牺牲了可靠性，但提高了实时性。

从可通信对象的数量来看，TCP 只支持**单播**，如果想要与多个对象进行通信，就必须针对每个通信对象建立一个连接。

在传输类型方面，TCP 是**流模式**，应用层待发送的数据都会首先经过 TCP 的处理（将数据分割为效率高的大小），再按照顺序被发送到网络层；而 UDP 属于**数据报模式**，应用层待发送的数据需要直接加上 UDP 首部，然后才被发送到网络层。

适合 TCP 的应用程序　准确无误地传输数据

比起实时性，更需要准确无误地传输数据的应用程序就会使用 TCP 协议。用于 Web 网页浏览的 HTTP 协议和用于发送邮件的 SMTP 协议可以说是最有代表性的例子。此外，用于文件上传和下载的 FTP 协议也是基于 TCP 协议的。

最近，YouTube 等视频网站的一部分服务使用了 TCP 协议。这些视频并非实时播放，而是一边少量下载一边播放的。这样的话，用户在观看视频时就不会因为频繁卡顿而烦躁了。另外，日本各金融机构和银行与用户计算机之间的通信（基于日本银行协会标准协议，该协议将于 2023 年废除）也使用了 TCP 协议。显然，这种情况下也是绝对不允许出现数据丢失的。

1.5 TCP 的基本功能

重传、顺序控制和拥塞控制

TCP 作为确保可靠性的通信协议，除了**重传**（对丢失的数据进行补偿）和**顺序控制**，还有**拥塞控制**功能，可以尽可能地避免网络拥塞，高效地转发数据。本节将介绍用来支撑上述功能的基本要素。

连接管理

TCP协议建立一对一的连接，通过数据与ACK的往返交互确保通信可靠性。

图1.11展示的是面向连接的通信的基本流程。首先，TCP会建立收发数据过程中的连接。**连接管理**（connection management）会使用TCP首部中的控制位字段，这一点详见第3章。当连接建立之后，发送方进入数据发送阶段。当数据发送完成后，连接会被断开。这样一来，整个通信过程就被完整地管理了起来。

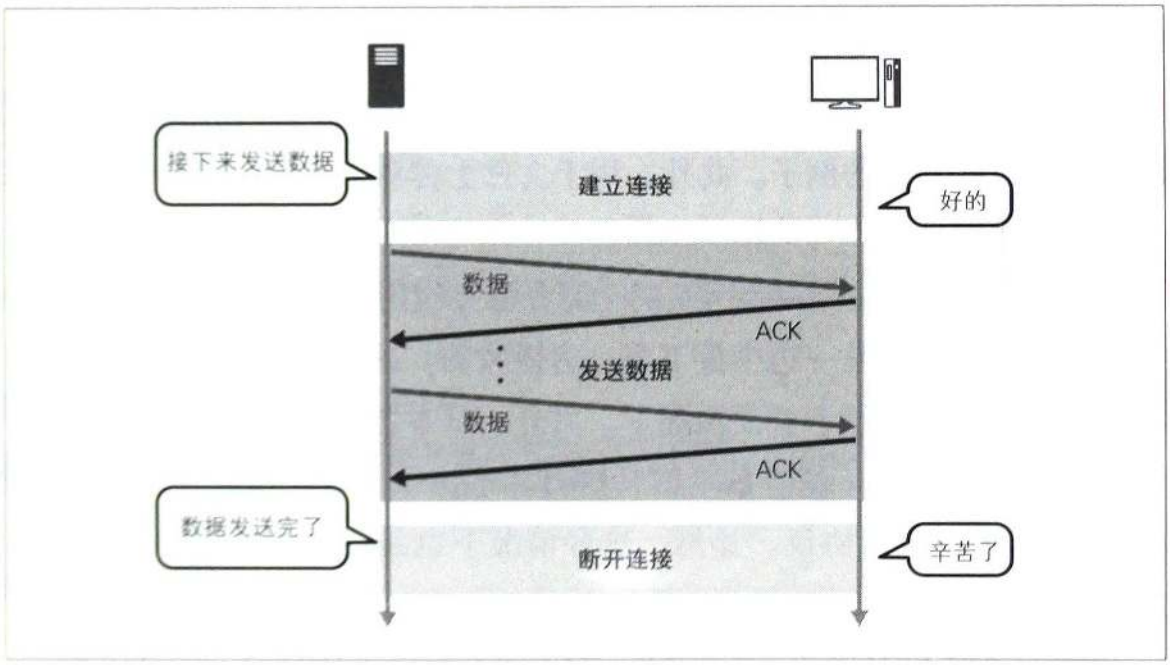

图1.11 面向连接的通信的基本流程

序列号

为了保证TCP报文段的顺序正确，以及检测出TCP报文段是否丢失，每个待发送的TCP报文段都会加上**序列号**（sequence number）。序列号是以1字节为单位来计数的。举例来说，当一次发送的数据长度[1]为1000字

[1] 这称为*MSS*（Maxium Segment Size，最大报文段长度），详见第3章。

节时，每发送一次数据，序列号就增加 1000。接收方会把接下来希望接收的数据的序列号写入 ACK，返回发送方。

图 1.12 中展示的是数据和 ACK 的交互过程示例。当发送序列号为 3001、长度为 1000 字节的数据时，接收方接收到的就是序列号到 4000 为止的数据。接下来要接收的数据的序列号是 4001，因此 ACK 中写入的序列号（**确认应答号**）是 4001。发送方得到 ACK 中的序列号后，从待发送的 TCP 报文段中选择 *MSS* 大小的数据发送。

收发双方要想顺利地按照以上流程利用序列号完成数据的互相通信，就必须同步双方 TCP 模块中的初始序列号和 ACK 初始值。此同步过程发生在连接建立的时候。

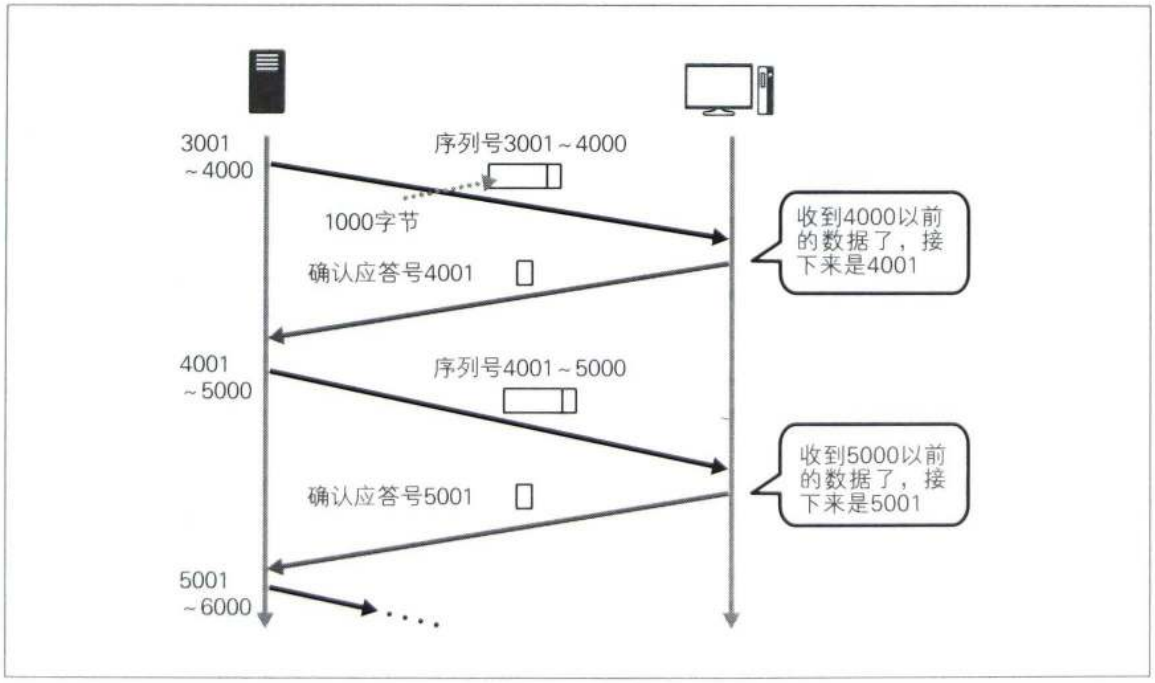

图 1.12 序列号

重传控制 重传计时器、RTT 和 ACK

重传可对丢失的数据进行补偿。那么，TCP 中使用的**重传控制**是如何知道哪个数据包丢失的？

最基本的方法是使用**重传计时器**（retransmission timer）。TCP 持续计

算从TCP报文段发送开始到收到ACK为止的时间，也就是***RTT***（Round Trip Time，往返时延），然后以此为基础，计算出一个比*RTT*长的时间*RTO*（Retransmission Time Out，超时重传时间），接着对已发送的某个序列号的TCP报文段，设置一个计时器，当经过了*RTO*之后仍然没有收到ACK时，就认为此TCP报文段已经丢失，并进行重传（图1.13）。TCP协议认为网络拥塞是造成数据丢失的原因。

另一种方法是**使用ACK**。如前文所述，接收方会将接下来待接收的序列号写入ACK并返回给发送方。当然，我们也可以称之为尚未收到的序列号。接收方在有未收到的TCP报文段时，会一直发送写有对应序列号的ACK，直到收到对应序列号的TCP报文段为止。结果就是，发送方将多次收到请求同一个序列号的ACK。如果收到3次以上，则认为该TCP报文段已经丢失，并尝试使用“快速重传算法”进行数据重传（详见第3章）。

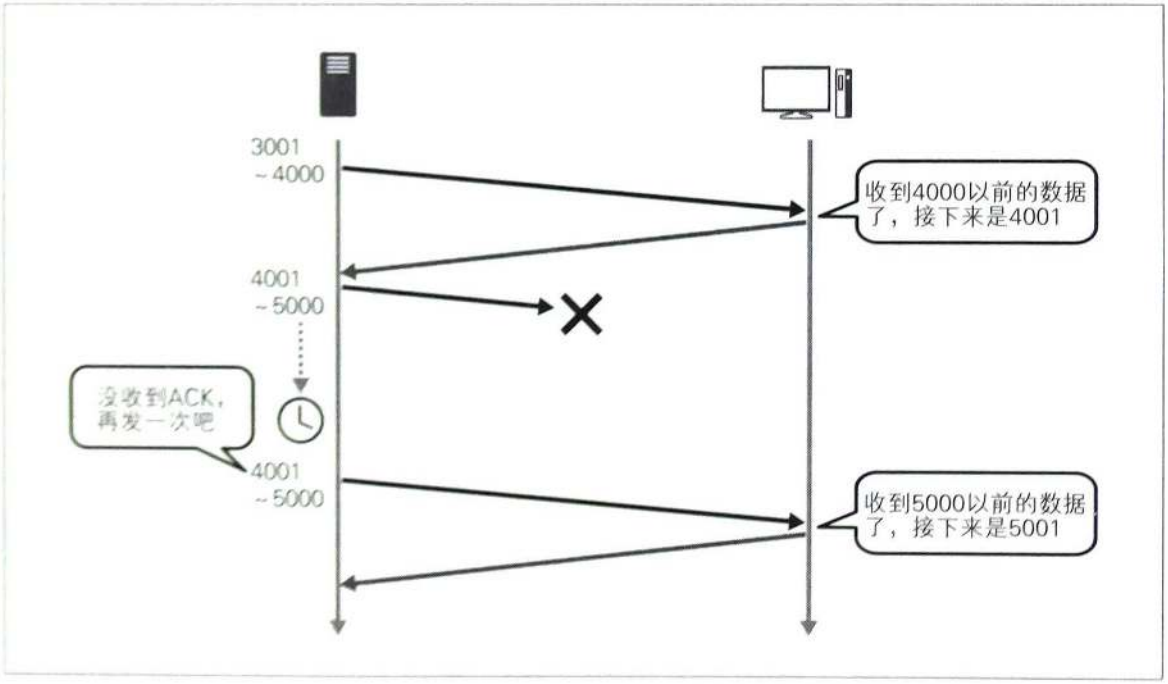

图1.13 超时重传

顺序控制

经由网络发送的数据如果顺序出错，则无法还原。TCP首部中有序列

号字段，接收方会读取或保存序列号，并按照正确的顺序整理好接收的 TCP 报文段（图 1.14）。这里的**顺序控制**（sequence control）也是确保通信可靠性的重要功能。

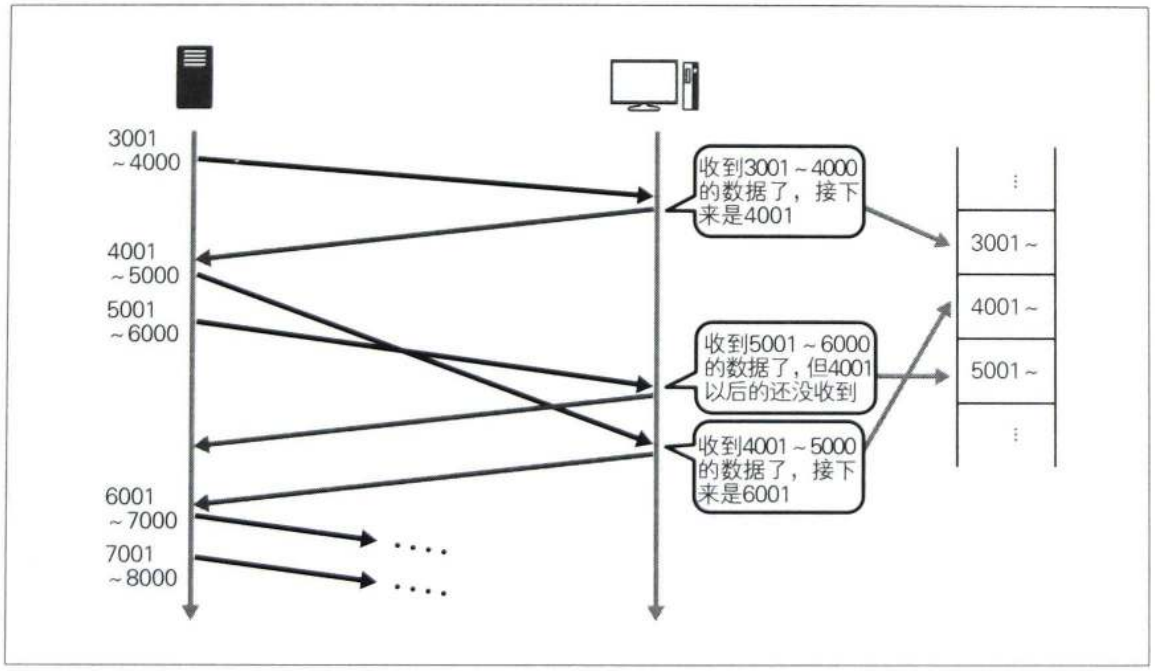

图 1.14 顺序控制

端口号

TCP 和 UDP 都分别定义了**端口号**（port number）。端口号主要用来识别各应用程序的通信会话和 TCP/UDP 的连接。在通常情况下，端口号和应用程序存在对应关系（表 1.5）。大家也许能在表 1.5 中找到眼熟的数字。

表 1.5 常见的端口号与应用程序的对应关系

端口号	应用层协议
21	FTP（文件传输协议）
23	Telnet
25	SMTP（Simple Mail Transfer Protocol，简单邮件传输协议）
80	HTTP（Hypertext Transfer Protocol，超文本传输协议）

（续）

端口号	应用层协议
110	POP3（Post Office Protocol-Version 3，邮局协议 - 版本 3）
143	IMAP（Internet Message Access Protocol，互联网报文存取协议）
443	HTTPS（Hypertext Transfer Protocol Secure，超文本传输安全协议）

这样一来，有人可能会觉得端口号和传输层协议之间也存在对应关系。然而，TCP 和 UDP 是可以使用同一个端口号的。应用程序在发送数据时，可以指定使用 TCP 或 UDP，但是接收方在接收数据时，就必须判断“究竟使用 TCP 和 UDP 中的哪个比较好”。

因此，传输层协议的信息就需要放在 IP 首部之中，这部分字段就是**协议号**（protocol number）。网络层以协议号为依据，判断要使用的传输层协议，并将数据移交给上层的 TCP 或 UDP 模块。

端口号有三种。一种是前文所述的为广泛使用的应用程序准备的公认（well-known）端口号，一种是一些事先注册好的注册（registered）端口号，还有一种是可以随意使用的动态或私有（dynamic/private）端口号。

提供服务的一方（server，**服务器**）需要使用应用程序确定好的端口号，而使用服务的一方（client，**客户端**）可以任意选择端口号。客户端的端口号是由操作系统分配的。基于此，当客户端同时打开多个浏览器时，即使是同一个应用程序（Web 浏览），也可以同时建立多条连接（浏览器和标签页）。

流量控制 窗口和窗口大小

数据发送方在给接收方发送数据时，不能随心所欲，想发就发。接收方设备有用来缓存接收的数据的缓冲区，如果一时间接收到超过缓冲区容量的数据，接收方将无法接收数据，这会导致缓冲区溢出，进而导致数据丢失。为了避免缓冲区溢出，接收方需要告知发送方自己所能接收的最大数据量，然后发送方以此为依据调整发送的数据量，这种机制称为**流量控制**（flow control）。

在 TCP 中，发送方一次可以发送的最大数据量称为**窗口**（window），

窗口的容量则称为**窗口大小**（window size）。

我们将发送方的窗口大小作为参数，记作 *swnd*（send window）。发送窗口大小 *swnd* 需要调整成接收方能处理的最大数据量。因此，接收方会将自己所能接收和处理的最大数据量，也就是**接收窗口大小**，告知发送方。我们将这个接收窗口大小作为接收方的参数，记作 *rwnd*（receive window）。发送方会调整数据发送量[①]，确保数据量不超过接收窗口大小 *rwnd*。

拥塞控制与拥塞控制算法 基于丢包和基于延迟

流量控制算法会根据通信对象的具体情况来发送数据，但这还不够，在发送数据时也必须考虑到网络整体的情况。如果所有的设备什么都不管，肆意地发送数据，就会导致网络频繁发生拥塞问题。为了规避这个问题，TCP 基于**拥塞控制算法**（congestion control algorithm）来调整数据发送量。

实现**拥塞控制**的基本功能，简而言之有以下 3 点。

❶ 收到 ACK 之后，发送接下来的数据。
❷ 逐渐增加数据发送量。
❸ 当发现数据丢失时，减少数据发送量。

这便是 TCP 中最为典型的基于丢包的控制方法。这里有一个参数，即**拥塞窗口大小**，它用于主动地调整发送方待发送的数据量，这里将其记作 *cwnd*（congestion window）。发送窗口大小 *swnd* 的值，取决于通信对象传过来的接收窗口大小 *rwnd* 和发送方自己的拥塞窗口大小 *cwnd*。基本的算法是优先使用两者中较小的那一个。❷ 的功能通过逐步增大 *cwnd* 实现，而 ❸ 主要是减小 *cwnd*。

基于丢包的拥塞控制算法，与字面意思一致，就是根据数据的丢包情况来判断网络的拥塞情况。以此为前提，可以得出最佳的 *cwnd* 的取值一定在发生拥塞前的 *cwnd* 与发生拥塞后的 *cwnd* 之间。如果检测到发生拥

① 接收窗口大小也称为通知窗口大小（advertised window）。

塞，就先减小 *cwnd*，然后再一边增大 *cwnd* 一边发送数据，慢慢将 *cwnd* 调节到最优结果附近。拥塞控制算法将根据情况调整基于❷的 *cwnd* 的增大方式和❸中的 *cwnd* 的减小方式，从而最终实现控制拥塞这一目的。

此外还有一种方法，即使用了 *RTT* 的基于延迟的拥塞窗口控制算法。第3章将对此进行详细介绍。

无线通信与TCP

现在，许多通信发生在智能手机等**移动设备**之间。也就是说，通信链路上存在**无线链路**。

此外，随着**物联网**的普及，各种终端设备应该都会通过无线连接接入网络，而非各种线缆。

在讨论今后的网络和TCP时，抛去**无线通信**是不可能的，因此本书会在方方面面涉及无线通信。接下来，我们就来介绍一下无线通信的基本原理和其与TCP的关系。

无线通信的基本情况 最大的优点与受限的通信范围，固定式与移动式

无线通信指的是含有信息的信号被调制到电波（**电磁波**）上，通过天线发射到周围的空间中，以周围的空间为媒介传输，最终到达目的地天线中的过程。

一方面，电波在空间中传播时是呈扩散形的放射状。也就是说，只要在电波可以到达的范围内，哪里都能收到信号。这一点可以说是无线通信的最大优点。另一方面，电波的强度会随着距离增大而逐渐衰减，因此相比有线传输来说，无线通信的通信范围比较有限。在一般情况下，光纤等有线传输的范围有几十千米，而无线通信的范围只有几百米到几千米。当然，也有范围在几十千米的类型。

无线通信主要分为**固定式**和**移动式**两种类型。

固定式的无线通信（**固定式通信**）指的是位置固定的天线之间的通信。举例来说，在山川地带、海上和城市大楼之间等难以铺设有线电缆的区域，无线通信可以成为一种高效铺设网络的方法。固定式通信的一般用

途是在互相都能确定对方位置的收发双方之间架设无线链路[①]。

移动式的无线通信（**移动式通信**）指的是参与通信的无线设备中任意一方或双方都可以移动的情况下的无线通信。人们所持有的各种设备就需要进行这种类型的通信。在可移动设备等进行通信时，**基站**（base station，访问点）为这些设备提供了访问互联网的入口。

无线特有的难点 对 TCP 的影响

由于无线通信链路的变化较小，所以固定式通信相对比较稳定，但移动式通信由于基站和终端设备之间可能出现被建筑物遮挡的情况，这导致接收功率显著下降，所以有时会出现无法通信的问题（图 1.15）。

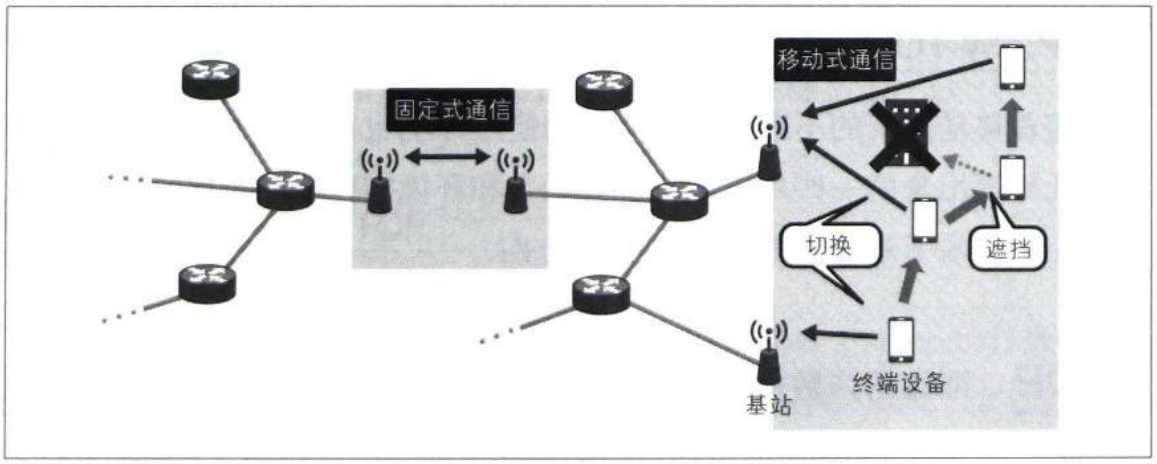

图 1.15 两种无线通信环境

此外，附近区域如果存在使用同一频率电波的通信，就会发生电磁波干涉现象，导致通信之间互相干扰。因此，这也是除了网络拥塞之外的导致丢包的重要原因。

无线通信具备根据接收功率等状态的变化自适应地调整传输速率的能力。由于此特性的影响，通信的传输速率会不时发生变化，这可能成为影响 TCP 运行的重要原因。

不仅如此，如前文所述，想要使无线通信从支持短距离通信进化到支持任意距离的通信，需要在所有区域设置大量基站，同时还要让终端

① 虽然传感器等设备和部分物联网应用终端（例如智能电表）也是位置固定的，但是请把它们想象为具有前面所述用途的设备。

设备能够随着自身位置的变化而改变接入的基站。这个过程称为**切换**（handover）。一旦发生切换，则意味着通信链路发生了变化。显而易见，这也会对TCP的运行产生影响。

从上述内容可以看出，原本以有线通信为前提而设计的TCP协议，在无线通信的过程中可能无法按照设计运作。也就是说，人们还需要根据无线通信的特点调整TCP。下一节将以W-TCP为例对此进行介绍。

1.6 面向特定用途的协议

RUDP、W-TCP、SCTP和DCCP

随着移动通信的不断发展，网络的使用环境也在不断变化。本节将介绍近几年提出来的一些用于适应各种场合和环境的传输层协议。第7章会详细介绍着眼于未来的TCP的新发展方向。

RUDP 可靠用户数据报协议

近年来，智能手机上的在线游戏逐渐普及。无论是协作游戏还是对战游戏，手机用户都可以与全世界的其他用户进行联机。在线游戏一旦断线，可是非常不妙的情况，特别是需要实时显示对手的游戏动作的动作类游戏。因此，这类应用的数据传输协议必须在保证高可靠性的同时，提供强实时性。

换句话说，这意味着要把TCP与UDP二者的特性结合起来。但是如前文所述，它们的特性是相反的。因此，基于UDP协议的RUDP（Reliable User Datagram Protocol）协议被设计出来[1]。除了确保可靠性以外，它还被添加了如下特性。

---

① 截至笔者执笔时，RUDP还没有进行标准化提案，并非正式的标准，所以仍停留在1999年互联网草案的状态下。

- 序列号
- ACK
- 重传机制
- 流量控制

如前文所述，流量控制功能用于根据接收方的情况调整发送方的窗口大小 *swnd*。

RUDP 在功能特性上与 TCP 相近，不过在实现上比 TCP 系统开销更小。TCP 首部的大小是 20 到 60 字节，而 RUDP 的首部大小只有 12 字节。在线游戏是输入和响应在网络上频繁往返的全双工通信。利用这一特性，RUDP 也为待发送数据添加了 ACK 的功能（也就是说，发送数据的同时也发送 ACK）。这降低了首部的系统开销，因此提高了效率。如果在此基础上加入重传机制，实时性和可靠性便可以两手兼得。

W-TCP 无线配置 TCP

W-TCP（Wireless Profiled TCP）制定在 TCP over 2.5G and 3G Wireless Networks（第 2.5 代和第 3 代无线网络上的 TCP，RFC 3481）文档中。它指的是为第 2.5 代到第 3 代移动电话通信专门优化的 TCP。如 1.5 节所述，无线通信会发生丢包、传输速率变化等问题。其原因不同于有线通信，主要是建筑物的遮挡、反射和终端设备的移动等情况导致电波的传输链路发生大幅变化。

因此，人们在无线终端设备与所连接的服务器之间设置了网关[①]（gateway），并在无线设备与网关之间使用 W-TCP，在网关与所连接的服务器之间使用普通的 TCP。这种方法称为 split-TCP，记载在 RFC 2757 中。使用此方法后，哪怕终端设备与网关的无线通信线路之间频繁进行重传，网关与内容服务器之间也不会产生不必要的重传流量。换句话说，通过将可靠性不同的两个网络分开，避免了可靠性低的无线网络成为可靠性

① 以防万一，这里要说明一下：基站并不一定是网关。

高的有线网络的累赘。

不仅如此，为了提高无线传输线路的效率，人们还对以下若干个TCP参数进行了调优。

- **窗口扩大选项（RFC 1323）**
  窗口大小原先最大为64 KB，通过窗口扩大选项（window scale option）可以调整到更大的值。
- **增大最小窗口大小（RFC 2414）**
  窗口大小的初始值一般是1个TCP报文段大小，但现在可以将其设置为2个以上的大小。这个优化对于高速传输线路很有效。
- **SACK（Selective Acknowledgement，选择性应答，RFC 2018）**
  通常情况下，在TCP检测到TCP报文段丢失的场景中，数据发送方会重复收到ACK应答，被要求重传对应序列号的TCP报文段数据。重传发生在其收到多次重复的ACK之后，因此一旦有多个TCP报文段丢失，那么TCP想要检测到第2个以后的报文段数据的丢失就会花费不少时间。而SACK支持在出现多个报文段丢失时，显式地要求重传相应的TCP报文段数据。通过SACK，TCP可以快速、准确地实现重传。
- **基于BDP（Bandwidth Delay Product，带宽时延积）的最佳窗口控制**
  基于BDP这一指标确定最佳的窗口大小。顾名思义，BDP是带宽和往返时延（即*RTT*）的乘积。从3G的终端设备到基站的上行带宽是64 Kbit/s，而相对的下行带宽是384 Kbit/s。将这个数据与*RTT*相乘，即可得到*BDP*。*BDP*的单位是数据量，换句话说，就是收发双方需要提前准备好的缓冲区大小。如果能使用比*BDP*更大的缓冲区和窗口大小，就可以实现带宽无闲置且更高效的通信。
- **时间戳选项（RFC 1323）**
  在宽带通信中，序列号容易在短时间内循环一遍，这样会导致不同的数据在序列号上重复。时间戳的作用便是和序列号一起作为索引数据，来解决序列号重复的问题。另外，*RTT*的值是在窗口大小每

次更新后，由 TCP 重新计算或更新从报文段发送到收到 ACK 的时间后得来的。但无线链路中线路变化快，*RTT* 也会在短时间内出现大幅变动，所以还需要更精细的追踪计算。对于重传的报文段数据，TCP 原本是不会计算 *RTT* 的（详见第 3 章），但如果使用了时间戳功能，就可以计算重传报文段的 *RTT* 数据，进而获取更多的 *RTT* 样本数据，提高计算出的 *RTT* 值的平滑精度。

- **检测链路上的 *MTU* 大小（RFC 1191）**

  此功能用于在需要经由多台设备收发数据的情况下，检测收发数据过程中 ***MTU***（Maximum Transmission Unit，最大传输单元）的最小值，并使用这个最小值发送数据。这是为了规避因较大的数据在传输过程中被分割为更小的单位而发生的效率降低问题（详见第 3 章）。

- **ECN（Explicit Congestion Notification，显式拥塞通知，RFC 2481）**

  这是一个提前获取拥塞状态，并通知发送方减小窗口大小的功能。它能在网络链路中的路由器等设备发生拥塞时，就将拥塞情况写入 TCP 首部的 ECE（ECN-Echo）字段告知发送方。利用这一功能，TCP 就可以在重传计时器超时之前获知拥塞状态。在无线通信中，利用此功能也可以尽早地检测出通信环境的恶化问题，并通过控制传输速率等快速处理。

从以上描述可以看出，W-TCP 针对复杂的无线通信环境进行了专门的优化，下了很多功夫。在无线通信中，下层的数据链路层也具备重传控制机制。换句话说，无线通信通过多个重传机制协同工作实现了更为稳定的通信[①]。

① 从第 3 代开始，移动电话系统都搭载了 HARQ（Hybrid Automatic Repeat reQuest，混合自动重传请求）功能。与 TCP 实现的端到端的重传控制有所不同，HARQ 基于数据链路层，在单个无线链路中运作。接收方检测到帧数据丢失，就会发送请求重传的消息，这与 TCP 相似。发送方要做的不是重传丢失的帧数据，而是只发送部分数据和部分解码所需的信息。接收方将第一次收到的数据与重传的数据组合起来，便可以完成解码，还原数据。它通过减少发送的数据量，实现了高效的无线传输。

SCTP 流控制传输协议

SCTP（Stream Control Transmission Protocol，RFC 4960）不仅和TCP一样提供高可靠性和顺序无误的数据传输，还像UDP一样面向消息而设计，能够确保消息之间有边界[①]。该技术的设计初衷是用来实现IP网络下的电话网信令，但最终则是作为一个通用性很高的传输层协议，于2000年被制定出来的。现在，SCTP被用作LTE（Long Term Evolution，长期演进）等第4代通信中的控制信号的传输层协议。

SCTP主要有以下特征。

- **面向消息**

  TCP协议把待发送的数据按照字节流的形式进行传输，忽略了消息的边界。UDP则是数据报形式的协议，因此消息可以保留边界，但无法保证消息的顺序。SCTP按照“块”（chunk）处理消息，经过改良后既可以保证消息的顺序，又能保留消息的边界。

- **多宿主（multihoming）**

  多宿主是指通过同时使用多个网络接口提高可用性的一项技术。举例来说，构建由有线LAN与无线Wi-Fi组成的多重通信链路[②]，就可以在有线LAN可用时使用有线LAN，在其他情况下使用Wi-Fi通信。

- **多流（multistreaming）**

  在同一个联合（association）中同时支持多个传输流。通过此技术可以将控制信息与数据分离，提高控制信息的响应速度。

- **链路有效性检测**

  对于一段时间内没有通信过的目的地址，使用**心跳**（heartbeat）包来检测链路是否可用。定时发送心跳包，根据ACK的返回情况来判断当前链路是否可用。

---

① TCP是流式的，不同消息直接混在一起，没有边界，而UDP是数据报式的，不同的消息是分开的。——译者注

② 在SCTP中，这称为联合（assosiation）。

- **初始化（initiation，建立连接）的优化**

  恶意用户伪造 IP 地址，发送大量的 SYN 数据包（SYN flood，SYN 洪水），导致连接资源枯竭，这便是 DoS（Denial of Service）攻击。以往的 TCP 协议对这种攻击比较无力，SCTP 协议则将连接建立时的握手次数改为 4 次（4-way）[①]，同时加入 Cookie，实现了对这种攻击的防御。不过，SCTP 在断开连接时是 3 次挥手[②]。

DCCP 数据报拥塞控制协议

人们设计 DCCP（Datagram Congestion Control Protocol，RFC 4340）的目的是缓解 UDP 网络拥塞问题。UDP 直接将数据流量发送到网络中，因此很容易引起网络拥塞。DCCP 使用类似 TCP 的基于流量的方法来处理网络拥塞。它虽然使用 ACK，但只是为了检测拥塞，重传并不是主要的目的。不过，它也支持增加重传功能。不仅如此，它还支持 ECN 功能，所以可以实现更为灵活的拥塞控制。显而易见，对于同时追求可靠性和低时延的应用程序来说，DCCP 非常有用。

1.7 小结

本章总览了网络中的各种通信协议，又介绍了作为本书主题的 TCP 的基本功能。从下一章开始，本书将上至历史发展，下至详细功能特性，结合实际运行模拟详细地介绍 TCP 的方方面面。此外，介绍传输层以外的 TCP/IP 协议群的优秀图书有很多，感兴趣的读者请务必参考学习。

① 原本的 TCP 建立连接是 3 次握手。详见第 3 章。

② 原本的 TCP 断开连接是 4 次挥手。