第 1 章：计算机网络和因特网

什么是因特网：

具体构成描述：终端称**主机**或**端系统**（运行应用程序）。 端系统通过**通信链路和分组交换机**连接到一起；链路的**传输速率**或**带宽**以比特每秒（bps）度量；分组交换机从它的一条入链路接收到达的分组，并从它的一条出通信链路转发该分组，最著名的分组交换机是**路由器**（通常用于网络核心）和**链路层交换机**（通常用于接入网）。 当端系统向另一端系统发送数据时，发送端将数据分段，并为每段加上首部字节，形成分组。 从发送到接收端系统，一个分组所经历的一系列通信链路和分组交换机称为通过该网络的路径。 端系统通过**因特网服务提供商（ISP）**接入因特网，每个 ISP 是由多个分组交换机和多段通信链路组成的网络，各 ISP 为端系统提供了各种不同类型的网络接入，也为内容提供者提供因特网接入服务，每个 ISP 都是独立管理的，ISP 之间是互联的。 端系统、分组交换机和其他因特网部件都要运行一系列协议，这些协议控制因特网中信息的发送和接收（终端与终端，终端与交换设备，交换设备与交换设备）。 TCP 和 IP 是因特网中最重要的两个协议，因特网中的主要协议统称为 TCP/IP。 **因特网标准**由**因特网工程任务组（IETF）**研发，IETF 的标准文档称为**请求评论（RFC）**。 **因特网定义一：**由一群遵循 TCP/IP 协议的 ISP 按照松散的层次结构组织而成的网络，这定义了对于通信功能的实现有指导作用：ISP 内部和之间互联。 **因特网的几个特点：**因特网是网络的网络，不存在严格的层次结构，没有统一的管理机构。

服务描述：**因特网定义二：**为分布式应用提供通信服务的基础设施，该定义对于服务接口的定义有指导作用：有序可靠的数据交付服务&不可靠的数据交付服务。 因特网**分布式应用程序**涉及多台相互交换数据的端系统，并不运行在网络核心的分组交换机中。 与因特网相连的端系统提供了**应用程序编程接口（API）**，该 API 规定了运行在一个端系统上的软件请求因特网基础设施向运行在另一个端系统上的特定目的地软件交付数据的方式。

什么是协议：一个协议定义了在两个或多个通信实体之间交换的报文格式和次序，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。

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成两类：**客户**和**服务器**。

接入网：**接入网**是指将端系统连接到其**边缘路由器**的物理链路，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。 宽带住宅接入有两种最流行的类型：数字用户线和电缆。 **数字用户线（DSL）(一线分频三用)：**使用已有的**数字电话线**（双绞铜线，每户一条线），同时（从高频到低频）承载高速下行信道、中速上行信道和普通的双向电话信道；电话公司为 ISP；DSL 调制解调器得到数字数据后将其转换为高频音；近距离 5~10 英里内接入。 **电缆因特网接入：**利用有线电视公司现有的**有线电视**基础设施，同时应用了光缆和同轴电缆，称为混合光纤同轴（HFC)，同时还有电缆调制解调器（cable modem）和电缆调制解调器端接系统（CMTS）；一个重要特征是共享广播媒体 **以太网（Ethernet）：**公司和大学校园以及有较多终端的家庭使用局域网（LAN）；以太网是最为流行的接入技术，使用双绞铜线；以太网交换机及链路构成接入网 **广域无线接入（3G、4G）：**移动通信公司提供，使用现有的蜂窝电话网络；数万米；3G 2Mbps，4G 下行 100Mbps 上行 20Mbps。

物理媒体：对于每个传输器-接收器对，通过跨越一种**物理媒体**传播电磁波或光脉冲来发送该比特。物理媒体可以有各种形状和形式，并且对沿途的每个传输器-接收器对而言不必具有相同的类型。 物理媒体划分为**导引型媒体**和**非导引型媒体**，沿着固体媒体/在空气或外层空间中传播。 **双绞线：**两根隔离的铜线，常用在 LAN、电话线网线，10Mbps 到 10Gbps，速率取决于线的粗细与传输距离。 **同轴电缆：**两根同心的绝缘铜导体，有线电视电缆，能作导引型共享媒体。 **光纤：**导引光脉冲的玻璃纤维，几十~几百 Gbps，抗电磁干扰，长距离传输低衰减，难窃听，低误码率。 **电磁波：**无物理线路，并具有穿透墙壁、提供与移动用户的连接以及长距离承载信号的能力；蓝牙（10m），WiFi（几十米），红外（室内短距离），陆地微波（长距离），卫星（长距离大范围），可见光（正在研究中）。

网络核心：

网络核心是由互联因特网系统的分组交换机和链路构成的网状网络。网络核心的任务是将数据包从发送侧的边缘路由器传送到接收侧的边缘路由器。基本问题：数据包如何在网络核心中高效地传递（分组传输延迟小，网络吞吐量高）？

分组交换：源将长报文划分为较小的数据块，称为**分组**。 分组以等于该链路的最大速率传输通过通信链路，L(bit 的分组) / R(bit/s 链路传输速率)秒。 **存储转发传输机制：**在交换机能够开始向输出链路传输该分组的第一个 bit 之前必须接收到整个分组。 每个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（输出队列）**，它用于存储路由器准备发往那条链路的分组。 分组还要承受输出缓存的**排队时延**，这是变化的，取决于网络拥塞程度。 因为缓存空间的大小是有限的，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*。当大量分组集中到达时，排队延迟和丢包较严重。 当一个分组到达网络中的路由器时，路由器检查该分组的目的地址的一部分，并向一台相邻路由器转发该分组。每台路由器具有一个**转发表**，用于将目的地址（或一部分）映射成输出链路。 因特网具有一些特殊的**路由选择协议**，用于自动地设置这些转发表（例如最短路径）。

电路交换：通过网络链路和交换机移动数据有两种基本办法：**电路交换**和**分组交换**。电路交换网络中在通信期间***预留了***端系统间通信资源（缓存，链路传输速率）（在分组交换网络中，报文按需使用这些资源，后果是可能不得不等待排队）。 传统的电话网是电路交换的例子。 该链接被称为一条**电路**，网络创建时预留了恒定的速率，则发送方能以**确保**的恒定速率向接收方传送数据。区分：链路是物理媒体，也称信道，可以通过某种方式划分为若干条独立的子信道；电路是物理媒体中的一条子信道。 当两台主机要通信时该网络在两主机之间创建一条专用的**端到端连接**。 链路中的电路是通过**频分复用（FDM）**或**时分复用（TDM）**实现的：FDM 的链路为每个连接专用一个频段，频段的宽度称为**带宽（bandwidth）**；TDM 中时间被划分为固定区间的**帧**，每帧被划分为固定数量的**时隙**，帧的每个时隙为某个连接专门使用，TDM 电路的传输速率=帧速率*一个时隙中的 bit 数。 **电路交换的缺点：静默期**转用电路空闲而效率较低；创建端到端电路和预留端到端带宽是复杂的，需要复杂的指令软件以协调沿端到端路径交换机的操作。 **分组交换的缺点：**分组交换不适合实时服务，因为其端到端时延是可变和不可预测的（可能丢包），并且不能保证服务质量（带宽）。 **分组交换的优点：**提供更好的带宽共享；比电路交换更简单更有效，实现成本更低。 分组交换适合突发数据。

网络的网络：**接入 ISP**；**地区 ISP**；**第一层 ISP**；**存在点（PoP）：**低层 ISP 接入高层 ISP 的地方；**多宿：**一个低层 ISP 可以接入多个高层 ISP；**对等：**相同层次的 ISP 连到一起以试图绕过上层；**因特网交换点（IXP）：**第三方公司创建，多个 ISP 可在这里对等；**内容提供商网络**。

分组交换中的时延、丢包和吞吐量：

分组交换网中的时延概述：当分组从一个结点沿着这条路径到后继结点，该分组在沿途的*每个*结点经受了几种不同类型的时延：**结点处理时延**，**排队时延**，**传输时延**，**传播时延**，累加起来是**结点总时延**。 **处理时延：**检查分组首部和决定将该分组导向何处，检查 bit 级别的差错（微秒或更低）。 **排队时延：**分组在链路上等待传输时。取决于先前到达的排队的分组数量和拥塞程度，是流量强度和性质的函数。(差异很大，毫秒到微秒) **传输时延：**L bit 长度的分组，在 R bps 的链路上，传输时延是 L/R，是将所有比特推向链路所需要的时间。(毫秒到微妙) **传播时延：**从该链路的起点到终点传播所需要的时间，速率取决于物理媒体，范围是 2/3~1 倍光速，传播时延是 d/s，d 距离，s 速率。(百毫秒到几微秒)。总时延 d_nodal=d_proc（处理）+d_queue（排队）+d_trans（传输）+d_prop（传播），这些成分所起的作用可能差异很大（如卫星和大学）。

排队时延和丢包：不同于其他三个，排队时延对不同的分组可能是不同的。表征排队时延一般使用统计量测度。 a 表示分组到达队列的平均速率（分组每秒），R 传输速率 bps，L 分组大小 bit，L*a/R 称为**流量强度**。 流量强度>1，排队时延趋向无穷大，*设计系统时流量强度不能大于 1*；流量强度<=1，到达流量的性质影响排队时延，分组周期性到达平均时延低，突发一次到达多个则平均时延高。 **平均排队时延和流量强度的定性关系：**随着流量强度接近于 1，平均排队时延迅速增加，该强度少量的增加将导致时延大得多的增加。 一条链路前的队列只有有限的容量，尽管排队容量极大的依赖路由器的设计和成本。因为该排队容量是有限的，随着流量强度接近 1，排队时延并不实际趋向无穷大。相反，到达的分组将发现一个满的队列，路由器**丢弃**该分组，即该分组将**丢失**。 分组丢失的份额随着流量强度的增加而增加。 一个结点的性能常常不仅根据时延来度量，而且根据分组丢失的概率来度量。

端到端时延：**端到端时延：**分组传输路径上所有结点的节点延迟之和。 对端到端延迟敏感的应用：高度敏感（实时交互应用如网络电话视频会议），中度敏感（在线交互应用如网页浏览）。 其他一些重要的时延：作为它协议的一部分，希望向共享媒体传输分组的端系统可以有意地延迟它的传输以与其他端系统共享媒体。 媒体分组化时延：经 IP 语音应用中，发送方在想因特网传递分组之前必须首先用编码的数字化语音填充一个分组，这个时间为分组化时延。

计算机网络中的吞吐量：在任何时间瞬间的**瞬时吞吐量**是目的主机接收到数据的速率（bps），文件传输的**平均吞吐量**是文件总 bit 数 / 总时间。 吞吐量是 min { R1 , R2 , ⋯ , Rn } 即为**瓶颈链路**的传输速率。10 个公共链路吞吐量是 min { Rc , Rs , R/10 }。 吞吐量取决于数据流过的链路的传输速率，不仅取决于沿着路径的传输速率，而且取决于干扰流量。特别是，如果许多其他的数据流也通过这条链路流动，一条具有高传输速率的链路仍然可能称为传输的瓶颈链路。

协议层次及其服务模型：

分层的体系结构：每个层次与其下面的层次结合在一起，实现了某些功能，*服务*。 **系统分层：**将系统按功能划分为一系列水平的层次，每一层实现一个功能（服务）。 **层次间关系：**每一层的功能实现都要依赖其下各层提供的服务。 **每个层次通过如下方式提供服务：**1. 在该层中执行了某些动作 2. 使用直接下层的服务。 **分层的好处：**显示的层次结构易于确定系统的各个部分及其相互关系，模块化简化了系统的维护和升级（改变某层服务的实现方式对于其他层次没有影响）。 网络设计者以**分层**的方式组织协议以及实现这些协议的网络硬件和软件，每个协议属于这些层次之一。某层向它的上一层提供的**服务**，即所谓一层的**服务模型**。 一个协议层能够用软件、硬件或二者的结合来实现（应用层和运输层是端系统的软件实现，物理层和数据链路层是在与链路相联系的网络接口卡中实现，网络层是硬软件混合）。 一个第 n 层协议也**分布**在构成该网络的端系统、分组交换机和其他组件中，也就是说第 n 层协议的不同部分常常位于这些网络组件的各部分中。 协议分层具有概念化和结构化的优点，分层提供了一种结构化方式来讨论系统组件，模块化使更新系统组件更为容易。 **分层的缺点：**一层可能冗余较低层的功能；某层的功能可能需要仅在其他某层才出现的信息，这违反了层次分离的目标。 各层的所有协议被称为**协议栈：物理层、链路层、网络层、运输层和应用层。** **应用层：**网络应用程序及它们的应用层协议存留的地方。支持各种网络应用.分布在多个端系统上，一个端系统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的分组：**报文（message）**。 **运输层：**在应用程序与网络的接口间（进程-进程）传输**报文段（segment）**。进程进程之间的分组传输. **网络层：**负责将称为**数据报（datagram）（PPT 称为分组 packet）**的网络层分组从一台主机移动到另一台主机。源主机-目的主机之间的分组传输. **链路层：**在相邻设备（结点）之间传输**帧（frame）**。相邻网络设备之间的分组传输 **物理层：**将帧中的一个一个**比特**从一个节点移动到下一个节点（物理媒体上）。物理媒介实现. **ISO/OSI：**应用层，表示层，会话层，运输层，⋯⋯；这两层留给应用程序开发者处理。

封装：与端系统类似，路由器和链路层交换机以多层次的方式组织他们的网络硬件和软件。而路由器和链路层交换机并不实现协议栈中的所有层次。链路层交换机实现了物理层和链路层，路由器实现了物理层链路层和网络层。主机实现了所有五个层次。 这体现了**封装**的概念。 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 2 章：应用层

网络应用是计算机网络**存在的理由。**

应用层协议原理：

研发新应用程序时，你需要编写将在多台端系统上运行的软件，并能通过网络相互通信。重要的是，你不需要也不能写在网络核心设备如路由器或链路层交换机上运行的软件。将应用软件限制在端系统的方法，促进了大量的网络应用程序的迅速研发和部署。

网络应用程序体系结构：应用程序体系结构明显不同于网络的体系结构，在应用程序研发者的角度看，后者是固定的，并为应用程序提供了特定的服务集合。网络**应用程序体系结构**由应用研发者设计，规定了如何在各种端系统上组织该应用程序。 现代网络应用程序的两个主流体系结构：**客户-服务器体系结构**和**对等（P2P）体系结构**。 **客户-服务器体系结构的特点：**在客户-服务器体系结构中，有一台总是打开（在线）的主机称为**服务器**，可能使用主机集群或数据中心提高处理能力。服务器主机具有固定的、周知的地址，该地址称为 IP 地址。客户之间不直接通信。 **客户：**用户终端上运行一个客户程序（client），需要时与服务器程序通信，请求服务。客户机使用动态 IP 地址，通常不会总是在线。 客户-服务器架构的**资源集中：**资源（服务）只在某些固定的终端上提供。**优点**：资源发现简单。**缺点**：集中式计算带来的问题如服务器扩容压力、网络流量不均衡、响应延迟长。 在一个 **P2P 体系结构**中，对位于数据中心的专用服务器具有最小的（或者没有）依赖。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**。对等方间断连接,使用动态 IP 地址,每个对等方可以是客户也可以是服务器. **P2P 特性：**自扩展性；是成本有效的，因为通常不需要庞大的服务器基础设施和服务器带宽。 **P2P 应用面临的挑战：**ISP 友好（大量上载给 ISP 带来巨大压力）；安全性；激励（说服用户自愿像应用提供带宽、存储和计算资源）。 P2P 架构**资源分散：**任何终端都可以提供资源（服务）；**优点：**易于扩容、均衡网络流量；**缺点：**资源发现困难，社会问题（版权、安全性）。

进程通信：进行通信的实际上是**进程**而不是应用程序。进程：主机上运行的程序。在分布式应用中，*不同*终端上的进程需要通信。 **进程通信的方法：**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。同一主机内使用 OS 提供的进程间通信机制。对每对通信进程，将发起请求的进程标识为**客户**，另一个在会话开始时等待联系的（接受请求的）进程标识为**服务器**。 进程通过一个称为**套接字（socket）**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**应用层与运输层的接口。也称为应用程序和网络之间的**应用程序编程接口（API）(对应用层控制一切,对运输层没有控制权(分层思想))**。套接字类似于一扇门,发送进程将报文推出套接字,套接字外的运输设施（因特网）将报文送到接收进程的门口,通过接收进程的套接字传递,并进行处理 为了标识该接收进程，需要定义两种信息：主机的地址；定义在目的主机中的接收进程标识符。在因特网中，主机用其 **IP 地址**标识，接收进程由目的**端口号**标识。众所周知的端口号被分配给服务器，成为服务的标识（HTTP 80，SMTP 25）

可供应用程序使用的传输服务：从四个方面对应用程序的服务要求进行分类：可靠数据传输、吞吐量、定时、安全性。 **可靠数据传输：**确保数据从一端正确、完全地交付给应用程序的另一端。**容忍丢失的应用**（多媒体应用），文件传输要求完全可靠的数据传输。 **吞吐量：**运输层协议能够以某种特定的速率提供确保的可用吞吐量，**带宽敏感的应用**（多媒体应用），**弹性应用**（电子邮件、文件传输与 web 传送）。 **定时：**例如发送方注入进套接字中的每个 bit 到达接收方套接字不迟于多少时间。交互式实时应用程序（因特网电话、虚拟环境、电话会议和多方环境），非实时的应用没有严格的约束。 **安全性：**比如运输协议能够加密发送进程传输的所有数据，然后在交付接收进程之前解密；也包括数据完整性和端点鉴别。

因特网提供的传输服务：因特网（更一般的是 TCP/IP 网络）*提供* **UDP** 和 **TCP** 两个运输层协议。 TCP 服务模型包括**面向连接服务**和**可靠数据传输服务**：握手过后创建一个 **TCP 连接**，连接是全双工的并且结束发送时必须拆除连接；无差错、按顺序交付所有发送的数据。还具有**拥塞控制**机制（不一定对通信进程带来直接好处，但能对因特网带来整体好处；公平共享网络带宽）和**流量控制**（发送进程不会“压垮”接收进程）。**不提供：**及时性、最低带宽保证，安全性。 TCP 的加强版本**安全套接字层（SSL）**提供了 TCP 所能提供的一切以及进程到进程的安全性服务（加密，数据完整性和端点鉴别），这种强化是在应用层实现的。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：**无连接，不可靠数据传输**（可能无法到达或乱序到达），没有拥塞控制机制（发送进程可以以任何速率向网络层注入数据），没有安全性。 **无论 UDP 还是 TCP 都不提供：吞吐量和定时保证**。应用程序被良好设计以对付这种保证的缺乏。 能够容忍某些丢失但要求达到一定最小速率才能有效工作的应用（如因特网电话）使用 UDP 以设法避开 TCP 的拥塞控制机制和分组开销。

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，特别是应用层协议定义了：交换的报文类型（请求/响应）；各种报文类型的语法（各个字段以及字段如何描述）；字段的语义（各字段中信息的语义）；一个进程何时以及如何发送报文，对报文进行响应的规则。 有些应用层协议是由 RFC 文档定义的，因此他们在公共域中，一些别的应用层协议（如 Skype）是专用的，有意不为公共层使用。 区分网络应用和应用层协议是重要的，应用层协议只是网络应用的一部分。

Web 和 HTTP：

HTTP 概况：web 的应用层协议是**超文本传输协议（HTTP）**。 HTTP 由两个程序实现：一个客户程序和一个服务器程序，二者分布在不同的端系统通过 HTTP 报文会话。 **Web 页面（也叫文档）是由对象组成的**，文件都是对象（HTML，JPEG），多数 web 页面含有一个 **HTML 基本文件**以及几个引用对象（一个有 1 个 HTML 基本文件和 5 个图形的 web 页面有共 6 个对象）。HTML 基本文件通过对象的 URL 地址引用页面中的其他对象，每个 URL 地址由两部分组成：存放对象的服务器主机名（www.someSchool.edu）和对象的路径名（/someDepartment/picture.gif）。**Web 浏览器**实现了 HTTP 的客户端，接收和显示 web 对象.**web 服务器**实现了 HTTP 的服务器端，它用于存储 Web 对象，应客户请求发送对象,每个用 URL 寻址。 **HTTP** 定义了 web 客户向 web 服务器请求，以及服务器向客户传送 web 页面的方式。 **HTTP 使用 TCP**。流程:客户机发起到服务器 80 端口的 TCP 连接(创建套接字)-服务器接收来自客户的 TCP 连接(创建套接字)-交换 HTTP 报文-关闭 TCP 连接 HTTP 服务器并不保存关于客户的任何信息，所以 HTTP 是一个**无状态协议(e.g.客户连续请求同一对象服务器都会做出反应)**。 服务器周知端口号：**80** **网络应用架构：客户-服务器架构**

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**；**有多少对象就产生多少 TCP 连接**. **持续连接：所有的请求及其响应经相同的 TCP 连接发送，服务器发送响应报文之后保持连接,后续 HTTP 报文可以继续使用该连接**.HTTP 1.0 使用非持续连接，HTTP 1.1 缺省使用持续连接。 非持续连接例子：客户在 80 端口发起服务器 TCP 连接->客户经连接发送一个请求报文->服务器接受请求，发送响应(此时双方都有一个套接字与该连接相关联)->服务器通知断开连接->客户接收响应，连接关闭。对每个对象重复一次步骤。每个 TCP 连接在服务器发送一个对象后关闭。 **往返时间（RTT）**是指一个*短（不考虑传输时延）*分组从客户到服务器然后再返回客户所花费的时间（包括传播、排队和处理时延） 非持续连接传输每个对象**需要两个 RTT+传输时延**。N 个对象即 2nRTT **非持续连接的缺点：**为每个请求的对象建立和维护一个全新的 TCP 连接，每个都要维护缓冲和变量，为服务器带来负担；每个对象经受 2RTT 的交付时延；浏览器要打开多个 TCP 连接来获取一个网页。 在采用**持续连接**的情况下，服务器再发送响应后保持该 TCP 连接打开，后续请求和响应报文能够通过相同的连接进行传送。特别是，一个完整的页面可以用单个 TCP 连接传送；更有甚者，同一服务器同一客户的多个页面使用一个连接。一般来说，一条连接经过一定时间间隔仍未使用 HTTP 就把他关闭。HTTP 默认使用**带流水线**的**持续连接**。无流水线:客户仅当收到前一个响应报文再发送新请求,n 个对象用时(n+1)RTT 流水线方式:客户每解析到一个引用对象(如超链接)就可以发送请求,可在一个 RTT 时间内请求所有引用对象,请求一个网页用时 3RTT

HTTP 报文格式：**ASCII** 文本组成；每行用 crlf，最后一行附加一个额外 crlf 结束；第一行为**请求行**：方法字段（**GET**（绝大部分使用,获取一个对象;或是上传表单输入,输入内容放在请求的 URL 里面），**POST(**上传表单输入,输入内容放在实体主体里面(如向搜索引擎提供关键词)**)**，**HEAD(**类似于 GET, 服务器不返回对象,只用一个报文头响应(实体为空),用于故障跟踪,调试**)**，**PUT(**将文件放在报文实体中,传到 URL 字段指定的路径**)**，**DELETE(**删除 URL 字段指定的文件**)**），URL 字段（请求对象的标识），HTTP 版本；之后的几行是**首部行**：Host（对象所在的主机），Connection close（不持续连接），User-agent（用户代理浏览器版本），Accept-language（没有这个语言的就发送默认版本）。额外的 crlf 表示首部行结束. 空行. **实体主体（entity body）**：GET 为空，POST 时不空 **HTTP 响应报文**包含**状态行**（协议版本，状态码，相应状态信息: 200 OK；301 Moved Permanently 请求的对象已永久转移,客户软件可自动获取新的 URL;400 Bad Request 不能被服务器理解；404 Not Found 请求的文档不在服务器上；505 HTTP Version Not Supported），**首部行**（Date 服务器产生这个报文的时间;Last-Modified 对象创建或最后修改的时间；Contest-Length; Contest-Type 对象类型应该由这个而不是扩展名标识）,空行,**实体体**（主要部分，包含了所请求对象的本身）.

用户与服务器的交互：Cookie：HTTP 无状态简化了服务器的涉及，允许工程师开发高性能 Web 服务器。**cookie** 允许站点对用户跟踪,识别用户,保存状态以将内容与用户身份联系，或限制用户的访问。 **cookie 技术的四个组件：**HTTP 响应报文的一个 Cookie 首部行；在 HTTP 请求报文中的一个 cookie 首部行；在用户端系统浏览器中保留管理一个 cookie 文件；Web 站点的一个后端数据库。 例子：A 第一次访问某网站，HTTP 请求到达网站时，网站为其创建：一个 ID，后端数据库中为该 ID 建立一个表项。**服务端：**信息保存在服务端的后端数据库，返回 ID 给客户。**客户端：**信息发回客户端，保存在 cookie 文件中，并随请求报文发送给服务器。 **cookie 可以**标识一个用户，所以 cookie 可以在无状态的 HTTP 之上建立一个用户会话层，但 cookie 允许网站收集用户的大量信息，带来**隐私问题**。

Web 缓存：**Web 缓存器**也叫**代理服务器**，它是能代表初始 Web 服务器来满足 HTTP 请求的网络实体。Web 缓存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。 用户设置浏览器，所有 HTTP 请求首先发往 web 缓存。**浏览器将 HTTP 请求发送给 web 缓存：**对象在 web 缓存中:，web 缓存返回对象 ；对象不在 web 缓存中，web 缓存从原始服务器获取对象，缓存在本地，然后返回给客户。 Web 缓存器既是服务器同时也是客户。 Web 缓存器通常由 ISP 提供，多级 ISP 可能形成多级 ISP 缓存。 部署 Web 缓存器的**原因：**大大减少客户请求的响应时间；减少一个机构的接入链路到因特网的通信量（降低了费用）；从整体上大大减低因特网上的 Web 流量，从而改善了所有应用的性能。

条件 GET 方法：存放在 Web 缓存器里的对象副本有可能是陈旧的。解决这个问题的机制是**条件 GET：**请求报文使用 GET 方法，请求报文中包含一个 If-Modified-Since:首部行。缓存器在缓存对象时也存储了最后修改日期。 **若对象无更新**，条件 GET 的响应报文没有包含该对象（否则只会浪费带宽），状态行为 **304 Not Modified**；若**有更新**，200 OK

文件传输协议：FTP

用户通过一个 FTP 用户代理与 FTP 交互。 TCP 连接。 FTP（和 HTTP 一样）是文件传输协议。 FTP 使用了两个并行的 TCP 连接来传输文件，一个是**控制连接（21 端口）**，一个是**数据连接（20 端口）**。 使用 7bit 的 ASCII 格式在控制连接上传送**命令/响应交互**（不是 HTTP 那样使用报文交互），为了区分连续的命令，每个命令后紧跟回车换行符。每个命令由 4 个大写 ASCII 组成，有些包含可选参数。 每一次数据连接只传送一个文件，发送方用关闭连接表示一个文件传输结束。 **将控制连接与数据连接分开：**不会混淆数据与命令/响应，简化协议设计和实现；在传输文件的过程中可以继续执行其它的操作，便于控制传输过程（如客户可以随时终止传输）。 **用关闭数据连接的方式结束文件传输：**允许动态创建文件（不需预先告知文件的大小）。 **常见命令：**USER username；PASS password；LIST（返回响应的文件列表是经过一个独立数据连接传送的而非使用控制连接）；RETR filename（传给用户文件）；STOR filename（用户存储文件）。 **回答**是一个 3 位数字，后跟可选信息：331 Username OK，Password required；125 Data connection already open; transfer starting；425 Can't open data connection；452 Error writing file。

因特网中的电子邮件：

3 个主要组成部分：**用户代理**，**邮件服务器**，**简单邮件传输协议（SMTP）**。 用户代理允许用户阅读、回复、转发、保存和撰写报文。 用户代理向邮件服务器发送邮件，此时邮件放在**邮件服务器**的**外出报文队列**中。邮件服务器是电子邮件体系结构的核心，每个接收方在其中的某个邮件服务器上有一个**邮箱**（计算机上的存储区域组成，每个信箱均被分配了唯一的**电子邮件地址**，电子邮件地址包括：标识用户信箱的字符串和@之后的邮件服务器的名字）。邮件服务器还包含**报文传输代理 MTA：**运行在服务器后台的系统守护进程，负责在邮件服务器之间传输邮件，及将收到的邮件放入用户信箱。 **一个典型的邮件发送过程是：**从发送方的用户代理开始，传送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。 发送者的邮箱也必须能处理接收者邮件服务器的故障，如果不能交付邮件，则邮件放在一个发送者的邮件服务器的**报文队列**中保持该报文并在以后尝试再次发送。

SMTP：**SMTP** 是因特网电子邮件中主要的**应用层**协议，**使用 TCP**，**端口 25**，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。(每台服务器既是客户也是服务器)。 SMTP **限制**所有邮件报文的体部分（不只是首部）只能**采用简单的 7 比特 ASCII 表示**。 SMTP **不使用中间邮件服务器**传送邮件，即使两个邮件服务器相聚很远，这意味着邮件不会在中间某个邮件服务器存留。 **采用命令/响应交互方式：**命令为 ASCII（HELO，MAIL FROM，RCPT TO，DATA，QUIT）,客户通过一个只包含一个句点的行（crlf.crlf）指示 DATA 报文结束。响应为状态码和可选英文解释短语。 使用**持续连接：**几个发往同一邮件服务器的报文共用一个连接。那么每个报文以新的 MAIL FROM 开始，以独立句点结束。所有都发完了才 QUIT。

与 HTTP 的对比：都用于从一台主机向另一台主机传送文件，都用持续连接；HTTP 主要是一个**拉协议**，而 SMTP 基本上是一个**推协议**。SMTP 要求**所有报文都是 ASCII**，非字符数据要编码成字符，HTTP 无此限制。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。

邮件报文格式和 MIME：**电子邮件报文格式：首部行**和**报文体**用空行（crlf）进行分割；首部行包括：From:，To:，(可以包含 Subject:和一些其他的)。这些首部行和 SMTP 命令是有区别的，现在讲的首部行是邮件报文的一部分。 **Base64 编码：**每 24 比特数据划分成 4 个 6 比特的单元，每个单元编码成一个 ASCII 字符，其对应关系为：0 ~ 25 编码成‘A’ ~ ‘Z’；26 ~ 51 编码成‘a’ ~ ‘z’；52 ~ 61 编码成‘0’ ~ ‘9’；62 和 63 分别编码成‘ + ’和‘ / ’；若最后一组只有 8 比特或 16 比特，分别加上‘ = = ’和‘ = ’后缀；回车和换行忽略，可以插在任何地方。 **quoted-printable 编码：**适用于绝大部分都是 ASCII 字符的报文实体，其编码方法是：每个 ASCII 字符保持不变；对于非 ASCII 字符（大于 127 的字符），将该字符的十六进制表示用两个 ASCII 字符标记，前面冠以特殊字符“=” 。 **多用途因特网邮件扩展协议 MIME：**扩展了 RFC 822，允许实体具有不同的数据类型，并规定了非 ASCII 文本信息在传输时的统一编码形式。扩充了一些首部行，MIME version:，最重要的是：Content-Transfer-Encoding: 实体采用的传输编码形式；Content-Type:实体的数据类型及子类型。

邮件访问协议：**邮件访问方式：**早期用户登陆到邮件服务器上，直接在服务器上运行一个邮件阅读程序来阅读邮件；今天，用户在终端上安装用户代理，获取和阅读邮件。 不能将用户信箱放在本地终端，因为用户终端不可能一直连在因特网上。 **用户代理可以用 SMTP 将邮件推向邮件服务器**，但 SMTP 不能从邮件服务器把邮件拉下来到用户代理。 引入邮件访问协议将邮件服务器上的报文传到本地 PC：**第三版的邮局协议（POP3）**，**因特网邮件访问协议（IMAP）**，以及 HTTP。 **POP3：TCP**，**110** 端口；有三个**工作阶段**：特许（代理明文传输用户名和密码）、事务处理（代理取回报文，对报文做删除标记，取消报文删除标记，获取邮件统计信息）、更新（quit 之后服务器删除被标记的报文）。**命令：**特许阶段 user，pass；事务处理 list，retr，dele，quit。**回答：**+OK（有时后面跟有数据），-ERR。**下载并删除**（其他终端不能再看到）和**下载并保留**（其他终端可以再获取）。一行单独的句点标识报文的结束。POP3 服务器保留了一些状态信息，特别是记录哪些用户报文被标记为删除了，但 POP3 服务器并不在 POP3 会话中携带状态信息，这简化了协议的实现。 **IMAP：**所有邮件保存在服务器上，允许用户将邮件组织在文件夹中，允许用户在文件夹之间移动邮件，维护了会话的用户状态信息，允许用户代理获得报文的一部分。 **HTTP**：使用 **HTTP 在用户代理和服务器之前上传和下载邮件**，而邮件服务器之间仍然是 SMTP。

DNS：因特网的目录服务

主机的一种标识方法是用它的**主机名**（www.yahoo.com），另一种是 **IP 地址**（32bit）。

DNS 提供的服务：**域名系统（DNS）**是：一个由分层的 **DNS 服务器**实现的分布式数据库；一个使得主机能够查询分布式数据库的应用层协议。 DNS 运行在 **UDP** 上，端口 **53**（也可以是 TCP，端口号 53）。 **DNS 在应用层上的原因：**使用客户服务器模式运行在通信的端系统之见，在通信的端系统之间通过下面的端到端运输协议来传送 DNS 报文。 **DNS 与其他应用的不同之处：**不是直接和用户打交道的应用，而是为因特网上的用户应用程序以及其它软件提供一种核心功能。 DNS 给使用它的应用带来了额外的延时。 **DNS 的服务：主机名到 IP 地址的转换；主机别名到规范主机名的映射**，迁移服务不需要修改主机名；**邮件服务器别名**，允许使用域名作为邮件服务器的别名；**负载分配**，允许一个规范主机名对应一组 IP 地址，将服务请求分配到功能相同的一组服务器上。

DNS 工作机理概述：应用程序（如浏览器）调用一个本地 DNS 客户端（解析器），主机名作为参数之一传递；解析器向网络中的 DNS 服务器发送查询报文，包含要查询的主机名；解析器收到包含 IP 地址的响应报文；解析器将 IP 地址返回给调用者（如浏览器）。 **不使用集中式 DNS 的原因：**单点故障，通信容量（单个服务器处理全部查询），远距离的集中式数据库（较远的地方会有高时延），维护（单个服务器维护庞大的数据库）。没有可扩展能力。 **三种类型的 DNS 服务器：根** DNS 服务器（最高层，13 个），**顶级域（TLD）**DNS 服务器（次高层，分为组织域（com，org），国家域（uk，fr）和反向域（域名 arpa，用来把一个 ip 地址映射为名字）），**权威** DNS 服务器（低层，每个组织机构，可以组织自己维护也可以由 ISP 维护）。DNS 服务器以**层次**方式组织。还有**本地 DNS 服务器**，它通常邻近本主机，起着代理的作用，并将请求转发到 DNS 服务器的层次结构中。 **DNS 名字树：域（domain）：**名字树中一个特定的节点以及该节点下所有的节点构成一个域。**标记（label）：**树上每一个节点都有一个标记（最多 63 个字符），树根的标记是一个空字符串。**域名（domain name）：**某个域的名字表示为从该域开始向上直到树根的标记序列，标记之间用句点隔开（类似国外邮政地址的写法）。域名的任一后缀也是一个域，同一个机构内的主机具有相同的域名后缀。*每个节点只需保证其孩子节点的标记不重名*。 **查询示例：**DNS 客户查询根服务器，得到 com 域的 TLD DNS 服务器地址；DNS 客户查询 com 域的 DNS 服务器，得到 amazon.com 域的权威 DNS 服务器地址；DNS 客户查询 amazon.com 域的 DNS 服务器，得到 www.amazon.com 的 IP 地址。 TLD 服务器一般只是知道中间的某个 DNS 服务器，该中间 DNS 服务器依次才能知道用于该主机的权威 DNS 服务器。 **递归查询：**高负载。**迭代查询：**收到查询报文的服务器将下一个需要查询的服务器地址返回给查询者。实践中查询通常是，从请求主机到本地 DNS 服务器的查询是递归的，其余的查询是迭代的。 整个 **DNS 名字空间**被划分为一些不重叠的区域，称为 **DNS zone**，每个 zone 包含域名树的一部分；在管理上，每个 zone 代表一个权威域的边界。 一个物理服务器保存的信息可能涉及域名空间的若干层，它也可以把它的域划分成若干子域，把其中的一些子域委托给其它服务器。实际的物理服务器的层次与域名空间的逻辑层次不同。 **DNS 缓存：**每当收到一个响应报文，DNS 服务器将报文中的映射信息**缓存在本地**。DNS 服务器首先使用缓存中的信息响应查询请求。DNS 缓存中的映射在一定时间后被**丢弃。**特别地，本地 DNS 服务器通常会缓存 TLD 服务器的 IP 地址，因而很少去访问根服务器。

DNS 记录和报文：所有 DNS 服务器存储了**资源记录（RR）**，每个 DNS 报文包含了一条或多条资源记录。 资源记录是一个包含了下列字段的**四元组：（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；**Type=A**，Name：主机名，Value：IP 地址；**Type=NS**，Name：域 (e.g. foo.com)，Value：该域的权威 DNS 服务器的主机名；**Type=CNAME**，Name：主机别名，Value：规范主机名；**Type=MX**，Name：域(e.g. foo.com)，Value：该域的邮件服务器名字。 一台权威 DNS 服务器包含其所用于的主机名的 A 记录；如果不是权威 DNS，则包含一条 NS 记录，和记录了在 NS 记录的 Value 字段中的 DNS 服务器 IP 地址的 A 记录。 DNS 只有**查询和回答两种报文**，并且有着相同的格式：前 12 字节是**首部区域**，包含 16bit 标识符来标识该查询，16bit 标志来标记报文是查询（0）还是回答（1）、回答者是权威（1）还是非权威（0）、客户希望递归查询（1）还是不希望、递归可用（1）或不可用，4 个 16bit 的有关数量的字段（依次是问题数，回答 RR 数，权威 RR 数，附加 RR 数）；**问题区域**包括正在进行的查询的名字字段和类型字段；**回答区域**包含了对最初请求的名字的 RR（可以有多条）；**权威区域**包含了其他权威服务器的记录；**附加区域**包含了其他有帮助的记录（如 MX 的回答包含一邮件服务器的规范主机名，那么附加区域包含一个该规范主机的 A 记录）。 **DNS 报文封装时**如果长度小于 512B，使用 UDP，否则使用 TCP；如果不知道响应报文的长度，先使用 UDP，一旦响应报文超过 512B，则截断该报文，置 DNS 首部 TC 标志为 1，客户机 DNS 解析器打开 TCP 连接并重复请求。 **往 DNS 数据库中插入资源记录**，则向 DNS **注册登记机构**注册域名，提供权威 DNS 服务器（主域名服务器和辅助域名服务器）的名字和 IP，对每个权威域名服务器，注册机构往 TLD 服务器插入 NS 记录和对应 NS 中 Value 名字的 A 记录。 **建立权威 DNS 服务器**，特别是建立 www.networkuptopia.com 的 Type A 记录和 networkuptopia.com 的 MX 记录以及相应邮件服务器的 A 记录。

P2P 应用：

P2P 文件分发：**分发时间**是所有 N 个对等方得到该文件副本所需要的时间。 **客户-服务器体系结构的分发时间：**D_cs = max { NF/u_s, F/d_min } **P2P 体系结构的分发时间：**D_P2P = max { F/u_s , F/d_min , NF/(u_s + Sigma(i=1 to N)u_i) } P2P 体系结构的应用程序是**能够自扩展的，其直接成因是：**对等方除了是比特消费者外也是重新分发者。 **BitTorrent：**用于文件分发的流行 P2P 协议。 **Torrent（洪流）：**参与一个特定文件分发的对等方集合。每个洪流有一个**追踪器（tracker）**跟踪洪流中的对等方。 **当对等方加入洪流时**，向跟踪器注册（并周期的通知它还在洪流中），获得一个随机的对等方子集，试图与该子集建立 TCP 连接（成功创立的称为邻近对等方）。 文件被划分为长 256KB 的**块**，周期性地，对等方询问每个临近对等方拥有的数据块集合，并向邻居请求它缺少的数据块，请求数据块的顺序遵循**最稀罕优先**的技术（最稀缺的块得到更为迅速的重新分发，目标是大致均衡每个块在洪流中的副本数量）。 为了决定对等方响应谁的请求，采用**对换算法（tit-for-tat，一报还一报）：**Alice 选择当前向其发送数据最快的 4 个邻居（称为**疏通**），响应他们的数据块请求；每隔 10 秒，重新评估向其提供数据最快的 4 个邻居；每隔 30 秒，随机选择另一个对等方（如 Bob）响应其请求（Alice 可能成为向 Bob 上载最快的 4 个邻居之一，Bob 也可能成为向 Alice 上载最快的 4 个邻居之一），其效果是对等方能趋于找到彼此协调的速率下载。

第 3 章：运输层

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**，不同主机上的应用进程仿佛直接连在一起。 运输层协议是运行在端系统中而不是在路由器里实现的。 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，称为运输层**报文段（segment）**。

运输层于网络层的关系：

网络层提供了主机之间的逻辑通信。传输层**依赖并增强**网络层服务，运输协议能够提供的服务常常**受制于**底层网络协议的服务模型（网络层没有时延和带宽保证，则运输层也没有），但运输层**也能**提供某些服务（可靠的数据传输，加密）。

因特网运输层概述：

因特网**网络层**的**网际协议 IP** 提供**尽力而为的交付服务**，但并不做任何确保（不确保交付、按需交付和数据完整性），这是**不可靠服务**。 **UDP 和 TCP 最基本的责任是**，将两个端系统间 IP 的交付扩展为运行在端系统上的两个进程之间的交付服务，被称为**运输层的多路复用**和**多路分解**。UDP 和 TCP 还可以在报文段首部加上差错检查字段提供完整性检查。进程到进程的数据交付和差错检查是两种最低限度的运输层服务，**也是 UDP 仅能提供的服务，UDP 是不可靠的**。

多路复用与多路分解：

多路复用和多路分解是所有计算机网络都需要的。 一个进程有一个或多个**套接字**，它相当于从网络向进程传递数据和从进程向网络传递数据的门户。 每个运输层报文段有几个字段，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**。 在源主机上从不同的套接字中收集数据块，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**。 运输层多路复用要求：套接字有唯一标识符；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。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 16bit 的数，大小 0~65535，0~1023 是**周知端口号**，是受限制的，保留给诸如 HTTP 的周知应用层协议。 **UDP 的多路分解：**在主机上的每个套接字分配一个端口号，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。UDP 套接字标识为<IP 地址，端口号>二元组，这意味着如果两个 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的目的进程。源端口号用作“返回地址”的一部分。 通常应用程序的客户端让运输层自动地分配端口号，而服务器端则分配一个特定的端口号。 TCP 套接字是由一个四元组<源 IP 地址，源端口号，目的 IP 地址，目的端口号>标识的，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链接的请求。

无连接运输：UDP

使用 UDP 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，UDP 是**无连接的**。 **有许多应用更适合 UDP，主要是因为：**关于何时，发送什么数据的应用控制更为精细（绕过 TCP 拥塞控制，并在应用层实现 UDP 不提供的某些服务）；无需连接建立（不会引入连接建立的时延，这可能是 DNS 运行在 UDP 上的原因）；无连接状态（维护连接需要额外的接收发送缓存，拥塞控制参数、需要与确认号，运行在 UDP 上可以节省资源支持更多用户）；分组首部开销小。 当分组丢包率低并且出于安全，某些机构阻塞 UDP 流量。UDP 没有拥塞控制，这使得在 UDP 上运行多媒体是有争议的。使用 UDP 的应用程序可以自身建立可靠性机制来达到可靠传输。

UDP 报文段结构：**UDP 首部 8 字节：源端口号，目的端口号，长度**（指示 UDP 报文段的字节数，包括首部和数据），**检验和。UDP 数据字段。**

UDP 检验和：**检验和计算：**对 UDP 报文段（以及 IP 首部的 32 位源和目的 IP，8 位前导 0 和 8 位 protocol 字段，16 位 UDP 总长度字段）的所有 16 比特字的和进行反码运算（第一次运算前检验和字段填入 0），求和时遇到的任何一处都被回卷，得到的结果被放在 UDP 报文段中的检验和字段。**接收方**将全部 16bit 字（包括检验和）加在一起，如果没有差错则结

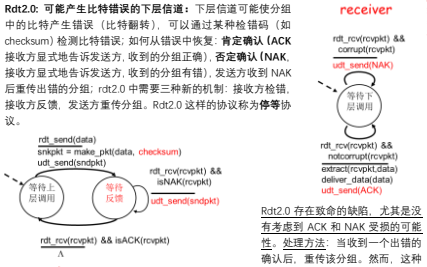
果为全 **1**，否则结果有错误。**但 UDP 对差错恢复无能为力**。

可靠数据传输原理：

实现"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收到损坏或丢失，而且所有数据都是按照其发送顺序进行交付"的服务抽象是可**靠传输协议（rdt）**的责任。可靠数据传输协议的下层协议也许是不可靠的，这决定了 rdt 的复杂性。

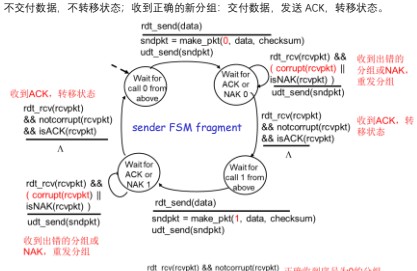
构造可靠数据传输协议：Rdt1.0: 可靠信道上的可靠传输：下层信道是完全可靠的（理想情况），没有比特错误，没有分组丢失，发送能力 = 接收能力；发送方 FSM：从上层接收数据，封装成分组送入下层信道；接收方 FSM：从下层信道接收分组，取出数据交给上层。

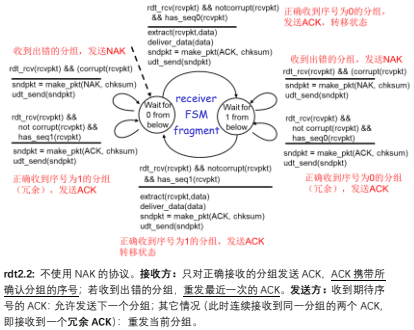
Rdt2.0: 可能产生比特错误的下层信道：下层信道可能使分组中的比特产生错误（比特翻转），可以通过某种检错码（如 checksum）检测比特错误；如何从错误中恢复：**肯定确认 (ACK**：接收方显式地告诉发送方，收到的分组正确），**否定确认 (NAK**：接收方显式地告诉发送方，收到的分组有错），发送方收到 NAK 后重传出错的分组；rdt2.0 中需要三种新的机制：接收方检错，接收方反馈，发送方重传分组。Rdt2.0 这样的协议称为**停等**协议。



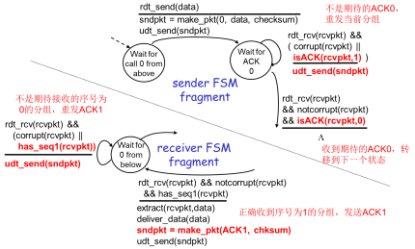
Rdt2.0 存在致命的缺陷，尤其是没有考虑到 ACK 和 NAK 受损的可能性。处理方法：当收到一个出错的确认后，重传该分组。然而，这种方法在信道中引入了**冗余分组**。解决这个新问题的方法是在数据分组中添加一个新字段，让发送方对其数据分组编号，将发送分组的**序号**放在该字段，接收方检测序号以确定是否是重传。对于停等协议，1bit 序号就够了。

Rdt 2.1：发送方：构造分组：加入序号；等待反馈：收到 NAK 或者出错的反馈，重发分组；收到 ACK，转移状态。**接收方：**收到出错的分组：发送 NAK；收到冗余的分组：发送 ACK，不交付数据，不转移状态；收到正确的新分组：交付数据，发送 ACK，转移状态。

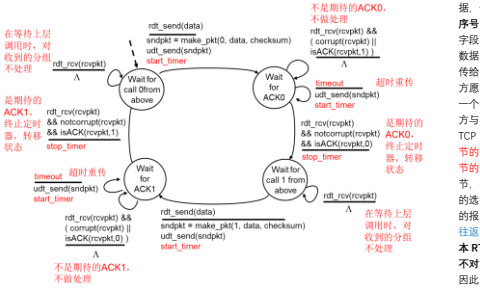




rdt2.2: 不使用 NAK 的协议。**接收方：**只对正确接收的分组发送 ACK，ACK 携带所确认分组的序号；若收到出错的分组，重发最近一次的 ACK。**发送方：**收到期待序号的 ACK：允许发送下一个分组；其它情况（此时连续接收到同一分组的两个 ACK，即接收到一个**冗余 ACK**）：重发当前分组。



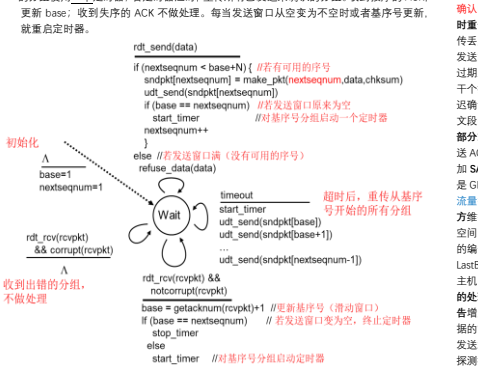
rdt3.0: 可能产生比特错误和丢包的下层信道（有时称为**比特交替协议**）：需要两项新技术：检测丢包（包括数据及 ACK），从丢包中恢复。**方法：**检测丢包：若发送方在"合理的"时间内未收到 ACK，认为丢包（需要定时器）。从丢包中恢复：发送方重发当前分组。问题：ACK 丢失或超时设置过短导致的重发，会在接收端产生冗余分组。**发送方：**发送分组后启动一个定时器；收到期待序号的 ACK，终止定时器，转移状态；收到非期待序号的 ACK，不做处理（为什么？分组 1 的 ACK 在定时器超时之后到达，那么超时之后的重发包（**冗余数据分组**）会被前一个 ACK 提早确认，状态转移；这时收到了重发分组 1 的 ACK，没必要处理）；定时器超时后，重发分组。**接收方：**数据分组丢失：接收方感知不到丢包，该事件对接收方 FSM 无影响；ACK 丢失，过早超时：（发送方重发）接收方收到重复的数据分组，接收方利用序号检测重复分组，重发前一次 ACK。**接收方 FSM 同 Rdt2.2。** Rdt3.0 是**停等协议**，**发送方（信道）利用率：**发送方实际上忙于将发送比特送进信道的那部分时间与



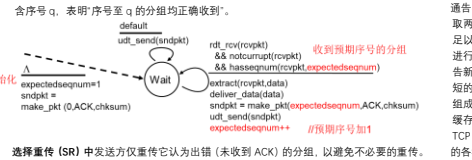
发送时间之比。停等协议的 **U_sender** = L/R / (RTT + L/R)。 **解决：**允许发送方发送多个分组而无需等待确认。这种技术被称为**流水线**。 流水线技术为可靠数据传输协议带来如下**影响：**必须增加序号范围；协议的发送方和接收方两端也许必须缓存多个分组；所需序号的范围和对缓冲的要求取决于数据传输协议如何处理丢失、损坏及延时过大的分组。 解决流水线的差错恢复有两种基本办法：**回退 N 步（GBN）**和**选择重传（SR）**。

将**基序号（base）**定义为最早的未确认分组的序号，将**下一个序号（nextseqnum）**定义为下一个待发的分组序号。[0 , base – 1] 已发送并被确认，[base , nextseqnum – 1] 已发送未确认，[nextseqnum , base + N – 1] 能用于缓存将要被发送的分组，大于 base + N 不可用。 N 常被称为**窗口长度**，GBN 协议也称为**滑动窗口协议**。 **流量控制**是对发送方施加限制 N 的原因之一。

回退 N 步协议的发送方最多允许 N 个已发送未确认的分组；只对于最早的已发送未确认的分组使用一个定时器；若定时器超时，重传所有已发送未确认的分组。收到按序的 ACK，更新 base；收到失序的 ACK 不做处理。每当发送窗口从空变为不空时或者基序号更新，就重启定时器。



回退 N 步的**接收方：**收到一个按需分组时发送一个 ACK 分组，ACK 携带序号，接收方丢弃所有失序分组（优点是接收缓存简单，缺点是浪费带宽且之后的重传也可能出错导致更多重传），并在丢弃后发送一个最近按序接收的分组发送 ACK；使用累积确认：若 ACK 包含序号 q，表明'序号至 q 的分组均正确收到'。



选择重传（SR）中发送方仅重传它认为出错（未收到 ACK）的分组，以避免不必要的重传。这种介别的接需的重传要求接收方**逐个地**确认正确接收的分组。 **SR 发送方**的已发送未确认的每个分组都要一个（逻辑）定时器；发送方收到 ACK 时，ACK 在窗口内则将分组标记为已发送已接收，如果分组序号恰为 send_base，则 send_base 滑动到最小序号的未确认分组处。 **SR 接收方**确认一个正确接收的分组而不论其是否按序。失序的分组将被缓存直到所有丢失分组皆被收到为止，这时将他们一起交给上层。序号在 [rcv_base , rcv_base + N – 1] 被接收，发送一个选择 ACK；序号在 [rcv_base – N , rcv_base – 1] 内的分组被接收**也必须产生一个 ACK**（如果正确收到分组然而 ACK 丢失，必须确认发送方重传的分组，否则发送方窗口无法前进）；其他情况忽略分组。对于哪些分组被接收了，哪些没有，发送和接受方并不一定能看到相同的结果，SR 协议内的发送方和接收方窗口并不总是一致的。 对于 SR 协议而言，窗口长度必须小于等于序号空间大小的一半，否则接收方能将重发的分组当作新分组（**考虑以下情形：**发送方发送了一个窗口 [0 , N – 1] 的分组；接收方全部接收正确，发送了 ACK，并滑动接收窗口至 [N , 2N – 1] ，但 N 个 ACK 全部丢失，发送端超时后逐个重发这 N 个分组。为使接收端不会将重发的分组当成新的分组，窗口 [0 , N – 1]和窗口 [N , 2N – 1]不能有重叠。） **SR 特点：**出错后重传代价小，接受端需要大量缓存。

面向连接的运输：TCP

TCP 连接：TCP 被认为是**面向连接的**，因为一个进程开始向另一个进程发送数据之前，二者必须先发送某些预备报文段，以建立确保数据传输的参数和状态（套接字、缓存、变量）。（其连接状态完全保留在端系统中，不同于电路或虚电路）。 TCP 提供**全双工服务：**数据可以同时从 A 流到 B 并从 B 流到 A。 TCP 连接是**点对点的**，是在单个发送方与单个接收方之间的。发起连接的称为**客户进程**，另一个为**服务器进程**。 TCP 将客户进程通过套接字的数据存到该链接的**发送缓存**里，在三次握手初期设立。 TCP 可从缓存中取出并放入报文段的数据量受限于**最大报文段长度 (MSS)**（这是指报文段里应用层数据的最大长度，而非包括 TCP 首部的 TCP 报文段的最大长度），MSS 通常根据最初确定的由本地发送主机发送的最大链路层帧长度（最大传输单元 MTU）来设置。 **TCP 连接的组成：**发送方和接受方所在的主机上的缓存、变量和进程连接的套接字（路由器交换机和中继器没有为连接分配任何缓存或变量）。

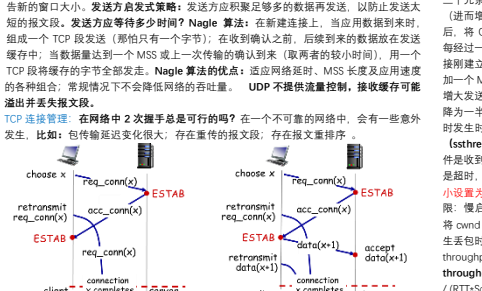
TCP 报文段结构：TCP 报文段由首部字段和一个数据字段组成。数据字段包含一块应用数据，长度受限于 MSS。 TCP 的**首部**一般 20 字节，分别是：16bit 源和目的端口号；32bit **序号**；32bit **确认号**；4bit 首部长度（指示以 32bit 为单位的首部长度，间接指示 TCP 选项字段的长度）；6bit 保留未用；6bit 标志字段 **URG**（报文段存在由发送端的上层置为紧急的数据，实际未用）、**ACK**（指示确认字段的值是 0 否 1 有效）、**PSH**（接收方应立即将数据传给上层，实际未用）、**RST**、**SYN**、**FIN**；16bit **接收窗口字段**（用于流量控制，指示接收方愿意接受的字节数量）；16bit 因特网检验和；16bit 紧急数据指针（指向紧急数据的最后一个字节，紧急数据放置在数据载荷前端，实际未用）；可选与变长的**选项字段**，用于发送方与接收方协商 MSS，或在高速网络环境下用作窗口调节因子，或提供选择确认 SACK。 TCP 把数据看成一个无结构的、有序的字节流，故**一个报文段的序号**是该报文段数据首字节的字节流编号。 主机 A 填充进报文段的**确认号**是主机 A 期望从主机 B 收到的下一个字节的序号。 TCP **流水式**发送报文段，提供**累计确认**，**实践中的 TCP 接收方**保留接收的字节，并等待缺少的字节填补间隔，RFC 标准没有规定。 一条 TCP 连接的双方均可以随机的选择初始序号。 对客户到服务器的数据的确认被装在在一个承载服务器到客户的数据的报文段中，这种确认被认为是被**捎带**在服务器到客户的数据报文段中的。

往返时间的估计与超时：超时间隔必须大于 RTT，否则造成不必要的重传。 报文段的**样本 RTT (SampleRTT)** 是从某报文段被发出到对该报文段的确认被收到之间的时间量。**TCP 不对重传的报文段测量 RTT** (TCP 是对接收到的数据而不是对携带数据的报文段进行确认，因此 TCP 的确认是有二义性的，对重传报文段的 RTT 估计不准确。解决方法是，忽略有二义性的确认，只对一次发送成功的报文段测量 SampleRTT ，并据此更新 EstimtedRTT。当 TCP 重传一个段时，停止测量 SampleRTT)，在任意时刻也仅为一个已发送但目前尚未被确认的报文段估计 SampleRTT，不是每个都估计。 **EstimatedRTT** = (1 – α)•EstimatedRTT + α•SampleRTT，α参考值 0.125，是 SampleRTT 的**指数加权移动平均（EWMA）**。 **RTT 偏差 DevRTT** 用于估算 SampleRTT 一般会偏离 EstimatedRTT 的程度：DevRTT = (1 – β)•DevRTT + β•| SampleRTT – EstimatedRTT |, β推荐值 0.25。**设置超时值 TimeoutInterval** = EstimatedRTT + 4•DevRTT, 初始 TimeoutInterval 推荐 1s, **当出现超时后，TimeoutInterval 加倍**（而非上述公式的估计值）；一旦报文段 ACK **被收到**并更新 EstimatedRTT 后，或者收到上层的应用数据之后，TimeoutInterval 就又用上述公式计算了。这种 TimeoutInterval 修改提供了一个形式受限的**拥塞控制**。

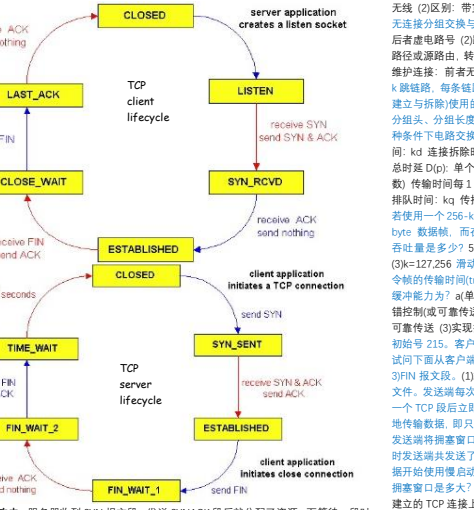
可靠数据传输：TCP 定时器管理过程**仅使用单一的重传定时器**，定时第一个发送未确认的报文段，像 SR 那样每个维护一个定时器需要相当大的开销。 定时器超时，仅重传最早未确认的报文段。 当收到 ACK 时，只要**确认序号是大于基序号的，就推进发送窗口。 超时重传间题之一是：**超时周期可能相对较长。报文段丢失时这种长周期迫使发送方延迟重传丢失的分组，因而增加了端到端时延。 **冗余 ACK** 就是再此确认某个报文段的 ACK，而发送方先前已经收到对它的确认。一旦收到三个冗余 ACK，TCP 执行**快速重传**，在定时器过期之前重传报文段。 为减小确信息，TCP 允许接收端**推迟确认**，接收端可以在收到若干个报文段后，发送一个累积确认的报文段；**缺点是**若延迟太大会导致不必要的重传且推迟确认造成 RTT 估计不准确。 TCP 规定推迟确认的时间**最多为 500ms**，且当收到一个报文段时，若接收方的前一个按序报文段等待传输 ACK 则立刻**同时确认两个报文段。 当能部分或完全填充接收数据间隔的报文段到达**，倘若该接收段**起始于间隔的低端**，则立即发送 ACK。 当**失序**报文段或**重复**报文段到达，则**立即发送冗余 ACK**。 TCP 的修改版本添加 **SACK 选项头**，给出收到的非连续数据块的上下边界。 TCP 的差错恢复机制可以看成是 GBN 和 SR 的混合体；TCP 在减小定时器开销和重传开销方面要优于 GBN 和 SR。

流量控制：TCP 提供**流量控制服务**以消除发送方使接收方缓存溢出的可能性。 TCP **发送方维护一个接收窗口**来提供流量控制，这个窗口指示发送方该接收方还有多少空间的缓存空间。 定义变量 **LastByteRead**（主机 B 的应用进程从缓存读出的数据流的最后一个字节的编号）和 **LastByteRcvd**（主机 B 接收缓存放入的最后一个字节的编号），LastByteRcvd – LastByteRead <= RcvBuffer, **接收窗口 rwnd** = RcvBuffer – [LastByteRcvd - LastByteRead]，主机 A 满足 **LaskByteSent – LastByteAcked** <= rwnd。 **该方法的一个小问题：对零窗口的处理：**当接收窗口为 0 时，发送方必须停止发送；当接收窗口变为非 0 时，接收方应**通告**增大的接收窗口。**TCP 协议规定：**发送方收到'零窗口通告'后，可以发送只有一个字节数据的'零窗口探测'报文段，以触发一个包含接收窗口的响应报文段。 **零窗口探测的实现：**发送端收到零窗口通告时，启动一个坚持定时器；定时器超时后，发送端发送一个零窗口探测报文段（序号为上一个段中最后一个字节的序号）；接收端在响应的报文段中通告当前接收窗口的大小；若发送端仍收到零窗口通告，重新启动坚持定时器。 **糊涂窗口综合症：**当发送速度很快、消费速度很慢时，零窗口探测的简单实现带来以下问题：接收方不断发送微小窗口通告；发送方不断发送很小的数据分组；大量带宽被浪费。**接收方启发式策略：**通告零窗口之后，仅当窗口大小**显著增加**（窗口大小达到缓存空间的一半或者一个 MSS，取两者的较小值）之后才发送更新的窗口通告。**TCP 执行该策略的做法：**当窗口大小不满足以上策略时，推迟发送确认（但最多推迟 500ms，且至少每隔一个报文段使用正常方式进行确认），寄希望于推迟间隔内有更多数据被消费；仅当窗口大小满足以上策略时，再通告新的窗口大小。**发送方启发式策略：**发送方应积聚足够多的数据再发送，以防止发送太短的报文段。**发送方应等待多少时间？ Nagle 算法：**在新建连接上，当应用数据到来时，组成一个 TCP 段发送（那怕只有一个字节）；在收到确认之前，后续到来的数据放在发送缓存中；当数据量达到一个 MSS 或上一次传输的确认到来（取两者的较小时间），用一个 TCP 段将缓存的字节全部发走。**Nagle 算法的优点：**适应网络延时、MSS 长度及应用速度的各种组合；常规情况下不会降低网络的吞吐量。 **UDP 不提供流量控制，接收缓存可能溢出并丢失报文段。**

TCP 连接管理：**在网络中 2 次握手总是可行的吗？**在一个不可靠的网络中，会有一些意外发生，**比如：**包传输延迟变化很大；存在重传的报文段；存在报文重排序 。



TCP 连接方式：第一步：客户端 TCP 向服务器发送一 SYN 置 1、不包含应用数据、随机选择一个初始序号并将确号放置在序号字段中的 **SYN 报文段**。**第二步：**服务器为 TCP 连接分配 TCP 缓存和变量，向客户发送一个允许连接的 **SYNACK 报文段**（SYN 和 ACK 置 1，确认号字段为 client_isn + 1，服务器选择自己的 server_isn 放入序号字段）。**第三步：**客户为自己的分配缓存和变量，向服务器发送另一报文段（将 server_isn + 1 放入确认字段，ACK = 1，SYN = 0，**报文段中可以负载数据**）。 此后每个报文段中，SYN 都为 0。 选择的起始序号必须避免新旧连接上的序号产生重叠：基于时钟，取较小的ΔT 递增时钟计数器（确保发送序号的增长速度不会超过起始序号的增长速度），取计数器低 32 位（确保序号回绕的时间远大于分组在网络中的最长寿命）。 **TCP 关闭连接：**客户应用进程发出一个关闭连接命令，这会引起客户 TCP 向服务器进程发送一个特殊的 TCP 报文段（FIN 置 1，seq = x），服务器收到后向发送方回送一个确认报文段（ACK=1，ACKseq=x+1），然后服务器发送它自己的特殊报文段 (FIN=1, seq=y)，最后客户对这个服务器的中止报文段进行确认 (ACK=1, ACKseq=y+1)。此时连接的所有资源都被释放。



SYN 洪泛攻击：服务器收到 SYN 报文段、发送 SYNACK 段后就分配了资源，而等待一段时间（30s~120s）才丢弃未完成的连接；攻击者伪造大量 SYN 段而不发送 ACK 段，耗尽服务器资源。防御系统 **SYN cookie**。 当收到的 TCP 报文段与本主机上已有的套接字不匹配时，主机发送一个**重置报文段**，**RST=1。**当收到的 UDP 报文段与套接字不匹配，主机发送特殊的 ICMP 数据报。 **端口扫描：**扫描程序利用与目标机器建立 TCP 连接过程中获得的响应消息来收集信息；**在典型的 TCP 端口扫描过程中，发送端向目标端口发送 SYN 报文段**，若收到 SYNACK 段，表明该目标端口上有服务在运行；若收到 RST 段，表明该目标端口上没有服务在运行；若什么也没有收到，表明防程上有防火墙（有些防火墙会丢弃来自外网的 SYN 报文段）。**FIN 扫描过程中：**发送端向目标端口发送 FIN 报文段；若收到 ACK=1、RST=1 的 TCP 段，表明目标端口上没有服务在监听；若没有响应，表明有服务在监听（RFC 973 的规定）。**缺点：**有些系统的实现不符合 RFC 973 规定，如在 Microsoft 的 TCP 实现中。总是返回 ACK=1、RST=1 的 TCP 段。

拥塞控制原理：

拥塞原因与代价：**拥塞造成：**丢包（缓存溢出）、分组延迟增大（链路接近满载）；**大量网络资源用于：**重传丢失的分组；（不必要地）重传延迟过大的分组；转发最终被丢弃的分组。**结果：**网络负载很重、但网络吞吐量很低。

拥塞控制方法：**网络辅助的拥塞控制：**路由器向端系统提供反馈（拥塞指示比特，发送速率指示），ATM 采用此类方法。 **端到端拥塞控制：**网络层不向端系统提供反馈，端系统通过观察丢包和延迟推断拥塞的发生，**TCP 采用此类方法**（因为 IP 层不提供显式反馈）。

TCP 拥塞控制：

运行在**发送方**的 TCP 拥塞控制跟踪变量**拥塞窗口（congestion window，cwnd）**，控制发送方速率，特别是 LastByteSent – LastByteAcked <= min { cwnd , rwnd }。只考虑 cwnd 时，粗略的讲，在每个 RTT 的起始点，限制条件允许发送方发送 cwnd 个字节的数据，在该 RTT 结束时发送方接收对数据的确认报文。发送方的**速率大概是 CongWin/RTT Bytes/sec**。**感知拥塞：**丢包或分组延迟过大对于发送端而言都是丢包，**丢包事件**（或者超时或者收到三个冗余 ACK）；TCP 使用确认的到达作为一切正常的指示，使用确认来增加窗口的长度（进而增加传输速率）。 **加性增，乘性减（AIMD，拥塞避免模式）：乘性减：**检测到丢包后，将 CongWin 的大小减半（迅速减小），但不能小于一个 MSS。**加性增：**若没有丢包，每经过一个 RTT 将 CongWin 增大一个 MSS，直到检测到丢包（缓慢增大）。 **慢启动：**连接刚建立时 CongWin = 1 MSS（发送速度≈ MSS/RTT），每收到一个 ACK 段，CongWin 增加一个 MSS（这一过程每过一个 RTT 就把 cwnd 翻一倍）（以一个很低的速率开始，按指数增大发送速率）直到出现丢包事件（此时终止慢启动过程）。 **收到 3 个冗余 ACK：**CongWin 降为一半，采用 AIMD 调节；**超时：**CongWin = 1MSS；使用慢启动增大 CongWin，至超时发生时 CongWin 的一半，使用 AIMD 调节。 **实现：**发送方维护一个变量 **Threshold（ssthresh）；发生丢包事件时，Threshold 设置为当前 CongWin 的一半**（整除 2），若事件是收到 3 次冗余 ACK，CongWin 从原 CongWin 的一半+3 开始，执行 AIMD；若发生的是超时，CongWin 从 1MSS 开始，执行慢启动。**慢启动翻倍后值大于 ssthresh 则将窗口大小设置为等于 ssthresh。Threshold 是从慢启动转为拥塞避免的分水岭：**CongWin 低于门限：慢启动；CongWin 高于门限：拥塞避免（AIMD）。 **TCP Tahoe：**只要是丢包事件就将 cwnd 变成 1MSS 并减半 ssthresh **TCP 的吞吐量的宏观描述**（忽略慢启动）：令 W=发生丢包时的 CongWin，此时 throughput = W/RTT；发生丢包后调整 CongWin=W/2，此时 throughput=W/2RTT；假设在 TCP 连接的生命期内，RTT 和 W 几乎不变，则：**Average throughout=0.75 W/RTT。 经高带宽路径的 TCP：**吞吐量与丢包率 L 的关系 = 1.22•MSS / (RTT•Sqrt[L]）；针对高速网络需要新的 TCP 拥塞控制算法。

公平性：公平性的目标：如果 K 条 TCP 连接共享某条带宽为 R 的瓶颈链路，每条连接具有平均速度 R/K。 **TCP 是公平的。** 若相互竞争的 TCP 连接具有不同的参数（RTT、MSS 等），**不能保证公平性**；若应用（如 web）可以建立多条**并行** TCP 连接，不能保证带宽在应用之间公平分配（比如，一条速率为 R 的链路上有 9 条连接；若新应用建立一条 TCP 连接，获得速率 R/10，若新应用建立 11 条 TCP，可以获得速率 R/2） 开发一种因特网中的拥塞控制机制，用于**阻止 UDP 流量不断压制（TCP）直至中断因特网吞吐量。**

小测题和易错点：

易错点：kbps = 1000bps。 平均**传输**次数=平均**重传**次数+1

习题课：若偶尔发送数据，只用 NAK 的协议会导致丢失的包在很久之后的下一个包被接收时在能发现。 丢失的包不能及时重传时延大大提升。若要发送大量数据且丢包率低，则上述情况不会发生。由于丢包率低，所需发送的 NAK 远少于 ACK，使用仅 NAK 的方法可以降低网络负担，并且发送方可以不用等待 ACK 持续发送降低了传输时延。

在网络传输中乱序是很常见的情况，**若收到第一个冗余 ACK 就重传，对每个乱序都要重传一次冗余的分组，极大降低了吞吐量。**收到 3 个冗余 ACK 后重传可以很大程度减轻乱序重传，同时兼顾快速重传。

小测答案：

常用的通信传输介质有哪些？它们之间的主要区别？(1)有线：双绞线、同轴电缆、光纤；无线 (2)区别：带宽、误码率、传输距离、价格、频谱及复用方式、是否支持移动通信等。

无连接分组交换与面向连接(虚电路)分组交换的区别？(1)分组格式:前者完全源、目的地址；后者虚电路号 (2)路由表：前者面向整个网络拓扑，转发时顺序查找路由表；后者面向特定路径或源路由，转发基于表引查找路由表。(3)可靠性、顺序性：前者无；后者有 (4)建立、维护连接：前者无；后者有 假定要传送的报文共有 x(单位 bit)，从源节点到目的节点共有 k 跳链路，每条链路的传播时延为 d(单位 s)，链路带宽为 b(单位 bit/s)；电路交换(包括连接建立与拆除)使用的控制帧(或信令)长度、在各节点的排队时延忽略不计；分组交换使用的分组头、分组长度分别为 h、p(单位 bit)，分组在各节点的排队时延 q(单位 s)。试分析在何种条件下电路交换的总时延要小于分组交换的总时延？电路交换总时延 D(c)：连接建立时间：kd 连接拆除时间：kd 数据传输时间：x/b 数据传播时间：kd D(c)=3kd+x/b 分组交换总时延 D(p)：单个分组传输时间：(p+h)/b 第 1 跳传输时间：(x/p)((p+h)/b) (x/p 为分组个数) 传输时间每 1 跳增加 1 个分组的传输时间，总的传输时间为 x/p•(p+h)/b+(k-1)•(p+h)/b 排队时间：kq 传播时间：kd D(p)=x/p•(p+h)/b+(k-1)•(p+h)/b+kd+kq 若 D(c)<D(p)，则 … 若使用一个 256-kbps 的无差错卫星信道(往返传播时延为 512-msec)一个方向上发送 512-byte 数据帧，而在另一个方向上返回很短的确认帧。则对于窗口大小为 1, 15, 127 的最大吞吐量是多少？ 512•8/256k=16ms (1)k=1,16/(16+512)•256=7.75 (2)k=15,7.75•15=116.36 (3)k=127,256 滑动窗口协议中,退后 N 帧与选择性重传利用链路缓冲能力连续发送多个帧,令帧的传输时间(transmission time)=1(归一化)、传播时间(propagation time)=a，则链路的缓冲能力为？ a(单向)或 2a(双向) TCP 协议中 ACK 的作用。 (1)建立连接、拆除连接 (2)差错控制(或可靠传送)(3)流量控制 (4)拥塞控制 TCP 连接的目标 (1)实现进程间通信 (2)实现可靠传送 (3)实现按序传送 (4)进行流量控制 (5)进行拥塞控制 在 TCP 连接中，客户端的初始号 215，客户打开连接，只发送一个携带有 100 字节数据的报文段，然后关闭连接。试问下面从客户端发送的各个报文段的序号分别是多少？ (1)SYN 报文段；(2)数据报文段；3)FIN 报文段。(1)215；(2)216；(3)316 在一条新建的 TCP 连接上发送一个长度为 32KB 的文件。发送端每次都发送一个最大长度的段（MSS），MSS 的长度为 1KB，接收端正确收到一个 TCP 段后立即给予确认。发送端的初始拥塞窗口门限设为 16KB。假设发送端尽可能快地传输数据，即只要发送窗口允许，发送端就发送一个 MSS。(1) 已知发生第一次超时后，发送端将拥塞窗口门限调整为 4KB。请问发生超时的时候，发送端的拥塞窗口是多大？ 此时发送端共发送了多少数据？其中有多少数据被成功确认了？ (2) 发送端从未被确认的数据开始使用慢启动进行重传。假设此后未再发生超时，当文件全部发送完毕时，发送端的拥塞窗口是多大？ 答：(1) 第一次超时发生时，发送端拥塞窗口大小 = 4KB•2 = 8KB 在新建立的 TCP 连接上，发送端采用慢启动开始发送。因此当第一次超时发生时，发送端已发送的数据量 = 1KB + 2KB + 4KB + 8KB = 15KB。此时，除最后一批 8 个 TCP 段未获确认外，之前发送的 TCP 段都被确认，因此成功确认的数据量为 7KB。(2) 发送端采用慢启动重新开始发送，在拥塞窗口达到 4KB 时发送数据量=1KB+2KB+ 4KB=7KB。然后进入拥塞避免阶段：在收到全部 4 个 MSS 的确认后，拥塞窗口增至 5KB，相应地发送端发送了 5KB 数据；收到全部 5 个 MSS 的确认后，拥塞窗口增至 6KB；收到全部 6 个 MSS 的确认后，拥塞窗口增至 7KB；此时刚好发完。因此，文件发送结束时，发送端的拥塞窗口大小为 7KB。