计算机网络

自顶向下方法

第一章 计算机网络和因特网

1.1 构成

主机（或端系统），通信链路，分组交换机，路由器，ISP（Internet Service Provider），协议

协议（protocol）定义了在两个或多个通信实体之间交换的报文的格式和顺序，以及报文发送和/或接收一条报文或其他事件所采取的动作。

1.2 网络边缘

端系统也称为主机，因为它们容纳（即运行）应用程序。主机有时进一步被划分为两类：客户机和服务器。

### 1.2.1 接入网

接入网是指将端系统物理连接到其边缘路由器的网络。边缘路由器是端系统到任何其他远程端系统的路径上的第一台路由器。

家庭接入:DSL、电缆、FTTH、拨号和卫星

今天，宽带住宅接入有两种最流行的类型：**数字用户线**（Digital Subscriber Line，DSL）和**电缆**。

住户通常从提供本地电话接入的本地电话公司处获得**DSL因特网接入**。每个用户的DSL调制解调器使用现有的电话线（即双绞铜线）与位于电话公司的本地中心局中的数字用户线接入复用器（DSLAM）交换数据。

**电缆因特网接入**利用了有线电视公司现有的有线电视基础设施。光缆将电缆头端连接到地区枢纽，从这里使用传统的同轴电缆到达各家各户和公寓。因为这个系统中应用了光纤和同轴电缆，因此被称为**混合光纤同轴**（Hybrid Fiber Coax， HFS）系统。

光纤到户（Fiber To The Home，FTTH）速率更高，从本地中心局直接到家庭提供了一条光纤路径。

企业（和家庭）接入：以太网和WiFi

广域无线接入：3G和LTE

### 1.2.2 物理媒体

物理媒体分两种类型：导引型媒体和非导引型媒体。

对于导引型媒体，电波沿着固体媒体前行，如光缆、双绞铜线或同轴电缆。

对于非导引型媒体，电波在空气或外层空间中传播，例如在无线局域网或数字卫星频道中。

双绞铜线

双绞线有两根绝缘的铜线组成，每根大约1mm粗，以规则的螺旋状排列着。这两根线被绞合起来，以减少临近类似的双绞线的电气干扰。

同轴电缆

同轴电缆由两个铜导体组成，但是这两个导体是同心的而不是并行的。借助于这种结构及特殊的绝缘体和保护层，同轴电缆能够达到较高的数据传输速率。同轴电缆能被用作导引型共享媒体，特别是，许多端系统能够直接与该电缆相连，每个端系统能接收由其他端系统发送的内容。

光纤

光纤是一种细而柔软的、能够导引光脉冲的媒体，每个脉冲表示一个比特。

陆地无线电信道

无线电通信承载电磁频谱中的信号。它不需要安装物理线路，并具有穿越墙壁、提供与移动用户的连接以及长距离承载信号的能力。无线电信道极大的依赖于传播的环境和信号传输的距离。

卫星无线电信道

一颗通信卫星连接地球上的两个或多个微波发射器/接收器，它们被称为地面站。该卫星在一个频段上接收传输，使用一个转发器再生信号，并在另一个频率上发射信号。

通信中常使用两类卫星：同步卫星和近地轨道卫星。

## 1.3 网络核心

通过网络链路和交换机移动数据有两种基本方法：电路交换和分组交换

### 1.3.1 分组交换

在各种网络应用中，端系统彼此交互报文。报文能够包含协议设计者需要的任何东西。报文可以执行一种控制功能，也可以包含数据。

为了从源端系统向目的端系统发送一个报文，源将长报文划分为较小的数据块，称之为**分组**。在源和目的地之间，每个分组都通过通信链路和分组交换机传送。（交换机主要有两类：**路由器**和**链路层交换机**。）分组以等于该链路**最大传输速率**的速度传输通过通信链路。因此，如果某源端系统或分组交换机经过一条链路发送一个L比特的分组，链路的传输速率为R比特/秒，则传输该分组的时间为L/R秒。

存储转发传输

**存储转发传输**是指在交换机能够开始向输出链路传输该分组的第一个比特之前，必须接收到整个分组。

排队时延和分组丢失

**输出缓冲**：每台分组交换机有多条链路与之相连。对于每条相连的链路，该分组交换机具有一个**输出缓存**（也称为**输出队列**），它用于存储路由器准备发往那条链路的分组。

**排队时延**：如果到达的分组需要传输到某条链路，但发现该链路正忙于传输其他分组，该到达分组必须在输出缓存中等待，由此产生的时延为排队时延。

**分组丢失**：因为缓存空间的大小有限，一个到达的分组可能发现该缓存已被其他等待传输的分组完全充满了。在此情况下，将出现分组丢失（丢包），到达的分组或已经排队的分组之一将被丢弃。

转发表和路由选择协议

转发表：每台路由器具有一个转发表，用于将目的地址（或目的地址的一部分）映射成为输出链路。

路由选择协议：一个路由选择协议可以决定从每台路由器到每个目的地的最短路径，并使用这些最短路径结果来配置路由器中的转发表。

### 1.3.2 电路交换

在电路交换网络中，在端系统通信会话期间，预留了端系统间沿路径通信所需要的资源（缓存，链路传输速率）。

在分组交换网络中，这些资源则不是预留的；会话的报文按需使用这些资源，其后果是可能不得不等待（即排队）进入通信线路。

在电路交换网络中，必须在发送方和接收方之间建立一条连接。这是一条名副其实的链接，因为在沿着发送方和接收方之间的路径上都将为该连接维护连接状态，即一条**电路**。当网络创建这种电路时，它也在连接期间在该网络链路上预留了恒定的传输速率（表示为每条链路传输容量的一部分）。既然已经为该发送方—接收方连接预留了带宽，则发送方能够以**确保的恒定速率**向接收方传送数据。

电路交换网络中的复用

链路中的电路时通过频分复用（Frequency-Division Multiplexing，FDM）或时分复用（Time-Division Multiplexing，TDM）来实现的。

对于FDM，链路的频谱由跨越链路创建的所有连接共享。特别是，在连接期间链路为每条连接专用一个频段。这个频段的带宽通常为4kHz。该频段的宽度称为带宽。

对于一条TDM链路，时间被划分为固定时间的帧，并且每个帧又被划分为固定数量的时隙。当网络跨越一条链路创建一条连接时，网络在每个帧中为该连接制定一个时隙。这些时隙专门由该连接单独使用，一个时隙（在每个帧内）可用于传输该连接的数据。

分组交换与电路交换的对比

分组交换优缺点（引用他人的观点）：

优点：1）它提供了比电路交换更好的带宽共享；

2）它比电路交换更简单、更有效，实现成本更低。

缺点：分组交换不适合实时服务（例如，电话和视频会议），因为他的端到端时延是可变的和不可预测的（主要是因为排队时延的变动和不可预测所致）。

**作者总结：**

1）分组交换的性能能够优于电路交换的性能。

2）电路交换不考虑需求，而预先分配了传输链路的使用，这使得已分配而并不需要的链路时间并未被利用。另一方面，分组交换按需分配链路使用。链路传输能力将在所有需要在链路上传输分组的用户之间逐步地被共享。

### 1.3.3 网络的网络

今天的因特网是一个网络的网络，其结构复杂，由十多个第一层ISP和数十万个较低层ISP组成。较低层ISP与较高层ISP相连，较高层ISP彼此互联。用户和内容提供商是较低层ISP的客户，较低层ISP是较高层ISP的客户。

## 1.4 分组交换网中的延迟、丢包和吞吐量

### 1.4.1 分组交换网中的时延概述

时延的类型

（1）处理时延

检查分组首部和决定将该分组导向何处所需要的时间是处理时延的一部分。处理时延也可能包括其他因素：如检查比特级别的差错所需要的时间，该差错出现在从上游节点向路由器A传输这些分组比特的过程中。

（2）排队时延

在队列中，当分组在链路上等待传输时，它经受排队时延。一个特定分组的排队时延长度将取决于先期到达的正在排队等待向链路传输的分组数量。

（3）传输时延

假定分组以先到先服务方式传输，仅当所有已经到的分组被传输后，才能传输刚到达的分组。用L比特表示该分组的长度，用R bps表示从路由器A到路由器B的链路传输速率，则传输时延是L/R。这是将所有分组的比特推向链路（即传输）所需要的时间。

（4）传播时延

一旦一个比特被推向链路，该比特需要向路由器B传播。从该链路的起点到路由器B传播所需要的时间是传播时延。该比特以链路的传播速率传播。该传播速率取决于该链路的物理媒体（即光纤、双绞铜线等），其速率范围是2 x 108~3 x 108m/s。

### 1.4.2 排队时延和丢包

流量强度

排队时延大小很大程度取决于流量到达该队列的速率、链路的传输速率和到大流量的性质，即流量是周期性到达还是以突发形式到达。

令a表示分组到达队列的平均速率（单位为分组/秒），传输速率为Rbps，所有分组大小均为L比特。比率La/R称为流量强度，它在估计排队时延的范围方面经常起到重要的作用。如果La/R>1，则比特到达队列的平均速率超过从该队列传输出去的速率。在这种不幸的情况下，该队列趋向于无限增加，并且排队时延将趋向于无穷大！因此：设计系统时流量强度不能大于1。

丢包

随着流量强度接近于1，排队时延并不真正趋向无穷大。

相反，到达的分组将发现一个满的队列。由于没有地方存储这个分组，路由器将丢弃该分组，即该分组会丢失。

### 1.4.3 端到端时延

Traceroute

用于计算端到端时延，其原理如下：

假定在源和目的地之间有N-1台路由器。源将向网络发送N个特殊的分组，其中每个分组地址指向最终目的地。这N个特殊分组标识为从1到N，第一个分组标识为1，第N个分组标识为N。当第N台路由器接收到标识为n的第n个分组时，该理由起路由器不是向它的目的地转发该分组，而是想源回送一个报文。当目的主机接收第N个分组时，它也会向源返回一个报文。该源记录了从它发送一个分组到它接收到对应返回报文所经历的时间；它也记录了返回该报文的路由器（或目的主机）的名字和地址。

### 1.4.4 计算机网络中的吞吐量

瞬时吞吐量：主机在任何时间瞬间接收到文件的速率。

平均吞吐量：主机接收到所有F比特用去T秒，则平均吞吐量为F/T bps。

## 1.5 协议层次及其服务模型

### 1.5.1 分层的体系结构

1.协议分层

各层的所有协议被称为**协议栈**。因特网的协议栈由五个层次组成：物理层、链路层、网络层、运输层和应用层。

（1）应用层

应用层是网络应用程序及它们的应用层协议存留的地方。因特网的应用层协议包括：HTTP、SMTP、FTP和DNS等。

应用层协议分部在多个端系统上，而一个端系统中的应用程序使用协议与另一个端系统中的应用程序交换信息分组。

应用层的信息分组称为**报文**（message）。

（2）运输层

因特网的运输层在应用程序端点之间传送应用层报文。有两种因特网运输协议：TCP和UDP。

TCP向它的应用程序提供了面向连接的服务。这种服务包括了应用层报文向目的地的确保传递和流量控制（即发送方/接收方速率匹配）。TCP也将长报文划分为短报文，并提供拥塞控制机制。

UDP协议向它的应用程序提供无连接服务。这是一种不提供不必要服务的服务，没有可靠性，没有流量控制，也没有拥塞控制。

运输层分组称为报文段（segment）。

（3）网络层

网络层负责将网络层分组—数据报（datagram）从一台主机移动到另一台主机。在一台源主机中的因特网运输层协议（TCP或UDP）向网络层递交运输层报文段和目的地址。

（4）链路层

因特网的网络层通过源和目的地之间的一系列路由器路由数据报。为了将分组从一个节点（主机或路由器）移动到路径上的下一个节点，网络层必须依靠该链路层的服务。特别是在每个节点，网络层将数据报下传给链路层，链路层沿着路径将数据报传递给下一个节点。在该下一个节点，链路层将数据报上传给网络层。

由链路层提供的服务取决于应用于该链路的特定链路层协议。如以太网、WiFi和PPP协议等。

链路层分组称为帧。

（5）物理层

链路层的任务是将整个帧从一个网络元素移动到邻近的网络元素，而物理层的任务是将该帧中的一个个比特从一个节点移动到下一个节点。在这层中的协议任然是链路相关的，并且进一步与该链路（如双绞铜线，单模光纤）的实际传输媒体相关。

2. OSI模型

开发系统互联模型（OSI）包括七层协议：应用层、表示层、会话层、运输层、网络层、链路层和物理层。

表示层的作用是使通信能够解释交换数据的含义。这些服务包括数据压缩和数据加密以及数据描述（这使得应用程序不必担心在个台计算机中表示/存储的内部各式不同的问题）。

会话层提供了数据交换的定界和同步功能，包括了建立检查点和恢复方案的方法。

### 1.5.2 封装

在发送主机端，一个**应用层报文**被发送给运输层。在最简单的情况下，运输层收取到报文并附上附加信息，该首部将被接收端的运输层使用。应用层报文和运输层首部信息一道构成了**运输层报文段**。运输层报文段因此封装了应用层报文。

附加的信息也许包括了下列信息：允许接收端运输层向上向适当的应用程序交付报文的信息；差错检测位信息，该信息让接收方能够判断报文中的比特是否在途中已被改变。

运输层则向网络层传递该报文段，网络层增加了如源和目的端系统地址等网络层首部信息，生成了**网络层数据报**。

该数据报接下来被传递给链路层，链路层增加它自己的链路层首部信息并生成链路层帧。

所以，在每一层，一个分组具有两种类型的字段：**首部字段**和**有效载荷字段**。有效载荷通常是来自上一层的分组。

第二章 应用层

## 2.1 应用层协议原理

### 2.1.1 网络应用程序体系结构

应用程序体系结构由应用程序研发者设计，规定了如何在各种端系统上组织该应用程序。两种主流体系结构：**客户-服务器体系结构**、**对等（P2P）体系结构**。

在**客户-服务体系结构**中，有一个总是打开的主机成为服务器，他服务于来自许多其他称为客户的主机的请求。

在一个P2P体系结构中，对位于数据中心的专用服务器有最小的依赖。相反，应用程序在间断连接的主机对之间使用直接通信，这些主机对被称为**对等方**。

### 2.1.2 进程通信

在同一台主机上的进程间的通信方式有：管道，消息队列，信号量，共享内存

在两个不同端系统上的进程，通过跨越计算机网络交换报文而相互通信。发送进程生成并向网络中发送报文；接收进程接收这些报文并可能通过回送报文进行响应。

1. 客户和服务器进程

在一对进程之间的通信会话场景中，发起通信（即在该会话开始时发起与其他进程的联系）的进程被标识为**客户**，在会话开始时等待联系的进程是**服务器**。

2. 进程与计算机网络之间的接口

进程通过一个称为**套接字**的软件接口向网络发送报文和从网络接收报文。套接字是同一台主机内应用层与传输层之间的接口。由于该套接字是建立网络应用程序的可编程接口，因此套接字也称为应用程序和网络之间的**应用程序编程接口**。

应用程序开发者可以控制套接字在应用层端的一切：①选择运输层协议；②也许能设定几个运输层参数，如最大缓存和最大报文段长度。

3. 进程寻址

标识接收进程，需要定义两种信息：①主机的地址；②在目的主机中指定接收进程的标识符。

在因特网中，主机由其IP地址标识，进程由端口号标识。

常见的应用程序对应的端口号如下：

表2.1 常见服务端口号

|  |  |
| --- | --- |
| 服务名称 | 端口号 |
| FTP | 21 |
| SSH、SCP | 22 |
| Telnet | 23 |
| SMTP | 25 |
| DNS | 53 |
| HTTP | 80 |
| POP3 | 110 |
| HTTPS | 443 |

### 2.1.3 可供应用程序使用的运输服务

1.可靠数据传输

定义：确保应用程序的一端发送的数据正确、完全地交付给该应用程序的另一端。

运输层协议能够潜在地向应用程序提供的一个重要服务是进程到进程的可靠数据传输。当一个运输协议提供这种服务时，发送进程只要将其数据传递进套接字，就可以完全相信该数据将能无差错地到达接收进程。

2. 吞吐量

在沿着一条网络路径上的两个进程之间的通信会话场景中，可用吞吐量就是发送进程能够向接收进程交付比特的速率。因为其他会话将共享沿着该网络路径的带宽，并且因为这些会话将会到达和离开，该可用吞吐量将**随时间波动**。

具有吞吐量要求的应用程序被称为**带宽敏感的应用**。

**弹性应用**能够根据当时可用的带宽或多或少地利用可供使用的吞吐量。

3. 定时

运输层协议也能提供定时保证。一个保证的例子如：发送方注入进套接字中的每个比特到达接收方的套接字不迟于100ms。

4. 安全性

最后，运输协议能够为应用程序提供一种或多种安全性服务。例如，在发送主机中，运输协议能够加密由发送进程传输的所有数据，在接收主机中，运输层协议能够在将数据交付给接收进程之前解密这些数据。

### 2.1.4 因特网提供的运输服务

因特网为应用程序提供两个传输层协议，即TCP和UDP。

1. TCP服务

TCP服务模型包括面向连接服务和可靠数据传输服务。

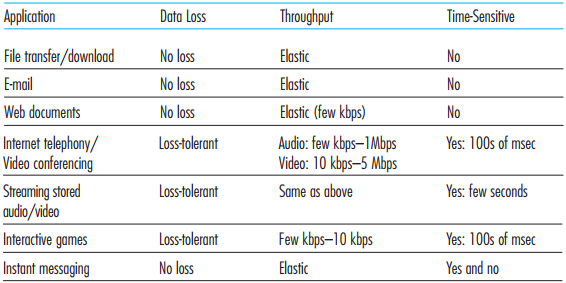


图2-1 选择的网络应用的要求

* **面向连接的服务**：在应用层数据报文开始流动之前，TCP让客户和服务器互相交换运输层控制信息。这个所谓的握手过程提醒客户和服务器，让它们为大量分组的到来做好准备。在握手阶段后，一个TCP连接就在两个进程的套接字之间建立了。这条连接是全双工的，即连接双方的进程可以在此连接上同时进行报文收发。当应用程序结束报文发送时，必须拆除该连接。
* **可靠的数据传输服务**：通信进程能够依靠TCP，无差错、按适当顺序交付所有发送的数据。当应用程序的一端将字节流传进套接字时，它能够依靠TCP将相同的字节流交付给接收方的套接字，而没有字节的丢失和冗余。

2. UDP服务

UDP是一种不提供不必要服务的轻量级运输协议，它仅提供最小服务。UDP是无连接的，因此在两个进程通信前没有我收过程。UDP协议提供一种不可靠数据传输服务，当进程将一个报文发送进UDP套接字时，UDP并不保证该报文将到达接收进程。不仅如此，到达接收进程的报文也可能是乱序到达的。

3. 因特网运输协议所不提供的服务

今天的因特网通常能够为时间敏感应用提供满意的服务，但它不提供任何定时或带宽保证。

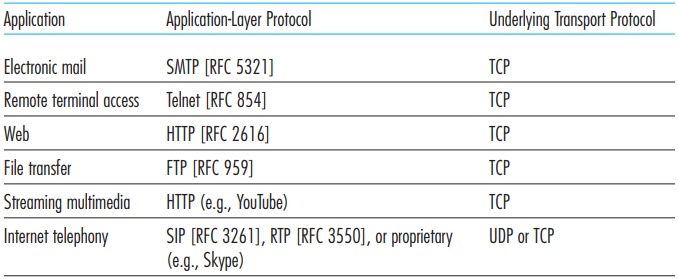


图2-2 流行的网络应用及其应用层协议和支撑的运输协议

### 2.1.5 应用层协议

应用层协议定义了运行在不同端系统上的应用程序进程如何相互传递报文。特别是定义了：

* 交换的报文类型，例如请求报文和响应报文
* 各种报文类型的语法，如报文中的各个字段及这些字段是如何描述的
* 字段的语义，即这些字段中的信息的含义
* 确定一个进程何时以及如何发送报文，对报文进行响应的规则。

## 2.2 Web和HTTP

### 2.2.1 HTTP概况

Web的应用层协议是**超文本传输协议**（HyperText Transfer Protocol， HTTP），它是Web的核心。

Web页面是由对象组成的。一个对象只是一个文件，如一个HTML文件、一个JPEG图形，一个Java小程序这样的文件，切它们可通过一个URL地址寻址。多数Web页面含有一个HTML基本文件以及几个引用对象。

HTTP使用TCP作为它的支撑运输协议。HTTP客户首先发起一个与服务器的TCP连接。一旦连接建立，该浏览器和服务器进程就可以通过套接字接口访问TCP。

**无状态协议**：服务器向客户发送被请求的文件，而不存储任何关于该客户的状态信息。假如某个特定的客户在短短的几秒内两次请求同一个对象，服务器并不会因为刚刚为客户提供了该对象就不再作出反应，而是重新发送该对象。

为什么说HTTP是无状态的协议呢？因为它的每个请求都是完全独立的，每个请求包含了处理这个请求所需的完整的数据（如主机地址，鉴权信息等），发送请求不涉及到状态变更。

### 2.2.2 非持续连接和持续连接

**非持续连接**：每个请求/响应对是经一个单独的TCP连接发送；

**持续连接**：所有的请求及其响应经相同的TCP连接发送。

HTTP默认情况下使用持续连接。

1. 采用非持续连接的HTTP

服务器向客户传送一个Web页面的步骤。假设该页面包含一个HTML基本文件和10个JPEG图形，并且这11个对象位于同一台服务器上。假设HTML文件的URL为：http://www.someSchool.edu/someDept/home.index。

1）HTTP客户进程在端口号80发起一个到服务器www.someSchool.edu的TCP连接，该端口号是HTTP的默认端口。在客户和服务器上分别有一个套接字与该链接相关联。

2）HTTP客户经它的套接字向该服务器发送一个HTTP请求报文。请求报文中包含路径名/someDept/home.index。

3）HTTP服务器进程经它的套接字接收该请求报文，从其存储器检索出对象www.someSchool.edu/someDept/home.index，在一个HTTP响应报文中封装对象，并通过其套接字向客户发送相应报文。

4）HTTP服务器进程通知TCP断开该TCP连接。（但是直到TCP确认客户已经完整地收到响应报文为止，它才会实际终端连接。）

5）HTTP客户接受响应报文，TCP连接关闭。该报文指出封装的对象是一个HTML文件，客户从响应报文中提取出该文件，检查该HTML文件，得到对10个JPEG图形的引用。

6）对每个引用的JPEG图形对象重复前4个步骤。

事实上，用户能够配置现代浏览器来控制连接的并行度。在默认方式下，发部分浏览器打开5~10个并行的TCP连接，每条连接处理一个请求响应事务。

往返时间（Round-Trip Time，RTT）：该时间是指一个短分组从客户到服务器然后再返回客户所花费的时间。RTT包括传播时延、分组在中间路由器和交换机上的排队时延以及分组处理时延。

TCP连接涉及“三次握手”过程，即客户向服务器发送一个小TCP报文段，服务器用一个小TCP报文段作出确认和响应，最后，客户向服务器返回确认。三次握手中前两个部分耗费一个RTT时间。完成了三次握手的前两个部分后，客户结合三次握手的第三部分向该TCP发送一个HTTP请求报文。一旦该请求报文到达服务器，服务器就在该TCP连接上发送HTTP文件。该HTTP请求/响应用去了另一个RTT。

TCP连接细节见第三章。

2. 采用持续连接的HTTP

非持续连接有一些缺点：

1）必须为每一个请求的对象建立和维护一个全新的连接。对于每个这样的连接，在客户和服务器中都要分配TCP的缓冲区和保持TCP变量，这个Web服务器带来了严重的负担。

2）每一个对象经受两倍RTT的交付时延，即一个RTT用于创建TCP，另一个RTT用于请求和接收一个对象。

在采用HTTP1.1持续连接的情况下，服务器在发送响应后保持该TCP连接打开。在相同的客户与服务器之间，后续的请求和响应报文能够通过相同的连接进行传送。一般来说，如果一条连接经过一段时间间隔（一个可配置的时间间隔）仍未被使用，则HTTP服务器就关闭该连接。

HTTP的**默认模式**是使用带**流水线的持续连接**。

### 2.2.3 HTTP报文格式

1. HTTP请求报文

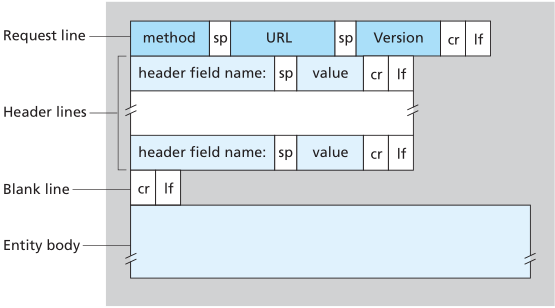


图2-3 HTTP请求报文的通用格式

HTTP请求报文的第一行叫**请求行**，其后继的行叫做**首部行**。

**1）请求行**

请求行有3个字段：方法字段、URL字段和HTTP版本字段。

方法字段的取值包括：GET、POST、HEAD、PUT和DELETE。

**2）首部行**

Host：指明对象所在的主机。

Connection：值close要求服务器在发送完被请求的对象后就关闭这条连接。

User-agent：指明用户代理，即向服务器发送请求的浏览器类型。

Accept-language：表示用户想要得到该对象的法语版本。

**3）实体体**

位于首部行（和附加的回车和换行）后。使用GET方法时实体体为空，而使用POST方法时才使用实体体。

HEAD方法类似于GET方法。当服务器收到一个使用HEAD方法的请求时，将会用一个HTTP报文进行响应，但是并不返回请求对象。应用程序开发者常用HEAD方法进行调试跟踪。

PUT方法常与Web发行工具联合使用，它允许用户上传对象到指定的Web服务器上指定的路径（目录）。

DELETE方法允许用户或者应用程序删除Web服务器上的对象。

2. HTTP响应报文

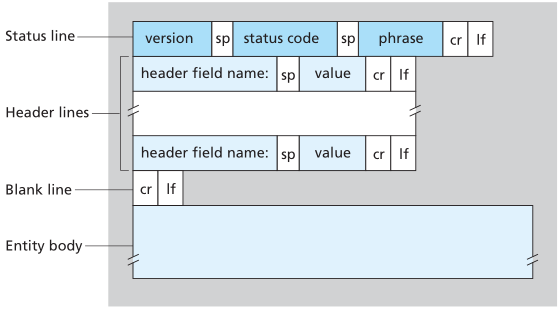


图2-4 HTTP 响应报文的通用格式

响应报文包含三部分：初始状态行、6个首部行、实体体。

**1）初始状态行**

协议版本字段、状态码和响应状态信息。

常见的状态码及含义见[附录1](#_附录1_HTTP常见状态码及含义)

**2）首部行**

Connection：close首部行告诉用户，发送完报文后将关闭该TCP连接。

Date：指示服务器产生并发送该响应报文的日期和时间。这个时间不是指对象创建或者最后修改的时间，而是服务器从它的文件系统中检索到该对象，将该对象插入响应报文并发送该响应报文的时间。

Server：指示发送该报文的服务器类型，类似于请求报文中的User-agent首部行。

Last-modified：指示了对象创建或者最后修改的日期和时间。

Content-Length：指示了被发送对象中的字节数。

Content-Type：指示了实体体中的对象类型。

**3）实体体**

包含了报文的主要部分，即它包含了所请求的对象本身。

### 2.2.4 用户与服务器的交互：cookie

目的：Web站点希望能够识别用户，可能是因为服务器希望限制用户的访问，或者因为它希望把内容与用户身份联系起来。

cookie技术有四个组件：①在HTTP响应报文中的一个cookie首部行；②在HTTP请求报文中的一个cookie首部行；③在用户端系统中保留有一个cookie文件，并由用户的浏览器进行管理；④位于web站点的一个后端数据库。

服务器用一个包含Set-cookie的首部行设置cookie。

### 2.2.5 Web缓存

Web缓存器也叫代理服务器，它是能够代表初始Web服务器来满足HTTP请求的网络实体。Web服务器有自己的磁盘存储空间，并在存储空间中保存最近请求过的对象的副本。

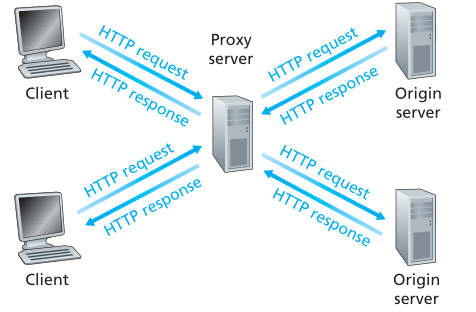


图2-5 客户通过Web缓存器请求对象

**Web缓存器的工作原理**：

假设浏览器请求对象http://www.someschool.edu/campus.gif

1）浏览器创建一个到Web缓存器的TCP连接，并向Web缓存器中的对象发送一个HTTP请求。

2）Web缓存器进行检查，看看本地是否存储了该对象的副本。如果有，Web缓存器就想客户浏览器用HTTP响应报文返回该对象。

3）如果Web缓存器中没有该对象，它就打开一个与该对象的初始服务器的TCP连接。Web缓存器则在这个缓存起到服务器的TCP连接上发送一个对该对象的HTTP请求。在收到该请求后，初始服务器向该Web缓存器发送具有该对象的HTTP响应。

4）当Web缓存器接收到该对象时，它在本地存储空间存储一份副本，并向客户的浏览器用HTTP响应报文发送该副本（通过现有的客户浏览器和Web缓存器之间的TCP连接）。

**在因特网上部署Web缓存器的原因**：

（1）Web缓存器可以大大减少对客户请求的响应时间，特别是当客户与初始服务器的瓶颈带宽远低于客户与Web缓存器之间的瓶颈带宽时更是如此。

（2）Web缓存器能够大大减少一个机构的接入链路到因特网的通信量。通过减少通信量，该机构就不必急于增加带宽，因此降低了费用。

### 2.2.6 条件GET方法

目的：确保客户端接收到的Web缓存器上的对象是最新的。

条件：①请求报文使用GET方法；②请求报文中包含了一个“If-Modified-Since：”首部行。

工作原理：

1）Web缓存器向服务器请求一个对象后，服务器在响应报文中添加**Last-Modified**首部行，表示该对象最后修改日期，Web缓存器将该对象存储在本地存储空间中。

HTTP/1.1 200 OK

Date: Sat, 8 Oct 2011 15:39:29

Server: Apache/1.3.0 (Unix)

Last-Modified: Wed, 7 Sep 2011 09:23:24

Content-Type: image/gif

2）经过一段时间后，另一个用户请求同一个对象。由于在这段时间里Web服务器上的该对象可能已经被修改了，该缓存器通过发送一个条件GET执行最新检查。

GET /fruit/kiwi.gif HTTP/1.1

Host: www.exotiquecuisine.com

If-modified-since: Wed, 7 Sep 2011 09:23:24

If-modified-since首部行的值正好等于此前服务器发送的响应报文中Last-Modified首部行的值。

该条件豹纹告诉服务器，仅当自指定日期之后该对象被修改过，才发送该对象。若该对象在这段时间内没有被修改，则服务器发送一个响应报文：

HTTP/1.1 304 Not Modified

Date: Sat, 15 Oct 2011 15:39:29

Server: Apache/1.3.0 (Unix)

状态行304 Not Modified告诉缓存器可以使用该对象，能向请求的浏览器转发它缓存的该对象副本。

## 2.3 因特网中的电子邮件

电子邮件系统包含三部分：用户代理、邮件服务器和简单邮件传输协议（SMTP）。

邮件服务器形成了电子邮件体系结构的核心。每个接收方（如Bob）在其中的某个邮件服务器上有一个邮箱。Bob的邮箱管理和维护着发送给它的报文。

一个典型的邮件发送过程是，从发送方的用户代理开始，传输到发送方的邮件服务器，再传输到接收方的邮件服务器，然后在这里被分发到接收方的邮箱中。当Bob要在他的邮箱中读取该报文时，包含他邮箱的邮件服务器来鉴别Bob。Alice的邮箱也必须能处理Bob的邮件服务器的故障。如果Alice的服务器不能将邮件交付给Bob的服务器，Alice的邮件服务器在一个报文队列中保持该报文并在以后尝试再次发送。

SMTP是因特网电子邮件中主要的应用层协议。它使用TCP可靠数据传输服务，从发送方的邮件服务器向接收方的邮件服务器发送邮件。SMTP有两个部分：运行在发送方邮件服务器的客户端和运行在接收方邮件服务器的服务器端。

### 2.3.1 SMTP

在用SMTP传送邮件之前，需要将二进制多媒体数据编码为ASCII码，并且在使用SMTP传输后要求将相应的ASCII码邮件解码还原为多媒体数据。

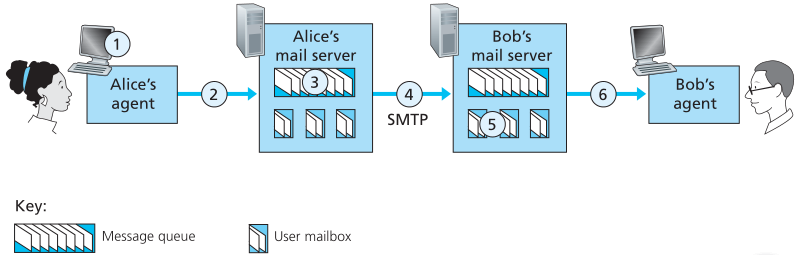


图2-6 Alice向Bob发送一条报文

假设Alice想给Bob发送一封简单的ASCII报文，其步骤如下：

1）Alice调用她的邮件代理程序并提供Bob的邮件地址，撰写报文，然后指示用户代理发送该报文。

2）Alice的用户代理把报文发给她的邮件服务器，在那里该报文被放在报文队列中。

3）运行在Alice的邮件服务器上的SMTP客户端发现了报文队列中的这个报文，它就创建一个到运行在Bob的邮件服务器上的STMP的TCP连接。

4）在经过一些初始SMTP握手后，SMTP客户通过该TCP连接发送Alice的报文。

5）在Bob的邮件服务器上，SMTP的服务器端接收该报文。Bob的邮件服务器然后将该报文放入Bob的邮箱中。

6）在Bob方便的时候，他调用用户代理阅读该报文。

SMTP用的是持续连接：如果发送邮件服务器有几个报文发往同一个接收邮件服务器，它可以通过同一个TCP连接发送这些所有的报文。

### 2.3.2 与HTTP的对比

首先，HTTP主要是一个拉协议，即在方便的时候，某些人在Web服务器上装载信息，用户使用HTTP从该服务器拉取这些信息。特别是TCP连接是由想接收文件的机器发起的。另一方面，SMTP基本上是一个推协议，即发送邮件服务器把文件推向接收邮件服务器。特别是，这个TCP连接是由要发送该文件的机器发起的。

第二个区别：SMTP要求每个报文采取7比特ASCII码格式。HTTP数据则不受这种限制。

第三个区别：如何处理一个及包含文本又包含图形的文档。HTTP把每个对象封装到它自己的HTTP响应报文中，而SMTP则把所有报文对象放在一个报文中。

### 2.3.3 邮件报文格式

每个首部含有一个From：首部行和一个To：首部行；一个首部也许包含一个Subject：首部行以及其他可选的首部行。

一个典型的报文首部看起来如下：

From: alice@crepes.fr

To: bob@hamburger.edu

Subject: Searching for the meaning of life.

### 2.3.4 邮件访问协议

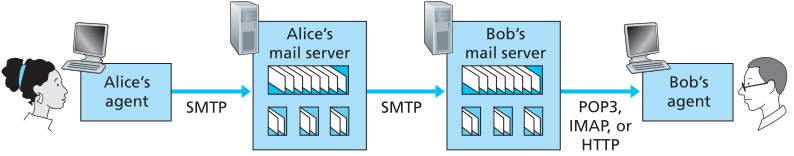


图2-7 电子邮件协议及通信实体

1. POP3

当用户代理打开了一个到邮件服务器端口110上的TCP连接后，POP3就开始工作了。POP3按照三个阶段进行工作：认证（authorization）、事务处理以及更新。

1）认证：用户代理发送（以明文形式）用户名和口令以鉴别用户。

2）事务处理：用户代理取回报文；同时用户代理在这个阶段还能进行如下操作，对报文做删除标记，取消报文删除标记，以及获取邮件的统计信息。

3）更新：它出现在客户发送了quit命令之后，目的是结束该POP3会话；这时，该邮件服务器删除那些被标记为删除的报文。

在POP的事务处理过程中，用户代理发出一些命令，服务器对每个命令做出回答。回答可能有两种：+OK（有时后面还跟有服务器到客户的数据），被服务器用来指示前面的命令是正常的；-ERR，被服务器用来指示前面的命令出现了某些差错。

POP3的用户代理通常被用户配置为“下载并删除”或“下载并保留”方式。

在用户代理与邮件服务器之间的POP3会话期间，该POP3服务器保留了一些状态信息；特别是记录了哪些用户报文被标记为删除了。然而，POP3服务器并不在会话过程中携带状态信息。

2. IMAP

POP3协议没有给用户提供任何创建远程文件夹并未报文指派文件夹的方法。

IMAP服务器把每个报文与一个文件夹联系起来；当报文第一次到达服务器时，它与收件人的INBOX文件夹相关联。收件人则能够把邮件到一个新的、用户创建的文件夹中，阅读邮件，删除邮件等。

IMAP为用户提供了创建文件夹以及将邮件从一个文件夹移动到另一个文件夹的命令；IMAP还为用户提供了在远程文件夹查询邮件的命令，按指定条件去查询匹配的邮件。

IMAP的另一个重要特性是它具有允许用户代理获取报文某些部分的命令。例如，一个用户代理可以只读取一个报文的报文首部，或只是一个多部分MIME报文的一部分。

3. 基于Web的电子邮件

使用这种服务，用户代理就是普通的浏览器，用户和他远程邮箱之间的通信则通过HTTP进行。当发件人（Alice）要发送一封电子邮件报文时，该电子邮件从Alice的浏览器发送到她的邮件服务器，使用的是HTTP而不是SMTP。而Alice的邮件服务器在与其他的邮件服务器之间发送和接收邮件时，仍然使用的是SMTP。

## 2.4 DNS：因特网的目录服务

### 2.4.1 DNS提供的服务

域名系统（Domain Name System，DNS）的主要任务是进行主机名到IP地址的转换。

DNS是：①一个由分层的DNS服务器实现的分布式数据库；②一个使得主机能够查询分布式数据库的应用层协议。

DNS提供的服务：

**1）主机别名**：名为relay1.west-coast.enterprise.com的主机，可能还有两个别名为enterprise.com和www.enterprise.comde。relay1.west-coast.enterprise.com也称为规范主机名。应用程序可以调用DNS来获得主机别名对应的规范主机名以及主机的IP地址。

**2）邮件服务器别名**：类似于主机别名，电子邮件应用程序可以调用DNS来获得主机别名对应的规范主机名以及主机的IP地址。

**3）负载分配**：DNS也用于在冗余的服务器之间进行负载分配。繁忙的站点被冗余分布在多台服务器上，每台服务器均运行在不同的端系统上，每个都有着不同的IP地址。当客户对映射到某地址集合的名字发出一个DNS请求时，该服务器用IP地址的整个集合进行响应，但在每个回答中循环这些地址次序。

### 2.4.2 DNS工作原理

1.分布式、层次数据库

有三种类型的DNS服务器：根DNS服务器、顶级（Top-level Domain，TLD）DNS服务器和权威DNS服务器。

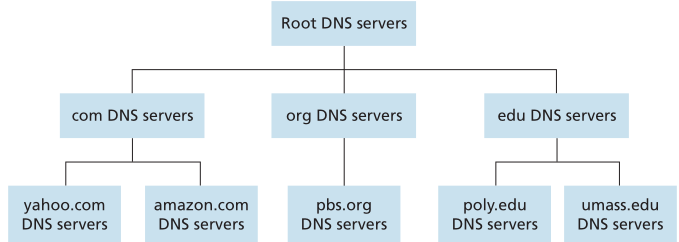


图2-8 部分DNS服务器的层次结构

**1）根域名服务器**

根名字服务器提供TLD服务器的IP地址。

**2）顶级域（DNS）服务器**

每个顶级域（如com、org、net、edu和gov）和所有国家的顶级域（如uk、fr、ca和jp），都有TLD服务器。TLD服务器提供了权威DNS服务器的IP地址。

**3）权威DNS服务器**

在因特网上具有公共可访问主机（如Web服务器和邮件服务器）的每个组织结构必须提供公共可访问的DNS记录，这些记录将这些主机的名字映射为IP地址。一个组织机构的权威DNS服务器收藏了这些DNS记录。一个组织机构能够选择实现它自己的权威DNS服务器以保存这些记录。

**本地DNS服务器**

每个ISP都有一台本地DNS服务器（也叫默认名字服务器）。当主机与某个ISP连接时，该ISP提供一台主机的IP地址，该主机具有一台或多台其本地DNS服务器的IP地址。

主机的本地DNS服务器通常“邻近”本主机。对某机构ISP而言，本地DNS服务器可能就与主机在同一个局域网中；对于某居民区ISP来说，本地DNS服务器通常与主机相隔不超过几台路由器。

**各种DNS服务器间的交互**

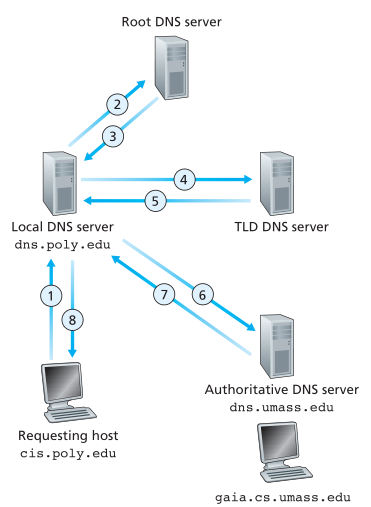


图2-9 DNS服务器间的交互

例：假设主机cse. poly.edu想知道主机gaia.cs.umass.edu的IP地址。同时假设cse. poly.edu主机的本地DNS服务器为dns.poly.edu，并且gaia.cs.umass.edu的权威域名服务器为dns.umass.edu。如图2-9所示：

1）主机cse. poly.edu首先向它的本地DNS服务器dns.poly.edu发送一个DNS查询报文。该查询报文含有被转换的主机名gaia.cs.umass.edu。

2）本地DNS服务器将该报文转发到根DNS服务器。

3）该根DNS服务器注意到其edu前缀并向本地DNS服务器返回负责edu的TLD的IP地址列表。

4）本地DNS服务器则再次向这些TLD服务器之一发送查询报文。

5）该TLD服务器注意到umass.edu前缀，并用权威DNS服务器的IP地址进行响应。

6）最后，本地DNS服务器直接向dns.umass.edu重发查询报文

7）dns.umass.edu用gaia.cs.umass.edu的IP地址进行响应。

注意到本例中，为了获得一台主机名的映射，共发送了8份DNS报文：4份查询报文和4份应答报文。

图2-18所示的例子利用了递归查询和迭代查询。从cse.poly.edu到dns.poly.edu发出的查询是递归查询，因为该查询以自己的名义请求dns.poly.edu来获得该映射。而后继的3个查询都是迭代查询，因为所有的回答都是直接返回给dns.poly.edu。

2. DNS缓存

为了改善时延性能并减少在因特网上到处传输的DNS报文数量，DNS广泛使用了缓存技术。

原理：在一个请求链中，当某DNS服务器接受一个DNS应答（例如，包含某主机名到IP地址的映射）时，它能将映射缓存在本地存储器中。例如，在图2-8中，每当本地DNS服务器dns.poly.edu从某个DNS服务器接收到一个应答，它能够缓存包含在该回答中的任何信息。如果在DNS服务器中缓存了一台主机名/IP地址对，另一个对相同主机名的查询到达该DNS服务器时，该DNS服务器就能够提供所要求的IP地址，即使它不是该主机名的权威服务器。由于主机和主机名的IP地址间的映射并不是永久的， DNS服务器在一段时间后将丢弃缓存的信息。

### 2.4.3 DNS记录和报文

共同实现DNS分布式数据库的所有DNS服务器存储了资源记录（Resource Record，RR），RR提供了主机名到IP地址的映射。每个DNS回答报文包含了一条或多条资源记录。

RR是一个包含了下列字段的4元组：

（Name，Value，Type，TTL）

TTL是该记录的生存时间，它决定了资源记录应当从缓存中删除的时间。Name和Value的值取决于Type：

1）如果Type=A，则Name是主机名，Value是该主机名对应的IP地址。因此，一条类型为A的RR提供了标准的主机名到IP地址的映射，如(relay1.bar.foo.com, 145.37.93.126, A)。

2）如果Type=NS，则Name是个域（如foo.com），而Value是个知道如何获得该域中主机IP地址的权威DNS服务器的主机名。这个记录用于沿着查询链来路由DNS查询，如(foo.com, dns.foo.com, NS)。

3）如果Type=CNAME，则Value是别名为Name的主机对应的规范主机名。该记录能够向查询的主机提供一个主机名对应的规范主机名(foo.com, relay1.bar.foo.com, CNAME)。

4）如果Type=MX，则Value是个别名为Name的邮件服务器的规范主机名。如(foo.com, mail.bar.foo.com, MX)。

1. DNS报文

DNS报文只有查询和回答报文两种，并且它们有着相同的格式。其报文格式如下：

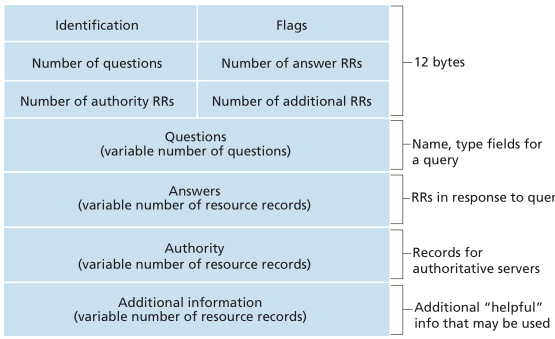


图2-10 DNS报文格式

1）首部区域

前12个字节是首部区域，第一个字段（Identification）是一个16比特的数，用于标识该查询。这个标识符会被复制到对查询的回答报文中，以便让客户用它来匹配发送的请求和接收到的回答。

标志（Flags）字段中含有若干标志。1比特的“查询/回答”标志位指出报文是查询报文（0）还是回答报文（1）。

首部区域中最后4个有关数量的字段，指出了在首部后的4类数据区域出现的数量。

2）**问题区域**包含着正在进行的查询信息。该区域包括：①名字字段，包含正在被查询的chunk主机名字；②类型字段，指出有关改名字的正被询问的问题类型。

3）**回答区域**包含了对最初请求的名字的资源记录（RR）。在回答报文的回答区域中可以包括多条RR，因此一个主机名能够有多个IP地址。

4）**权威区域**包含了其他权威服务器的记录

5）**附加区域**包含了其他有帮助的记录。

## 2.5 P2P文件分发

1. P2P体系结构的扩展性

分发时间是所有N个对等方得到该文件的副本所需要的时间。

对于客户­服务器体系结构，随着对等方数量的增加，分发时间呈线性增长并且没有界。然而，对于P2P体系结构，最小分发时间不仅总是小于客户—服务器体系结构的分发时间，并且对于任意的对等方数量N，总是小于1小时。因此，具有P2P体系结构的应用程序能够是自扩展的。

2. BitTorrent

BitTorrent是一种用于文件分发的流行P2P协议。参与一个特定文件分发的所有对等方的集合被称为一个洪流（torrent）。在一个洪流中的对等方彼此下载等长度的文件块（chunk），典型的块长度为256KB。当一个对等方首次加入一个洪流时，它没有块。随着时间的流逝，它累积了越来越多的块。

每个洪流具有一个基础设施节点，称为追踪器（tracker）。当一个对等方加入某洪流时，它向追踪器注册自己，并周期性地通知追踪器它仍在该洪流中。以这种方式，追踪器跟踪参与在洪流中的对等方。

在任何给定的时间，每个对等方将具有来自该文件的块的子集，并且不同的对等方具有不同的子集。Alice周期性地（经过TCP连接）询问每个邻近对等方它们所具有的块流标。如果Alice有L个不同的邻居，她将获得L个块列表。有了这信息，Alice将对她当前还没有的块发出请求。

在决定请求哪些块的过程中，Alice使用一种最稀缺优先的块，并首先请求那些最稀缺的块。这样，最稀缺块得到更为迅速的重新分发，其目标是均衡每个块在洪流中的副本数量。

## 2.6 视频流和内容分发网

### 2.6.1 因特网视频

## 2.7 FTP

如图2-11所示，用户通过FTP用户代理和FTP交互。用户首先提供远程主机的主机名，使得在本地主机的FTP客户端进程建立一个与远程主机上的FTP服务器进程的TCP连接。然后，用户提供用户标识和密码，作为FTP命令的一部分通过TCP连接一并发送。一旦服务器对用户授权，用户就能从远程文件系统复制一个或多个文件存储到本地文件系统上（反之亦然）。

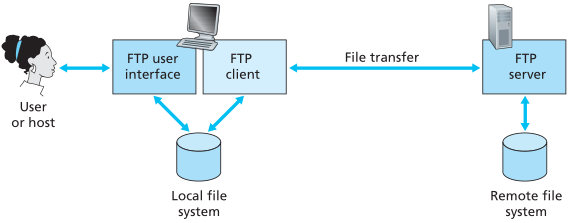


图2-11 FTP在本地和远端文件系统间移动文件

HTTP和FTP都是文件传输协议并且有许多共同特征。如：它们都运行在TCP协议上。然而，它们也有诸多重要的不同。最重要的不同之处是FTP使用两条并行的TCP连接来传输一个文件，一条控制连接和一条数据连接。

控制连接用来在两台主机之间发送控制信息—如像用户标识，密码，更改远程目录的命令以及上传（“put”）和下载（“get”）文件的命令等。

数据连接用来发送一个文件。

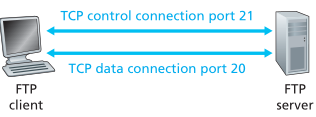


图2-12 控制和数据连接

由于FTP使用一个分开的控制连接，因此称FTP发送控制信息为带外数据（out-of-band）。HTTP发送请求和响应首部行联通传输文件一起在相同的TCP连接发送。因此，称HTTP发送控制信息为带内数据（in-band）。SMTP也是in-band。

当一个用户发起一个与远程主机的FTP会话时，客户端先启动一个与服务端的TCP连接在服务器端口号21上。客户端通过控制连接发送用户标识和密码。当服务端通过控制连接接收到一个文件传输的命令时，服务端启动一个到客户端的TCP数据连接。FTP只通过该连接发送一个文件便关闭连接。如果，在同一个会话中，用户想要传输另一个文件，FTP开启另一个数据连接。因此，在FTP中，控制连接在整个用户会话期间保持开启，而对每一个文件传输都要创建一个新的数据连接（即数据连接是非持续的）。

在整个会话中，FTP服务器必须维持用户的状态。特别是，服务器必须将一个控制连接关联一个特定的用户账户，并且当用户在远程目录树中徘徊时，服务器必须跟踪用户的当前目录。

### 2.7.1 FTP命令及响应

常用命令

* USER username：发送用户名
* PASS password：发送密码
* LIST：让服务器返回当前远端目录的文件列表。文件列表通过数据连接发送。
* RETR filename：用来接收从远程主机的当前目录接收一个文件。这个命令导致远程主机发起一个数据连接并通过该连接发送请求的文件。
* STOR filename：用户存储（即上传）一个文件到远程主机的当前目录。

典型的响应

* 331 用户名ok，要求密码
* 125 数据连接已经开启，开始传输
* 425 不能开启数据连接
* 452 写文件错误

# 第三章 运输层

## 3.1 概述和运输层服务

运输层协议为运行在不同主机上的应用进程之间提供了**逻辑通信**功能。应用进程使用运输层提供的逻辑通信功能彼此发送报文，而无须考虑承载这些报文的物理基础设施的细节。

运输层协议是在端系统中而不是在路由器中实现的 。在发送端，运输层将从发送应用程序进程接收到的报文转换成运输层分组，即运输层**报文段**（segment）。实现的方法是将应用报文划分为较小的块，并为每块加上一个运输层首部以生成运输层报文段。然后，在发送端系统中，运输层将这些报文段传递给网络层，网络层将其封装成网络层分组（即数据报）并向目的地发送。

### 3.1.1 运输层和网络层的关系

在协议栈中，运输层刚好位于网络层之上。网络层提供了主机之间的逻辑通信，而运输层为运行在不同主机上的进程之间提供了逻辑通信。

如果网络层协议无法为进程之间发送的应用程序报文提供时延或带宽保证的话，运输层协议也就无法为进程之间发送的应用程序报文提供时延或带宽保证。

然而，即使底层网络协议不能在网络层提供相应的服务，运输层协议也能提供某些服务。

### 3.1.2 因特网运输层概述

IP（Internet Protocol）为主机之间提供了逻辑通信。IP的服务模型是尽力而为交付服务（best-effort delivery service）。它不确保报文段的交付，不保证报文段的按序交付，不保证报文段中数据的完整性。因此，IP被称为不可靠服务。

TCP和UDP最基本的责任是，将两个端系统间IP的交付服务扩展为运行在端系统上的两个进程之间的交付服务。将主机间交付扩展到进程间交付被称为运输层的多路复用与多路分解。

TCP为应用程序提供了几种附加服务。首先，它提供**可靠数据传输**。通过使用流量控制、序号、确认和定时器，TCP确保正确地、按序地将数据从发送进程交付给接收进程。TCP还提供拥塞控制，TCP拥塞控制防止任何一条TCP连接用过多流量来淹没通信主机之间的链路和交换设备。TCP力求为每个通过一条拥塞网络链路的连接平等地共享网络链路带宽。

UDP流量是不可调节的，使用UDP传输的应用程序可以根据其需要以其愿意的任何速率发送数据。

## 3.2 多路复用与多路分解

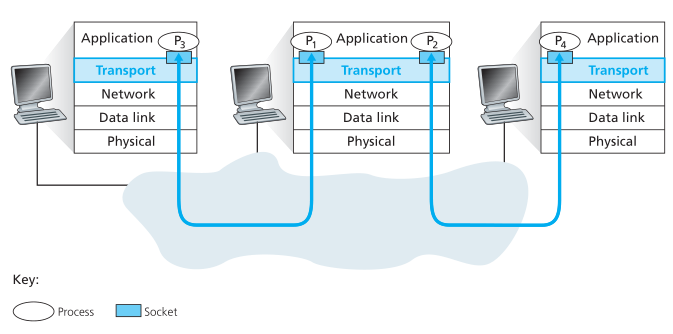


图3-1 运输层的多路复用和多路分解

每个运输层报文段中有几个字段，在接收端，运输层检查这些字段，标识出接收套接字，进而将报文段定向到该套接字。

将运输层报文段中的数据交付到正确的套接字的工作称为**多路分解**。

在源主机从不同套接字收集数据块，并为每个数据块封装上首部信息从而生成报文段，然后将报文段传递到网络层，所有这些工作称为**多路复用**。

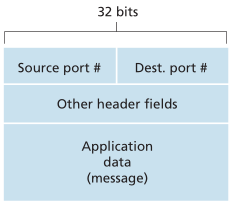


图3-2 运输层报文段中的源与目的端口号

多路复用要求：①套接字由唯一标识符；②每个报文段有特殊字段来指示该报文段所要交付到的套接字。如图3-2所示，这些特殊字段是**源端口号字段**和**目的端口号字段**。端口号是一个16比特的数，其大小在0~65535之间。0~1023范围的端口号称为周知端口号，保留给注入HTTP和FTP之类的周知应用程序使用。

在主机上的每个套接字能够分配一个端口号，当报文段到达主机时，运输层检查报文段中的目的端口号，并将其定向到相应的套接字。然后报文段中的数据通过套接字进入其所连接的进程。

1. 无连接的多路复用与多路分解

一个UDP套接字是有一个二元组全面标识的，该二元组包含一个目的IP地址和一个目的端口号。因此，如果两个UDP报文段有不同的源IP地址和/或源端口号，但具有相同的目的IP地址和目的端口号，那么这两个报文段将通过相同的目的套接字被定向到相同的目的进程。

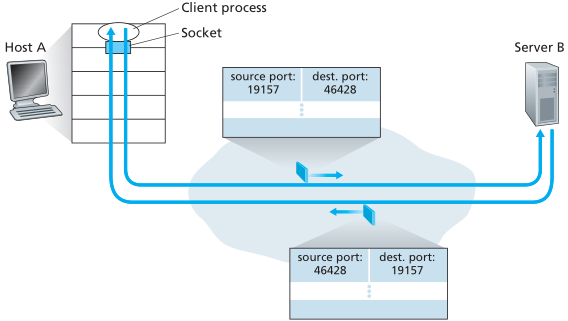


图3-3 源端口号与目的端口号的反转

如图3-3所示，在A到B的报文段中，源端口号用作“返回地址”的一部分，即当B需要回发一个报文段给A时，B到A的报文段中的目的端口号便从A到B的报文段中的源端口号中取指。

2. 面向连接的多路复用与多路分解

TCP套接字是有一个四元组（源IP地址，源端口号，目的IP地址，目的端口号）来标识的。因此，当一个TCP报文段从网络到达一台主机时，该主机使用全部4个值来将报文段定向（分解）到相应的套接字。特别与UDP不同的是，两个具有不同源IP地址或源端口号的到达TCP报文段将被定向到两个不同的套接字，除非TCP报文段携带了初始创建连接的请求。

3. Web服务器与TCP

连接套接字与进程之间并非总是有着一一对应的关系。事实上，当今的高性能Web服务器通常只使用一个进程，但是为每个新的客户连接创建一个具有新连接套接字的新线程。

## 3.3 无连接运输：UDP

UDP只是做了运输层协议能够做的最少工作。除了复用/分解功能及少量的差错检测外，它几乎没有对IP增加别的东西。

使用UDP时，在发送报文段之前，发送方和接收方的运输层实体之间没有握手。因此，UDP被称为是无连接的。

DNS是一个使用UDP作为运输层协议。

更适合用UDP的原因：

1）关于发送什么数据以及何时发送的应用层控制更为精细。采用UDP时，只要应用进程将数据传递给UDP，UDP就会将此数据打包进UDP报文段并立即将其传递给网络层。在另一方面，TCP有一个拥塞控制机制，以便当源和目的主机间的一条或多条链路变得极度拥塞时来遏制运输层TCP发送方。TCP仍将继续重新发送数据报文直到目的主机收到此报文并加以确认，而不管可靠交付需要多长时间。因为实时应用通常要求最小的发送速率，不希望过分的延迟报文段的传送，且能容忍一些数据丢失，TCP服务模型并不是特别适合这些应用的需要。

2）无需建立连接。TCP在开始数据传输之前要经过三次握手，UDP却不需要任何准备即可进行数据传输，因此UDP不会引入建立连接的时延。这可能是DNS运行在UDP上的主要原因。

3）无连接状态。TCP需要在端系统中维护连接状态，此连接状态包括接收和发送缓存、拥塞控制参数以及序号与确认号的参数。另一方面，UDP不维护连接状态，也不跟踪这些参数。

4）分组首部开销小。每个TCP报文段都有20字节的首部开销，而UDP仅有8个字节的开销。

### 3.3.1 UDP报文段结构

UDP报文段结构如图3-4所示，应用层数据占用UDP报文段的数据字段。UDP首部只有4个字段，每个字段由两个字街组成。通过端口号可以使目的主机将应用数据交给运行在目的端系统中的相应进程（即执行分解功能）。长度字段指定了在UDP报文段中的字节数（首部加数据）。因为数据字段的长度在一个UDP段中不同于在另一个段中，故需要一个明确的长度。接收方使用检验和来检查在该报文段中是否出现了差错。

图3-4 UDP报文段结构

### 3.3.2 UDP校验和

UDP检验和提供了差错检测功能。发送方的UDP对报文段的所有16比特字的和进行反码运算，求和时遇到的任何溢出都被回卷。得到的结果被放在UDP报文段中的检验和字段。

在既无法确保逐链路的可靠性，又无法确保内存中的差错检测的情况下，如果端到端数据传输服务要提供差错检测，UDP就必须在端到端基础上在运输层提供差错检测。

## 3.4 可靠数据传输原理

图3-5说明了我们学习可靠数据传输的框架。为上层实体提供的服务抽象是：数据可以通过一条可靠的信道进行传输。借助于可靠信道，传输数据比特就不会受到损坏（0->1, 1->0）或丢失，而且所有数据都是按照其发送顺序进行交付。

实现这种服务抽象是**可靠数据传输协议**的责任。

rdt（reliable data transmission）的发送端和接收端都要通过udt\_send()发送分组给对方。

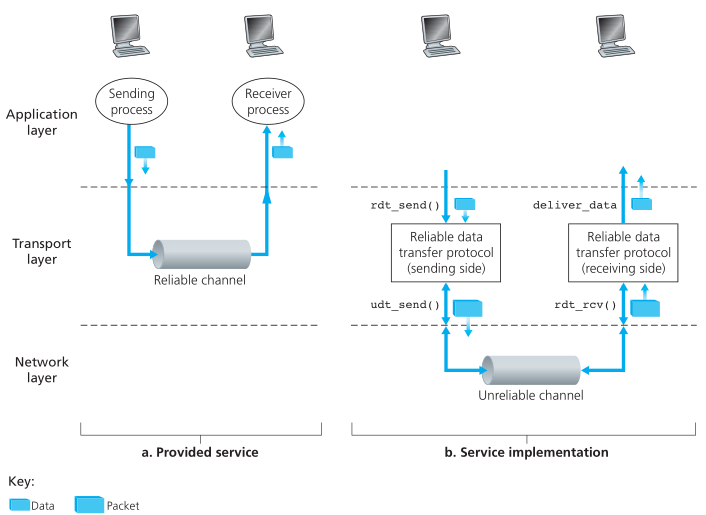
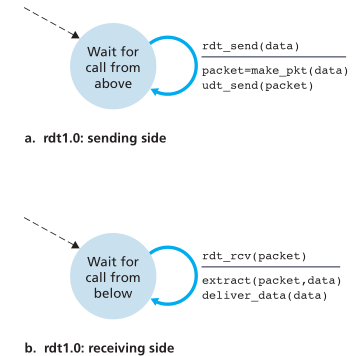


图3-5 可靠数据传输：服务模型与服务实现

### 3.4.1 构造可靠数据传输协议

1. 经完全可靠信道的可靠数据传输：rdt1.0

首先，假设底层信道是完全可靠的。图3-6显示了rdt1.0发送方和接收方的有限状态机（Finite-State Machine， FSM）的定义。FSM描述图中的箭头指示了协议从一个状态变迁到另一个状态。引起变迁的事件显示在表示变迁的横线上方，事件发生所采取的动作显示在横线下方。∧表示缺少动作或事件。FSM的初始状态用虚线表示。

rdt的发送端只通过rdt\_send(data)事件接收来自较高层的数据，产生一个包含该数据的分组，并将分组发送到信道中。实际上，rdt\_send(data)是由较高层应用的过程调用产生的。

图3-6 rdt1.0：用于完全可靠信道的协议

在接收端，rdt通过rdt\_rcv(packet)事件从底层信道接收一个分组，从分组中取出数据并将数据上传给较高层。实际上，rdt\_rcv(packet)事件是由较高层的过程调用产生的。

2. 经具有比特差错信道的可靠数据传输：rdt2.0

假设分组中的比特可能受损。

通过**肯定确认**（positive acknowledgment）与**否定确认**（negative acknowledgment），接收方可以让发送方知道哪些内容被正确接收，哪些内容有误并因此需要重复。基于这样重传机制的可靠数据传输协议称为**自动重传请求**（Automatic Repeat reQuest， ARQ）协议。

ARQ协议中还需要另外三种协议功能来处理存在比特差错的情况：

1）**差错检测**。

2）**接收方反馈**。因为发送方和接收方通常在不同端系统上执行，发送方要了解接收方情况的唯一途径就是让接收方提供明确的反馈信息给发送方。rdt2.0将从接收方向发送方会送ACK与NAK分组。理论上，这些分组只需要1bit，如0表示NAK，1表示ACK。

3）**重传**。接收方收到有差错的分组时，发送方将重传该分组。

图3-10说明了表示rdt2.0的FSM，该数据传输协议采用了差错检测、肯定确认与否定确认。rdt2.0发送端有两个状态。在最左边的状态中，发送端协议正等待来自上层传下来的数据。当rdt\_send(data)事件出现时，发送方将产生一个包含待发送数据的分组，带有校验和，然后经由udt\_send(sndpkt)操作发送该分组。在最右边的状态中，发送方协议等待来自接收方的ACK或NAK分组。如果收到一个ACK分组，则发送方知道最近发送的分组已被正确接收，因此协议返回到等待来自上层的数据的状态。如果收到一个NAK分组，该协议重传上一个分组并等待接收方响应重传分组而回送的ACK或NAK。

注意到下列事实很重要：当发送方处于等待ACK或NAK的状态时，它不能从上层获得更多的数据；这就是说，rdt\_send()时间不可能出现；仅当接收到ACK并离开该状态时才能发生这样的事件。因此，发送方将不会发送一块新数据，除非发送方确信接收方已正确接收当前分组。由于这种行为，rdt2.0这样的协议被称为**停等**（stop-and-wait）协议。

rdt2.0接收方的FSM仍然只有单一状态。当分组到达时，接收方要么回答一个ACK，要么回答一个NAK，这取决于收到的分组是否受损。符号rdt\_rcv(rcvpkt)&&corrupt(rcvpkt)对应于收到一个分组并发现有错的事件。

rdt2.0存在一个致命的缺陷，即没有考虑到ACK或NAK分组受损的可能性。如果一个ACK或NAK分组受损，发送方无法知道接收方是否正确接收了上一块发送的数据。

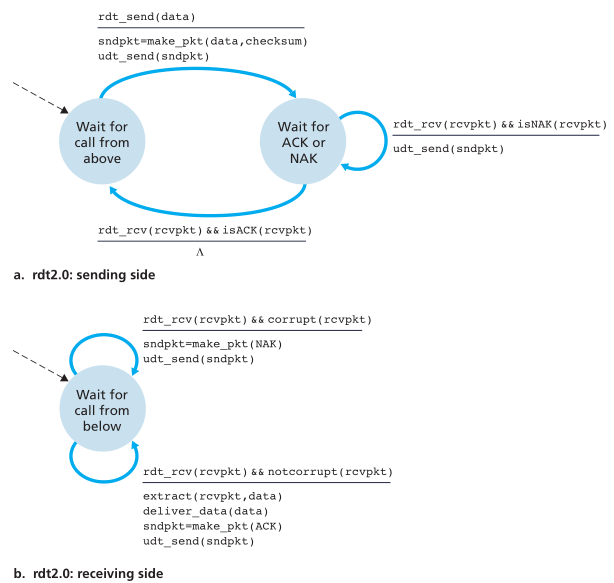


图3-7 rdt2.0：用于具有比特差错信道的协议

解决这个问题的简单方法是在数据分组中添加一新字段，让发送方对其数据分组编号，即将发送数据分组的序号放在该字段。于是，接收方只需要检查序号即可确定收到的分组是否是一次重传。对于停等协议这种简单情况，1比特序号就够了，因为它可让接收方知道发送方是否正在重传前一个分组（接收到的分组序号与最近收到的分组序号相同），或是一个新分组。因为目前我们假定信道不丢分组，ACK和NAK分组本省不需要指明它们要确认的分组序号。

图3-8和3-9给出了rdt2.1的FSM描述，这是rdt2.0的修订版。

rdt2.1使用了从接收方到发送方的肯定确认和否定确认。当接收到失序的分组时，接收方对所接收的分组发送一个肯定确认。如果收到受损的分组，则接收方将发送一个否定确认。

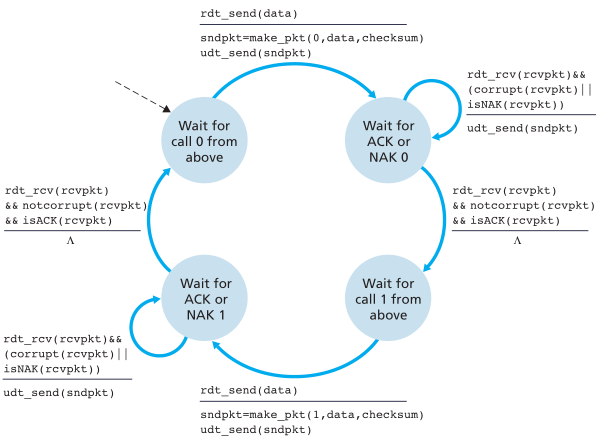


图3-8 rdt2.1发送方

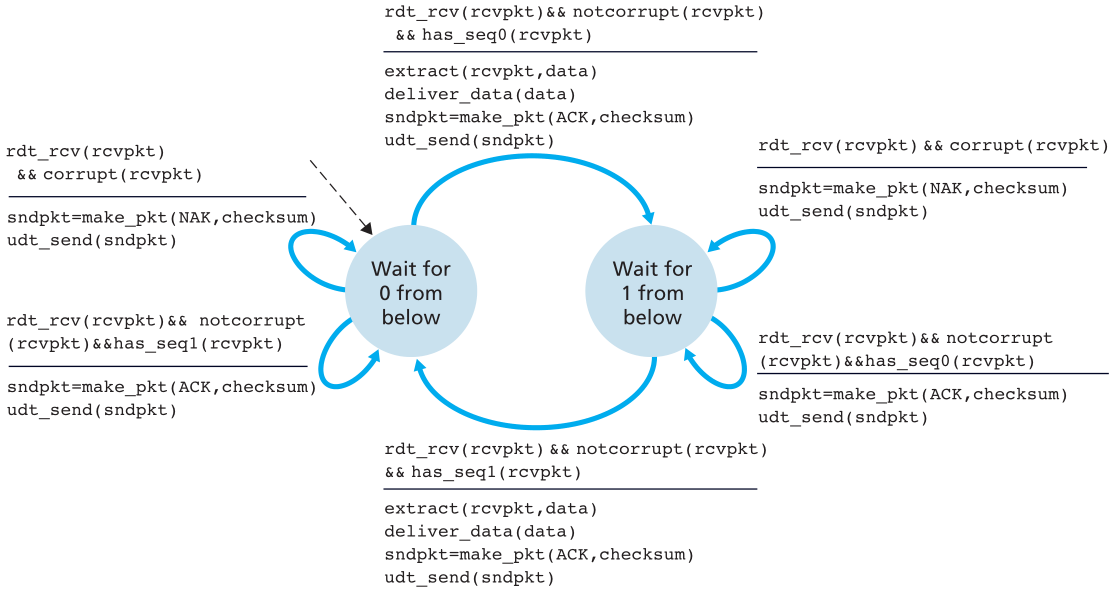


图3-8 rdt2.1接收方

如果不发送NAK，而是对上次正确接收的分组发送一个ACK，我们也能实现与NAK一样的效果。发送方接到对同一个分组的两个ACK（即接收冗余ACK）后，就知道接收方没有正确接收到跟在被确认两次的分组后面的分组。rdt2.2是在由比特差错信道上实现的一个无NAK的可靠数据传输协议，如图3-9和3-10所示。

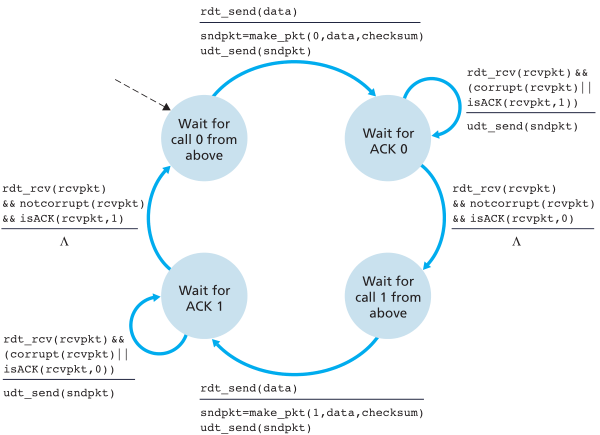


图3-9 rdt2.2发送方

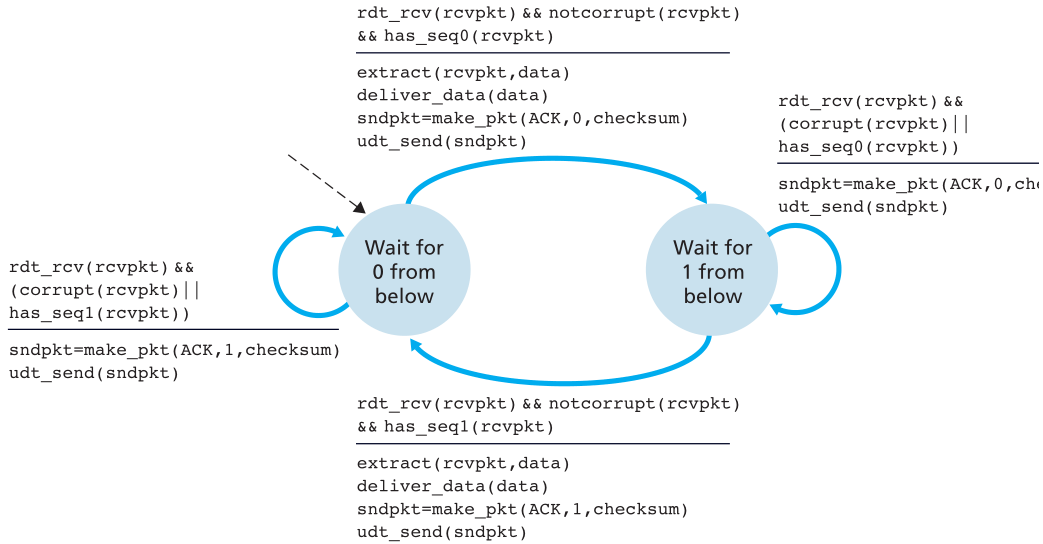


图3-10 rdt2.2接收方

rdt2.1和rdt2.2之间的细微变化在于，接收方此时必须包括由一个ACK报文所确认的分组序号（在make\_pkt()中包括参数ACK 0或ACK 1来实现），发送方此时必须检查接收到的ACK报文中被确认的分组序号（在isACK()中包括参数0或1来实现）。

3. 经具有比特差错的丢包信道的可靠数据传输：rdt3.0

假定除了比特受损外，底层信道还会丢包。

为了实现基于时间的重传机制，需要一个倒计时定时器，在一个给定的时间量过期后，可中断发送方。因此，发送方需要能做到：①每次发送一个分组（包括第一次分组和重传分组）时，便启动一个定时器；②响应定时器中断（采取适当的动作）；③终止定时器。

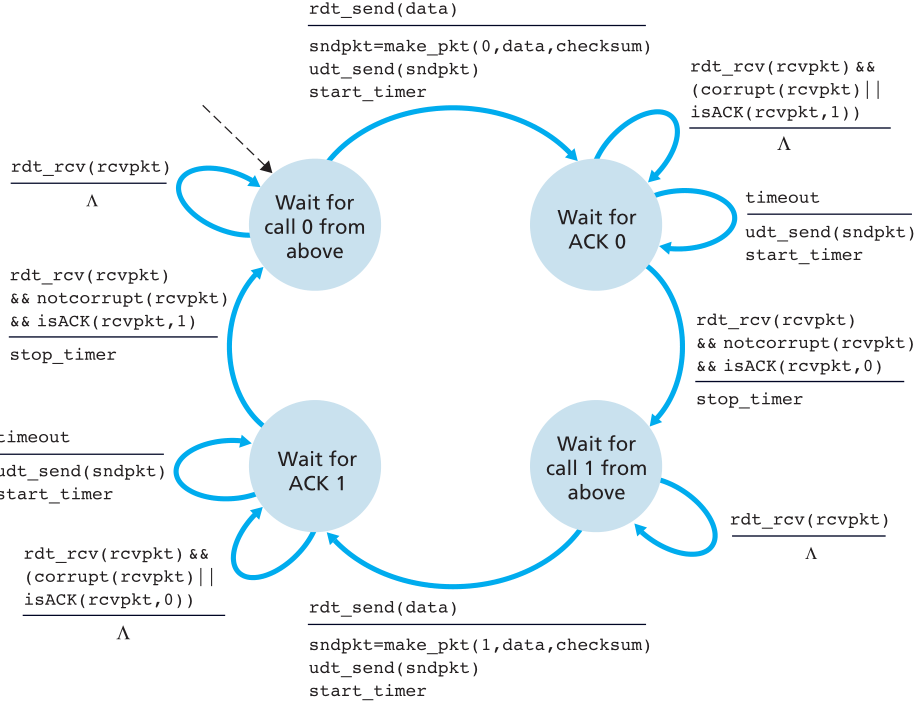


图3-11 rdt3.0发送方

图3-11给出了rdt3.0的发送方FSM，这是一个在可能出错和丢包的信道上可靠传输数据的协议。rdt3.0接收方的FSM与rdt2.2接收方相同。

### 3.4.2 流水线可靠数据传输协议

为解决停等协议利用率非常低的问题，允许发送方发送多个分组而无须等待确认。图3-12显示了如果发送方可以在等待确认之前发送3个报文，其利用率也基本上提高了3被。因为许多从发送方向接收方输送的分组可以被看成是填充到一条流水线中，故这种技术被称为流水线。

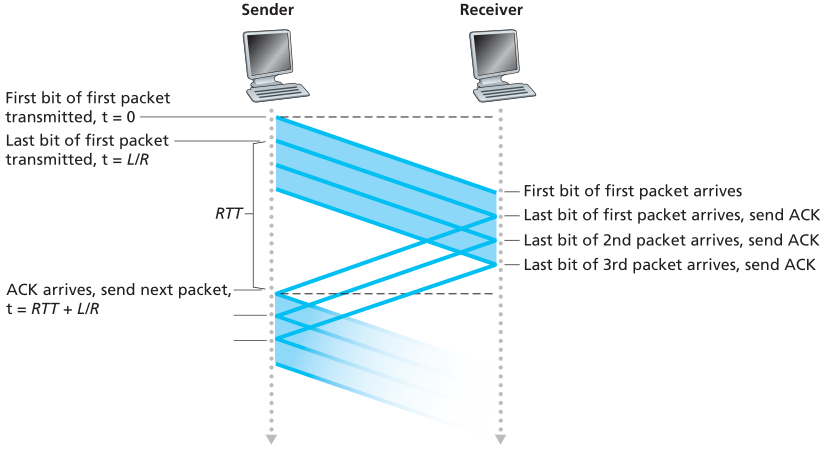


图3-12 停等和流水线发送

流水线技术对可靠数据传输协议可带来如下影响：

* 必须增加序号范围，因为每个输送中的分组必须有一个唯一的序号，而且也许有多个在输送中的未确认报文。
* 协议的发送方和接收方两端也许不得不缓存多个分组。发送方最低限度应当能缓冲那些。如下面讨论的那样，接收方或许也需要缓存那些已正确接收的分组。
* 所需序号范围和对缓冲的要求取决于数据传输协议如何处理丢失、损坏及延时过大的分组。解决流水线的差错恢复有两种基本方法：回退N步和选择重传。

### 3.4.3 回退N步

允许发送方发送多个分组（当有多个分组可用时）而不需要等待确认，但它也受限于在流水线中确认的分组数不能超过某个最大允许数N。

如果我们将基序号（base）定义为最早未确认分组的序号，将下一个序号（nextseqnum）定义为最小的未使用序号（即下一个待发分组的序号），即可将序号范围分割成4段。在[0, base - 1]段内的序号对应于已经发送并被确认的分组。[base, nextseqnum - 1]段内对应已经发送但未被确认的分组。[nextseqnum, base + N - 1]段内的序号能用于那些要被立即发送的分组，如果有数据来自上层的话。最后，大于或等于base+N的序号是不能使用的，指导当前流水线中未被确认的分组（特别是序号为base的分组）已得到确认为止。

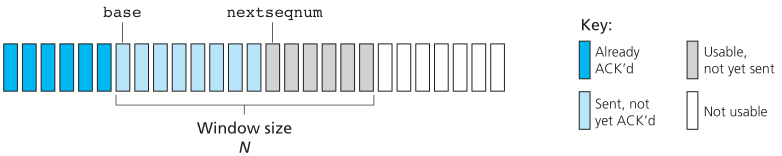


图3-13 GBN中发送方看到的序号

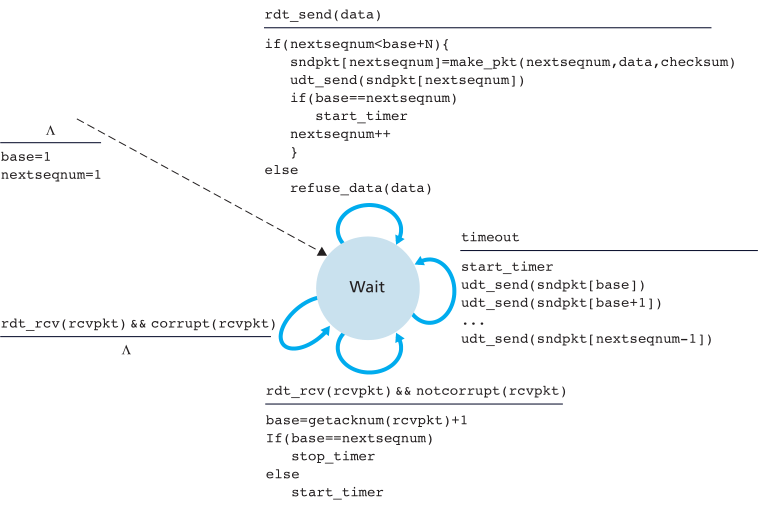


图3-14 GBN发送方的扩展FSM描述

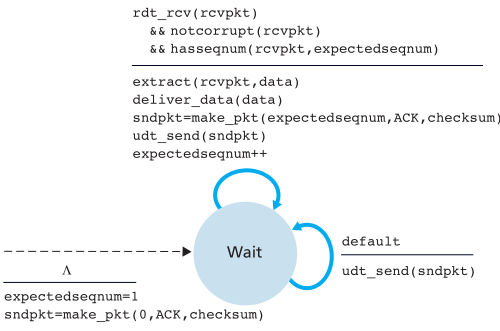


图3-15 GBN接收方的扩展FSM描述

GBN发送方必须响应三种类型的事件：

* **上层的调用**。当上层调用rdt\_send()时，发送方首先检查发送窗口是否已满，即是否有N个已发送但未被确认的分组。如果窗口未满，则产生一个分组并将其发送，并相应地更新变量。如果窗口已满，发送方只需将数据返回给上层，隐式地指示上层该窗口已满。然后上层可能会过一会儿再试。在实际实现中，发送方更可能缓存（并不立即发送）这些数据，或者使用同步机制（如一个信号量或标志）允许上层在仅当窗口不满时才调用rdt\_send()。
* **收到一个ACK**。在GBN协议中，对序号为n的分组的确认采取**累积确认**的方式，表明接收方已正确接收到序号为n的以前且包括n在内的所有分组。
* **超时事件**。协议的名字“回退N步”来源于出现丢失和时延过长分组时发送方的行为。就像停等协议中那样，定时器将再次用于恢复数据或确认分组的丢失。如果出现超时，发送方重传所有已发送但还未被确认过的分组。图3-14中的发送方仅使用一个定时器，它可被当做是最早的已发送但未被确认的分组所使用的定时器。如果收到一个ACK，但仍有已发送但未被确认的分组，则定时器被重新启动。如果没有已发送但未被确认的分组，停止该定时器。

在GBN中，如果一个序号为n的分组被正确接收到，并且按序，则接收方为分组n发送一个ACK，并将该分组中的数据部分交付到上层。在所有其他情况下，接收方丢弃该分组，并为最近按序接收的分组重新发送ACK。注意到因为一次交付给上层一个分组，如果分组k已接收并交付，则所有序号比k小的分组也已经交付。因此，使用累积确认是GBN一个自然的选择。

有上述可知，接收方不需要缓存任何失序分组，需要维护的唯一信息就是下一个按序接收的分组的序号expectedseqnum。

### 3.4.4 选择重传

选择重传（SR）协议通过让发送方仅重传那些它怀疑在接收方出错（即丢失或受损）的分组而避免了不必要的重传。这种个别的、按需的重传要求接收方逐个的确认正确接收的分组。再次用窗口长度N来限制流水线中未完成、未被确认的分组数。然而，与GBN不同的是，发送方已经收到了对窗口中某些分组的ACK。图3-16显示了SR发送方看到的序号空间。

SR接收方将确认一个正确接收的分组而不管其是否按序。失序的分组将被缓存直到所有丢失的分组皆被收到为止，这时才可以将一批分组按序交付给上层。

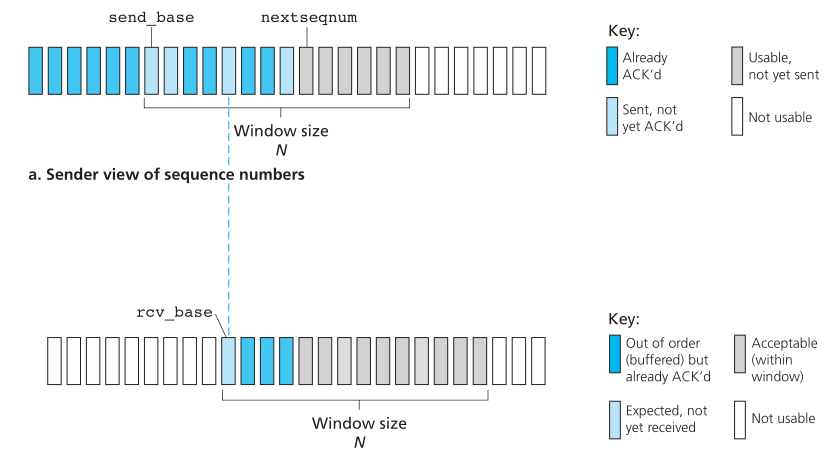


图3-23 选择重传发送方与接收方的序号空间

1. SR发送方的事件与动作

1）从上层收到数据。当从上层接收到数据后，SR发送方检查下一个可用于该分组的序号。如果序号位于发送方的窗口内，则将数据打包并发送；否则就像在GBN中一样，要么将数据缓存，要么将其返回给上层以便以后传输。

2）超时。定时器再次被用来防止丢失分组。然而，现在每个分组必须拥有其自己的逻辑定时器，因为超时发生后只能发送一个分组。可以使用单个硬件定时器模拟多个逻辑定时器的操作。

3）收到ACK。如果收到ACK，倘若该分组序号在窗口内，则SR发送方将那个被确认的分组标记为已接收。如果该分组的序号等于send\_base，则窗口基序号向前移动到具有最小序号的未确认分组处。如果窗口移动了并且有序号落在窗口内的未发送分组，则发送这些分组。

2. SR接收方的事件与动作

1）序号在[rcv\_base, rcv\_base + N - 1]内的分组被正确接收。在此情况下，收到的分组落在接收方的窗口内，一个选择ACK被回送到发送方。如果该分组以前没收到过，则缓存该分组。如果该分组的序号等于接收窗口的基序号，则该分组以及以前缓存的序号连续（起始于rcv\_base的）的分组交付给上层。然后，接收窗口按向前移动分组的编号向上交付这些分组。

2）序号在[rcv\_base - N, rcv\_base - 1]内的分组被正确收到。在此情况下，必须产生一个ACK，即使该分组是接收方以前已确认过的分组。

3）其他情况。忽略该分组。

SR协议中，窗口长度必须小于或等于序号空间大小的一半。

## 3.5 面向连接的运输：TCP

### 3.5.1 TCP连接

### 3.5.2 TCP报文段结构

### 3.5.3 往返时间的估计与超时

### 3.5.4 可靠数据传输

### 3.5.5 流量控制

### 3.5.6 TCP连接管理

## 3.6 拥塞控制原理

### 3.6.1 拥塞原因与代价

### 3.6.2 拥塞控制方法

## 3.7 TCP拥塞控制

### 3.7.1 公平性

### 3.7.2 明确拥塞通告：网络辅助拥塞控制

# 附录1 HTTP常见状态码及含义

|  |  |  |  |
| --- | --- | --- | --- |
| 状态码系列 | 状态码 | 状态信息 | 含义 |
| 1XX：通知  1XX系列响应代码仅在与HTTP服务器沟通时使用。 | 100 | Continue | 客户端应重新发送初始请求，并在请求中附上第一次请求时未提供的（可能很大或者包含敏感信息的）表示。客户端这次发送的请求不会被拒绝 |
| 101 | Switching Protocols | 当客户端通过在请求里使用Upgrade报头，以通知服务器它想改用除HTTP协议之外的其他协议时，客户端将获得此响应代码 |
| 2XX: 成功  2XX系列响应代码表明操作成功了 | 200 | OK | 表示服务器成功执行了客户端所请求的动作 |
| 201 | Created | 当服务器依照客户端的请求创建了一个新资源时，发送此响应代码 |
| 202 | Accepted | 客户端的请求无法或将不被实时处理，请求稍后会被处理 |
| 203 | Non-Authoritative Information | 这个响应代码跟200一样，只不过服务器想让客户端知道，有些响应报头并非来自该服务器--他们可能是从客户端先前发送的一个请求里复制的，或者从第三方得到的。 |
| 204 | No Content | 若服务器拒绝对PUT、POST或者DELETE请求返回任何状态信息或表示，那么通常采用此响应代码。服务器也可以对GET请求返回此响应代码，这表明“客户端请求的资源存在，但其表示是空的”。注意与304("Not Modified")的区别。204常常用在Ajax应用里。服务器通过这个响应代码告诉客户端：客户端的输入已被接受，但客户端不应该改变任何UI元素。 |
| 3XX 重定向  3XX系列响应代码表明：客户端需要做些额外工作才能得到所需要的资源。它们通常用于GET请求。他们通常告诉客户端需要向另一个URI发送GET请求，才能得到所需的表示。那个URI就包含在Location响应报头里。 | 301 | Moved Permanently | 服务器知道客户端试图访问的是哪个资源，但它不喜欢客户端用当前URI来请求该资源。它希望客户端记住另一个URI，并在今后的请求中使用那个新的URI。你可以通过这个响应代码来防止由于URI变更而导致老URI失效。 |
| 303 | See Other | 请求已经被处理，但服务器不是直接返回一个响应文档，而是返回一个响应文档的URI。该响应文档可能是一个静态的状态信息，也可能是一个更有趣的资源。对于后一种情况，303是一种令服务器可以“发送一个资源的表示，而不强迫客户端下载其所有数据”的方式。客户端可以向Location报头里的URI发送GET请求，但它不是必须这么做。 |
| 304 | Not Modified | 这个响应代码跟204("No Content")类似：响应实体主体都必须为空。但204用于没有主体数据的情况，而304用于有主体数据，但客户端已拥有该数据，没必要重复发送的情况。 |
| 4XX：客户端错误  这些响应代码表明客户端出现错误。不是认证信息有问题，就是表示格式或HTTP库本身有问题。客户端需要自行改正。 | 400 | Bad Request | 这是一个通用的客户端错误状态，当其他4XX响应代码不适用时，就采用400。此响应代码通常用于“服务器收到客户端通过PUT或者POST请求提交的表示，表示的格式正确，但服务器不懂它什么意思”的情况。 |
| 401 | Unauthorized | 客户端试图对一个受保护的资源进行操作，却又没有提供正确的认证证书。客户端提供了错误的证书，或者根本没有提供证书。这里的证书（credential）可以是一个用户名/密码，也可以是一个API key，或者一个认证令牌。 |
| 403 | Forbidden | 客户端请求的结构正确，但是服务器不想处理它。这跟证书不正确的情况不同--若证书不正确，应该发送响应代码401。该响应代码常用于一个资源只允许在特定时间段内访问，或者允许特定IP地址的用户访问的情况。 |
| 404 | Not Found | 404表明服务器无法把客户端请求的URI转换为一个资源 |
| 405 | Method Not Allowd | 客户端试图使用一个本资源不支持的HTTP方法 |
| 5XX 服务端错误  这些响应代码表明服务器端出现错误。一般来说，这些代码意味着服务器处于不能执行客户端请求的状态，此时客户端应稍后重试。 | 500 | Internal Server Error | 这是一个通用的服务器错误响应。对于大多数web框架，如果在执行请求处理代码时遇到了异常，它们就发送此响应代码。 |
| 501 | Not Implemented | 客户端试图使用一个服务器不支持的HTTP特性。 |
| 502 | Bad Gateway | 只有HTTP代理会发送这个响应代码。它表明代理方面出现问题，或者代理与上行服务器之间出现问题，而不是上行服务器本身有问题。若代理根本无法访问上行服务器，响应代码将是504。 |
| 503 | Service Unavailable | 此响应代码表明HTTP服务器正常，只是下层web服务服务不能正常工作。最可能的原因是资源不足：服务器突然收到太多请求，以至于无法全部处理。由于此问题多半由客户端反复发送请求造成，因此HTTP服务器可以选择拒绝接受客户端请求而不是接受它，并发送503响应代码。 |

1． A类IP地址

一个A类IP地址由1字节的网络地址和3字节主机地址组成，网络地址的最高位必须是“0”， 地址范围从1.0.0.0 到126.0.0.0。可用的A类网络有126个，每个网络能容纳1亿多个主机。

2． B类IP地址

一个B类IP地址由2个字节的网络地址和2个字节的主机地址组成，网络地址的最高位必须是“10”，地址范围从128.0.0.0到191.255.255.255。可用的B类网络有16382个，每个网络能容纳6万多个主机 。

3． C类IP地址

一个C类IP地址由3字节的网络地址和1字节的主机地址组成，网络地址的最高位必须是“110”。范围从192.0.0.0到223.255.255.255。C类网络可达209万余个，每个网络能容纳254个主机。

4． D类地址用于多点广播（Multicast）。

D类IP地址第一个字节以“lll0”开始，它是一个专门保留的地址。它并不指向特定的网络，目前这一类地址被用在多点广播（Multicast）中。多点广播地址用来一次寻址一组计算机，它标识共享同一协议的一组计算机。

5． E类IP地址

以“llll0”开始，为将来使用保留。

在IP地址3种主要类型里，各保留了3个区域作为私有地址，其地址范围如下：

A类地址：10.0.0.0～10.255.255.255

B类地址：172.16.0.0～172.31.255.255

C类地址：192.168.0.0～192.168.255.255

另外IPv6地址长度为128位，是IETF（互联网工程任务组，Internet Engineering Task Force）设计的用于替代现行版本IP协议（IPv4）的下一代IP协议。