第5章 传输层

[本章主要内容]

1. 了解传输层的功能及端口、套接字的概念。

2. 了解 UDP 的特点、UDP 数据报格式、UDP 校验方法。

3. 重点掌握 TCP，包括 TCP 提供的服务、TCP 报文的格式、TCP 的连接管理。

4. 了解传输层可靠传输的原理的相关协议：停等协议、连续 ARQ 协议。

5. 掌握 TCP 可靠传输的实现的机制：包括滑动窗口协议的原理及 TCP 利用滑动窗口实现的差错控制、流量控制、拥塞控制方法。

5.1 传输层协议概述

5.1.1 划分传输层的必要性

首先我们以数据通信原理为基础具体了解各层的基本作用，以此来分析划分“传输层”的必要性。我们知道，物理层为数据通信提供实际的物理线路和通信信道，这是所有数据通信的基础；数据链路层为相邻结点提供通信，可以根据不同的链路类型对物理层的比特流进行帧封装和传输；网络层为主机间的数据通信提供了路由、转发功能，把数据包从一个网络中的主机选择最佳路径传送到位于另一个网络中的目的主机上。这时可能就有读者会问到，既然网络层已把源主机上发出的数据包传送给了目的主机，那么为什么还需要设置一个传输层呢？

首先，这是因为位于两个网络中主机间的真正数据通信主体不是这两台主机，而是两台主机中的各种应用进程。因为在同一时刻，两台主机间可以进行多个应用通信。例如，两个用户在进行视频电话时，还进行了 FTP 下载，或者同时还在收发电子邮件，这就需要多个应用进程同时通信，如图 5-1 所示（仅列出了一个方向）。正因为如此，在进行具体的网络通信应用时必须为每一个网络应用配备一个唯一的应用进程标识，否则所传输的报文就不知道要提交给哪个用户应用了。而这里的应用进程识别就是依靠本章所要介绍的“传输层”了，它是通过“端口”（具体概念见 5.1.4 节）来与不同应用进程进行对应的。所以仅靠网络层把数据传送到目的主机上还是不够的，来自应用层的用户数据必须依靠传输层协议在不同网络中的主机间进行传输，最终把用户数据交给目的主机的应用进程。

其次，由于许多通信子网只能提供“尽力交付”的服务，就是说网络层不能保证发送端可

靠地把数据送至目的端。因此传输层要实现的功能就很复杂的。传输层要保障通过传输协议，可以把尽力交付的不可靠的网络服务演变成为可靠的网络服务。因此，设计传输层的目的也是提高传输服务的可靠性与保证服务质量（QoS），并在“点—点”通信的基础上完成“端—端”通信，实现一次质的飞跃。为此传输层需要引入很多新的概念和机制。

第三，在 OSI/RM 网络体系结构中，传输层位于第 4 层，在网络层之上，会话层之下。从通信和信息处理的角度看，“传输层”既是面向通信部分的最高层，它与下三层共同构建进行网络通信所需的线路和建立数据传输通道并向高层提供通信服务；同时又是面向用户的最低层，因为无论何种网络应用，最终都需要把各种数据传送到对方。所以，“传输层”的位置非常特殊，起到一个承上启下的桥梁作用。当网络的边缘部分的两台主机使用网络的核心部分的功能进行端到端的通信时，只有主机的协议栈才有传输层，而网络核心部分中的路由器在转发分组时都只需要下三层的功能。

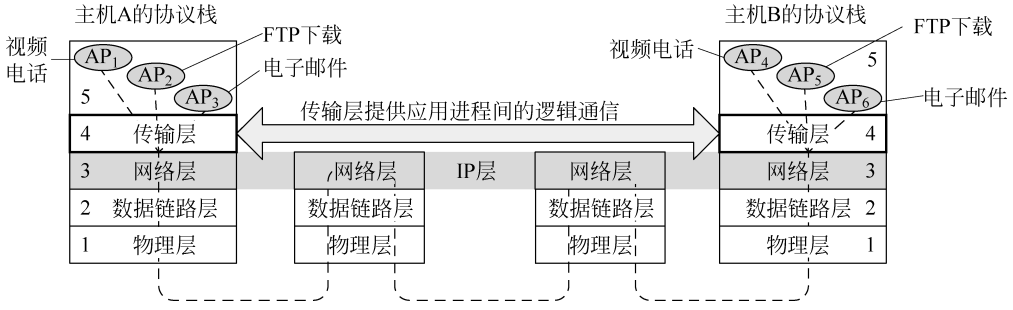

图 5-1 传输层的概念

5.1.2 传输层的功能

传输层的基本功能是利用网络层提供给它的服务去开发自己本层的功能，在两个应用进程之间提供面向连接的、可靠的、全双工的进程到进程的（端到端的）传输，实现本层对上一层的服务。针对这些需求，传输层应具有如下功能：

（1）提供端到端（应用进程之间）的通信。传输层协议是利用网络层所提供的不可靠的主机到主机服务的基础上，在源主机的应用进程与目的主机的应用进程之间提供可靠的、端到端的（应用进程之间的）、全双工的数据传输。如图 5-2 所示为传输层协议和网络层协议的作用范围。

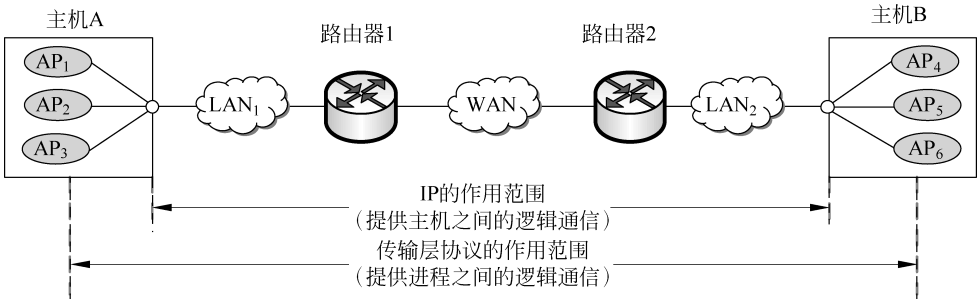

图 5-2 传输层协议和网络层协议的作用范围

另外,传输层可以屏蔽下面网络核心细节(如网络拓扑结构、所采用的路由选择协议等)的差异性,弥补网络层所提供服务的不足,使得应用层在设计各种网络应用系统时,只需要考虑选择什么样的传输层协议可以满足应用进程通信的要求,使应用进程看见的就是好像在两个传输层实体之间有一条端到端的逻辑信道,而不需要考虑数据传输的细节问题。

(2) 差错控制。互联网中的路由器与通信线路构成了传输网。传输网一般是由电信公司运营和管理的。如果传输网提供的服务不可靠(例如频繁丢失报文),用户无法对传输网加以控制。解决这个问题需要从两个方面入手:一是电信公司进一步提高传输网的服务质量;二是传输层对报文丢失、线路故障进行检测,并采取相应的差错控制措施,在不可靠的线路上实现可靠的通信。而 TCP/IP 的网络层,IP 数据报首部的校验和字段,只对首部进行差错检验而不对数据部分进行检查,所以,需要传输层对收到的报文进行差错检测。这样才能保证传输层接收到的数据的可靠性。在 TCP/IP 的传输层中两个协议都能实现对报文的差错检验。

(3) 复用和分用的功能。如果在一台主机中进程有多个应用进程同时分别和另一台主机中的多个应用进程通信,如图 5-1 所示。主机 A 的应用进程 AP_1(视频电话)与主机 B 的应用进程 AP_4 通信,与此同时,主机 A 的应用进程 AP_2(FTP 下载)与主机 B 的应用进程 AP_5 通信及主机 A 的应用进程 AP_3(电子邮件)与主机 B 的应用进程 AP_6 通信。TCP/IP 允许源主机应用层的不同进程的报文分别通过不同的端口向下交到传输层,传输层使用其复用功能向下共用网络层(即 IP 层)提供的服务。这表明,传输层有一个很重要的功能——复用(multiplexing)和分用(demultiplexing)。这里的“复用”是指在发送方 IP 将 TCP 或 UDP 的传输协议数据单元 TPDU 都封装成一个 IP 分组发送出去,多个不同的应用程序的数据,同时使用同一个 IP 地址和物理链路来发送。这些分组沿着图中虚线到达目的主机。而“分用”是指在接收方,IP 协议将从 IP 分组中拆开的传输协议数据单元 TPDU 传送到传输层,由传输层根据不同 TPDU 的端口号,区分出不同 TPDU 的种类,分别传送给对应的 3 个应用进程。图 5-1 给出了传输层的多路复用与多路分解过程示意图。

应用进程的报文从源主机的传输层到目的主机的传输层,从效果上看,就好像是直接沿水平方向传送到远地的传输层(这就是图 5-1 中两个传输层之间用一个粗的双向箭头表示的意思),即传输层提供传输实体间的逻辑通信。所谓逻辑信道就是好像在两个传输实体之间有一条端到端的通信信道,实际上这两个应用进程之间并没有一条水平方向的物理连接,要传送的数据是沿着图中上下多次的实线方向传送的。

当这些报文沿着图中的实线到达目的主机后,目的主机的传输层就使用其分用功能,通过不同的端口将报文分别交付到相应的应用进程。

(4) 可靠与不可靠交付。为适应不同的应用需要,传输层的这条逻辑信道对上层的表现却因为传输层使用不同的协议而有很大的差别。当传输层采用面向连接的 TCP 时,尽管下面的网络只提供最大努力服务,不能保证可靠传输,但是这种逻辑通信信道相当于一条全双工的可靠信道,报文在这样的信道中传输,可以做到无差错、按序(接收的顺序和发送的顺序一样)、无丢失、无重复。但当传输层采用无连接的 UDP 时,这种逻辑通信信道则是不可靠的。

在图 5-3 中我们将可靠信道画成一个管道,这意味着报文在这样的“管道”中传输时,可以做到无差错、按序、无丢失和无重复。对不可靠信道就用一个云状网络来表示。不可靠信

道的特点就是不保证交付，即接收时可能不按序、可能会出现丢失和重复。但传输层检查出到达的报文有差错时就将其丢弃，不会接收有差错的报文。

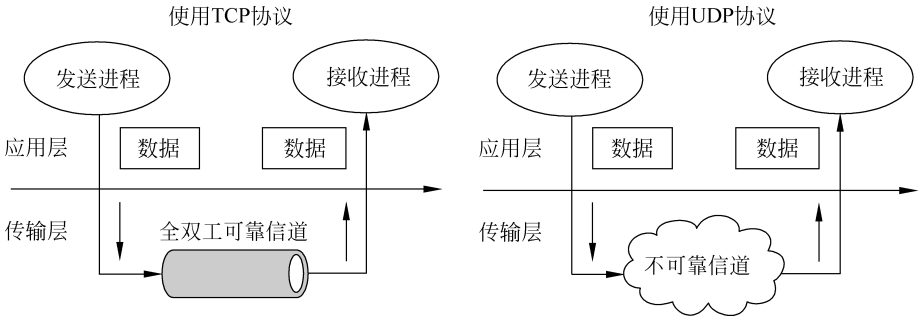

图 5-3 传输层向上提供可靠的或不可靠的逻辑通信信道

(5) 无连接与面向连接。UDP 提供无连接的服务，在传送数据之前不需要建立连接，远程主机的传输层在收到 UDP 报文后，也不需要给出任何确认。因此，UDP 不提供可靠交付。尽管如此，UDP 有时却是一种有效的工作方式，例如 DNS、SNMP 和 NFS 等应用服务器都使用 UDP 传输方式。虽然使用 UDP 不能保证传输的安全可靠，但由于它的一些机制的运用(例如：UDP 没有拥塞控制)，在一些实时应用的场合，它更加有效。在那些应用中差错控制由应用层的进程提供。

TCP 则提供面向连接的服务，在传送数据之前必须先建立连接，数据传送结束后要释放连接。TCP 不提供广播或多播服务，而且由于 TCP 要提供可靠的、面向连接的服务，因此要增加许多的开销，如确认、流量控制、差错控制、重传机制、计时器及连接管理等。通过接下来的学习会发现，TCP 首部的长度相对 UDP 要大很多，并且实现 TCP 机制需要更多的处理机资源。

总之，传输层提供可靠的交付，是指传输层将数据可靠地交付给接收端。

5.1.3 TCP/IP 传输层的两个协议——UDP 与 TCP

如上节所述，TCP/IP 的传输层有两个不同的协议，如图 5-4 所示，它们都是互联网的正式标准。

(1) 用户数据报协议(User Datagram Protocol，UDP)[RFC 768]。这是一种不可靠无连接的传输层协议。目的主机收到 UDP 报文后，不需要给出任何确认。

(2) 传输控制协议(Transmission Control Protocol，TCP)[RFC 793]。这是一种可靠的面向连接的协议。

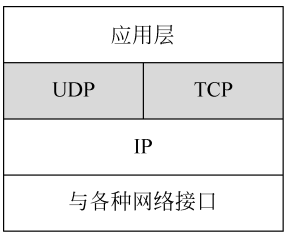

图 5-4 TCP/IP 体系中的传输层中的两个协议

而传输层通过实现 TCP 和 UDP 两种不同的协议使得对应用层的表现很不相同。

在传输层之间传输的报文叫作传输协议数据单元 TPDU。TPDU 有效载荷(数据部分)是应用层交付给传输层的数据,传输层在有效载荷之前加上传输层首部就形成了 TPDU。TPDU 传送到网络层后,加上 IP 分组首部后形成 IP 数据报;IP 数据报传送到数据链路层后,加上帧头、帧尾形成数据链路层的帧。帧经过物理层传输到目的主机后,经过数据链路层与网络层处理,传输层接收到 TPDU 后,读取 TPDU 首部,按照传输层协议的要求完成相应的工作。图 5-5 描述了 TPDU 结构与 IP 分组、帧结构之间的关系。

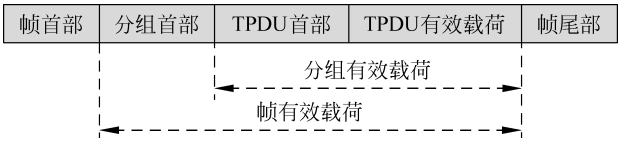

图 5-5 TPDU 结构、IP 分组及帧结构的关系

但在 TCP/IP 体系中,根据所使用的协议是 TCP 或 UDP,分别为 TCP 报文段或 UDP 用户数据报。

另外,对于不同的应用层协议有的使用 TCP,有的使用 UDP,表 5-1 中列出了应用层协议对应的传输层协议。

表 5-1 应用层协议对应的传输层协议

名字转换	DNS	UDP
文件传输	TFTP	UDP
路由选择协议	RIP	UDP
IP 地址配置	DHCP	UDP
网络管理	SNMP	UDP
远程文件服务器	NFS	UDP
IP 电话	专用协议	UDP
流式多媒体通信	专用协议	UDP
多播	IGMP	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

5.1.4 传输层端口

传输层的功能是实现应用进程间端到端的通信。计算机中的不同进程可以同时进行通信的原因是同一主机中的不同应用进程都被赋予一个非常明确的标志——端口号。这也是能实现传输层复用和分用功能的根本所在。

1. 端口的意义

端口位于传输层与应用层接口处,它就是 OSI/RM 的传输层服务访问点 TSAP。TCP

和 UDP 都使用端口与上层的应用进程进行通信，即应用层的各种进程通过相应端口将数据向下交给传输实体。反之，当传输层收到 IP 层交上来的数据（TCP 报文段或 UDP 用户数据报）时，也要根据其首部的端口号来决定应当通过哪一个端口上交给应该接收此数据的应用进程。由此可知，没有端口，传输层就无法知道应将接收的数据交给应用层的哪一个进程，也就谈不上应用进程与传输实体进行交互。所以，端口是用来标识应用进程的。

由此可见，两个计算机中的进程要互相通信，不仅要知道对方的 IP 地址，还要知道对方的端口号。

但是，由于在传输层使用了复用和分用技术，在传输层与网络层的交互中看不见各种应用进程，而只有 TCP 报文段或 UDP 用户数据，就如同网络层和数据链路层的交互只有 IP 数据报一样。

端口号是一个整数描述符，用来标识不同的端口或进程。端口号只用来标识本主机应用层中的各个进程，不同主机中的相同端口号之间没有联系，端口号只具有本地意义。

还需注意的是，这种在协议栈间的抽象的协议端口是软件端口，和路由器或交换机上的硬件端口是完全不同的概念。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传输层实体间交互的地址。

2. 端口的格式

在 TCP/IP 传输层，规定使用一个 16 位表示的整数作为端口标识，也就是说可定义 2^{16} 个端口号，其端口号为 $0 \sim 2^{16}-1$。由于 TCP/IP 传输层的 TCP 和 UDP 协议是两个完全独立的软件模块，各自的端口号也相互独立，即各自可独立拥有 2^{16} 个端口号。TCP 和 UDP 都采用 16 位的端口号，分别提供 65 536 个端口。

如图 5-6 所示，每种应用层协议或应用进程都具有与传输层唯一连接的端口，并且使用唯一的端口号将这些应用进程区分开来。当数据流从一个应用进程发送到远程网络中的某一个应用进程时，传输层可以根据这些端口号，就能够判断出数据来自于哪一个应用进程，想要访问另一个网络的哪一个应用进程，从而将数据传递到相应的应用进程。

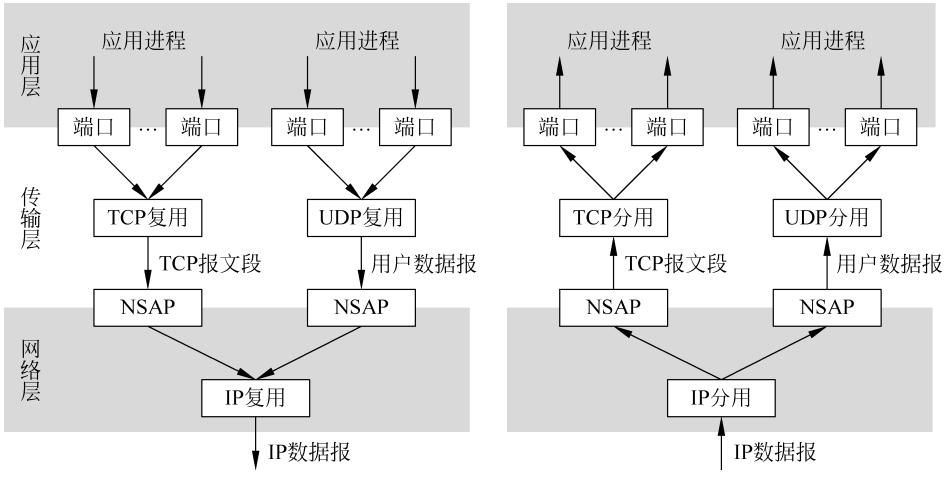

图 5-6 应用层与传输层之间的接口

3. 端口分配

端口根据其对应的协议或应用不同,被分配了不同的端口号。目前,传输层使用的端口共分为下面两大类。

(1) 服务器端使用的端口号

服务器进程必须使用一个端口号定义它自己,但这个端口号不能随机选择。如果一个服务器站点的计算机运行一个服务器进程,并随机分配一个数字作为端口号,那么当客户站点的进程想访问该服务器进程时,并不知道其端口号,则访问无法进行。

服务器端使用的端口号分为两种:最重要的一类叫作熟知端口(保留端口、全局端口或系统端口号)。这一类是由国际互联网代理成员管理局 IANA 把这些端口分配给 TCP/IP 最重要的一些应用程序,让所有用户都知道。熟知端口号一般为 0～1023。例如,分配给 FTP 的端口号为 21,Telnet 的端口号为 23,SMTP 的端口号为 25,DNS 的端口号为 53,HTTP 的端口号为 80 等。目前,熟知端口的端口号分配已经被广大网络应用者接受,形成了标准,如表 5-2 所示为 TCP 和 UDP 的一些常用的熟知(保留)端口。

表 5-2 TCP 和 UDP 的一些常用的保留端口

项目	端口号	关键字	应用协议
UDP 保留端口举例	53	DNS	域名服务
	67/68	DHCP	动态主机配置协议
	69	TFTP	简单文件传输协议
	161/162	SNMP	简单网络管理协议
	520	RIP	RIP 路由选择协议
TCP 保留端口举例	21	FTP	文件传输协议
	23	Telnet	虚拟终端协议
	25	SMTP	简单邮件传输协议
	53	DNS	域名服务
	80	HTTP	超文本传输协议
	119	NNTP	网络新闻传输协议
	179	BGP	边界路由协议
	3389	RDP	远程桌面协议

在各种网络的应用中调用这些端口号就意味着使用它们所代表的应用协议。由于这些端口已经有了固定的使用者,所以不能再被分配给其他应用程序。

另一类叫注册端口(登记端口)。注册端口比较特殊,是为没有熟知端口号的应用程序使用的。它所代表的不是已经形成标准的应用层协议,而是某个软件厂商开发的应用程序。一般来说,这些特定的软件要使用注册端口,其厂商必须向端口的管理机构注册,以防止重复。大多数注册端口的端口号大于 1023,为 1024～49 151。

(2) 客户端使用的端口号

客户进程用端口号定义它自己,这称为临时端口号(动态端口或一般端口)。为了客户/服务器进程能正常通信,临时端口号推荐值大于 49 151。这类端口号仅在客户进程运行时才动态选择。它的端口号一般为 49 152～65 535。这些端口号没有固定的使用者,可以动

态地分配给请求通信的应用进程使用。如图 5-7 所示，A 主机打开浏览器，准备访问百度和沈阳理工大学网站，这就需要建立两个 TCP 连接。A 主机会给每一个 TCP 连接分配一个客户端端口（要求本地唯一），从百度和沈阳理工大学网站返回的报文段的目的端口分别是 50000 和 50001，这样 A 计算机就知道这些报文段来自哪个网站，应交给哪一个应用进程。

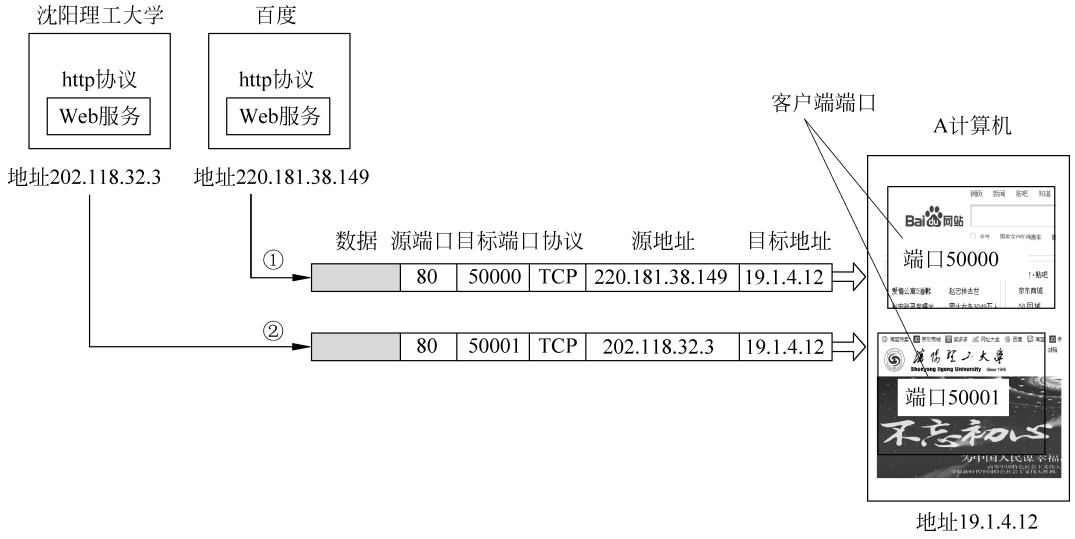

图 5-7 客户端端口示例

如图 5-8 给出 IANA 对于端口号数字范围的划分。

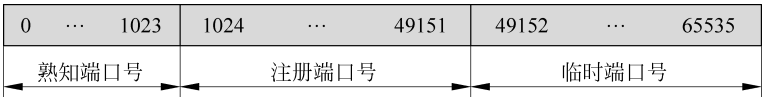

图 5-8 IANA 对于端口号数字范围的划分

5.1.5 套接字

传输层需要解决的一个重要问题是进程标识。在一台计算机中，不同进程需要用进程号（Process ID）唯一地标识，进程号也称为端口号。在网络环境中，为了区别不同主机的不同进程，标识一个进程必须把主机的 IP 地址（32 位）和端口号（16 位）结合在一起使用，这样形成的 48 位可以唯一确定网络中的一个传输层端点，这样的端点也叫作插口或套接字（Socket）。RFC 793 定义的套接字是由 IP 地址与对应的端口号（IP 地址：端口号）组成。只有知道 IP 地址与端口号，才能唯一地找到准备通信的进程。

1. 套接字格式

套接字（插口）是 IP 地址加上一个端口地址。这个概念不复杂，但非常重要。例如：如果 IP 地址＝130.24.27.1，端口号＝49152，则：

套接字＝（IP 地址：端口号）＝（130.24.27.1：49152）

套接字又可分为发送套接字和接收套接字。

发送套接字＝源 IP 地址＋源端口号

接收套接字＝目的 IP 地址＋目的端口号

有了套接字(或插口)的概念,那么一个 TCP 连接就可由它的两个端点来标识。

2. 套接字举例

例如在图 5-9 中,服务器端运行了 Web 服务、邮件服务和文件传输服务,这三个服务分别使用 HTTP、SMTP 和 FTP 与客户端通信。现在客户端主机 A、B、C 分别要访问 Web 服务、邮件服务和文件传输服务。发送了三个报文段①②③,这三个报文段的源端口都是临时端口号,目标端口号分别是 80,25 和 21,服务器收到这三个报文段,就根据目标端口地址将报文段提交给不同的应用进程。

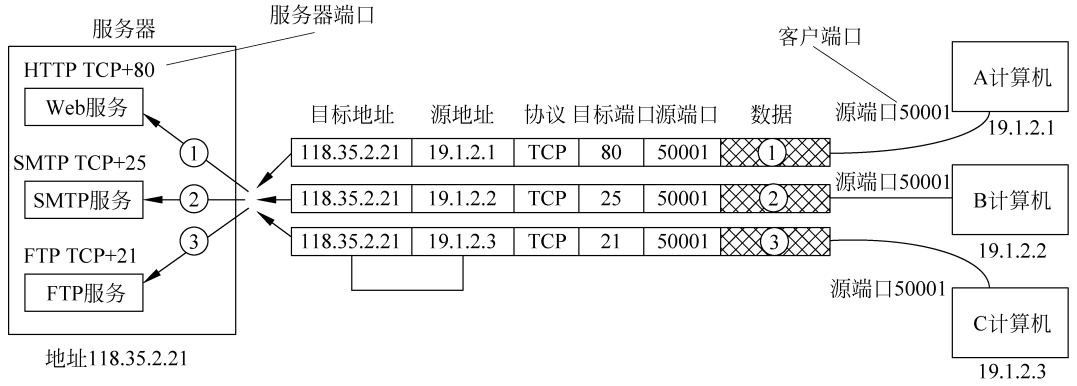

图 5-9 服务器与主机 A、B、C 与服务器建立 3 个连接

需要注意的是:

(1) 源端口是计算机临时为客户端进程分配的,当服务器向 A、B、C 发送响应报文段,源端口就变成了目标端口。

(2) 报文段的目的 IP 地址用来定位网络中的某一个服务器,而目的端口号是用来定位服务器中的某个应用进程。

现在,在整个互联网中,IP 地址为 19.1.2.1 的主机 A 用端口 50001 和 IP 地址为 118.35.2.21 的服务器端口 80 建立了连接①,相应的一对插口是:

发送套接字＝(19.1.2.1:50001)

接收套接字＝(118.35.2.21:80)

IP 地址为 19.1.2.2 的主机 B 用端口 50001 和 IP 地址为 118.35.2.21 的服务器端口 25 建立了连接②,连接②的一对插口是:

发送套接字＝(19.1.2.2:50001)

接收套接字＝(118.35.2.21:25)

IP 地址为 19.1.2.3 的主机 C 用端口 50001 和 IP 地址为 118.35.2.21 的服务器端口 21 建立了连接③,连接③的一对插口是:

发送套接字＝(19.1.2.3:50001)

接收套接字＝(118.35.2.21:21)

如果使用无连接的 UDP，虽然不需要在相互通信的两个进程之间建立一条虚连接，但为了区分多个主机之间同时通信的多个进程，发送端 UDP 一定要有一个发送端口，而在接收端 UDP 也一定要有一个接收端口，因而同样可以使用套接字（或插口）的概念。

端口在传输层的作用有点类似 IP 地址在网络层的作用或 MAC 地址在数据链路层的作用，只不过 IP 地址和 MAC 地址标识的是主机（一个是逻辑地址，一个是物理地址），而端口标识的是应用进程。由于同一时刻一台主机上可能会有许多的应用进程在运行，所以需要有大量的端口号来标识（或区分）不同的应用进程。

正是由于 TCP 和 UDP 使用通信端口来识别连接，才使得一台主机上的某个 IP 地址可以被多个连接所共享。

5.1.6 TCP、UDP 与应用层协议的关系

传输层协议与应用层协议的关系如图 5-10 所示。从图中可以看出，应用层协议可以分为三类，一类依赖于 UDP，一类依赖于 TCP，另一类既依赖于 UDP 协议又依赖于 TCP。依赖于 TCP 的应用层协议主要是需要大量传输交互式报文的一类应用，这类协议包括虚拟终端协议（Telnet）、电子邮件协议（SMTP）、文件传输协议（FTP）以及超文本传输协议（HTTP）等。简单网络管理协议（SNMP）依赖于 UDP。域名服务（DNS）协议既可以使用 TCP，也可以使用 UDP。但是所有的 TCP 报文与 UDP 用户数据报在网络层都使用 IP。UDP 简洁、效率高、处理速度快的优点，在 P2P 会话类应用中显得更为突出。

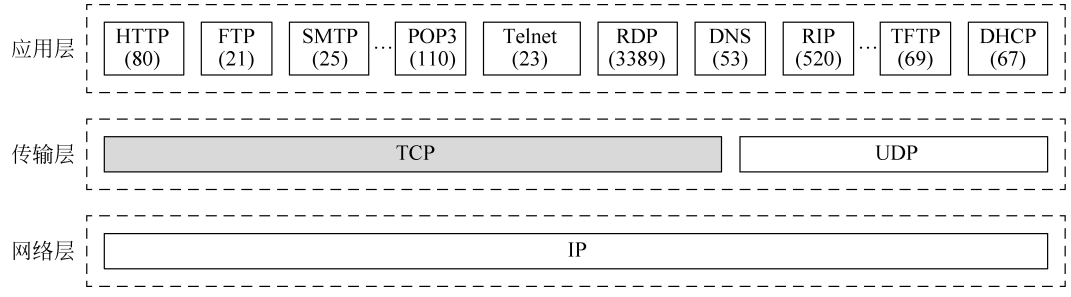

图 5-10 TCP、UDP 与应用层其它协议的关系

5.2 UDP

TCP 和 UDP 都是传输层协议，但它们实现的功能不同，因此其首部也不相同，本节讲解 UDP 的特点及首部格式，5.3 节我们将展开对 TCP 的介绍。1980 年公布的文档 RFC 768 是 UDP 的协议标准。

5.2.1 UDP 的主要特点

UDP 只是在 IP 的数据报服务之上增加了很少的一点功能，即复用和分用以及差错检测的功能。设计 UDP 的主要原则是协议简洁，运行快。这里所说的复用和分用，就是使用端口标识不同的应用层协议。UDP 的主要特点是：

(1) 提供进程到进程的通信。UDP 使用套接字地址提供进程到进程的通信。

(2) UDP 是一种无连接的协议。UDP 是无连接的,即在发送数据之前不需要建立连接,发送数据结束后也不需要连接释放,因而减少了许多协议开销和数据发送之前的时延。因此它只提供进程到进程之间的通信,复用和分用及有限的差错校验功能。那么为什么 UDP 功能如此之差,还有应用进程使用它? 是因为在实现一些网络应用(如对网络进行故障检测)时,是无法或不能建立连接的,这时只能使用 UDP 协议。

(3) UDP 只是尽最大努力交付的不可靠的传输层协议。它提供了有限的差错检验功能。除校验和外,UDP 也没有差错控制机制。这就表示发送方不知道报文是否丢失、是否重复。当接收方使用校验和检测出差错时,它就悄悄地将此报文丢弃。既不确认,也不通知发送端重传。即不保证可靠交付,但节省了系统资源。

(4) UDP 用户数据报只有 8 个字节的首部,比 TCP 固定首部的 20 字节要短,开销要小。设计比较简单的 UDP 的目的是希望以最小的开销来达到网络环境中进程通信的目的。

(5) UDP 不使用拥塞控制。因为不保证可靠交付,所以主机不需要维持具有许多参数的连接状态表。当网络出现拥塞时,也不会使源主机的发送速率降低,这对某些实时应用是非常重要的。例如 IP 电话、实时视频会议等都要求源主机以恒定的速率发送数据,但对于在网络出现拥塞时丢失少量的数据却是可以容忍的。也就是说,即便因为网络拥塞而丢失了一些数据,也不允许数据时延太大。UDP 不使用拥塞控制正好满足这种要求。在某些情况下,UDP 中缺少拥塞控制可以看作是一个优势。

(6) 无流量控制。UDP 无流量控制,因而也没有窗口机制。但是如果到达的报文太多时,接收方可能会溢出。因此,UDP 提供的是“尽力而为”的传输服务。

(7) UDP 是面向报文的,对应用进程交下来的报文不再划分为若干个小报文段来传送,既不合并,也不拆分。也就是说,应用层交给 UDP 多长的报文,UDP 就原样发送,即一次发送一个报文。如图 5-11 所示,在接收端的 UDP,对 IP 层交上来的一个完整报文,在去除首部后就原封不动地交付给上层的应用进程,就要求应用进程要选择大小合适的报文。若报文太长,UDP 把它交给 IP 层后,IP 层在传送时可能会分片,这会降低 IP 层的效率。反之,若报文太短,UDP 把它交给 IP 层后,加上 IP 数据报相对较长的首部,这样使网络的利用率较低。

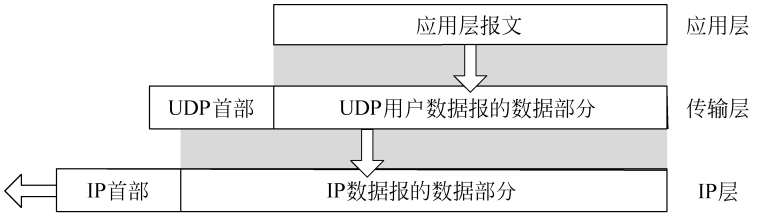

图 5-11 UDP 是面向报文的协议

(8) UDP 支持一对多、一对一、多对多和多对一通信。TCP 只支持一对一的通信。但是由于 UDP 不具有拥塞控制功能,所以在一些实时通信中可能会满足应用的需要,但当很多的源主机同时向网络发送高速的实时视频流时,网络就可能会发生严重的拥塞,造成大量

数据的丢失，以致大家都无法正常接收。因此，在一些使用 UDP 的实时应用要求对 UDP 的不可靠传输进行适当改进，以减少数据的丢失。为此，可以在不影响实时性的前提下，在应用进程本身增加一些提供可靠性的措施，例如采用前向纠错或重传已丢失的报文等。

UDP 常用于一次性传输数据量较小、可靠性要求不高的网络应用，如 SNMP、DNS 等应用数据的传输。进程发送的报文较短，同时对报文的可靠性要求不高，那么可以使用 UDP 协议。因为对于这些一次性传输数据量较小的网络应用，若采用 TCP 服务，则付出的连接建立、维护和拆除的开销是非常不合算的。

5.2.2 UDP 用户数据报格式

UDP 用户数据报包括首部和数据两部分。首部只有 8 个字节，分 4 个字段，每个字段为两个字节，如图 5-12 所示。

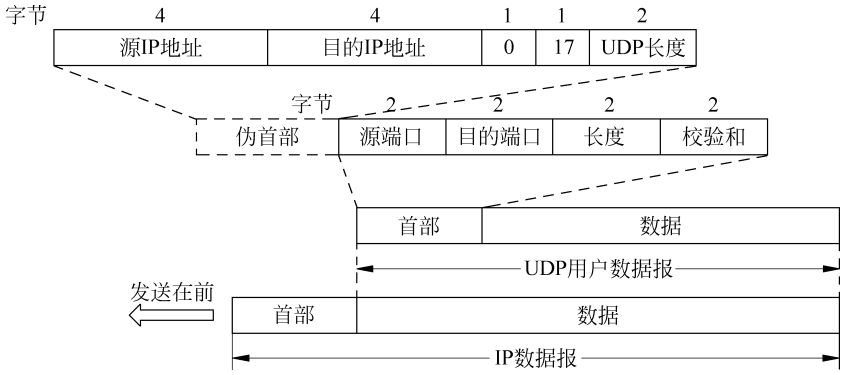

图 5-12 UDP 用户数据报格式

各字段意义如下：

(1) 源端口号：长度 16 位。源端口号是在源主机运行的进程使用的端口号。源端口号是一般端口号(或动态分配的端口)，它由应用进程请求，由源主机上运行的 UDP 软件进行分配。一般在需要对方回信时选用。不需要时可用全为 0。

(2) 目的端口号：长度 16 位。它是服务器端的端口号，在接收端交付报文时必须使用。如果接收端 UDP 发现收到的报文中的目的端口号不正确，就丢弃该报文。

在传输过程中，源端口号和目的端口号根据发送和接收者的身份来确定。

(3) 长度字段：长度 16 位。它定义了包括用户数据报首部在内的用户数据报的总长度。因此其最大长度为 65 535 字节，最小长度是 8 字节(只有首部，而没有数据)。

(4) 校验和字段：长度 16 位。UDP 校验和是可选的，是用来防止 UDP 用户数据报在传输中出错之用，出错就丢弃。校验和字段用来检验整个用户数据报(包括首部)在传输中是否出现差错，发送方可以选择不计算校验和，这一点正反映出设计者效率优先的思想。因为计算校验和肯定是要花费时间的，如果应用进程对通信效率的要求高于可靠性时，应用进程可以不选择校验和。

UDP 校验和要检验的内容包括三个部分：伪首部(Pseudo Header)、UDP 首部以及 UDP 数据部分。伪首部是 IP 分组首部的一部分，其中填充域字段要填入 0，目的是使伪首

部的长度为16位的整数倍。协议字段填入协议号17,17表示UDP。UDP长度包括UDP数据报的长度,不包括伪首部的长度。

所谓"伪首部",是因为这种伪首部并不是UDP用户数据报真正的首部。只是在计算校验和时,临时和UDP用户数据报连接在一起,得到一个过渡的UDP用户数据报。校验和就是按照这个过渡的UDP用户数据报来计算的。伪首部既不向下传递也不向上递交,而仅仅是为了计算校验和。图5-12的最上面给出了伪首部各字段的内容。

使用伪首部的目的是验证UDP数据报是否传到正确的目的进程。因为UDP数据报的目的地址包括两部分:目的主机IP地址和目的端口号。UDP数据报本身只包含目的端口号,由伪首部补充目的主机IP地址部分。UDP数据报发送与接收端计算校验和时均加上伪首部信息。如果接收端发现校验和正确,则在一定程度上说明UDP数据报到达了正确主机上的正确端口。UDP伪首部来自于IP报头,因此在计算UDP校验和之前,UDP首先必须从IP层获取有关信息。这说明UDP与IP之间存在一定程度的交互作用。在UDP/IP这个协议结构中,UDP校验和是保证数据正确性的唯一手段。

5.2.3 UDP校验和的计算示例

RFC 1071提供了UDP校验和计算的有效实现方法。在本节中通过引BehouzA. Forouzan所著的*TCP/IP Protocol Suite*(*Second Edition*)中一个计算校验和的例子,来说明校验和计算的具体步骤。

UDP计算校验和的计算方法和计算IP数据报首部校验和的方法相似。但不同的是IP数据报的校验和只检验IP数据报的首部,但UDP的校验和是将首部和数据部分一起都检验。

1. 校验和的计算方法

在发送端,首先是将全零放入校验和字段,再将伪首部以及UDP用户数据报看成是由许多16位的字串接起来的,作为一个整体来计算。若UDP用户数据报的数据部分不是偶数个字节,则要填入一个全零字节(但此字节不发送),然后按二进制反码计算出这些16位字的和。二进制求和的计算方法是:0+0=0,0+1=1,1+1=0,但需要产生一个进位,加到下一列。如果最高位相加后产生进位,则最后的结果加1。

2. 发送端的计算步骤

在发送端:计算校验和可以按照以下8个步骤进行:

① 将伪首部加到UDP用户数据报上;
② 将校验和字段置为0;
③ 将所有的位分为2B(16位)的字;
④ 如果字节总数不是偶数,则增加一个字节的填充(全为0);
⑤ 对所有的16位字进行二进制反码求和计算;
⑥ 将计算所得的16位的和取反码,并插入到校验和字段;
⑦ 将伪首部和其他增加的填充位删除;
⑧ 将已经有校验和的UDP用户数据报送至网络层实体进行封装。

3. 接收端的计算步骤

在接收端：校验和的验证计算一般需要按以下 5 个步骤进行：

① 将伪首部加到 UDP 用户数据报上；

② 如果需要就增加填充；

③ 将所有的位分为 2B(16 位)的字；

④ 对所有的 16 位字进行二进制反码求和计算；

⑤ 如果所得的结果为全 1，说明数据传输正确，则丢弃伪首部和所有的填充字段，并接受已经经过校验的正确的 UDP 用户数据报。如果结果不为全 1，则认为 UDP 用户数据报传输中出现错误，丢弃该数据报。

如图 5-13 所示为一个计算校验和的例子。在这个例子中，假设发送端用户数据报是长度为 7 字节的短报文“TESTING”，因为它不是 2 字节的整数倍，因此需要添加一个全为 0 的字节。下面是 UDP 校验和计算的实现方法。

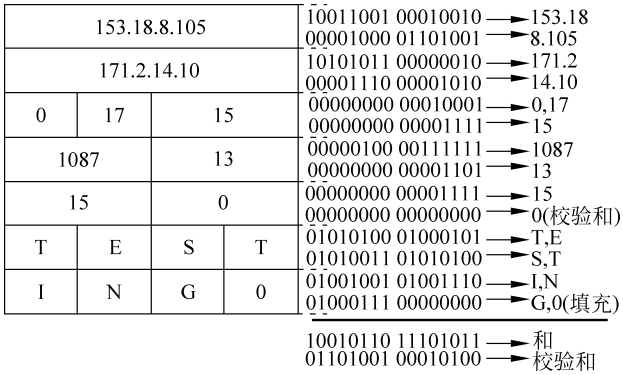

图 5-13 一个 UDP 发送端计算校验和的示例

从这个计算校验和的例子可以看出，使用伪首部是为了验证 UDP 用户数据报是否传到了正确的目的进程。

校验和是保证 UDP 用户数据报传输正确性的一种重要手段。这种简单的校验方法的检错能力不是很强，但它的好处是简单，处理起来较快。设计者的重点是考虑如何使协议简洁以及如何提高软件处理速度。

因此可以看出，这样的校验和，既可以检查 UDP 用户数据报的源端口号、目的端口号、UDP 用户数据报的数据部分，也可以检查 IP 数据报的源 IP 地址和目的 IP 地址。

5.2.4 UDP 适用的范围

使用 UDP 的典型应用有以下三种：

1. 视频播放应用

在互联网上播放视频，用户最关注的是视频流能尽快和不间断地播放，丢失个别数据报文对视频节目的播放效果不会产生重要的影响。如果采用 TCP，它可能因为重传个别丢失

的报文而加大传输延迟,反而会对视频播放造成不利的影响。因此,视频播放程序对数据交付实时性要求较高,而对数据交付可靠性要求相对较低,UDP 更为适用。

2. 简短的交互式应用

有一类应用只需要进行简单的请求与应答报文的交互,客户端发出一个简短的请求报文,服务器端回复一个简短的应答报文,在这种情况下应用程序应该选择 UDP。应用程序可以通过设置"定时器/重传机制"来处理由于 IP 数据分组丢失问题,而不需要选择有"确认/重传"的 TCP,以提高系统的工作效率。

3. 多播与广播应用

UDP 支持一对一、一对多与多对多的交互式通信,这点 TCP 是不支持的。

当然,任何事情都有两面性。简洁、快速、高效是 UDP 的优点,但是由于它不能提供必需的差错控制机制,同时在拥塞严重时缺乏必要的控制与调节机制。这些问题需要使用 UDP 的应用程序设计者在应用层设置必要的机制加以解决。UDP 是一种适用于实时语音与视频传输的传输层协议。

5.3 TCP

通过前面的介绍可以了解到,TCP/IP 协议栈中的网络层提供的是一种面向无连接的 IP 数据报服务。而传输层中的 UDP,仅仅在一些可靠性要求不高的实时传输中比较适用,因此传输层可靠性的保障还不能依靠 UDP。传输层的 TCP 旨在向 TCP/IP 的应用层提供一种端到端的、面向连接的、可靠的数据流传输服务。本节将介绍更加复杂的 TCP。通过对 TCP 的深入讨论,来了解 TCP 是如何在网络层提供的不可靠服务的基础上提供可靠的保障。尽管 UDP 和 TCP 都使用相同的网络层 IP,但是 TCP 向应用层提供的服务与 UDP 完全不同。TCP 是一种面向连接的、可靠的、全双工的传输层协议。TCP 在 IP 服务的基础上,增加了面向连接和可靠性的特点,它提供面向连接的流传输,是互联网中最常见的传输层协议。

TCP 常用于一次要传输大量报文的情形,如文件传输、电子邮件、WWW、远程登录等。

TCP 比 UDP 实现的功能丰富也复杂得多,数据传输过程要解决的问题也比 UDP 多。本节首先介绍 TCP 提供的服务,接下来讲解 TCP 报文首部格式及 TCP 连接管理。

5.3.1 TCP 提供的服务

1. TCP 是一种面向连接的协议

为了保证连接建立过程和连接释放过程的可靠性,TCP 首先是在进行实际数据传输前,必须在源主机的发送进程与目的主机的接收进程之间建立起一条传输连接,创建应用进程到应用进程之间的通信,TCP 可以使用端口号来实现这种连接。这样即使由于某种原因,传输连接建立不成功,则源进程不会像 UDP 一样向目的进程发送数据,白白地浪费网络资源。只有当连接建立完成后,才将数据流分割成为可传输的单元(TCP 报文),并逐个

发送它们。接收方 TCP 等待属于同一个进程的不同单元(TCP 报文)到达后,检查那些没有差错的单元,并将它们作为一个数据流交付给接收进程。当整个数据流发送完毕后,传输层关闭这个连接。同时,面向连接传输的每一个报文都需有接收方的确认,未被确认的报文被认为是出错报文。

使用 TCP 来建立连接和释放连接的具体方法将在 5.3.3 小节介绍。但要注意的是,TCP 连接是逻辑连接,而非物理连接。

2. 实现进程到进程的通信服务

像 UDP 一样,TCP 通过使用端口号来提供进程到进程的通信。每一条 TCP 连接只能有两个端点,只能实现点对点(一对一)的通信。但 TCP 可以同时建立多条连接。这一特点在服务器端表现得尤为突出。例如,一个 Web 服务器可以同时处理多个客户端的访问。

3. 可靠的服务

由于 TCP 协议同样也是建立在不可靠的网络层 IP 协议之上,所以 TCP 的可靠性完全是由自己实现的。TCP 为了保证数据传输的可靠、按序、无丢失和无重复,TCP 采用的是最基本的可靠性技术:差错控制、序号与确认、流量与拥塞控制、超时重发等。具体内容参见 5.5~5.7 节。

4. 面向字节流的传递服务

与 UDP 不同,TCP 是一个面向流的协议。TCP 中的"流"(Stream)是指流入到进程或从进程流出的字节序列,如图 5-14 所示。流相当于一个管道,从一端输入什么内容,从另一端可以照原样取出什么内容。它描述了一个无差错、无丢失、无重复和无乱序的数据传输过程。这个"管道"通过互联网传送这些数据。

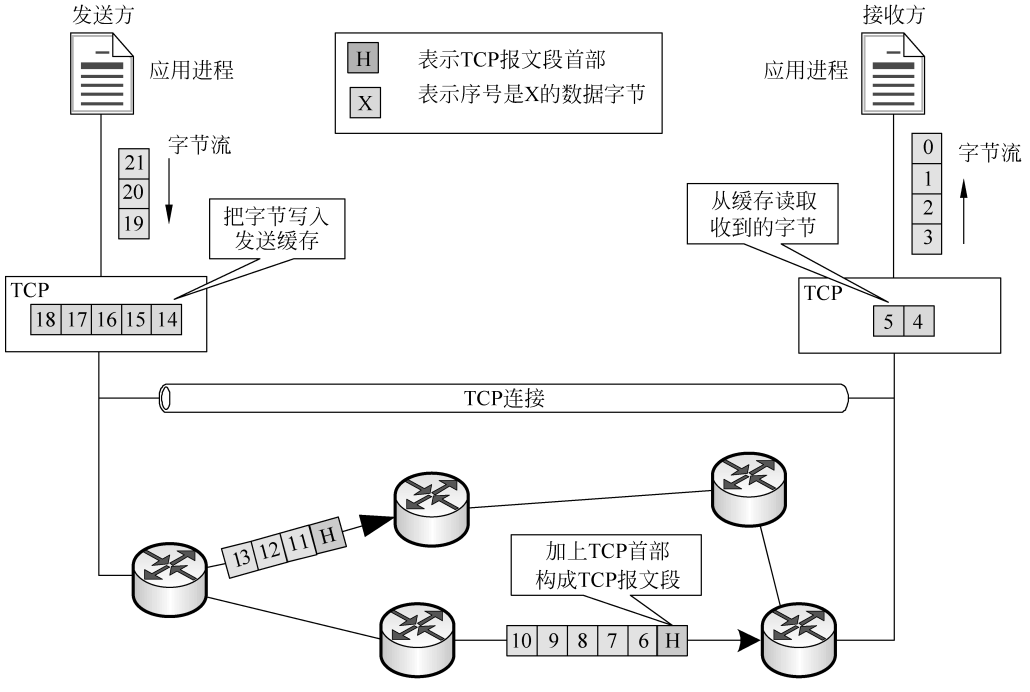

图 5-14 TCP 面向字节流的概念示意图

“面向字节流”的含义是：虽然应用程序和 TCP 的交互是一次一个数据块(大小不等)，但 TCP 把应用程序交下来的数据看成仅仅是一连串的无结构的字节流。TCP 协议提供一个流接口(Stream Interface)，TCP 允许发送进程以字节流的形式发送数据，接收进程以字节流的形式接收数据。TCP 对字节流的内容不做任何解释(其实 TCP 并不知道所传送的字节流的含义)，TCP 也不保证接收方应用进程收到的数据块和发送方应用进程所发送的数据块具有对应大小的关系(例如，发送方应用进程交给发送方的 TCP 共 8 个数据块，但接收方的 TCP 可能只用了 4 个数据块就把收到的字节流全部交付给上层的应用进程)。

需要注意，为了更方便地说明“面向字节流”的概念，即使 TCP 是一个全双工的协议，我们只画出了一个方向的数据流。另外，在实际的网络中，一个 TCP 报文段包含上千个字节是很常见的，而图中的各部分都只画出了几个字节。另一点很重要的是：图 5-14 中的 TCP 连接是一条虚连接而不是一条真正的物理连接。TCP 报文段先要传送到 IP 层，由 IP 协议封装成 IP 分组后传送到接收端。

TCP 和 UDP 在发送报文时所采用的方式完全不同。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，只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)。如果应用进程传送到 TCP 缓存的数据块太长，TCP 就可以把它划分得短一些再传送。如果应用进程一次只发来一个字节，TCP 也可以等待积累到足够多的字节后再构成报文段发送出去。

5. 支持全双工服务

TCP 支持数据在同一时间双向流动的全双工服务(Full-Duple Service)。在两个应用程序已经建立传输连接之后，客户与服务器进程可以同时发送和接收数据。TCP 连接可以从进程 A 向进程 B 发送数据，而在同一时间可以从进程 B 向进程 A 发送数据。当报文从 A 发往 B 时，也可以携带 A 对 B 发来的报文的确认。当报文从 B 向 A 发送时，同样可以携带 B 对 A 发来的报文的确认，这称为捎带确认。确认随数据一起发送，可以节省系统资源。当然，如果一方没有数据可发送，它就只能发送确认报文而不包含数据。

6. 发送和接收缓存

因为发送和接收进程可能以不同的速率写入和读出数据，因此 TCP 需要在发送方和接收方分别设置一个发送缓存和接收缓存。其中发送缓存用于存放发送方应用进程交给 TCP 而发送方 TCP 来不及发送的数据；接收缓存用于存放发送方 TCP 传递过来而接收方 TCP 尚未提交给接收方应用进程的数据。稍后我们会看到，这些缓冲区也用于 TCP 的流量控制和拥塞控制。

实现缓冲区的一种方法是使用以字节为存储单元的循环数组，如图 5-15 所示。为了简要说明缓存的概念和工作原理，我们只画了发送方的发送缓冲和接收方的接收缓存(其实发送方和接收方各有一对发送缓存和接收缓存)，每个缓存 20 个字节(实际上的缓存要比这大得多)，并给出发送缓存与接收缓存的大小是一样的(实际上可能有很大的差异)。

图 5-15 表示了在一个方向上数据的移动。在发送端，缓冲区有三种类型的单元。白色的部分是空存储单元，可以由发送端的应用进程填充；灰色的部分用于保存已经发送但还

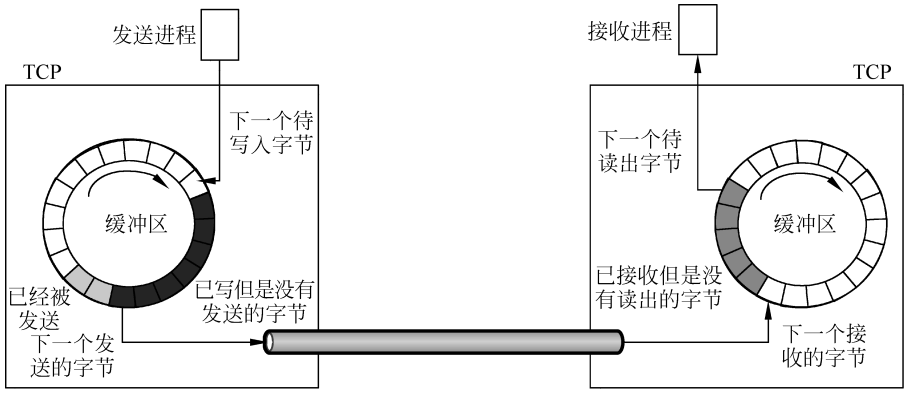

图 5-15 发送与接收缓存

没有得到确认的字节，TCP 在缓冲区中暂时保留这些字节直到收到确认为止；深灰色缓冲区是将要由 TCP 发送的字节。需要注意的是，灰色存储单元的字节一旦被确认，这些存储单元可以回收发送进程能重新使用，这也是我们给出一个环形缓冲区的原因。

接收端缓冲区的概念和操作都比较简单。环形缓冲区分成白色和灰色两个区域。白色区域是空存储单元，可以接收发送方 TCP 传输来的字节；灰色区域表示接收到的字节，可以被接收进程读取。当某个字节被接收进程读出后，这个存储单元将被回收，并加入到空存储单元池中。

7. 生成报文(或段)

发送缓存与接收缓存尽管能够处理发送端进程的发送速度和接收端进程的接收速度之间的不一致性，但在发送数据之前，还需要多个步骤。IP 层作为 TCP 服务的提供者，需要以分组的方式而不是以字节流的方式发送数据。在传输层，TCP 将多个字节组合在一起，这些组合在一起的字节称为段(Segment)或报文。TCP 给每个报文添加首部(为了达到控制目的)，并将该段向下传递给网络层。报文被封装在 IP 数据报中，然后再进行传输(IP 数据报的传输就是我们已经了解的网络层的内容)，整个操作对接收进程是透明的。所有的操作均由发送端的 TCP 进行处理，接收进程不会察觉到任何的操作。图 5-16 表示了在缓存区中如何从字节生成段。

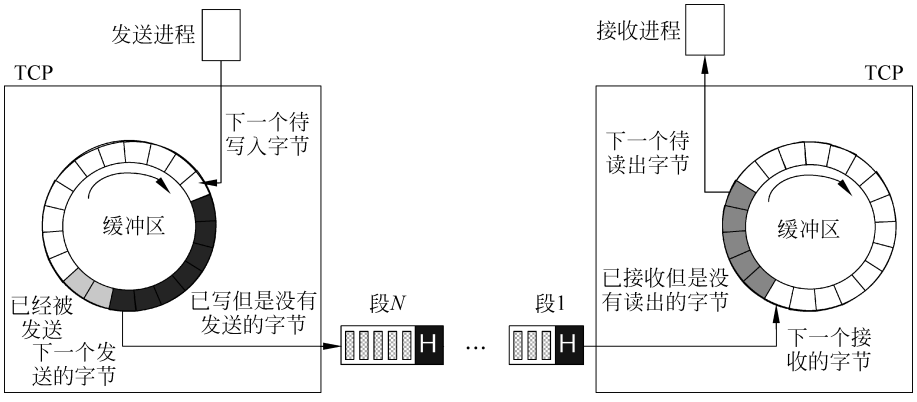

图 5-16 TCP 报文段的生成

需要说明的是,段的大小不一定相同。为了简单说明问题,我们只在图中表示了一个包含 3 个字节和另一个包含 5 个字节的段。实际上的段可能包含千百个字节。

8. 提供流量控制与拥塞控制服务

TCP 协议采用了大小可以变化的滑动窗口协议进行流量控制。在连接建立时,窗口的大小由双方商定。在通信过程中,接收端可以根据自己的资源使用情况及接收能力,随时动态地对发送窗口的大小进行调整,以保证接收端不因缓冲区溢出而丢失大量数据,导致许多报文的重传。此外,TCP 同样采用滑动窗口协议进行拥塞控制,以进一步提高可靠性。

9. 多路复用与多路分解

与 UDP 类似,TCP 在发送端执行多路复用,在接收端执行多路分用。然而,由于 TCP 是一个面向连接的协议,因此需要在每对进程之间建立连接。

通过以上讨论可以看出,TCP 协议提供的服务为:面向连接、面向字节流、支持全双工、支持并发连接、提供序号、提供确认/重传机制、流量控制及拥塞控制等。

传输层使用 TCP 来实现可靠的流传输,增加了大量的附加数据开销。所有接下来我们会看到 TCP 报文的首部比 UDP 的首部大得多。

5.3.2 TCP 报文段的格式

TCP 虽然是面向字节流的,但 TCP 传送的数据单元却是报文段。TCP 协议首部各个字段的作用体现了要实现的功能。因此只有掌握 TCP 首部各字段的功能才能掌握 TCP 的工作原理。

TCP 报文段的格式如图 5-17 所示。一个 TCP 报文段分为首部和数据两部分,数据部分为应用层报文。

TCP 报文段首部为 20～60 字节,分为固定和选项两部分。固定部分 20 字节,选项 $4N$ 字节(N 为正整数)是根据需要而增加的。因此 TCP 首部的最小长度是 20 字节。

1. 源端口和目的端口

端口号包括源端口字段和目的端口字段,各占 16 位,分别表示发送与接收该报文的应用进程的 TCP 端口号。传输层的复用和分用功能都是通过端口才能实现。

2. 序号

序号也称报文段序号,占 32 位,序号范围是[$0 \sim 2^{32}-1$](即 4 294 967 296 个序号)。虽然 TCP 软件能够记录发送或接收的段,但是在段的首部没有段序号字段。TCP 是面向数据流的,可以将 TCP 传送的报文看成连续的数据流,TCP 把在一个 TCP 连接中传送的数据流中的每一个字节都对应地编上一个序号。在 TCP 首部采用序号和确认号字段,这两个字段指的是字节序号,而不是段序号。在首部中的序号是指本报文段数据字段携带数据的第一个字节的序号。

整个数据的起始序号在连接建立时设置,每一方都使用随机生成器产生一个初始序号

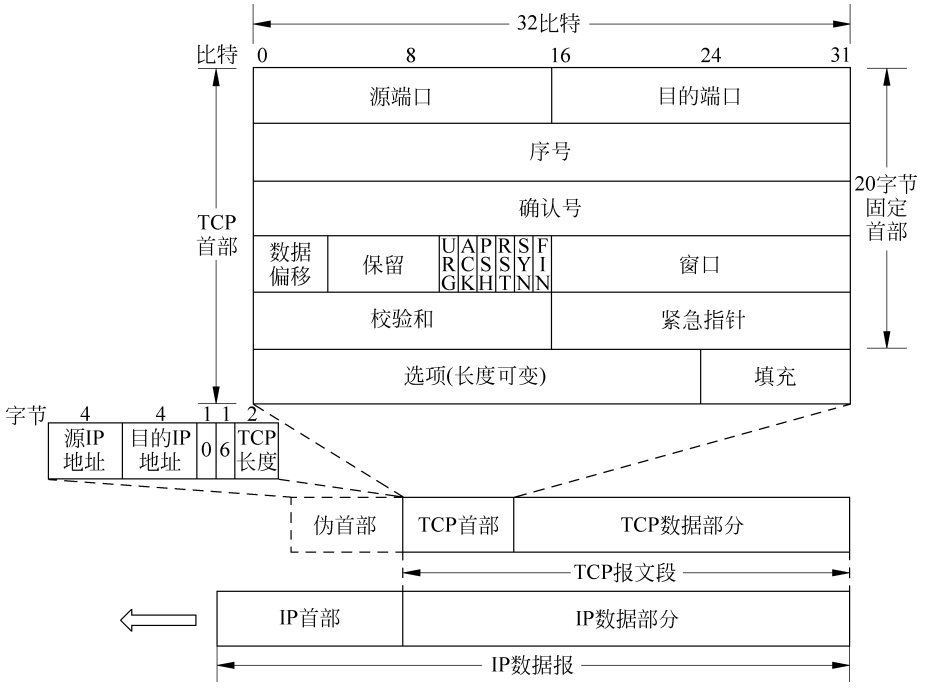

图 5-17 TCP 报文段的格式

(Initial Sequence Number,ISN),通常每一个方向的 ISN 都不同。例如:一个报文段的首部中的序号字段的值是 201,而它携带的数据长度为 200 字节,这表明,该报文段数据的最后一个字节的序号应该是 400,并且下一个报文段的首部序号字段应为 401。下面将看到序号字段主要用于差错控制、流量控制及拥塞控制。

理解序号的作用,需要注意以下问题:

(1) TCP 是面向字节流的,它要为发送字节流中的每个字节都按序编号。被编上号后,TCP 对发送的每一个报文分配一个序号。

(2) 第一个报文的序号是初始序号,这是一个随机数。由于连接双方各自随机产生初始序号,因此一个 TCP 连接的通信双方的序号是不同的。

(3) 其他报文的序号是之前报文的序号加上该报文携带的字节数。

【例 5.1】 假设一个 TCP 连接正在传送一个 6000 字节的文件,第一个字节是 10001,如果数据被平均分成 3 段,试问每个段的序号是什么?

解答:每个段的序号如下所示:

段 1 序号:10001,范围为 10001～12000

段 2 序号:12001,范围为 12001～14000

段 3 序号:14001,范围为 14001～16000

一个段的序号字段的值定义了该段包含的数据部分第一个字节的序号。

当一个报文携带数据和控制信息(捎带)时,它使用一个序号。如果一个报文没有携带用户数据,那么逻辑上不定义序号,虽然序号字段存在,但是值是无效的。然而,当有些段仅携带控制信息时也需要一个序号用于接收方的确认,这些字段用作连接建立、连接释放等

(TCP 连接管理将在本章的 5.3.3 小节讨论)。每一个这样的报文中都好像携带一个字节那样使用一个序号,但没有实际的数据。

3. 确认号

确认号占 32 位。由于 TCP 是一个全双工的协议,当连接建立时,双方同时都能发送和接收数据。确认号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,即期望收到下一个报文段首部的序号字段的值。例如,如图 5-18 所示,发送端连续发送两个报文段,长度分别是 200 字节和 300 字节,发送序号分别是 1 和 201。接着接收方发送了一个确认报文段,确认号是 501,表明序号在 1～500 的数据已正确收到。另外,确认号是可以积累的(累积确认),这意味着接收方已经正确收到了最后一个字节,然后将它加 1,并将这个结果作为确认号报告给发送方。图 5-18 中的确认报文用的就是累计确认的方式。同时表明,下次希望收到报文段首部中序号字段为 501 报文。序号字段长 32 位,可对 4GB 个报文段进行编号。如此巨大的编号空间,可保证当序号重复使用时,网络中的旧序号的报文段已经从网络中消失。

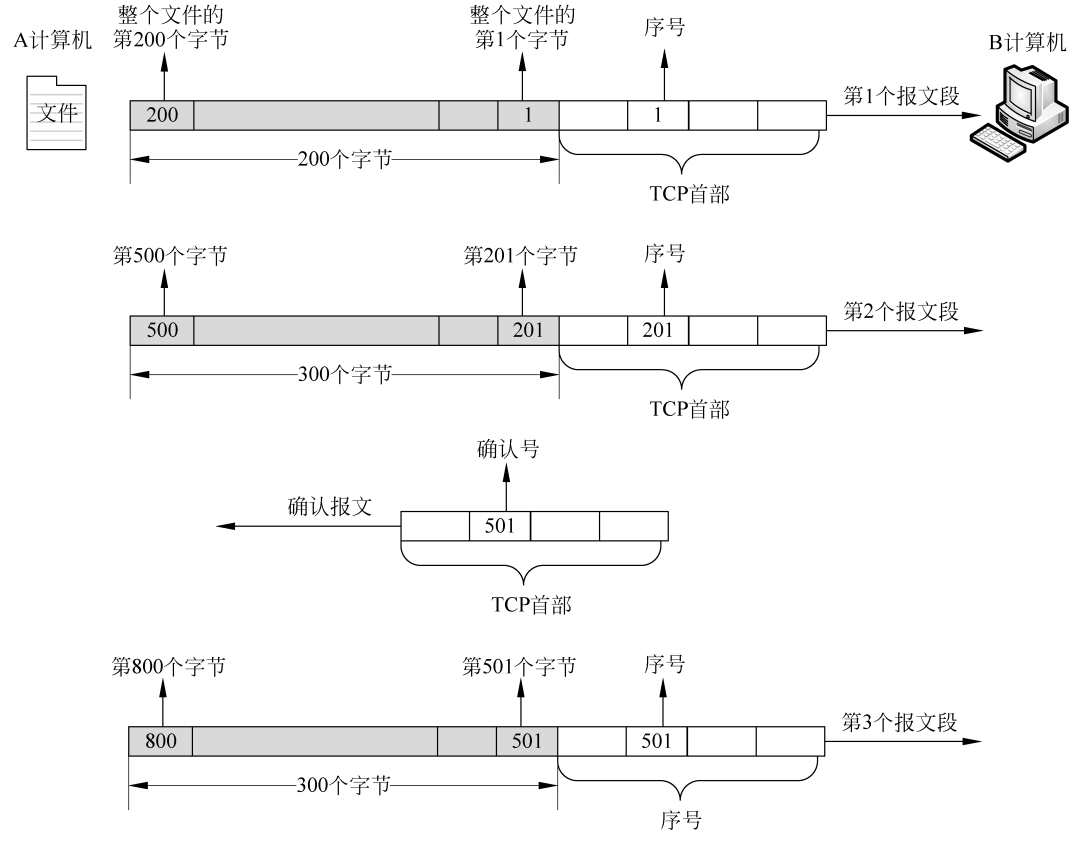

图 5-18　理解序号和确认号

报文段中的确认字段的值定义了通信一方预期接收的下一个字节的编号。确认号是可以累积的。

4. 数据偏移(首部长度)

数据偏移占4位。TCP报文固定首部的长度是20字节。设置这个字段的原因是TCP首部长度是不固定的,这个字段指出数据起始的地方离TCP报文段的起始处有多少个4字节(32位)距离,这其实就是TCP报文首部(包括选项字段)的长度。数据偏移的单位是4字节。由于4位能够表示的最大十进制数是15,因此数据偏移的值在20字节到15×4=60(B)(即TCP首部的最大长度)之间。同时,也意味着TCP首部的选项部分不能超过40字节。

5. 保留

保留占6位,留待今后使用,目前置为0。

6. 控制位

控制位占6位。共6个控制字段,分别是URG、ACK、PSH、RST、SYN和FIN 6种,每个控制字段占1位。这6个控制位说明本TCP报文段的性质,在同一时间可以设置一位或多位,这些位用在TCP的流量控制、连接建立和释放、连接失败和数据传送方式等方面。它们的意义如下:

(1) 紧急位URG(URGent):当URG=1时,紧急指针字段有效,它告诉系统本报文段中有紧急数据,应超越原来的排队顺序尽快传送(相当于高优先级的数据),而不按照原来的排队顺序传送,即便窗口为零时也可发送紧急数据。于是发送TCP就将紧急数据插入到报文段的数据的最前面,并用首部中的紧急指针(Urgent Pointer)字段指出在本报文段中紧急数据的最后一个字节的序号,以使对方知道紧急数据共有多少个字节,其余的数据都是普通数据。紧急数据到达接收端后,且接收端处理完所有的紧急数据时,TCP就告诉应用程序恢复正常操作。

(2) 确认位ACK(ACKnowledgment):当ACK=1时,确认字段有效。当ACK=0时,确认号无效。并且TCP规定,在TCP连接建立之后发送的所有报文段的ACK位都要置1。

(3) 推送位PSH(Push)(也叫紧迫位):当两个应用进程进行交互式通信时,有时一端的应用进程在键入一个命令后希望立即收到对方的响应。这时,TCP可以使用推送操作,发送端TCP将PSH置1,并立即创建一个报文段发送出去。接收端TCP收到PSH=1的报文段后,就尽快("推送")向上交给接收应用进程,而不是等到整个缓冲填满后再向上交付。一般应用在一端的应用进程输入一个命令后想立即收到对方响应的情况。

(4) 复位(或重建、重置)位RST(ReSet):当RST=1时,说明TCP连接出现严重差错(如主机崩溃等原因),必须先释放传输连接后,然后再重新建立连接。此外还可以用复位来拒绝一个非法的报文段或拒绝打开一个连接。

(5) 同步位SYN(SYNchronization):当SYN=1而ACK=0时,表示这是一个连接请求报文段。对方若同意建立连接,则在响应报文段中使SYN=1和ACK=1。也就是说,同步位置1就表示这是一个连接请求或连接响应报文段,而ACK为1或0表示确认序号是否有效。因此,SYN在连接建立时起同步序号的作用。关于连接的建立和释放,在后面TCP连接管理部分还要进行详细讨论。

(6) 终止位 FIN(FINis)：用来释放一个传输连接。当 FIN＝1 时，表示此报文段的发送端已将数据发送完毕，并要求释放传输连接。

7. 窗口

窗口占 16 位。窗口字段用来根据接收端的接收能力来控制发送端的数据发送量，单位为字节，是[0，$2^{16}-1$]的整数。即 TCP 连接的一端根据自己设置缓存空间的大小确定自己接收窗口的大小，然后通知对方从而确定对方的发送窗口的上限，目的是进行流量控制。

例如，设 TCP 连接的两端是 A 和 B，若 A 发送的报文段首部中的窗口 WIN＝100，确认序号为 100，B 在收到该报文段后就以这个窗口的数值 WIN＝100 作为 B 的发送窗口。也就是告诉 B 的 TCP，你(B)在未收到我(A)的确认时所能发送的数据量的上限就是本首部中 WIN 个字节数，即：表示 B 可以在未收到确认的情况下，向 A 发送序号是 100～199 的数据(因为 B 收到了确认序号为 100，窗口值是 100)。注意，在 B 向 A 发送的报文段中的首部也有一个窗口字段，但这是根据 B 的接收能力来确定 A 的发送窗口上限，一定不要弄混淆这两个窗口值。窗口的值是动态变化的。

8. 校验和

校验和占 16 位。校验和字段检验的范围包括首部和数据两部分。TCP 技术校验和与 UDP 计算校验和的过程相同，也需要加 12 个字节的伪首部，如图 5-17 所示，只是第 4 个字段是 IP 首部中的协议字段的值，对 TCPv4 协议为 6(TCP 的协议号是 6)，把第 5 个字段中 UDP 的长度改为 TCP 的长度。若使用 IPv6，则相应的伪首部也要改变。

上述差错检验方法的检错能力不强，但检错范围较宽，既检查了 TCP 报文段的源、目的端口号和 TCP 报文段的数据部分，又检查了 IP 数据报的源、目的 IP 地址。

但要注意的是，在 UDP 数据报中校验和是可选的。然而，对 TCP 来说，将校验和包含进去是强制的。

9. 紧急指针

紧急指针占 16 位，这个字段只有当紧急位 URG＝1 时才有效，这个字段包含了紧急数据的字节数。它定义了一个数，将此数加到序号上就得到了此段数据部分中最后一个紧急字段。因此，紧急指针指出了紧急数据的末尾在报文段中的位置(紧急数据结束后就是普通数据)。要注意的是，即使窗口为零也可以发送紧急数据。

10. 选项

选项字段的长度可变。选项字段在默认情况下是不选的，也就是说可以没有选项。但有时也是可选的，最初的 TCP 协议中只规定了最大报文段长度(Maximum Segment Size，MSS)一种选项。严格地讲，MSS 是 TCP 报文段中的数据字段的最大长度，用来告诉对方的 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"。但是这个选项并不是为了考虑接收方的接收缓存可能放不下 TCP 报文段的数据。实际上，MSS 的大小与接收方的窗口值没有关系。MSS 的作用主要是提高网络的利用率。显然，当没有选项时，TCP 报文段的首部长度为 20 字节。

MSS的选择并不简单，既不能太小也不能太大，若选择太小的MSS长度，将降低网络的利用率。而太大则在IP层传输时就有可能要分解多个较短的数据报片，在目的端要将收到的数据报片组装成原始的数据报。当传输出错时要进行重传，这样就要增大系统开销。一般地讲，只要在IP层传输时不需要分片，MSS应尽可能大些。在TCP建立连接过程中，双方将自己能够支持的MSS写入选项字段。在数据传送阶段，MSS取双方的较小值。若主机未填写这一项，则MSS的默认值为536字节。因为，所有互联网上的主机都能接受的报文长度是556字节(其中20字节为TCP报文段的固定首部长)。

随着互联网的发展，选项字段又陆续增加了几个，如窗口扩大选项、时间戳选项等(见建议标准RFC 7323)。后来又增加了有关选择确认(SACK)选项(建议标准RFC 2018)。

以上介绍了TCP首部各个字段的作用。下面通过一个具体的例子，了解TCP首部中序号的功能。

【例5.2】 主机A向主机B连续发送了两个TCP报文段，其序号分别是70和100。试问：

(1) 第一个报文段携带了多少字节的数据？

(2) 主机B收到第一个报文段后发回的确认中的确认号应当是多少？

(3) 如果主机B收到第二个报文段后发回的确认中的确认号是180，试问A发送的第二个报文段中数据有多少个字节？

(4) 如果主机A发送的第一个报文段丢失了，但第二个报文段到达了B。B在第二个报文段到达后向A发送确认。试问这个确认号是多少？

答：(1) 第一个报文段携带的字节数为：100－70＝30(B)。第一个报文段是起始序号为70，第二个报文段是起始序号为100，它们之差为第一个报文段的长度。

(2) 主机B收到第一个报文段后发回的确认中的确认号应当是100。说明B主机已经正确收到100字节前的所有数据，下次希望收到的是起始序号为100的报文段。

(3) A发送的第二个报文段中数据有180－100＝80(B)；第二个报文段是起始序号为100，第三个报文段是起始序号为180，它们之差为第二个报文段的长度。

(4) B在第二个报文段到达后向A发送确认的确认号是70。因为接收端是按序接收的，它下一个的接收序号是70，即起始序号是70的报文段。虽然所有接收的报文段是正确的，但是不按序，接收端不会接收。

5.3.3 TCP连接的建立与释放

由于TCP是面向连接的协议，连接建立的目的是更可靠地传输TCP报文。因此收发两端的传输实体之间的通信具有传输连接的建立、数据传输和连接拆除3个阶段。

在连接建立过程中，需要解决三个基本问题：

(1) 要使每一方能够确切知道对方存在。

(2) 双方协商一些通信参数，如是否使用选项(如最大报文长度)、最大窗口大小、服务质量等。

(3) 对传输实体的资源(如缓存大小、连接表中的项目等)进行分配。

TCP采用客户—服务器方式来实现TCP的连接和释放。主动发起连接建立的进程称为客户，而被动等待连接建立的进程称为服务器。服务器随系统一起启动并常驻内存，它拥

有一个熟知端口号。设主机 A 中运行客户进程,它先向 A 主机的 TCP 发出主动打开(Active Open)命令,表示要向某个 IP 地址的某个端口建立传输连接。客户根据对方的主机 IP 地址和端口号向对方服务器发起连接请求(同时带有自己的端口号和所在主机的 IP 地址)。设主机 B 中运行一个服务器进程,它先发出一个被动打开(Passive Open)命令,告诉本机的 TCP 准备接收客户进程的连接请求,然后服务器就处于"听"的状态,只要检测出客户进程发起的连接请求就立即作出响应。

TCP 连接的申请、打开和关闭必须遵守 TCP 的规定。

1. TCP 连接的建立

TCP 使用"三次握手"来建立连接,"握手"即在客户和服务器之间交换三个 TCP 报文段。并以全双工的方式传输数据,连接可以由任何一方发起,也可以由双方同时发起。如果一台主机上的 TCP 软件已经主动发起连接请求,运行在另一台主机上的 TCP 软件就被动地等待握手。当双方建立起连接后,它们就能够同时向对方发送报文了。三次握手建立 TCP 连接的简单示意图如图 5-19 所示。

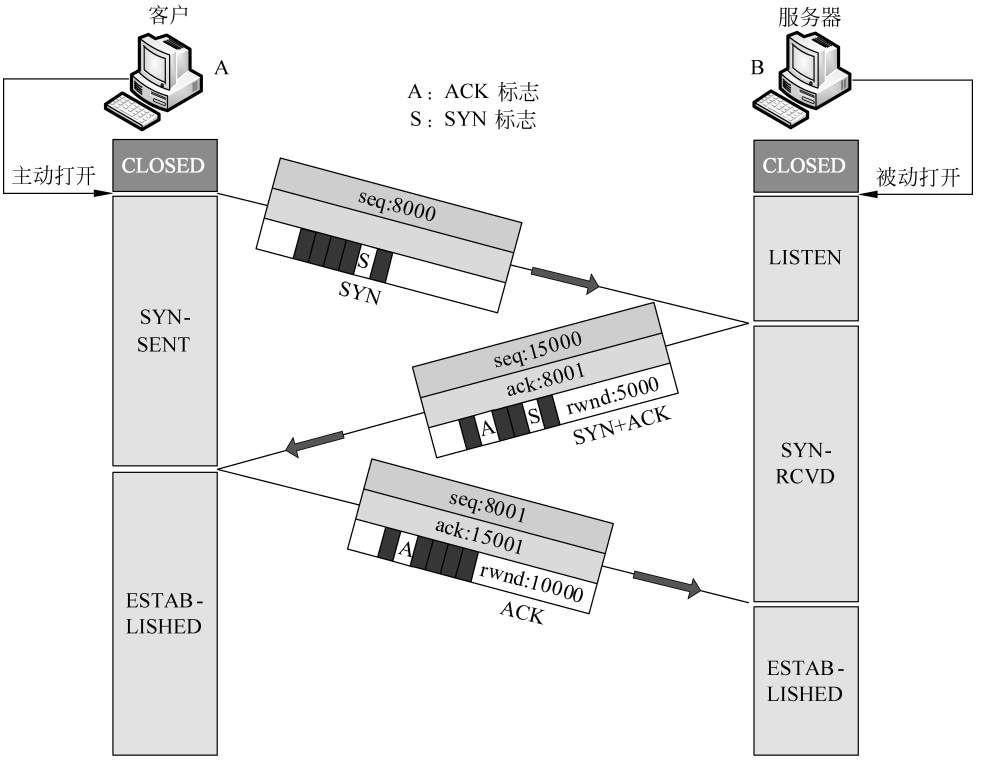

图 5-19 三次握手建立 TCP 连接示意图

当客户主机想和服务器主机通信时,服务器主机必须同意,否则 TCP 连接无法建立。为了确保 TCP 连接的成功建立,TCP 采用了三次握手的方式,使得"序号/确认号"系统能够正常工作,从而使连接双方的序号达成同步。如果三次握手成功,则连接建立成功,可以开始传送数据信息。

下面介绍采用三次握手方式建立连接的操作步骤。

为了表示连接建立的过程，共使用了两个序号，每端一个。每个报文段的首部的所有字段都有相应的值，或许选项字段也要有值。但是，为了表述方便，在此我们仅表示少数几个建立连接必须知道的字段的值。

假定主机 A 运行的是 TCP 客户进程，而 B 运行 TCP 的服务器进程。最初两端的 TCP 进程都处于 CLOSED 状态。图中在主机下面的方框分别是 TCP 进程所处的状态。请注意，在本例中 A 主动打开连接，B 被动打开连接。这个阶段的步骤如下：

(1) 客户主机的 TCP 向服务器主机发出连接请求报文段，是一个 SYN 报文。其首部中的 SYN(同步位)=1(或者说有效)，ACK=0，表示想与服务器主机进行通信，同时选择一个同步序列号(例如：seq=8000)进行同步，表明在后面传送数据时的第一个数据字节的序号是 8000+1(即 8001)。这个报文段不包含确认号，它也没有窗口大小，窗口大小的定义只有当该报文段包含确认号时才有意义。需要特别强调的是，SYN 段是一个控制段，并且不携带数据，然而它消耗一个序号，这是因为 SYN 报文段非常重要，是不允许丢失的(传错或丢失就要重传，否则无法建立连接)，所以必须进行编号。可以认为 SYN 段携带了一个虚字节的数据，让 SYN 报文段消耗一个序号。

SYN 段不携带数据，但它占用一个序号。

这时 TCP 客户进程进入 SYN-SENT(同步已发送)状态。

(2) 服务器主机的 TCP 收到连接请求报文段后，如同意，则发回确认报文段，该报文段是 SYN+ACK 报文段。在确认报文段中将 ACK(确认位)=1 和 SYN=1。这个报文段有两个目的，首先，它是另一个方向通信的 SYN 报文段，服务器主机使用这个段来初始化序号，这个序号是从服务器主机向客户主机发送数据的字节编号。同时服务器主机也通过 ACK 有效来表示这是一个对客户主机的 SYN 报文的确认报文，确认号应为 8000+1，是服务器主机预期从客户主机接收的，同时也为自己选择一个序号 15000，并定义自己的窗口值为 5000 字节。

SYN+ACK 段不携带数据，但它占用一个序号。

这时 TCP 服务器进程进入 SYN-RCVD(同步收到)状态。

(3) 客户主机的 TCP 收到服务器主机的确认后要向服务器主机发回第三个报文进行确认，即 ACK 报文段。它使用 ACK 标志和确认序号字段来确认收到了第二个报文段。在该报文段中将 ACK=1，确认号为 15000+1，而自己的序号为 8000+1。

注意：*在该报文段中，如果不携带数据，ACK 报文段不占用任何序号，但是如果一些情况下允许第三个段在连接阶段从客户端携带第一块数据时，该段消耗的序号与数据字节数相同。并定义自己的窗口值为 10000 字节。*

ACK 段，如果不携带数据，则它不占用序号。

TCP 的标准规定，SYN 置 1 的报文段要消耗掉一个序号。运行客户进程的客户主机的 TCP 通知上层应用进程，连接已经建立。A 进入 ESTABLISHED(已建立连接)状态。当客户主机向服务器主机发送第一个数据报文段时，其序号仍为 8000+1，因为上一个确认报文段并不消耗序号。

当运行服务进程的服务器主机的 TCP 收到客户主机的确认后，也进入 ESTABLISHED(已建立连接)状态。至此建立了一个全双工的连接。

这样建立连接的过程叫“三次握手”或三次联络。

为什么要进行三次握手呢？两次握手为什么不行呢？这主要是为了防止延迟的连接请求报文段突然传送到目的主机而产生错误。

如果采用两次握手，考虑这样一种情况：客户主机 A 向服务器主机 B 发出连接请求报文段，但因该连接请求报文段丢失而未收到确认，于是，客户主机超时再重传一次，这一次，客户主机收到了服务器主机的确认，建立了连接，数据传输完毕后就释放了连接。现在假设客户主机 A 发出的第一个连接请求报文段并没有丢失，而是在一些网络结点滞留的时间太长，以至于延误到第二个连接请求建立的连接释放后才到达服务器主机 B，但是服务器主机在收到这个延迟的报文后，它无法鉴别是失效的连接请求报文还是新的连接请求报文，而误认为客户主机又发出了一次新的连接请求，于是向客户主机发出确认报文段，同意建立连接。客户主机由于没有要求建立连接，当然不会理睬服务器主机的确认，但服务器主机却以为传输连接已建立，并等待客户主机发来数据，这样就白白浪费了服务器主机的资源。由此可知，采用两次握手可能造成主机资源的浪费。

而采用三次握手就可以防止上述问题的产生。因为客户主机知道这是一个不正常的连接，就不会向服务器主机发出确认，服务器主机收不到确认，连接就建立不起来。

至此，客户主机和服务器主机便可以进入数据传输阶段。

2. 传送数据

连接建立后，客户端和服务器端可以进行全双工通信，即双方都可以发送数据和进行确认，并且任何一方在发送数据的同时可以携带确认。

当位于 TCP/IP 参考模型上层的应用程序传输数据流给 TCP，TCP 接收到数据流并且把它们分解成报文段。如果数据流被分成一个个报文段，那么该数据流的每一个报文段都被分配一个序号。在服务器端，这个序列号用来把接收到的报文段重新排序成原来的数据流。

如图 5-20 所示为两台主机在成功建立连接后发送报文的过程。

在图 5-20 中，当连接建立后，客户端用两个报文段发送 2000 个字节的数据；然后，服务器端用一个报文段发送 2000 个字节，并对收到的数据进行确认；最后客户端又发送一个确认报文段。其中，前三个报文段携带数据和确认，但是最后一个报文段仅携带确认，因为已经没有数据要发送了。在这个例子中，我们要注意每个报文段的序号和确认号的变化。另外要注意客户端发送的数据段有 PSH(推送)标示，说明服务器端的 TCP 在收到数据时要立刻传递给服务器进程。另一方面，服务器端发送的报文段没有设置推送位。大多数的 TCP 的实现 PSH 都是可选标示，可以设置也可以不设置。

为了保证 TCP 工作正常、有序地进行，TCP 设置了保活(保持)计时器(Keep Timer)，用来防止 TCP 连接处于长时期空闲。如果客户端建立到服务器端的连接过程中，传输一些数据，然后停止传输，可能是这个客户端出故障了。在这种情况下，这个连接将永远处于打开状态。为了解决这种问题，一般是在服务器端设置保活计时器。当服务器端收到客户端的报文时就将保持计时器复位。如果服务器端过了设定的时间没有收到客户端的信息，它就发送探测报文。如果发送 10 个探测报文(每一个相隔 75s)还没有响应，就假设客户端出现故障，进而终止该连接。

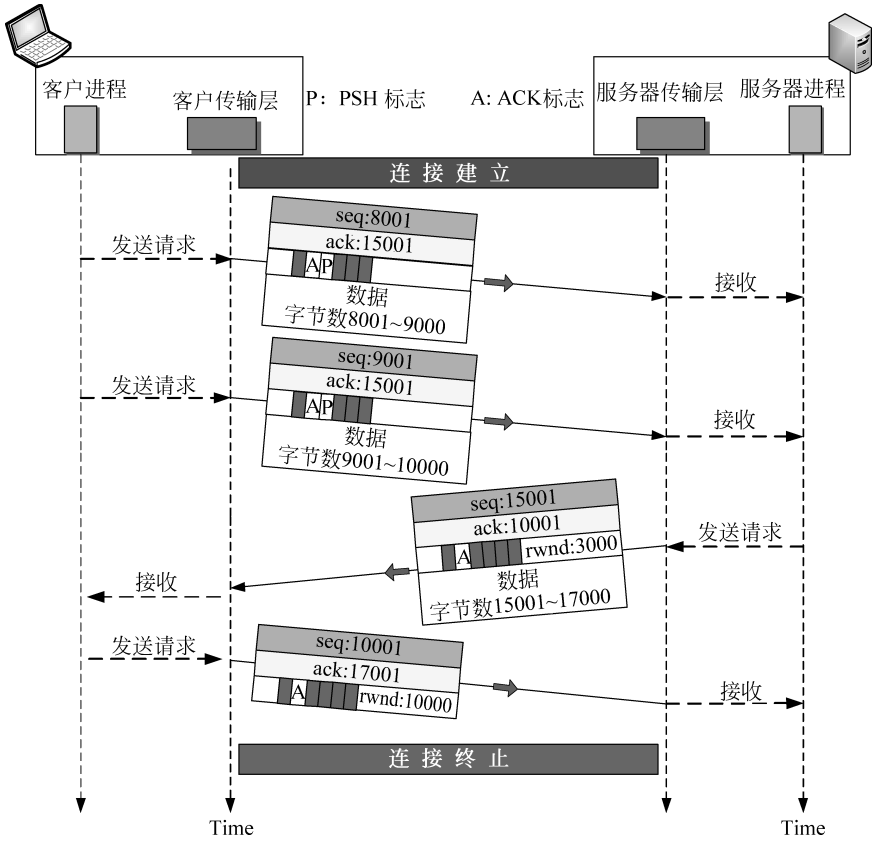

图 5-20 发送 4 个数据段的过程示意图

3. 释放连接

一个 TCP 连接建立之后，即可发送数据，一旦数据发送结束，就需要释放连接。选择 A 和 B 都处于 ESTABLISHED 状态，如图 5-21 所示。由于 TCP 连接是一个全双工的数据通道，因此参与通信的任何一方都可以关闭连接，尽管通常情况下是由客户端发起的，连接的关闭必须由通信双方共同完成。

半关闭(Half-Close)：

由于 TCP 是全双工通信，所以如果一端在停止发送数据后，另一端可以继续发送数据。这就是所谓的半关闭。

假设客户端 A 发起连接的关闭，可以使用 FIN 报文段向对方发送关闭连接请求，其序号是 seq＝x，x 等于前面 A 已传送的报文的最后一个字节的序号加 1。这时 A 进入 FIN-WAIT-1(终止等待 1)状态，等待 B 的确认。B 收到连接释放报文段后即发出确认，确认号是 ack＝x＋1，而这个确认报文段自己的序号是 y，等于 B 前面已传送的报文的最后一个字节的序号加 1，然后 B 就进入 CLOSED-WAIT(关闭等待)状态。TCP 服务器进程这时应通知高层应用进程，因而从 A 到 B 这个方向的连接就关闭了。这时的 TCP 连接处于半关闭状态，即 A 虽然不再发送数据，但对方如果有数据要传送，它仍可以在这个连接上继续接收数据。也就是说，从 B 到 A 这个方向的连接并未关闭。A 收到 B 的确认后，就进入 FIN-

WAIT-2(终止等待 2)状态。

如果 B 也没有数据向 A 发送,其应用进程就通知 TCP 释放连接。这时 B 发出 FIN=1,其序号 seq=z(在半关闭状态 B 可能又发送了一些数据)及 ack=$x+1$ 的连接释放请求报文。这时 B 就进入 LAST-ACK(最后确认)状态,等待 A 的确认。A 收到连接释放报文段后即发出确认,在该报文段中,ACK=1,确认号是 ack=$z+1$,而自己的序号为 $x+1$,然后进入到 TIME-WAIT(时间等待)状态。需要注意的是,当 TCP 关闭一个连接时,它并不认为这个连接马上就真正地关闭,还必须经过时间等待计时器(TIME-WAIT Timer)设置的时间 2MSL(Maximum Segment Lifetime,MSL),即两个最大报文寿命时间后,A 才能进入到 CLOSED 状态。这样可以保证 TCP 连接释放过程可以正常的进行(具体原因可以参考相关书籍)。

如图 5-21 所示是客户主动发起的连接释放请求的四次握手过程。

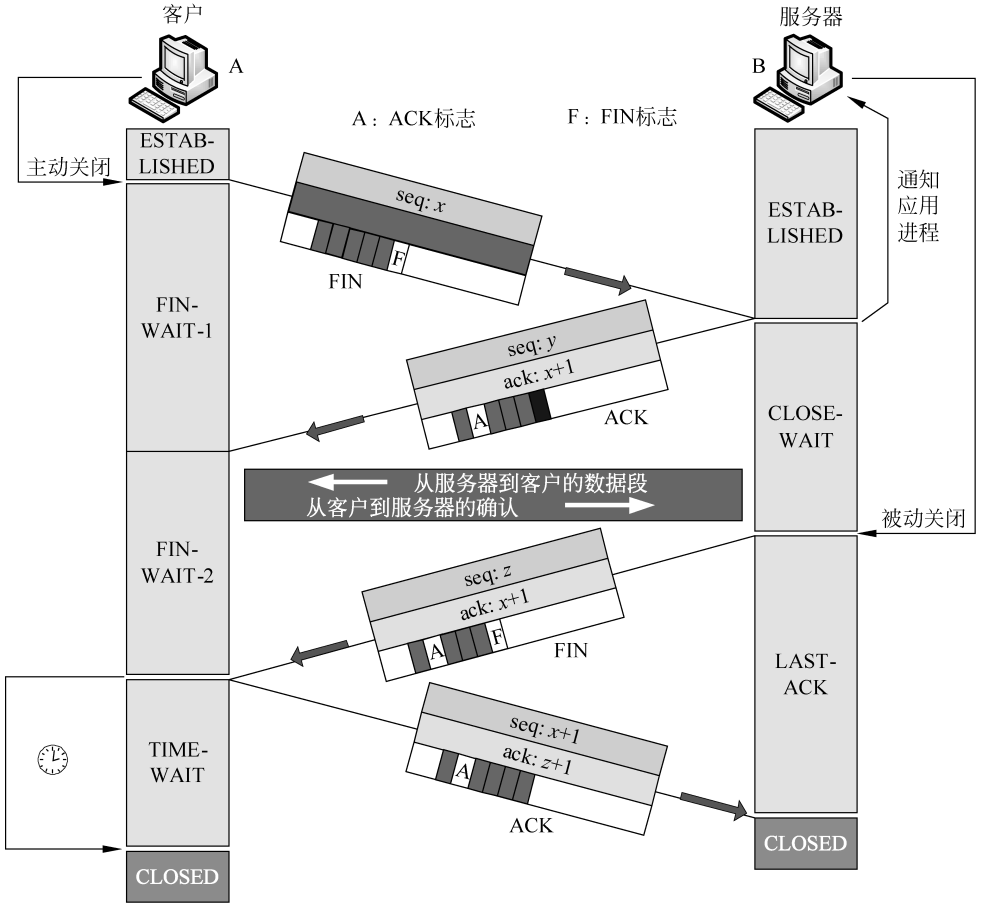

图 5-21 TCP 的连接释放请求的第四次握手过程

5.4 传输层可靠传输原理

我们知道,TCP 发送的报文段是要向下交给 IP 层(网络层)来传送的,而 IP 层只能提供尽最大努力服务,不保证可靠传输。因此,TCP 必须采取适当的措施来保证通信双方的

传输层之间的传输变得可靠。为了更好地理解传输层协议的原理，我们从一个最简单的协议入手。

需要大家注意的是，以下讨论 TCP 可靠传输的原理及实现的具体问题时，我们用到的数据单位可能是分组或报文，这只是为了讨论方便使用的数据单位，和第 1 章介绍的每一层的数据传输单元（网络层为分组，传输层为报文）没有关系。

5.4.1 简单的停等协议

停止等待（Stop and Wait）协议是最简单也是最基本的传输层协议，它虽简单却涉及许多有关可靠传输的基本概念。为了便于理解，下面从最理想情况开始讨论。

1. 理想条件下的数据传输

假设通信条件是理想的，即满足以下两方面的要求：

（1）传输完全可靠，不产生差错；

（2）不论发送方的发送速率快或慢，接收方均能及时收到数据，并上交主机。

第一个条件表明没有差错，不需要差错控制；第二个条件表明接收方的接收、处理数据的速率永远大于发送方发送数据的速率，不存在缓冲溢出而造成数据丢失的可能，因此不需要流量控制。在这种理想环境下，不需要任何的传输层协议，如图 5-22(a)所示，但这种理想环境实际是不存在的。

2. 具有简单流量控制的传输层协议

假定信道仍然是无差错的理想信道，但接收方接收数据的速率跟不上发送方发送数据的速率。这种情况下，为了保证接收方的接收缓冲区在任何情况下都不溢出，最简单的办法是，发送方每发送一组数据就停下来等待接收方的应答，接收方收到数据分组并交给主机后发送一个表示接收完毕的应答（ACK）给发方，发送方只有在得到这一应答后才发送下一组数据。这种方法的实质是使接收方控制发送方的发送速率，即进行流量控制，其过程如图 5-22(b)所示。

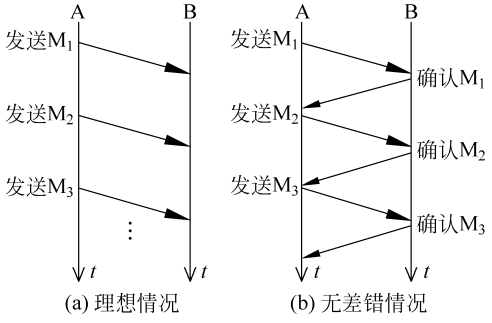

图 5-22 简单的停止等待协议

假设数据在传输中不会出错，因此严格地讲，接收方将收到的数据分组交给主机 B 后向发送方主机 A 发送的应答不需要说明所收到的数据是否正确，只要发回一个没有内容的

应答就能起到流量控制作用,这种用停止等待方式实现的传输层可靠传输的协议称为简单的停等协议。

然而实际的网络信道都不具备以上两个理想条件(甚至完全不出差错的信道也是不存在的)。但我们可以采用一些可靠的协议,在本来不可靠的传输信道上就能够实现可靠传输了。下面我们从实用的停等协议讲起。

5.4.2 实用的停等协议

实际的通信环境不是理想的,实用的停等协议既需要差错控制又需要流量控制,它是一个提供流量控制和差错控制的面向连接的协议。现在假设数据分组在传输过程中由于干扰出现了差错,那么 TCP 如何检错和纠错?

1. 检错

对于检错,TCP 计算校验和的方法和 UDP 一样。

2. 纠错

停止等待协议的基本原理是,发送方发出一个分组后必须等待应答信号,收到肯定应答信号 ACK 后继续发送下一个分组;在一定的时间间隔内没有收到应答信号必须重传该分组。

正常情况下,接收方收到一个正确的数据分组,则交付给主机 B,同时向发送方主机 A 发送一个确认 ACK,主机 A 收到 ACK 后才发送下一个新的数据分组,如图 5-23(a)所示。一旦发现数据有错,接收方首先丢弃有差错的分组,并且不会向发送方回送任何应答信息,而发送方要等收到应答后再发送下一组数据,这样势必永远等待下去,出现死锁。解决死锁问题的办法是发送方每发一组数据就启动一个超时定时器,若超过定时器所设置的定时时间 T_{out} 仍未收到接收方的应答信息,则发送方重传前面所发送的数据,然后等待。为此,在发送端必须暂时保存已发过数据的副本,直到确认发送完成或决定放弃为止。另一种差错是数据分组在传输中丢失。接收方因没有收到数据分组,当然同样不会向发送方回送任何应答信息,也将出现死锁。一般情况下数据出错和数据丢失将同样处理。这就叫作超时重传。

同理,若接收方发给发送方的应答信息丢失,也会产生死锁。解决死锁问题的办法仍然是发送方每发一个数据分组就启动一个超时定时器,若超过定时器所设置的定时时间 T_{out} 仍未收到接收方的应答,则发送方重传前面所发送的数据分组。

若丢失的是应答信息,如图 5-23(b)所示,则重发将使接收方收到两组同样的数据。由于接收方无法识别重复的数据分组,因而在接收方容易出现重复分组的差错。为解决这个问题,让每个数据分组带上发送序号 $N(S)$,每发送一组新的数据就把 $N(S)$加 1。这样,当接收方收到发送序号相同的数据分组时,就意味着上次发的确认分组发送方未收到。因此,不仅丢弃重复分组,而且再向发送方发一个确认 ACK。发送序号 $N(S)$占的位数越少,数据传输的额外开销就越小,在停等协议中它只要能区分两组相邻的数据是否重复就可以了,因此只用两个编号,用 1 位就能区分新的数据和重发的数据了。

图 5-23(c)也是一种可能出现的情况。传输过程中没有出现差错,但接收方对数据分组

M_1 的确认延迟了。发送方会收到重复的确认。对重复的确认的处理很简单：收下后就丢弃。接收方仍然会收到重复的 M_1，并且同样要丢弃重复的 M_1，并重传确认分组。

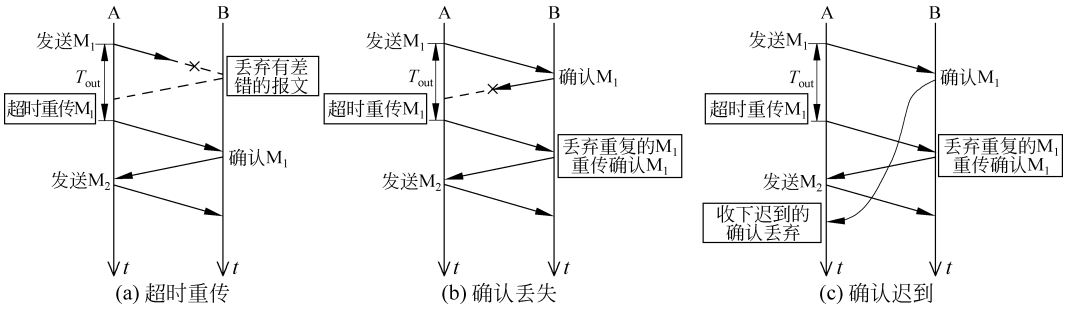

图 5-23 实用的停等协议

这里应注意以下三点：

(1) 发送端在发送完一个分组后，必须暂时保留已发送的分组的副本(为了在发生超时重传时使用)。只有在收到相应的确认后才能清除暂时保留的副本。

(2) 分组和确认分组都必须进行编号。这样才能确定哪一个发出的分组收到了确认，而哪一分组还没收到确认。同时接收方还必须记住已收到的分组的编号，直到下一个分组来到并进行比较，若相同则丢弃该分组。

(3) 超时计时器设置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。如图 5-23(a)中的一段虚线表示如果 M_1 正确到达接收方同时发送方也正确收到确认的过程。超时时间 T_{out} 值的选择应恰当，若选得太长，浪费时间；若选得太短，则发送方有可能在重发了数据后，才收到对方为前一个接收数据所回送的应答信息，这样就会造成混乱。一般将定时时间选为略大于“从发送完数据到收到应答信息所需的平均时间”。可见重传时间设定为比平均往返时间更长一些。其实关于重传时间的选择是一个复杂的问题，在 5.5 节中还要具体介绍。

一般情况下，发送方最终会收到对所有发出的数据的确认。如果发送方不断重传数据但总是收不到确认，就说明通信的线路太差，不能进行通信。

使用上述的序号、确认和超时重传机制，就可以在不可靠的传输线路上实现可靠的通信。

像上述的这种可靠的传输协议常称为自动重传请求(Automatic Repeat reQuest，ARQ)。意思是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丢失或出错的数据分组。

停止—等待协议的特点是简单，但传输效率低。为了提高传输效率，发送方可以采用流水线传输(见图 5-24)。流水线传输就是发送端可连续发送多个分组，不必每发完一个分组就停下来等待对方的确认。这样可以使信道上一直有数据不间断的传送。显然这种传输方式可以获得更高的信道利用率。

在使用流水线方式传输时，就要使用下面将要介绍的连续 ARQ 协议和滑动窗口协议。由于滑动窗口协议比较复杂，是 TCP 的精髓所在。因此我们在介绍连续 ARQ 协议时，先介绍其最基本的原理，但不涉及具体实现的许多细节问题。在本章的后续部分我们将详细

讨论实现可靠传输的滑动窗口协议。

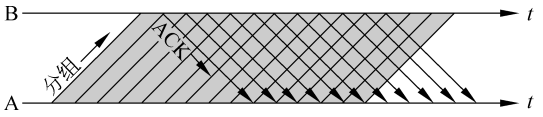

图 5-24 流水线传输可获得更高的利用率

5.4.3 连续 ARQ 协议

在发送端 A 设置一个发送窗口,窗口大小的单位是字节,如果发送窗口为 400 个字节,一个分组有 100 个字节,在发送窗口中就有 M_1、M_2、M_3 和 M_4 四个分组,发送方连续发送这四个分组,发送完毕后,就停止发送,接收端 B 收到这四个连续分组,只给 A 发送一个 M_4 确认即可。发送端 A 收到对 M_4 分组的确认,发送窗口就向前滑动,M_5、M_6、M_7、M_8 分组就进入发送窗口,这四个分组就可以连续发送,发送完后,停止发送,等待确认。具体如图 5-25 所示。

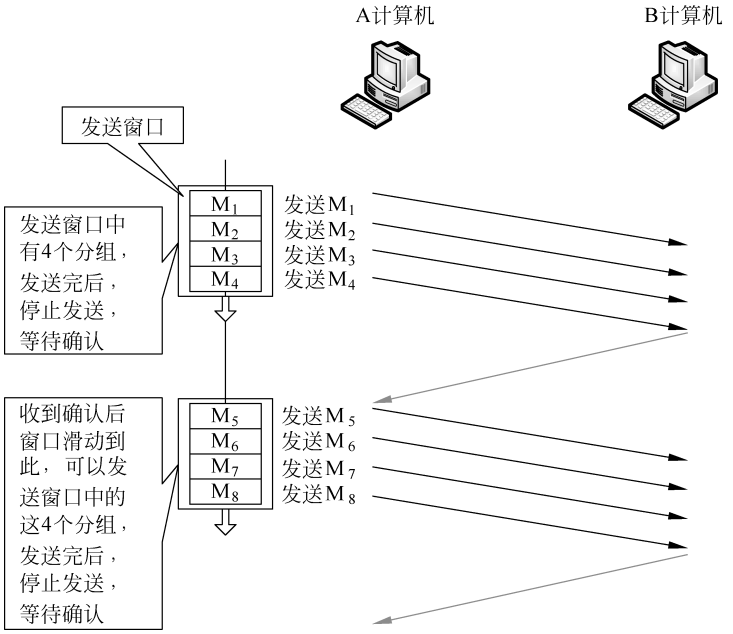

图 5-25 连续 ARQ 协议

由于将窗口的尺寸开到足够大时,分组在线路上可以连续地流动,因此称其为连续 ARQ 协议。根据对出错数据和丢失数据处理上的不同,连续 ARQ 协议分为后退 *N* 帧 ARQ(Go-Back-N,GBN)协议和选择性重传 ARQ(Selective-Repeat,SR)协议。这些协议都是滑动窗口技术和自动请求重发技术的结合。

1. 后退 *N* 帧 ARQ 协议(拉回方式)

后退 *N* 帧 ARQ 协议的基本原理是:当接收方检测出失序的分组后,不管之后接收的

分组是否准确，要求发送方重发最后一个正确接收的分组之后的所有未被确认的分组；或者说当发送方发送了 n 个分组后，若发现该 n 个分组的前一个分组在计时器超时后仍未返回其确认信息，则该分组被判定为出错或丢失，此时发送方就不得不重新发送该出错分组（或丢失分组）及其后的 n 个分组。显然，后退 N 帧 ARQ 协议的效率不高。

连续 ARQ 协议规定：发送方每收到个确认，就把发送窗口向前滑动一个位置。而接收窗口的大小总是 1。接收方总是按顺序寻找特定的分组是否到达。任何失序的分组到达都会被丢弃并要求发送方重发。

例如：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，只好把后面的三个分组都再重传一次。这就是 Go-back-N（回退 N）叫法的由来，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，后退 N 帧 ARQ 协议会带来负面的影响。有时效率甚至不如停止—等待协议。

2. 选择性重传 ARQ 协议

在连续 ARQ 协议中，接收方的窗口大小总是 1。如果接收方的窗口也可以开到发送窗口那么大，则允许不按顺序接收，这样可以只是选择性地重发出错或丢失的分组。因此得到一种更有效的协议——选择性重传 ARQ 协议。

选择性重传 ARQ 协议的基本原理是：当接收方发现某分组出错后，其后续送来的正确分组虽然不能立即提交给接收方的高层，但接收方仍可收下来存放在接收方的缓冲区中，同时要求发送方重新传送出错的那一分组。一旦收到重新传来的分组后，就可与原已存于缓冲区中的其余分组一起按正确的顺序递交高层。

这样 TCP 的发送方只发送丢失（或出错）的数据而不用发送后续全部数据，提高了数据的传输效率。

5.4.4 三种协议的比较

一般来说，凡是在一定范围内到达的分组，即使它们不按顺序，接收方也要接下来。若把这个范围看成是接收窗口的话，则接收窗口的大小应该是大于 1 的。而连续 ARQ 协议正是接收窗口等于 1 的一个特例，选择性 ARQ 也可以看作是一种滑动窗口协议，只不过其发送、接收窗口均大于 1。若从滑动窗口的观点来统一看待停等 ARQ、连续 ARQ 和选择性 ARQ 三种协议，它们的区别仅在于各自窗口尺寸的大小不同而已，如表 5-3 所示。

表 5-3 三种协议的区别

协议	发送窗口	接收窗口
停等 ARQ	1	1
后退 N 帧 ARQ	>1	1
选择性 ARQ	>1	>1

注意，选择性重传 ARQ 协议的发送窗口和接收窗口的大小相等。

5.5 传输层可靠传输的实现

上一节我们了解了传输层可靠传输的原理,接下来我们介绍传输层可靠传输的实现方法。TCP 协议通过滑动窗口机制来跟踪和记录发送字节的状态,实现差错控制功能。为保障传输层的可靠传输,需要滑动窗口技术和自动请求重发技术的结合,还包括必要的流量控制和拥塞控制。在介绍差错控制、流量控制及拥塞控制之前,我们先介绍 TCP 中使用的以字节为单位的滑动窗口技术。

5.5.1 滑动窗口协议原理

通过之前的介绍,我们对窗口的概念有了一定的了解,下面我们深入讨论滑动窗口技术的原理。

TCP 是全双工协议,因此在每个方向的数据传送上使用两个窗口(发送窗口和接收窗口),这意味着通信的双方共有四个窗口。为了使讨论简单,我们假设通信只是单方向的(这与实际不符),即数据传输只在一个方向进行,这样做的好处是使讨论仅限于两个窗口,发送方 A 的发送窗口和接收方 B 的接收窗口。如果再考虑 B 也向 A 发送数据,那么就要增加 A 的接收窗口和 B 的发送窗口,这对了解窗口的概念没什么帮助,反而加大了复杂性。

滑动窗口是面向字节流的,为了方便大家记住每个分组的序号,下面就假设每一个分组是 100 个字节。为了方便画图和讨论,将分组进行编号简化表示,如图 5-26 所示,大家需要记住,每一个分组的序号是多少。

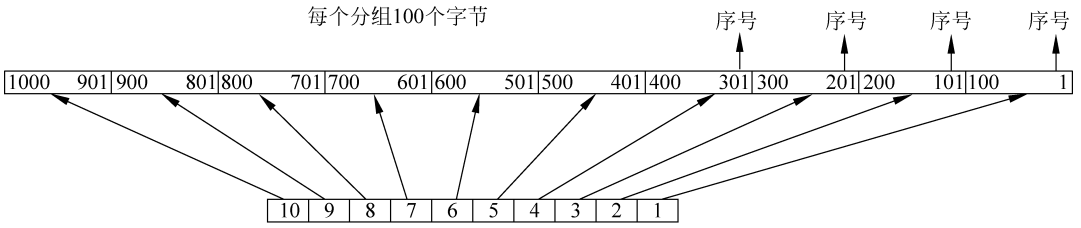

图 5-26 简化分组表示

下面就以 A 计算机给 B 计算机发送一个文件为例,详细讲解 TCP 面向字节流的可靠传输实现过程,整个过程如图 5-27 所示。

(1) A 计算机和 B 计算机通信之前先建立 TCP 连接,B 计算机的接收窗口为 400 字节,在建立 TCP 连接时,B 计算机告诉 A 计算机自己的接收窗口为 400 字节,A 计算机为了匹配 B 计算机的接收速度,将发送窗口设置为 400 字节。

(2) 在 t_1 时刻,A 计算机发送应用程序将要传输的数据以字节流形式写入发送缓存,发送窗口为 400 字节,每个分组为 100 字节,第 1～第 4 个分组在发送窗口内,这四个分组按顺序发送给 B 计算机。在发送窗口中的这四个分组,没有收到 B 的确认,就不能从发送窗口中删除,因为丢失或出现错误还需要重传。

(3) 在 t_2 时刻,B 计算机将收到的四个分组放入缓存中的接收窗口,按 TCP 首部的序号排列分组,窗口中的分组编号连续,接收窗口向前移动。接收窗口就留出空余空间。接收

应用程序按顺序读取接收窗口外连续的字节。

(4) B计算机向A计算机发送一个确认,图中ACK=1,代表TCP首部ACK位标记位有效,ack=401,代表确认号是401。

(5) t_3 时刻,A计算机收到B计算机的确认,确认号是401,发送窗口向前移动,401后面的字节就进入发送窗口。进入发送窗口的第5～第8四个分组按顺序发出。从发送窗口移出的第1～第4四个分组已经确认发送成功,就可以从发送缓存中删除,发送方可以使用回收的空间存放后续字节(大家应该记得之前介绍发送缓存和接收缓存使用的循环数组,因此回收的空间可以循环使用)。

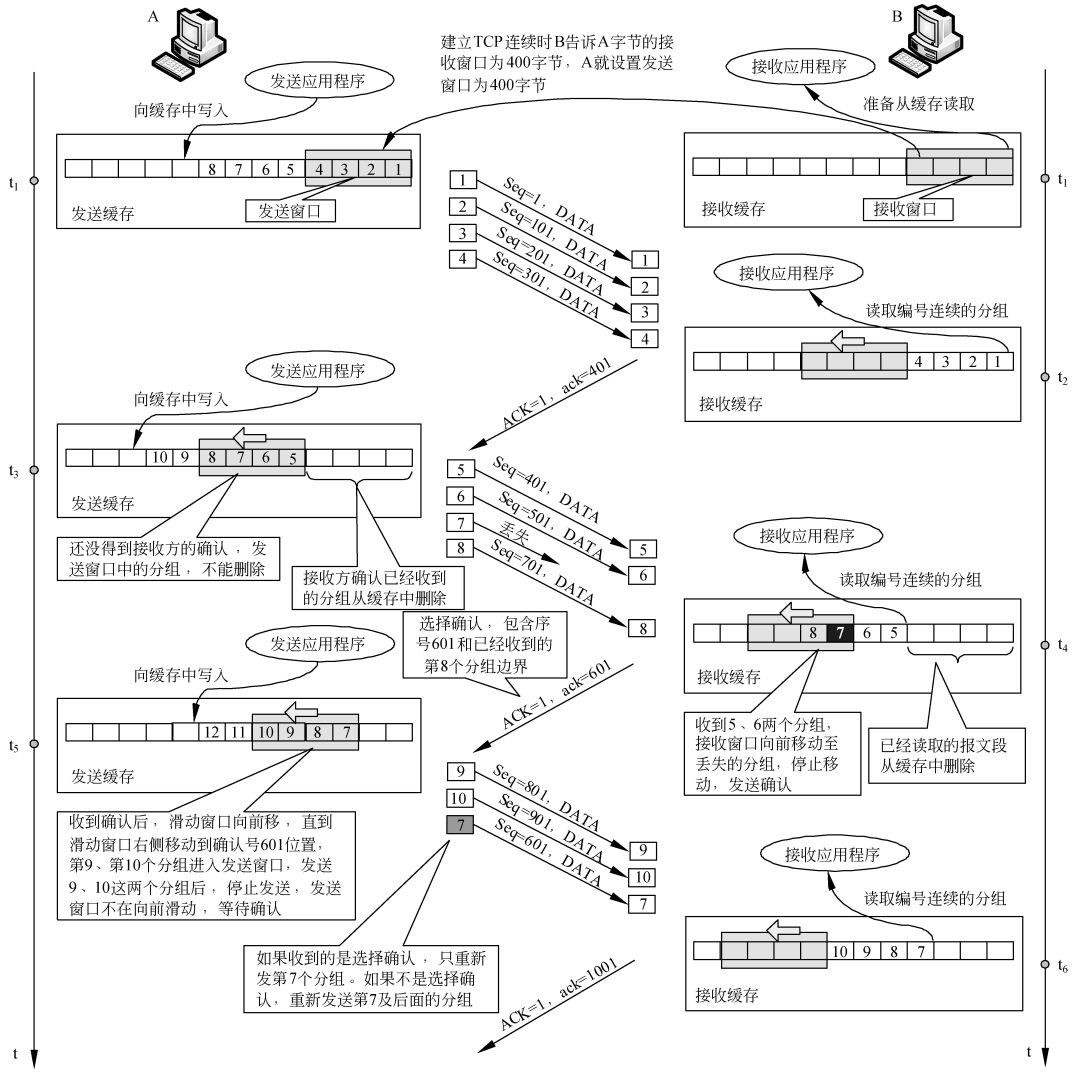

图 5-27 滑动窗口技术示意图

(6) 第5～第8四个分组在发送过程中,第7个分组丢失或出现错误。

(7) 在 t_4 时刻,B计算机收到了第5、第6、第8三个分组,接收窗口只能向前移200个字节,等待第7个分组,第5、第6个分组移出接收窗口,接收应用程序就可以读取编号连续

的字节并且可以从接收缓存中删除,回收的空间可以被循环使用。

(8) B计算机向A计算机发送一个确认,确认号是601,告诉A计算机已经成功接收到600以前的字节,可以从601个字节开始发送。

(9) A计算机收到确认后,发送窗口向前移动200个字节,这样,第9、第10个分组进入发送窗口,按顺序发送这两个分组,发送窗口中的分组全部发送,停止发送,等待确认。第7个分组超时后,重传第7个分组。

(10) B计算机收到第7个分组后,接收窗口的分组序号就能连续,接收窗口前移,同时给A计算机发送确认,序号为1001。

(11) A计算机收到确认后,发送窗口向前移,按序发送窗口中的分组。以此类推,直至完成数据发送。

TCP在建立连接时,客户端就和服务器协商了是否支持选择确认(SACK),如果都支持选确认,第7个分组丢失(或出错),发送端只重传第7个分组。否则,发送端要重传第7及以后的分组(后退 N 帧ARQ)。

5.5.2 TCP的差错控制机制

我们在本章之前一直在说TCP是一个可靠的传输层协议。这意味着TCP协议可以保证无差错、按序、不丢失、不重复地完成数据流的传输,那么它是如何保证可靠传输的呢?本节就进行具体的分析。

TCP使用差错控制提供可靠性。其使用了校验和、序号、确认与重传等机制,这些机制是TCP可靠服务的重要措施。

差错控制包括用于检测报文并且重发差错报文及重发丢失报文的机制、用于存储失序的报文直到失序报文到达的机制,以及检测并丢弃重复报文的机制。

以下是TCP为保证可靠传输所采取的一些有效手段。

1. 字节编号机制(序号)

TCP是面向字节流的,它要为发送字节流中的每个字节都按顺序编号,接收方也按序接收,这样保证报文在传输过程中不乱序、不丢失、不重复。

2. 校验和

每个报文都包括校验和字段,用来检测一个报文是否在传输过程中出错。如果报文出错,它将被目的端的TCP丢弃,并被认为丢失了(这样接下来就与报文丢失进行同样的处理,减少了TCP的复杂性)。通过之前对TCP报文格式的介绍,我们了解到每一个TCP报文都有一个16位的校验和来检测该报文段是否出错。这样保证报文在传输过程中不会出错。

3. 数据正确接收的确认机制

当接收端的TCP收到发自TCP连接的另一端的数据时,它将发送一个确认来证实收到了发送方发送的正确数据。不携带数据但占用序号的一些控制段也要确认,但ACK报文不需要确认。下面我们讨论确认的几种类型。

(1) 累积确认(ACK)：最初的 TCP 被设计成累积确认接收报文。这样可以减少传输开销。TCP 报文首部的 32 位的确认号字段用来累积确认，只有当控制位 ACK 字段为 1 时才有效。接收方通过确认号告诉发送方下一个预期接收的报文的起始序号，忽略所有失序的报文。关于累积确认的概念之前已有介绍，在此不再赘述。

(2) 捎带确认：TCP 能提供全双工通信，因此通信中的每一方都不必专门发送确认报文段，而是在传送数据时顺便把确认信息捎带传送，这种方式叫捎带确认。这样做能提高传输效率。

确认报文一般不立即发送的，通常将推迟几分之一秒。但要注意两点：一是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重传，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；二是捎带确认实际上并不经常发生，因为大多数应用进程很少同时在两个方向上同时发送。

(3) 选择性确认(Selective Acknowledgment，SACK)：现在越来越多地实现加入了另外一种称为选择性确认的确认类型。SACK 并不代替 ACK，但它向发送方报告额外的信息。SACK 报告失序的报文，也报告重复的报文。然而，由于 TCP 首部没有对应的字段完成相应的功能，SACK 以一种 TCP 报文的选项的形式来实现。

4. 计时器

当发送方 TCP 发出一个报文段时，先在重传队列中放入一个副本，并启动一个计时器，接收方在收到报文段后进行检测，若无差错则发确认，当发送方收到确认后再删除重传队列中的副本。若在计时器计数结束时还没有收到确认，则重传此报文段。TCP 协议采用自适应的超时及重传策略。这样保证发送报文或确认报文如果在传输过程中丢失不能造成死锁。

5. 超时重传机制

差错控制的核心是报文的超时重传。超时重传有以下功能：

(1) 保障不丢失的重传：在 TCP 的数据传输过程中，当发送方的 TCP 发送一个报文段后，将同时在自己的重传队列中存放一个副本。若发送方在规定的设置时间内收到接收方的确认，就删除此副本。若计时器时间到之前没有收到确认，就要将未被确认的报文段重新发送，即重传此报文段的副本。接收方发出的确认，是为了表明在接收端的 TCP 已经正确地收到了发送方所发送的报文段。

(2) 保障无差错的重传：若接收方收到的是有差错的报文段，则丢弃此报文段但不发送否认信息，等待发送方超时重传。若接收方收到的是重复的报文段(根据序号判断)，也要将其丢弃，但要发回(或捎带发回)确认信息。

(3) 保障按序的方法：由于 TCP 报文段要封装到 IP 数据报中来传输，而 IP 数据报的到达可能会乱序，因此，TCP 报文到达时也可能出乱序的现象。若收到的报文段无差错，只是序号未按先后次序，TCP 对此未做明确规定，由 TCP 的实现者自行确定，或者将不按序号的报文段丢弃，或者将其暂存于接收缓存内，待所缺序号的报文段到齐后再一起上交给应用层。当今的 TCP 实现是不丢弃失序报文，它们暂时存储这些失序报文直到缺失的报文到达。TCP 需要对接收到的报文进行重新排序，然后将按序的报文传送给应用层进程。上述

功能的实现依靠 TCP 报文中的序号机制。

注意,失序报文可以失序到达,并被接收的 TCP 暂时存储。但 TCP 要确保传递给应用进程的报文是按序的。

重传机制是 TCP 中最重要和最复杂的问题之一。其重要性通过以上的介绍我们已经了解,而复杂性的原因在于重传时间的选择。下面我们重点讨论这个问题。

6. 重传(超时)计时器重传时间计算

传输层超时计时器重传时间的设置比较复杂。因为 TCP 的下层是一个互联网络环境,发送的报文段可能只经过一个高速的局域网,也可能经过多个低速率的广域网,还可能既经过高速的局域网,也经过低速的广域网,并且数据报所选择的路由还可能发生变化,使传输层传送一个报文段的往返时间相差很大。若将超时时间设置小了,就会引起许多不必要的重传,给网络增加许多不必要的开销;但若将超时时间设置过长,则显然会使网络的传输速率降低很多。

因此,对重传时间的设置,TCP 采用了一种自适应算法。该方法记录每一个报文段发出的时间和收到相应的确认报文段的时间,报文段的往返时间 RTT 等于这两个时间之差。然后,将各个报文段的往返时间加权平均,便可得到报文段的平均往返时间 SRTT(S 表示 Smoothed)。每测量一个新的往返时间样本就按式(5-1)重新计算一次平均往返时间:

$$\text{平均往返时间 SRTT}=(1-\alpha)(\text{旧的 RTT})+\alpha(\text{新的 RTT 样本}) \tag{5-1}$$

式中 $0 \leqslant \alpha < 1$。若 α 很接近于零,则表示新算出的平均往返时间 RTT 和原来的值相比变化不大,RTT 值更新较慢。若选择 α 接近于 1,则 RTT 值更新较快。α 的典型值为 1/8[RFC 6298]。

【例 5.3】 已知:当前 TCP 连接的 RTT 值为 35ms,连续收到 3 个确认报文段,它们比相应的数据报文段的发送时间滞后了 27ms、30ms 与 21ms。设 $\alpha=0.2$。计算第三个确认报文段到达后的新的 RTT 估计值。

解:新的估计值:SRTT=(1−α)×(旧的 RTT)+α×(新的 RTT 样本)

根据式(5-1):$\text{SRTT}_1=(1-0.2)\times 35+0.2\times 27=33.4(\text{ms})$

$\text{SRTT}_2=(1-0.2)\times 33.4+0.2\times 30\approx 32.7(\text{ms})$

$\text{SRTT}_3=(1-0.2)\times 32.7+0.2\times 21\approx 30.4(\text{ms})$

所以当第三个确认报文到达后,新的 RTT 估计值是 30.4ms。

用这种方法得出的加权平均往返时间 SRTT 就比测量出的 RTT 值更加平滑。

显然,超时计时器设置的超时重传时间 RTO(Retransmission Time-Out)应略大于上面得出的加权平均往返时间 SRTT。RFC 6298 建议使用式(5-2)计算 RTO:

$$\text{RTO}=\text{SRTT}+4\times \text{RTT}_\text{D} \tag{5-2}$$

而 RTT_D 是 RTT 的偏差的加权平均值,它与 SRTT 和新的 RTT 样本之差有关。RFC 6298 建议:

① 当第一次测量时,取 RTT_D1 值为测量到的 RTT 样本值的一半。

② 在以后的测量中,则使用式(5-3)计算加权平均的 RTT_D:

$$\text{新的 RTT}_\text{D}=(1-\beta)\times(\text{旧的 RTT}_\text{D})+\beta\times|\text{SRTT}-\text{新的 RTT 样本}| \tag{5-3}$$

这里 β 是个小于 1 的系数,它的推荐值是 1/4,即 0.25。

上面所说的往返时间的测量，实现起来相当复杂。下面举例说明超时重传时间 RTO 的计算方法。

【例 5.4】 假定 TCP 在开始建立连接时，发送方设定超时重传时间 RTO＝6s。

(1) 当发送方收到对方的连接确认报文段时，测量出 RTT 样本值为 1.5s。试计算现在的 RTO 值。

(2) 当发送方发送数据报文段并收到确认时，测量出 RTT 样本值为 2.5s。试计算现在的 RTO 值。

解：RTO 值计算如下：

(1) 当第一次测量到 RTT 样本时，SRTT 值就取为这个测量到的 RTT 样本值。即：SRTT＝1.5s。

根据 RFC 6298 的建议，当第一次测量时，RTT_D 值取为测量到的 RTT 样本值的一半。

因此，$RTT_D=(1/2)\times1.5=0.75$s。

根据式(5-2)，$RTO=SRTT+4\times RTT_D=1.5+4\times0.75=4.5$(s)

(2) 新的 RTT 样本＝2.5s

根据式(5-1)，新的 SRTT ＝(1－α)×(旧的 SRTT)＋α×(新的 RTT 样本)

＝(1－1/8)×1.5＋1/8×2.5

＝1.625(s)

按式(5-3)，新的 RTT_D ＝(1－β)×(旧的 RTT_D)＋β×|SRTT-新的 RTT 样本|

＝(1－1/4)×0.75＋1/4×|1.625－2.5|

＝0.78125≈0.78(s)

按式(5-2)，$RTO=SRTT+4\times RTT_D=1.625+4\times0.78\approx4.75$(s)

这里有一个重要的问题要考虑，那就是重传报文段如何计算 RTT 值。假设发送端发送出一个报文段，设定的重传时间到了，还没有收到确认，于是重传该报文段。经过了一段时间后，发送端收到了确认报文段。由于重传的报文段和原来的报文段完全一样，那么发送端如何判定此确认报文段是对先发送的报文段的确认，还是对后来重传的报文段的确认？而正确的判断对确定加权平均 SRTT 的值关系很大。

若收到的确认是对重传报文段的确认，但被源主机当成是对原来的报文段的确认，则这样计算出的 SRTT 和超时重传时间 RTO 就会偏大。若后面再发送的报文段又是经过重传后才收到确认报文段，则按此方法得出的超时重传时间 RTO 就越来越长。

反之，若收到的确认是对原来的报文段的确认，但被当成是对重传报文段的确认，则由此计算出的 SRTT 和 RTO 都会偏小。这就必然导致报文段过多地重传，这样就有可能使 RTO 越来越短。

为此，Karn (一位无线电爱好者)提出了一个算法：在计算加权平均 SRTT 时，只要报文段重传了，就不采用其往返时间作为计算 SRTT 和 RTO 的样本。这样得出的加权平均 SRTT 和 RTO 就较准确。

但是，这又引起新的问题。设想出现这样的情况：报文段的时延突然增大了很多。因此在原来得出的重传时间内不会收到确认报文段，于是就重传报文段。但根据 Karn 算法，不考虑重传的报文段的往返时间样本。这样，超时重传时间就无法更新。

因此要对 Karn 算法进行修正。方法是：报文段每重传一次，就把超时重传时间 RTO

增大一些。典型的做法是取新的重传时间为旧的重传时间的2倍。当不再发生报文段的重传时,才根据上面给出的式(5-2)计算超时重传时间。实践证明,这种策略较为合理。

总之,Karn算法能够使传输层区分开有效的和无效的往返时间样本,从而改进了往返时间的估测,使计算结果更加合理。

5.6 流量控制

5.6.1 利用滑动窗口实现流量控制

前面讨论了TCP协议保证可靠性的一些机制:连接的建立与释放、检错、序号、确认与重传技术,这一节将介绍保证TCP协议可靠性的另外一种机制:流量控制。

一般来说,为了使效率更高,我们希望发送方发送速率更快一点。但如果发送方发送得过快,使接收方来不及接收,就会造成数据的丢失,流量控制平衡了发送方发送速率与接收方接收速率。TCP将流量控制与差错控制分开。在本节,我们讨论流量控制,忽略上节讨论的差错控制,即假设发送和接收TCP的逻辑信道是无差错的。

当一个数据传输连接建立时,连接的双方都要分配一块缓冲区来存储接收到的数据。TCP提供了一种基于滑动窗口协议的流量控制机制,用接收端接收能力(缓冲区的容量)的大小来控制发送端发送的数据量。以防止由于发送端与接收端之间发送速率与接收速率的不匹配而造成的数据丢失。TCP采用可变长的滑动窗口,使发送端与接收端根据自己的处理能力和数据缓存区的大小对数据发送和接收能力做出动态调整,从而灵活性更强,也更合理。

1. 确定初始窗口值

在建立连接时,通信双方使用连接响应报文段或确认报文段中的窗口字段捎带着各自的接收窗口的尺寸,在该TCP报文段首部的窗口字段写入的数值就是当前给对方设置的发送窗口的数据上限(初始窗口值)。

2. 传输中修改窗口值

在数据传输过程中,发送方按接收方通知的窗口尺寸和序号发送一定量的数据。接收方根据接收缓冲区的使用情况在传输过程中可重新设置接收窗口值,要求对方动态地调整发送窗口的尺寸,并在发送TCP报文段或确认报文段时将新的窗口尺寸和确认号通知给发送方。

3. 窗口的大小是使用字节数来定量的

在TCP连接建立时,发送窗口由双方商定。在数据传输过程中,发送方只发送窗口尺寸内的数据,只有在收到了对方的确认后发送窗口才可前移,窗口尺寸不变,当把窗口内的数据发送完而没收到对方的确认时,就不能再发送报文段了。发送方按接收方通知的窗口尺寸和序号发送一定量的数据。正是由于发送方的发送窗口由接收端的接收能力确定,因此接收端的接收窗口总是等于发送端的发送窗口。所以,一般只使用发送窗口这个词汇。

【例 5.5】 如图 5-28 所示，是一个简单的单向传输的例子，假设发送方主机 A 向接收方主机 B 发送数据，我们现在给出发送方和接收方如何在连接建立阶段设置窗口的大小，并给出它们在数据传输阶段变化的情况。在此，为了不增加复杂性，我们假定通信是单向的，因此只给出两个单向数据传输的窗口。尽管发送方在第三个报文中将接收方的窗口设 2000 字节，但我们并不使用那个窗口。

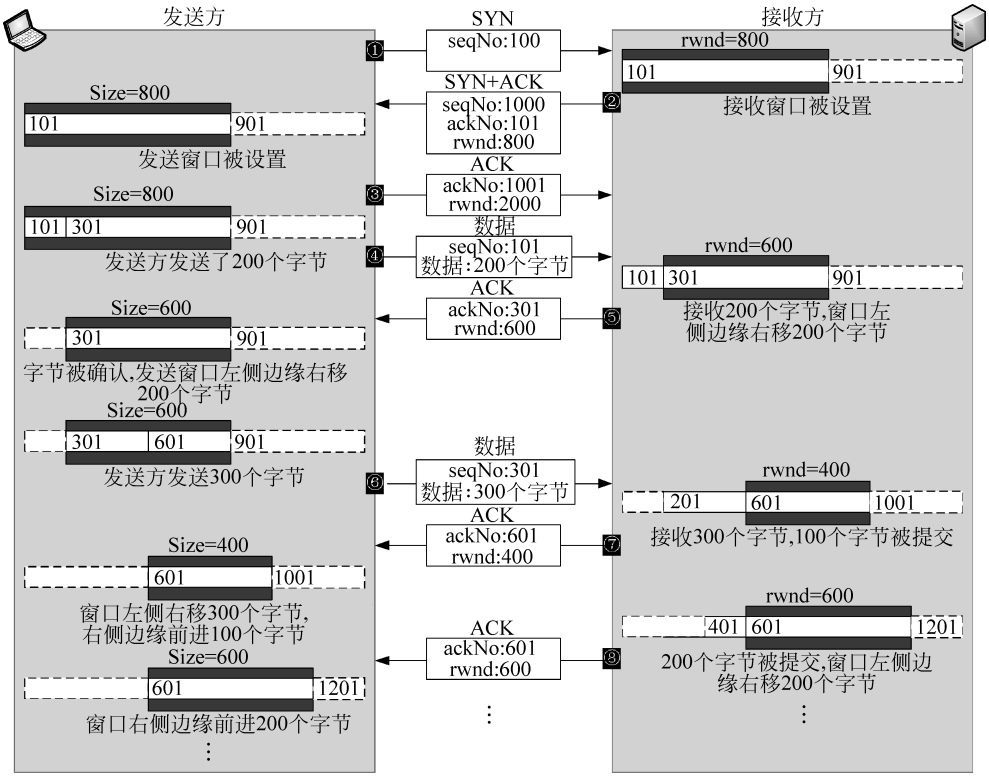

图 5-28 利用可变滑动窗口进行流量控制示例

发送方和接收方之间交换了 8 个报文。

(1) 第 1 个报文是发送方给接收方的 SYN 请求报文。这个报文声明报文段的起始字节 seqNo＝100。注意，下一个将要到达的是起始字节为 101 报文段(SYN 报文段消耗 1 个序号)。

(2) 第 2 个报文是接收方到发送方的 ACK＋SYN 报文段。报文中 ackNo＝101，这表示接收方期待接收的字节从 101 开始。该报文也指出了发送方可以设置的发送窗口大小为 800 字节。

(3) 第 3 个报文是发送方到接收方的确认报文 ACK 报文。注意，在这个报文中，发送方给出自己的接收窗口为 2000 字节，但是，在该例子中，我们不使用这个值，因为通信是单向的。

(4) 第 4 个报文段是发送方开始发送数据。发送方设置了接收方指定的接收窗口(800 字节)的大小之后，发送方的 TCP 发送 200 个字节的数据。报文开始的字节数是 101 并且携带了 200 个字节的数据(编号是 101～300)，之后发送方的窗口进行调整，发送窗口的左

侧边缘移到 301 字节位置,表示 200 个字节已经发送,等待对方确认。当这个报文被接收方接收,则接收方窗口左移到 301 字节处,表示下一个期待接收的是起始字节是 301 的报文。

(5) 第 5 个报文段是接收方给发送方的确认。接收方确认了发送方发送的 200 字节,期望下次接收 301 字节起始的报文段。该确认报文还携带了接收窗口的收缩通知,现在接收窗口为 600 个字节。在收到这个接收端的确认报文后,发送窗口左侧边缘向右移到 301 字节处,但发送窗口的右侧边缘不动(因为根据接收方的接收窗口的值,发送窗口大小正好减少 200 个字节)。

(6) 第 6 个报文是发送方发送的第 2 个数据报文。这次发送方的 TCP 发送了 300 个字节的数据,即 301～600 字节是数据。当该报文到达接收方时,接收方存储着 300 字节的数据,同时它减小自己接收窗口的大小。当接收方向高层提交了 100 个字节的数据后,接收窗口的左侧边缘向右移动 300 个字节,但是右侧边缘向右移动 100 个字节,所以接收窗口总体缩小 200 个字节,为 400 字节。

(7) 第 7 个报文是接收方向发送方发送的第 2 个确认报文。接收方确认接收数据,并再次声明接收窗口收缩到 400 个字节。当该确认报文到达发送方时,发送方只能再次收缩自己的发送窗口。只是发送窗口的左侧边缘向右移动 300 字节,而右侧边缘仅向右移动 100 字节。

(8) 第 8 个报文是接收方向发送方发送的窗口调整报文。当接收方的 TCP 向高层提交 200 个字节后,接收窗口增大到 600 个字节,接收窗口的右侧边缘向右移动 200 个字节。该报文通知发送方现在新的接收窗口的大小是 600 个字节,预期接收的报文仍然是 601 字节开始的报文段。当接收方收到该报文后,发送窗口的右侧边缘向右移动 200 个字节,发送窗口增大到 600 个字节。

我们应注意到,接收方的主机 B 进行了 3 次流量控制。第一次把窗口减小到 rwnd=600,第二次减到 rwnd=400,最后一次又增加到 rwnd=600。另外我们还应注意到,B 向 A 发送的三个确认报文都设置了 ACK=1,只有在 ACK=1 时确认号字段才有意义。

考虑一种情况,如果 B 的缓存区已满,于是 B 向 A 发送了零窗口的报文段,之后不久,B 的接收缓存又有了一些存储空间,于是 B 向 A 发送了 rwnd=400 的报文段,然而这个报文段在传送过程中丢失了,A 一直等待收到 B 发送的非零窗口的通知,而 B 也一直等待 A 发送的数据。如果没有一种解决方法,这种互相等待的死锁局面将一直延续下去。

为了解决非零窗口通知丢失造成“死锁”的现象,TCP 为每一个连接设置了一个持续(坚持)计时器(Persistence Timer)。只要 TCP 连接的一方收到对方的零窗口通知,就自动启动持续计时器。若持续计时器设置的时间到,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现在的窗口值。如果窗口仍然是零,那么收到这个报文段的一方就重新设置持续计时器;如果窗口不是零,那么死锁的僵局就可以打破了。

一般持续计时器的值设置为重传时间的数值,最大为 60s。

5.6.2 TCP 的传输效率

在数据通信时,TCP 发送端的数据来自于它上层的应用进程,发送端的应用进程不断地将数据块(其长短不一定相同)写入 TCP 的发送缓存中,TCP 再从发送缓存中取出一定

数量的数据组装成报文段逐个发送出去。接收端 TCP 收到报文段后，先将其暂存在接收缓存中并发回确认，然后接收端的应用进程在有空闲时，再从接收缓存中将数据块逐个读出。

那么发送端怎样控制发送一个报文段和确认报文段的时机呢？每次发送多长的报文合适呢？

1. 常用的 3 种选择发送时机的方法

(1) 控制报文段长度的方法。让 TCP 维持一个变量，它通常等于最大报文段长度 MSS，当发送缓存的数据达到 MSS 字节时，就组装一个 TCP 报文段发送出去。

(2) TCP 支持推送(Push)方法。当发送端的应用进程指明要发送报文段时，TCP 采用推送操作，立即发送。

(3) 计时器方法，发送端设置一个计时器，时间到了就将当前缓存区已有的数据装成一个 TCP 报文段发送出去。

但是如何控制 TCP 发送报文的时机仍然是一个较复杂的问题。因为发送时机掌握不好会导致系统效率低。一般都要适当地推迟发回确认报文的时间，并尽量使用捎带确认的方法。

2. Nagle 算法选择发送时机

在 TCP 的实现中有一种广泛使用的 Nagle 算法来控制发送时机，算法如下：

若发送端的应用进程将欲发送的数据以一个字节一个字节的方式送到发送端的 TCP 缓存中，那么发送端先将第一个字符(一个字符的长度是一个字节)发送出去，将后面到达的字符都缓存起来。当收到接收端对第一个字符的确认后，再将缓存的所有字符装成一个报文段发送出去，同时继续缓存到达的字符，待收到对上一个报文段的确认后才继续发送下一个报文段。算法还规定，只要上层到达的字符达到发送窗口大小的一半或已达到报文段的最大长度时，就立即发送这个报文段。显然，用此种方法在 TCP 中形成的报文长度不等。当上层交付字符速度较快而网络传输速率较慢时，可用这种算法形成一个较长的报文，再按此算法规定发送至网络可以明显地减少所用的网络带宽。但此种算法也存在不足之处，例如在应用层上，当将鼠标移动的信息传到远地主机时，响应会很慢，用户无法忍受，这时最好关闭这个算法。

3. 确认报文的时机

这个问题叫作糊涂窗口综合症(Silly Window Syndrome)，有时因为这个时机选择不当可能导致 TCP 的性能变坏。设想一种极端的情况：在交互式应用情况下，接收端的缓存已满，而应用进程一次只从缓存中读取一个字符，在缓存中产生一个字节的空位子，这时接收端就向发送端发送确认，并将窗口设置成一个字节。若此时接收端没有数据要发送，不能采用捎带确认方式，只好发送一个只有确认功能的报文段，它具有 20 字节的 TCP 报文首部，再加上 20 字节的 IP 数据报首部，形成一个 40 字节的 IP 数据报后将其发送到发送端。然后，发送端又发来只含有一个字符，但长度却是 41 字节的 IP 数据报(20 字节 IP 首部长，20 字节 TCP 首部长)。接收端发回确认，窗口仍为 1。这样一直进行下去，网络的传输速率极低。

解决问题的方法是让接收端等待一段时间,使得缓存能容纳一个最长的报文段或者已有一半的缓存空间处于空闲的状态才发出确认报文,并向发送端通知当前的窗口大小。同时,发送端不发送很小的报文段,而是将数据积累成足够大的报文段,或达到接收端缓存空间的一半大小才发送。这样可以很好地提高系统的效率。

上述的解决方法和 Nagle 方法可配合使用,使得发送端不再发送很小的报文段,同时,接收端的通知窗口不要太小。

5.7 TCP 拥塞控制

拥塞控制是指在拥塞发生前,应用某种策略来预防拥塞现象的产生,或在拥塞发生后消除拥塞的技术。拥塞与两个问题有关,即吞吐量和延迟,这两个概念我们在第 1 章讨论过。

5.7.1 拥塞控制的基本概念及原理

1. 拥塞控制的基本概念

拥塞(Congestion)是指到达通信子网中某一部分的分组数量太多,超出了网络所能承受的处理能力,使得该部分网络几乎不能够正确地传送任何分组,以致引起这部分乃至整个网络性能下降的现象。严重时甚至会导致所有的信息缓冲区全部占满而无法空出,使得网络通信停止,即出现所谓的死锁现象(或称为拥塞崩溃)。

这种现象跟公路网中经常遇见的交通拥挤一样,当节假日公路网中车辆大量增加时,各种走向的车流相互干扰,使每辆车到达目的地的时间都相对增加(即时延增加),甚至有时在某段公路上车辆因堵塞而无法开动(即发生局部死锁)。

拥塞的定义: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,网络的性能就要变坏,这种情况就叫拥塞。即下面不等式成立:

$$\sum \text{对资源的需求} > \text{可用资源} \tag{5-4}$$

也就是说,网络对资源的需求大于网络可用的资源。

造成拥塞的原因有很多,如果突然之间,分组流同时从多个输入线到达,并且要求输出到同一线路,这就将建立队列。如果没有足够的空间来保存这些分组,有些分组就会丢失。结点的处理器速度慢也能导致拥塞。在由路由器互连形成的网络中,如果路由器处理器的处理速度太慢,以至于不能及时地执行缓冲区排队、更新路由表等任务,那么,即使有多余的线路容量,也可能使队列饱和。类似的低带宽线路也会导致拥塞。如果结点没有空闲缓冲区,它必须丢弃新到来的分组。当有一个分组被丢弃时,发送方可能会因为超时而重传此分组,或许要重发多次。由于发送方在未收到确认之前必须在缓冲区中保存该分组,故拥塞将迫使发送方不能释放在通常情况下,应该释放的缓冲区。这样便形成了恶性循环,使拥塞加重。

拥塞控制与流量控制不同。拥塞控制主要用于保证网络能够传送待传送的分组,将设计网络中所有与之相关的主机、路由器、路由器存储—转发处理的行为,它的目的是保持网络中的分组数不要超过某一限度,否则,网络性能将显著下降。一个通信子网一般由许多路

由器和通信链路组成。发送者可能以过高的速率向网络发送数据，这些分组可能会在路由器中排队，可能造成缓冲区溢出，进而导致分组丢失、重传，降低网络的性能。拥塞控制确保通信子网能够有效地为主机传递分组，这是一个全局性的问题，涉及所有主机、所有路由器、路由器中的存储—转发处理以及所有导致削弱通信子网能力的其他因素，是一种全局性的控制措施。流量控制只设计发送方和接收方之间的点到点的流量控制行为，主要用于确保发送方的发送速率与接收方的缓冲区容量相匹配，以防止在接收方缓冲区不足时发生数据丢失。

加上合适的拥塞控制后，网络就不易出现拥塞现象和死锁。付出的代价就是：当提供的负载较小时，有拥塞控制的吞吐量反而比无拥塞控制时要小。

如图 5-29 为拥塞控制与网络延迟及网络负载之间的关系；图 5-30 为拥塞控制与网络吞吐量及网络负载之间的关系。

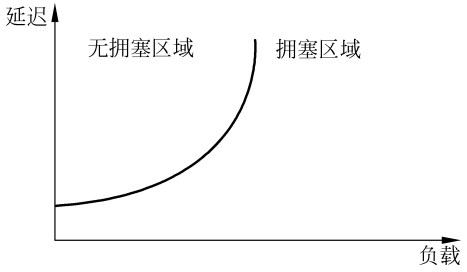

图 5-29 作为负载函数的延迟曲线

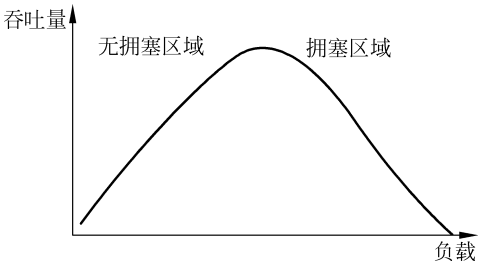

图 5-30 作为负载函数的吞吐量曲线

当负载远远小于网络容量时，延迟是最小的。最小延迟由传播时延和处理时延组成，排队时延可忽略。然而，当负载达到网络容量时，延迟急剧增加，因此现在要将排队时延加到总时延当中。注意，当负载大于网络容量时延迟为无限大。

当负载小于网络容量时，吞吐量随着负载的加大而增加。我们希望在分组达到网络容量之后负载保持不变，但是吞吐量会急剧下降，原因在于路由器在丢弃分组。当分组超过网络容量时，路由器中的队列变满造成分组溢出。然而，丢弃的分组不会减少网络中的分组数量，因为当分组没有到达目的端时，源端使用超时重传机制。

2. 拥塞控制的基本原理

从原理上讲，寻找拥塞控制的方案无非是寻找使不等式(5-4)不再成立的条件。这或者是增加网络的某些可用资源(如增加链路的带宽)，或减少一些用户对某些资源的需求(如拒

绝新的建立连接的请求,或要求用户减轻其负荷,这些都属于以降低服务质量为代价)。但正如前面所述,在采用某种措施时,还必须考虑到该措施带来的其他影响。

实践证明,拥塞控制是很难设计的,因为它是一个动态的(不是静态的)问题。而当前高速化的网络,很容易出现缓存不够大而造成分组丢失。但分组丢失是网络拥塞的征兆而不是原因。在很多情况下,甚至正是拥塞控制机制本身成为引起网络性能恶化甚至发生死锁的原因。这点更应该引起重视。

从控制论的角度看拥塞控制,可以把拥塞控制算法分成开环控制和闭环控制两大类。开环控制算法通过良好的网络系统设计来避免拥塞问题的发生。在进行网络设计时,应用某种策略来预防拥塞现象的发生,即事先将有关发生拥塞的因素考虑周到,力求网络在工作中不产生拥塞。一旦整个系统运行起来,就不能中途进行改正。在网络运行过程中,何时接收新分组,何时丢弃分组以及丢弃哪些分组都是事先规划好的,并不考虑当前的网络流量状况。闭环控制算法是在网络发生拥塞后,通过反馈机制来调整当前网络流量,使网络流量与网络可用资源相协调,从而使网络拥塞问题得到缓解。由于闭环控制算法能够根据当前网络状况对流量进行动态控制,具有较高的效率。因此,现代网络系统大都采用闭环控制算法来解决网络拥塞问题。

在闭环控制算法中,关键措施在于:

(1) 监测机制,以便检测网络何时何地发生了拥塞。

(2) 反馈机制,将发生拥塞的信息传送到可能采取行动的地方(如控制点)。

(3) 调整机制,调整网络的运行以解决出现的问题。

监测机制将根据当前网络状况来监测网络是否发生了拥塞,判断的依据主要有:因缺少缓冲区空间而丢弃的分组数量、平均分组队列长度、超时重发分组的数量、平均分组延迟时间等。如果检测数据超过了临界值,则意味着可能发生了拥塞。

反馈机制将发生拥塞的信息从拥塞点传送到控制点。反馈方式有显示反馈和隐式反馈两种。显示反馈采用由拥塞点向控制点反馈一个警告分组的方式来通告网络已发生拥塞;隐式反馈通过发送端(控制点)观察应答分组返回所用时间的方式来判断网络是否发生了拥塞。如在路由器转发的分组中保留一位或一个字段,以此表示网络中的拥塞情况。当然,通知拥塞发生的分组同样会使网络更加拥塞。

调整机制通过拥塞点和控制点相互协调来解决拥塞问题。控制点通过降低负载,即降低分组发送速率来缓解拥塞;拥塞点通过负载脱落(Load Shedding),即丢弃一些分组来疏导通信,或者通过启用备份的空闲系统资源来提高通信容量。

将以上方法应用于实际可总结出以下几种拥塞控制方法:

(1) 缓冲区预分配方法

缓冲区预分配方法(Buffer Allocation)用于虚电路分组交换网中。在建立虚电路时,让呼叫请求分组途经的结点为虚电路预先分配一个或多个数据缓冲区。若某个结点缓冲器已被占满,则呼叫请求分组会绕过这个结点选择其他路径,或者返回一个“忙”信号给呼叫者。

(2) 分组丢弃方法

分组丢弃方法(Packet Elimination)不必预先保留缓冲区,当缓冲区占满时,将到来的分组丢弃。若通信子网提供的是数据报服务,则用分组丢弃法来防止拥挤的发生,从而不会引起大的影响。但若通信子网提供的是虚电路服务,则必须在某处保存被丢弃分组的备份,

以便拥塞解决后能重新传送。

有两种解决被丢弃分组重发的方法：一种是让发送被丢弃分组的结点超时，并重新发送分组直至分组被收到；另一种是让发送被丢弃分组的结点在一定次数后放弃发送，并迫使数据源结点超时而重新开始发送。

(3) 通信量控制方法

拥塞发生的主要原因在于通信量常常是突发性的，如果主机能以一个恒定的速率发送分组，拥塞将会少得多。而对于子网来说，子网强迫分组以某种预定的速率传送。这种方法被广泛应用在 ATM 网络中，也称为通信量整形(Traffic Shaping)。

5.7.2 TCP 的拥塞控制

利用滑动窗口技术进行流量控制可以使接收端来得及接收发送端发送的报文，即接收方使用接收窗口(rwnd)的大小来控制发送窗口大小。但使用这个策略保证了接收窗口不会被发送方的数据吞没(发生溢出)。但是，实现滑动窗口技术并非仅仅为了使接收方来得及接收。如果发送方发出的报文过多，就有可能使网络负荷过重，从而导致报文段传输的时延增大，使发送主机由于不能及时收到确认而重传更多的报文段，使网络发生拥塞。而采用滑动窗口机制还可对网络进行拥塞控制，即将网络中的分组数量维持在一定的数量之下，当超过该数值时，网络的性能会急剧恶化。为了避免发生拥塞，主机应当降低发送速率。

综上所述，发送方的主机在发送数据时，要从两方面来考虑，既要考虑接收方的接收能力，又要从全局考虑不要使网络发生拥塞。因此，关于拥塞问题的讨论，我们从以下几个方面来考虑。

1. 拥塞窗口

对于每一个 TCP 连接，需要有以下两个状态变量：

(1) 接收窗口 rwnd(Receiver Window)，又称为通知窗口(Advertised Window)。这是来自接收方对发送方的流量控制。接收方的接收窗口值是放在 TCP 首部的窗口字段发送给发送方。它是接收方根据其目前的接收缓存大小所提供的最新窗口值。

(2) 拥塞窗口 cwnd(Congestion Window)，是来自发送方的拥塞控制，即发送方根据网络拥塞情况而设置的窗口值。

因此，发送方的发送窗口上限值应取接收方根据接收能力的通知窗口和发送方根据拥塞情况得出的拥塞窗口的最小值，即

$$\text{实际发送窗口的上限值} = \min[\text{rwnd}, \text{cwnd}]$$

也就是说，TCP 发送方的发送数量是由目的主机的接收窗口或网络的拥塞窗口中较小的值来制约。

2. 拥塞检测

在讨论 cwnd 的值如何设置之前，我们先了解一下发送方的 TCP 如何发现网络中出现了拥塞。

首先就是出现了超时(Time-Out)。如果发送方的 TCP 在超时之前没有收到对于某个或某些报文的确认，那么它就假设相应报文丢失了，并且丢失可能是由拥塞引起的。

其次是如果收到三次重复的 ACK(四个带有相同确认号的 ACK)。因为当接收方的 TCP 发送一个重复 ACK,这是报文已经被延迟的信号。但是如果发送三次重复的 ACK 是丢失报文的标志,这可能是由于网络拥塞造成的。然而,三次重复 ACK 的情况下拥塞的严重程度低于超时情况。当接收方发送三次重复 ACK 时,这意味着一个报文丢失,但后续的三个报文已经接收到。网络或者处于轻度拥塞或者已经从拥塞中恢复。

TCP 拥塞控制中,TCP 的发送方只使用一种反馈从另一端来检测拥塞:即确认报 ACK。没有周期性地、及时地接收到 ACK,而导致超时,是严重拥塞的标志;接到三次重复 ACK 报文是网络中轻微拥塞的标志。

3. 拥塞策略及算法

2009 年 9 月公布的草案标准[RFC 5681]定义了以下 4 种算法,即慢开始(Slow-Start)、拥塞避免(Congestion-Avoidance)、快重传(Fast-Retransmit)和快恢复(Fast-Recovery)。下面介绍这些算法的要点。

下面讨论的拥塞控制也称为基于窗口的拥塞控制。为此,发送方的 TCP 维持一个拥塞窗口 cwnd 的状态变量,拥塞窗口是根据网络的拥塞程度来设定的,并且动态地变化着。发送方让自己的发送窗口等于接收端的接收窗口和拥塞窗口的最小值。

发送端确定拥塞窗口的原则是:只要网络没有出现拥塞,发送端就使拥塞窗口再增大一些,以便将更多的分组发送出去。但只要网络出现拥塞,发送端就使拥塞窗口减小一些,以减少注入网络中的分组数。发送端又是怎样发现网络出现拥塞呢?我们知道,当网络发生拥塞时,路由器就要丢弃分组。如果通信线路带来的差错所引起的分组丢失的概率很小(远小于 1%),那么只要出现分组丢失或时延过长而导致超时重传,就意味着网络某处发生了拥塞,即网络拥塞是引起超时重传的主要原因。

(1) 慢开始和拥塞避免

那么发送方如何具体控制拥塞窗口 cwnd 的大小。我们从“慢开始”算法开始。

① 慢开始:指数增加

这个算法的名字不是很合适,有些误导。该算法启动慢,但它是以指数增长的,即增长是非常快的。为说明原理方便,我们给出三个假设条件:

- 用报文段的个数作为窗口大小的单位,每个报文的长度为一个 MSS;
- 每个报文段是同长度的;
- 接收端窗口 rwnd 足够大,因此发送窗口只受发送方的拥塞窗口的制约。

如我们之前讨论的,MSS 是连接建立期间由选项协商产生的值。

如图 5-31 所示,开始时,发送端先设置 cwnd=1(一个 MSS),发送第一个报文段 M_1,接收方收到后发回确认 M_1。发送端收到对 M_1 确认后,将 cwnd 从 1 增大到 2,于是发送端可以接着发送 M_2 和 M_3 两个报文段。接收端收到后发送对 M_2 和 M_3 的确认。发送端每收到一个对新报文段的确认(重传的不算),就使发送端的拥塞窗口加 1,因此现在发送端的 cwnd 又从 2 增大到 4,并可发送 $M_4 \sim M_7$ 共 4 个报文段。因此使用慢开始算法后,每经过一个传输轮次(Transmission Round),拥塞窗口就加倍。可见慢开始的“慢”并不是指 cwnd 的增长速率慢。经验证明,较好的方法是先探测一下,即由小到大逐渐增大发送窗口,即拥塞窗口的数值,即使 cwnd 增长得很快,同一开始就将 cwnd 设置为较大的数值相比,使用慢

开始算法可以使发送端在开始发送时向网络注入的分组数大大减少。这对防止网络出现拥塞是个非常有力的措施。

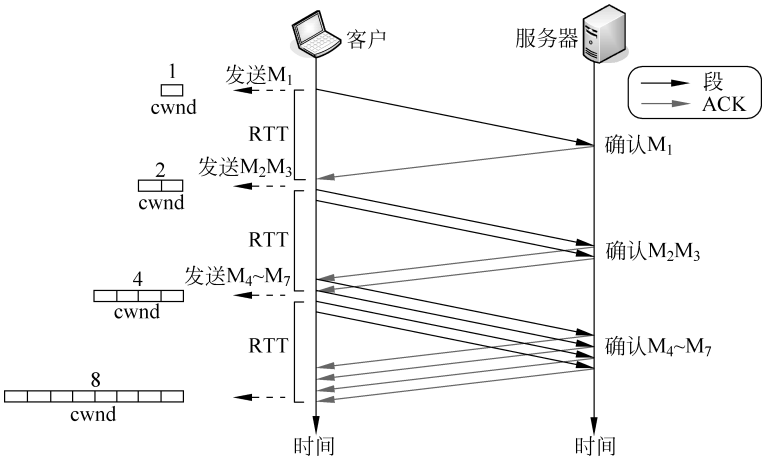

图 5-31 慢开始算法示例

从图 5-31 可以看出，一个传输轮次时间其实就是往返时间 RTT。不过使用“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地发送出去，并已收到了对已发送的最后一个字节的确认。例如，拥塞窗口 cwnd 的大小是 8 个报文段，那么这时的往返时间 RTT 就是连续发送 8 个报文段，并收到对这 8 个报文段的确认总共经历的时间。

慢开始不能一直进行下去，肯定存在一个停止该阶段的阈值。即发送方为了防止拥塞窗口 cwnd 的增长过大而引起网络拥塞，还需要另一个状态变量，即慢开始门限（阈值）ssthresh。在慢开始与拥塞避免算法中对于拥塞窗口（cwnd）和慢开始门限（ssthresh）之间的关系可以做这样的规定：

- 当 cwnd＜ssthresh 时，使用慢开始算法。
- 当 cwnd＞ssthresh 时，停止使用慢开始算法，使用拥塞避免算法。
- 当 cwnd＝ssthresh 时，既可以使用慢开始算法，也可以使用拥塞避免算法。

在慢开始阶段，如果拥塞窗口为 32 时出现超时，则发送端就可以将慢开始门限设置为出现拥塞的拥塞窗口值 32 的一半，即 ssthresh＝16。

当窗口中的字节数达到总阈值时，慢启动停止且下一个阶段——拥塞避免开始。

② 拥塞避免：线性增加

拥塞避免就是为了降低拥塞窗口的增长速率，避免拥塞。这个算法是线性增加 cwnd 而非指数增加。具体的做法如下：

当拥塞窗口达到慢开始阈值时，慢开始算法停止，而拥塞避免算法开始。使发送端的拥塞窗口 cwnd 每经过一个往返时间 RTT 就增加一个 MSS 的大小而不是加倍（而不管在 RTT 时间内收到了几个确认）。这样，拥塞窗口 cwnd 按线性规律缓慢增长，比按指数增长的慢开始算法缓慢得多。无论在慢开始阶段还是在拥塞避免阶段，只要发送端没有按时收到确认 ACK 或收到了重复的确认 ACK，就认为网络出现拥塞，就要将慢开始门限 ssthresh 设置为出现拥塞时的发送窗口值的一半（但不能小于 2）。这样设置的考虑就是：既然出现了网络拥塞，那就要减少向网络注入的分组数，然后将拥塞窗口 cwnd 重新设置为 1，并执行

慢开始算法。这样做的目的是要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。如图 5-32 所示是一个拥塞避免的例子。

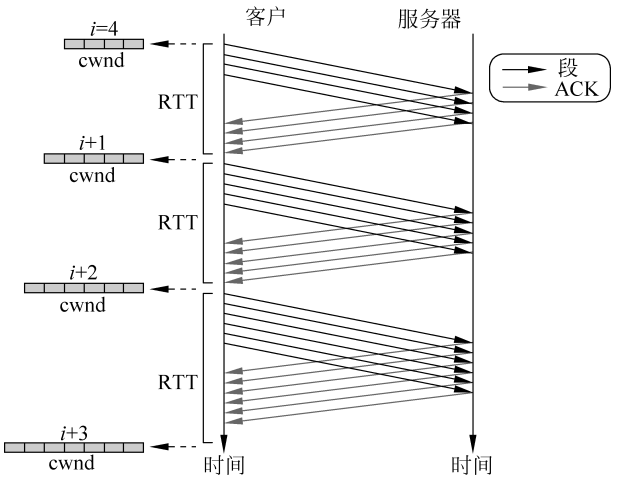

图 5-32 拥塞避免算法示例

图 5-33 说明了拥塞控制的具体过程。常见的执行步骤如下:

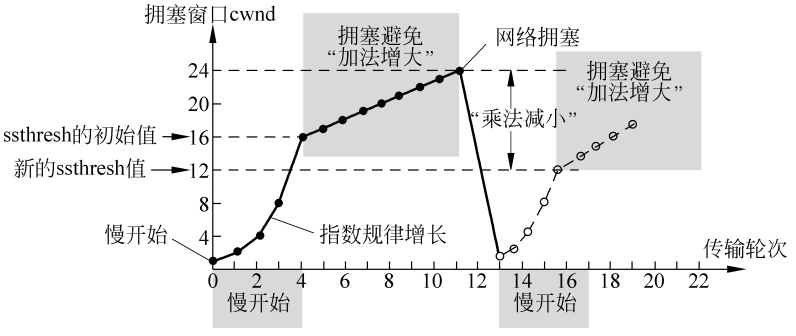

图 5-33 慢开始和拥塞避免算法的实现示例

在 TCP 连接初始化时,将拥塞窗口置为 1,前面已说过,为了便于理解,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。我们假定接收端窗口足够大,因此发送窗口值等于拥塞窗口的值。

• 慢开始阶段

在慢开始阶段,当经过 4 个传输轮次,拥塞窗口按指数规律增长到慢开始门限 16(即当 cwnd=16 时)时,改为执行拥塞避免算法。传输轮次 1～4 使用的拥塞窗口值分别为 2、4、8、16。

• 拥塞避免阶段

进入拥塞避免阶段后,拥塞窗口按线性规律缓慢增长,假定拥塞窗口的数值增大到 24 时,网络出现超时,即表明网络出现拥塞了。这时要重新设置慢开始门限与拥塞窗口的值。然后再一次进入慢开始与拥塞避免阶段。在这个阶段中,传输轮次 5～12 使用的拥塞窗口值分别为 17～24。

• 重新进入慢开始与拥塞避免阶段

出现网络拥塞后，将慢开始门限调整为 12，即发送窗口数值 24 的一半，此时拥塞窗口再重新设置为 1，然后重新执行慢开始和拥塞避免算法。传输轮次 13～17 使用的拥塞窗口值分别为 1、2、4、8、12。由于第 2 次的 ssthresh 设置为 12，第 17 个传输轮次的拥塞窗口值不能大于 12，只能取 12。传输轮次 18～21 使用的拥塞窗口值分别为 13、14、15、16。若网络又出现超时，过程同上。

③ 乘法减小与加法增大

在 TCP 拥塞控制的文献中经常可看见“乘法减小”(Multiplicative Decrease)和“加法增大”(Additive Increase)这样的提法。“乘法减小”主要对慢开始门限值 ssthresh 的计算方法，不论在慢开始阶段还是拥塞避免阶段，只要出现一次超时(即出现一次网络拥塞)，就将慢开始门限值 ssthresh 设置为当前的拥塞窗口值乘以 0.5(乘法减小)。当网络频繁出现拥塞时，ssthresh 值就下降得很快，这样可以大大减少注入网络中的分组数。而“加法增大”是指执行拥塞避免算法后，对拥塞窗口 cwnd 的计算方法。当收到对所发报文段的确认后就将拥塞窗口 cwnd 增加一个 MSS 大小，使拥塞窗口缓慢增大(加 1 增大)，以防止网络过早出现拥塞。

上面两种方法常结合起来称为 AIMD(Additive Increase Multiplicative Decrease，加法增大乘法减小)算法。

还要再次强调的是，“拥塞避免”并非指完全能够避免了拥塞。“拥塞避免”是说在拥塞避免阶段将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。

(2) 快重传和快恢复

上面介绍的 AIMD 方法适合网络出现拥塞的情况。而网络还有一个新的情况，就是发送方一连收到 3 个对同一个报文段的重复确认。关于这个问题要解释如下：

有时，个别报文段会在网络中丢失，但实际上网络并未发生拥塞。如果发送方迟迟收不到确认，就会产生超时，就会误认为网络发生了拥塞。这就导致发送方错误地启动慢开始，把拥塞窗口 cwnd 又设置为 1，因而降低了传输效率。

而 AIMD 是在 TCP 中最早使用的拥塞控制算法(又称为 TCP Tahao)，用相同的方式来对待拥塞检测的两种情况，即超时和发送方收到三次重复 ACK。但后来人们发现这种拥塞控制算法还需要改进，因为如果只使用慢开始和拥塞避免算法，还有两个问题没有得到解决：

① 有时一条 TCP 连接会因等待超时重传而空闲较长时间；

② 当发送端发现网络拥塞时就一律将窗口下降为 1，然后执行慢开始算法，会使网络不能很快地恢复到正常工作状态。

于是在新版的 TCP(Reno TCP)的拥塞控制中又增加了两个新的拥塞控制算法，这就是快重传和快恢复。在这个版本中用不同的方法来处理超时和发送方连续收到三次重复 ACK 这两种情况。即如果发送超时，TCP 进入慢启动状态，这时不使用快重传的情况；如果发送方收到三次重复的 ACK，则 TCP 进入快重传与快恢复算法，并且只要有更多的重复 ACK 到达，它就保持这种状态。

a. 快重传的工作原理

对第一个问题可采用快重传解决，即接收端每收到一个报文后，不等待自己发送数据捎

带 ACK,而是立即发出确认 ACK。当发送端发现某个报文丢失时,不等待超时计时器到时,便立即重传丢失的报文。

如图 5-34 所示,假定发送端发送了 M_1～M_4 共 4 个报文段,接收端每收到一个报文段后都要立即发出确认 ACK。当接收端收到了 M_1 和 M_2 后,就发出确认 ACK_2 和 ACK_3。假定由于网络拥塞使 M_3 丢失了,接收端后来收到下一个 M_4,发现其序号不对,但仍收下放在缓存中,同时发出确认,不过发出的是重复的 ACK_3(不能够发送 ACK_5,因为 ACK_5 表示 M_3 和 M_4 都已经收到了)。这样,发送端知道现在可能是网络出现了拥塞造成报文丢失,但也可能是报文段 M_3 尚滞留在网络中的某处,还要经过较长的时延才能到达接收端。发送端接着发送 M_5 和 M_6。接收端收到了 M_5 和 M_6 后,也还要分别发出重复的 ACK_3。这样,发送端共收到了接收端的 4 个 ACK_3,其中三个是重复的。快重传算法规定,发送端只要一连收到三个重复的 ACK 即可断定有报文丢失了,就应立即重传丢失的报文段 M_3 而不必继续等待为 M_3 设置的超时计时器的超时。不难看出,快重传并非取消超时计时器,而是在某些情况下可更早地重传丢失的报文段。

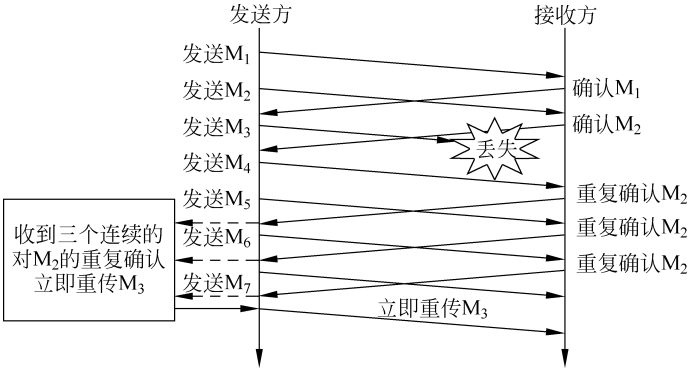

图 5-34 快重传示例

由于发送方能尽早重传未被确认的报文段,因此采用快重传算法后可以使整个网络的吞吐量提高约 20%。

b. 快恢复算法的工作原理

与快重传配合使用的还有快恢复算法,对第二个问题采用快恢复算法可以较好地解决。其具体步骤如下:

第一:当发送端收到连续三个重复的 ACK 时,就重新按照前面讲过的"乘法减小"重新设置慢开始门限 ssthresh=cwnd/2。这一点和慢开始算法是一样的。需要注意的是,接下来不执行慢开始算法。

第二:发送方知道现在只是丢失了个别的报文段,因此与慢开始不同之处是拥塞窗口 cwnd 不是设置为 1,而是设置为收到连续三个重复 ACK 时的拥塞窗口值的一半。然后开始拥塞避免算法,使窗口缓慢地增大。

第三:若收到的重复的 ACK 为 n 个,则将 cwnd 设置为 ssthresh+n×MSS。n 为收到的重复 ACK 个数($n \geqslant 3$)。这样做的理由是:如果发送端收到 n 个重复的 ACK 表明有 n 个报文已经离开了网络,它们不会再消耗网络的资源,这 n 个报文是停留在接收端的缓存中(接收端发送出三个重复的 ACK 就证明了这个事实),可见现在网络中并不是堆积了报

文而是减少了三个报文，这些报文已到达接收端。因此，将拥塞窗口扩大些并不会加剧网络的拥塞。

第四：如果发生超时，TCP 假设网络中有真实的拥塞，进入慢开始状态。

如图 5-35 所示是快重传与快恢复的示意图，并标明了“TCP Reno 版本”，是目前使用最多的版本。图中还画出了已经废弃不用的虚线部分（TCP Tahao 版本）。请注意，它们的区别是：新的 TCP Reno 版本在快重传之后采用的是快恢复算法而不是慢开始算法。

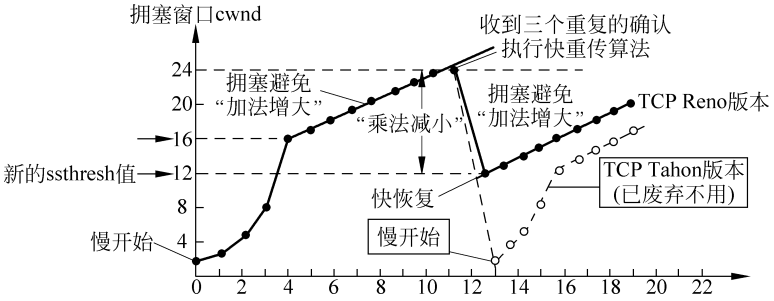

图 5-35 从连续收到三个重复的确认转入拥塞避免

在采用快恢复算法时，慢开始算法只是在 TCP 连接建立时和网络出现超时时才使用。采用这样的拥塞控制方法使得 TCP 的性能有明显的改进。

5.8 本章经典习题与解析

通过本章内容的学习，我们已经对计算机网络的传输层的功能、特点及其相关的协议有了一定的了解，为了加深对传输层相关概念的理解，特例举如下经典习题解析，帮助大家学习。例题中可能会涉及以前章节的知识点，请适当回顾相关章节的内容。

1. 在停止等待协议中，如果收到重复的报文段时不予理睬（即悄悄地丢弃它，而其他什么也不做）是否可行？试举出具体例子说明理由。

解答：不可行。试考虑下面情况：A 发送报文段 M_1，B 收到后发送确认，但这个确认丢失了。

A 超时重传报文段 M_1，B 收到后不予理睬（因为重复）。这就导致 A 再次超时重传报文段 M_1。

B 收到重复的报文段都不予理睬，A 就一直超时重传报文段 M_1。可见，如果收到重复的报文段时不予理睬是不行的，会导致一直重传。

2. IP 数据报的分片和重组是由 IP 协议控制的，而对 TCP 协议而言是透明的，这是否意味着 TCP 不用担心 IP 数据报以错误的次序到达？为什么？

解析：不是。尽管 IP 数据报的分片和重组是由 IP 协议控制的，但是由于 IP 提供的是无连接、不可靠的网络服务，IP 数据报到达的顺序可能会不同，会出现乱序现象，因此 TCP 必须提供能进行差错控制和处理乱序情况的能力。

3. 为什么 UDP 的校验和要与 IP 的校验和分开进行？你是否反对在包含 UDP 用户数据报的整个 IP 数据报中仅使用一个校验和的协议？

解析：反对。UDP 与 IP 协议所处的协议层不同，UDP 处于传输层，而 IP 处于网络层。如果在整个 IP 数据报中只使用一个校验和，则需要对 IP 数据报的整个部分进行校验，包括首部与数据部分。这样，当 IP 数据报在通信子网中进行传输时，由于每个路由器都需要对接收的整个 IP 数据报进行校验以判断是否出现了传输差错，路由器用于处理每个 IP 数据报的时间无疑会增加，也就是路由器的负载将会变重。因此，IP 数据报的传输时延将会变长，同时，传输效率也会下降。

反之，如果整个 IP 数据报的校验分为独立的两次校验，即 UDP 对 UDP 数据报的首部以及数据部分进行校验，IP 仅对 IP 数据报的首部进行校验，路由器只需要校验 IP 数据报的首部，而较长的数据部分只由目的端完成。这大大降低了路由器处理 IP 数据报的时间，进而减小了 IP 数据报的传输延迟。而且，两次校验相互独立，IP 与 UDP 采用的校验方法可以相同也可以不同。

因此，在包含 UDP 用户数据报的整个 IP 数据报中，不应该采用仅使用一个校验和的机制。

4. 一个应用程序用 UDP，到了 IP 层将数据报再划分 4 个数据报片发送出去。结果前两个数据报片丢失，后两个到达目的端。过了一段时间应用进程重传 UDP，而 IP 层仍然划分为 4 个数据报片传送。结果这次前两个数据报片到达目的端，而后两个丢失。试问：在目的端能否将两次传输的 4 个数据报片组装成为完整的数据报？假定目的端第一次收到的后两个数据报片仍然保持在目的端的接收缓存中。

解析：不能。因为重传时，IP 数据报的标识字段会有另一个标示符。仅当标识位相同的数据报片才能组装成一个 IP 数据报。前两个数据报片的标示符和后两个 IP 数据报片的标示符不同，所以不能将两次传输的 4 个数据报片组装成一个 IP 数据报。

5. 假定使用连续 ARQ 协议，发送窗口大小是 3，序号范围是[0，15]，而传输媒体保证在接收方能够按序收到分组。在某一时刻，在接收方，下一个期望收到的序号是 5。试问：

(1) 在发送方的发送窗口中可能出现的序号组合有哪些？

(2) 接收方已经发送出的，但仍滞留在网络中（即还未到达发送方）的确认分组可能有哪些？说明这些确认分组是用来确认哪些序号的分组。

解析：(1)在发送方的发送窗口中可能出现的序号组合有：[2，4]，[3，5]，[4，6]，[5，7]中的任何一个。

(2) 接收方已经发送出的，但仍滞留在网络中（即还未到达发送方）的确认分组为 2，3，4。这些确认是用来确认序号为 2，3，4 的分组的。

6. 设 TCP 使用的最大窗口尺寸为 64KB，TCP 报文在网络上的平均往返时间为 20ms，问 TCP 所能得到的最大吞吐量是多少？(假设传输信道的带宽是不受限的)

解析：最大吞吐量表明在 1 个 RTT 内将窗口中的字节全部发送完毕。在平均往返时间 20ms 内，发送的最大数据量为最大窗口值，即 64×1024B。

$$64\times1024\times8\div(20\times10^{-3})\approx26.2(\text{Mbps})$$

因此，所能得到的最大吞吐量是 26.2(Mbps)。

7. 网络允许的最大报文段的长度为 128 字节，序号用 8 比特表示，报文段在网络中的寿命为 30s。求发送报文段的一方所能达到的最高数据率。

解析：具有相同编号的报文段不应该同时在网络中传输，必须保证当序列号循环回来

重复使用的时候，具有相同序列号的报文段已经从网络中消失。现在序号用8位表示，报文段的寿命为30s，那么在30s的时间内发送方发送的报文段的数目不能多于255个。

$$255\times128\times8\div30=8704(\text{bps})$$

所以，每一条TCP连接所能达到的最高数据率为8704bps。

8. 设TCP的拥塞窗口的慢开始门限值初始为12(单位为报文段)，当拥塞窗口达到16时出现超时，再次进入慢启动过程。从此时起若恢复到超时时刻的拥塞窗口大小，则需要的往返时间次数是多少？

解析：在慢启动和拥塞避免算法中，拥塞窗口初始为1，窗口大小开始按指数增长。当拥塞窗口大于慢开始门限后停止使用慢开始算法，改用拥塞避免算法。此处慢开始的门限值初始为12，当拥塞窗口增大到12时改用拥塞避免算法，窗口大小按线性增长，每次增加1个报文段，当增加到16时，出现超时，重新设门限值为8(16的1/2)，拥塞窗口再重新设为1，执行慢启动算法，到门限值8时执行拥塞避免算法。

这样，拥塞窗口的变化为1,2,4,8,12,13,14,15,16,1,2,4,8,9,10,11,12,13.14,15,16,…可见从出现超时时拥塞窗口为16到恢复拥塞窗口大小为16，需要的往返时间次数是12。

9. 一个TCP首部的数据信息(以十六进制表示为：0x0D 28 00 15 50 5F A9 06 00 00 00 00 70 02 40 00 C0 29 00 00。TCP首部的格式如图5-17所示。请回答：

(1) 源端口号和目的端口号各是多少？

(2) 发送的序列号是多少？确认号是多少？

(3) TCP首部的长度是多少？

(4) 这是一个使用什么协议的TCP连接？该TCP连接的状态是什么？

解析：(1) 源端口号为第1、第2字节，即0D 28，转换为十进制数3368。目的端口号为第3、第4字节，即00 15，转换为十进制数21。

(2) 第5～第8字节为序列号，即50 5F A9 06，转换为十进制数6269190。第9～第12字节为确认号，即00 00 00 00，十进制为0。

(3) 第13字节的前4位为TCP首部的长度，这里的值是7(以4b为单位)，故乘以4后得到TCP首部的长度为28字节，说明该TCP首部还有8字节的选项数据。

(4) 根据目的端口是21可以知道这是一条FTP的连接，而TCP的状态则需要分析第14字节。第14字节的值为02，即SYN置为1了，而且ACK＝0表示该数据段没有捎带的确认，这说明是第一次握手时发出的TCP连接请求报文。

10. TCP协议是面向连接的，但TCP使用的IP协议却是无连接的。这两种协议有哪些主要的区别？

解析：TCP是面向连接的、可靠的(无丢失、无重复、按序交付)、全双工的应用进程之间的通信，有流量控制和拥塞控制等功能。IP协议提供的是无连接的、不可靠的(可能丢失、可能重复、可能失序)、尽最大努力交付的主机到主机之间的通信，有无流量控制和拥塞控制等功能。但TCP所使用的网络则可以是面向连接的(如X.25网络)，但也可以是无连接的(如现在大量使用的IP网络)。选择无连接网络就使得整个系统非常灵活，当然也带来了一些问题。

显然，TCP提供的功能和服务要比IP所能提供的多得多。这是因为TCP使用了诸如

序号、确认、窗口通知、计时器等机制,因而可以检测出有差错的报文、重复的报文和失序的报文。

11. TCP 协议能够实现可靠的端到端传输。在数据链路层和网络层的传输还有没有必要来保证可靠传输呢?

解析:在旧的 OSI 体系中,在数据链路层使用 HDLC 协议而在网络层使用 X.25 协议,

这些协议都有确认机制和窗口机制,因而能够保证可靠传输。但是技术的进步使得链路的传输已经相当可靠了,因此在数据链路层和网络层重复地保证可靠传输就显得多余了。现在互联网在链路层使用的 PPP 协议和在网络层使用的 IP 协议都没有确认机制和窗口机制。如果出现差错就由传输层的 TCP 来处理(但若使用 UDP 则传输层也不处理出错的问题)。

12. TCP 都使用哪些计时器?

解析:TCP 共使用以下四种计时器,即重传计时器、持续计时器、保活计时器和时间等待计时器。

这几个计时器的功能如下:

重传计时器:

当 TCP 发送报文段时,就创建该特定报文段的重传计时器。可能发生两种情况:

(1) 若在计时器截止时间到来之前收到了对此特定报文段的确认,则撤销此计时器。

(2) 若在收到了对此特定报文段的确认之前计时器截止期到,则重传此报文段,并将计时器复位。

持续计时器:

为了应对零窗口大小通知,TCP 需要另一个计时器。假定“接收 TCP”给出了窗口大小为零,“发送 TCP”就停止传送报文段,直到“接收 TCP”发送确认并给出一个非零的窗口大小。但这个确认可能会丢失。我们知道在 TCP 中,对确认报文段是不发送确认的。若确认丢失了,“接收 TCP”并不知道,而是会认为它已经完成任务了,并等待着“发送 TCP”,接着会发送更多的报文段。但“发送 TCP”由于没有收到确认,就等待对方发送确认来通知窗口的大小。双方的 TCP 都在永远地等待着对方。

要打开这种死锁,TCP 为每一个连接使用一个持续计时器。当“发送 TCP”收到一个窗口大小为零的确认时,就启动持续计时器。当持续计时器期限到时,“发送 TCP”就发送一个特殊的报文段,叫作探测报文段。这个报文段只有一个字节的数据。它有一个序号,但它的序号永远不需要确认;甚至在计算对其他部分数据的确认时,该序号也被忽略。探测报文段提醒接收 TCP:确认已丢失,必须重传。

持续计时器的值设置为重传时间的数值。但是,若没有收到从接收端来的响应,则需发送另一个探测报文段,并将持续计时器的值加倍和复位,直到这个值增大到门限值(通常是 60s)为止。在这以后,发送端每隔 60s 就发送一个探测报文段,直到窗口重新打开。

保活计时器:

保活计时器使用在某些实现中,用来防止在两个 TCP 之间的连接出现长时期的空闲。假定客户打开了到服务器的连接,传送了一些数据,然后就保持静默了。也许这个客户出故障了。在这种情况下,这个连接将永远地处于打开状态。

要解决这种问题,在大多数的实现中都是使服务器设置保活计时器。每当服务器收到

客户的信息，就将计时器复位。超时通常设置为 2h。若服务器过了 2h 还没有收到客户的信息，它就发送探测报文段。若发送了 10 个探测报文段（每一个相隔 75s）还没有响应，就假定客户出了故障，这时就终止该连接。

时间等待计时器：

当 TCP 关闭一个连接时，它并不认为这个连接马上就真正地关闭了。在时间等待期间，连接还处于一种中间过渡状态。这就可以使重复的 FIN（终止）报文段（如果有的话）可以到达目的站，从而可将其丢弃。这个计时器的值通常设置为一个报文段的寿命期待值的两倍。

13. 假定在一个互联网中，所有的链路的传输都不出现差错，所有的结点也都不会发生故障。试问在这种情况下，TCP 的“可靠交付”功能是否就是多余的？

解析：不是多余的。TCP 的“可靠交付”功能在互联网中起着至关重要的作用。

因为实现主机到主机通信的 IP 协议是无连接、不可靠的协议，并且每个 IP 数据报独立地选择路由，由此可能出现失序、丢失、重复的现象。这些问题的解决都必须依靠 TCP 的“可靠交付”功能才能保证在目的主机的目的进程接收到正确的报文。由此 TCP 的“可靠交付”功能是必不可少的。

14. 如果收到的报文段无差错，只是未按序号，则 TCP 对此未作明确规定，而是让 TCP 实现者自行确定。试讨论两种可能的方法的优劣：

（1）将不按序的报文段丢弃。

（2）先将不按序的报文段暂存于接收缓存内，待所缺序号的报文段收齐后再一起上交应用层。

解析：第 1 种方法将不按序的报文段丢弃，会引起被丢弃报文段的重复传送，增加对网络带宽的消耗，但由于用不着将该报文段暂存，可避免对接收方缓冲区的占用。

第 2 种方法先将不按序的报文段暂存于接收缓存内，待所缺序号的报文段收齐后再一起上交应用层进程；这样可以避免发送方对已经被接收方收到的不按序的报文段的重传，减少了对网络带宽的消耗，但增加了接收方缓存区的开销。

15. 为什么超时时间发生时 cwnd 被置为 1，而收到 3 个冗余 ACK 时 cwnd 减半？

解析：大家可以从这个角度考虑：超时事件发生和收到 3 个冗余 ACK，哪个意味着网络拥塞程度更严重？通过分析不难发现，在收到 3 个冗余 ACK 的情况，网络虽然拥塞，但是至少还有 ACK 报文段能够被正确交付。而当超时发生时，说明网络可能已经拥塞得连 ACK 报文段都传输不了了，发送方只能等待超时后重传数据。因此，超时发生时，网络拥塞更严重，那么发送方就应该最大限度地抑制数据发送量，所以 cwnd 置为 1；收到 3 个冗余 ACK 时，网络拥塞不是很严重，发送方稍微抑制一下发送的数据量即可，所以 cwnd 减半。

16. TCP 使用的是 GBN 还是选择重传呢？

解析：这是一个有必要弄清的问题。在前面讲过，TCP 使用累计确认，这看起来像是 GBN 的风格。但是，正确收到的失序的报文并不会被丢弃，而是缓存起来。并且发送冗余 ACK 指明期望收到的下一个报文段，这是 TCP 方式和 GBN 的显著区别。例如，A 发送 N 个报文段，其中第 $k(k<N)$ 个报文段丢失，其余 $N-1$ 个报文段正确地按序到达接收方 B。当使用 GBN 时，A 需要重传分组 k，以及所有后继分组 $k+1, k+2, \cdots, N$。相反，在实际中 TCP 却至多重传一个报文段，即失序报文段 k。另外，TCP 中提供一个 SACK（Selective

ACK)选项,也就是选择确认选项。当使用选择确认选项的时候,TCP 看起来就和 SR 非常相似了。

17. MSS 设置的太大或者太小会有什么影响?

解析:规定最大报文段 MSS 的大小并不是考虑到接收方的缓存可能放不下 TCP 报文段。实际上,MSS 与接收窗口没有关系。TCP 的报文段的数据部分,至少要加上 40 字节的首部(TCP 首部和 IP 首部各至少 20 字节),才能组装成一个 IP 数据报。若选择较小的 MSS 值,网络的利用率就很低。设想在极端情况下,当 TCP 报文段中只有 1 字节的数据时,在 IP 层传输的数据报的开销至少有 40 字节。这样,网络的利用率就不会超过 1/41。到了数据链路层还要加上一些开销,网络的利用率就更降低。但反过来,若 TCP 报文段很长,那么在 IP 层传输时有可能要分解成多个短数据报片,在目的端还要把收到的各数据报片装配成原来的 TCP 报文段。当传输有差错时,还要进行重传,这些都会使开销增大。

因此,MSS 应尽量大些,只要在 IP 层传输时不要再分片就行。由于 IP 数据报所经历的路径是动态变化的,在一条路径上确定的不需要分片的 MSS,如果改走另一条路径就可能需要进行分片。因此,最佳的 MSS 是很难确定的。MSS 的默认值为 536 字节,因此在互联网上的所有主机都能接收的 TCP 报文段长度是 536+20(TCP 固定首部长度)=556 字节。

5.9 Wireshark 实验:TCP 协议分析

1. TCP 数据包报文

基于 Wireshark 捕获的 TCP 数据包如图 5-36 所示。

TCP 数据包主要字段分析:

```
Transmission Control Protocol, Src Port: 80, Dst Port: 2911, Seq: 10141, Ack: 415, Len: 5
        # TCP 协议,源端口号为 80,目的端口号为 2911,序号 10141,确认号 415,长度 5
    Source Port: 80                                                    # 源端口号为 80
    Destination Port: 2911                                             # 目的端口号为 2911
    [Stream index: 11]                                                 # 流结点号 11
    [TCPSegment Len: 5]                                                # TCP 报文长度 5
    Sequence number: 10141     (relative sequence number)              # 序列号
    [Next sequence number: 10146     (relative sequence number)]       # 下一个序列号
    Acknowledgment number: 415     (relative ack number)               # 确认号
    0101 .... = Header Length: 20 bytes (5)                            # 首部长度
    Flags: 0x018 (PSH, ACK)                                            # 标志字段
        000..... = Reserved: Not set                                   # 保留位
        ...0 ..... = Nonce: Not set
        ....0..... = Congestion Window Reduced (CWR): Not set
        .....0..... = ECN-Echo: Not set
        ......0..... = Urgent: Not set                                # 紧急指针位
        .......1 .... = Acknowledgment: Set                            # 确认位
        ........1... = Push: Set                                       # 推送位
        .........0.. = Reset: Not set                                  # 重置位
        ..........0. = Syn: Not set                                    # 同步 SYN 位
```

```
    ..........0 = Fin: Not set                          # 终止 FIN 位
    [TCPFlags:·····AP···]
Window size value: 556                                  # 窗口大小
[Calculated window size: 71168]                         # 估计的窗口大小
[Window size scaling factor: 128]                       # 窗口大小缩放比例因素
Checksum: 0xaa91 [unverified]                           # 校验和
[Checksum Status: Unverified]
Urgent pointer: 0                                       # 紧急指针字段
```

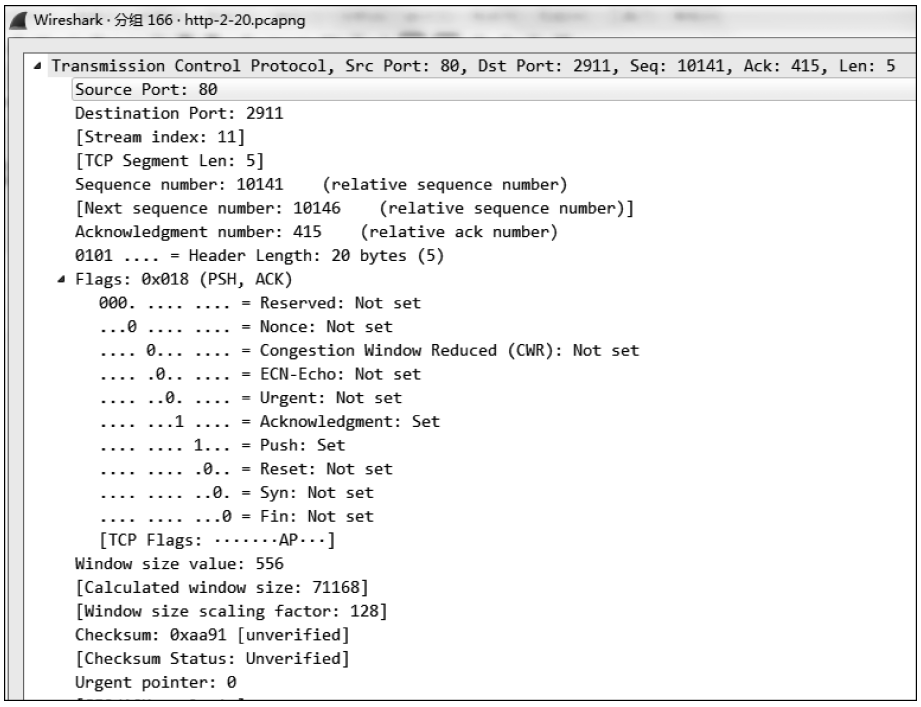

图 5-36 TCP 数据包信息

习题 5

5-1 试说明传输层的作用。网络层提供数据报或虚电路服务对上面的传输层有何影响?

5-2 接收端收到有差错的 UDP 用户数据报应如何处理?

5-3 应用程序使用面向连接的 TCP 和无连接的 IP 时,这种传输是面向连接的还是面向无连接的?

5-4 一个 TCP 报文段的数据部分最多有多少字节? 为什么? 如果用户要传送的数据的字节长度超过 TCP 报文段中的序号字段编出的最大序号,试问还能用 TCP 来传送吗?

5-5 主机 A 的 TCP 向主机 B 连续发送 3 个 TCP 报文段。第 1 个报文段的序号为 90,第 2 个报文段的序号为 120,第 3 个报文段的序号为 150。问:

(1) 第 1、第 2 个报文段中有多少数据?

(2) 假设第 2 个报文段丢失而其他两个报文段到达主机 B,那么在主机 B 发往主机 A

的确认报文中,确认号应该是多少?

5-6 使用 TCP 传送数据时,如果有一个确认报文丢失了,是否一定会引起与该确认报文段对应的数据的重传?试说明理由。

5-7 通信信道带宽为 1Gbps,端到端传播时延为 10ms。TCP 的发送窗口为 65 535 字节。试问:可能达到的最大吞吐量是多少?信道的利用率是多少?

5-8 为什么要使用 UDP?让用户进程直接发送原始的 IP 分组不就足够了吗?

5-9 为什么在 TCP 首部中有一个首部长度字段,而 UDP 的首部中就没有这个字段?

5-10 为什么 TCP 首部的最开始的 4 个字节是 TCP 的端口号?

5-11 在网络中,针对三个协议:Stop-and-Wait 协议、Go-Back-N 协议和 Selective-Repeat 协议,哪个是可以被网络使用且接收窗口的大小是一个报文的协议?

5-12 停止等待协议中,如果不使用编号是否可行,为什么?

5-13 在一个 UDP 用户数据报的数据字段为 9000 字节,使用以太网传送。试问应当划分为几个数据报片?说明每一个数据报片的数据字段长度和片偏移字段的值。

5-14 一个 UDP 用户数据报的首部的十六进制表示是 07 33 00 35 00 1C E3 18。试求源端口、目的端口、用户数据报的总长度、数据部分长度。这个数据报是从客户发送给服务器,还是从服务器发送给客户?使用 UDP 的这个服务器程序是什么?

5-15 TCP 的拥塞窗口 cwnd 大小与传输轮次 n 的关系如表 5-4 所示:

表 5-4 TCP 的拥塞窗口 cwnd 大小与传输轮次 n 的关系

n	1	2	3	4	5	6	7	8	9	10	11	12	13
cwnd	1	2	4	8	16	32	33	34	35	36	37	38	39
n	14	15	16	17	18	19	20	21	22	23	24	25	26
cwnd	40	41	42	21	22	23	24	25	26	1	2	4	8

(1) 指明 TCP 工作在慢开始阶段的时间间隔。

(2) 指明 TCP 工作在拥塞避免阶段的时间间隔。

(3) 在第 16 轮次和第 22 轮次之后发送方是通过收到三个重复的确认还是通过超时检测到丢失了报文段?

(4) 在第 1 轮次、第 18 轮次和第 24 轮次发送时,门限 ssthresh 分别被设置为多大?

(5) 在第几轮次发送出第 70 个报文?

(6) 假设在第 26 轮次之后收到了三个重复的确认,因而检测出了报文段的丢失,那么拥塞窗口 cwnd 的门限 ssthresh 应设置为多大?

5-16 请举例说明 TCP 连接为什么采用三次握手,两次不行?

5-17 请图示说明 TCP 连接的建立过程。

5-18 请图示说明 TCP 连接的释放过程。

5-19 已知第一次测得 TCP 的往返时间 RTT 是 30ms,接着收到了三个确认报文段,用它们测量出的往返时间样本 RTT 分别是:26ms,32ms 和 24ms。设 a=0.1,试计算每一次的新的加权平均往返时间值 SRTT。讨论所得出的结果。