第5章 运输层

张瑞
ruizhang@shu.edu.cn

1

第5章 运输层

- 5.1 运输层协议概述
- 5.2 用户数据报协议UDP
- 5.3 传输控制协议TCP
- 5.4 TCP可靠传输的实现
- 5.5 TCP的流量控制与拥塞控制

2

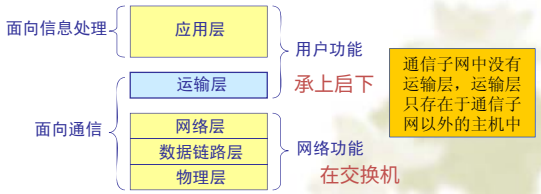
运输层提供的服务

- 数据链路层的任务
  - 在相邻的两个节点之间实现数据帧的透明传输。
- 网络层的任务
  - 负责将分组从源节点传送到目的节点。
- 局域网中网络层的功能很弱
  - 在单个的局域网中，网络层并不重要。网络层主要实现局域网的互连，流量控制和差错处理都放在链路层完成。

3

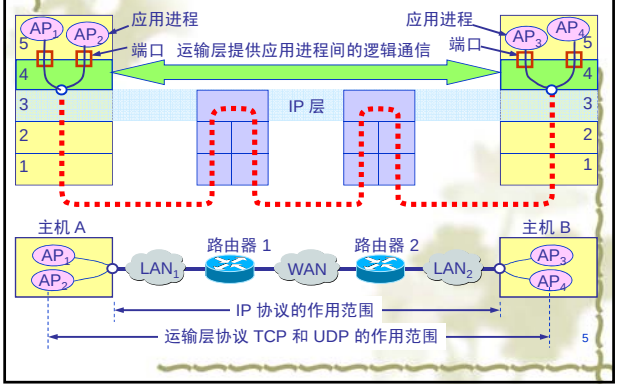
运输层协议概述

从通信和信息处理的角度看，运输层向它上面的应用层提供通信服务，它属于面向通信部分的最高层，同时也是用户功能中的最低层。



4

运输层为相互通信的应用进程提供了逻辑通信



5

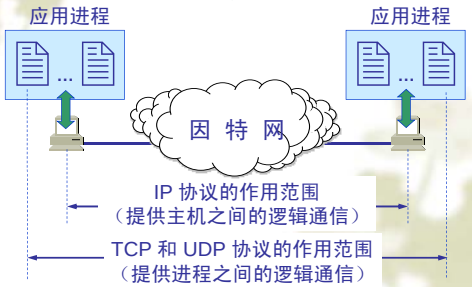
应用进程之间的通

- 两个主机进行通信实际上就
  进程互相通信。
- 应用进程之间的通信又称为
- 运输层的一个很重要的功能就是复用和分用。应用层不同进程的报文通过不同的端口向下交到运输层，再往下就共用网络层提供的服务。
- “运输层提供应用进程间的逻辑通信”。“逻辑通信”的意思是：运输层之间的通信好像是沿水平方向传送数据。但事实上这两个运输层之间并没有一条水平方向的物理连接。

分组到达目的主机后，目的主机的运输层使用其分用功能，将不同端口的报文交付给不同的应用进程

6

运输层协议和网络层协议的主要区别



7

运输层的主要功能

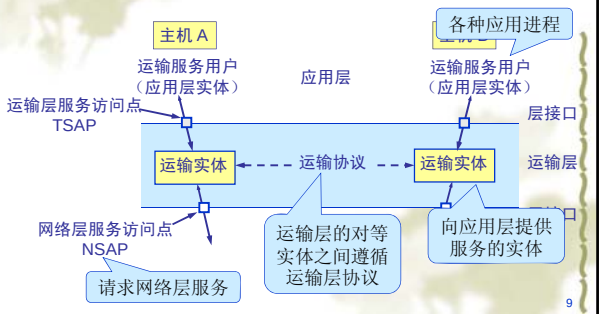
- 运输层为应用进程之间提供端到端的逻辑通信（但网络层是为主机之间提供逻辑通信）。
- 运输层还要对收到的报文进行差错检测。
- 运输层需要有两种不同的运输协议，即面向连接的 TCP 和无连接的 UDP。

网络层无法同时实现两个协议，仅用无连接的IP协议

因为在网络层，IP数据报中的校验和字段，仅对数据报的首部作校验，而不检查数据部分

8

运输层与其上下层之间的关系的 OSI 表示法



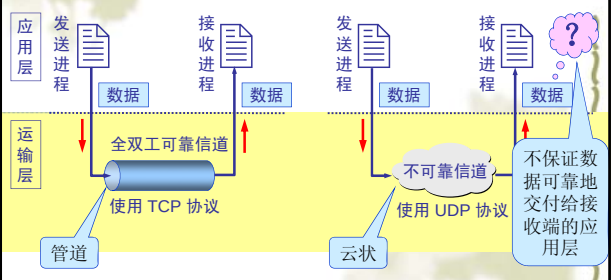
9

两种不同的运输协议

- 运输层向高层用户屏蔽了下面网络核心的细节（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就是好像在两个运输层实体之间有一条端到端的逻辑通信信道。
- 当运输层采用面向连接的 TCP 协议时，尽管下面的网络是不可靠的（只提供尽最大努力服务），但这种逻辑通信信道就相当于一条全双工的可靠信道。
- 当运输层采用无连接的 UDP 协议时，这种逻辑通信信道是一条不可靠信道。

10

运输层向上提供可靠的和不可靠的逻辑通信信道



11

几点注意

- TCP可靠信道含义
  - 管道中运输：无差错、按序序）、无丢失、无重复。
- UDP的不可靠信道含义
  - “收下来的数据不可靠，里面有差错”？
  - 特点：不保证交付。接收时不按序、可能出现丢失和重复。
- 运输层的可靠交付含义
  - 运输层将数据可靠地交付给接收端的应用层。

不是。因为运输层一旦收到有差错的报文，就将其丢弃，因此不会收下有差错的报文

12

运输层的两个主要协议

TCP/IP 的运输层有两个不同的协议：

(1) 用户数据报协议 UDP
(User Datagram Protocol)

(2) 传输控制协议 TCP
(Transmission Control Protocol)

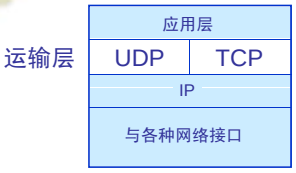
13

TCP 与 UDP

- 两个对等运输实体在通信时传送的数据单位叫作运输协议数据单元 TPDU (Transport Protocol Data Unit)。
- TCP 传送的数据单位协议是 TCP 报文段(segment)
- UDP 传送的数据单位协议是 UDP 报文或用户数据报。

14

TCP/IP 体系中的运输层协议



15

TCP 与 UDP

- 1、UDP 在传送数据之前不需要先建立连接。对方的运输层在收到 UDP 报文后，不需要给出任何确认。虽然 UDP 不提供可靠交付，但在某些情况下 UDP 是一种最有效的工作方式。
- 2、TCP 则提供面向连接的服务。TCP 不提供广播或多播服务。由于 TCP 要提供可靠的、面向连接的运输服务，因此不可避免地增加了许多的开销。这不仅使协议数据单元的首部增大很多，还要占用许多的处理机资源。

16

TCP 与 UDP

- 3、运输层的UDP用户数据报与网际层的IP数据报有很大区别。IP数据报要经过互连网中许多路由器的存储转发，但 UDP 用户数据报是在运输层的端到端抽象的逻辑信道中传送的。
- 4、TCP 报文段是在运输层抽象的端到端逻辑信道中传送，这种信道是可靠的全双工信道。但这样的信道却不知道究竟经过了哪些路由器，而这些路由器也根本不知道上面的运输层是否建立了 TCP 连接。 都是封装在IP中

17

运输层的端口

- 运行在计算机中的进程是用进程标识符来标志的。
- 运行在应用层的各种应用进程却不应当让计算机操作系统指派它的进程标识符。这是因为在因特网上使用的计算机的操作系统种类很多，而不同的操作系统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能够互相通信，就必须用统一的方法对 TCP/IP 体系的应用进程进行标志。

不是每个计算机自行指派

18

需要解决的问题

- 由于进程的创建和撤销都是动态的，发送方几乎无法识别其他机器上的进程。
- 有时我们会改换接收报文的进程，但并不需要通知所有发送方。
- 我们往往需要利用目的主机提供的功能来识别终点，而不需要知道实现这个功能的进程。

19

端口号(protocol port number)
简称为端口(port)

- 解决这个问题的方法就是在运输层使用协议端口号(protocol port number)，或通常简称为端口(port)。
- 虽然通信的终点是应用进程，但我们可以把端口想象是通信的终点，因为我们只要把要传送的报文交到目的主机的某一个合适的目的端口，剩下的工作（即最后交付目的进程）就由 TCP 来完成。

20

软件端口与硬件端口

- 在协议栈层间的抽象的协议端口是软件端口。
- 路由器或交换机上的端口是硬件端口。
- 硬件端口是不同硬件设备进行交互的接口，而软件端口是应用层的各种协议进程与运输实体进行层间交互的一种地址。

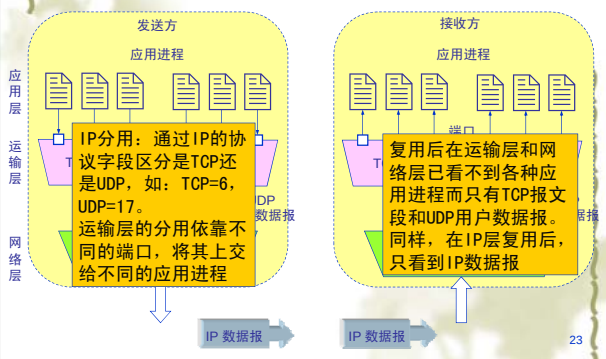
21

端口的概念

- 端口就是运输层服务访问点 TSAP。
- 端口的作用就是让应用层的各种应用进程都能将其数据通过端口向下交付给运输层，以及让运输层知道应当将其报文段中的数据向上通过端口交付给应用层相应的进程。
  - 运输层的复用和分用功能依赖端口来完成的。
- 端口是用来标志应用层的进程。

22

端口在进程之间的通信中所起的作用



23

端口

- 端口用一个 16 位（64K个）端口号进行标识。
- 端口号只具有本地意义
  - 即端口号只是为了标志本计算机应用层中的各进程。
  - 在因特网中不同计算机的相同端口号是没有联系的。

24

两类端口

- 服务器端使用的端口号
  - 熟知端口号或系统端口号
  - 登记端口号
- 客户端使用的端口号或短暂端口号

25

熟知端口号

- 英特网名字和号码公司ICANN负责常用的应用层程序的熟知端口分配；其数值一般为 0~1023；
- TCP/IP体系确定，所有用户进程都知道；
- 服务器进程随时检测熟知端口，发现通信请求；
- 当一种新的应用程序出现时，必须为它指派一个熟知端口。
- 如：FTP(21)、Telnet (23)、SMTP (25)、DNS(53)、TFTP(69)、HTTP(80)、SNMP (161)

26

登记端口号

- 数值为1024~49151
- 为没有熟知端口号的应用程序使用的。使用这个范围的端口号必须在 IANA 登记，以防止重复。

27

客户端口号

- 数值为49152~65535
- 留给客户进程选择暂时使用。当服务器进程收到客户进程的报文时，就知道了客户进程所使用的动态端口号。通信结束后，这个端口号可供其他客户进程以后使用。

28

第5章 运输层

- 5.1 运输层协议概述
- 5.2 用户数据报协议UDP
- 5.3 传输控制协议TCP
- 5.4 TCP可靠传输的实现
- 5.5 TCP的流量控制与拥塞控制

29

UDP 概述

- UDP只在IP的数据报服务之上增加了很少一点的功能，即端口的功能和差错检测的功能。
- 虽然 UDP 用户数据报只能提供不可靠的交付，但 UDP 在某些方面有其特殊的优点。
  - 发送数据之前不需要建立连接。
  - UDP 的主机不需要维持复杂的连接状态表。
  - UDP 用户数据报只有8个字节的首部开销。
  - 网络出现的拥塞不会使源主机的发送速率降低。这对某些实时应用（IP电话、视频会议）是很重要的。

30

使用UDP和TCP的应用举例

应用	应用层协议	运输层协议
名字转换	DNS	UDP
文件传送	TFTP	UDP
路由选择协议	RIP	UDP
IP地址配置	BOOTP、DHCP	UDP
网络管理	SNMP	UDP
远程文件服务	NFS	UDP
电子邮件	SMTP	TCP
远程终端接入	TELNET	TCP
万维网	HTTP	TCP
文件传送	FTP	TCP

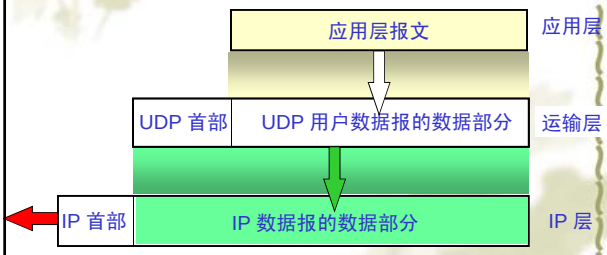
31

面向报文的UDP

- 发送方 UDP 对应用程序交下来的报文，在添加首部后就向下交付 IP 层。UDP 对应用层交下来的报文，既不合并，也不拆分，而是保留这些报文的边界。
- 应用层交给 UDP 多长的报文，UDP 就照样发送，即一次发送一个报文。
- 接收方 UDP 对 IP 层交上来的 UDP 用户数据报，在去除首部后就原封不动地交付上层的应用进程，一次交付一个完整的报文。
- 应用程序必须选择合适大小的报文。

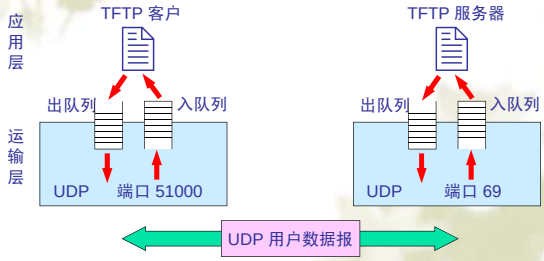
32

UDP是面向报文的



33

端口是用报文队列来实现



34

UDP队列工作原理

- TFTP服务器端
  - 启动
    - 服务进程一直运行，等待客户端的服务请求；
    - 服务器使用周知端口69。
- TFTP客户端
  - 客户端启动
    - 客户端进程启动的时侯，操作系统临时分配一般端口、出队列和入队列。进程终止时分配的资源被撤销。
  - 客户端发报文
    - 客户进程将报文发送到出队列中。按照队列顺序发送，加上UDP首部，填入目的端口号69。若队列溢出，通知客户进程，暂停发送。

35

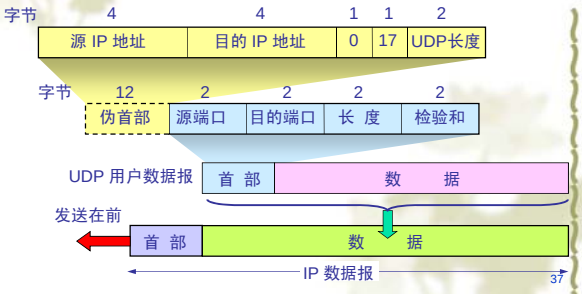
UDP队列工作原理

- 客户端接收IP层报文
  - UDP先检查报文中的端口号是否正确，不正确则UDP丢弃该报文，并发送ICMP端口不可达。若端口正确，UDP将该报文放到入队列的末尾，客户进程按照报文的FIFO的顺序取走报文。
- 服务器收、发报文
  - ***接收：*** 服务器UDP先判断目的端口是否为69？若是，则将报文放到入队列的末尾，否则，丢弃该报文，并发送ICMP不可达给客户端。
  - ***发送：*** 将报文放到出队列的末尾，加上UDP的首部，然后传送给IP层。

36

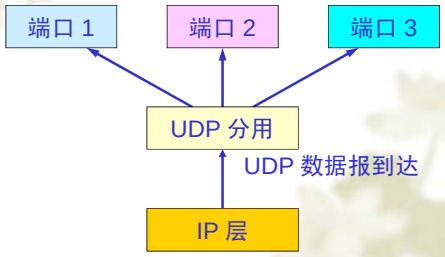
伪首部不传输，仅仅计算校验和

UDP 的首部格式



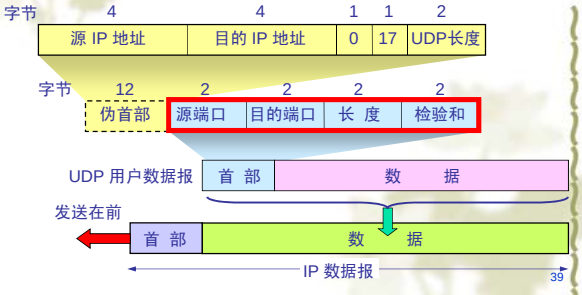
37

UDP基于端口的分用



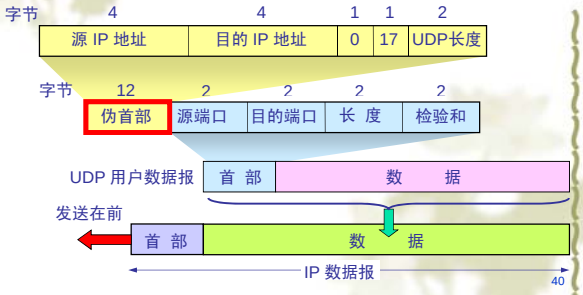
38

用户数据报 UDP 有两个字段：数据字段和首部字段。首部字段有 8 个字节，由 4 个字段组成，每个字段都是两个字节。



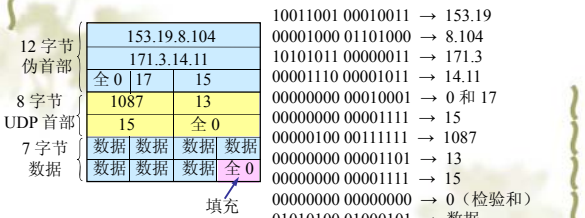
39

在计算检验和时，临时把“伪首部”和 UDP 用户数据报连接在一起。*伪首部仅仅是为了计算检验和。*



40

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

41

请注意：进行反码运算求和时，最高位有进位 2，这个 2 应当加到最低位。

UDP的校验和

- IP的校验和只校验了IP数据报的首部；
- UDP的校验和，检查了：
  - UDP数据报的源端口、目的端口；
  - UDP数据报的数据部分；
  - IP数据报的源地址和目的地址。

42

第5章 运输层

- 5.1 运输层协议概述
- 5.2 用户数据报协议UDP
- 5.3 传输控制协议TCP
- 5.4 TCP可靠传输的实现
- 5.5 TCP的流量控制与拥塞控制

43

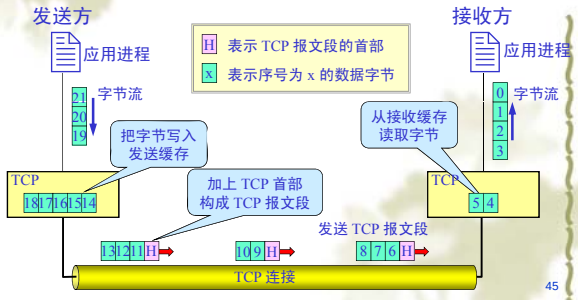
TCP最主要的特点

- TCP 是面向连接的运输层协议。
- 每一条 TCP 连接只能有两个端点 (endpoint)，每一条 TCP 连接只能是点对点的（一对一）。
- TCP 提供可靠交付的服务。
- TCP 提供全双工通信。
- 面向字节流。

44

自主选择一个报文有几个字节，按字节编号

TCP面向流的概念



45

TCP报文的发送

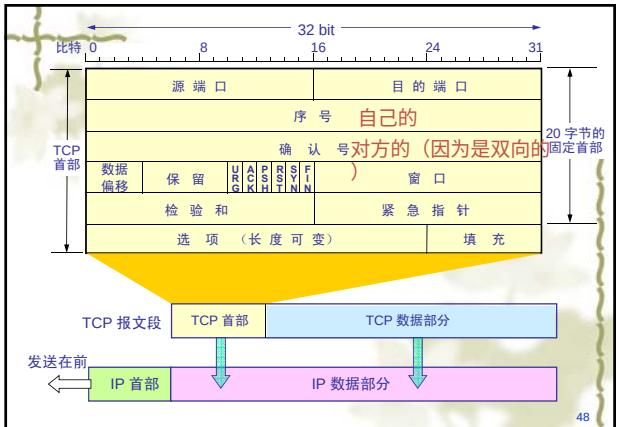
- TCP 连接是一条虚连接而不是一条真正的物理连接。
- TCP 对应用进程一次把多长的报文发送到TCP 的缓存中是不关心的。
- TCP 根据对方给出的窗口值和当前网络拥塞的程度来决定一个报文段应包含多少个字节（UDP 发送的报文长度是应用进程给出的）。
- TCP 可把太长的数据块划分短一些再传送。TCP 也可等待积累有足够多的字节后再构成报文段发送出去。

46

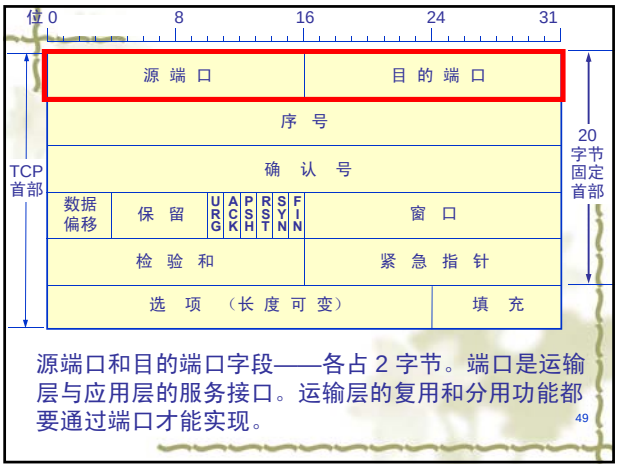
TCP 报文段首部

- TCP报文段（Segment）包括：
  - 首部
    - TCP的全部功能体现在各个首部字段中；
    - 首部的前20个字节是固定的，后4N（N为整数）字节是根据需要而增加的选项。
    - 因此，TCP首部的最小长度为20字节。
  - 数据部分

47

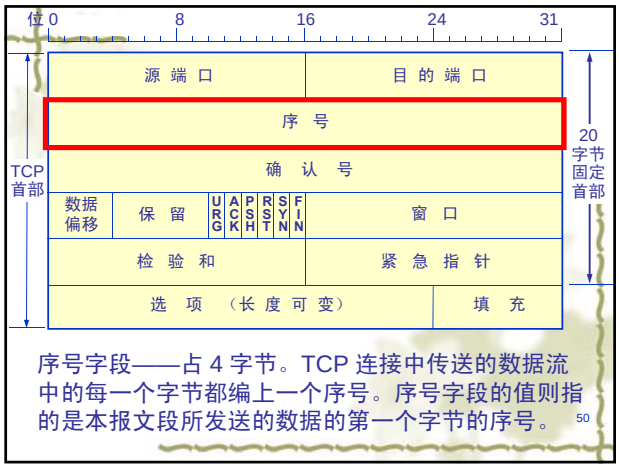


48



源端口和目的端口字段——各占 2 字节。端口是运输层与应用层的服务接口。运输层的复用和分用功能都要通过端口才能实现。

49



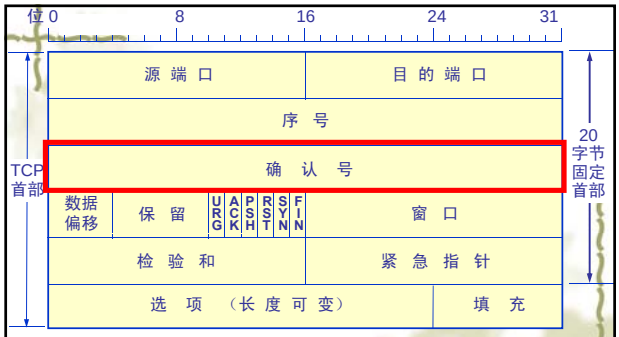
序号字段——占 4 字节。TCP 连接中传送的数据流中的每一个字节都编上一个序号。序号字段的值则指的是本报文段所发送的数据的第一个字节的序号。

50

序号字段

- TCP把一个TCP连接中的每一个字节都编上号；
- 整个数据的起始序号在连接的建立时设置；
- 序号字段指本报文段所发送数据的第一个字节的序号；
  - 若序号字段值为301，携带数据共100字节；则本报文段数据的第一个字节的序号为301，最后一个字节的序号为400。那么下一个报文段的序号字段为401。

51



确认号字段——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。

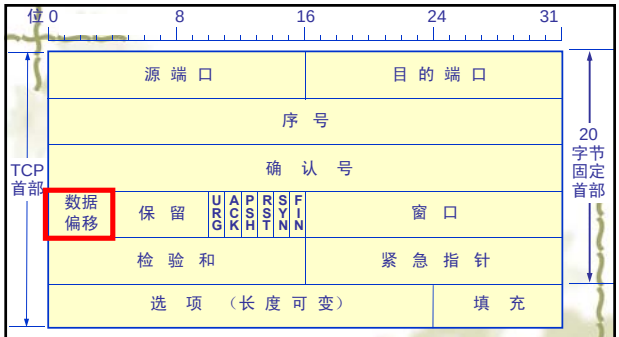
52

确认号字段

- 期望收到对方下一个报文段的数据的第一个字节的序号；
- 如：A正确收到B发送过来的一个报文段，其序号字段为501，数据长度为200字节。则，A在发送给B的响应报文中确认号应为？
- 因为A正确接收了B发送的序号从501到700之间的数据，A期望收到B的下一个报文段的第一个字节的序号为701。

若确认号=N，则表明：到序号N-1为止的所有数据都已正确收到。

53

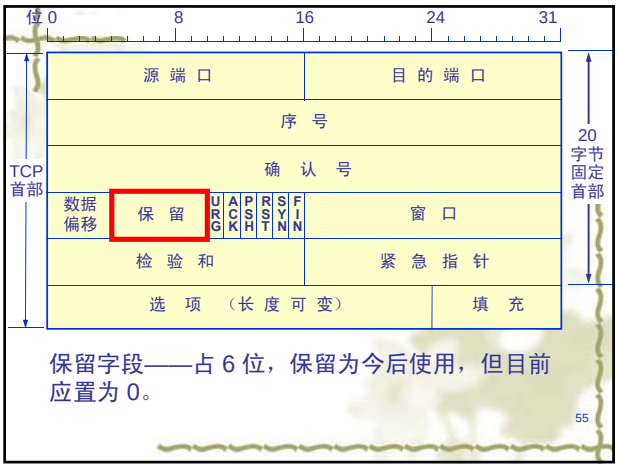


数据偏移（即首部长度）——占 4 位，它指出 TCP 报文段的数据起始处距离 TCP 报文段的起始处有多远。“数据偏移”的单位是 32 位字（以 4 字节为计算单位）。

54

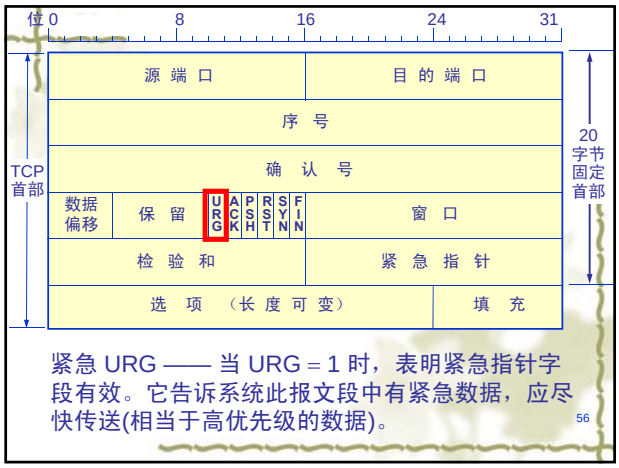
首部长度最多少位？
4个bit -> 15，
15 * 4 = 60 字节
60 -20（固定） = 40 字节

ACK，SYN，FIN 很重要



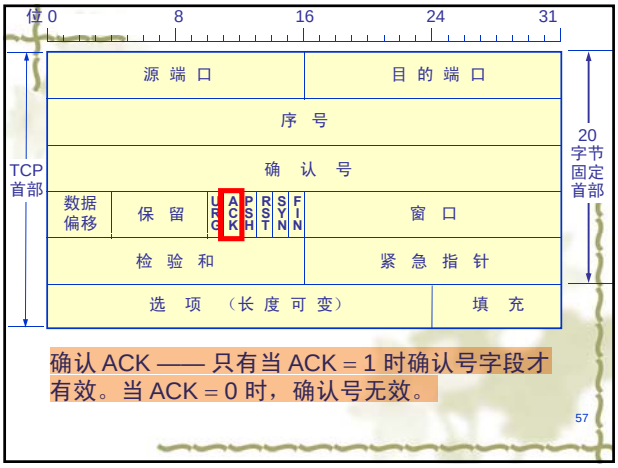
保留字段——占 6 位，保留为今后使用，但目前应置为 0。

55



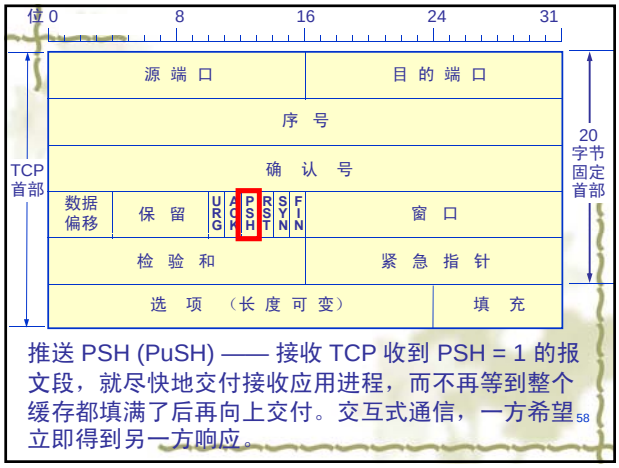
紧急 URG —— 当 URG = 1 时，表明紧急指针字段有效。它告诉系统此报文段中有紧急数据，应尽快传送(相当于高优先级的数据)。

56



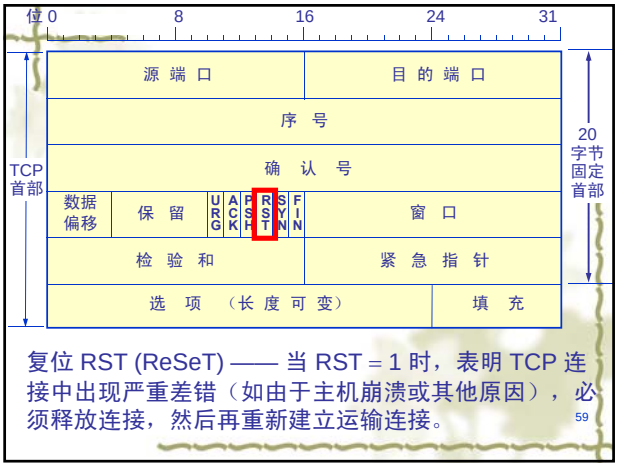
确认 ACK —— 只有当 ACK = 1 时确认号字段才有效。当 ACK = 0 时，确认号无效。

57



推送 PSH (PuSH) —— 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了后再向上交付。交互式通信，一方希望立即得到另一方响应。

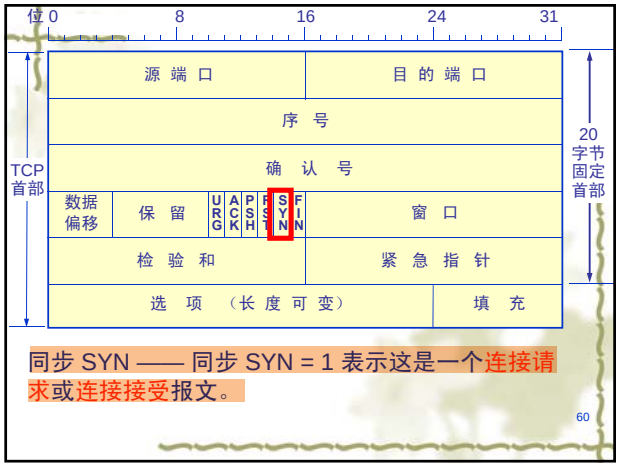
58



复位 RST (ReSeT) —— 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，然后再重新建立运输连接。

59

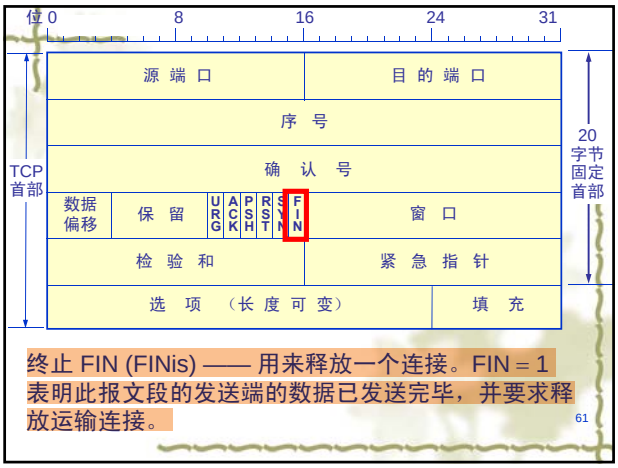
也会用于拒绝非法连接。



同步 SYN —— 同步 SYN = 1 表示这是一个连接请求或连接接受报文。

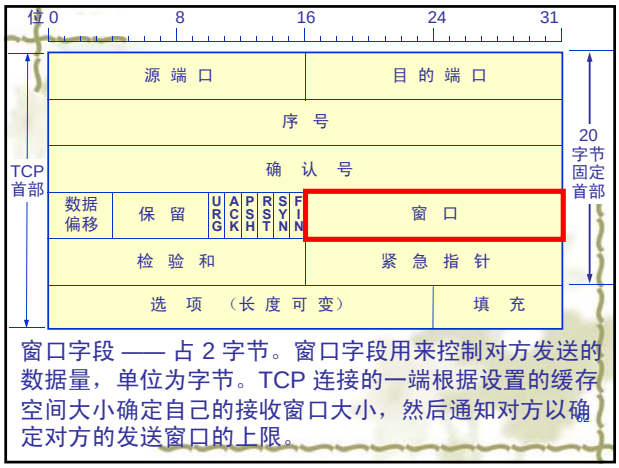
60

SYN = 1 ACK = 0　连接请求报文段
SYN = 1 ACK = 1　连接接受报文段



终止 FIN (FINis) —— 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，并要求释放运输连接。

61

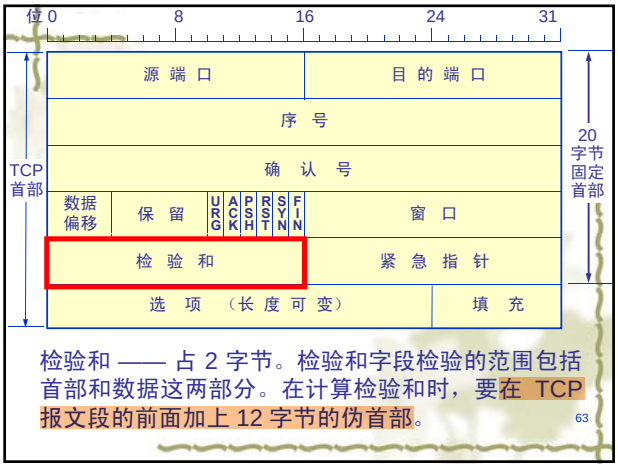


窗口字段 —— 占 2 字节。窗口字段用来控制对方发送的数据量，单位为字节。TCP 连接的一端根据设置的缓存空间大小确定自己的接收窗口大小，然后通知对方以确定对方的发送窗口的上限。

A ———> B
A给B的反馈信息：
窗口 500 （即B给A最多一次发500数据）
确认号：201

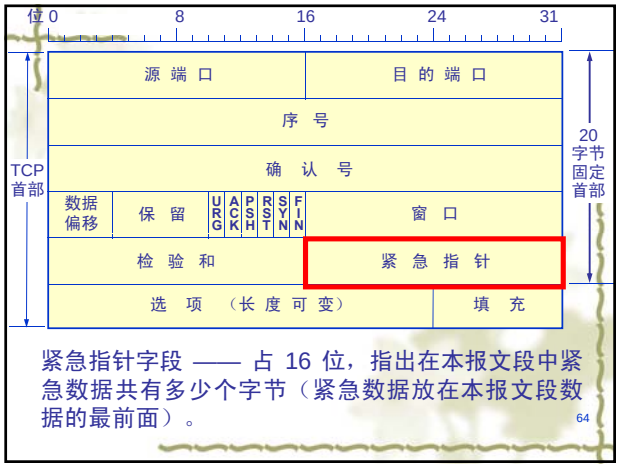
B ->A :201 - 700 (500数据)

最终作用，告诉对方我的窗口大小，让对方确定窗口大小
窗口号和确认号连用



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。

63



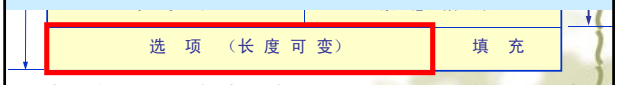
紧急指针字段 —— 占 16 位，指出在本报文段中紧急数据共有多少个字节（紧急数据放在本报文段数据的最前面）。

64

伪首部 = 源IP地址（4）+目的IP地址（4）+0+6+TCP长度 共12位

不考 ->

MSS (Maximum Segment Size)
是 TCP 报文段中的数据字段的最大长度。
数据字段加上 TCP 首部
才等于整个的 TCP 报文段。



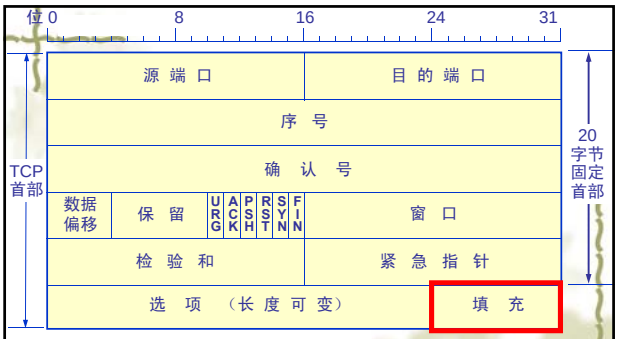
选项字段 —— 长度可变。TCP 最初只规定了一种选项，即最大报文段长度 MSS。MSS 告诉对方 TCP：“我的缓存所能接收的报文段的数据字段的最大长度是 MSS 个字节。”

65

其他选项

- 窗口扩大选项 ——占 3 字节，其中有一个字节表示移位值 S。新的窗口值等于TCP 首部中的窗口位数增大到(16 + S)，相当于把窗口值向左移动 S 位后获得实际的窗口大小。
- 时间戳选项——占10 字节，其中最主要的字段时间戳值字段（4 字节）和时间戳回送回答字段（4 字节）。
- 选择确认选项——针对TCP可靠传输的实现。

66



填充字段 —— 这是为了使整个首部长度是 4 字节的整数倍。

67

TCP的连接

- TCP 把连接作为最基本的抽象。
- 每一条 TCP 连接有两个端点。
- TCP 连接的端点不是主机，不是主机的 IP 地址，不是应用进程，也不是运输层的协议端口。TCP 连接的端点叫做套接字(socket)或插口。
- 端口号拼接到(contatenated with) IP 地址即构成了套接字。

68

同一个IP地址/端口号都可以建立多个TCP连接

套接字：连接的端点

套接字 (socket)

套接字 socket = (IP地址: 端口号)

- 每一条 TCP 连接唯一地被通信两端的两个端点（即两个套接字）所确定。即：

TCP 连接 ::= {socket1, socket2}
= {(IP1: port1), (IP2: port2)}

69

TCP连接的建立和释放

- 运输连接就有三个阶段，即：连接建立、数据传送和连接释放。运输连接的管理就是使运输连接的建立和释放都能正常地进行。
- 连接建立过程中要解决以下三个问题：
  - 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）。
  - 能够对运输实体资源（如缓存大小，连接表中的项目等）进行分配。

70

TCP连接和释放！！！！重要！！！！

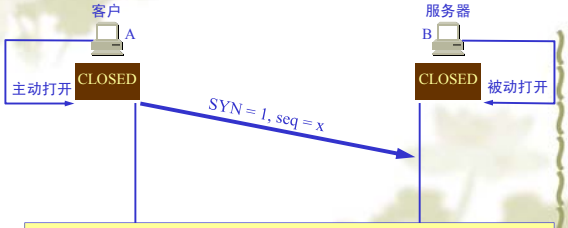
客户服务器方式

- TCP 连接的建立都是采用客户服务器方式。
- 主动发起连接建立的应用进程叫做客户(client)。
- 被动等待连接建立的应用进程叫做服务器(server)。

71

TCP 的连接建立

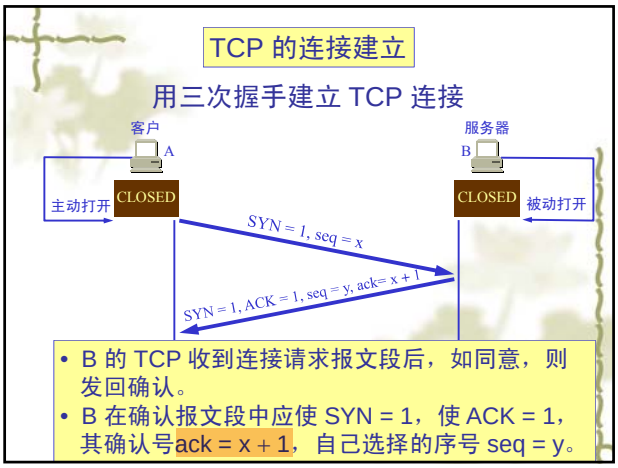
用三次握手建立 TCP 连接



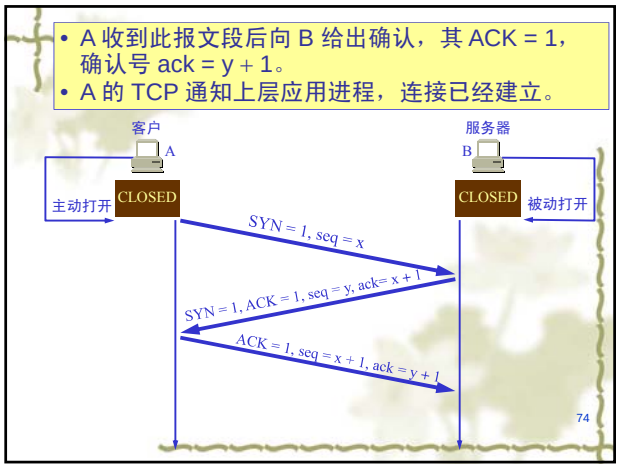
A 的 TCP 向 B 发出连接请求报文段，其首部中的同步位 SYN = 1，并选择序号 seq = x，表明传送数据时的第一个数据字节的序号是 x。

ACK = 0

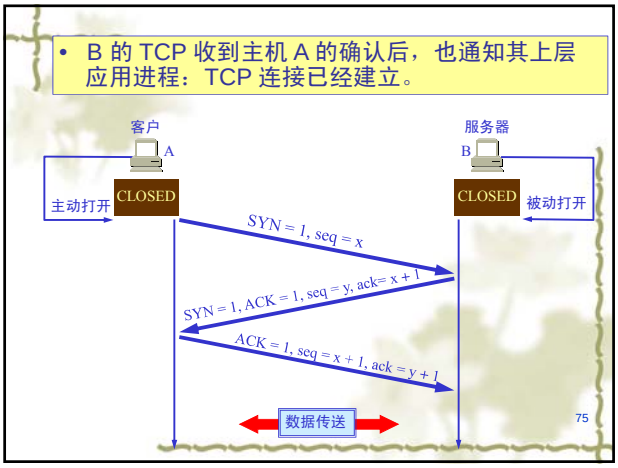
SYN = 0

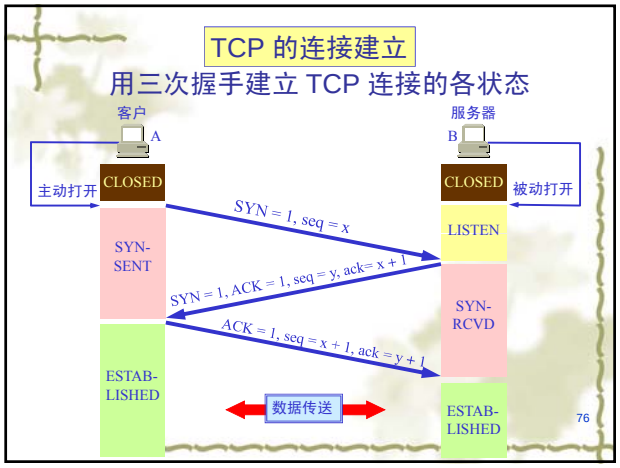


前两步，无数据但是也会消耗一个序号。
只有SYN = 1 时，才会有消耗

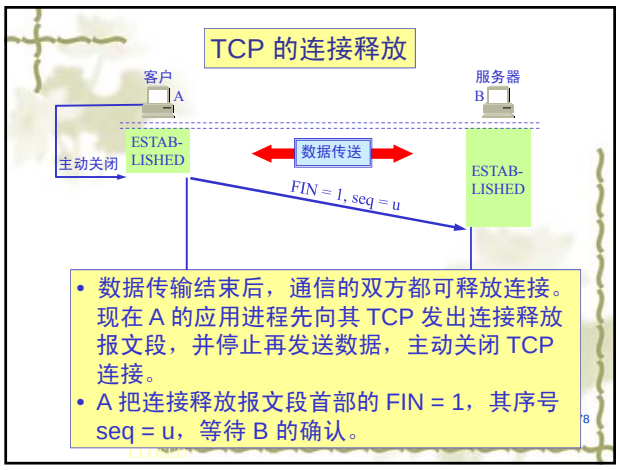


ack：期望收到哪一位开始
seq: 第一个数据从哪里开始

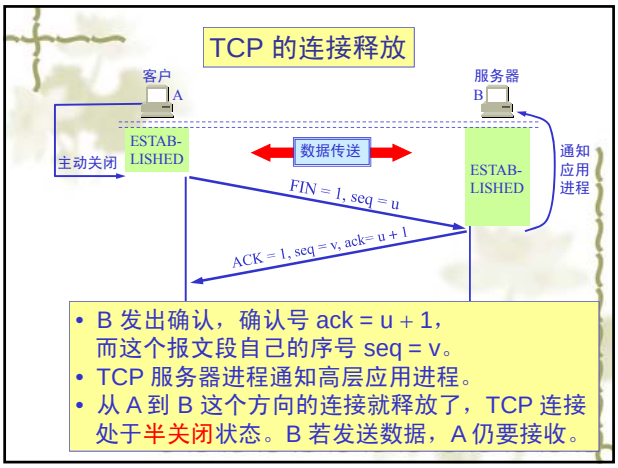




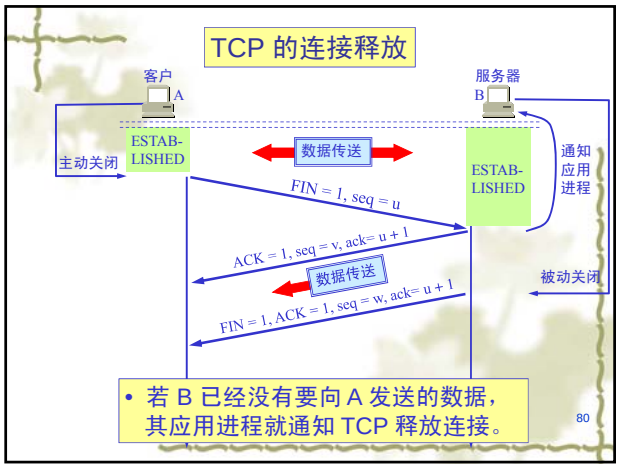
第三次作用：防止已失效的连接请求报文端突然又传送到了，因而产生错误。



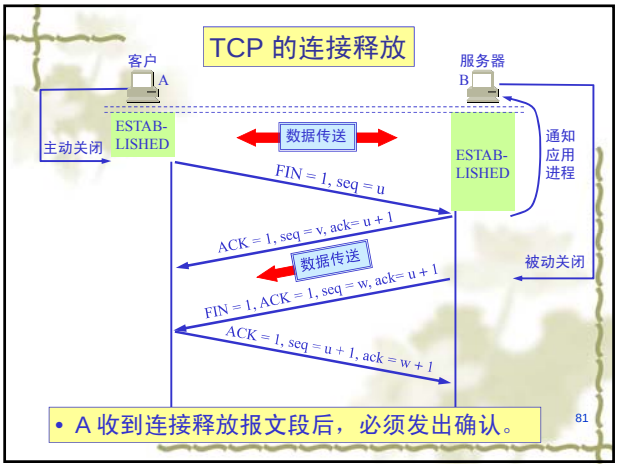
u并非随机数，而是和之前 的序列有联系

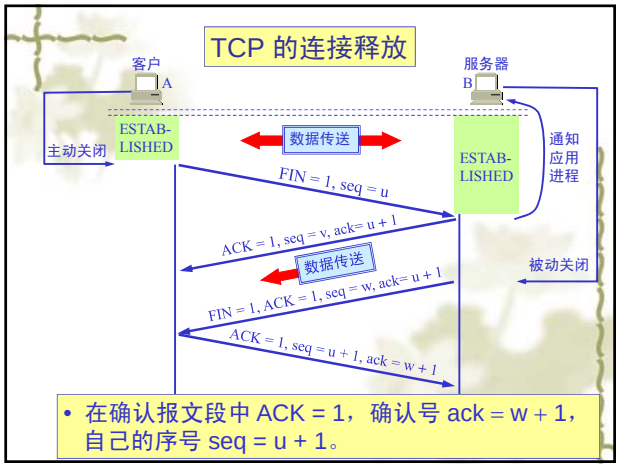


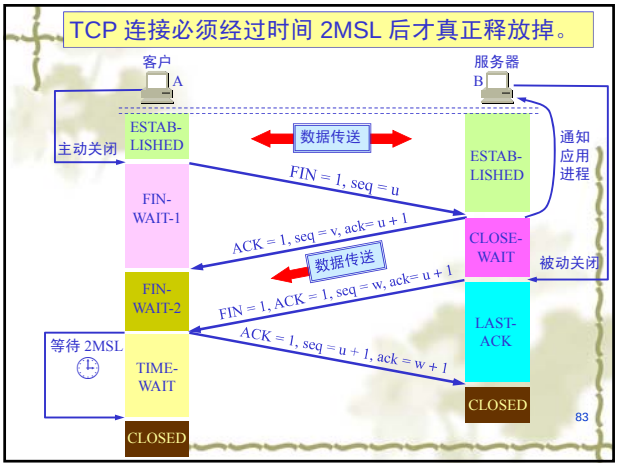
补充：只有SYN = 1， FIN = 1 时，消耗一个序号



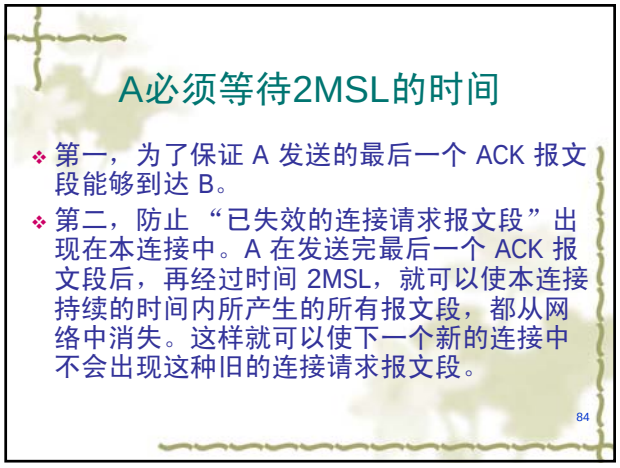
如果B 到A不发送数据，seq = v







四次挥手之后，并没有真正的释放。
因为,如果最后一个报文确认信息丢失，则会出错。服务器一直不能释放。



第5章 运输层

- 5.1 运输层协议概述
- 5.2 用户数据报协议UDP
- 5.3 传输控制协议TCP
- 5.4 TCP可靠传输的实现
- 5.5 TCP的流量控制与拥塞控制

85

完全理想化的数据传输

- 两个假设：
- 假定 1： 传输信道不产生差错。
  - 有差错时怎么办？
- 假定 2： 不管发送方以多快的速度发送数据，接收方总是来得及处理收到的数据。
  - 接收方来不及处理时怎么办？

86

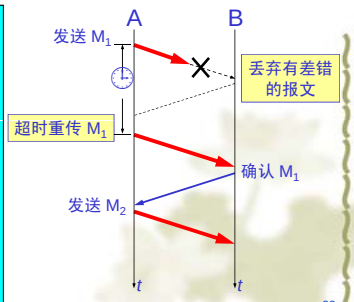
停止等待协议

- 现在去掉上述的第二个假定。但是，仍然保留第一个假定。
- 即主机 A 向主机 B传输数据的信道仍然是无差错的理想信道。然而现在不能保证接收端向主机交付数据的速率永远不低于发送端发送数据的速率。
- 停止等待协议：每发送完一个分组就停止发送，等待对方的确认，收到确认后再发送下一个分组。

87

停止等待协议

- 两个假设都去掉
- 当发送方发完一帧后，停止发送，等待对方的应答
- 如果收到对方肯定应答，接着发送下一帧
- 如果超过规定时间没有收到应答，则重发该帧



(b) 出现差错

88

超时重传

- 设置超时计时器
  - 给计时器设定初始值，发送完一个分组后，启动计时器
  - 如果在超时计时器到期之前收到了对方的确认，则表明发送正常，撤销该计时器
  - 如果在计时时间内没有收到应答，则认为数据丢失，重新发送该分组
- 在发送完一个分组后，必须暂时保留已发送的分组的副本 ➡ 发生超时重传时使用

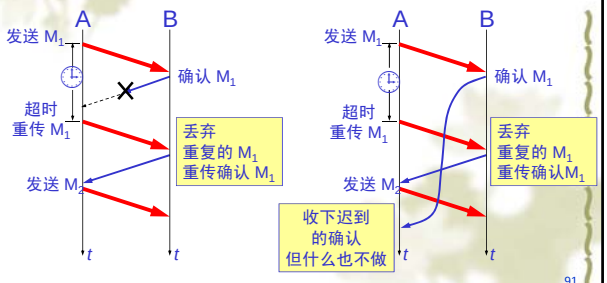
89

超时重传

- 若数据帧被正确接收，而应答帧丢失或延迟到达，这样发送方在定时时间内仍收不到应答帧，重发该帧，导致帧重复。
- 分组和确认分组都必须进行编号
  - 明确哪个分组收到了确认，哪个没收到
- 超时计时器的重传时间应当比数据在分组传输的平均往返时间更长一些。

90

确认丢失和确认迟到



(a) 确认丢失　　(b) 确认迟到

91

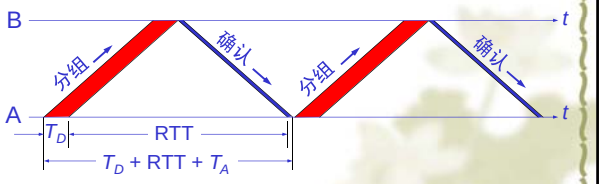
可靠通信的实现

- 使用上述的确认和重传机制，我们就可以在不可靠的传输网络上实现可靠的通信。
- 这种可靠传输协议常称为自动重传请求 ARQ (Automatic Repeat reQuest)。
- ARQ 表明重传的请求是自动进行的。接收方不需要请求发送方重传某个出错的分组 。

92

信道利用率

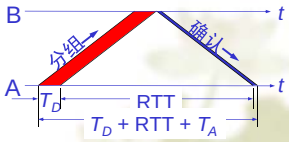
- 停止等待协议的优点是简单，但缺点是信道利用率太低。



93

信道的利用率 *U*

- T_D：A发送分组需要的时间
- T_A：B发送确认分组需要的时间
- RTT：往返时延
- 忽略双方的处理时延



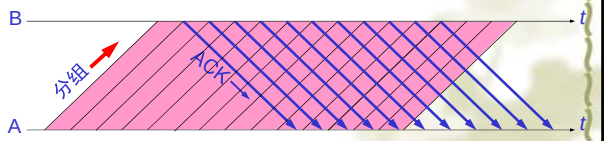
$$U = \frac{T_D}{T_D + \mathrm{RTT} + T_A}$$

各自的发送时延+往返时延

94

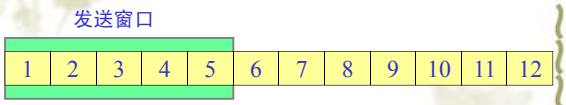
流水线传输

- 发送方可连续发送多个分组，不必每发完一个分组就停顿下来等待对方的确认。
- 由于信道上一直有数据不间断地传送，这种传输方式可获得很高的信道利用率。

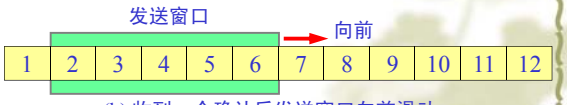


95

连续 ARQ 协议



(a) 发送方维持发送窗口（发送窗口是 5）



(b) 收到一个确认后发送窗口向前滑动

96

连续 ARQ 协议

- 发送端：
  - 在发送完一个分组后，不是停下来等待确认分组，而是可以连续再发送若干个分组。
  - 如果这时收到了接收端发来的确认，那么还可以接着发送分组。
  - 如果在超时时间到时，仍然没有收到相应分组的确认，则重新从这个分组开始传起(Go-back-N)。
- 接收端：只按序接收分组
  - 当接收到一个有差错的分组时，丢弃该分组和它以后的所有分组，让它们在发送端超时。
  - 重复发送已发送过的最后一个确认分组。

97

累积确认

- 接收方一般采用累积确认的方式。即不必对收到的分组逐个发送确认，而是对按序到达的最后一个分组发送确认，这样就表示：到这个分组为止的所有分组都已正确收到了。
- 累积确认有的优点是：容易实现，即使确认丢失也不必重传。缺点是：不能向发送方反映出接收方已经正确收到的所有分组的信息。

98

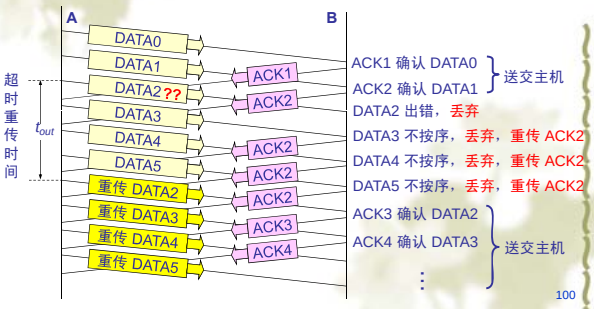
Go-back-N（回退N）

- 如果发送方发送了前 5 个分组，而中间的第 3 个分组丢失了。这时接收方只能对前两个分组发出确认。发送方无法知道后面三个分组的下落，而只好把后面的三个分组都再重传一次。
- 这就叫做 Go-back-N（回退 N），表示需要再退回来重传已发送过的 N 个分组。
- 可见当通信线路质量不好时，连续 ARQ 协议会带来负面的影响。

99

即使DATA3，4，5都正确收到了；但是DATA2出错，因此还是会丢弃2以及之后的部分

连续ARQ协议的工作原理



100

对于连续ARQ协议

发送窗口的最大值

- 当用 n 个比特进分组进行编号时，若接收窗口的大小为 1，则只有在发送窗口的大小 $W_T \le 2^n - 1$ 时，连续 ARQ 协议才能正确运行。
- 例如，当采用 3 bit 编码时，发送窗口的最大值是 7 而不是 8。
- 简单证明（用反例）：
  - 若发送窗口大小为8，则分组编号为0～7；
  - 假设所有确认分组都到达了发送端，此时发送端又发送8个新分组，编号为0~7，接收端认为是新分组；√
  - 但假设所有确认分组丢失，发送端超时后重发0~7号分组，接收端同样将其当作新分组接收（实际是重传的旧分组）。 ×

101

假设发送分组为8

发送端：	接收端：
0	0
1	1
2	2
3	3
.	.
.	.
.	.
6	7
7	无法区分0组是旧/新确认号
0	
1	

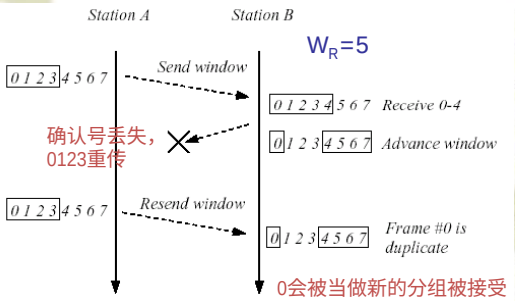
选择重传ARQ协议

- 连续ARQ协议的问题：当线路的出错率高时，将出错帧之后的所有帧都丢弃掉，重传这些帧会带来效率上的大幅度降低。
- 加大接收窗口，使得W_R>1。先收下发送序号不连续但仍处在接收窗口中的那些数据帧。等到所缺序号的数据帧收到后再一并送交主机。这就是选择重传ARQ协议。
- **接收窗口的尺寸不能超过2^{n-1}**（即序号范围的1/2），否则可能造成帧的重叠。发送窗口的尺寸一般和接收窗口的尺寸相同。
- 优点：避免重复传送那些本来已经正确到达接收端的数据帧。

102

当n =3, 理论上不能超过4

接收窗口的尺寸超过2^{n-1}造成帧的重叠



0会被当做新的分组被接受

信道利用率

- 假设允许连续发送n帧。通常计算传送n帧的平均利用率，即：

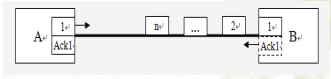
$$\text{信道利用率}=\frac{\text{发送n帧的时间}}{\text{成功传送n帧所需的总时间}}$$

- 两种类型：小窗口ARQ和大窗口ARQ

104

小窗口ARQ

- 小窗口：即$nT_D < (T_D + RTT + T_A)$
- 此时，发送方在发送完n帧后需等待对第1帧的确认后才能再发送新的帧。
- 发送n帧的时间为nT_D，数据帧从A站到B站所用时间，加上应答帧从B站到A站所用时间为$T_D + RTT + T_A$，因此
- 信道利用率$U=nT_D/(T_D + RTT + T_A)$



105

大窗口ARQ

- 大窗口：即$nT_D >= (T_D + RTT + T_A)$
- 此时信道没有任何空闲时间，因此信道利用率U=1

106

信道利用率 = n * 发送时间 /(发送时间 + 传播时延 + 应答时间)

连续ARQ和停等协议的信道利用率，分母不改变

停止等待协议和连续ARQ协议的问题

- 停止等待协议：
  - 发送-停止-等待，效率降低，当传播时间比发送时间大得多时，性能变得不可接受。
- 连续ARQ协议：
  - 未经确认的分组一次传送过多，如果出错，重传的代价太大；
  - 序号占的位数过多，影响效率。例如一次能够传送1024个分组，需10位编号。

107

TCP可靠通信的具体实现

- TCP 连接的每一端都必须设有两个窗口——一个发送窗口和一个接收窗口。
- TCP 的可靠传输机制用字节的序号进行控制。TCP 所有的确认都是基于序号而不是基于报文段。
- TCP 两端的四个窗口经常处于动态变化之中。
- TCP连接的往返时间 RTT 也不是固定不变的。需要使用特定的算法估算较为合理的重传时间。

108

滑动窗口协议

- 滑动窗口协议是停等协议和连续ARQ协议的折中。
- 窗口机制
  - 允许发送方连续发送多个分组而不需要等待应答。
    - 区别于停止等待协议
  - 一次连续发送未经确认的分组的个数是有限的。
    - 区别于连续ARQ协议
  - 发送端和接收端分别设定发送窗口和接收窗口 。

109

发送窗口

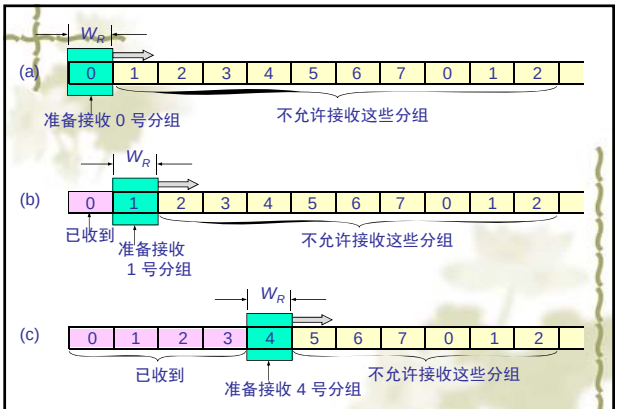
- 发送窗口的大小W_T表示：在还没有收到接收端确认信息的情况下，发送端最多可以发送多少个分组。
- 即只有落在发送窗口的分组才是可以发送的。
- 每收到对一个分组的确认，窗口就向前滑动。
- 停止等待协议的W_T =1，连续ARQ协议的W_T非常大。

110

接收窗口

- 接收窗口是接收端允许接收的分组的序号表。
- 只有落到接收窗口的分组才是可以接收的。
- 若接收到的分组落在接收窗口之外，一律将其丢弃。
- 在连续 ARQ 协议中，接收窗口的大小 W_R=1。
  - 只有当收到的分组的序号与接收窗口一致时才能接收该分组。否则，就丢弃它。
  - 每收到一个序号正确的分组，接收窗口就向前（即向右方）滑动一个分组的位置。同时发送对该分组的确认。

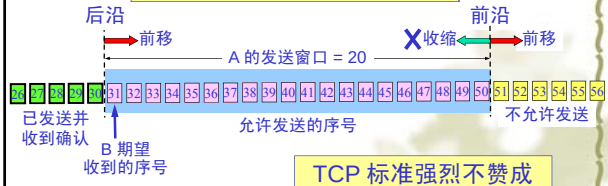
111



112

以字节为单位的滑动窗口

根据 B 给出的窗口值
A 构造出自己的发送窗口

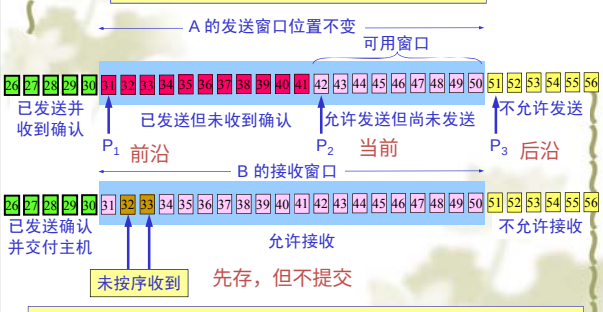


TCP 标准强烈不赞成
发送窗口前沿向后收缩

窗口前沿，尽量向前滑动

113

A 发送了 11 个字节的数据

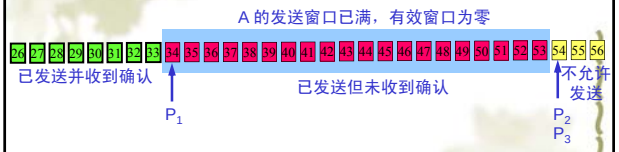


$P_3 - P_1$ = A 的发送窗口（又称为通知窗口）
$P_2 - P_1$ = 已发送但尚未收到确认的字节数
$P_3 - P_2$ = 允许发送但尚未发送的字节数（又称为可用窗口）

A 收到新的确认号，发送窗口向前滑动

115

A 的发送窗口内的序号都已用完，
但还没有再收到确认，必须停止发送。



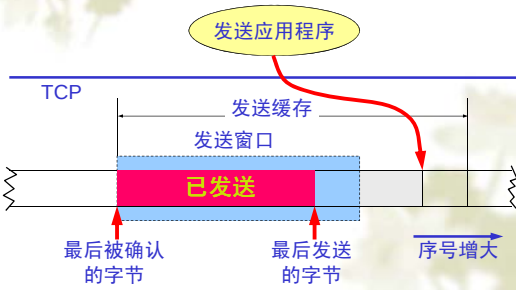
116

滑动窗口特性

- 只有在接收窗口向前滑动时（与此同时也发送了确认），发送窗口才有可能向前滑动。
- 收发两端的窗口按照以上规律不断地向前滑动，因此这种协议又称为滑动窗口协议。
- 当发送窗口和接收窗口的大小都等于1时，就是停止等待协议。

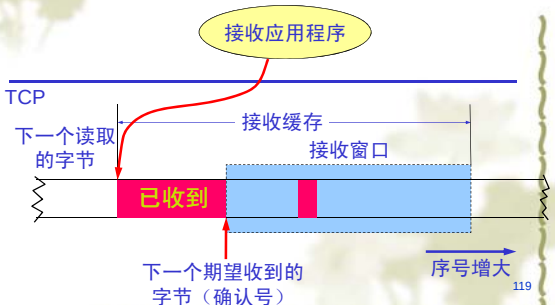
117

发送缓存



118

接收缓存



119

发送缓存与接收缓存的作用

- 发送缓存用来暂时存放：
  - 发送应用程序传送给发送方 TCP 准备发送的数据；
  - TCP 已发送出但尚未收到确认的数据。
- 接收缓存用来暂时存放：
  - 按序到达的、但尚未被接收应用程序读取的数据；
  - 不按序到达的数据。

120

需要强调三点

- A 的发送窗口并不总是和 B 的接收窗口一样大（因为有一定的时间滞后）。
- TCP 标准没有规定对不按序到达的数据应如何处理。通常是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。
- TCP 要求接收方必须有累积确认的功能，这样可以减小传输开销。

121

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。
- 如何选择合适的超时时间？
  - 由于 TCP 的下层是一个互连网环境，IP 数据报所选择的路由变化很大。
  - 信源和信宿之间的距离可远可近，且每个时刻网络的拥塞不一样。

122

往返时延 = 发出时间 - 确认时间
RTT ->再去加权平均

往返时延的自适应算法

- 记录每一个报文段发出的时间，以及收到相应的确认报文段的时间。这两个时间之差就是报文段的往返时延。
- 将各个报文段的往返时延样本加权平均，就得到加权平均往返时间 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，就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 很接近于零，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。

123

超时重传时间 RTO (RetransmissionTime-Out)

- 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：

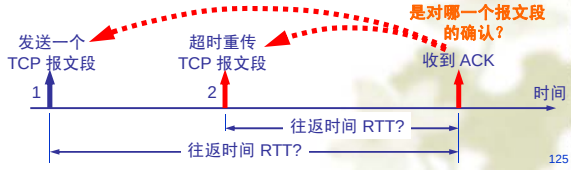
$$\text{新的 } RTT_D = (1-\beta) \times (\text{旧的} RTT_D) + \beta \times |RTT_S - \text{新的 RTT 样本}|$$

- β 是个小于 1 的系数，其推荐值是 1/4，即 0.25。

124

往返时间的测量相当复杂

- TCP 报文段 1 没有收到确认。重传（即报文段 2）后，收到了确认报文段 ACK。
- 如何判定此确认报文段是对原来的报文段 1 的确认，还是对重传的报文段 2 的确认？



125

重传 样本不采用，没重传一下把加时重传的加倍一下

Karn 算法

- 在计算平均往返时间 RTT 时，只要报文段重传了，就不采用其往返时间样本。
- 这样得出的加权平均往返时间 RTT_S 和超时重传时间 RTO 就较准确。

126

修正的 Karn 算法

- 报文段每重传一次，就把 RTO 增大一些：

新的 RTO = γ × (旧的 RTO)

- 系数 γ 的典型值是 2 。
- 当不再发生报文段的重传时，才根据报文段的往返时延更新平均往返时延 RTT 和超时重传时间 RTO 的数值。
- 实践证明，这种策略较为合理。

127

第5章 运输层

128

利用滑动窗口实现流量控制

- 一般说来，我们总是希望数据传输得更快一些。但如果发送方把数据发送得过快，接收方就可能来不及接收，这就会造成数据的丢失。

发送方：控制发送速率适合

- 流量控制(flow control)就是让发送方的发送速率不要太快，既要让接收方来得及接收，也不要使网络发生拥塞。
- 利用滑动窗口机制可以很方便地在 TCP 连接上实现流量控制。

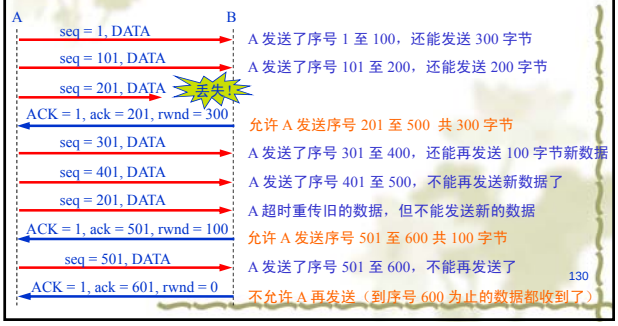
129

实现：滑动窗口机制

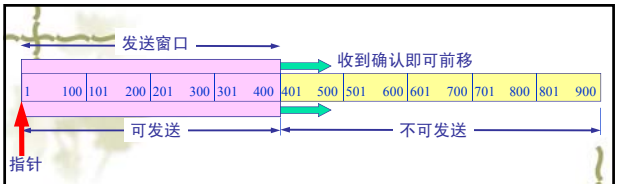
利用rwnd控制最大的范围

流量控制举例

A 向 B 发送数据。在连接建立时，
B 告诉 A：“我的接收窗口 rwnd = 400（字节）”。

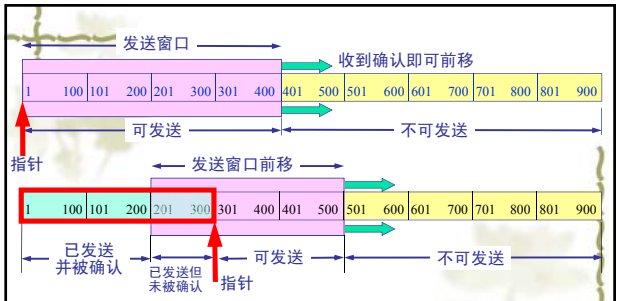


130



- 发送端要发送 900 字节长的数据，划分为 9 个 100 字节长的报文段。
- 发送窗口确定为 400 字节，初始序号设为1。
- 发送端只要收到了对方的确认，发送窗口就可前移。
- 发送 TCP 要维护一个指针。每发送一个报文段，指针就向前移动一个报文段的距离。

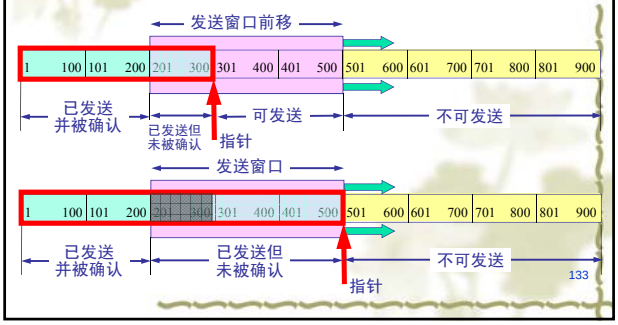
131



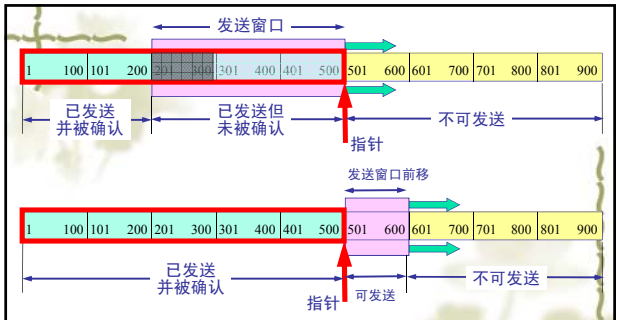
- 发送端已发送了 300 字节的数据，但只收到对前 200 字节数据的确认，同时窗口减小到300。
- ACK = 1, ack = 201, rwnd = 300
- 现在发送端还可发送 200 字节。

132

- 发送端没有收到新的确认，发送窗口不能前移，因此不能再发送新数据。
- 计时器超时，发送端重新发送编号201-300的数据。



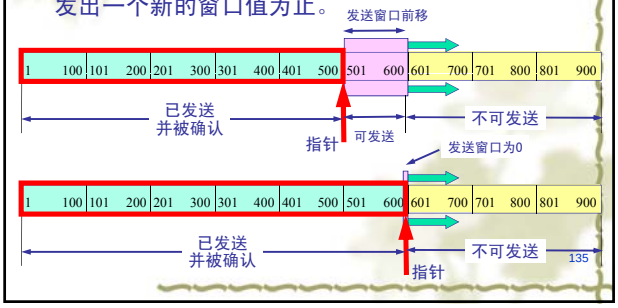
133



- 发送端收到对前 500 字节数据的确认，同时窗口减小到100。
- ACK = 1, ack = 501, rwnd = 100
- 现在发送端还可发送 100 字节。

134

- 发送端收到确认，但窗口减小到0，因此不能再发送新数据。
- ACK = 1, ack = 601, rwnd = 0
- 这种使发送端暂停发送的状态将持续到接收端重新发出一个新的窗口值为止。



135

持续计时器(persistence timer)

- TCP 为每一个连接设有一个持续计时器。
- 只要 TCP 连接的一方收到对方的零窗口通知，就启动持续计时器。
- 若持续计时器设置的时间到期，就发送一个零窗口探测报文段（仅携带 1 字节的数据），而对方就在确认这个探测报文段时给出了现在的窗口值。
- 若窗口仍然是零，则收到这个报文段的一方就重新设置持续计时器。
- 若窗口不是零，则死锁的僵局就可以打破了。

136

必须考虑传输效率

- 可以用不同的机制来控制 TCP 报文段的发送时机:
- 第一种机制是 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。
- 第二种机制是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。
- 第三种机制是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。

137

拥塞控制的一般原理

- 在某段时间，若对网络中某资源的需求超过了该资源所能提供的可用部分，网络的性能就要变坏——产生拥塞(congestion)。
- 出现资源拥塞的条件：

  对资源需求的总和 > 可用资源

- 若网络中有许多资源同时产生拥塞，网络的性能就要明显变坏，整个网络的吞吐量将随输入负荷的增大而下降。

138

拥塞控制与流量控制的关系

- 拥塞控制所要做的都有一个前提，就是网络能够承受现有的网络负荷。
- 拥塞控制是一个全局性的过程，涉及到所有的主机、所有的路由器，以及与降低网络传输性能有关的所有因素。
- 流量控制往往指在给定的发送端和接收端之间的点对点通信量的控制。
- 流量控制所要做的就是抑制发送端发送数据的速率，以便使接收端来得及接收。

139

TCP的拥塞控制

- TCP实体根据是否超时来判断网络拥塞。
- 除了接收窗口外，TCP还维护一个拥塞窗口，来避免网络拥塞。
  - 接收窗口rwnd (receiver window) ：是接收端根据其目前的接收缓存大小所许诺的最新的窗口值，是来自接收端的流量控制。接收端将此窗口值放在 TCP 报文的首部中的窗口字段，传送给发送端。
  - 拥塞窗口cwnd (congestion window)：是发送端根据自己估计的网络拥塞程度而设置的窗口值，并且动态地在变化。

140

无拥塞，多发；有拥塞，少发

发送窗口

- 发送窗口的上限值：
  - 发送端的发送窗口的上限值应当取为接收端窗口 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，是接收端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。

141

TCP拥塞控制算法

- 发送方控制拥塞窗口的原则是：只要网络没有出现拥塞，拥塞窗口就再增大一些，以便把更多的分组发送出去。但只要网络出现拥塞，拥塞窗口就减小一些，以减少注入到网络中的分组数。
- 慢启动阶段(slow-start)
- 拥塞避免阶段(congestion avoidance)
- 慢启动门限值(ssthresh)

142

慢开始门限ssthresh

- 当 cwnd < ssthresh 时，使用慢启动算法。
- 当 cwnd > ssthresh 时，使用拥塞避免算法。
- 当 cwnd = ssthresh 时，既可使用慢开始算法，也可使用拥塞避免算法。

143

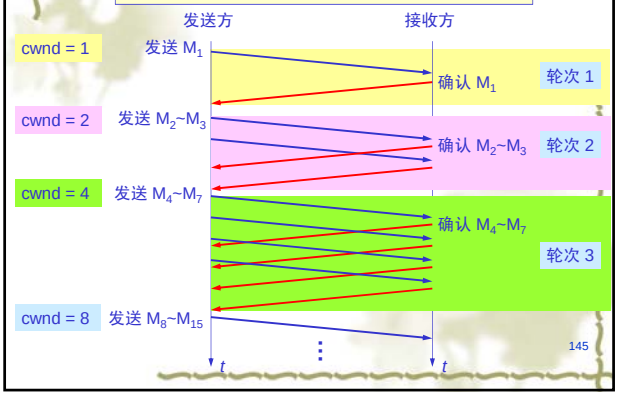
慢启动算法的原理

- 在主机刚刚开始发送报文段时可先设置拥塞窗口 cwnd = 1，即设置为一个最大报文段MSS的数值。
- 在每收到一个对新的报文段的确认后，将拥塞窗口加 1，即增加一个 MSS 的数值。
- 用这样的方法逐步增大发送端的拥塞窗口 cwnd，可以使分组注入到网络的速率更加合理。
- cwnd 呈指数级增长。

144

经过一个RTT时间

发送方每收到一个对新报文段的确认（重传的不算在内）就使 cwnd 加 1。



传输轮次(transmission round)

- 使用慢启动算法后，每经过一个传输轮次，拥塞窗口 cwnd 就加倍。
- 一个传输轮次所经历的时间其实就是往返时间RTT。
- “传输轮次”更加强调：把拥塞窗口 cwnd 所允许发送的报文段都连续发送出去，并收到了对已发送的最后一个字节的确认。
- 例如，拥塞窗口 cwnd = 4，这时的往返时间 RTT 就是发送方连续发送 4 个报文段，并收到这 4 个报文段的确认，总共经历的时间。

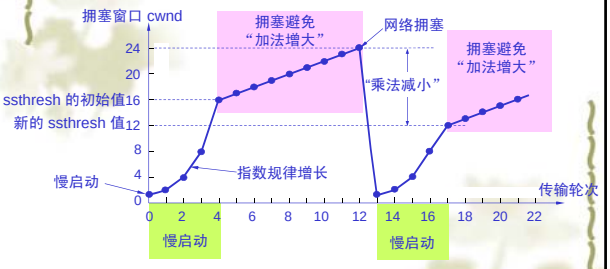
拥塞避免算法的思路

- 让拥塞窗口 cwnd 缓慢地增大，即每经过一个往返时间 RTT 就把发送方的拥塞窗口 cwnd 加 1，而不是加倍。
- 即每收到一个对新的报文段的确认后，将拥塞窗口增加1/cwnd，也就是增加 (MSS×MSS/cwnd) 的数值。
- cwnd 按线性规律缓慢增长。

当网络出现拥塞时

- 无论在慢启动阶段还是在拥塞避免阶段，只要发送方判断网络出现拥塞（其根据就是没有按时收到确认），就要把慢启动门限 ssthresh 设置为出现拥塞时的发送方窗口值的一半（但不能小于2）。
- 把拥塞窗口 cwnd 重新设置为 1，执行慢启动算法。
- 即： ssthresh=max(cwnd/2,2), cwnd=1
- 目的：迅速减少主机发送到网络中的分组数，使得发生拥塞的路由器有足够时间把队列中积压的分组处理完毕。

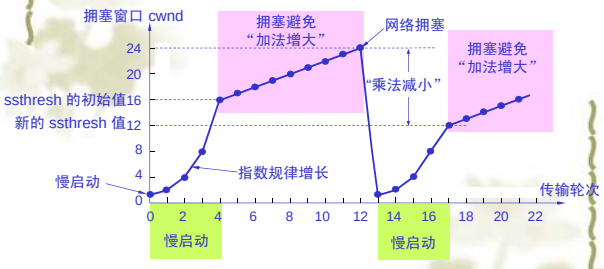
慢启动和拥塞避免算法的实现举例



当 TCP 连接进行初始化时，将拥塞窗口置为 1。图中的窗口单位不使用字节而使用报文段。

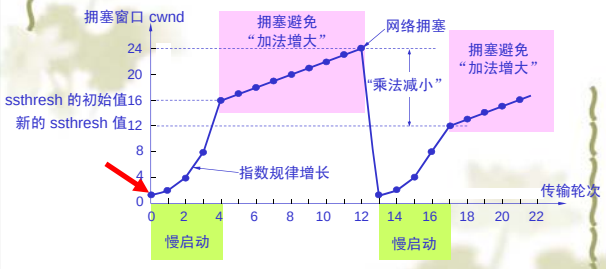
慢开始门限的初始值设置为 16 个报文段，即 ssthresh = 16。

慢启动和拥塞避免算法的实现举例



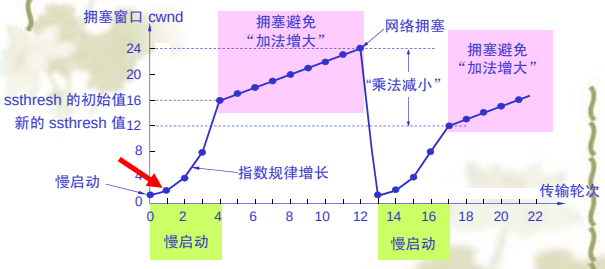
发送端的发送窗口不能超过拥塞窗口 cwnd 和接收端窗口 rwnd 中的最小值。我们假定接收端窗口足够大，因此现在发送窗口的数值等于拥塞窗口的数值。

慢启动和拥塞避免算法的实现举例



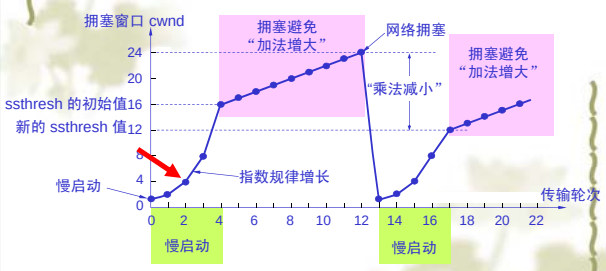
在执行慢启动算法时，拥塞窗口 cwnd 的初始值为 1，发送第一个报文段 M_0。

慢启动和拥塞避免算法的实现举例



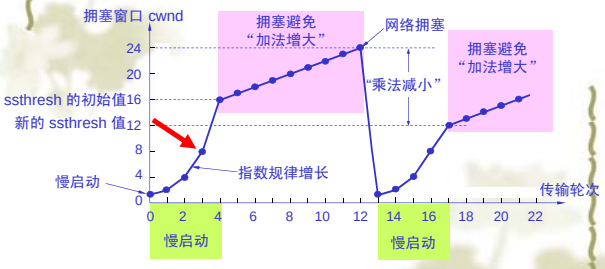
发送端每收到一个确认，就把 cwnd 加 1。于是发送端可以接着发送 M_1 和 M_2 两个报文段。

慢启动和拥塞避免算法的实现举例



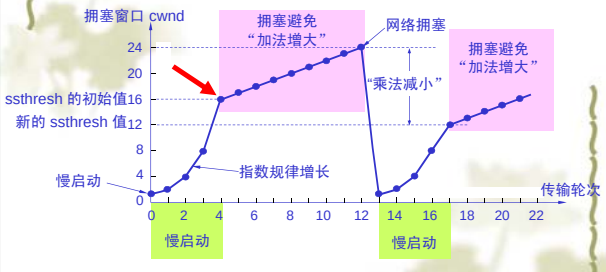
接收端共发回两个确认。发送端每收到一个对新报文段的确认，就把发送端的 cwnd 加 1。现在 cwnd 从 2 增大到 4，并可接着发送后面的 4 个报文段。

慢启动和拥塞避免算法的实现举例



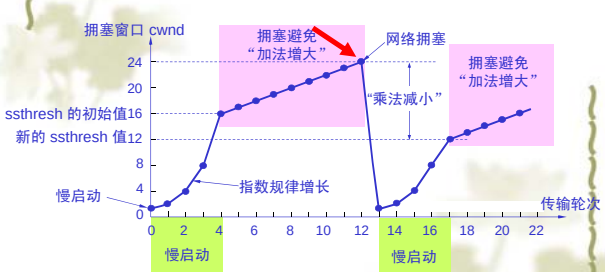
发送端每收到一个对新报文段的确认，就把发送端的拥塞窗口加 1，因此拥塞窗口 cwnd 随着传输轮次按指数规律增长。

慢启动和拥塞避免算法的实现举例



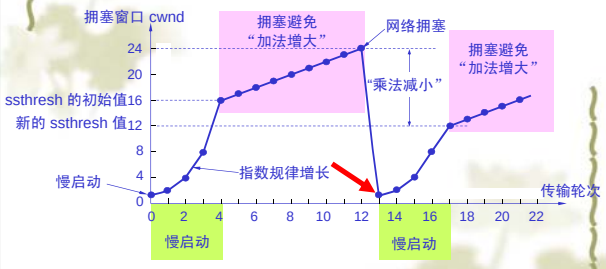
当拥塞窗口 cwnd 增长到慢启动门限值 ssthresh 时（即当 cwnd = 16 时），就改为执行拥塞避免算法，拥塞窗口按线性规律增长。

慢启动和拥塞避免算法的实现举例



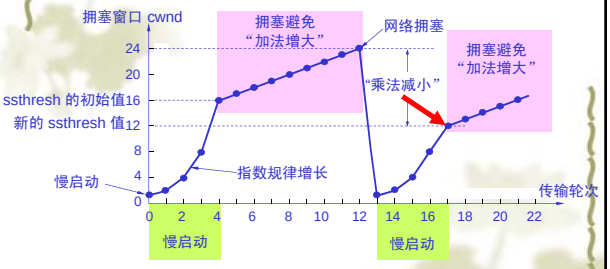
假定拥塞窗口的数值增长到 24 时，网络出现超时，表明网络拥塞了。

慢启动和拥塞避免算法的实现举例



更新后的 ssthresh 值变为 12（即发送窗口数值 24 的一半），拥塞窗口再重新设置为 1，并执行慢启动算法。

慢启动和拥塞避免算法的实现举例



当 cwnd = 12 时改为执行拥塞避免算法，拥塞窗口按按线性规律增长，每经过一个往返时延就增加一个 MSS 的大小。

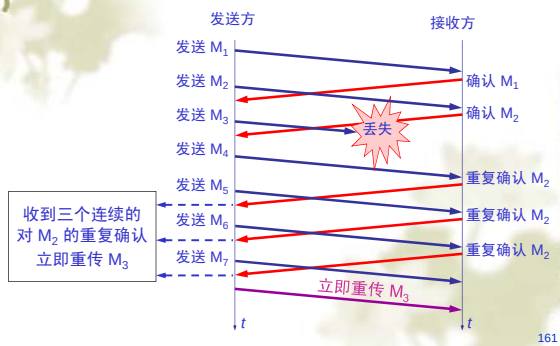
必须强调指出

- “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”是说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。

快速重传和快速恢复

- 快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。

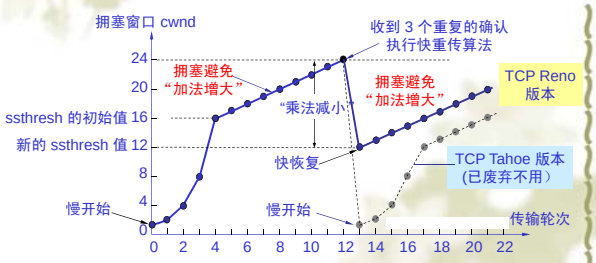
快速重传举例



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，把慢开始门限 ssthresh 减半。但接下去不执行慢开始算法。
- 由于发送方现在认为网络很可能没有发生拥塞，因此现在不执行慢开始算法，即拥塞窗口 cwnd 现在不设置为 1，而是设置为慢开始门限 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。

从连续收到三个重复的确认转入拥塞避免



163

本章小结

- 掌握运输层基本功能，特别是Internet的传输层协议TCP和UDP，以及TCP连接建立机制、流量控制机制、拥塞控制机制。

164