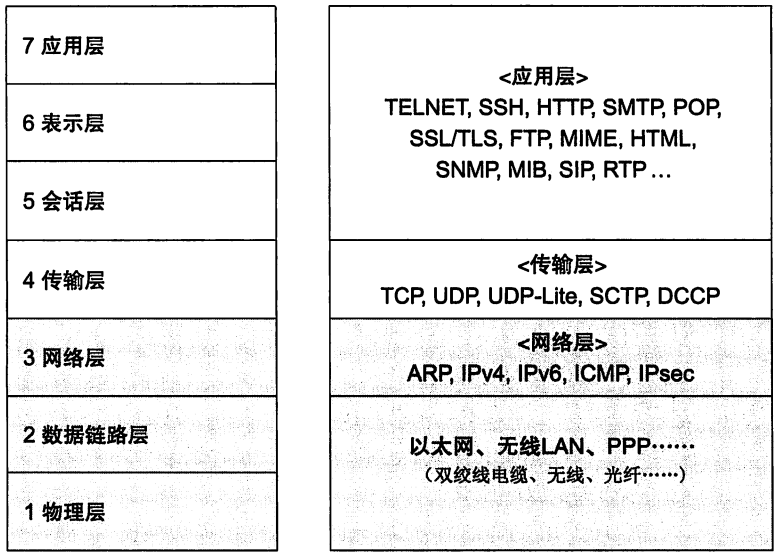
第6章

TCP与UDP

本章旨在介绍传输层的两个主要协议TCP（Transmission Control Protocol）与UDP（User Datagram Protocol）。



6.1 传输层的作用

TCP/IP中有两个具有代表性的传输层协议，它们分别是TCP和UDP。TCP提供可靠的通信传输，而UDP则常被用于让广播和细节控制交给应用的通信传输。总之，根据通信的具体特征，选择合适的传输层协议是非常重要的。

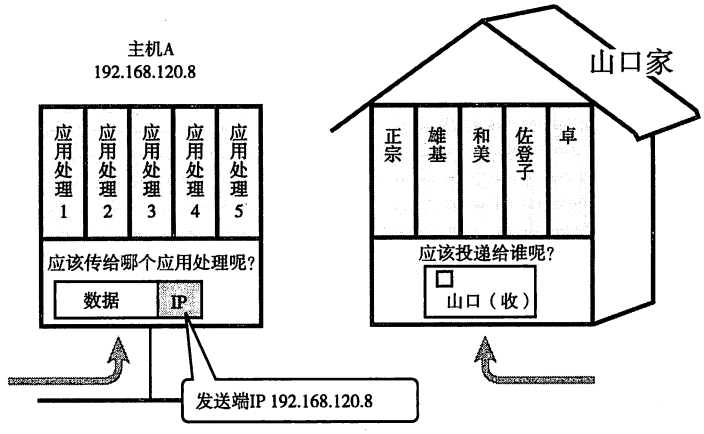
6.1.1 传输层定义

在第4章中也曾提到，IP首部中有一个协议字段，用来标识网络层（IP）的上一层所采用的是哪一种传输层协议。根据这个字段的协议号，就可以识别IP传输的数据部分究竟是TCP的内容，还是UDP的内容。

同样，传输层的TCP和UDP，为了识别自己所传输的数据部分究竟应该发给哪个应用，也设定了这样一个编号。

以包裹为例，邮递员（IP）根据收件人地址（目标IP地址）向目的地（计算机）投递包裹（IP数据报）。包裹到达目的地以后由对方（传输层协议）根据包裹信息判断最终的接收人（接收端应用程序）。

图6.1
一台计算机中运行着众多应用程序



通常一台计算机中运行着多个应用程序，因此有必要将其区分进行识别。

如果快递单上只写了家庭地址和姓氏，那该如何是好呢？你根本无法判断快递究竟应该投递给哪一位家庭成员。同样，如果收件人地址是学校或公司▼，而且也只写了一个姓氏，会给投递工作带来麻烦。因此，在日本的投递业务中都会要求寄件人写清楚接收人的全名。其实在中国，一个人的姓氏不像日本那样复姓居多▼，人们也通常不会仅以姓氏称呼一个人。但是也有一种特殊情况，那就是如果一个收件地址中有多个同名同姓的接收者该怎么办？此时，往往会通过追加电话号码来加以区分。

▼投递给公司或学校，还需要填写具体的部门或所属机构名称。

▼在中国邮政快递业务中通常也需要收件人的详细地址和全称。甚至在普通快递中可能还需要追加联系电话加以区分同名同姓的收件人。

在TCP/IP的通信当中也是如此，需要指定“姓氏”，即“应用程序”。而传输层必须指出这个具体的程序，为了实现这一功能，使用端口▼号这样一种识别码。根据端口号就可以识别在传输层上一层的应用层中所要进行处理的具体程序▼。

▼注意此处的端口与路由器、交换机等设备上指网卡的端口有所不同。

▼一个程序可以使用多个端口。

6.1.2 通信处理

再以邮递包裹为例，详细分析一下传输层的协议工作机制。

前面提到的“应用程序”其实就是用来进行 TCP/IP 应用协议的处理。因此，TCP/IP 中所要识别的“姓氏”就可以被理解为应用协议。

▼客户端（Client）具有客户的意思。在计算机网络中是提供服务和使用服务的一方。

▼服务端（Server）在计算机网络中则意味着提供服务的程序或计算机。

TCP/IP 的众多应用协议大多以客户端/服务端的形式运行。客户端▼类似于客户的意思，是请求的发起端。而服务端▼则表示提供服务的意思，是请求的处理端。另外，作为服务端的程序有必要提前启动，准备接收客户端的请求。否则即使有客户端的请求发过来，也无法做到相应的处理。

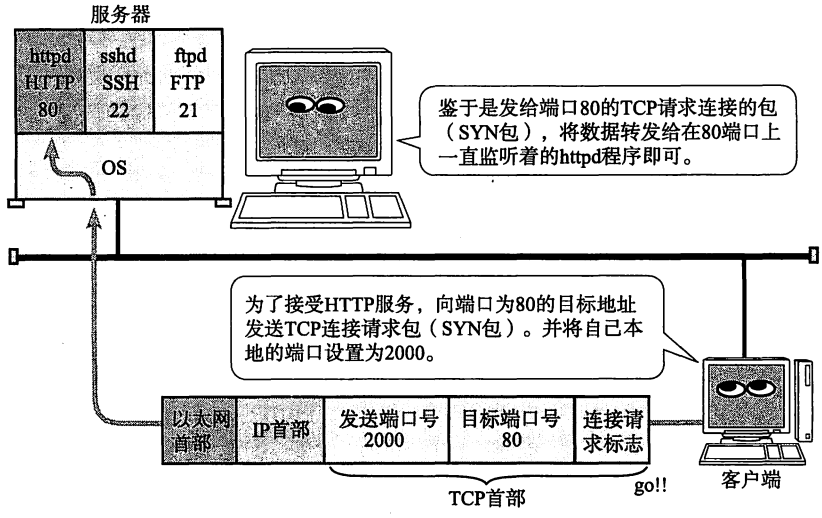


图 6.2 HTTP 连接请求

这些服务端程序在 UNIX 系统当中叫做守护进程。例如 HTTP 的服务端程序是 httpd（HTTP 守护进程），而 ssh 的服务端程序是 sshd（SSH 守护进程）。在 UNIX 中并不需要将这些守护进程逐个启动，而是启动一个可以代表它们接收客户端请求的 inetd（互联网守护进程）服务程序即可。它是一种超级守护进程。该超级守护进程收到客户端请求以后会创建（fork）新的进程并转换（exec）为 sshd 等各个守护进程。

确认一个请求究竟发给的是哪个服务端（守护进程），可以通过所收到数据包的目标端口号轻松识别。当收到 TCP 的建立连接请求时，如果目标端口为 22，则转给 sshd，如果是 80 则转给 httpd。然后，这些守护进程会继续对该连接上的通信传输进行处理。

传输协议 TCP、UDP 通过接收数据中的目标端口号识别目标处理程序。以图 6.2 为例，传输协议的数据将被传递给 HTTP、TELNET 以及 FTP 等应用层协议。

6.1.3 两种传输层协议 TCP 和 UDP

在 TCP/IP 中能够实现传输层功能的、具有代表性的协议是 TCP 和 UDP。

TCP

TCP 是面向连接的、可靠的流协议。流就是指不间断的数据结构，你可以把它想象成排水管道中的水流。当应用程序采用 TCP 发送消息时，虽然可以保证发

送的顺序，但还是犹如没有任何间隔的数据流发送给接收端▼。

TCP为提供可靠性传输，实行“顺序控制”或“重发控制”机制。此外还具备“流控制（流量控制）”、“拥塞控制”、提高网络利用率等众多功能。

▼例如，在发送端应用程序发送了10次100字节的消息，那么在接收端，应用程序有可能会收到一个1000字节连续不间断的数据。因此在TCP通信中，发送端应用可以在自己所要发送的消息中设置一个表示长度或间隔的字段信息。

UDP

UDP是不具有可靠性的数据报协议。细微的处理它会交给上层的应用去完成。在UDP的情况下，虽然可以确保发送消息的大小▼，却不能保证消息一定会到达。因此，应用有时会根据自己的需要进行重发处理。

▼例如，发送端应用程序发送一个100字节的消息，那么接收端应用程序也会以100字节为长度接收数据。UDP中，消息长度的数据也会发送到接收端，因此在发送的消息中不需要设置一个表示消息长度或间隔的字段信息。然而，UDP不具备可靠传输。所以，发送端发出去的消息在网络传输途中一旦丢失，接收端将收不到这个消息。

6.1.4　TCP与UDP区分

可能有人会认为，鉴于TCP是可靠的传输协议，那么它一定优于UDP。其实不然。TCP与UDP的优缺点无法简单地、绝对地去做比较。那么，对这两种协议应该如何加以区分使用呢？下面，我就对此问题做一简单说明。

TCP用于在传输层有必要实现可靠传输的情况。由于它是面向有连接并具备顺序控制、重发控制等机制的，所以它可以为应用提供可靠传输。

而在一方面，UDP主要用于那些对高速传输和实时性有较高要求的通信或广播通信。我们举一个通过IP电话进行通话的例子。如果使用TCP，数据在传送途中如果丢失会被重发，但这样无法流畅地传输通话人的声音，会导致无法进行正常交流。而采用UDP，它不会进行重发处理。从而也就不会有声音大幅度延迟到达的问题。即使有部分数据丢失，也只是会影响某一小部分的通话▼。此外，在多播与广播通信中也使用UDP而不是TCP。RIP（7.4节）、DHCP（5.5节）等基于广播的协议也要依赖于UDP。

▼在实时传送动画或声音时，途中一小部分网络的丢包可能会导致画面或声音的短暂停顿甚至出现混乱。但在实际使用当中，这一点干扰并无大碍。

因此，TCP和UDP应该根据应用的目的按需使用。

套接字（Socket）

应用在使用TCP或UDP时，会用到操作系统提供的类库。这种类库一般被称为API（Application Programming Interface，应用编程接口）。

使用TCP或UDP通信时，又会广泛使用到套接字（socket）的API。套接字原本是由BSD UNIX开发的，但是后被移植到了Windows的Winsock以及嵌入式操作系统中。

应用程序利用套接字，可以设置对端的IP地址、端口号，并实现数据的发送与接收。

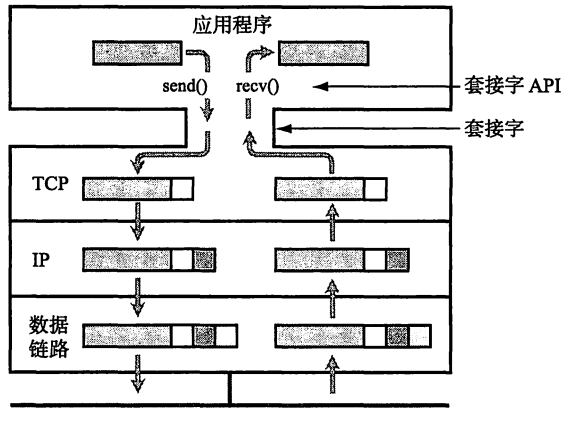


图6.3
套接字

6.2 端口号

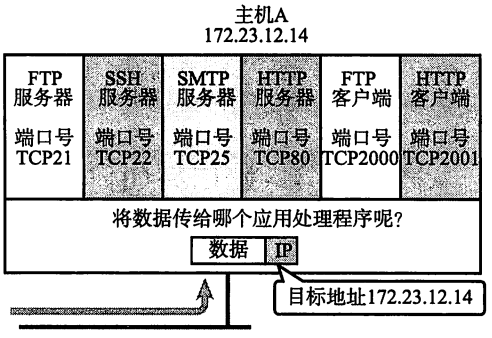
6.2.1 端口号定义

数据链路和 IP 中的地址，分别指的是 MAC 地址和 IP 地址。前者用来识别同一链路中不同的计算机，后者用来识别 TCP/IP 网络中互连的主机和路由器。在传输层中也有这种类似于地址的概念，那就是端口号。端口号用来识别同一台计算机中进行通信的不同应用程序。因此，它也被称为程序地址。

6.2.2 根据端口号识别应用

一台计算机上同时可以运行多个程序。例如接受 WWW 服务的 Web 浏览器、电邮客户端、远程登录用的 ssh 客户端等程序都可同时运行。传输层协议正是利用这些端口号识别本机中正在进行通信的应用程序，并准确地将数据传输。

图 6.4
根据端口号识别应用



6.2.3 通过 IP 地址、端口号、协议号进行通信识别

仅凭目标端口识别某一个通信是远远不够的。

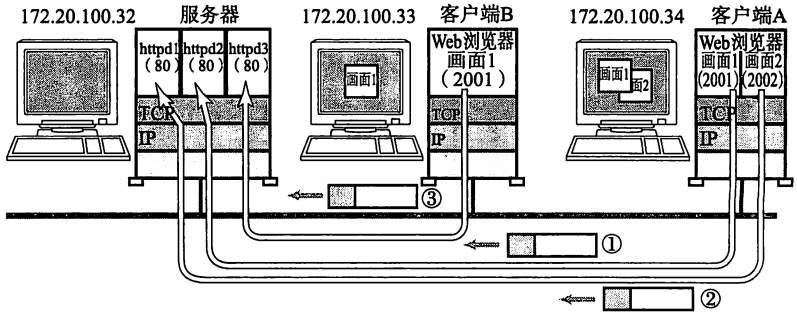
如图 6.5 所示，①和②的通信是在两台计算机上进行的。它们的目标端口号相同，都是 80。例如打开两个 Web 浏览器，同时访问两个服务器上不同的页面，就会在这个浏览器跟服务器之间产生类似前面的两个通信。在这种情况下也必须严格区分这两个通信。因此可以根据源端口号加以区分。

下图中③跟①的目标端口号和源端口号完全相同，但是它们各自的源 IP 地址不同。此外，还有一种情况上图中并未列出，那就是 IP 地址和端口全都一样，只是协议号（表示上层是 TCP 或 UDP 的一种编号）不同。这种情况下，也会认为是两个不同的通信。

因此，TCP/IP 或 UDP/IP 通信中通常采用 5 个信息来识别▼一个通信。它们是“源 IP 地址”、“目标 IP 地址”、“协议号”、“源端口号”、“目标端口号”。只要其中某一项不同，则被认为是其他通信。

▼这个信息可以在 Unix 或 Windows 系统中通过 netstat -n 命令显示。

图6.5
识别多个请求



	IP首部			TCP首部		
①	源IP地址 172.20.100.34	目标IP地址 172.20.100.32	TCP 6	源端口号 2001	目标端口号 80	数据
②	源IP地址 172.20.100.34	目标IP地址 172.20.100.32	TCP 6	源端口号 2002	目标端口号 80	数据
③	源IP地址 172.20.100.33	目标IP地址 172.20.100.32	TCP 6	源端口号 2001	目标端口号 80	数据

通过源IP地址、目标IP地址、协议号、源端口号和目标端口号这5个数字识别一个通信。

6.2.4　端口号如何确定

在实际进行通信时，要事先确定端口号。确定端口号的方法分为两种：

标准既定的端口号

这种方法也叫静态方法。它是指每个应用程序都有其指定的端口号。但并不是说可以随意使用任何一个端口号。每个端口号都有其对应的使用目的▼。

▼当然，这也不是说"绝对地只能有这样一个目的"。在更高级的网络应用中有时也会别作他用。

例如，HTTP、TELNET、FTP等广为使用的应用协议中所使用的端口号就是固定的。这些端口号也被称之为知名端口号（Well-Known Port Number）。表6.1与表6.2中就列出了TCP与UDP具有代表性的知名端口号。知名端口号一般由0到1023的数字分配而成。应用程序应该避免使用知名端口号进行既定目的之外的通信，以免产生冲突。

除知名端口号之外，还有一些端口号也被正式注册。它们分布在1024到49151的数字之间。不过，这些端口号可用于任何通信用途。关于知名端口号以及注册端口号的最新消息，请参考如下网址：

```
http://www.iana.org/assignments/port-numbers
```

时序分配法

第二种方法也叫时序（或动态的）分配法。此时，服务端有必要确定监听端口号，但是接受服务的客户端没必要确定端口号。

在这种方法下，客户端应用程序可以完全不用自己设置端口号，而全权交给操作系统进行分配。操作系统可以为每个应用程序分配互不冲突的端口号。例如，每需要一个新的端口号时，就在之前分配号码的基础上加1。这样，操作系统就可以动态地管理端口号了。

根据这种动态分配端口号的机制，即使是同一个客户端程序发起的多个 TCP 连接，识别这些通信连接的 5 部分数字也不会全部相同。

动态分配的端口号取值范围在 49152 到 65535 之间[▼]。

▼在较老的系统中有时会依次使用 1024 以上空闲的端口。

6.2.5 端口号与协议

端口号由其使用的传输层协议决定。因此，不同的传输协议可以使用相同的端口号。例如，TCP 与 UDP 使用同一个端口号，但使用目的各不相同。这是因为端口号上的处理是根据每个传输协议的不同而进行的。

数据到达 IP 层后，会先检查 IP 首部中的协议号，再传给相应协议的模块。如果是 TCP 则传给 TCP 模块、如果是 UDP 则传给 UDP 模块去做端口号的处理。即使是同一个端口号，由于传输协议是各自独立地进行处理，因此相互之间不会受到影响。

此外，那些知名端口号与传输层协议并无关系，只要端口一致都将分配同一种程序进行处理。例如，53 号端口在 TCP 与 UDP 中都用于 DNS[▼] 服务，而 80 端口用于 HTTP 通信。从目前来看，由于 HTTP 通信必须使用 TCP，因此 UDP 的 80 端口并未投入使用。但是将来，如果 HTTP 协议的实现也开始应对 UDP 协议以及应用协议被相应扩展的情况下，就可以原样使用与 TCP 保持相同的 80 端口号了。

▼由域名确定 IP 地址时所用的协议。更多细节请参考 5.2 节。

表 6.1 TCP 具有代表性的知名端口号

端口号	服务名	内　容
1	tcpmux	TCP Port Service Multiplexer
7	echo	Echo
9	discard	Discard
11	systat	Active Users
13	daytime	Daytime
17	qotd	Quote of the Day
19	chargen	Character Generator
20	ftp-data	File Transfer [Default Data]
21	ftp	File Transfer [Control]
22	ssh	SSH Remote Login Protocol
23	telnet	Telnet
25	smtp	Simple Mail Transfer Protocol
43	nicname	Who Is
53	domain	Domain Name Server
70	gopher	Gopher
79	finger	Finger
80	http (www, www-http)	World Wide Web HTTP
95	supdup	SUP DUP
101	hostname	NIC Host Name Server

（续）

端口号	服务名	内　容
102	iso-tsap	ISO-TSAP
109	pop2	Post Office Protocol - Version 2
110	pop3	Post Office Protocol - Version 3
111	sunrpc	SUN Remote Procedure Call
113	auth（ident）	Authentication Service
117	uucp-path	UUCP Path Service
119	nntp	Network News Transfer Protocol
123	ntp	Network Time Protocol
139	netbios-ssn	NETBIOS Session Service（SAMBA）
143	imap	Internel Message Access Protocol v2，v4
163	cmip-man	CMIP/TCP Manager
164	cmip-agent	CMIP/TCP Agent
179	bgp	Border Gateway Protocol
194	irc	Internet Relay Chat Protocol
220	Imap3	Interactive Mail Access Protocol v3
389	ldap	Lightweight Directory Access Protocol
434	mobileip-agent	Mobile IP Agent
443	https	http protocol over TLS/SSL
515	printer	Printer spooler（lpr）
587	submission	Message Submission
636	ldaps	ldap protocol over TLS/SSL
989	ftps-data	ftp protocol，data，over TLS/SSL
990	ftps	ftp protocol，control，over TLS/SSL
993	imaps	imap4 protocol over TLS/SSL
995	pop3s	pop3 protocol over TLS/SSL

表6.2
UDP具有代表性的知名端口号

端口号	服务名	内　容
7	echo	Echo
9	discard	Discard
11	systat	Active Users
13	daytime	Daytime
17	qotd	Quote of the Day
19	chargen	Character Generator

（续）

端口号	服务名	内　容
49	tacacs	Login Host Protocol（TACACS）
53	domain	Domain Name Server
67	bootps	Bootstrap Protocol Server（DHCP）
68	bootpc	Bootstrap Protocol Client（DHCP）
69	tftp	Trivial File Transfer Protocol
111	sunrpc	SUN Remote Procedure Call
123	ntp	Network Time Protocol
137	netbios-ns	NETBIOS Name Service（SAMBA）
138	netbios-dgm	NETBIOS Datagram Service（SAMBA）
161	snmp	SNMP
162	snmptrap	SNMP TRAP
177	xdmcp	X Display Manager Control Protocol
201	at-rtmp	AppleTalk Routing Maintenance
202	at-nbp	AppleTalk Name Binding
204	at-echo	AppleTalk Echo
206	at-zis	AppleTalk Zone Information
213	ipx	IPX
434	mobileip-agent	Mobile IP Agent
520	router	RIP
546	dhcpv6-client	DHCPv6 Client
547	dhcpv6-server	DHCPv6 Server

6.3 UDP

UDP的特点及其目的

UDP是User Datagram Protocol的缩写。

UDP不提供复杂的控制机制，利用IP提供面向无连接的通信服务。并且它是将应用程序发来的数据在收到的那一刻，立即按照原样发送到网络上的一种机制。

即使是出现网络拥堵的情况下，UDP也无法进行流量控制等避免网络拥塞的行为。此外，传输途中即使出现丢包，UDP也不负责重发。甚至当出现包的到达顺序乱掉时也没有纠正的功能。如果需要这些细节控制，那么不得不交由采用UDP的应用程序去处理▼。UDP有点类似于用户说什么听什么的机制，但是需要用户充分考虑好上层协议类型并制作相应的应用程序。因此，也可以说，UDP按照“制作程序的那些用户的指示行事”。

▼由于互联网中没有一个能够控制全局的机制，因此通过互联网发送大量数据时，各个节点将力争不给其他用户添麻烦。为此，拥塞控制成为必要的功能（拥塞控制往往不是因为自身需要）。然而，当不想实现拥塞控制时，有必要使用TCP。

由于UDP面向无连接，它可以随时发送数据。再加上UDP本身的处理既简单又高效，因此经常用于以下几个方面：

- 包总量较少的通信（DNS、SNMP等）
- 视频、音频等多媒体通信（即时通信）
- 限定于LAN等特定网络中的应用通信
- 广播通信（广播、多播）

用户与程序员

此处所使用的“用户”并不单单指“互联网的使用者”。曾经它也表示为那些编写程序的程序员。因此，UDP的“用户”（User）在现在看来其实就相当于程序员。也就是说，认为UDP是按照程序员的编程思路在传送数据报也情有可原▼。

▼与之相比，由于TCP拥有各式各样的控制机制，所以它在发送数据时未必按照程序员的编程思路进行。

6.4 TCP

UDP 是一种没有复杂控制，提供面向无连接通信服务的一种协议。换句话说，它将部分控制转移给应用程序去处理，自己却只提供作为传输层协议的最基本功能。

与 UDP 不同，TCP 则“人如其名”，可以说是对“传输、发送、通信”进行“控制”的“协议”。

TCP 与 UDP 的区别相当大。它充分地实现了数据传输时各种控制功能，可以进行丢包时的重发控制，还可以对次序乱掉的分包进行顺序控制。而这些在 UDP 中都没有。此外，TCP 作为一种面向有连接的协议，只有在确认通信对端存在时才会发送数据，从而可以控制通信流量的浪费▼。

▼由于 UDP 没有连接控制，所以即使对端从一开始就不存在或中途退出网络，数据包还是能够发送出去。（当 ICMP 错误返回时，有时也实现了不再发送的机制。）

根据 TCP 的这些机制，在 IP 这种无连接的网络上也能够实现高可靠性的通信。

连接

连接是指各种设备、线路，或网络中进行通信的两个应用程序为了相互传递消息而专有的、虚拟的通信线路，也叫做虚拟电路。

一旦建立了连接，进行通信的应用程序只使用这个虚拟的通信线路发送和接收数据，就可以保障信息的传输。应用程序可以不用顾虑提供尽职服务的 IP 网络上可能发生的各种问题，依然可以转发数据。TCP 则负责控制连接的建立、断开、保持等管理工作。

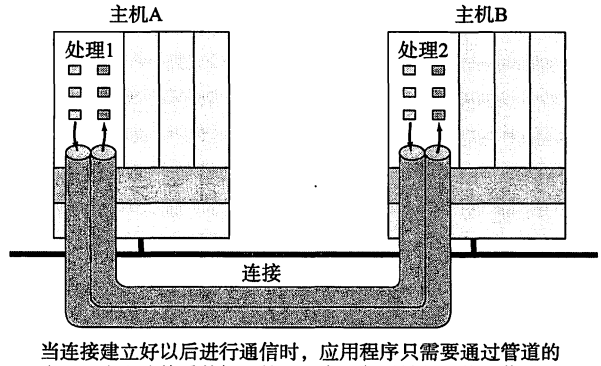


图 6.6 连接

当连接建立好以后进行通信时，应用程序只需要通过管道的出入口发送或接受数据，就可以实现与对端的网络通信。

6.4.1　TCP的特点及其目的

为了通过IP数据报实现可靠性传输，需要考虑很多事情，例如数据的破坏、丢包、重复以及分片顺序混乱等问题。如不能解决这些问题，也就无从谈起可靠传输。

TCP通过检验和、序列号、确认应答、重发控制、连接管理以及窗口控制等机制实现可靠性传输。

6.4.2　通过序列号与确认应答提高可靠性

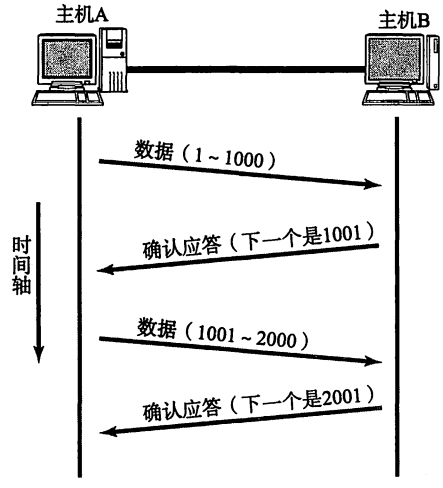
在TCP中，当发送端的数据到达接收主机时，接收端主机会返回一个已收到消息的通知。这个消息叫做确认应答（ACK▼）。

▼ACK（Positive Acknowledgement）意指已经接收。

通常，两个人对话时，在谈话的停顿处可以点头或询问以确认谈话内容。如果对方迟迟没有任何反馈，说话的一方还可以再重复一遍以保证对方确实听到。因此，对方是否理解了此次对话内容，对方是否完全听到了对话的内容，都要靠对方的反应来判断。网络中的“确认应答”就是类似这样的一个概念。当对方听懂对话内容时会说：“嗯”，这就相当于返回了一个确认应答（ACK）。而当对方没有理解对话内容或没有听清时会问一句“咦?”这好比一个否定确认应答（NACK▼）。

▼NACK（Negative Acknowledgement）

图6.7
正常的数据传输

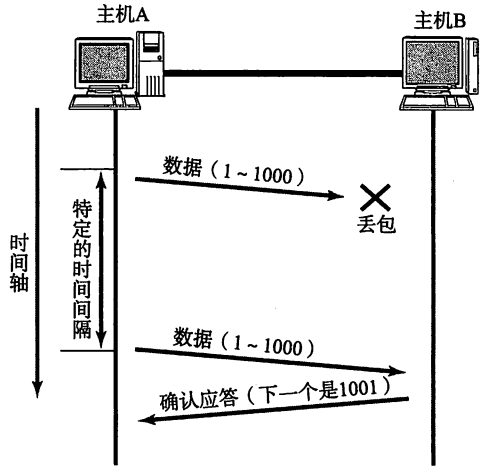


当数据从主机A发送到主机B时，主机B会返回给主机A一个确认应答。

TCP通过肯定的确认应答（ACK）实现可靠的数据传输。当发送端将数据发出之后会等待对端的确认应答。如果有确认应答，说明数据已经成功到达对端。反之，则数据丢失的可能性很大。

如图6.8所示，在一定时间内没有等到确认应答，发送端就可以认为数据已经丢失，并进行重发。由此，即使产生了丢包，仍然能够保证数据能够到达对端，实现可靠传输。

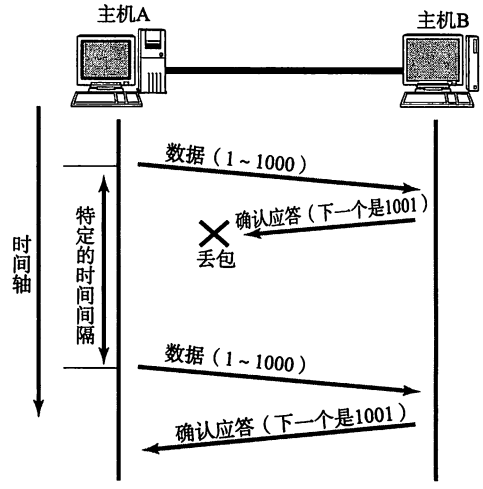
图 6.8
数据包丢失的情况



当数据由主机A发出后如果因网络拥堵等原因丢失的话，该数据将无法到达主机B。此时，如果主机A在一个特定时间间隔内都未收到主机B发来的确认应答，将会对此数据进行重发。

未收到确认应答并不意味着数据一定丢失。也有可能是数据对方已经收到，只是返回的确认应答在途中丢失。这种情况也会导致发送端因没有收到确认应答，而认为数据没有到达目的地，从而进行重新发送。如图 6.9 所示。

图 6.9
确认应答丢失的情况



由主机B返回的确认应答，因网络拥堵等原因在传送的途中丢失，没有到达主机A。主机A会等待一段时间，若在特定的时间间隔内始终未能收到这个确认应答，主机A会对此数据进行重发。此时，主机B将第二次发送已接收此数据的确认应答。由于主机B其实已经收到过1~1000的数据，当再有相同数据送达时它会放弃。

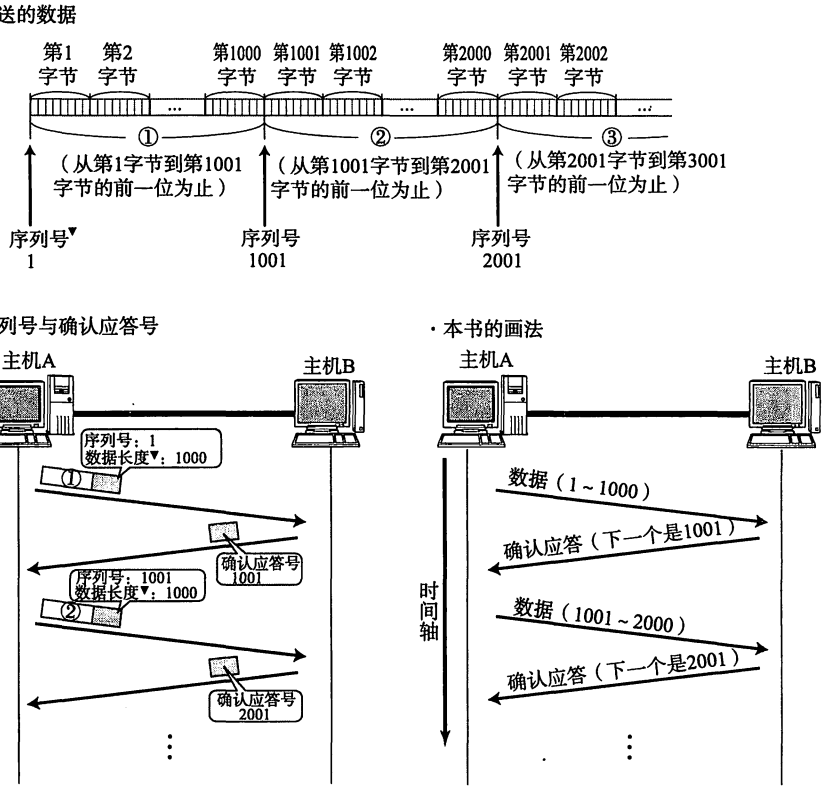
此外，也有可能因为一些其他原因导致确认应答延迟到达，在源主机重发数据以后才到达的情况也履见不鲜。此时，源发送主机只要按照机制重发数据即可。但是对于目标主机来说，这简直是一种“灾难”。它会反复收到相同的数据。而为了对上层应用提供可靠的传输，必须得放弃重复的数据包。为此，就必须引入

一种机制，它能够识别是否已经接收数据，又能够判断是否需要接收。

上述这些确认应答处理、重发控制以及重复控制等功能都可以通过序列号实现。序列号是按顺序给发送数据的每一个字节（8位字节）都标上号码的编号▼。接收端查询接收数据TCP首部中的序列号和数据的长度，将自己下一步应该接收的序号作为确认应答返送回去。就这样，通过序列号和确认应答号，TCP可以实现可靠传输。

▼序列号的初始值并非为0。而是在建立连接以后由随机数生成。而后面的计算则是对每一字节加一。

图6.10 发送数据



▼序列号（或确认应答号）也指字节与字节之间的分隔。

▼TCP的数据长度并未写入TCP首部。实际通信中求得TCP包的长度的计算公式是：IP首部中的数据包长度-IP首部长度 TCP首部长度。

* 1到1000的记录方法是指从第1字节开始到第1000字节全部包含的意思。

* 从本图开始，为了易于阅读，书中多处图中序列号都从1开始，MSS▼都为1000。

▼关于MSS（报文最大长度）的更多细节请参考6.4.5节。

6.4.3　重发超时如何确定

重发超时是指在重发数据之前，等待确认应答到来的那个特定时间间隔。如果超过了这个时间仍未收到确认应答，发送端将进行数据重发。那么这个重发超时的具体时间长度又是如何确定的呢？

最理想的是，找到一个最小时间，它能保证“确认应答一定能在这个时间内返回”。然而这个时间长短随着数据包途径的网络环境的不同而有所变化。例如在高速的LAN中时间相对较短，而在长距离的通信当中应该比LAN要长一些。即使是在同一个网络中，根据不同时段的网络拥堵程度时间的长短也会发生变化。

TCP要求不论处在何种网络环境下都要提供高性能通信，并且无论网络拥堵情况发生何种变化，都必须保持这一特性。为此，它在每次发包时都会计算往返时间▼及其偏差▼。将这个往返时间和偏差相加重发超时的时间，就是比这个总和

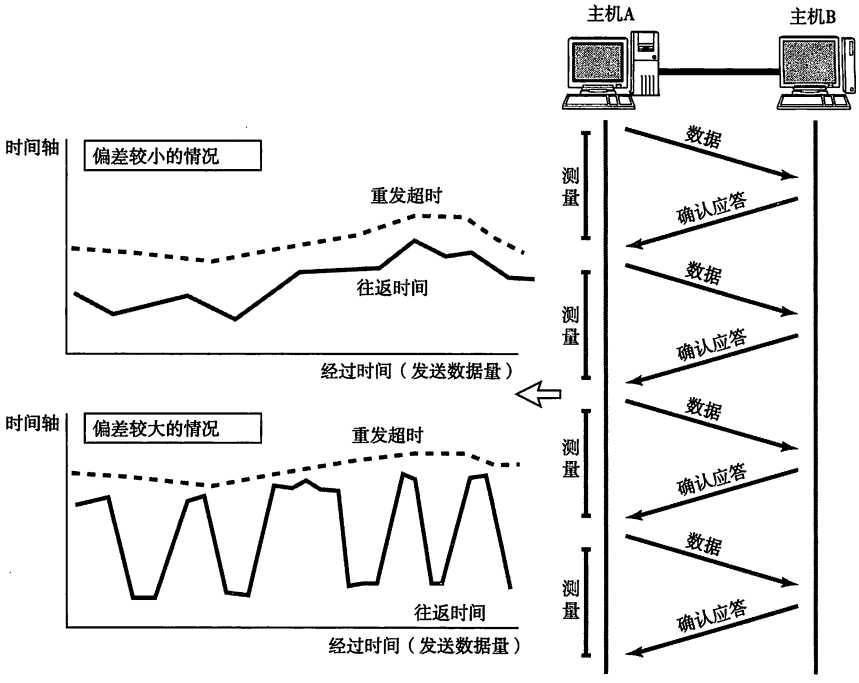
▼Round Trip Time也叫RTT。是指报文段的往返时间。

▼RTT时间波动的值、方差。有时也叫抖动。

要稍大一点的值。

重发超时的计算既要考虑往返时间又要考虑偏差是有其原因。如图 6.11 所示，根据网络环境的不同往返时间可能会产生大幅度的摇摆，之所以发生这种情况是因为数据包的分段是经过不同线路到达的。TCP/IP 的目的是即使在这种环境下也要进行控制，尽量不要浪费网络流量。

图 6.11
往返时间的计算与重发超时的时间推移



在 BSD 的 Unix 以及 Windows 系统中，超时都以 0.5 秒为单位进行控制，因此重发超时都是 0.5 秒的整数倍▼。不过，由于最初的数据包还不知道往返时间，所以其重发超时一般设置为 6 秒左右。

▼偏差的最小值也是 0.5 秒。因此最小的重发时间至少是 1 秒。

数据被重发之后若还是收不到确认应答，则进行再次发送。此时，等待确认应答的时间将会以 2 倍、4 倍的指数函数延长。

此外，数据也不会被无限、反复地重发。达到一定重发次数之后，如果仍没有任何确认应答返回，就会判断为网络或对端主机发生了异常，强制关闭连接。并且通知应用通信异常强行终止。

6.4.4 连接管理

TCP 提供面向有连接的通信传输。面向有连接是指在数据通信开始之前先做好通信两端之间的准备工作。

UDP是一种面向无连接的通信协议，因此不检查对端是否可以通信，直接将UDP包发送出去。TCP与此相反，它会在数据通信之前，通过TCP首部发送一个SYN包作为建立连接的请求等待确认应答▼。如果对端发来确认应答，则认为可以进行数据通信。如果对端的确认应答未能到达，就不会进行数据通信。此外，在通信结束时会进行断开连接的处理（FIN包）。

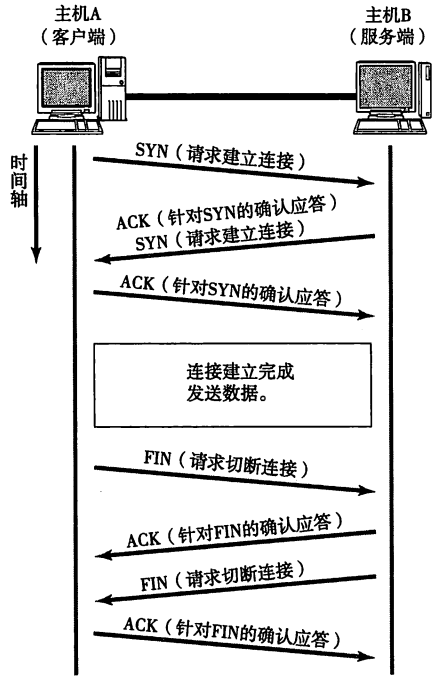
▼TCP中发送第一个SYN包的一方叫做客户端，接收这个的一方叫做服务端。

可以使用TCP首部用于控制的字段来管理TCP连接▼。一个连接的建立与断开，正常过程至少需要来回发送7个包才能完成▼。

▼也叫控制域。更多细节请参考6.7节。

▼建立一个TCP连接需要发送3个包。这个过程也称作“三次握手”。

图6.12
TCP连接的建立与断开



6.4.5　TCP以段为单位发送数据

在建立TCP连接的同时，也可以确定发送数据包的单位，我们也可以称其为“最大消息长度”（MSS：Maximum Segment Size）。最理想的情况是，最大消息长度正好是IP中不会被分片处理的最大数据长度。

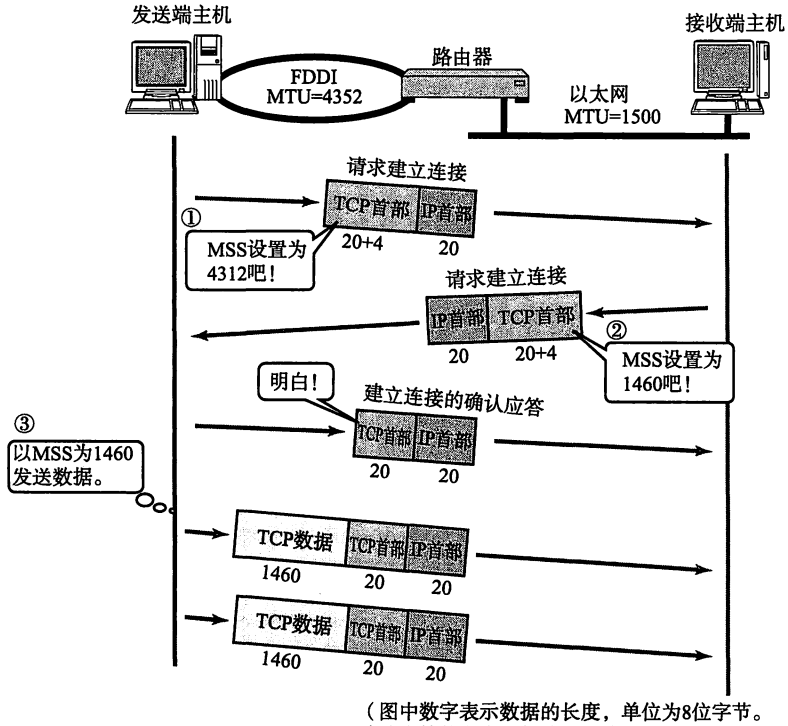
TCP在传送大量数据时，是以MSS的大小将数据进行分割发送。进行重发时也是以MSS为单位。

▼为附加MSS选项，TCP首部将不再是20字节，而是4字节的整数倍。如图6.13所示的+4。

MSS是在三次握手的时候，在两端主机之间被计算得出。两端的主机在发出建立连接的请求时，会在TCP首部中写入MSS选项，告诉对方自己的接口能够适应的MSS的大小▼。然后会在两者之间选择一个较小的值投入使用▼。

▼在建立连接时，如果某一方的MSS选项被省略，可以选为IP包的长度不超过576字节的值（IP首部20字节，TCP首部20字节，MSS 536字节）。

图 6.13
接入以太网主机与接入FDDI主机之间通信的情况



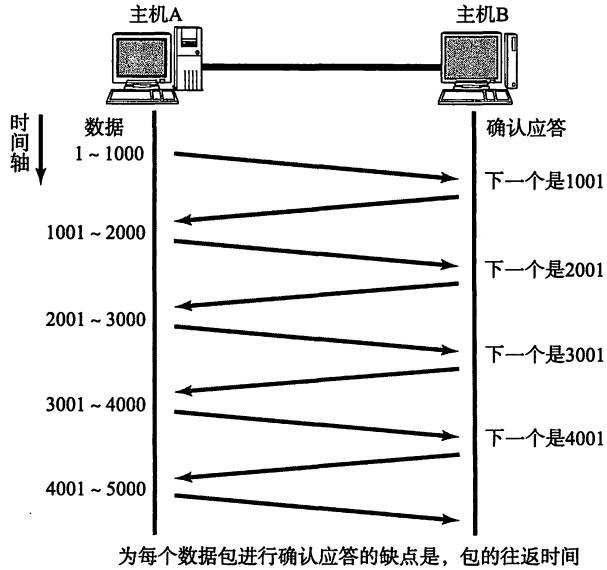
（图中数字表示数据的长度，单位为8位字节。
确认应答的段有一部分已省略。）

①②通过建立连接的SYN包相互通知对方网络接口的MSS值。
③在两者之间选一个较小的作为MSS的值，发送数据。

6.4.6 利用窗口控制提高速度

TCP以1个段为单位，每发一个段进行一次确认应答的处理，如图6.14。这样的传输方式有一个缺点。那就是，包的往返时间越长通信性能就越低。

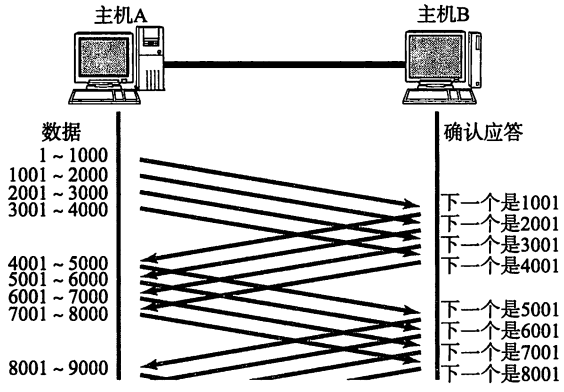
图 6.14
按数据包进行确认应答



为每个数据包进行确认应答的缺点是，包的往返时间越长，网络的吞吐量会越差。

为解决这个问题，TCP引入了窗口这个概念。即使在往返时间较长的情况下，它也能控制网络性能的下降。图6.15所示，确认应答不再是以每个分段，而是以更大的单位进行确认时，转发时间将会被大幅度的缩短。也就是说，发送端主机，在发送了一个段以后不必要一直等待确认应答，而是继续发送。

图6.15
用滑动窗口方式并行处理



·根据窗口为4000字节时返回的确认应答，下一步就发送比这个值还要大4000个序列号为止的数据。这跟前面每个段接收确认应答以后再发送另一个新段的情况相比，即使往返时间变长也不会影响网络的吞吐量。

窗口大小就是指无需等待确认应答而可以继续发送数据的最大值。图6.15中，窗口大小为4个段。

▼缓冲区（Buffer）在此处表示临时保存收发数据的场所。通常是在计算机内存中开辟的一部分空间。

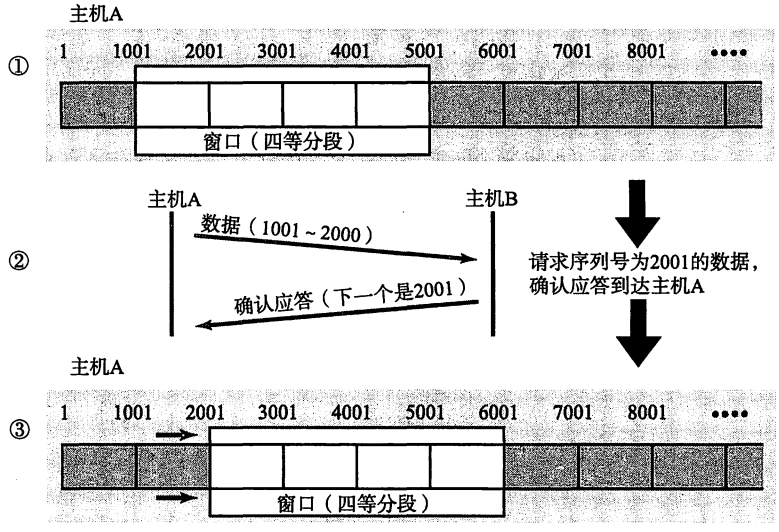
这个机制实现了使用大量的缓冲区▼，通过对多个段同时进行确认应答的功能。

如图6.16所示，发送数据中高亮圈起的部分正是前面所提到的窗口。在这个窗口内的数据即便没有收到确认应答也可以发送出去。此外，从该窗口中能看到的数据因其某种数据已在传输中丢失，所以发送端才能收到确认应答，这种情况也需进行重发。为此，发送端主机在等到确认应答返回之前，必须在缓冲区中保留这部分数据。

在滑动窗口以外的部分包括尚未发送的数据以及已经确认对端已收到的数据。当数据发出后若如期收到确认应答就可以不用再进行重发，此时数据就可以从缓存区清除。

收到确认应答的情况下，将窗口滑动到确认应答中的序列号的位置。这样可以顺序地将多个段同时发送提高通信性能。这种机制也被称为滑动窗口控制。

图 6.16
滑动窗口方式



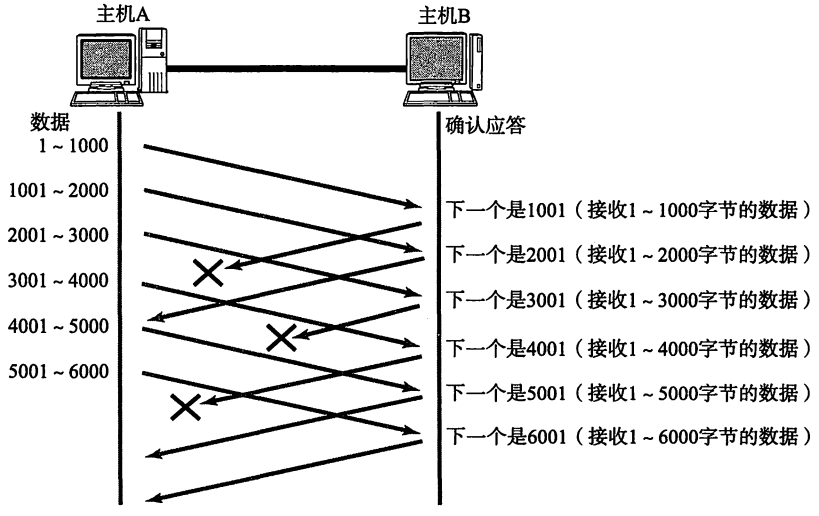
在①的状态下，如果收到一个请求序列号为2001的确认应答，那么2001之前的数据就没有必要进行重发，这部分的数据可以被过滤掉，滑动窗口成为③的样子。（这是在1个段为1000个字节，窗口为4个段的情况）

6.4.7 窗口控制与重发控制

在使用窗口控制中，如果出现段丢失该怎么办？

首先，我们先考虑确认应答未能返回的情况。在这种情况下，数据已经到达对端，是不需要再进行重发的。然而，在没有使用窗口控制的时候，没有收到确认应答的数据都会被重发。而使用了窗口控制，就如图 6.17 所示，某些确认应答即便丢失也无需重发。

图 6.17
没有确认应答也不受影响



窗口在一定程度上较大时，即使有少部分的确认应答丢失也不会进行数据重发。可以通过下一个确认应答进行确认。

其次，我们来考虑一下某个报文段丢失的情况。如图 6.18 所示，接收主机如

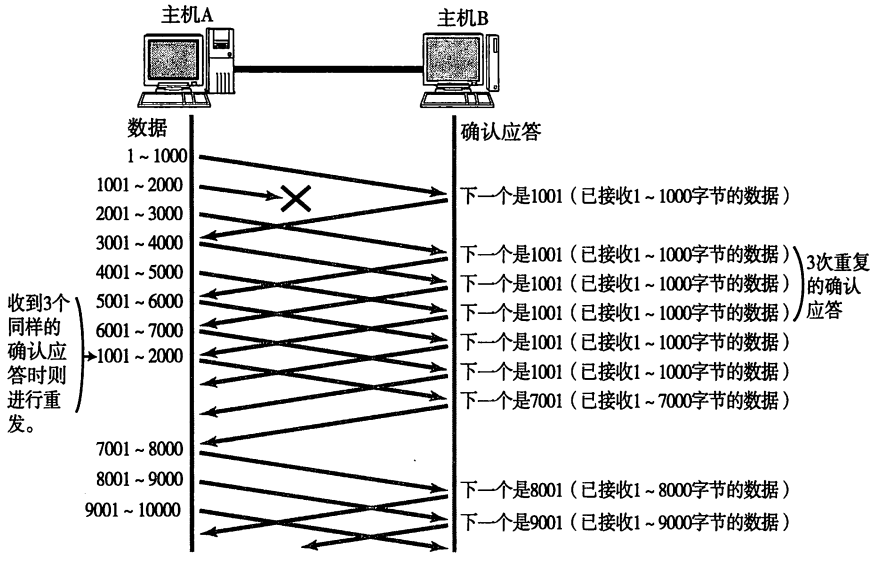
果收到一个自己应该接收的序号以外的数据时，会针对当前为止收到数据返回确认应答▼。

▼不过即使接收端主机收到的包序号并不连续，也不会将数据丢弃而是暂时保存至缓冲区中。

如图6.18所示。当某一报文段丢失后，发送端会一直收到序号为1001的确认应答，这个确认应答好像在提醒发送端“我想接收的是从1001开始的数据”。因此，在窗口比较大，又出现报文段丢失的情况下，同一个序号的确认应答将会被重复不断地返回。而发送端主机如果连续3次收到同一个确认应答▼，就会将其所对应的数据进行重发。这种机制比之前提到的超时管理更加高效，因此也被称作高速重发控制。

▼之所以连续收到3次而不是两次的理由是因为，即使数据段的序号被替换两次也不会触发重发机制。

图6.18
高速重发控制（Fast Retransmission）



接收端在没有收到自己所期望序号的数据时，会对之前收到的数据进行确认应答。发送端则一旦收到某个确认应答后，又连续3次收到同样的确认应答，则认为数据段已经丢失，需要进行重发。这种机制比起超时机制可以提供更为快速的重发服务。

6.4.8　流控制

发送端根据自己的实际情况发送数据。但是，接收端可能收到的是一个毫无关系的数据包又可能会在处理其他问题上花费一些时间。因此在为这个数据包做其他处理时会耗费一些时间，甚至在高负荷的情况下无法接收任何数据。如此一来，如果接收端将本应该接收的数据丢弃的话，就又会触发重发机制，从而导致网络流量的无端浪费。

为了防止这种现象的发生，TCP提供一种机制可以让发送端根据接收端的实际接收能力控制发送的数据量。这就是所谓的流控制。它的具体操作是，接收端主机向发送端主机通知自己可以接收数据的大小，于是发送端会发送不超过这个限度的数据。该大小限度就被称作窗口大小。在前面6.4.6节中所介绍的窗口大小的值就是由接收端主机决定的。

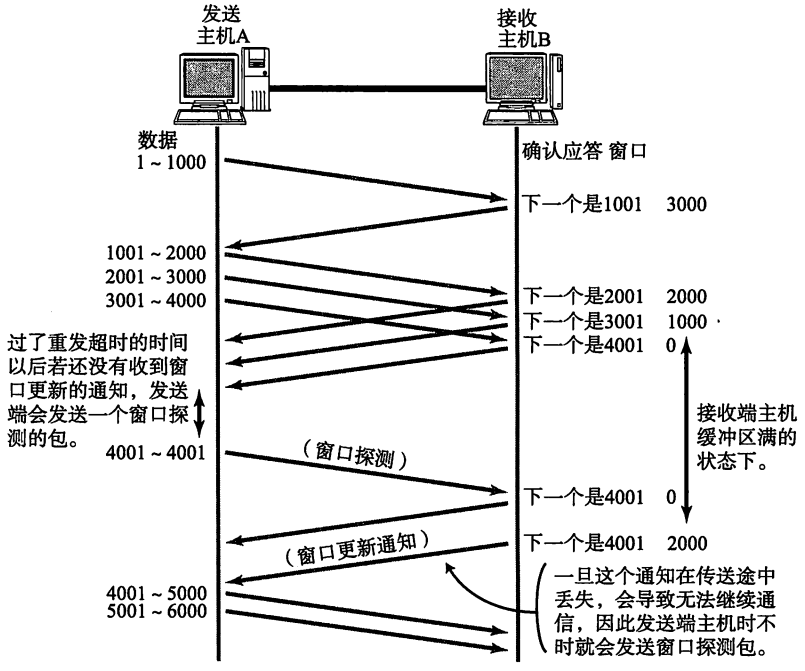
TCP首部中，专门有一个字段用来通知窗口大小。接收主机将自己可以接收的缓冲区大小放入这个字段中通知给发送端。这个字段的值越大，说明网络的吞吐量越高。

不过，接收端的这个缓冲区一旦面临数据溢出时，窗口大小的值也会随之被

设置为一个更小的值通知给发送端，从而控制数据发送量。也就是说，发送端主机会根据接收端主机的指示，对发送数据的量进行控制。这也就形成了一个完整的 TCP 流控制（流量控制）。

图 6.19 为根据窗口大小控制流量过程的示例。

图 6.19
流控制



发送端主机根据接收端主机的窗口大小通知进行流量控制。由此也可以防止发送端主机一次发送过大数据导致接收端主机无法处理的情况发生。

如图 6.19 所示，当接收端收到从 3001 号开始的数据段后其缓冲区即满，不得不暂时停止接收数据。之后，在收到发送窗口更新通知后通信才得以继续进行。如果这个窗口的更新通知在传送途中丢失，可能会导致无法继续通信。为避免此类问题的发生，发送端主机会时不时的发送一个叫做窗口探测的数据段，此数据段仅含一个字节以获取最新的窗口大小信息。

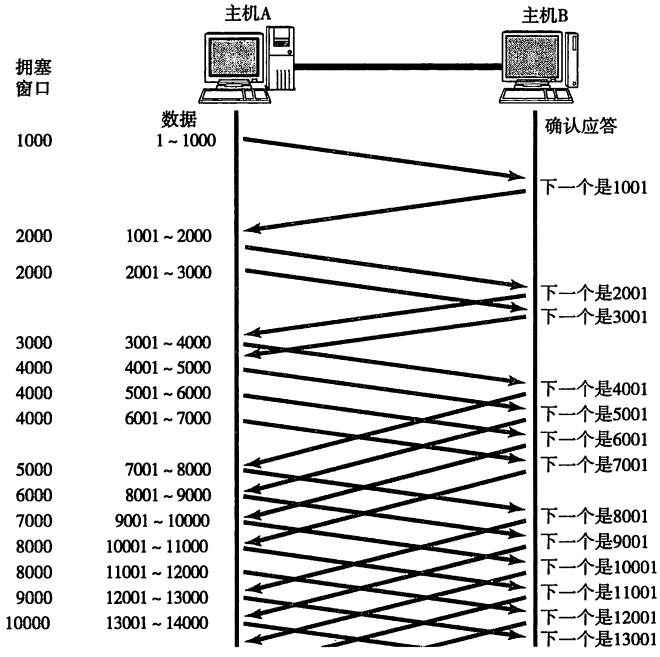
6.4.9 拥塞控制

有了 TCP 的窗口控制，收发主机之间即使不再以一个数据段为单位发送确认应答，也能够连续发送大量数据包。然而，如果在通信刚开始时就发送大量数据，也可能会引发其他问题。

一般来说，计算机网络都处在一个共享的环境。因此也有可能会因为其他主机之间的通信使得网络拥堵。在网络出现拥堵时，如果突然发送一个较大量的数据，极有可能会导致整个网络的瘫痪。

TCP 为了防止该问题的出现，在通信一开始时就会通过一个叫做慢启动的算法得出的数值，对发送数据量进行控制。

图 6.20
慢启动



最初将发送端的窗口（拥塞窗口）设置为1。每收到一个确认应答，窗口的值会增加1个段。（图中所示为没有延迟确认应答的情况，因此与实际情况有所不同）

首先，为了在发送端调节所要发送数据的量，定义了一个叫做“拥塞窗口”的概念。于是在慢启动的时候，将这个拥塞窗口的大小设置为 1 个数据段（1MSS）▼发送数据，之后每收到一次确认应答（ACK），拥塞窗口的值就加 1。在发送数据包时，将拥塞窗口的大小与接收端主机通知的窗口大小做比较，然后按照它们当中较小那个值，发送比其还要小的数据量。

如果重发采用超时机制，那么拥塞窗口的初始值可以设置为 1 以后再进行慢启动修正。有了上述这些机制，就可以有效地减少通信开始时连续发包▼导致的网络拥堵，还可以避免网络拥塞情况的发生。

不过，随着包的每次往返，拥塞窗口也会以 1、2、4 等指数函数的增长，拥堵状况激增甚至导致网络拥塞的发生。为了防止这些，引入了慢启动阀值的概念。只要拥塞窗口的值超出这个阀值，在每收到一次确认应答时，只允许以下面这种比例放大拥塞窗口：

$$\frac{\text{1 个数据段的字节数}}{\text{拥塞窗口（字节）}} \times \text{1 个数据段字节数}$$

▼连接建立以后即刻从 1MSS 开始进行慢启动的话，通过卫星通信等手段提高通信吞吐量所耗的时间会比较长。为此，有时也会将慢启动的初始值设置大于 1MSS 的值。具体来说，MSS 的值小于 1095 字节时最大为 4MSS，小于 2190 字节时最大为 4390 字节，超过 2190 字节时最大值大于 2MSS。以太网的标准 MSS 值为 1460 字节，因此慢启动的初始值从 4380 字节（3MSS）开始就可以。

▼连续发包的情况也叫“爆发”（Burst）。慢启动正是减少爆发等网络拥堵情况的一种机制。

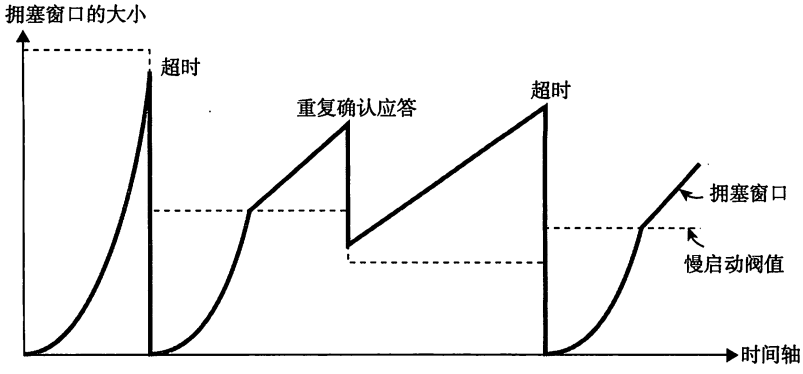


图 6.21 TCP 的窗口变化

拥塞窗口越大，确认应答的数目也会增加。不过随着每收到一个确认应答，其涨幅也会逐渐减少，甚至小过比一个数据段还要小的字节数。因此，拥塞窗口的大小会呈直线上升的趋势。

TCP 的通信开始时，并没有设置相应的慢启动阀值▼。而是在超时重发时，才会设置为当时拥塞窗口一半的大小。

▼与窗口的最大值相同。

由重复确认应答而触发的高速重发与超时重发机制的处理多少有些不同。因为前者要求至少 3 次的确认应答数据段到达对方主机后才会触发，相比后者网络的拥堵要轻一些。

而由重复确认应答进行高速重发控制时，慢启动阀值的大小被设置为当时窗口大小的一半▼。然后将窗口的大小设置为该慢启动阀值+3 个数据段的大小。

▼严格来说，是设置为“实际已发送但未收到确认应答的数据量”的一半。

有了这样一种控制，TCP 的拥塞窗口如图 6.21 所示发生变化。由于窗口的大小会直接影响数据被转发时的吞吐量，所以一般情况下，窗口越大，越会形成高吞吐量的通信。

当 TCP 通信开始以后，网络吞吐量会逐渐上升，但是随着网络拥堵的发生吞吐量也会急速下降。于是会再次进入吞吐量慢慢上升的过程。因此所谓 TCP 的吞吐量的特点就好像是在逐步占领网络带宽的感觉。

6.4.10 提高网络利用率的规范

Nagle 算法

TCP 中为了提高网络的利用率，经常使用一个叫做 Nagle 的算法。

该算法是指发送端即使还有应该发送的数据，但如果这部分数据很少的话，则进行延迟发送的一种处理机制。具体来说，就是仅在下列任意一种条件下才能发送数据。如果两个条件都不满足，那么暂时等待一段时间以后再进行数据发送。

- 已发送的数据都已经收到确认应答时
- 可以发送最大段长度（MSS）的数据时

根据这个算法虽然网络利用率可以提高，但是可能会发生某种程度的延迟。为此，在窗口系统▼以及机械控制等领域中使用 TCP 时，往往会关闭对该算法的启用。

▼X Window System 等。

延迟确认应答

接收数据的主机如果每次都立刻回复确认应答的话，可能会返回一个较小的

窗口。那是因为刚接收完数据，缓冲区已满。

当某个接收端收到这个小窗口的通知以后，会以它为上限发送数据，从而又降低了网络的利用率▼。为此，引入了一个方法，那就是收到数据以后并不立即返回确认应答，而是延迟一段时间的机制。

- 在没有收到2×最大段长度的数据为止不做确认应答（根据操作系统的不同，有时也有不论数据大小，只要收到两个包就即刻返回确认应答的情况。）
- 其他情况下，最大延迟0.5秒发送确认应答▼（很多操作系统设置为0.2秒左右▼）

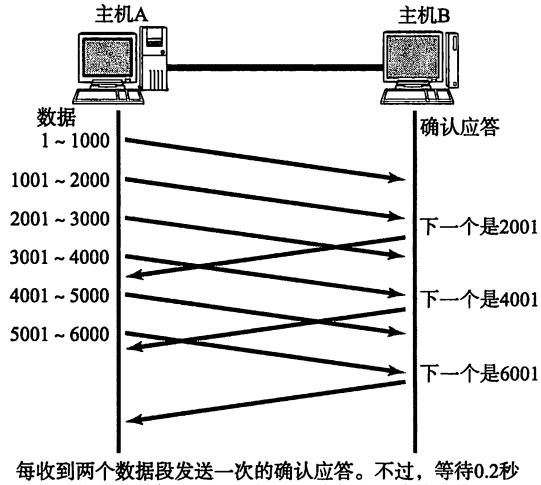
▼这其实是窗口控制特有的问题，专门术语叫做糊涂窗口综合征（SWS：Silly Window Syndrome）。

▼如果延迟多于0.5秒可能会导致发送端重发数据。

▼这个时间越小、CPU的负荷会越高，性能也下降。反之，这个时间越长，越有可能触发发送主机的重发处理，而窗口为只有1个数据段的时候，性能也会下降。

事实上，大可不必为每一个数据段都进行一次确认应答。TCP采用滑动窗口的控制机制，因此通常确认应答少一些也无妨。TCP文件传输中，绝大多数是每两个数据段返回一次确认应答。

图6.22
延迟确认应答



每收到两个数据段发送一次的确认应答。不过，等待0.2秒以后没有其他数据包到达的情况下才会发送确认应答。

捎带应答

根据应用层协议，发送出去的消息到达对端，对端进行处理以后，会返回一个回执。例如，电子邮件协议的SMTP或POP、文件传输协议FTP中的连接控制部分等。如图6.23所示，这些应用协议使用同一个连接进行数据的交互。即使是使用WWW的HTTP协议，从1.1版本以后也是如此。再例如远程登录中针对输入的字符进行回送校验▼也是对发送消息的一种回执。

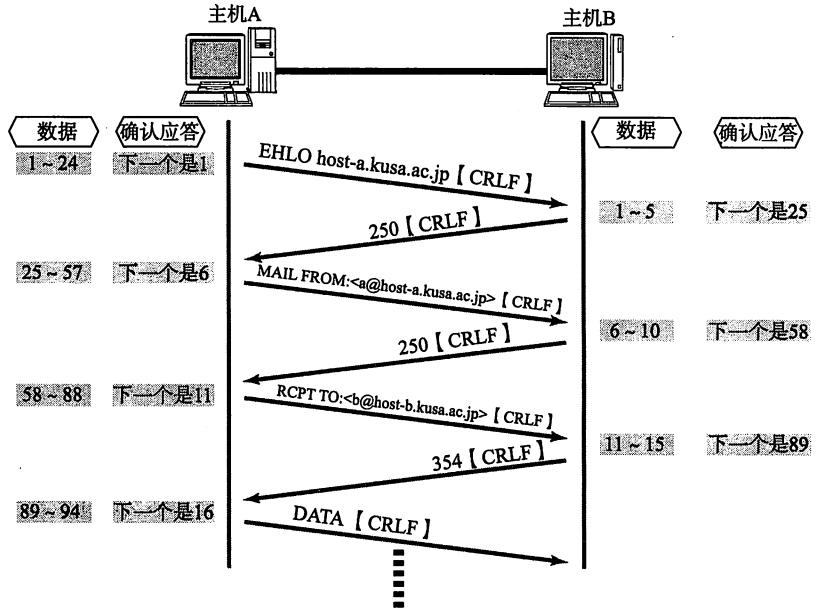
在此类通信当中，TCP的确认应答和回执数据可以通过一个包发送。这种方式叫做捎带应答▼（PiggyBack Acknowledgement）。通过这种机制，可以使收发的数据量减少。

另外，接收数据以后如果立刻返回确认应答，就无法实现捎带应答。而是将所接收的数据传给应用处理生成返回数据以后进再进行发送请求为止，必须一直等待确认应答的发送。也就是说，如果没有启用延迟确认应答就无法实现捎带应答。延迟确认应答是能够提高网络利用率从而降低计算机处理负荷的一种较优的处理机制。

▼回送校验是指在远程登录中，从键盘中输入的字符到达服务器以后再返回来显示给客户端的意思。

▼在农村人们到集市上卖猪时，顺便在猪背拖上几篮子菜一起带去集市的场景。其实就是顺带、捎带的意思。

图 6.23
捎带应答



捎带应答是指在同一个TCP包中既发送数据又发送确认应答的一种机制。由此，网络的利用率会提高，计算机的负荷也会减轻。不过，确认应答必须得等到应用处理完数据并将作为回执的数据返回为止，才能进行捎带应答。

6.4.11 使用 TCP 的应用

到此为止，读者可以了解到 TCP 使用各种各样的控制机制。甚至它还会使用本书中未提及的其他更为复杂的控制机制。TCP 采用这些机制可以提供高速、可靠的通信服务。

不过，有时这些机制也会受其一定缺陷的困扰。为此，在开发应用的时候，有必要考虑一下是全权交给 TCP 去处理好，还是由应用自己进行更细微的控制好。

如果需要应用自己处理一些更为细节上的控制，使用 UDP 协议是不错的选择。如果转发数据量较多、对可靠性的要求比较高时，可以选择使用 TCP。TCP 和 UDP 两者各有长短，在设计和开发应用时，应准确掌握它们各自协议的特点酌情选择。

6.5 其他传输层协议

在互联网中，很长一段时间主要使用的传输层协议是TCP和UDP两种。然而，除了这两个协议之外还有其他几种传输层协议曾被提案并进行了实验。最近更是有几个协议从实验阶段步入了实用阶段。

本节旨在介绍部分已经被提案并在今后可能会广泛使用的传输层协议。

6.5.1 UDP-Lite

UDP-Lite（Lightweight User Datagram Protocol，轻量级用户数据报协议）是扩展UDP机能的一种传输层协议。在基于UDP的通信当中如果校验和出现错误，所收到的包将被全部丢弃。然而，现实操作中，有些应用▼在面对这种情况时并不希望把已经收到的所有包丢弃。

▼例如那些使用H.263+，H.264，MPEG-4等图像与音频数据格式的应用。

如果将UDP中校验和设置为无效，那么即使数据的一部分发生错误也不会将整个包废弃。不过，这不是一个很好的方法。因为如果发生的错误有可能是UDP首部中的端口号被破坏或是IP首部中的IP地址被破坏▼，就会产生严重后果。因此，不建议将校验和关闭。为了解决这些问题，UDP的修正版UDP-Lite协议就出现了。

▼识别一个通信需要IP地址，而UDP的校验和可以检查IP地址是否正确。更多细节请参考6.6节。

UDP-Lite提供与UDP几乎相同的功能，不过计算校验和的范围可以由应用自行决定。这个范围可以是包加上伪首部的校验和计算，可以是首部与伪首部的校验和计算，也可以是首部、伪首部与数据从起始到中间某个位置的校验和计算▼。有了这样的机制，就可以只针对不允许发生错误的部分进行校验和的检查。对于其他部分，即使发生了错误，也会被忽略不计。而这个包也不会被丢弃，而是直接传给应用继续处理。

▼在UDP首部有一个字段表示“包长”。在这个字段里放入是从协议首部的第1个字节到第多少个字节要进行校验和计算的部分。如果值为0表示整个包都要进行校验和计算，如果值为8表示只对首部与伪首部进行校验和计算。

6.5.2 SCTP

SCTP（Stream Control Transmission Protocol，流控制传输协议）▼与TCP一样，都是对一种提供数据到达与否相关可靠性检查的传输层协议。其主要特点如下：

▼SS7协议最初被应用于TCP/IP上时，由于TCP本身使用起来不是很方便，所以人们开发了SCTP协议。今后它可能会出现各种各样的使用途径。

- 以消息为单位收发

  TCP中接收端并不知道发送端应用所决定的消息大小。在SCTP中却可以。
- 支持多重宿主

  在有多个NIC的主机中，即使其中能够使用的NIC发生变化，也仍然可以继续通信▼。
- 支持多数据流通信

  TCP中建立多个连接以后才能进行通信的效果，在SCTP中一个连接就可以。▼
- 可以定义消息的生存期限

  超过生存期限的消息，不会被重发。

▼这与TCP相比提高了故障应对能力。

▼吞吐量得到有效提升。

SCTP主要用于进行通信的应用之间发送众多较小消息的情况。这些较小的应

用消息被称作数据块（Chunk），多个数据块组成一个数据包。

此外，SCTP 具有支持多重宿主以及设定多个 IP 地址的特点。多重宿主是指同一台主机具备多种网络的接口。例如，笔记本电脑既可以连接以太网又可以连接无线 LAN。

同时使用以太网和无线 LAN 时，各自的 NIC 会获取到不同的 IP 地址。进行 TCP 通信，如果开始时使用的是以太网，而后又切换为无线 LAN，那么连接将会被断开。因为从 SYN 到 FIN 包必须使用同一个 IP 地址。

然而在 SCTP 的情况下，由于可以管理多个 IP 地址使其同时进行通信，因此即使出现通信过程当中以太网与无线 LAN 之间的切换，也能够保持通信不中断。所以 SCTP 可以为具备多个 NIC 的主机提供更可靠的传输▼。

▼持有多个 NIC 的应用服务器中，即使某一个 NIC 发生故障，只要有一个能够正常工作的 NIC 就可以保持通信无阻。

6.5.3 DCCP

DCCP（Datagram Congestion Control Protocol，数据报拥塞控制协议）是一个辅助 UDP 的崭新的传输层协议。UDP 没有拥塞控制机制。为此，当应用使用 UDP 发送大量数据包时极容易出现问题。互联网中的通信，即使使用 UDP 也应该控制拥塞。而这个机制开发人员很难将其融合至协议中，于是便出现了 DCCP 这样的规范。

DCCP 具有如下几个特点：

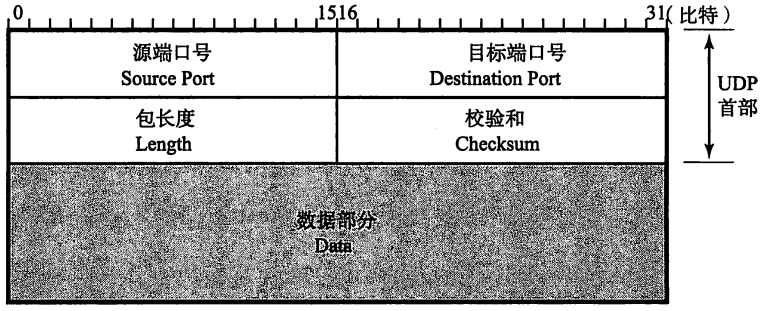
- 与 UDP 一样，不能提供发送数据的可靠性传输。
- 它面向连接，具备建立连接与断开连接的处理。在建立和断开连接上是具有可靠性。
- 能够根据网络拥堵情况进行拥塞控制。使用 DCCP（RFC4340）应用可以根据自身特点选择两种方法进行拥塞控制。它们分别是“类似 TCP（TCP-Like）拥塞控制”和“TCP 友好升级控制”（TCP-Friendly Rate Control）▼（RFC4341）。
- 为了进行拥塞控制，接收端收到包以后返回确认应答（ACK）。该确认应答将被用于重发与否的判断。

▼流控制的一种。它根据单位时间内能够发送的比特数（字节数）进行流控制。相比 TCP 的窗口控制，可以说 TFRC 是针对声频和视频等多媒体的一种控制机制。

6.6 UDP首部的格式

图6.24展示了UDP首部的格式。除去数据的部分正是UDP的首部。UDP首部由源端口号，目标端口号，包长和校验和组成。

图6.24
UDP数据报格式



源端口号（Source Port）

▼例如，只针对某个主机或应用，亦或针对某个组织，只单方面发送更新消息，不需要接收端返回任何确认或应答。

表示发送端端口号，字段长16位。该字段是可选项，有时可能不会设置源端口号。没有源端口号的时候该字段的值设置为0。可用于不需要返回的通信中▼。

目标端口号（Destination Port）

表示接收端端口，字段长度16位。

包长度（Length）

▼在UDP-Lite（6.5.1节）中，该字段变为Checksum Coverage，表示校验和的计算范围。

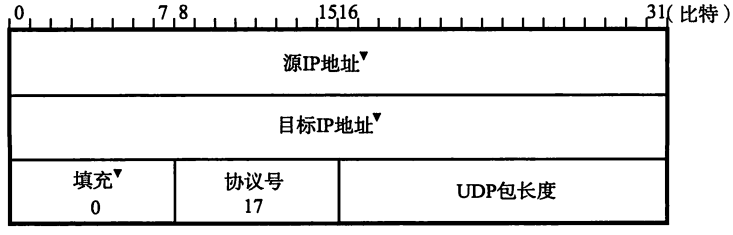
该字段保存了UDP首部的长度跟数据的长度之和▼。单位为字节（8位字节）。

校验和（Checksum）

▼通常在计算机的整数计算中常用2的补码形式。而在校验和计算中之所以使用1的补码形式，是因为即使有一位溢出会回到第1位，也不会造成信息丢失。而且在这种形式下0可以有两种表示方式，因此有用0表示两种不同意思的优点。

校验和是为了提供可靠的UDP首部和数据而设计。在计算校验和时，如图6.25所示，附加在UDP伪首部与UDP数据报之前。通过在最后一位增加一个“0”将全长增加16倍。此时将UDP首部的校验和字段设置为“0”。然后以16比特为单位进行1的补码▼和，并将所得到的1的补码和写入校验和字段。

图6.25
校验和计算中使用的UDP伪首部



▼源IP地址与目标IP地址为IPv4地址的情况下都是32位字段，为IPv6地址时都是128位字段。

▼填充是为了补充位数，一般填入0。

接收主机在收到UDP数据报以后，从IP首部获知IP地址信息构造UDP伪首部，再进行校验和计算。校验和字段的值是校验和字段以外剩下部分的1的补码和。因此，包括校验和字段在内的所有数据之和结果为“16位全部为1▼”时，才会被认为所收到的数据是正确的。

▼1的补码中该值为0（负数0）、二进制中为1111111111111111，十六进制中为FFFF，十进制中则为65535。

另外，UDP中也有可能不用校验和。此时，校验和字段中填入0。这种情况下，由于不进行校验和计算，协议处理的开销▼就会降低，从而可以提高数据转

▼在处理实际数据之外，为了进行通信控制的处理而不得不付出的必要的消耗部分。

发的速度。然而，如果 UDP 首部的端口号或是 IP 首部的 IP 地址遇到损坏，那么可能会对其他通信造成不好的影响。因此，在互联网中比较推荐使用校验和检查。

校验和计算中计算 UDP 伪首部的理由

为什么在进行校验和计算时，也要计算 UDP 伪首部呢？关于这个问题，与 6.2 节中所介绍的内容有所关联。

TCP/IP 中识别一个进行通信的应用需要 5 大要素，它们分别为“源 IP 地址”、“目标 IP 地址”、“源端口”、“目标端口”、“协议号”。然而，在 UDP 的首部中只包含它们当中的两项（源端口和目标端口），余下的 3 项都包含在 IP 首部里。

假定其他 3 项的信息被破坏会产生什么样的后果呢？很显然，这极有可能会导致应该收包的应用收不到包，不该收到包的应用却收到了包。

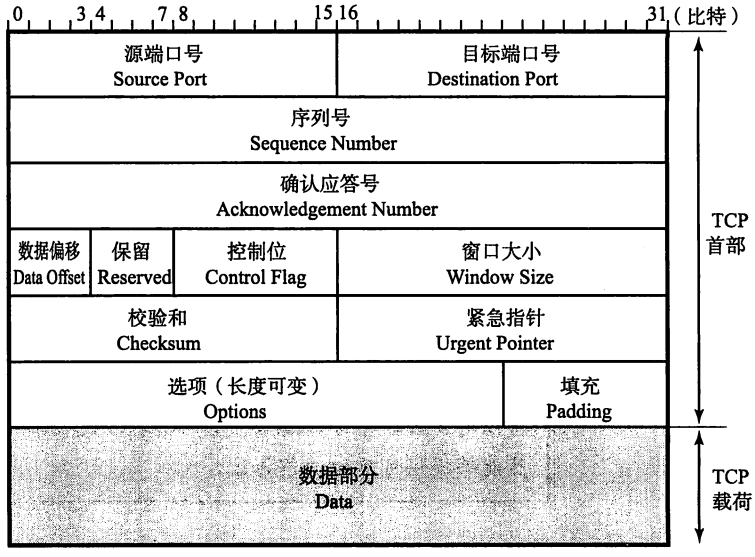
为了避免这类问题，有必要验证一个通信中必要的 5 项识别码是否正确。为此，在校验和的计算中就引入了伪首部的概念。

此外，IPv6 中的 IP 首部没有校验和字段。TCP 或 UDP 通过伪首部，得以对 5 项数字进行校验，从而实现即使在 IP 首部并不可靠的情况下仍然能够提供可靠的通信传输。

6.7 TCP首部格式

图6.26展示了TCP首部的格式。TCP首部相比UDP首部要复杂得多。

图6.26
TCP数据段格式



另外，TCP中没有表示包长度和数据长度的字段。可由IP层获知TCP的包长由TCP的包长可知数据的长度。

■ 源端口号（Source Port）

表示发送端端口号，字段长16位。

■ 目标端口号（Destination Port）

表示接收端端口号，字段长度16位。

■ 序列号（Sequence Number）

字段长32位。序列号（有时也叫序号）是指发送数据的位置。每发送一次数据，就累加一次该数据字节数的大小。

序列号不会从0或1开始，而是在建立连接时由计算机生成的随机数作为其初始值，通过SYN包传给接收端主机。然后再将每转发过去的字节数累加到初始值上表示数据的位置。此外，在建立连接和断开连接时发送的SYN包和FIN包虽然并不携带数据，但是也会作为一个字节增加对应的序列号。

■ 确认应答号（Acknowledgement Number）

确认应答号字段长度32位。是指下一次应该收到的数据的序列号。实际上，它是指已收到确认应答号减一为止的数据。发送端收到这个确认应答以后可以认为在这个序号以前的数据都已经被正常接收。

■ 数据偏移（Data Offset）

该字段表示TCP所传输的数据部分应该从TCP包的哪个位开始计算，当然也可以把它看作TCP首部的长度。该字段长4位，单位为4字节（即32位）。不包括选项字段的话，如图6.26所示TCP的首部为20字节长，因此数据偏移字段可

以设置为5。反之，如果该字段的值为5，那说明从TCP包的最一开始到20字节为止都是TCP首部，余下的部分为TCP数据。

保留（Reserved）

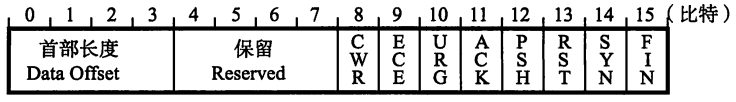
该字段主要是为了以后扩展时使用，其长度为4位。一般设置为0，但即使收到的包在该字段不为0，此包也不会被丢弃▼。

▼保留字段的第4位（如图6.27中的第7位）用于实验目的，相当于NS（Nonce Sum）标志位。

控制位（Control Flag）

字段长为8位，每一位从左至右分别为CWR、ECE、URG、ACK、PSH、RST、SYN、FIN。这些控制标志也叫做控制位。当它们对应位上的值为1时，具体含义如图6.27所示。

图6.27 控制位



- CWR（Congestion Window Reduced）

  CWR标志▼与后面的ECE标志都用于IP首部的ECN字段。ECE标志为1时，则通知对方已将拥塞窗口缩小。

▼关于CWR标志的设定请参考5.8.4节。

- ECE（ECN-Echo）

  ECE标志▼表示ECN-Echo。置为1会通知通信对方，从对方到这边的网络有拥塞。在收到数据包的IP首部中ECN为1时将TCP首部中的ECE设置为1。

▼关于ECE标志的设定请参考5.8.4节。

- URG（Urgent Flag）

  该位为1时，表示包中有需要紧急处理的数据。对于需要紧急处理的数据，会在后面的紧急指针中再进行解释。

- ACK（Acknowledgement Flag）

  该位为1时，确认应答的字段变为有效。TCP规定除了最初建立连接时的SYN包之外该位必须设置为1。

- PSH（Push Flag）

  该位为1时，表示需要将受到的数据立刻传给上层应用协议。PSH为0时，则不需要立即传而是先进行缓存。

- RST（Reset Flag）

  该位为1时表示TCP连接中出现异常必须强制断开连接。例如，一个没有被使用的端口即使发来连接请求，也无法进行通信。此时就可以返回一个RST设置为1的包。此外，程序宕掉或切断电源等原因导致主机重启的情况下，由于所有的连接信息将全部被初始化，所以原有的TCP通信也将不能继续进行。这种情况下，如果通信对方发送一个设置为1的RST包，就会使通信强制断开连接。

- SYN（Synchronize Flag）

  用于建立连接。SYN为1表示希望建立连接，并在其序列号的字段进行序列号初始值的设定▼。

▼Synchronize本身有同步的意思。也就意味着建立连接的双方，序列号和确认应答号要保持同步。

- FIN（Fin Flag）

  该位为1时，表示今后不会再有数据发送，希望断开连接。当通信结束希望断开连接时，通信双方的主机之间就可以相互交换FIN位置为1的TCP

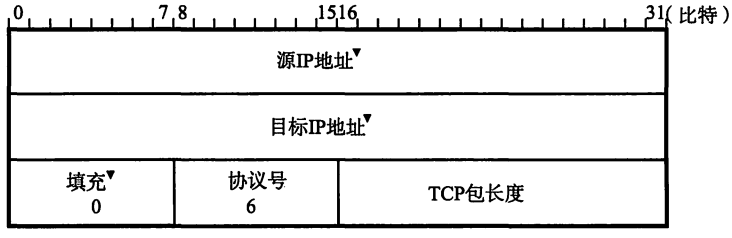
段。每个主机又对对方的FIN包进行确认应答以后就可以断开连接。不过，主机收到FIN设置为1的TCP段以后不必马上回复一个FIN包，而是可以等到缓冲区中的所有数据都因已成功发送而被自动删除之后再发。

窗口大小（Window Size）

该字段长为16位。用于通知从相同TCP首部的确认应答号所指位置开始能够接收的数据大小（8位字节）。TCP不允许发送超过此处所示大小的数据。不过，如果窗口为0，则表示可以发送窗口探测，以了解最新的窗口大小。但这个数据必须是1个字节。

校验和（Checksum）

图6.28
用于校验和计算的TCP伪首部



▼源IP地址与目标IP地址在IPv4的情况下都是32位字段，在IPv6地址时都为128位字段。

▼填充是为了补充位数时用，一般填入0。

TCP的校验和与UDP相似，区别在于TCP的校验和无法关闭。

TCP和UDP一样在计算校验和的时候使用TCP伪首部。这个伪首部如图6.28所示。为了让其全长为16位的整数倍，需要在数据部分的最后填充0。首先将TCP校验和字段设置为0。然后以16位为单位进行1的补码和计算，再将它们总和的1的补码和放入校验和字段。

接收端在收到TCP数据段以后，从IP首部获取IP地址信息构造TCP伪首部，再进行校验和计算。由于校验和字段里保存着除本字段以外其他部分的和的补码值，因此如果计算校验和字段在内的所有数据的16位和以后，得出的结果是“16位全部为1▼”说明所收到的数据是正确的。

▼1的补码中该值为0（负数0）、二进制中为1111111111111111，十六进制中为FFFF，十进制中则为正整数65535。

使用校验和的目的是什么？

有噪声干扰的通信途中如果出现位错误，可以由数据链路的FCS检查出来。那么为什么TCP或UDP中也需要校验和呢？

其实，相比检查噪声影响导致的错误，TCP与UDP的校验和更是一种进行路由器内存故障或程序漏洞导致的数据是否被破坏的检查。

有过C语言编程经验的人都知道，如果指针使用不当，极有可能会破坏内存中的数据结构。路由器的程序中也可能会存在漏洞，或程序异常宕掉的可能。在互联网中发送数据包要经由好多个路由器，一旦在发送途中的某一个路由器发生故障，经过此路由器的包、协议首部或数据就极有可能被破坏。即使在这种情况下，TCP或UDP如果能够提供校验和计算，也可以判断协议首部和数据是否被破坏。

紧急指针（Urgent Pointer）

该字段长为 16 位。只有在 URG 控制位为 1 时有效。该字段的数值表示本报文段中紧急数据的指针。正确来讲，从数据部分的首位到紧急指针所指示的位置为止为紧急数据。因此也可以说紧急指针指出了紧急数据的末尾在报文段中的位置。

如何处理紧急数据属于应用的问题。一般在暂时中断通信，或中断通信的情况下使用。例如在 Web 浏览器中点击停止按钮，或者使用 TELNET 输入 Ctrl + C 时都会有 URG 为 1 的包。此外，紧急指针也用作表示数据流分段的标志。

选项（Options）

选项字段用于提高 TCP 的传输性能。因为根据数据偏移（首部长度）进行控制，所以其长度最大为 40 字节。

另外，选项字段尽量调整其为 32 位的整数倍。具有代表性的选项如表 6.3 所示，我们从中挑些重点进行讲解。

表 6.3
具有代表性的 TCP 选项

类型	长度	意　义	RFC
0	-	End of Option List	RFC793
1	-	No-Operation	RFC793
2	4	Maximum Segment Size	RFC793
3	3	WSOPT-Window Scale	RFC1323
4	2	SACK Permitted	RFC2018
5	N	SACK	RFC2018
8	10	TSOPT-Time Stamp Option	RFC1323
27	8	Quick-Start Response	RFC4782
28	4	User Timeout Option	RFC5482
29	-	TCP Authentication Option（TCP-AO）	RFC5925
253	N	RFC3692-style Experiment 1	RFC4727
254	N	RFC3692-style Experiment 2	RFC4727

类型 2 的 MSS 选项用于在建立连接时决定最大段长度的情况。这选项用于大部分操作系统。

类型 3 的窗口扩大，是一个用来改善 TCP 吞吐量的选项。TCP 首部中窗口字段只有 16 位。因此在 TCP 包的往返时间（RTT）内，只能发送最大 64K 字节的数据▼。如果采用了该选项，窗口的最大值可以扩展到 1G 字节。由此，即使在一个 RTT 较长的网络环境中，也能达到较高的吞吐量。

▼例如在 RTT 为 0.1 秒时，不论数据链路的带宽多大，最大也只有 5Mbps 的吞吐量。

类型 8 时间戳字段选项，用于高速通信中对序列号的管理。若要将几个 G 的数据高速转发到网络时，32 位序列号的值可能会迅速使用完。在传输不稳定的网络环境下，就有可能会在较晚的时间点却收到散布在网络中的一个较早序列号的包。而如果接收端对新老序列号产生混淆就无法实现可靠传输。为了避免这个问题的发生，引入了时间戳这个选项，它可以区分新老序列号。

类型 4 和 5 用于选择确认应答（SACK：Selective ACKnowledgement）。TCP 的

▼这个形象的比喻是指数据段在途中丢失的情况。尤其是时不时丢失的情况。其结果就是在接收方收到的数据段的序号不连续，呈有一个没一个的状态。

确认应答一般只有1个数字，如果数据段总以“豁牙子状态▼”到达的话会严重影响网络性能。有了这个选项，就可以允许最大4次的“豁牙子状态”确认应答。因此在避免无用重发的同时，还能提高重发的速度，从而也能提高网络的吞吐量。

窗口大小与吞吐量

TCP通信的最大吞吐量由窗口大小和往返时间决定。假定最大吞吐量为T_{max}，窗口大小为W，往返时间是RTT的话，那么最大吞吐量的公式如下：

$$T_{max}=\frac{W}{RTT}$$

假设窗口为65535字节，RTT为0.1秒，那么最大吞吐量T_{max}如下：

$$T_{max}=\frac{65535\text{（字节）}}{0.1\text{（秒）}}=\frac{65535\times 8\text{（比特）}}{0.1\text{（秒）}}$$

$$=5242800\text{（bps）}\fallingdotseq 5.2\text{（Mbps）}$$

以上公式表示1个TCP连接所能传输的最大吞吐量为5.2Mbps。如果建立两个以上连接同时进行传输时，这个公式的计算结果则表示每个连接的最大吞吐量。也就是说，在TCP中，与其使用一个连接传输数据，使用多个连接传输数据会达到更高的网络吞吐量。在Web浏览器中一般会通过同时建立4个左右连接来提高吞吐量。