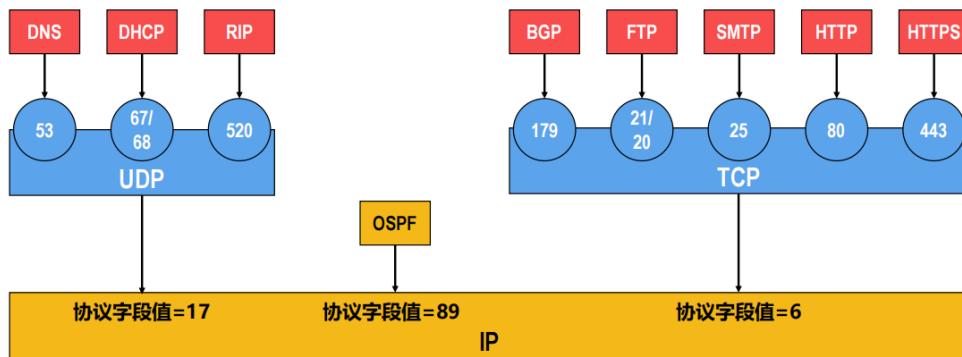


### 1.4. 发送方的复用和接收方的分用

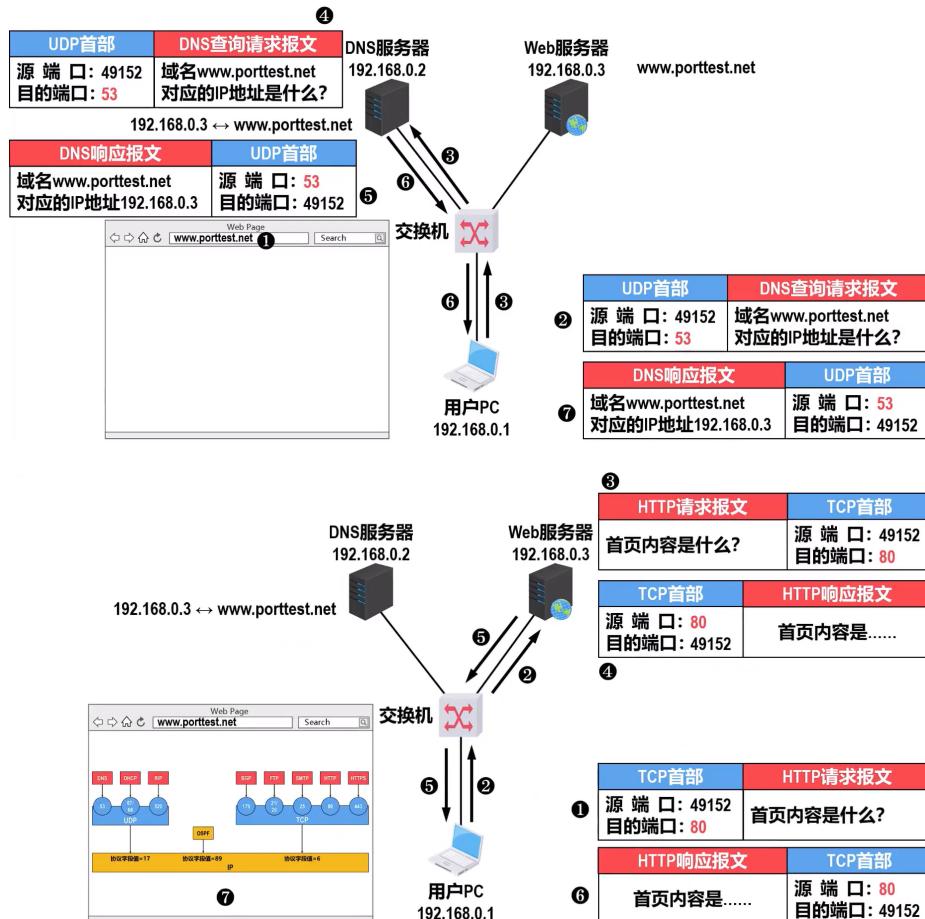


#### 1.4.1. TCP/IP体系结构应用层常用协议所使用的运输层协议和熟知端口号

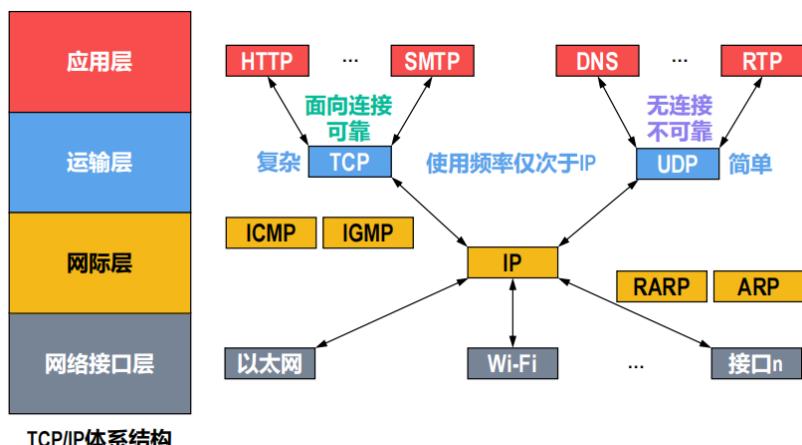
- OSPF报文并不使用运输层的UDP或TCP进行封装，而是直接使用网际层的IP进行封装。



## 1.5. 运输层端口号应用举例



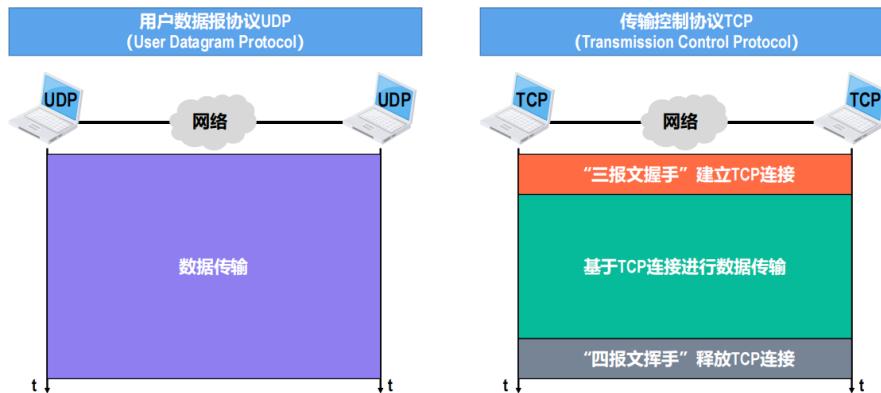
## 2. UDP和TCP的对比



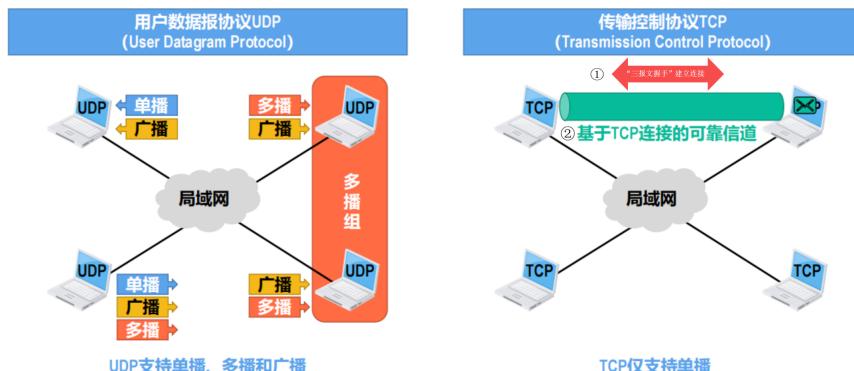
| UDP                            | TCP                         |
|--------------------------------|-----------------------------|
| 无连接                            | 面向连接                        |
| 支持“一对一”、“一对多”、“多对一”和“多对多”交互通信。 | 每一条TCP连接只能有两个端点EP，只能是一对一通信。 |
| 面向应用报文                         | 面向字节流                       |
| 尽最大努力交付，即不可靠；不使用流量控制和拥塞控制。     | 可靠传输，使用流量控制和拥塞控制。           |
| 首部开销小，仅8字节。                    | 首部最小20字节，最大60字节。            |

## 2.1. 无连接的UDP和面向连接的TCP

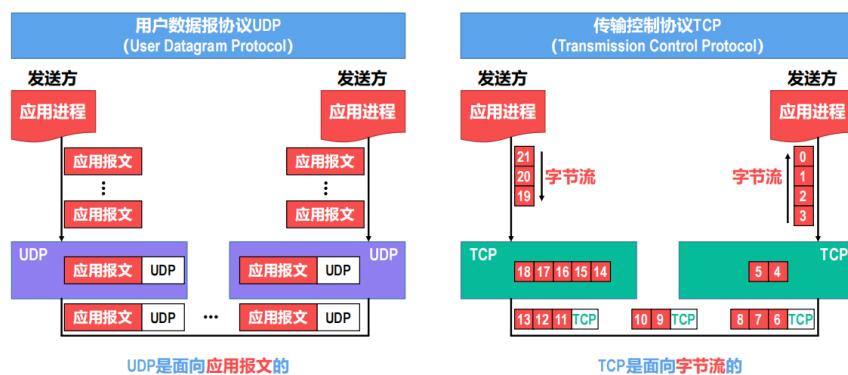
- 这里的“连接”指的是逻辑链接关系，不是物理连接。



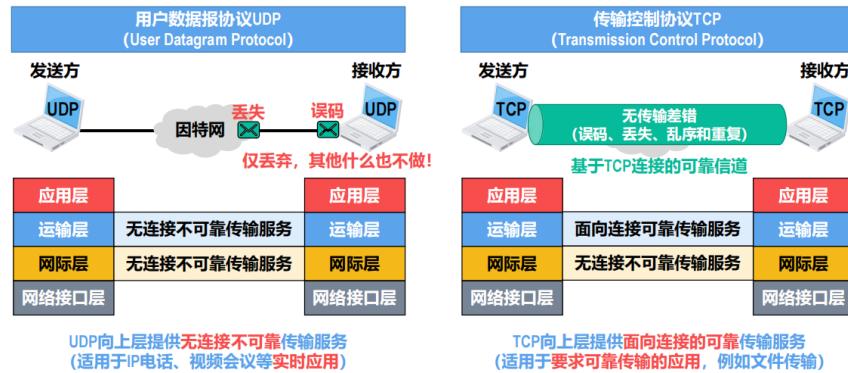
## 2.2. 对单播、多播和广播的支持情况



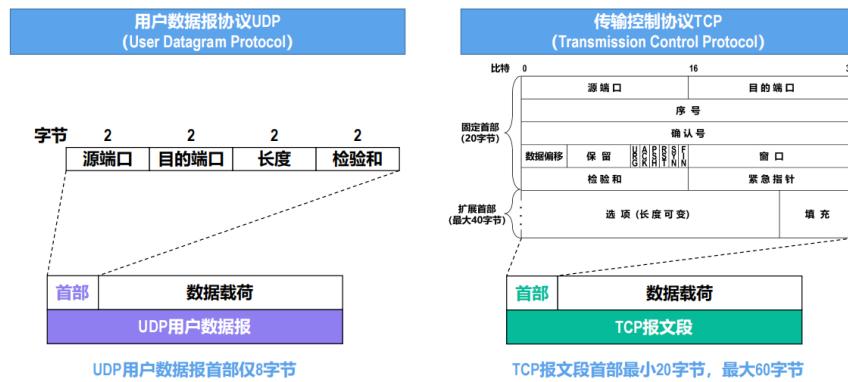
## 2.3. 对应用层报文的处理



## 2.4. 对数据传输可靠性的支持情况

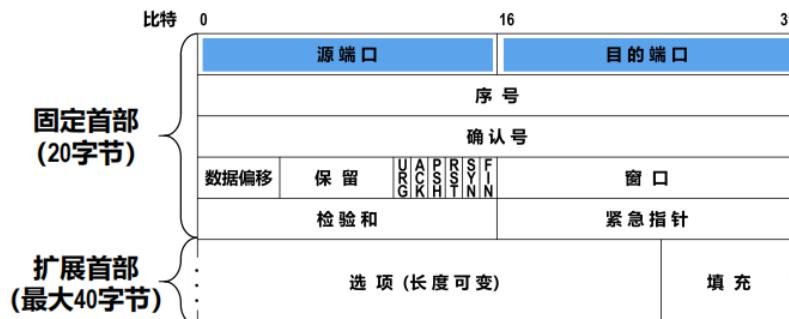


## 2.5. 首部对比



## 3. 传输控制协议

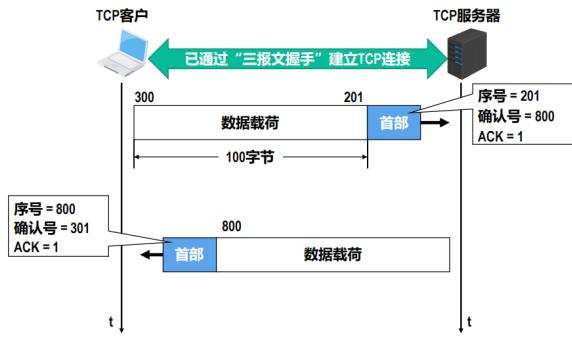
### 3.1. TCP报文段的头部格式



#### 3.1.1. 序号、确认号、确认标志位ACK

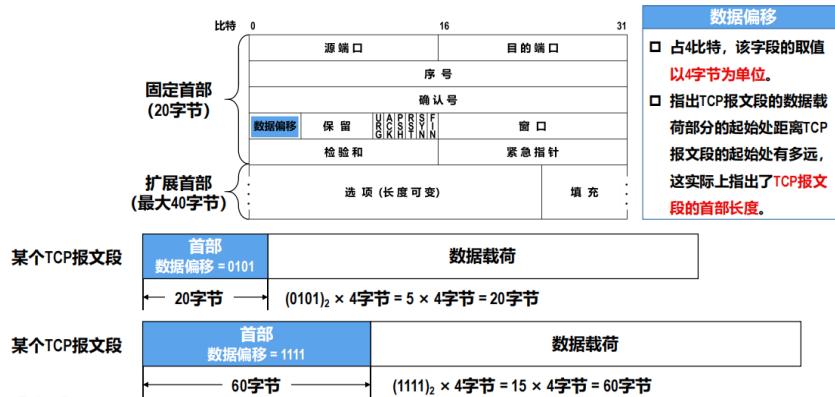
|          |                                                                                                                |
|----------|----------------------------------------------------------------------------------------------------------------|
| 序号       | 占32比特, 取值范围0~2 <sup>32</sup> -1。当序号增加到最后一个时, 下一个序号又回到0。用来指出本TCP报文段数据载荷的第一个字节的序号。                               |
| 确认号      | 占32比特, 取值范围0~2 <sup>32</sup> -1。当确认号增加到最后一个时, 下一个确认号又回到0。用来指出期望收到对方下一个TCP报文段的数据载荷的第一个字节的序号, 同时也是对之前收到的所有数据的确认。 |
| 确认标志位ACK | 只有当ACK取值为1时, 确认号字段才有效。ACK取值为0时, 确认号字段无效。<br>TCP规定: 在TCP连接建立后, 所有传送的TCP报文段都必须把ACK置1。                            |

- 举例



- 确认号应该是已接收且按序到达的最后一次的数据载荷的第一个字节序号。（连续发送中间有丢失情况）

### 3.1.2. 数据偏移

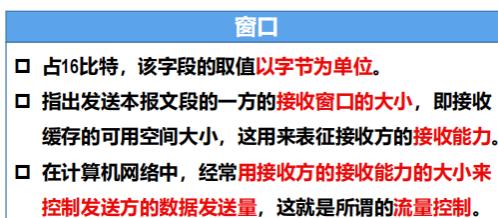


### 3.1.3. 保留

- 占6比特
- 保留为今后使用
- 目前应置为0

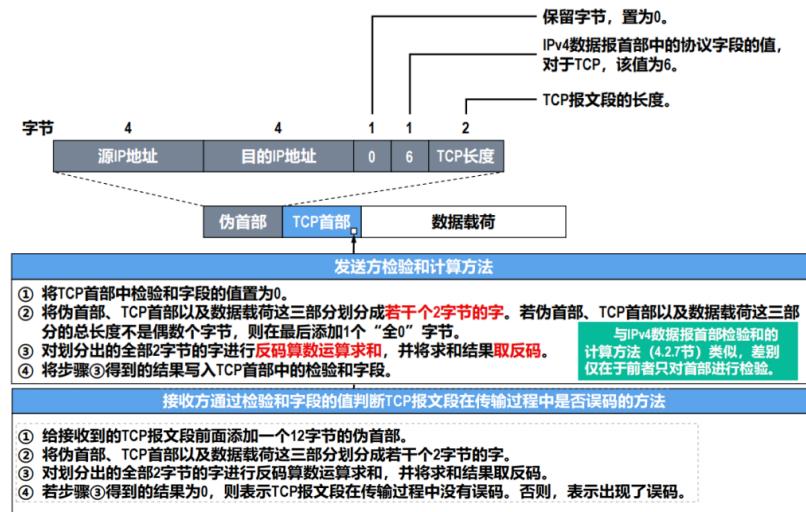
### 3.1.4. 窗口

- 取值范围 $[0, 2^{16} - 1]$



### 3.1.5. 检验和

- 占16比特
- 用来检查整个TCP报文段在传输过程中是否出现了误码。
- 与UDP类似，在计算检验和时，要在TCP报文段前面加上12字节的伪首部



#### • 对比UDP



#### 3.1.6. 同步标志位SYN



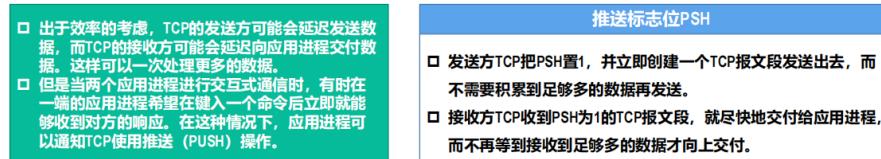
#### 3.1.7. 终止标志位FIN



#### 3.1.8. 复位标志位RST



#### 3.1.9. 推送标志位PSH



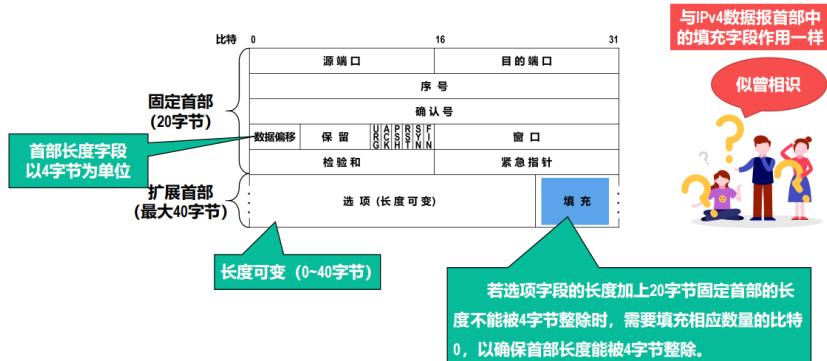
### 3.1.10. 紧急标志位URG、紧急指针

| 紧急标志位URG                                   | 紧急指针                                                                                                                 |
|--------------------------------------------|----------------------------------------------------------------------------------------------------------------------|
| <input type="checkbox"/> 当URG=1时，紧急指针字段有效。 |                                                                                                                      |
| <input type="checkbox"/> 当URG=0时，紧急指针字段无效。 |                                                                                                                      |
|                                            | <input type="checkbox"/> 占16比特，以字节为单位，用来指明紧急数据的长度。                                                                   |
|                                            | <input type="checkbox"/> 当发送方有紧急数据时，可将紧急数据“插队”到发送缓存的最前面，并立刻封装到一个TCP报文段中进行发送。紧急指针会指出本报文段数据载荷部分包含了多长的紧急数据，紧急数据之后是普通数据。 |
|                                            | <input type="checkbox"/> 接收方收到紧急标志位为1的TCP报文段，会按照紧急指针字段的值从报文段数据载荷中取出紧急数据并直接上交应用进程，而不必在接收缓存中排队。                        |

### 3.1.11. 选项（长度可变，最大40字节）

| 选项（长度可变，最大40字节）                                                                                                                    |
|------------------------------------------------------------------------------------------------------------------------------------|
| <input type="checkbox"/> 最大报文段长度MSS选项：指出TCP报文段数据载荷部分的最大长度，而不是整个TCP报文段的长度。                                                          |
| <input type="checkbox"/> 窗口扩大选项：用来扩大窗口，提高吞吐率。                                                                                      |
| <input type="checkbox"/> 时间戳选项： <ul style="list-style-type: none"> <li>用于计算往返时间RTT</li> <li>用于处理序号超范围的情况，又称为防止序号绕回PAWS。</li> </ul> |
| <input type="checkbox"/> 选择确认选项：用来实现选择确认功能。                                                                                        |

### 3.1.12. 填充



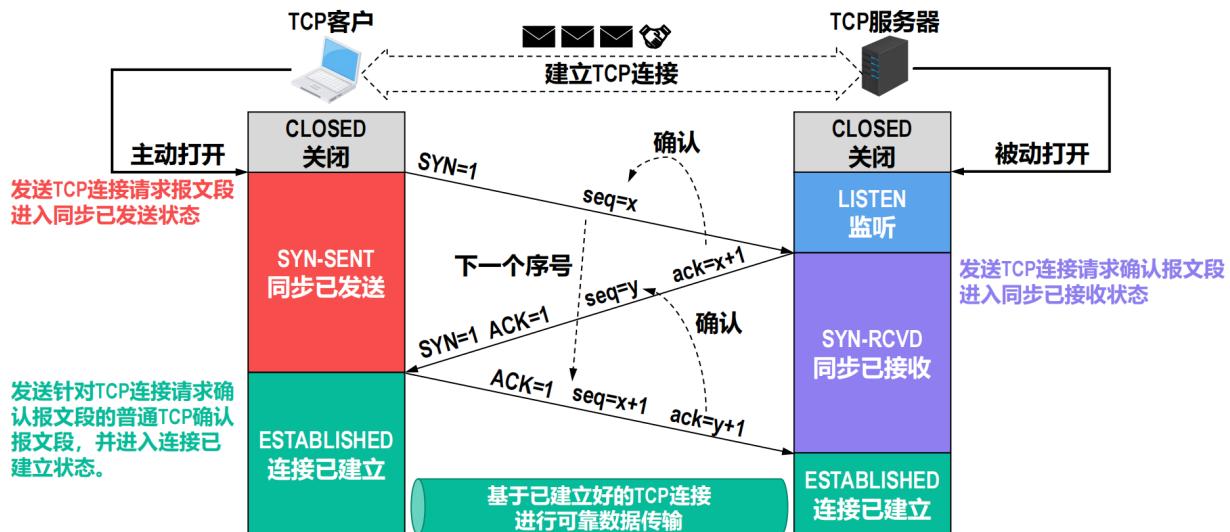
## 3.2. TCP的运输层连接管理

- **TCP是面向连接的协议，它基于运输连接来传送TCP报文段。**
- TCP运输连接的建立和释放，是每一次面向连接的通信中必不可少的过程。**
- **TCP运输连接有以下三个阶段：**
  - ① 通过“三报文握手”来建立TCP连接。
  - ② 基于已建立的TCP连接进行可靠的数据传输。
  - ③ 在数据传输结束后，还要通过“四报文挥手”来释放TCP连接。

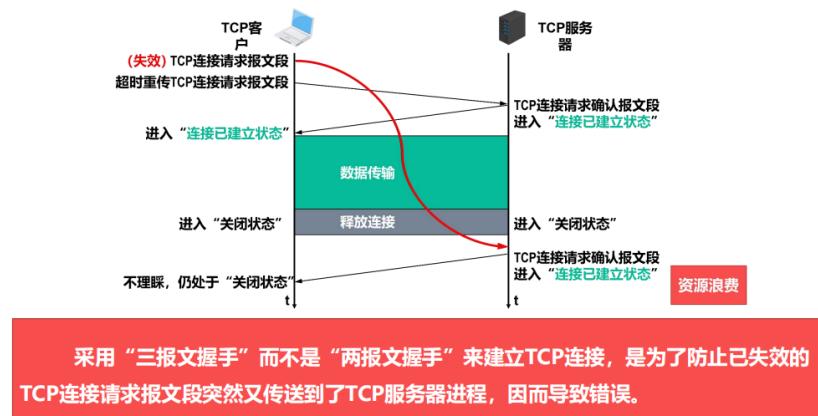


### 3.2.1. “三报文握手”建立连接

- 三报文握手”建立TCP连接的目的
  - 使TCP双方能够确知对方的存在。
  - 使TCP双方能够协商一些参数
    - 例如最大报文段长度、最大窗口大小、时间戳选项等。
    - 使TCP双方能够对运输实体资源进行分配和初始化。
    - 运输实体资源包括缓存大小、各状态变量、连接表中的项目等。

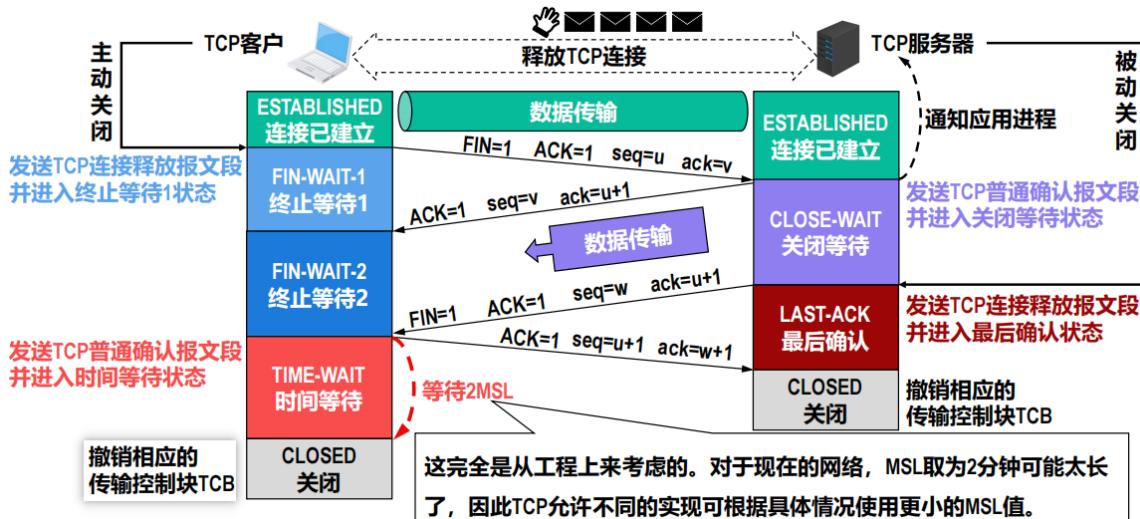


- “三报文”而不是“两报文”

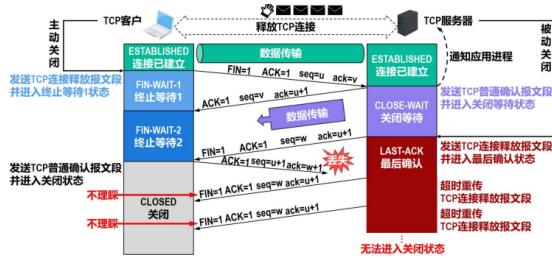


采用“三报文握手”而不是“两报文握手”来建立TCP连接，是为了防止已失效的TCP连接请求报文段突然又传送到了TCP服务器进程，因而导致错误。

### 3.2.2. “四报文挥手”释放连接

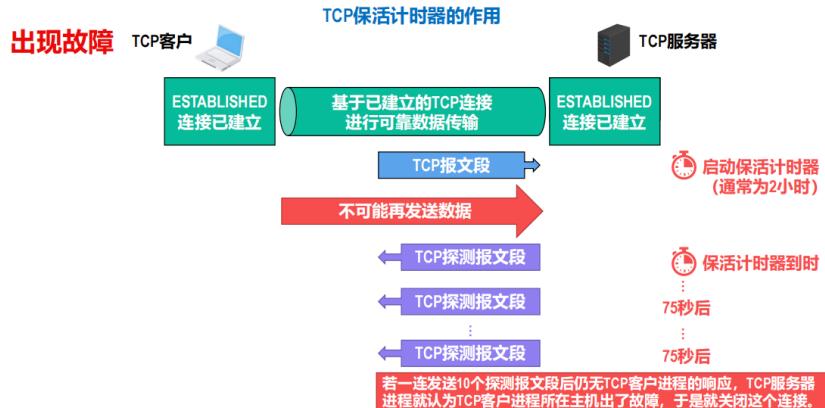


- 等待2MSL原因



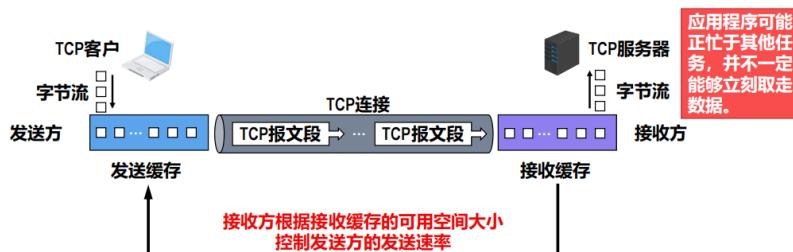
- 处于TIME-WAIT状态后要经过2MSL时长，可以确保TCP服务器进程能够收到最后一个TCP确认报文段而进入CLOSED状态。
- 另外，TCP客户进程在发送完最后一个TCP确认报文段后，再经过2MSL时长，就可以使本次连接持续时间内所产生的所有报文段都从网络中消失。这样就可以使下一个新的TCP连接中不会出现旧连接中的报文段。

### • TCP保活计时器的作用



## 3.3. TCP的流量控制

### 3.3.1. 基本概念



- TCP为应用程序提供了流量控制（Flow Control）机制，以解决因发送方发送数据太快而导致接收方来不及接收，造成接收方的接收缓存溢出的问题。
- 流量控制的基本方法：接收方根据自己的接收能力（接收缓存的可用空间大小）控制发送方的发送速率。

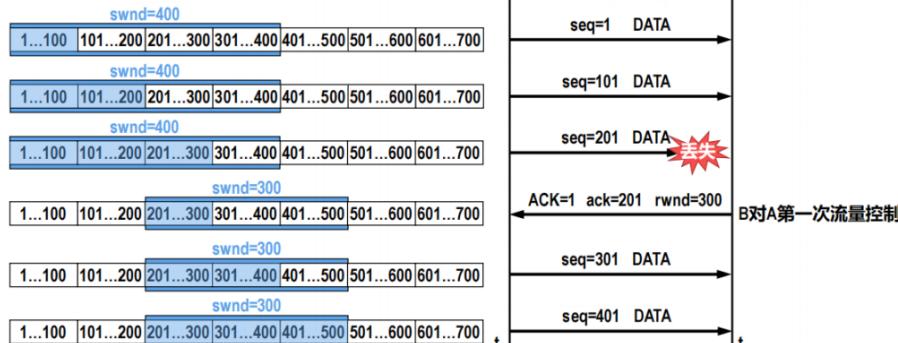
### 3.3.2. 控制方法

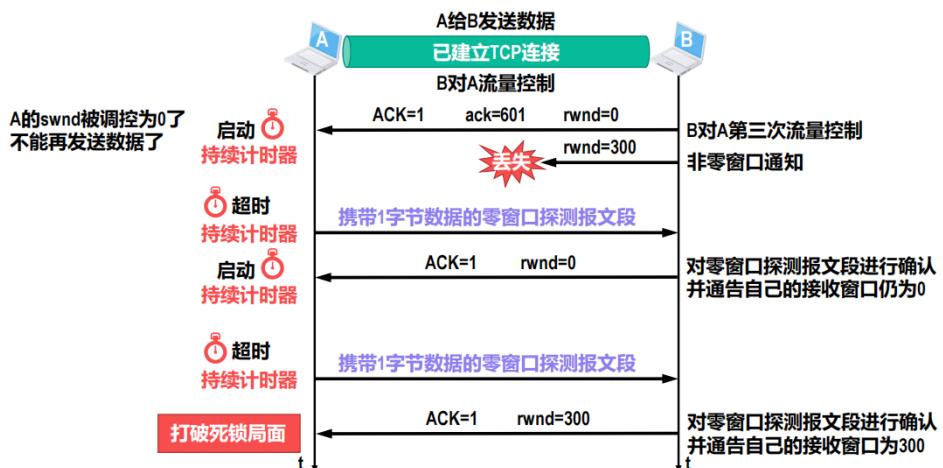
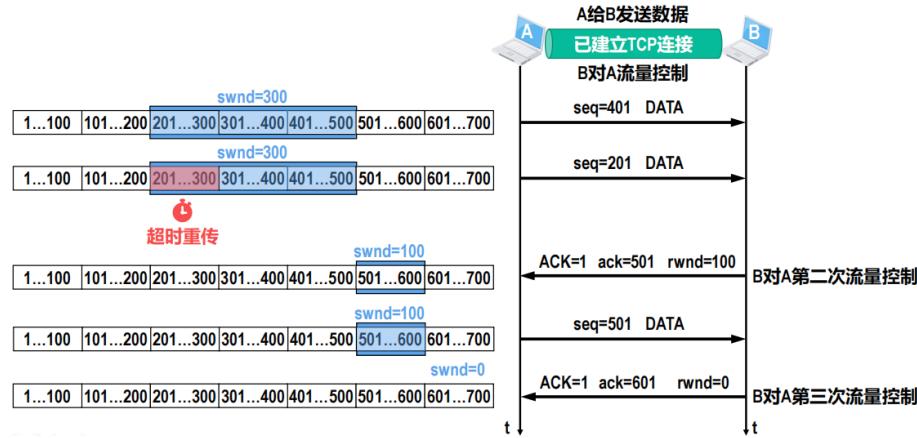
假设：

① 网络不会拥塞（不考虑TCP的拥塞控制）

② 在A和B建立TCP连接时，B告诉A：“我的接收窗口

rwnd=400”，因此A将自己的发送窗口swnd也设置为400。



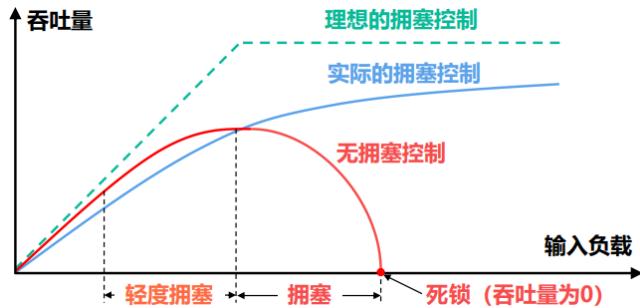


- A发送的零窗口探测报文段到达B时，如果B此时的接收窗口值仍然为0，那么B根本就无法接受该报文段，又怎么会针对该报文段给A发回确认呢？
  - 实际上TCP规定：即使接收窗口值为0，也必须接受零窗口探测报文段、确认报文段以及携带有紧急数据的报文段。
- 如果零窗口探测报文段丢失了，还会打破死锁的局面吗？
  - 回答是肯定的。因为零窗口探测报文段也有重传计时器，当重传计时器超时后，零窗口探测报文段会被重传。

### 3.4. TCP的拥塞控制

### 3.4.1. 基本概念

- 拥塞 (congestion)
  - 在某段时间，若对网络中某一资源的需求超过了该资源所能提供的可用部分，网络性能就要变坏的情况。
- 计算机网络中的链路容量（带宽）、交换节点中的缓存和处理机等都是网络的资源。
- 若出现拥塞而不进行控制，整个网络的吞吐量将随输入负载的增大而下降。

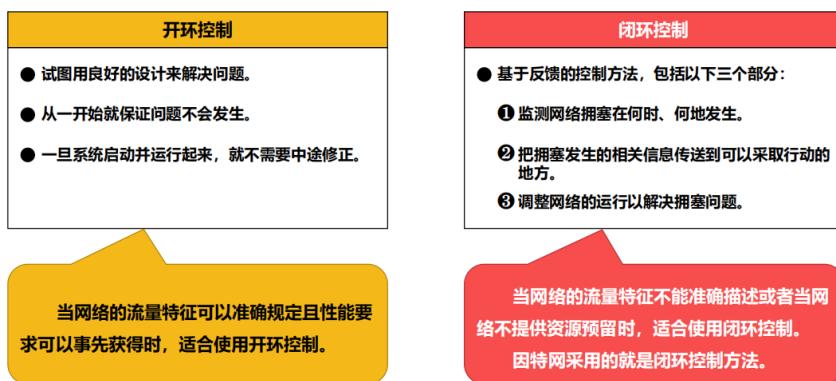


### 3.4.2. 基本方法

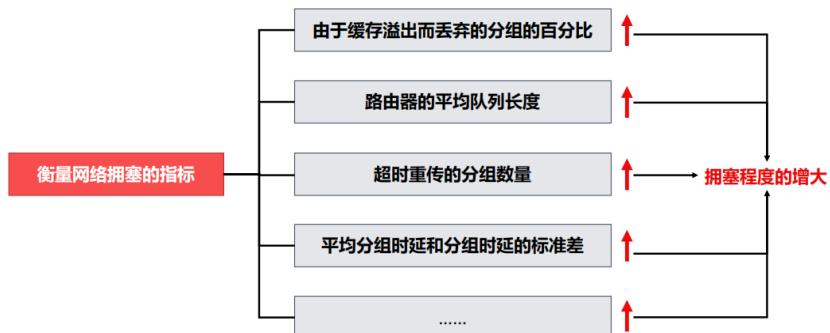
#### 3.4.2.1. 对比流量控制



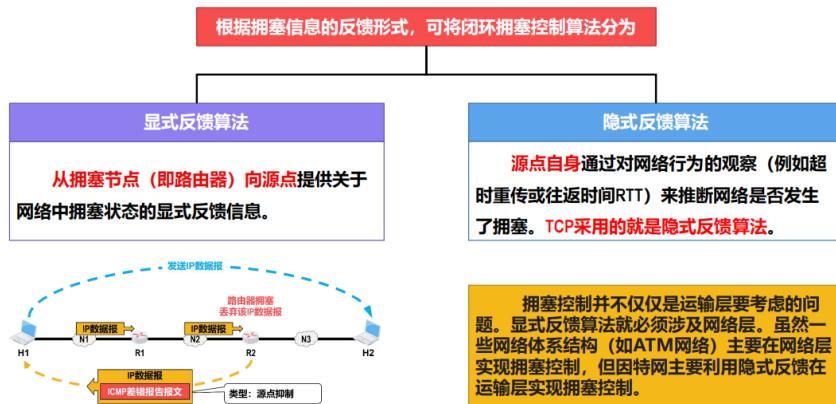
#### 3.4.2.2. 开环控制与闭环控制



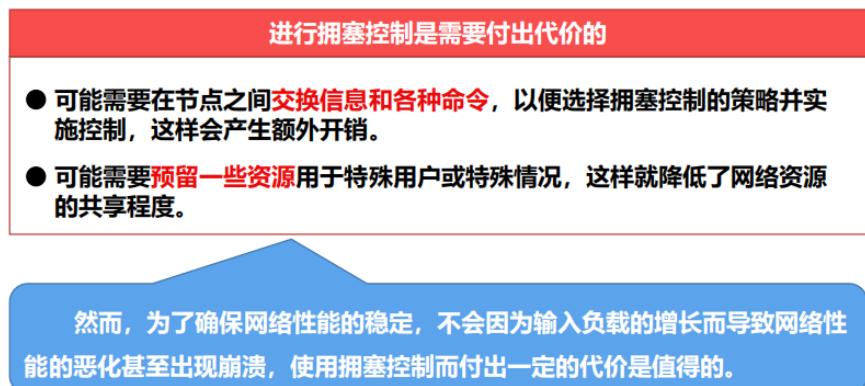
#### 3.4.2.3. 衡量网络拥塞的指标



### 3.4.2.4. 闭环拥塞控制算法



### 3.4.2.5. 代价

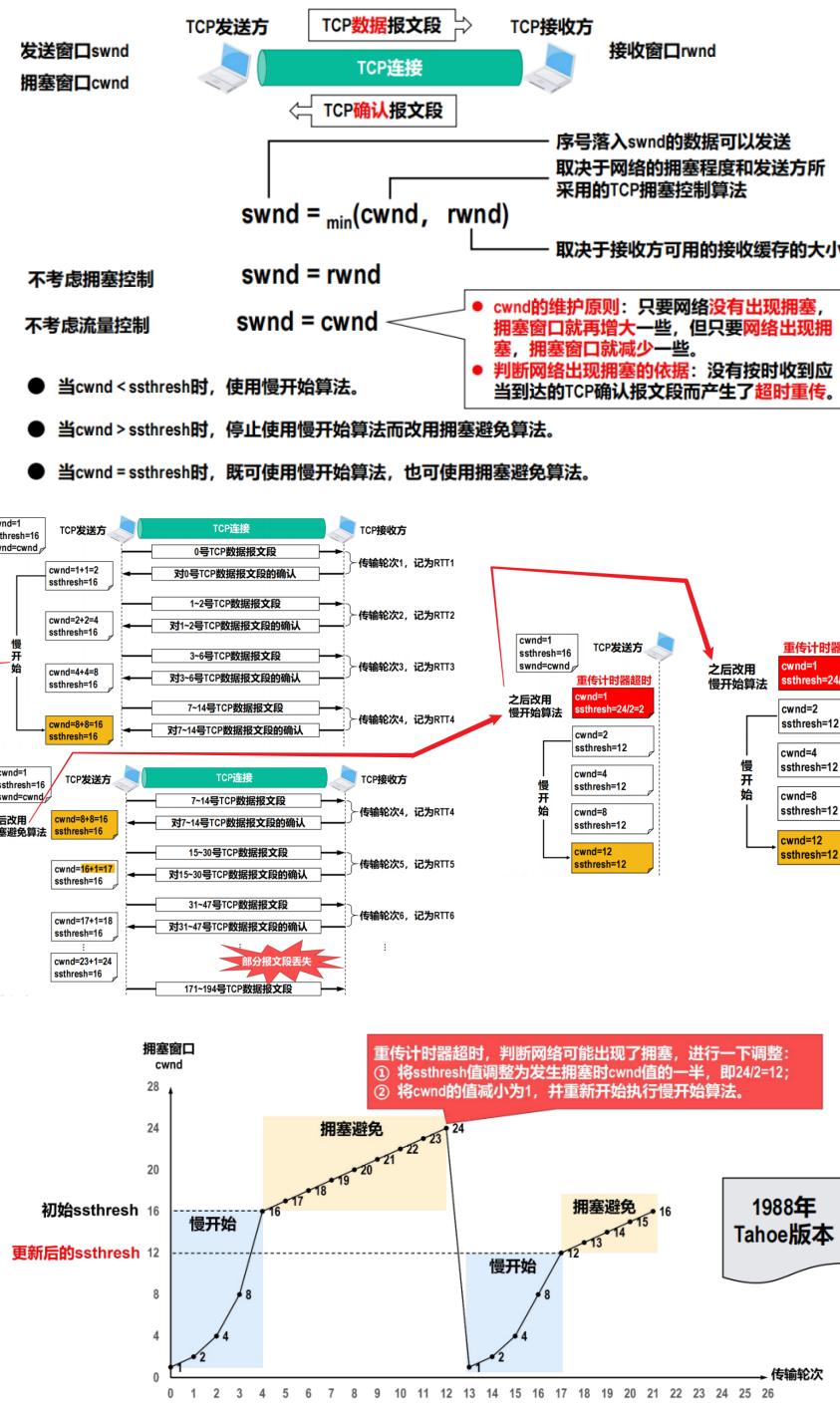


### 3.4.3. TCP的四种拥塞控制方法



#### 3.4.3.1. 慢开始算法和拥塞避免算法

- “慢开始”是指一开始向网络注入的报文段少，而并不是指拥塞窗口cwnd的值增长速度慢。
- “拥塞避免”也并非指完全能够避免拥塞，而是指在拥塞避免阶段将cwnd值控制为按线性规律增长，使网络比较不容易出现拥塞。

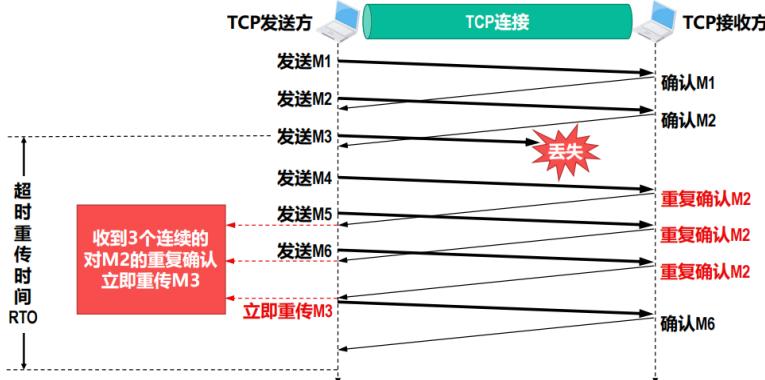


### 3.4.3.2. 快重传算法和快恢复算法 (改进TCP性能, 1990年Reno版本)

- 改进: IP数据包误码被丢弃, 导致重传计时器超时, 使发送方误认为网络出现了拥塞, 重新执行慢开始算法, 降低了传输效率。
- “快重传”是指使发送方尽快(尽早)进行重传, 而不是等重传计时器超时再重传。
  - 这就要求接收方不要等待自己发送数据时才进行捎带确认, 而是要立即发送确认, 即使收到了失序的报文段也要立即发出对已收到的报文段的重复确认。
  - 发送方一旦收到3个连续的重复确认, 就将相应的报文段立即重传, 而不是等该报文段的重传计时器超时再重传。
- 快重传: 收到不是按序到达的报文段就发送上一个报文段的重复确认。

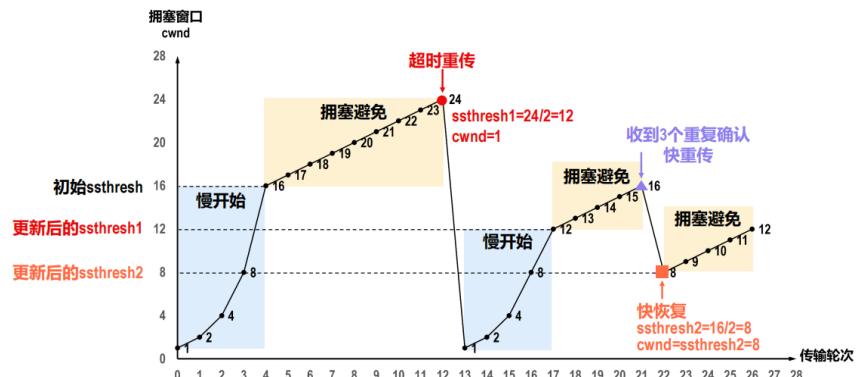
- 采用快重传算法可以让发送方尽早知道发生了个别TCP报文段的丢失。

**【举例】快重传算法**

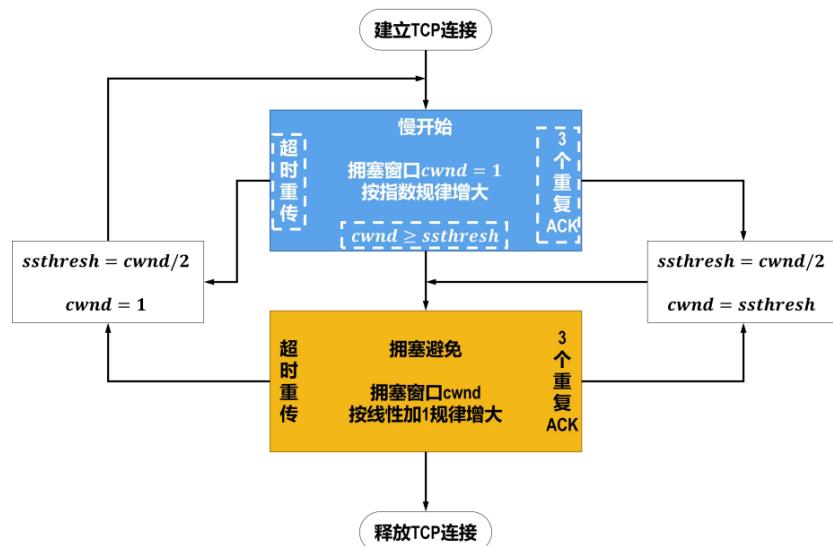


- 快恢复：发送方一旦收到3个重复确认，就知道现在只是丢失了个别的报文段，于是不启动慢开始算法，而是执行快恢复算法。
  - 发送方将慢开始门限ssthresh的值和拥塞窗口cwnd的值都调整为当前cwnd值的一半，并开始执行拥塞避免算法。
  - 也有的快恢复实现是把快恢复开始时的cwnd值再增大一些，即 $cwnd = \text{新}ssthresh + 3$ 。
    - 既然发送方收到了3个重复的确认，就表明有3个数据报文段已经离开了网络。
    - 这3个报文段不再消耗网络资源而是停留在接收方的接收缓存中。
    - 可见现在网络中不是堆积了报文段而是减少了3个报文段，因此可以适当把cwnd值增大一些。

### 3.4.3.3. 四种控制方法拥塞窗口变化图



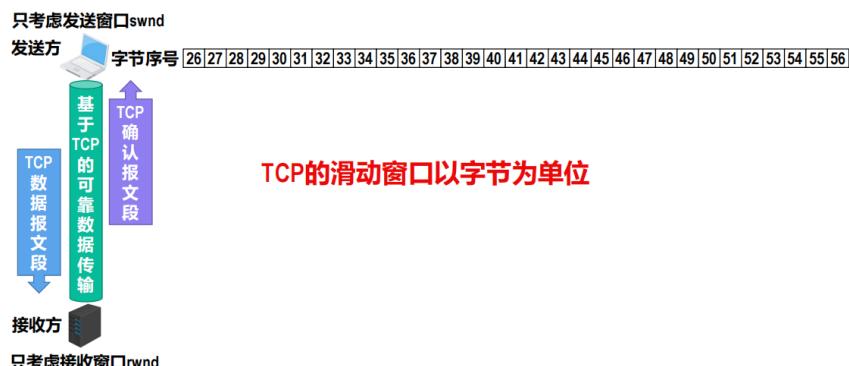
### 3.4.3.4. TCP拥塞控制流程图

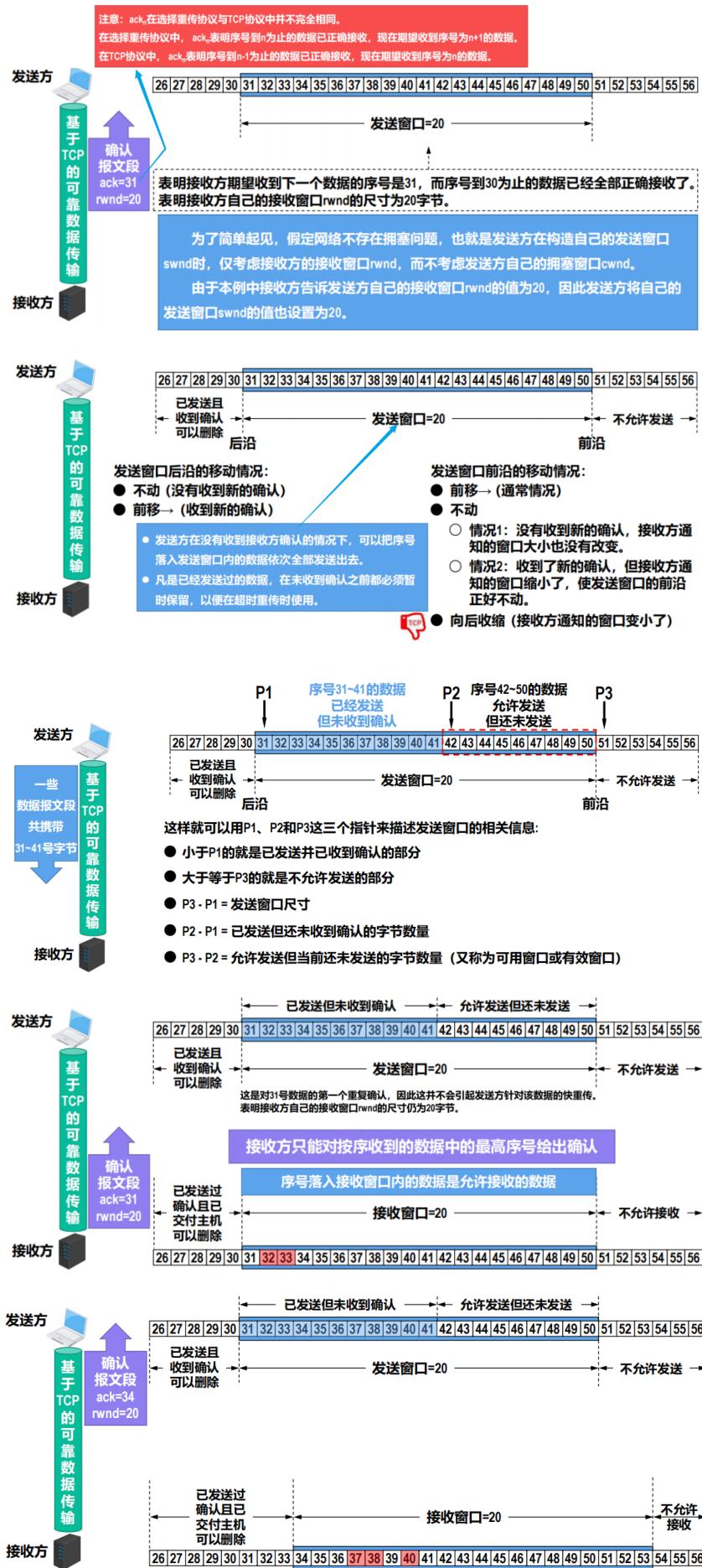


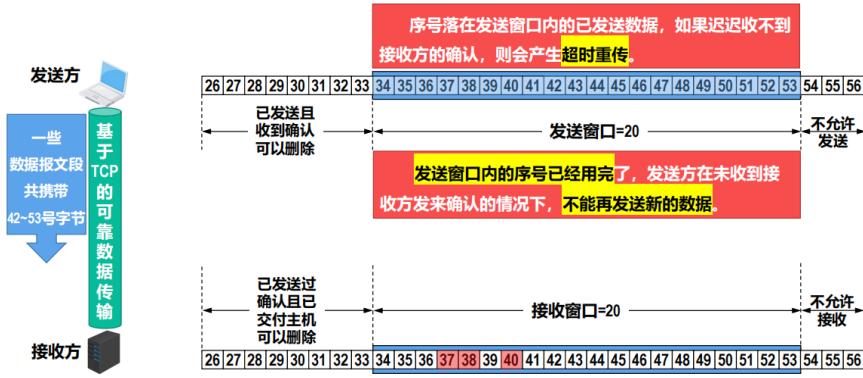
### 3.4.4. TCP拥塞控制与网际层拥塞控制的关系



## 3.5. TCP可靠传输的实现



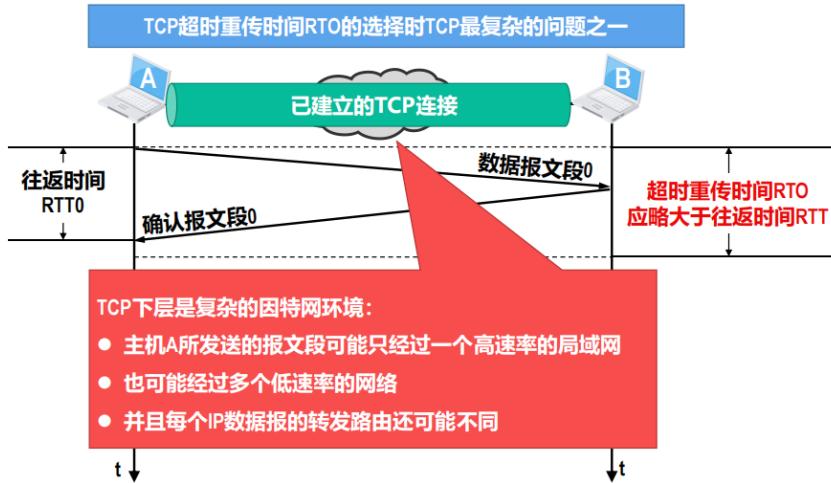




对于TCP可靠传输的实现，还需要做以下补充说明

- (1) 虽然发送方的发送窗口是根据接收方的接收窗口设置的，但在同一时刻，**发送方的发送窗口并不总是和接收方的接收窗口一样大**，这是因为：
  - 网络传送窗口值需要经历一定的时间滞后，并且这个时间还是不确定的。
  - 发送方还可能根据网络当时的拥塞情况适当减小自己的发送窗口尺寸。
- (2) 对于不按序到达的数据应如何处理，**TCP并无明确规定**。
  - 如果接收方把不按序到达的数据一律丢弃，那么接收窗口的管理将会比较简单，但这样做对网络资源的利用不利，因为发送方会重复传送较多的数据。
  - TCP通常对不按序到达的数据先临时存放在接收窗口中，等到字节流中所缺少的字节收到后，再按序交付上层的应用进程。
- (3) TCP要求接收方**必须有累积确认**（这一点与选择重传协议不同）和**捎带确认机制**。这样可以减小传输开销。接收方可以在合适的时候发送确认，也可以在自己有数据要发送时把确认信息顺便捎带上。
  - **接收方不应过分推迟发送确认**，否则会导致发送方不必要的超时重传，这反而浪费了网络资源。TCP标准规定，确认推迟的时间不应超过0.5秒。若收到一连串具有最大长度的报文段，则必须每隔一个报文段就发送一个确认 [RFC 1122]。
  - **捎带确认实际上并不经常发生**，因为大多数应用程序很少同时在两个方向上发送数据。
- (4) **TCP的通信是全双工通信**。通信中的每一方都在发送和接收报文段。因此，每一方都有自己的发送窗口和接收窗口。在谈到这些窗口时，一定要弄清楚是哪一方的窗口。

### 3.6. TCP超时重传时间的选择



- RTTS计算

- 不能直接使用略大于某次测量得到的往返时间RTT样本的值作为超时重传时间RTO。
- 但是，可以利用每次测量得到的RTT样本计算**加权平均往返时间RTTs**，这样可以得到比较平滑的往返时间。

$$RTT_{S1} = RTT_1$$

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

在上式中， $0 \leq \alpha < 1$

若 $\alpha$ 很接近于0，则新RTT样本对RTTs的影响不大；

若 $\alpha$ 很接近于1，则新RTT样本对RTTs的影响较大；

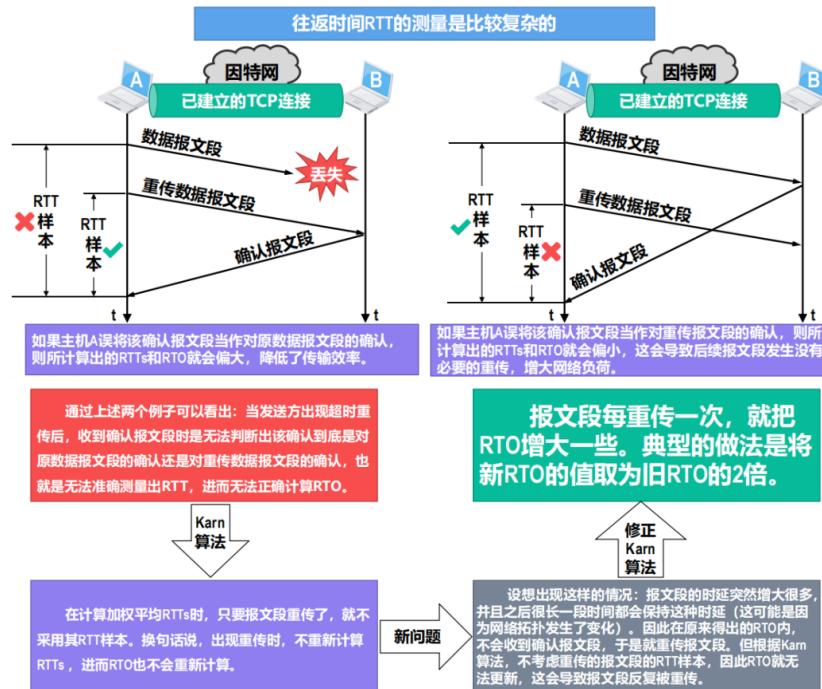
已成为建议标准的[RFC 6298]推荐的 $\alpha$ 值为1/8，即0.125。

■ 显然，超时重传时间RTO的值应略大于加权平均往返时间RTTs的值（而不是某个RTT样本的值）。

■ [RFC 6298]建议使用下式来计算超时重传时间RTO：

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

| 加权平均往返时间RTT <sub>s</sub>                                                                                                                                                                | RTT偏差的加权平均RTT <sub>D</sub>                                                                                                                                                                          | 如果所测量到的RTT样本不正确，那么所计算出的RTT <sub>s</sub> 和RTT <sub>D</sub> 自然就不正确，进而所计算出的RTO也就不正确。然而，RTT的测量确实是比较复杂的。 |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------|
| $RTT_{s1} = RTT_1$<br>新的 $RTT_s = (1 - \alpha) \times \text{旧的}RTT_s + \alpha \times \text{新的RTT样本}$<br>在上式中， $0 \leq \alpha < 1$ ，已成为建议标准的[RFC 6298]推荐的 $\alpha$ 值为 $1/8$ ，即 $0.125$ 。 | $RTT_{D1} = RTT_1 \div 2$<br>新的 $RTT_D = (1 - \beta) \times \text{旧的}RTT_D + \beta \times  RTT_s - \text{新的RTT样本} $<br>在上式中， $0 \leq \beta < 1$ ，已成为建议标准的[RFC 6298]推荐的 $\beta$ 值为 $1/4$ ，即 $0.25$ 。 |                                                                                                     |



- 总结

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

| 加权平均往返时间RTT <sub>s</sub>                                                                                                                                                                | 报文段发生重传，就不采用RTT样本计算RTO，而是把RTO增大一些。典型的做法是将新RTO的值取为旧RTO的2倍。 |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------|
| $RTT_{s1} = RTT_1$<br>新的 $RTT_s = (1 - \alpha) \times \text{旧的}RTT_s + \alpha \times \text{新的RTT样本}$<br>在上式中， $0 \leq \alpha < 1$ ，已成为建议标准的[RFC 6298]推荐的 $\alpha$ 值为 $1/8$ ，即 $0.125$ 。 |                                                           |

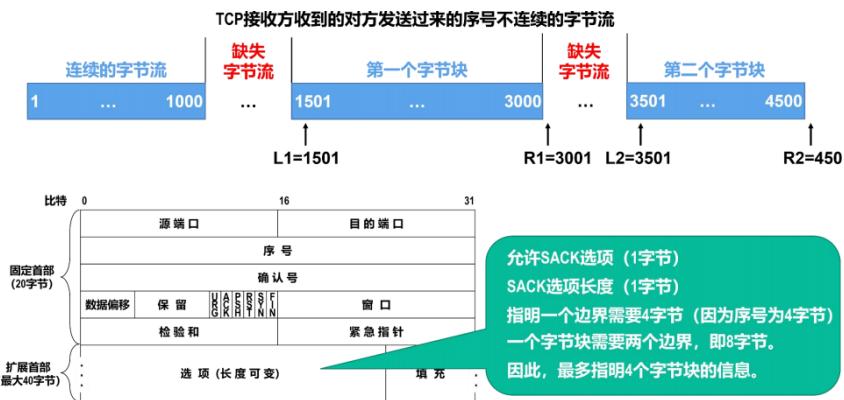
  

| RTT偏差的加权平均RTT <sub>D</sub>                                                                                                                                                                          |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $RTT_{D1} = RTT_1 \div 2$<br>新的 $RTT_D = (1 - \beta) \times \text{旧的}RTT_D + \beta \times  RTT_s - \text{新的RTT样本} $<br>在上式中， $0 \leq \beta < 1$ ，已成为建议标准的[RFC 6298]推荐的 $\beta$ 值为 $1/4$ ，即 $0.25$ 。 |

### 3.7. TCP的选择确认

■ 在之前介绍TCP的快重传和可靠传输时，TCP接收方只能对按序收到的数据中的最高序号给出确认。当发送方超时重传时，接收方之前已收到的未按序到达的数据也会被重传。





SACK相关文档并没有指明发送方应当怎样响应SACK。因此大多数的TCP实现还是重传所有未被确认的数据块。

## 4. 题目

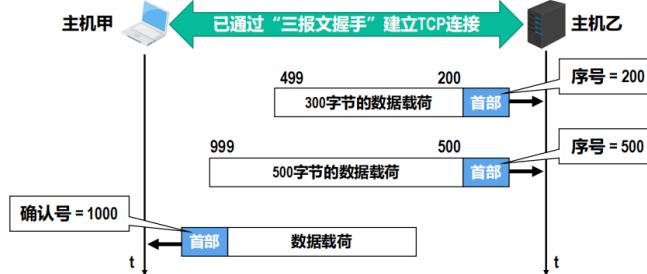
### 4.1. TCP序号、确认号

#### 4.1.1. 【2009 38】

【2009年题38】主机甲与主机乙之间已建立一个TCP连接，主机甲向主机乙发送了两个连续的TCP段，分别包含300字节和500字节的有效载荷，第一个段的序列号为200，主机乙正确接收到两个段后，发给主机甲的确认序列号是（D）。

- A. 500      B. 700      C. 800      D. 1000

解析

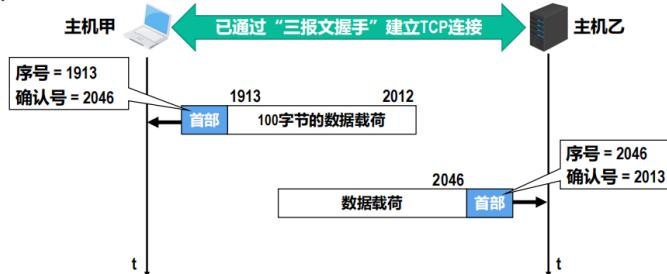


#### 4.1.2. 【2013 39】

【2013年题39】主机甲与主机乙之间已建立一个TCP连接，双方持续有数据传输，且数据无差错与丢失。若甲收到1个来自乙的TCP段，该段的序号是1913、确认序号为2046、有效载荷为100字节，则甲立即发送给乙的TCP段的序号和确认序号分别是（B）。

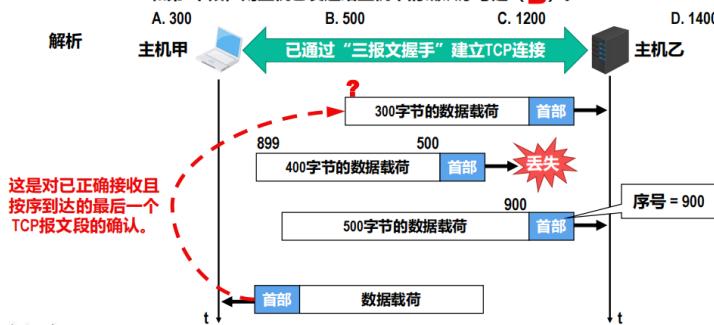
- A. 2046、2012      B. 2046、2013      C. 2047、2012      D. 2047、2013

解析



#### 4.1.3. 【2011 40】

【2011年题40】主机甲与主机乙之间已建立一个TCP连接，主机甲向主机乙发送了3个连续的TCP段，分别包含300字节、400字节和500字节的有效载荷，第3个段的序号为900，若主机乙仅正确接收到第1和第3个段，则主机乙发送给主机甲的确认序号是（B）。

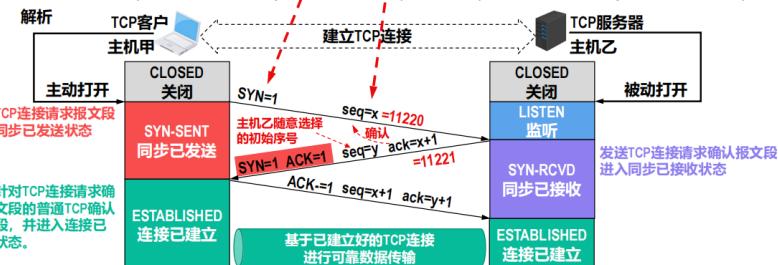


## 4.2. 建立TCP连接

### 4.2.1. 【2011 39】

【2011年题39】主机甲向主机乙发送一个（SYN=1, seq=11220）的TCP段，期望与主机乙建立TCP连接，若主机乙接受该连接请求，则主机乙向主机甲发送的正确的TCP段可能是（C）。

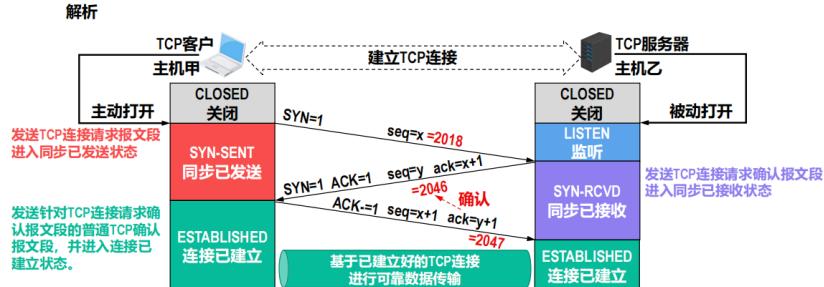
- A. (SYN=0, ACK=0, seq=11221, ack=11221)      B. (SYN=1, ACK=1, seq=11220, ack=11220)  
C. (SYN=1, ACK=1, seq=11221, ack=11221)      D. (SYN=0, ACK=0, seq=11220, ack=11220)



### 4.2.2. 【2019 39】

【2019年题39】若主机甲主动发起一个与主机乙的TCP连接，甲、乙选择的初始序列号分别为2018和2046，则第三次握手TCP段的确认序列号是（D）。

- A. 2018      B. 2019      C. 2046      D. 2047



## 4.3. 释放TCP连接

### 4.3.1. 【2020 39】

【2020年题39】若主机甲与主机乙建立TCP连接时发送的SYN段中的序号为1000，在断开连接时，甲发送给乙的FIN段中的序号为5001，则在无任何重传的情况下，甲向乙已经发送的应用层数据的字节数为 (C)。

A. 4002

B. 4001

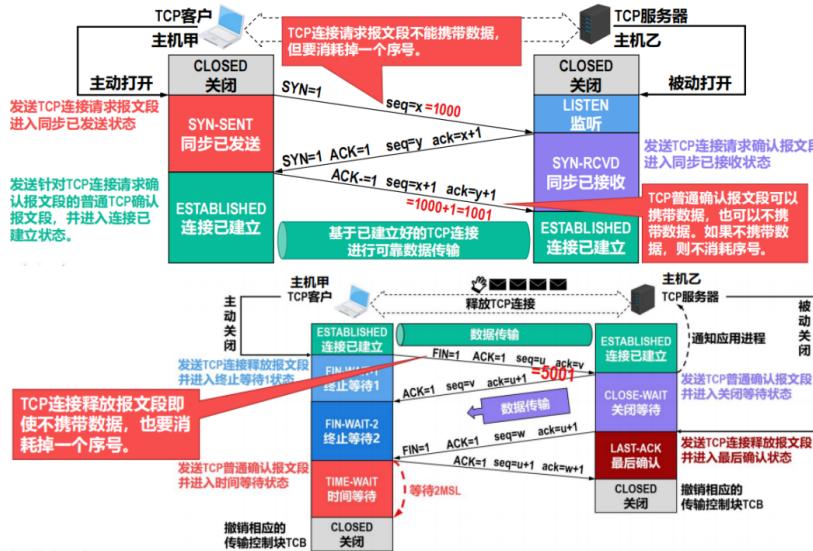
C. 4000

D. 3999

解析 甲给乙发送的第一个应用层数据字节的TCP序号为1001，因为应用层数据作为数据载荷被封装在TCP报文段中。

甲在发送FIN段之前，给乙发送的最后一个应用层数据字节的TCP序号为5000。

综上所述，甲向乙已经发送了字节序号为1001~5000共4000个字节的应用层数据。



## 4.4. TCP流量控制

### 4.4.1. 【2010 39】

【2010年题39】主机甲和主机乙之间建立了一个TCP连接，TCP最大报文段长度为1000字节。若主机甲的当前拥塞窗口为4000字节，在主机甲向主机乙连续发送两个最大报文段后，成功收到主机乙发送的第一个段的确认段，确认段中通告的接收窗口大小为2000字节，则此时主机甲还可以向主机乙发送的最大字节数是 (A)。

A. 1000

B. 2000

C. 3000

D. 4000

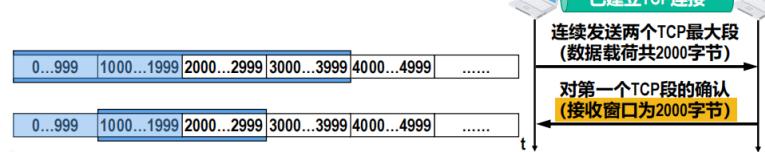
解析

TCP发送方的发送窗口值 =  $\min[\text{TCP发送方的拥塞窗口值}, \text{TCP接收方的接收窗口值}]$

题目未给出TCP发送方的发送窗口值以及TCP接收方的接收窗口值，则取拥塞窗口值作为发送窗口值。

TCP最大报文段长度MSS，并不是指整个TCP报文段的长度，而是指TCP报文段的数据载荷的长度。

甲还可向乙发送2000~2999号字节数据，共1000个字节。



## 4.5. TCP拥塞控制

### 4.5.1. 【2009 39】

【2009年题39】一个TCP连接总是以1KB的最大报文段发送TCP段，发送方有足够的数据要发送。当拥塞窗口为16KB时发生了超时，如果接下来的4个RTT（往返时间）时间内的TCP段的传输都是成功的，那么当第4个RTT时间内发送的所有TCP段都得到肯定应答时，拥塞窗口大小是 (C)。

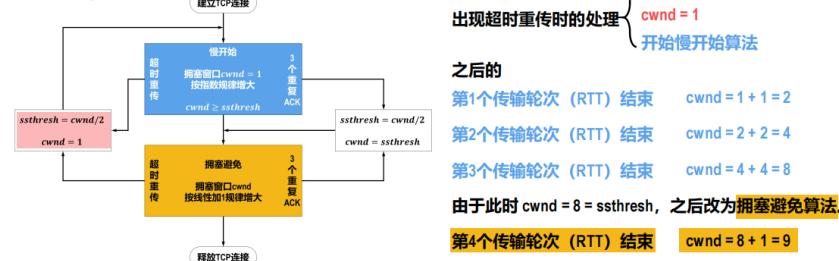
A. 7KB

B. 8KB

C. 9KB

D. 16KB

解析



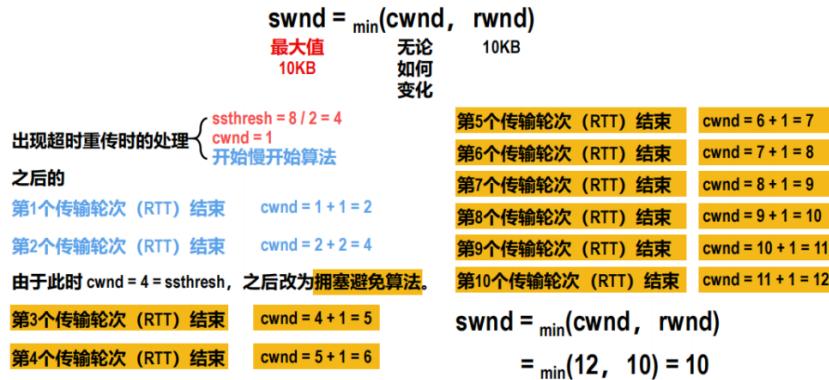
## 4.5.2. 【2014 38】

【2014年题38】主机甲和主机乙已建立了TCP连接，甲始终以MSS=1KB大小的段发送数据，并一直有数据发送；乙每收到一个数据段都会发出一个接收窗口为10KB的确认段。若甲在t时刻发生超时并拥塞窗口为8KB，则从t时刻起，不再发生超时的情况下，经过10个RTT后，甲的发送窗口是（A）。

- A. 10KB      B. 12KB 排除      C. 14KB 排除      D. 15KB 排除

解析

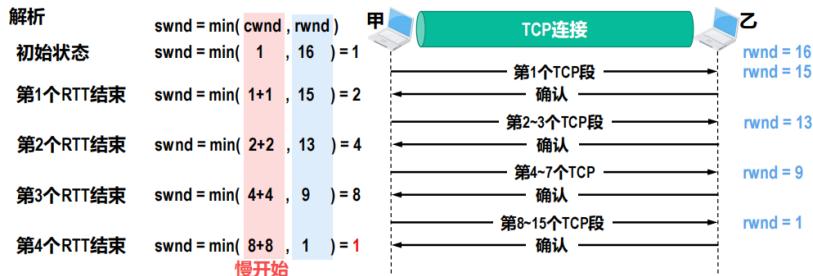
本题不知是出题人在给出的选项上有小小的失误，还是有意而为之，使得我们可以“秒杀”。



## 4.5.3. 【2015 39】

【2015年题39】主机甲和主机乙新建一个TCP连接，甲的拥塞控制初始阈值为32KB，甲向乙始终以MSS=1KB大小的段发送数据，并一直有数据发送；乙为该连接分配16KB接收缓存，并对每个数据段进行确认，忽略段传输延迟。若乙收到的数据全部存入缓存，不被取走，则甲从连接建立成功时刻起，未发生超时的情况下，经过4个RTT后，甲的发送窗口是（A）。

- A. 1KB      B. 8KB      C. 16KB      D. 32KB



## 4.5.4. 【2017 39】

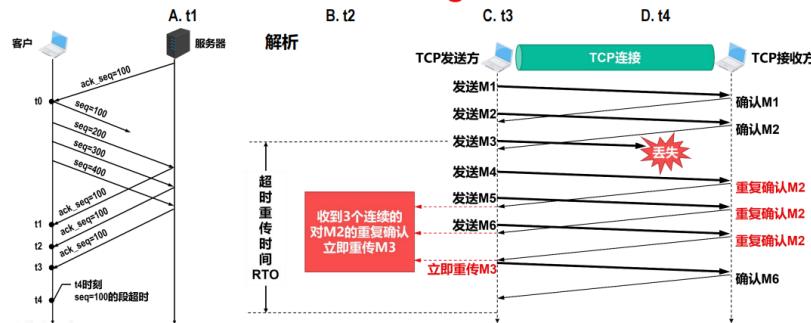
【2017年题39】若甲向乙发起一个TCP连接，最大段长MSS=1KB，RTT=5ms，乙开辟的接收缓存为64KB，则甲从连接建立成功至发送窗口达到32KB，需经过的时间至少是（A）。

- A. 25ms      B. 30ms      C. 160ms      D. 165ms



## 4.5.5. 【2019 38】

【2019年题38】若客户通过一个TCP连接向服务器发送数据的部分过程如下图所示。客户在t0时刻第一次收到确认序列号ack\_seq=100的段，并发送序列号seq=100的段，但发生丢失。若TCP支持快速重传，则客户重新发送seq=100段的时刻是 (C)。



#### 4.5.6. 【2020 38】

【2020年题38】若主机甲与主机乙已建立一条TCP连接，最大段长MSS为1KB，往返时间RTT为2ms，则在不出现拥塞的前提下，拥塞窗口从8KB增长到20KB所需的最长时间是 (C)。

- A. 4ms      B. 8ms      C. 24ms      D. 48ms

解析

若拥塞窗口从8KB增长到20KB所需的时间最长，则该过程应处于拥塞避免算法的执行过程中。

在上述过程中：

第1个传输轮次 (RTT) 结束  $cwnd = 8 + 1 = 9$

第2个传输轮次 (RTT) 结束  $cwnd = 9 + 1 = 10$

⋮

第12个传输轮次 (RTT) 结束  $cwnd = 19 + 1 = 20$

共经历了12个传输轮次 (RTT)，总耗时为  $2ms \times 12 = 24ms$

## 4.6. TCP超时重传的时间选择

【练习】假设要进行五次TCP往返时间RTT的测量，从第一次开始，依次测得RTT为30ms, 26ms, 32ms, 24ms，而第五次测量RTT时出现了超时重传。设 $\alpha = \beta = 0.1$ ，请计算每次测量后所计算出的超时重传时间RTO。

解析

根据题意可知， $RTT_1=30ms$ ,  $RTT_2=26ms$ ,  $RTT_3=32ms$ ,  $RTT_4=24ms$ ，测量 $RTT_5$ 时出现超时重传。

$$RTT_{S1} = RTT_1 = 30ms$$

$$RTT_{D1} = RTT_1 / 2 = 15ms$$

$$\begin{aligned} RTT_{S2} &= (1 - \alpha) \times RTT_{S1} + \alpha \times RTT_2 \\ &= (1 - 0.1) \times RTT_{S1} + 0.1 \times RTT_2 \\ &= 0.9 \times 30ms + 0.1 \times 26ms = 29.6ms \end{aligned}$$

$$\begin{aligned} RTT_{D2} &= (1 - \beta) \times RTT_{D1} + \beta \times |RTT_{S2} - RTT_2| \\ &= (1 - 0.1) \times RTT_{D1} + 0.1 \times |RTT_{S2} - RTT_2| \\ &= 0.9 \times 15ms + 0.1 \times |29.6ms - 26ms| = 13.86ms \end{aligned}$$

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

$$RTO_1 = RTT_{S1} + 4 \times RTT_{D1} = 30ms + 4 \times 15ms = 90ms$$

$$\begin{aligned} RTT_{S3} &= (1 - \alpha) \times RTT_{S2} + \alpha \times RTT_3 \\ &= 0.9 \times 29.6ms + 0.1 \times 32ms = 29.84ms \end{aligned}$$

$$\begin{aligned} RTT_{D3} &= (1 - \beta) \times RTT_{D2} + \beta \times |RTT_{S3} - RTT_3| \\ &= 0.9 \times 13.86ms + 0.1 \times |29.84ms - 32ms| = 12.69ms \end{aligned}$$

$$RTO_2 = RTT_{S2} + 4 \times RTT_{D2} = 29.6ms + 4 \times 13.86ms = 85.04ms$$

$$RTO_3 = RTT_{S3} + 4 \times RTT_{D3} = 29.84ms + 4 \times 12.69ms = 80.6ms$$

$$\begin{aligned} RTT_{S4} &= (1 - \alpha) \times RTT_{S3} + \alpha \times RTT_4 \\ &= 0.9 \times 29.84ms + 0.1 \times 24ms = 29.256ms \end{aligned}$$

$$\begin{aligned} RTT_{D4} &= (1 - \beta) \times RTT_{D3} + \beta \times |RTT_{S4} - RTT_4| \\ &= 0.9 \times 12.69ms + 0.1 \times |29.256ms - 24ms| = 11.9466ms \end{aligned}$$

$$RTO_4 = RTT_{S4} + 4 \times RTT_{D4} = 29.256ms + 4 \times 11.9466ms = 77.0424ms$$

由于测量 $RTT_5$ 时出现超时重传，因此不用计算 $RTT_{S5}$ 。

由于测量 $RTT_5$ 时出现超时重传，因此不用计算 $RTT_{D5}$ 。

$$RTO_5 = 2 \times RTO_4 = 2 \times 77.0424ms = 154.0848ms$$