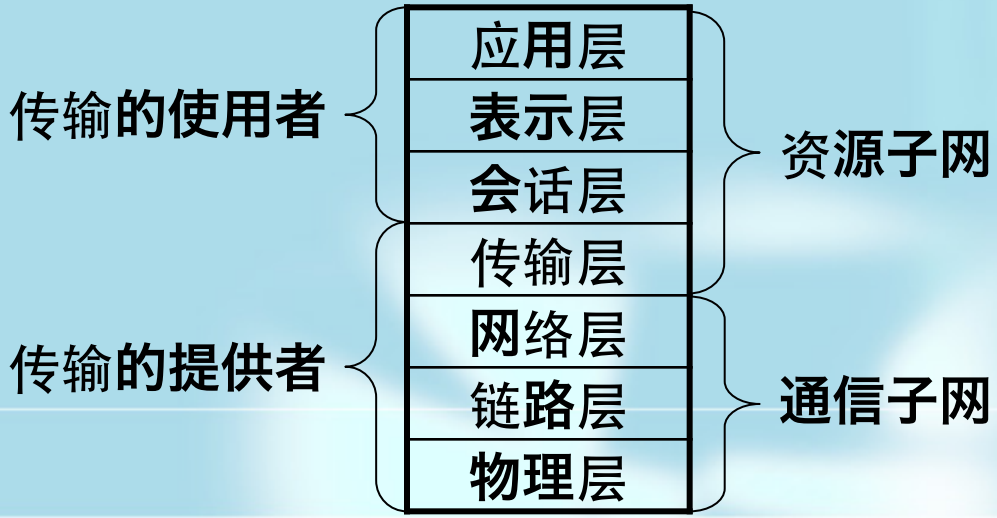
第6章 传输层

- 传输服务
- 传输协议的要素
- Internet的传输协议

传输服务

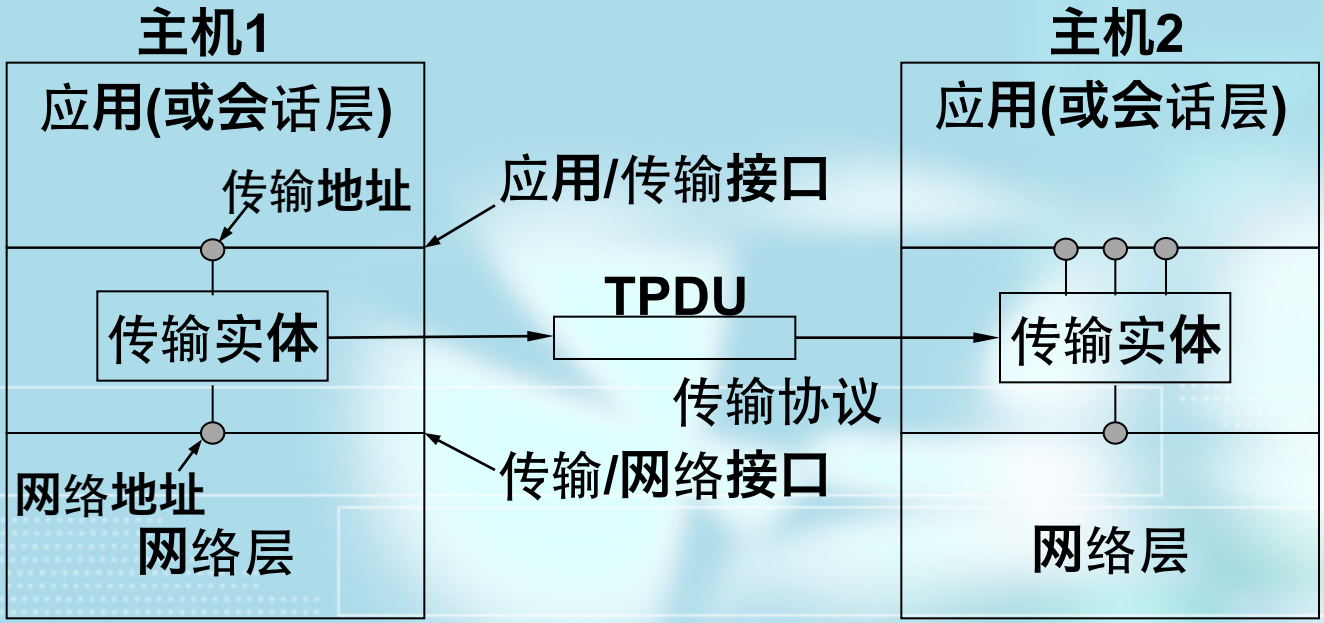
- 传输层的功能及在协议层中的作用
- 传输层提供的服务
- 传输服务原语

传输层在OSI模型中的位置



- 介于通信子网和资源子网之间，对高层用户屏蔽了通信的细节
- 弥补了通信子网所提供服务的差异和不足，提供端到端之间的无差错保证
- 传输层工作的简繁取决于通信子网提供服务的类型

传输层与上下层之间的关系



Tnbm P482 Fig. 6-1 网络层、传输层和应用层

- 传输层使高层用户看见的好象就在两个传输层实体之间有一条端到端的、可靠的、全双工的通信通道（即数字管道）

传输服务

- 传输层的功能及在协议层中的作用
- 传输层提供的服务
- 传输服务原语

传输层提供的服务

- 面向连接的服务：通信可靠，对数据有校验和重发

  如TCP/IP模型中应用层协议FTP、Telnet等

- 面向非连接的服务：对数据无校验和重发，通信速率高

  如TCP/IP模型中应用层协议SNMP、DNS等

传输服务

- 传输层的功能及在协议层中的作用
- 传输层提供的服务
- 传输服务原语

传输服务原语是应用程序和传输服务之间的接口

- 一个典型的面向连接的服务原语

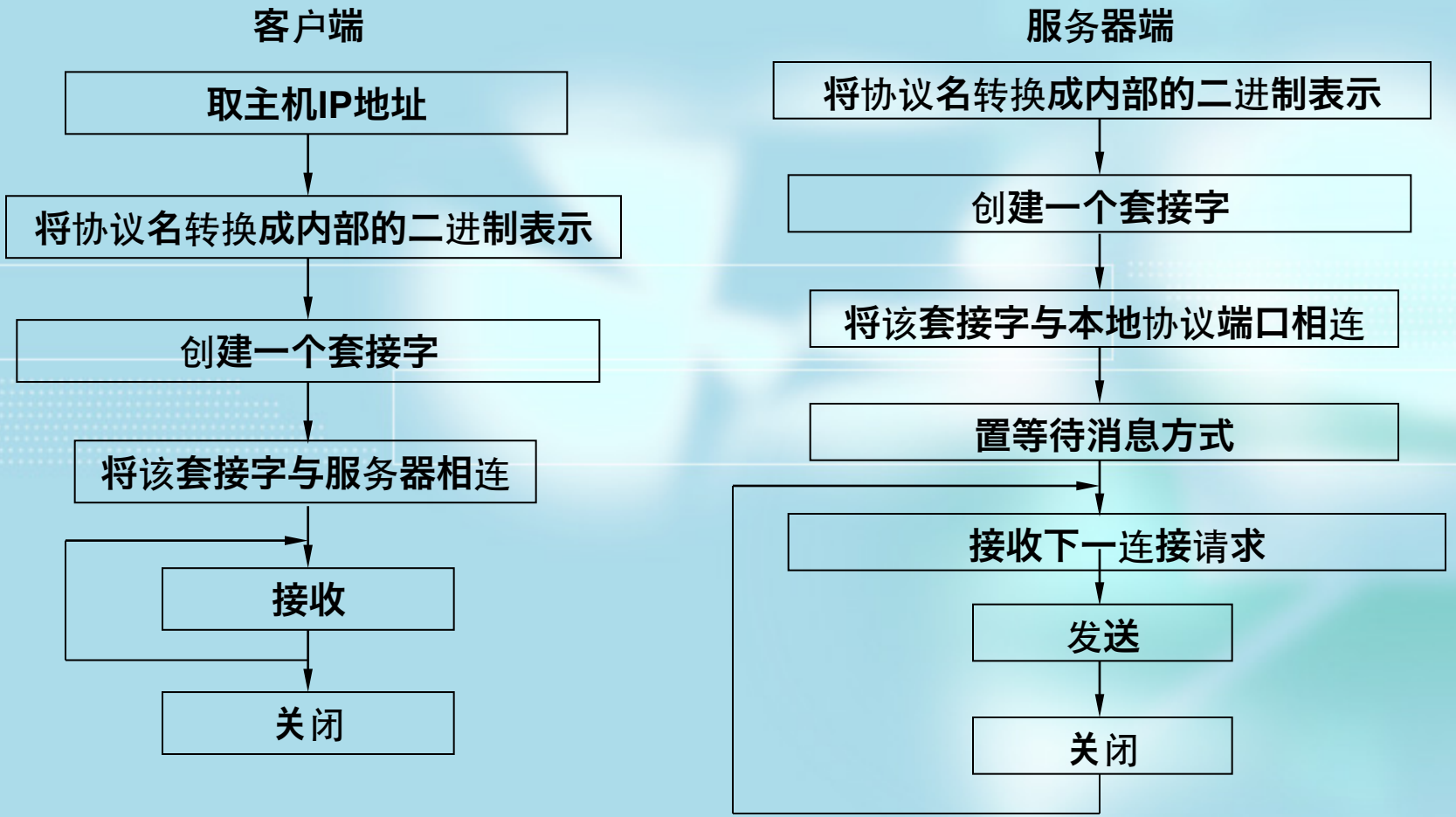
原　语	TPDU发送的	含　　义
LISTEN	(无)	阻塞，直到某个过程试图连接
CONNECT	CONNECTION REQ	建立一个连接的活动尝试
SEND	DATA	发送信息
RECEIVE	(无)	阻塞，直到一个DATA TPDU到达
DISCONNECT	DISCONNECTION REQ	该方希望释放连接

伯克利套接字（Berkeley Sockets）

原 语	含 义
SOCKET	创建一个新的通信端点
BIND	将本地地址关联到套接字上
LISTEN	宣布愿意接受连接，给出队列大小
ACCEPT	阻塞呼叫者，直到连接尝试到达
CONNECT	主动建立连接的尝试
SEND	在指定连接上发送数据
RECEIVE	从指定连接中接收数据
CLOSE	释放指定的连接

典型的套接字应用过程

- 套接字的使用与文件的使用类似



第6章 传输层

- 传输服务
- 传输协议的要素
- Internet的传输协议

传输协议的要素

❖ 传输层与数据链路层的比较

相同点：
- 可靠的数据传输

不同点：
- 数据链路层通过物理通道直接通信，而在传输层，其面对的传输通道是一个网络
- 数据链路层的连接建立很简单，而传输层要复杂得多
- 数据链路层的通信是点对点的，每条输出线对应了唯一的一个设备，而传输层则需要给出目的端地址
- 在数据链路层无中间存储环节，而在传输层，每一途经的路由器都必须存储、寻径、转发，而寻径到转发的时间随路由器本身的性能和路由算法而定
- 数据链路层通常使用一对发送缓冲区和接收缓冲区，而在传输层，对每个连接都必须分配一定的缓冲区，其缓冲区的管理将复杂得多

传输层必须讨论：

- 寻址
- 连接建立
- 释放连接
- 流量控制和缓冲策略
- 多路复用
- 崩溃的恢复

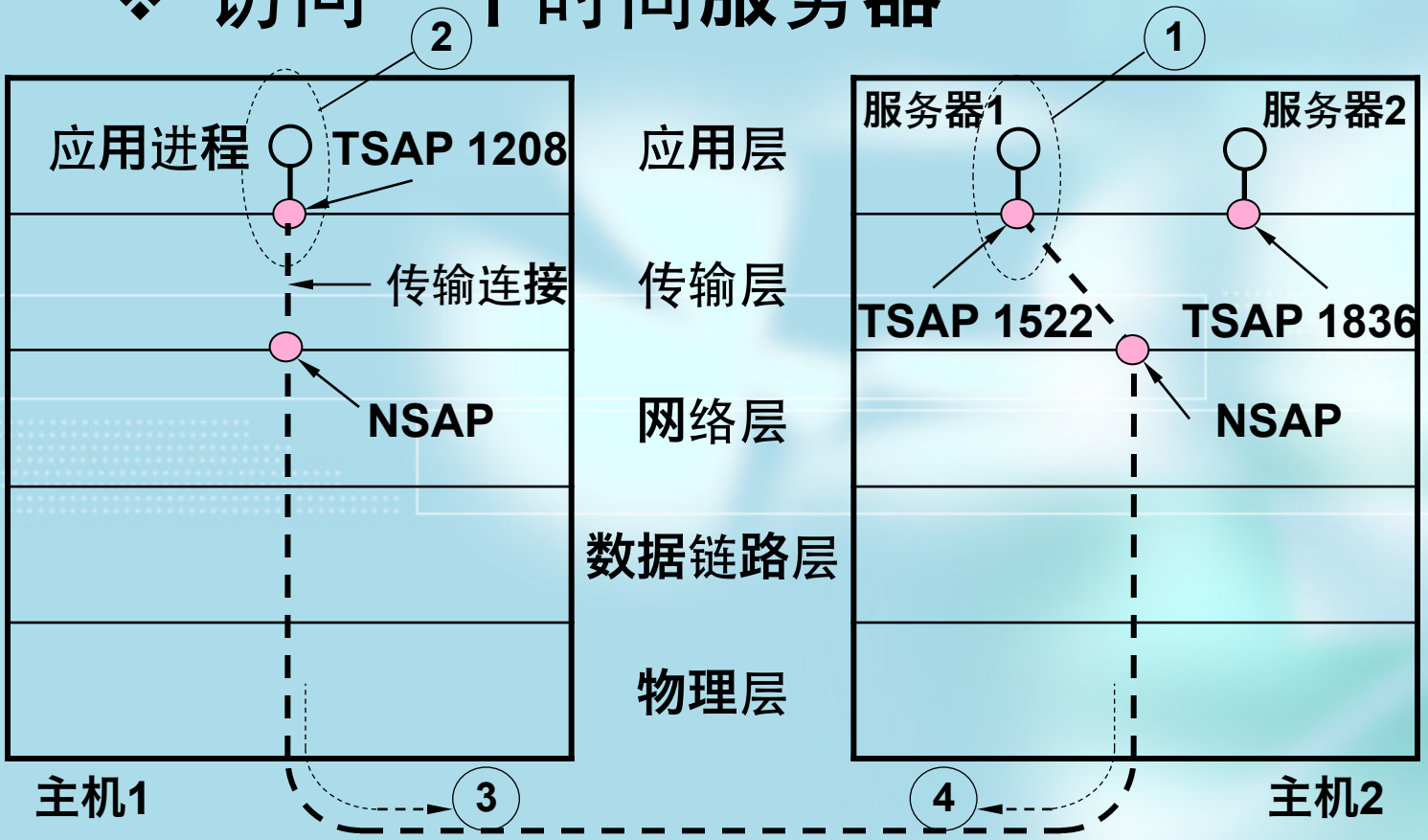
传输服务访问点TSAP

（Transport Service Access Point）

- 两个程序要建立连接时，必须指明对方是哪一个应用程序，这个标记称为传输层地址，也称为传输服务访问点（TSAP）
  - 在TCP协议中传输层地址即TCP的端口号
  - 网络层地址称为网络服务访问点NSAP（Network Service Access Point），NSAP在IP协议中即IP地址

连接方案举例

- 访问一个时间服务器



Tnbm P494 Fig. 6-8 TSAP、NSAP和传输连接

访问一个时间服务器的说明

1. 主机2上的时间服务进程将自己连到1522号TSAP上，等待即将到来的请求，例如，可以用LISTEN调用
2. 主机1上的一个应用进程想找出当天的时间，便发出一个CONNECT请求，将1208号TSAP设定为源地址，将1522号TSAP设定为目的地址
3. 主机1上的应用进程发送一个时间请求
4. 主机2上的服务器进程以当前的时间作为应答
5. 传输连接释放

如何知道对方的TSAP

- well-known TSAP

  每个服务都有自己固定的TSAP，所有网络用户都知道

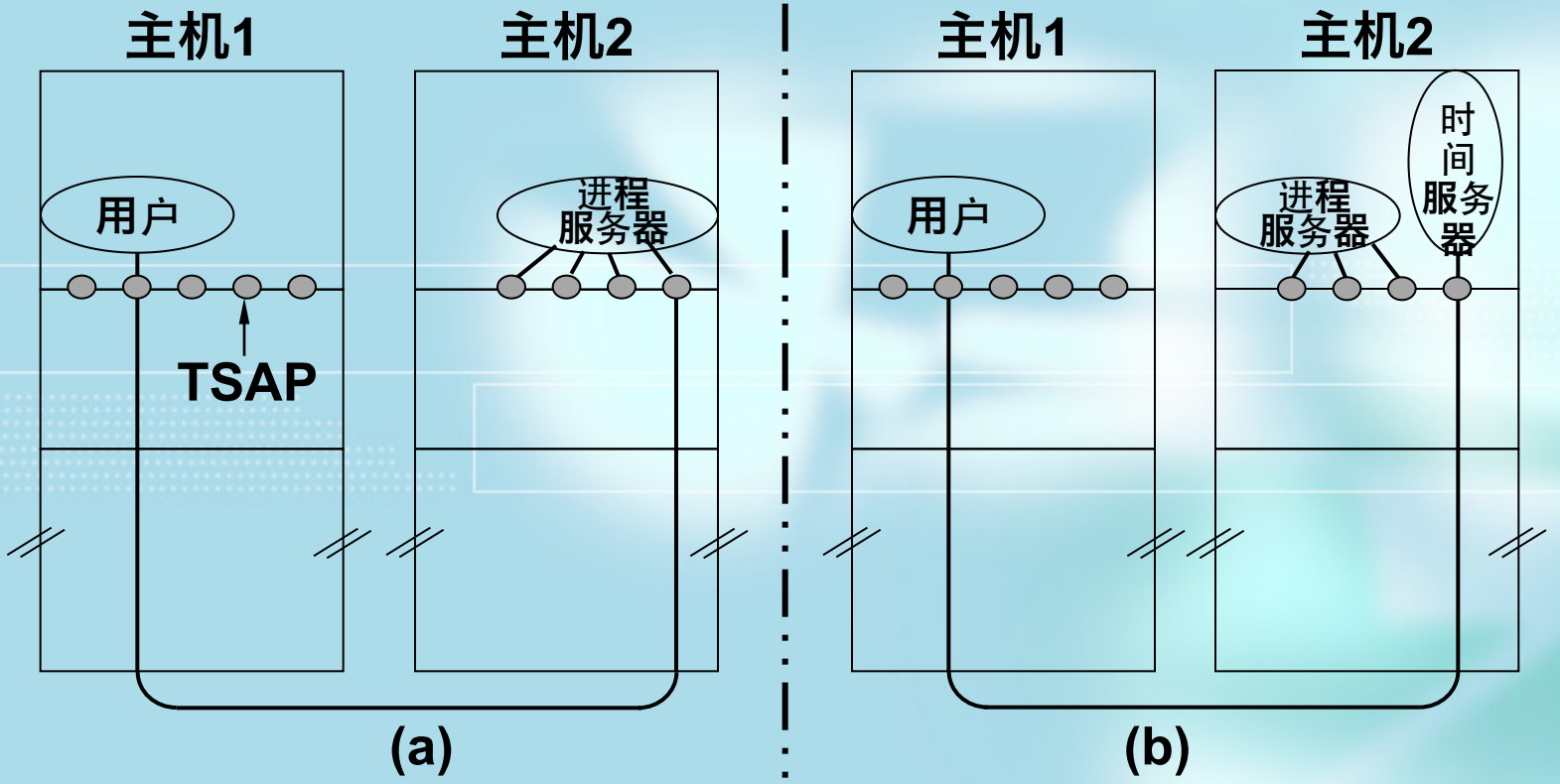
- 采用名字服务器（name server）或目录服务器（directory server）

  用户与名字服务器建立连接，向服务器发送一个报文，指明服务的名称，服务器将该服务对应的TSAP返回给用户，类似于114查号

- 服务器方将分配的TSAP通知主机

服务器方将分配的TSAP通知主机

- 初始连接协议（initial connection protocol）



Tnbm P496 Fig. 6-9 主机1的用户进程如何与主机2的时间服务器建立连接

初始连接协议说明

1. 服务器（主机2）上运行一个特殊的进程服务器，同时侦听一系列的端口，等待用户的TCP连接请求
2. 用户（主机1）设定所需服务的TSAP地址，发出一个CONNECT请求
3. 主机2的进程服务器在收到请求后，便将请求装入被请求服务器，被请求服务器将继承已建立的连接并为用户提供服务
4. 主机2的进程服务器又继续侦听新的请求

传输层必须讨论：

- 寻址
- 连接建立
- 释放连接
- 流量控制和缓冲策略
- 多路复用
- 崩溃的恢复

连接建立

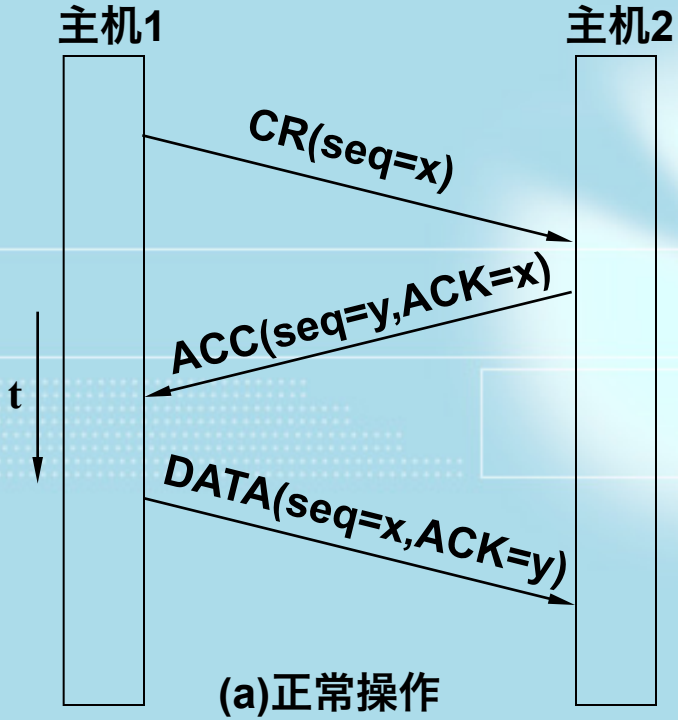
- 通信子网的不可靠性
- 通信子网中存在着延时和分组的丢失，以及由于延时和丢失而带来的重复分组
- 由于通信子网的尽力而为的传输原则，一个早已超时的分组最终还是到达了目的端，所以有必要将分组的生命周期限制在一个适当的范围内
- 连接建立时，如何处理过期分组，保证连接的唯一性是连接建立过程中首要考虑的问题
- 常用的方法是：三次握手法

连接建立过程

- 正常的三次握手过程
- 非正常的连接建立过程
  - 由延迟而重复的TPDU的连接过程
  - 同时出现作废的CR和ACC的情况

正常的连接建立过程

- 正常连接的三次握手过程



(a)正常操作

主机1发出的连接请求序号为x (seq=x)，主机2应答接受主机1的连接请求，并声明自己的序列号为y (seq=y,ACK=x)，主机1收到确认后发送第一个数据TPDU并确认主机2的序列号(seq=x,ACK=y)，至此，整个连接建立过程正常结束，数据传输已正式开始

CR：Connection Request（连接请求）
ACC：Connection Accepted（接受连接）

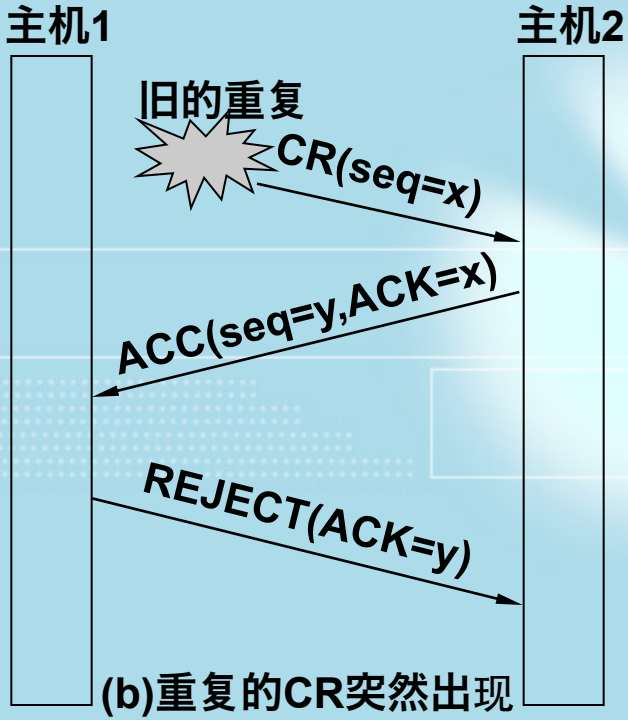
Tnbm P501 Fig. 6-11
采用三次握手方法建立的正常情况

连接建立过程

- 正常的三次握手过程
- 非正常的连接建立过程
  - 由延迟而重复的TPDU的连接过程
  - 同时出现作废的CR和ACC的情况

非正常的连接建立过程1

- 出现延迟的重复TPDU时三次握手的工作过程



Tnbm P501 Fig. 6-11
采用三次握手建立的第二种情况

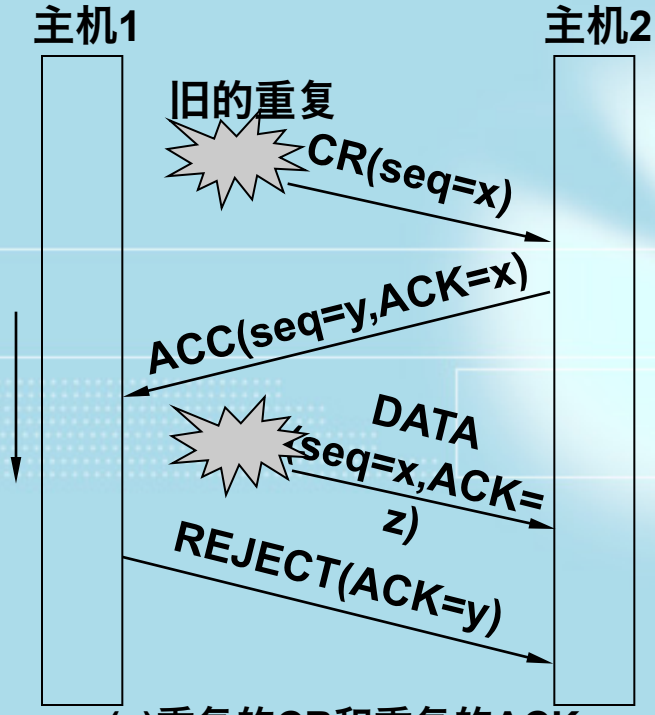
来自一个已经释放连接的主机1的延时重复的连接请求，该TPDU在主机1毫不知晓的情况下到达主机2，主机2通过向主机1发送一个接受连接请求的TPDU来响应该TPDU，并声明自己的序号为y(seq=y,ACK=x)，主机1收到这个确认后感到莫名其妙并当即拒绝，主机2收到了主机1 的拒绝才意识到自己受到了延时的重复TPDU的欺骗并放弃该连接，据此，延时的重复请求将不会产生不良后果

连接建立过程

- 正常的三次握手过程
- 非正常的连接建立过程
  - 由延迟而重复的TPDU的连接过程
  - 同时出现作废的CR和ACC的情况

非正常的连接建立过程2

- 子网中同时有作废的CR和ACC的情况



(c)重复的CR和重复的ACK

Tnbm P501 Fig. 6-11

采用三次握手建立的第三种情况

与上例一样，主机2收到了一个延时的CR并做了确认应答，在这里，关键是要认识到主机2已经声明使用y作为从主机2到主机1进行数据传输的初始序号，因此主机2十分清楚在正常情况下，主机1的数据传输应捎带对y确认的TPDU，于是，当第二个延时的TPDU到达主机2时，主机2根据它确认的是序号z而不是y知道这也是一个过时的重复TPDU，因此也不会无故建立无人要求的连接

传输层必须讨论：

- 寻址
- 连接建立
- 释放连接
- 流量控制和缓冲策略
- 多路复用
- 崩溃的恢复

释放连接

- **非对称释放**

  一方中止连接，则连接即告中断

  缺陷：可能导致数据丢失

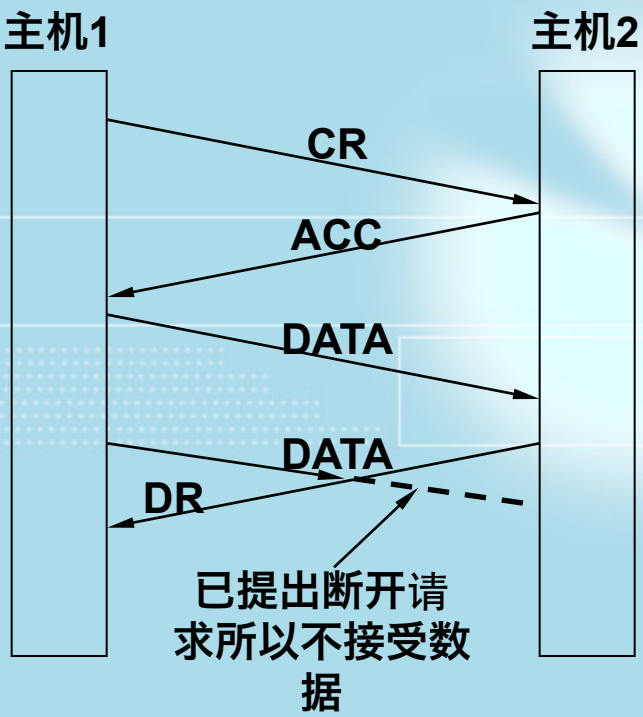
- **对称释放**

  A提出中止请求，B同意即中止

  问题：B如何知道A 收到了它的确认

非对称释放

- 一方中止连接，则连接即告中断



Tnbm P502 Fig. 6-12

突然释放连接将造成数据丢失

当连接建立后，主机1发送了一个数据TPDU并正确抵达主机2，接着，主机1发送了第二个数据TPDU，然而，主机2在收到第二个TPDU之前先突然发出了释放连接请求DISCONNECT，结果是连接立即被释放，数据被丢失

释放连接

- **非对称释放**

  一方中止连接，则连接即告中断

  缺陷：可能导致数据丢失

- **对称释放**

  A提出中止请求，B同意即中止

  问题：B如何知道A 收到了它的确认？

对称释放

- A提出中止请求，B同意即中止
  - 对称释放方式适用于每个用户进程有固定数量的数据需要发送，而且清楚地知道何时发送完毕的情况
  - 其他情况下，决定所有工作是否已经完成，连接是否应该释放，可能是没有把握的
  - 可以假想一种完美的协议：

    A说：“我发送完了，你呢?”

    如果B响应：“我也发送完了，再见，”

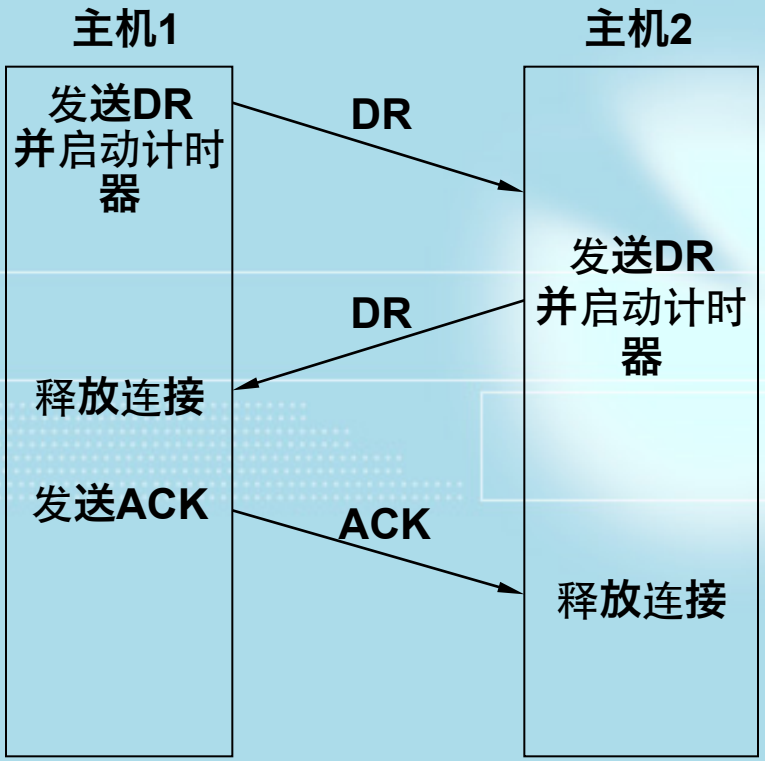
    A收到了B的确认，连接便可以被安全释放

对称释放的几种情况

- 三次握手的正常情况
- 最后的确认TPDU丢失
- 应答丢失
- 应答丢失以及后续的DR丢失

对称释放1

- 三次握手的正常情况



Tnbm P505 Fig. 6-14(a)
三次握手的连接正常释放情况

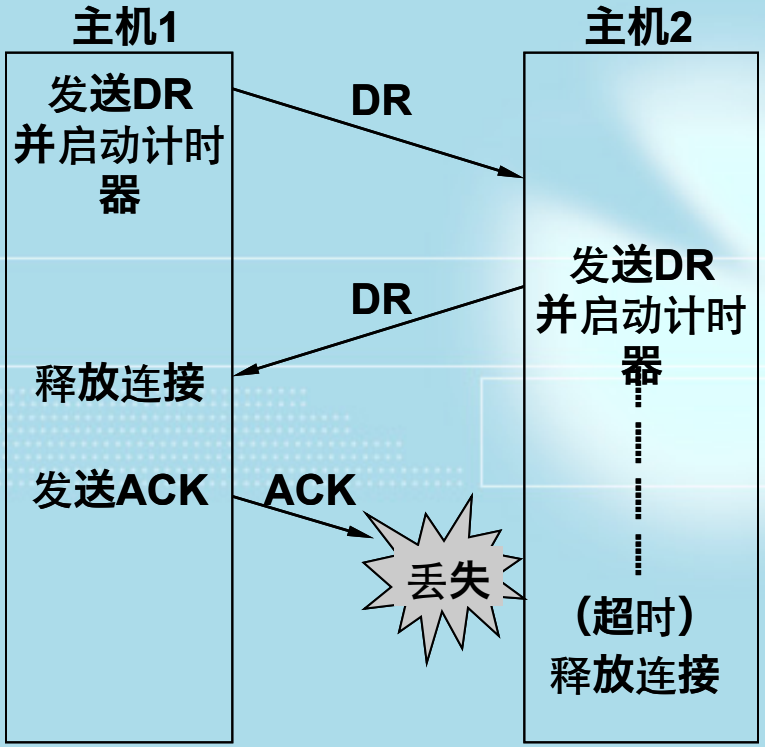
主机1在结束数据传输后决定释放连接，于是发送DR并启动计时器，主机2在收到主机1 的DR后同意释放连接，也发送DR并启动计时器，主机1 在计时器没有超时前收到主机2 的DR，便正式释放连接并发送ACK，主机2也在计时器没有超时前收到主机1 的ACK，于是也释放了连接，至此整个数据传输过程，包括建立连接、传输数据和释放连接的过程正常结束

对称释放的几种情况

- 三次握手的正常情况
- 最后的确认TPDU丢失
- 应答丢失
- 应答丢失以及后续的DR丢失

对称释放2

- 最后的确认TPDU丢失



Tnbm P505 Fig. 6-14(b)
最后的确认TPDU丢失的情况

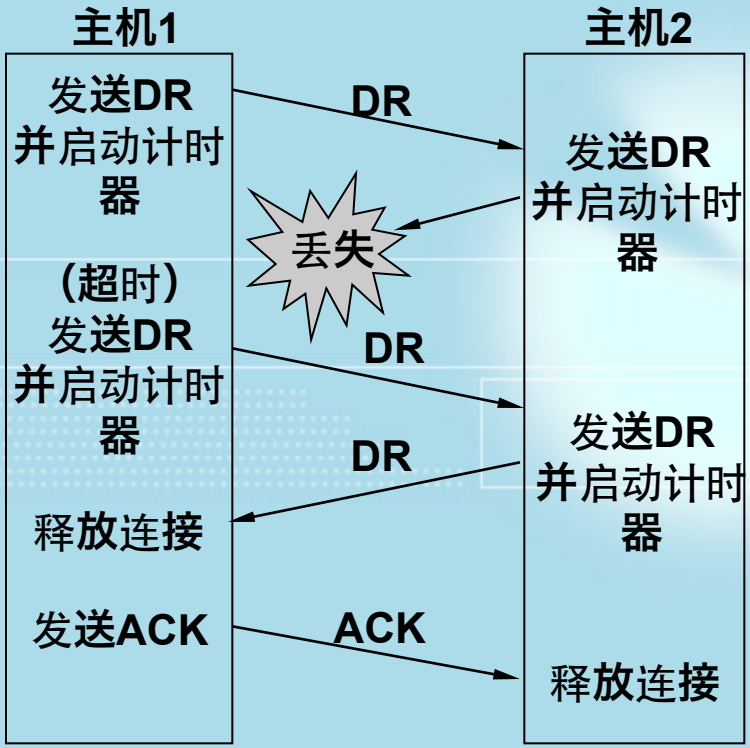
主机1在结束数据传输后决定释放连接，于是发送DR并启动计时器，主机2在收到主机1 的DR后同意释放连接，也发送DR并启动计时器，主机1 在计时器没有超时前收到主机2 的DR，便正式释放连接并发送ACK，然而主机2在计时器超时后还未收到主机1 的ACK，但是由于已经超时，于是也释放了连接

对称释放的几种情况

- 三次握手的正常情况
- 最后的确认TPDU丢失
- 应答丢失
- 应答丢失以及后续的DR丢失

对称释放3

❖ 应答丢失



Tnbm P505 Fig. 6-14(c)

应答丢失的情况

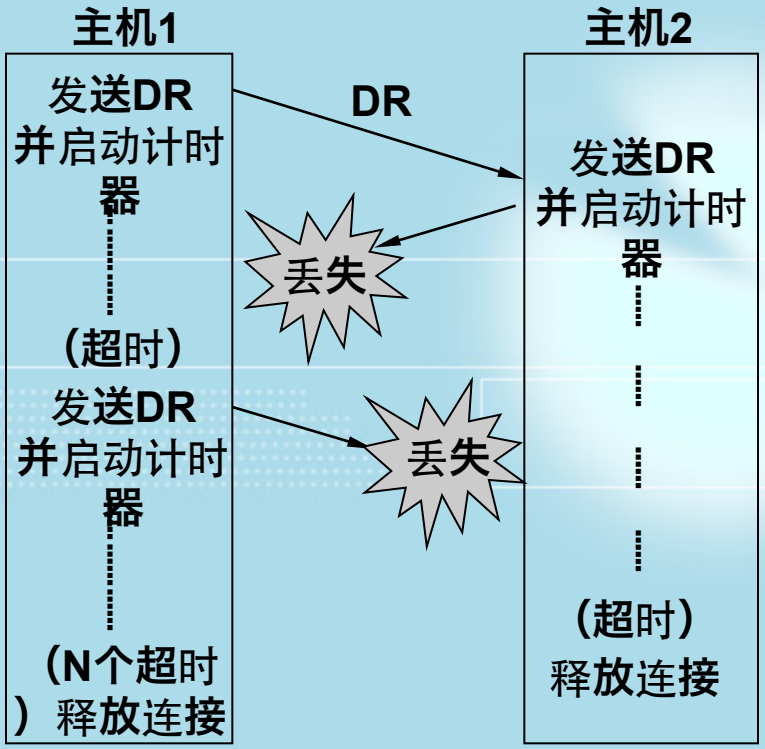
主机1在结束数据传输后决定释放连接，于是发送DR并启动计时器，主机2在收到主机1 的DR后同意释放连接，也发送DR并启动计时器，然而，主机1 在计时器超时后还未收到主机2 的DR，于是又重新发送DR并启动计时器，下面便是一个正常的三次握手，并最后正常释放连接，即整个数据传输过程正常结束

对称释放的几种情况

- 三次握手的正常情况
- 最后的确认TPDU丢失
- 应答丢失
- 应答丢失以及后续的DR丢失

对称释放4

- 应答丢失以及后续的DR丢失



Tnbm P505 Fig. 6-14(d)
应答丢失以及后续的DR丢失

主机1在结束数据传输后决定释放连接，于是发送DR并启动计时器，主机2在收到主机1 的DR后同意释放连接，也发送DR并启动计时器，然而，紧接着的一段时间内，线路遇到了灾难性的干扰，无论是哪一方的超时重发的TPDU都不能到达对方，最终，接收方计时器的超时而也释放连接，发送方经过n次重发和超时后只能无奈地放弃努力并释放连接

传输层必须讨论：

- 寻址
- 连接建立
- 释放连接
- 流量控制和缓冲策略
- 多路复用
- 崩溃的恢复

流量控制和缓冲策略

- 流量控制是发送方和接收方之间的传输速率上的匹配，为使没有得到确认的PDU在超时后的重发，通常必须在缓冲区中暂存
  - 在数据链路层，实现的是点对点的通信，双方缓冲区的大小根据滑动窗口协议而定
  - 而传输层实现的是端到端的通信，某一时刻，一台主机可能同时与多台主机建立了连接，多个连接必须有多组缓冲区，所以缓冲区的动态分配和管理策略与数据链路层相比较要复杂得多

流量控制和缓冲策略举例

TPDU序号	主机A	消息	主机B	注　释
1	→	<请求8个缓冲区>	→	A想要8个缓冲区
2	←	<ack=15,buf=4 >	←	B只有4个缓冲区
3	→	<seq=0,data=m0>	→	A发送了m0，A剩下3个缓冲区
4	→	<seq=1,data=m1>	→	A发送了m1，A剩下2个缓冲区
5	→	<seq=2,data=m2>	✗	A发送了m2，A剩下1个缓冲区
6	←	<ack=1,buf=3>	←	B确认了m0,m1，并剩下3个缓冲区
7	→	<seq=3,data=m3>	→	A发送了m3，A剩下1个缓冲区
8	→	<seq=4,data=m4>	→	A发送了m4后剩下0个缓冲区，暂停
9	→	<seq=2,data=m2>	→	m2超时A重发m2，A剩下0个缓冲区
10	←	<ack=4,buf=0>	←	B确认了m4，并剩下0个缓冲区
11	←	<ack=4,buf=1>	←	B确认了m4，并剩下1个缓冲区
12	←	<ack=4,buf=2>	←	B确认了m4，B有了2个缓冲区
13	→	<seq=5,data=m5>	→	A发送了m5，A剩下1个缓冲区
14	→	<seq=6,data=m6>	→	A发送了m6后剩下0个缓冲区，暂停
15	←	<ack=6,buf=0>	←	B确认了m6，并剩下0个缓冲区
16	✗	<ack=6,buf=4>	←	B确认了m6，并剩下4个缓冲区

B没有收到m2

表示上交了1个

A没有收到ACK 可能导致死锁

Tnbm P509 Fig. 6-16 动态缓冲区的分配

传输层必须讨论：

- 寻址
- 连接建立
- 释放连接
- 流量控制和缓冲策略
- 多路复用
- 崩溃的恢复

多路复用

- **向上多路复用**

  多个传输层的连接共用一个网络层的连接，将提高网络层连接的利用率

- **向下多路复用**

  一个传输层的连接合用多个网络层的连接来发送，可增加其有效带宽，以提高传输速率

传输层必须讨论：

- 寻址
- 连接建立
- 释放连接
- 流量控制和缓冲策略
- 多路复用
- 崩溃的恢复

崩溃的恢复

- 网络崩溃的恢复
- 主机崩溃的恢复
- 确认和写操作的问题

网络崩溃的恢复

- **数据报子网：如果传输层对丢失的TPDU留有副本，可以通过重发来解决**
- **虚电路子网：必须重新建立连接，并询问远端的传输实体哪些TPDU已经收到，没有收到的则必须重发**

崩溃的恢复

- 网络崩溃的恢复
- 主机崩溃的恢复
- 确认和写操作的问题

主机崩溃的恢复

- 客户端主机崩溃，恢复将较为简单
- 服务器崩溃，如能及时重新起动，则如何恢复与原客户主机的数据传输

重新连接后，客户端可能处于两种状态之一：

S1—有一个未被确认的TPDU

S0—没有未被确认的TPDU

在一般情况下，远端服务器的传输实体在接收到一个客户端的TPDU后，先发送一个确认，再对应用进程执行一个写操作（如存盘或交上层）。发送一个确认和执行一个写操作，是两个不同的而又不可分的事件，但两者不能同时进行

崩溃的恢复

- 网络崩溃的恢复
- 主机崩溃的恢复
- 确认和写操作的问题

确认和写操作的问题

- 先发送确认，然后再执行写操作，中间发生崩溃

  此时客户端将收到这个确认，当崩溃恢复声明到达时它处于状态S0，客户端将因此不再重发，因为它错以为那个TPDU已经到达服务器端，客户端的这种决定会导致丢失一个TPDU

- 先执行写操作，然后再发送确认，中间发生崩溃

  设想已经完成了写操作但在确认发出前系统发生了崩溃，此时客户端将处于状态S1并因此重传数据，从而会导致在服务器应用进程的输出流上出现一个无法检测的重复的TPDU

无论怎样对发送方和接收方的协议进行编程，总是存在协议不能正确地从故障中恢复的情况，传输层无法彻底解决该问题，将由高一层协议处理

第6章 传输层

- 传输服务
- 传输协议的要素
- **Internet的传输协议**

Internet的传输协议

- TCP和UDP都是Internet提供的传输层协议
  - UDP (User Datagram Protocol)
  - TCP (Transmission Control Protocol)

UDP (User Datagram Protocol)

- UDP简介
- UDP提供服务的应用
  - RPC（Remote Procedure Call）
  - RTP（Real-Time Transport Protocol）

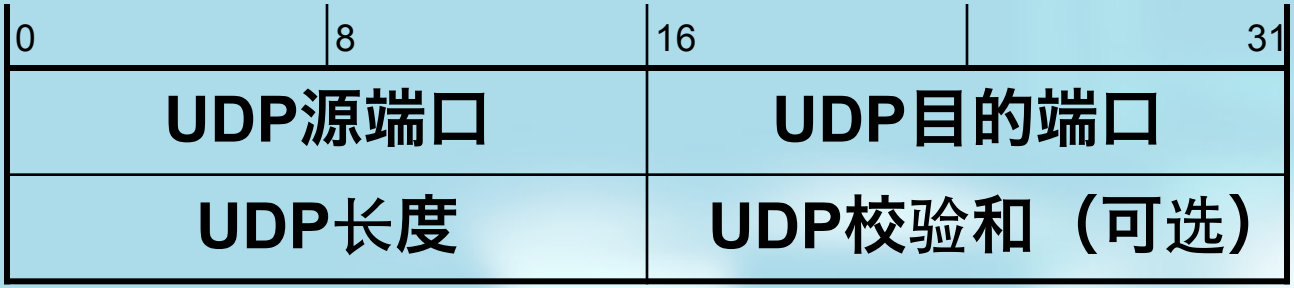
UDP简介

- UDP协议是无连接的数据报协议，它不提供可靠性服务，但其相应的协议开销也较小、效率较高
- 域名系统DNS、简单网络管理协议SNMP使用的传输层协议是UDP

UDP提供的服务

- UDP是面向非连接的，与IP协议相比，它唯一增加的能力是提供端口协议，以保证进程通信，传输的可靠性要靠应用程序来解决，但效率高

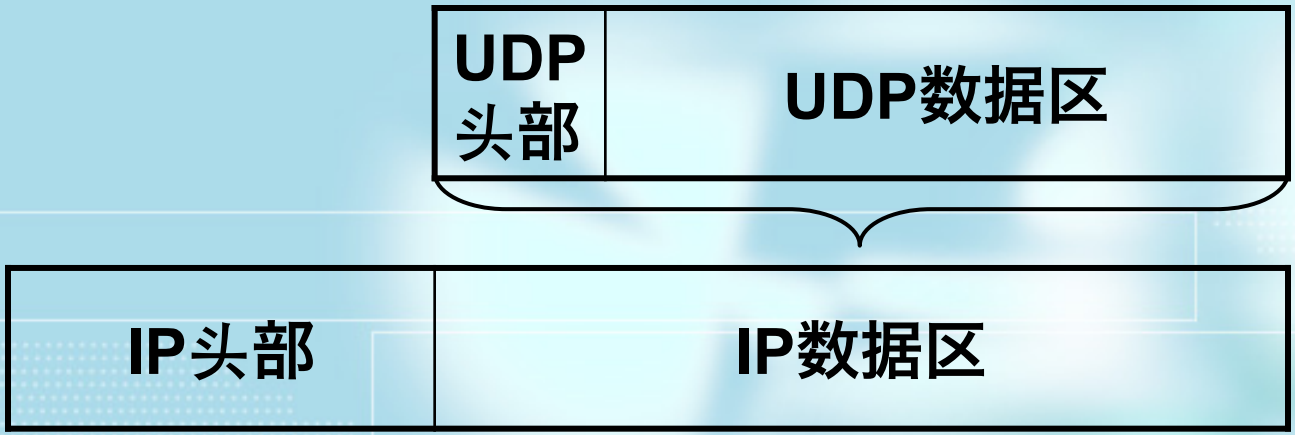
UDP的数据报格式



Tnbm P526 Fig. 6-23 UDP的头部格式

- UDP源端口：UDP端口号，当不需要返回数据时，源端口域置0
- UDP目的端口：UDP端口号
- UDP长度：整个数据段的长度，包括头部和数据部分以字节计，最小值为8（仅头部长度）
- UDP校验和：可选域，全0为未选，全1表示校验和为0

UDP数据段的封装



UDP (User Datagram Protocol)

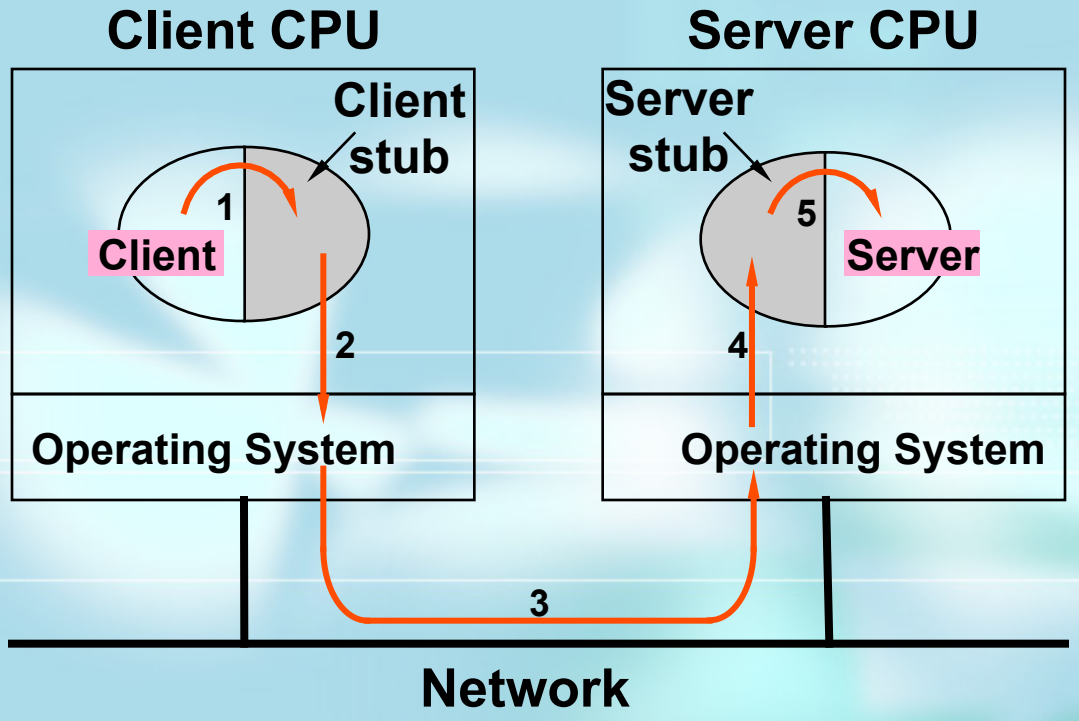
- UDP简介
- UDP提供服务的应用
  - RPC（Remote Procedure Call）
  - RTP（Real-Time Transport Protocol）

远程过程调用RPC

- 将网络中的请求-应答交互表示成过程调用形式，例如：调用get-IP-address（主机名）将发送一个UDP包给DNS服务器，并等待回答
- RPC对程序员屏蔽了网络运作的细节
- RPC是UDP的一个重要应用

一次RPC的过程

1. 客户调用Client stub
2. Client stub 将参数封装并发送
3. 客户端OS内核将消息发送到服务器
4. 服务器OS内核将分组交Server stub
5. 服务器取出参数并调用过程，最后将结果返回



Tnbm P528 Fig. 6-24 一次RPC过程中的步骤

Stub是一个程序模块，用于在客户机和服务器之间传输远程过程调用和响应的承接程序

RPC过程的说明

1. 客户机通过在本地调用Client stub，使用常规的方法将参数压栈
2. Client stub把参数封装（marshal）成消息并通过系统调用进行发送
3. 客户机操作系统内核通过网络发送消息到服务器
4. 服务器操作系统内核通过网络从客户机接收消息
5. Server stub调用服务器进程对参数进行拆封
6. 应答即为同一路径的逆向过程

RPC的应用

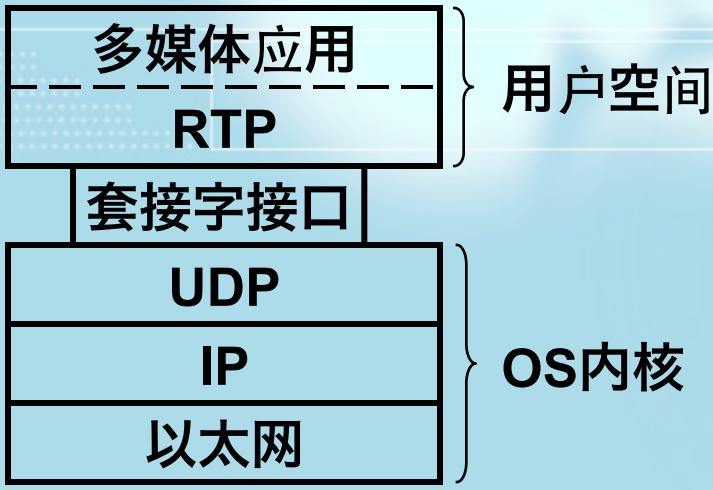
- RPC是客户机/服务器模式中实现分布式计算的通信协议
- RPC采用非连接对话方式，具有错误控制能力
- RPC使服务器系统使用客户机提供的参数，执行客户机指定的规程并返回结果
- 从逻辑上看，按OSI的七层构架，RPC属会话层协议，但有些功能属应用层

UDP (User Datagram Protocol)

- UDP简介
- UDP提供服务的应用
  - RPC（Remote Procedure Call）
  - RTP（Real-Time Transport Protocol）

实时传输协议RTP

- UDP的另一个重要应用是RTP
- RTP是一个传输层协议，但在应用层实现
- RTP是用于多媒体数据传输的协议

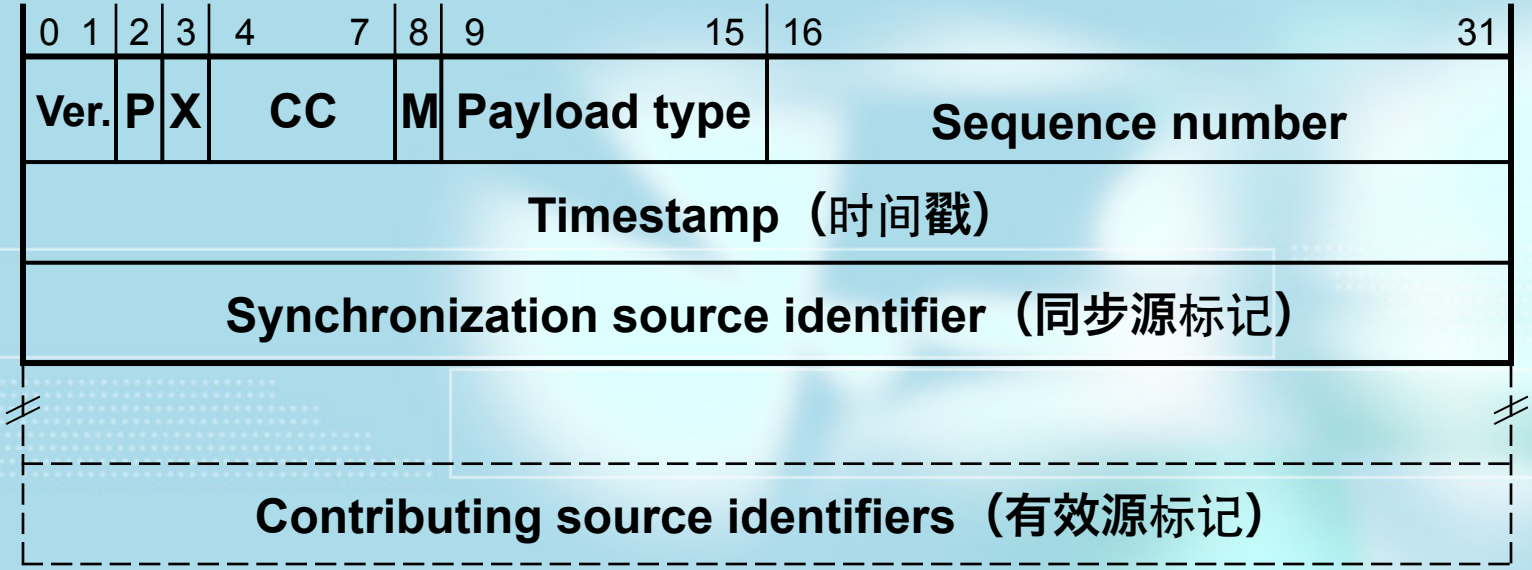


Tnbm P529 Fig. 6-25(a)
RTP在协议栈中的位置

RTP的功能

- 将多个实时数据流多路复用到一个UDP流上，UDP流能被送给某个地址（单址传输）或多个地址（多址传输）
  - 每个RTP流的数据包有一个连续的编号，接收方可以根据此编号确定是否有数据包丢失
  - RTP没有流量控制、差错控制、应答以及重传机制
  - RTP的载荷可以是不同的多媒体信息，如单个的音频流，可以是8 bit-8 kHz的PCM编码，也可以是GSM、MP3等
  - 编码方式在RTP的包头上指出
- 实时应用的另一个特征是需要时间戳，时间戳是相对于流的第一个数据包的，这有助于在接收方消除抖动，以及多个流的同步（如数字电视中的视频流和两个音频流）

RTP的头部格式



Tnbm P531 Fig. 6-26 RTP的头部格式

RTP的头部字段

- P：指出数据包是否被填充为4字节的整倍数
- X：是否有扩展头
- CC：有多少个有效源（0-15）
- M：应用指定的标记位，如表示video帧的开始、音频通道中文字的开始或其它可理解的应用
- 载荷类型：信息的编码方式（如非压缩的8 bit音频、MP3等）
- 顺序号：RTP包的序号，以此检查是否丢包
- 时间戳：由发送源产生，表示与第一个包的时间间隔
- 同步源标记：数据包属于哪个流，用于多路复用或解多路复用
- 有效源标记：用于混合数据源，如果该字段出现，则该混合源是同步数据源，每个分数据源被列在这里

RTP的姐妹协议RTCP

- RTCP（Realtime Transport Control Protocol）实时传输控制协议
- RTCP处理反馈、同步和用户接口，而不传输数据
- 反馈消息包括延时、抖动、带宽、拥塞以及其它网络性能，可协调源端的传输速率、编码方式等，以适应当前的传输环境，得到最好的服务质量
- 同步是指不同的流数据可以用不同的时钟、不同的颗粒度、不同的速率进行传输，而RTCP可使其保持同步
- RTCP提供一种对不同的源端进行命名的方法，这将使接收方可在屏幕上显示当前谁在发送数据

Internet的传输协议

- TCP和UDP都是Internet提供的传输层协议
  - UDP (User Datagram Protocol)
  - TCP (Transmission Control Protocol)

TCP提供的服务

- 面向连接（Connection Orientation）
- 端到端的服务（End – to – End Communication）
- 完全可靠性服务（Complete Reliability）

  IP协议不提供可靠性服务，而TCP将在IP的基础上提供可靠性服务并保证数据发送和接收次序一致
- 全双工服务（Full Duplex Communication）
- 流接口（Stream Interface）
- 可靠的连接建立（Reliable Connection Startup）
- 完美的连接终止（Graceful Connection Shutdown）

TCP协议将讨论：

- TCP服务模型
- TCP数据段头
- TCP连接和释放管理
- TCP传输策略
- TCP拥塞控制
- TCP定时器管理

TCP服务模型

- **端口port**
  - TCP的TSAP，与某一个应用程序相关
  - 通用端口（well-known port） 端口号小于1024
  - 其他可由各主机自己定义
- **套接字socket**
  - 传输层的通信端点，它由 IP地址 + 端口号 组成

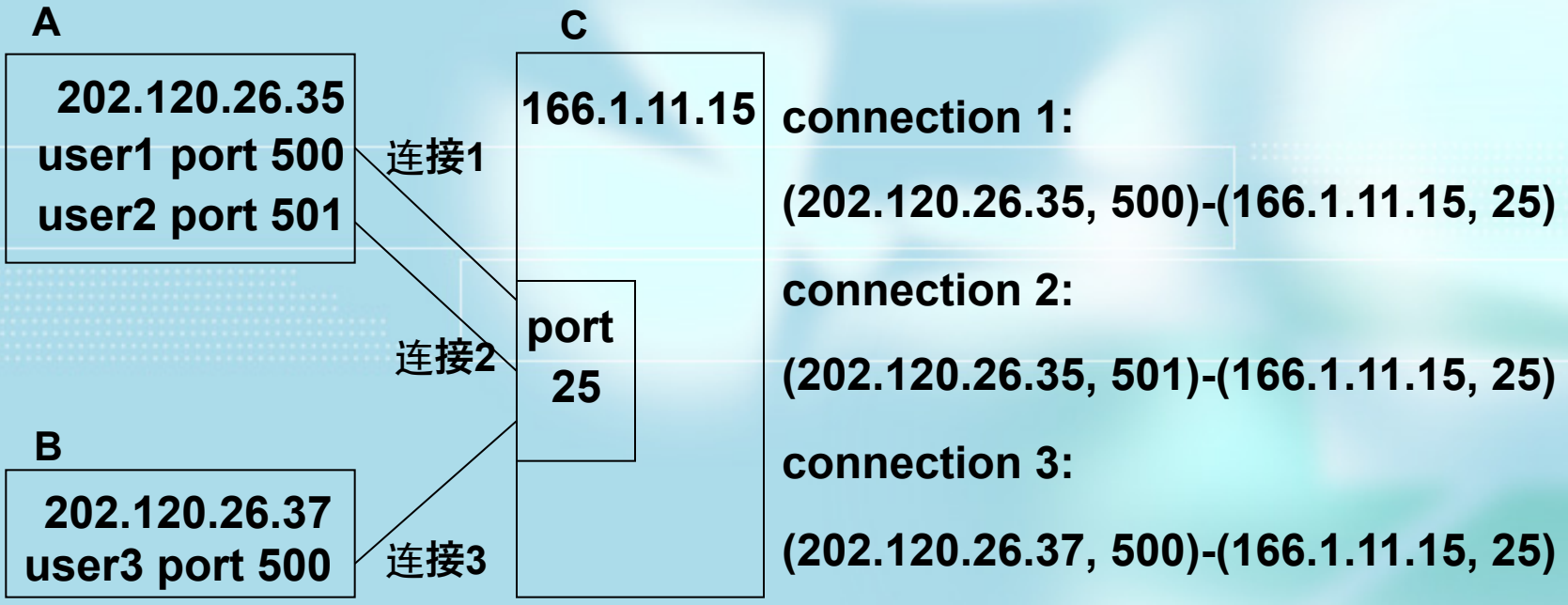
常用TCP端口表

端口号	关键字	服务说明
21	FTP	文件传输协议
23	TELNET	终端连接
25	SMTP	简单邮件传输协议
53	DNS	域名解析协议
80	HTTP	超文本传输协议
110	POP-3	远程E-mail访问

Tnbm P534 Fig. 6-27 已分配的常用的TCP端口

套接字举例

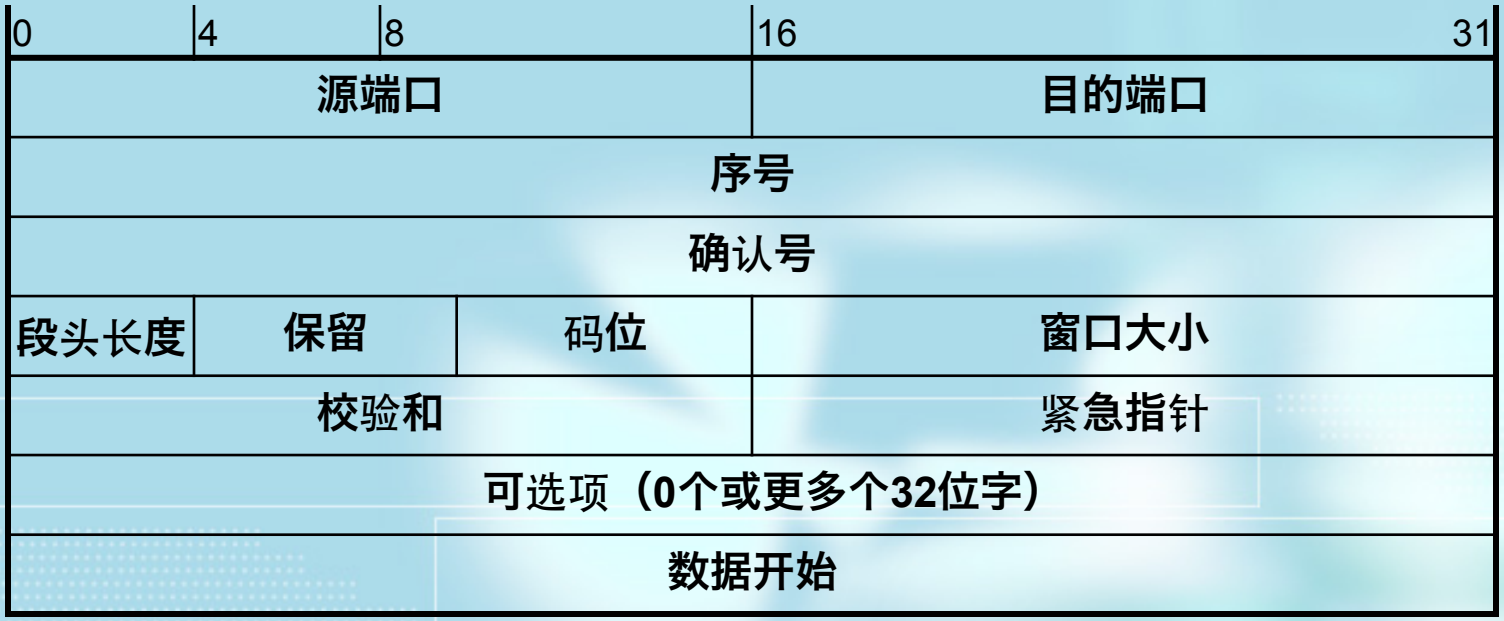
- 一个套接字允许被多个连接同时使用，即多个连接可同时连接到同一个套接字上



TCP协议将讨论：

- TCP服务模型
- TCP数据段头
- TCP连接和释放管理
- TCP传输策略
- TCP拥塞控制
- TCP定时器管理

TCP数据段头及说明



Tnbm P537 Fig. 6-29 TCP头

端口：每个端口对应一个应用程序

序号：发送的字节序号

确认号：接收到的字节序号

段头长度：段头中包含多少个32位字

TCP数据段头及说明（续）

- 保留：以备扩展之用
- 码位：

URG	ACK	PSH	RST	SYN	FIN

  URG：紧急指针有效

  ACK：确认号有效

  PSH：请求接收方，数据一到，立即送往应用程序

  RST：复位由于主机崩溃或其他原因而出现的错误的连接

  SYN：用于建立连接

  FIN：用于断开连接

- 窗口：接收方窗口大小

TCP数据段头及说明（续）

- 校验和：包括报头、数据和伪段头

Tnbm P539 Fig. 6-30 包括在TCP校验和中的伪头

- 紧急指针：当前序号到紧急位置的偏移量
- 选项字段：用于提供一种增加额外设置的方法，如连接建立时，双方说明最大的负载能力
- 数据：真正要传输的数据

常用的选项字段

- 最大数据段长度MSS

  目的站可接收的最大的数据段长度，这个值在0到65535之间，默认值是536字节

- 窗口扩大因子n（n<=14） （RFC1323）

  窗口的大小=

  首部中定义的窗口大小*2^窗口扩大因子

- 选择性重发协议（RFC1106）

引入了NAK机制

TCP协议将讨论：

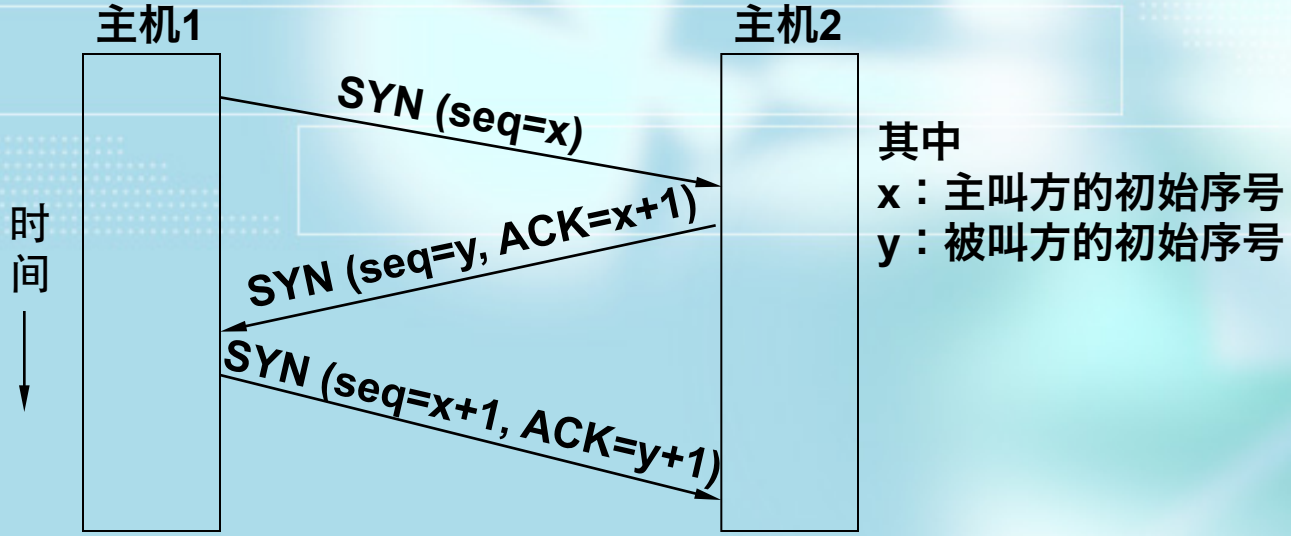
- TCP服务模型
- TCP数据段头
- TCP连接和释放管理
- TCP传输策略
- TCP拥塞控制
- TCP定时器管理

TCP连接管理

❖ TCP连接的建立

采用三次握手（3-way handshake）的方法

用TCP包头中的同步位（SYN）和结束位（FIN）描述连接建立和终止时的三次握手消息

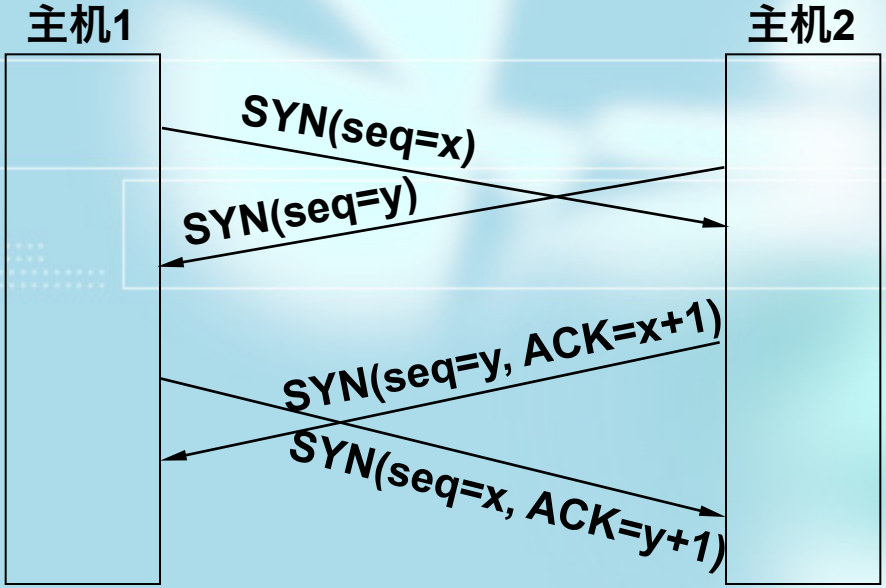


Tnbm P540 Fig. 6-31(a) 一般情况下TCP连接的建立过程

TCP连接管理

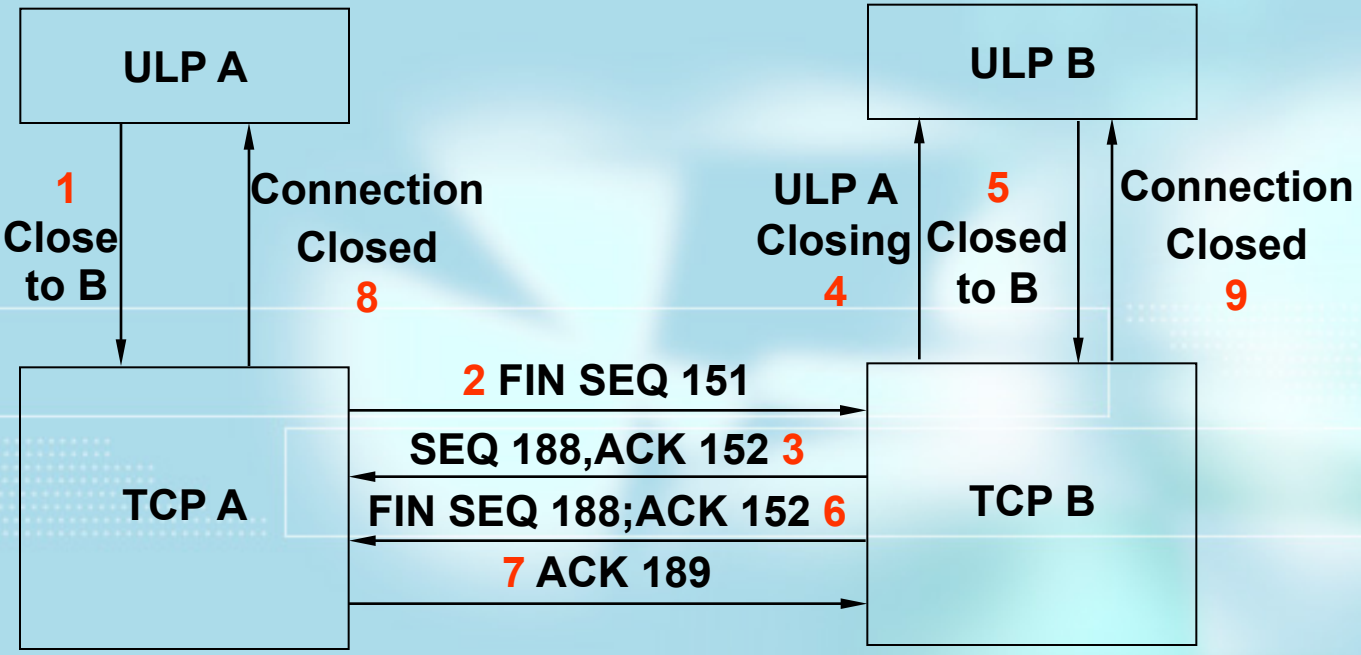
❖ TCP连接的建立（续）

连接以端地址 ~ 端地址为标识
即使两边发起同一连接，也不会建立两个连接



Tnbm P540 Fig. 6-31(b) 呼叫冲突的情况

连接释放



TCP关闭操作[7]

ULP：upper layer protocol

连接释放图说明

事件	动　　作
1	TCP A的用户打算关闭它在TCP B中对等的上层协议的操作
2	TCP A发送一个FIN位置1的段，SEQ=151是上一次操作的继续，是TCP模块要求传输的下一个序号
3	TCP B确认TCP A的FIN SEQ151，它的SEQ=188、ACK=152
4	TCP B向其用户发出一个close原语
5	TCP B用户应用程序确认同意关闭
6	TCP B再发出最后一个段，FIN位置1并SEQ=188、ACK=152同上
7	TCP A确认，ACK=189
8	TCP A连接关闭
9	TCP B连接关闭

注：握手用的报文长度仅一个字节

TCP协议将讨论：

- TCP服务模型
- TCP数据段头
- TCP连接和释放管理
- TCP传输策略
- TCP拥塞控制
- TCP定时器管理

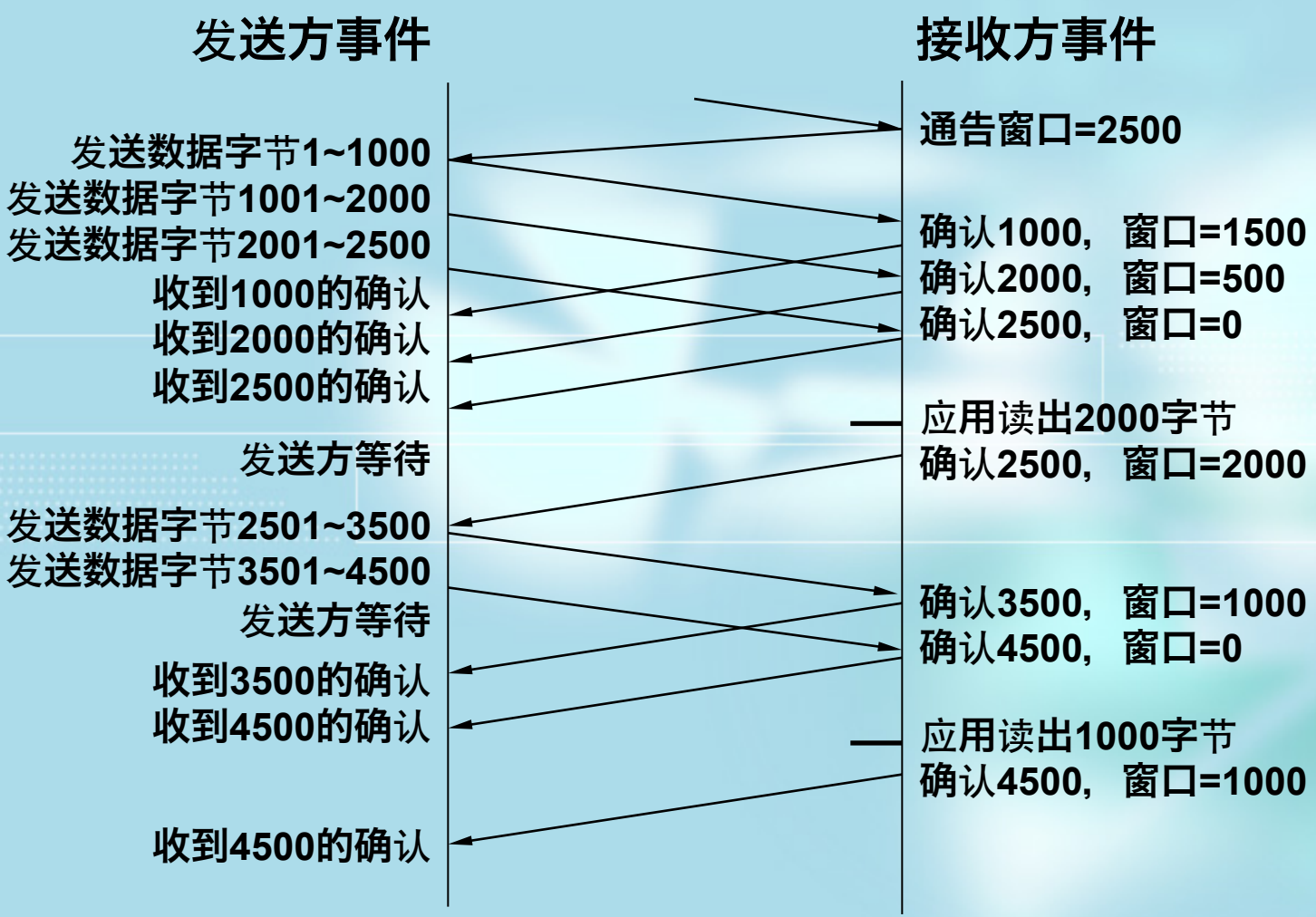
TCP传输策略

- TCP使用与数据链路层不同的窗口协议来实现流量控制
  - 数据链路层：在数据链路层的滑动窗口协议中，发送窗口的滑动依赖于确认的到达
  - TCP层：TCP的窗口大小是其缓冲区的尺寸，建立连接的双方都将分配一个缓冲区作为接收数据的存放空间，并相互通知对方，此后，每次对接收数据确认的同时发布一个窗口公告（window advertisement），报告剩余窗口的大小，任何时刻，剩余的缓冲区空间的数量叫窗口大小

零窗口公告

- 发送方收到一个零窗口公告时，必须停止发送，直到接收方重新发送一个非零的窗口公告，但有两种情况可以除外
  - 发送紧急数据，如允许用户终止（kill）当前正在远端机上运行的进程
  - 发送方可以发送一个字节的数据段通知对方，要求接收方重申希望接收的下一字节及当前的窗口大小，以防止可能的窗口公告丢失而导致的死锁

TCP层流量控制举例


发送方事件
接收方事件
通告窗口=2500
发送数据字节1~1000
发送数据字节1001~2000
发送数据字节2001~2500
确认1000，窗口=1500
确认2000，窗口=500
确认2500，窗口=0
收到1000的确认
收到2000的确认
收到2500的确认
应用读出2000字节
确认2500，窗口=2000
发送方等待
发送数据字节2501~3500
发送数据字节3501~4500
确认3500，窗口=1000
确认4500，窗口=0
发送方等待
收到3500的确认
收到4500的确认
应用读出1000字节
确认4500，窗口=1000
收到4500的确认

TCP协议将讨论：

- TCP服务模型
- TCP数据段头
- TCP连接和释放管理
- TCP传输策略
- TCP拥塞控制
- TCP定时器管理

TCP拥塞控制

- 数据的丢失有两种情况
  - 接收方的容量太小（接收缓冲区）
  - 网络的容量太小（网络带宽）
  - Tnbm P548 Fig. 6-36所示
- 源主机如何知道拥塞
  - 收到ICMP的源抑制报文
  - 因报文丢失引起的超时

拥塞窗口

- 网络拥塞造成数据包丢失，重发机制进一步加剧了拥塞

TCP的拥塞控制方案应将由接收方的容量和由网络的容量所产生的拥塞问题分别处理，除了已定义的接收窗口外，还定义了拥塞窗口，拥塞窗口的动态调整：一旦发现数据包丢失，则降低重发数据包的速率

- 接收方承认的接收窗口表示接收缓冲区的容量
- 拥塞窗口表示网络的容量
- 接收窗口和拥塞窗口，两者取其小

拥塞窗口的具体实现

- 拥塞窗口的初始化
- 拥塞窗口大小的修正
- 选择发送数据量

  取两个窗口的最小值作为可以发送的数据量

拥塞窗口的初始化

- 连接建立时，发送方将拥塞窗口的初始大小设置为最大的数据段长度，并随后发一个最大长度的数据段，如该数据段在定时器超时前得到了确认，发送方在原来的拥塞窗口的基础上再增加一倍长度，发送两个数据段，如两个数据段都得到了确认，则再增加一倍长度，直到数据传输超时或到达接收方的窗口大小为止
- 当拥塞窗口的大小为n个数据段时，如果发送的n个数据段都得到了确认，那么此时拥塞窗口的大小即为n个数据段对应的字节数

拥塞窗口的具体实现

- 拥塞窗口的初始化
- 拥塞窗口大小的修正
- 选择发送数据量

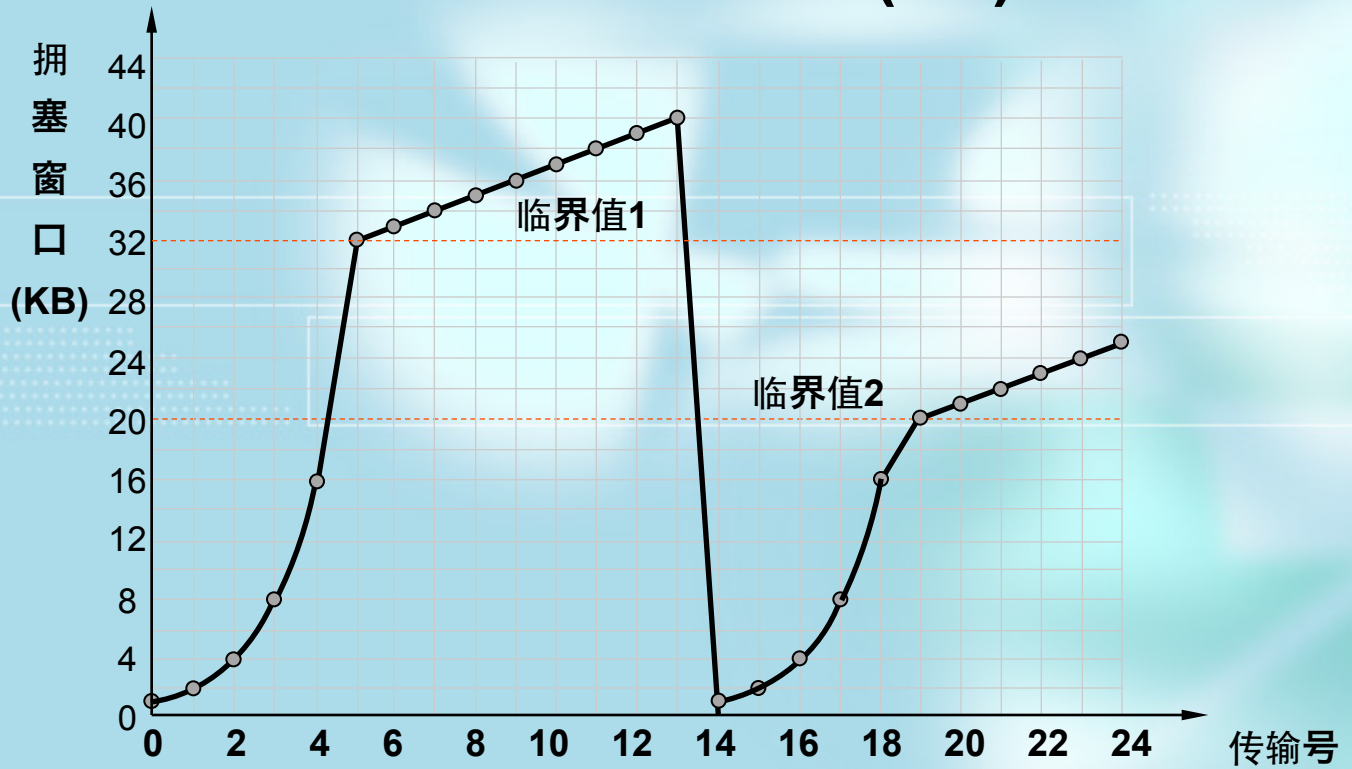
  取两个窗口的最小值作为可以发送的数据量

拥塞窗口大小的修正

- 除接收窗口和拥塞窗口外，拥塞控制时还需指定一个临界值，临界值的初始值为64K，如果发生数据传输超时，则将临界值置为当前拥塞窗口的1/2，并使拥塞窗口恢复到最大的数据段长度，成功的传输使拥塞窗口按指数增加（成倍），到达临界值后将按线性增加（按最大的数据段长度）

拥塞窗口动态调整举例

假定原来拥塞窗口为64KB，但已超时最大的数据段长度为1024(1K)



Tnbm P550 Fig. 6-37 Internet拥塞算法的一个实例

TCP协议将讨论：

- TCP服务模型
- TCP数据段头
- TCP连接和释放管理
- TCP传输策略
- TCP拥塞控制
- TCP定时器管理

TCP定时器管理

- “保活”定时器

  一旦建立连接，双方都将启动“保活”定时器，当连接长时间无数据传输，所设定的“保活”定时器超时，超时一方会发送一个探测报文检查对方是否存在，如没有得到响应，则终止连接

- 重发定时器

- 持续定时器

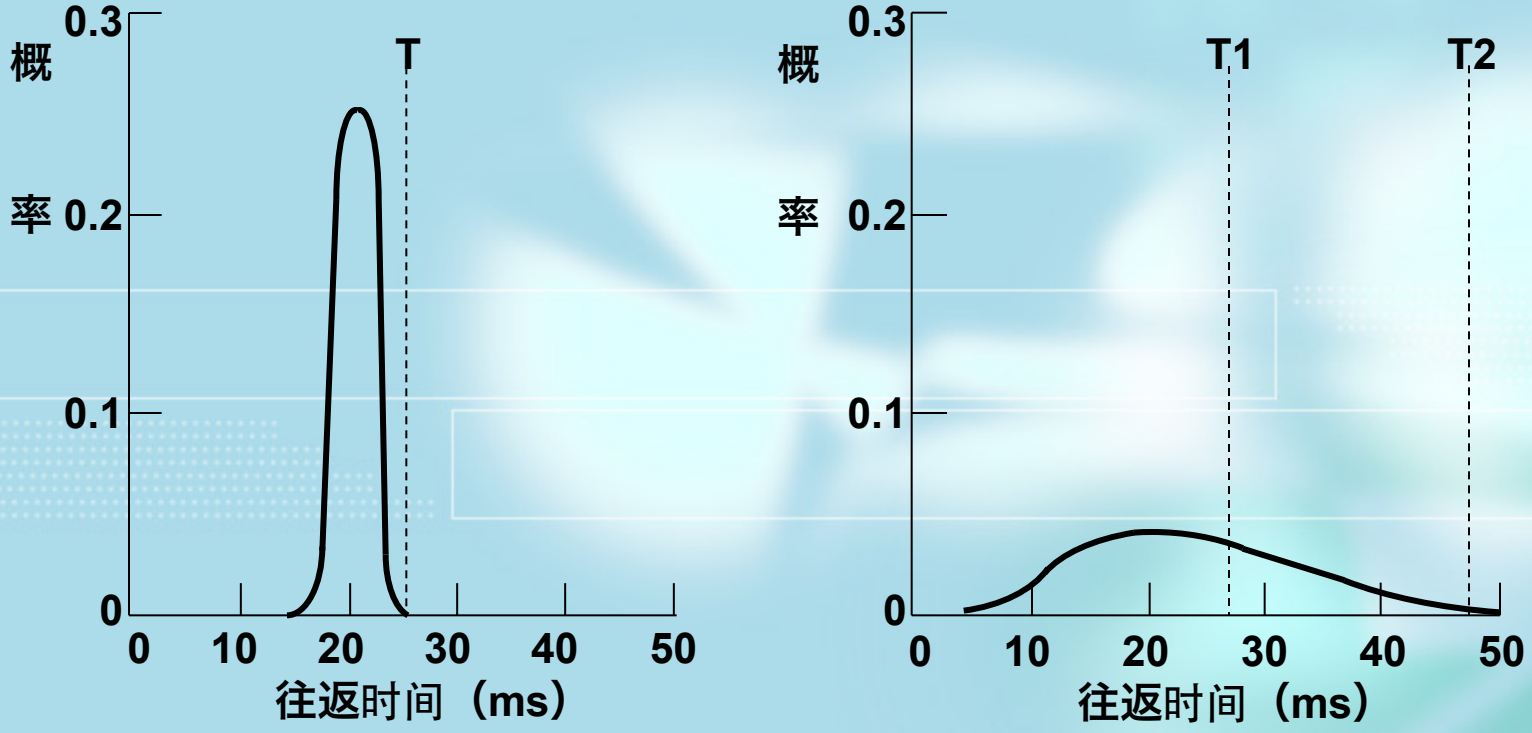
数据重发定时器

- TCP在发送一个数据段的同时，启动一个数据重发定时器，如果在定时器超时前该数据段被确认，则关闭该定时器
- 如果在确认到达之前定时器超时，则需要重发该数据段（并且该定时器重新开始计时）
- 重发定时器初始值的设定

  与数据链路层点对点的情况不同，在TCP层，源站点与目的站点之间的网络距离是动态变化的，数据传输和信号传播时间的离散性很大，线路的状态更是瞬息万变

  所以，重发定时器的初始值不宜设定为固定大小

链路层和TCP层的延时分布比较



Tnbm P551 Fig. 6-38

(a)数据链路层中确认到达时间的概率密度 (b) TCP层中确认到达时间的概率密度

延时分布比较说明

- 对于点对点的数据链路层，延时由数据传输延时和信号传播延时两部分组成，对这两部分延时的估计基本上是准确的（即误差很小），所以定时器初始值设定为大于估计的确认返回时间即可，如图6-33(a)所示
- TCP所面临的是完全不同的情况，端到端的连接可能远隔重洋，需经过很多路由器的存储转发，途经路由器的实际情况又是动态变化的，所以TCP确认返回所需时间的概率密度函数更接近于图6-33(b)所示

超时后的适应性重发

- 计算新的往返时间估计值

  TCP不使用固定的重发定时器，而是根据对网络性能的不断测定，主要是对远程的确认报文作延时分析，不断调整超时间隔，自适应地修正往返时间的估计值：RTT（Round-Trip Timer）

超时后的适应性重发（续）

- 往返时间估计值RTT的确定

为尽可能避免因实际延时较大而进行的错误重发，同时尽可能提高系统的吞吐率，所以适应性重发机制中往返时间估计值RTT的确定将根据前一次的估计值RTT_0，并参考实测的往返时间M_0作动态调整

$$RTT = \alpha RTT_0 + (1 - \alpha)M_0$$

其中：RTT_0：前一次计算得到的往返时间估计值

M_0：前一次实测得到的往返时间

α：修正因子，RTT_0和M_0的参考权值（通常取$\alpha = 7/8$）

重发定时器初始值的设定

- 固定为βRTT

  固定为往返时间估计值RTT的β倍，初期β= 2，但经验表明常量是很不灵活的，当环境发生变化时不能很好地适应

- 偏差值方法

  偏差值D =$\alpha D_0 + (1 - \alpha) | RTT_0 - M_0 |$

  超时值 = RTT + 4*D

TCP定时器管理

- “保活”定时器

  一旦建立连接，双方即启动“保活”定时器，当连接长时间无数据传输，所设定的“保活”定时器超时，超时一方会发送一个探测报文检查对方是否存在，如没有得到响应，则终止连接

- 重发定时器

- 持续定时器

持续定时器

- 如接收方向发送方发出一个零窗口公告，发送方当即停止发送，当接收方的上层处理了所收到的数据并释放了全部的缓冲区后，将向发送方发出一个新的窗口公告，以通知发送方可继续发送，如果此更新公告丢失，双方将相互等待，出现死锁
- 为防止死锁，每当发送方收到一个零窗口公告，即启动一个持续定时器，如持续定时器超时，发送方将向接收方发一探测报文（仅一字节），接收方的应答报文将避免相互等待（这正是上面所说：即使在发送方已收到一个零窗口通告，也允许发送一个字节的数据报询问对方）

Thanks

jdyu@cs.sjtu.edu.cn