*计算机网络提供的最重要功能：连通性、共享*

**第一章 概述**

## 1. 1 什么是因特网

因特网的两种描述：

• 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的网络

• 为分布式应用提供通信服务的基础设施

**[具体构成]**

• 终端设备：称为主机或端系统，运行网络应用程序

• 通信链路：同轴电缆、光纤、 铜线、无线电频谱等

• 交换设备：转发分组，包括路由器（网络核心）和链路层交换机（接入网）。端系统间通过通信链路和分组交换机连接在一起，通过因特网服务提供商(ISP)接入因特网。ISP 是多个分组交换机与多段通信链路组成的网络

**[服务描述]**

• 网络协议：规定了终端、交换设备之间通信需要遵循的规则（报文格式、次序、收发行为等）。因特网最核心的两个协议为 TCP、IP

• 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)，规定了运行在某端系统上的软件应如何请求因特网基础设施，从而向运行在另一个端系统上的目的地软件交付数据。

## 1.2 网络边缘

**[细化网络结构]**

• 端系统：与因特网相连的计算机和其他设备（运行应用程序）

• 接入网：将终端连接到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

• 网络核心：路由器和通信链路组成的网络

**[接入网]**

• 家庭住宅接入：数字用户线 DSL （传统电话信号，上行 2.5Mbps 下行 24Mbps）、电缆（有线电视提供，上行 3Mbps 下行 30Mbps）、光纤、拨号和卫星上网

• 企业（和家庭）接入：以太网（10Mbps～10Gbps）、WIFI（11Mbps～450Mbps）

• 广域无线接入：3G（2Mbps）、4G（上行 20Mbps 下行 100Mbps）

**[物理媒体]**

设备之间通过物理媒体相连，两端各需要一对收发设备。分为：导引型（双绞铜线 - 电话线网线、同轴电缆 - 有线电视、光纤）和非导引型（电磁波、无线局域网、数字卫星频道）

• 以太网用户使用双绞铜线与以太网交换机连接

## 1.3 网络核心：由分组交换机与链路构成的网状网络

任务：在边缘路由器之间高效、准确传输数据包。功能：选路、转发

**[分组交换]**

主机将应用报文划分成分组，交换机仅在接收到完整分组后， 才可以开始转发。为此交换机在每条链路上都设置有输出缓存，当分组到达速率大于链路输出速率时，会在缓存中排队，从而引入排队延迟。缓存满了会丢包。大量分组集中到达时，排队与丢包较为严重。

• 不考虑信号传播时间，P 个分组经过 N 条链路的总耗时是（P+N-1）L/R。可以看到当 P 远大于 N 时，存储转发不会引入过多延迟

• 不采用边收边发的原因：便于检错和重传

• 优点：更好的带宽共享，简单有效，适合突发数据，不需建立电路连接；支持更多的端系统、能快速传输端系统产生的大量数据，按需分配（不预留资源)

• 缺点：可能产生严重拥塞（延迟、丢包)（不适合实时服务）

**[电路交换]：例如传统的电话网络与接线员**

通信前预留端到端资源，资源独占。通信结束后资源释放。

• 优点：能在请求时间内为端到端保持一个确定量的带宽

• 缺点：不考虑需求。静默期电路空闲而效率较低

• TDM（时分复用)：时间划分为固定期间的帧，每个帧划分为固定数量的时隙，每条电路分配专有时隙，周期性得到所有带宽

• FDM（频分复用)：每条链路分配专用频段，频段宽度为带宽。FDM 需要复杂的模拟硬件来将信号转换到合适频带上。持续得到部分带宽

端对端电路传输时间与链路数量无关

**[网络的网络：ISP 的策略]**

ISP 采用多层结构：局域网连接到区域 ISP，区域 ISP 间直连/ 通过因特网交换点 IXP 对等相连/连接到更高层 ISP

**[为什么互联网采用分组交换]**

同样的链路容量，分组交换允许支持更多的用户（概率意义下）；轻负载时，分组交换可以更快地服务用户

*网络应用是计算机网络存在的理由*

**第一章补充**

**分组交换和电路交换中的同步时分复用的区别**：

• 分组交换：用户使用链路的模式不固定

• TDM ：用户使用链路的模式固定

**链路与电路**：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

不同的网络以所提供的服务为区分

不同的服务以什么为区分： 延迟、带宽、丢失率、端节点数目、服务接口、可靠性， 单播/多播，实时，消息/字节流

**路由：**分组从发送终端到接收终端经过的通信链路及分组交换机的序列。

## 1.5 协议层次与服务模型

网络协议定义了通信实体之间交换的报文的格式和次序，以及在发送/接收报文、或其它事件后采取的动作

**[协议分层]**

每一层的功能实现都要依赖其下各层提供的服务

• 优点：显式的层次结构易于确定系统的各个部分及其相互关系，且模块化使更新系统组件为容易，不会影响其他组件的工作

• 缺点：高层可能冗余低层功能，可能违反层次分离(跨层获取信息)

**[五层因特网协议栈]（由高到低）**

• 应用层：端系统上各种网络应用之间传输报文：FTP,SMTP,HTTP

• 传输层：程序的网络接口间（进程-进程）报文段传输：TCP UDP

• 网络层：源主机-目的主机之间数据报传输：IP/路由选择协议

• 链路层：相邻网络设备之间帧传输：PPP/以太网

• 物理层：在物理媒体上传输比特比特(帧/数据报/报文段/报文)

应用层、传输层是端系统的软件实现，物理层与链路层在链路相连网卡中实现，网络层为软硬件混合

**[OSI 模型] 共 7 层**

开放系统互联模型：应用层传输层间加了表示层(顶)与会话层。

• 表示层：解释交换数据含义，如压缩、解密等

• 会话层：提供数据交换定界，同步功能，建立检查点，提供恢复**[封装]**

因特网体系结构将其复杂性放在网络边缘

• 路由器底三层，链路层交换机底二层，主机全五层

• 每一层的分组有两种类型的字段：首部+有效载荷字段(来自上层)

## 1.6 网络攻击与安全性

网络攻击形式包括

• 向计算机注入有害程序：恶意软件、病毒等

• 攻击服务器和网络基础设施：拒绝服务攻击 DoS

• 嗅探：监听网络中的数据包并试图获取信息

• 伪装：伪造地址进行通信

**第二章 应用层**

**2.1 应用层协议原理**

网络核心设备不在应用层起作用，而仅在较低层起作用。软件开发只能在端系统扇进行，而无法进入网络核心设备

**[网络应用程序体系结构]**

客户-服务器架构：资源集中，流量不均衡，相应延迟高，成本高

• 客户机：发起通信的进程，需要时与服务器通信，不是时时在线，通常使用动态 IP，不与其他客户机直接通信

• 服务器：等待联系的进程，是总是在线的主机，具有永久 IP 地址

（例子：百度、谷歌、微信、邮件）

对等架构（P2P)：易于扩容，成本低，安全性、可靠性、性能面临挑战

没有总是运行的服务器，任意一对端系统可直接通信，每个对等方(间断连接的主机对)可以请求服务也可提供服务。动态 IP

（例子：文件共享、下载器、电话、视频）

**[进程通信]**

网络应用程序由成对的进程组成，不同主机上的进程通过交换报文通信（相同主机则采用系统调用）。将发起对话的进程标识为客户

一个进程可以既是服务器又是客户

进程通过套接字收发报文。套接字是一台主机应用层与传输层的接口，也是应用程序与网络间的 API。为接收报文，每一个进程要有标识（端口号)，因为主机进程数目较多而无法用 IP 地址标识进程。常用端口号：HTTP 80、Mail(SMTP) 25

**[传输服务] 因特网提供 UDP、TCP 两个传输层协议**

• 运输层提供的服务种类：可靠性,吞吐量,定时,安全性。吞吐量和定时未运用在因特网协议中

• TCP：面向连接的可靠传输，有流量控制、拥塞控制，不提供及时性、最低带宽保证。连接需要建立与拆除。*应用层强化安全版本：SSL*

• UDP：提供最小服务，是无连接、不可靠传输，不提供连接建立、可靠传输、流量控制、拥塞控制、及时性、最低带宽保证等服务

• 应用层协议：HTTP SMTP Talnet FTP SIP RTP

• 应用层协议定义了：报文类型、报文语法、字段语义、进程何时以及如何发送报文 and 响应报文的规则

**2.2 Web 与 <超文本传输协议>HTTP**

HTTP 定义了 Web 客户向 Web 服务器请求 Web 页面的方式，以及服务器向客户传送 Web 页面的方式，是一种无状态通信协议：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

TCP 为 HTTP 支撑运输协议：HTTP 协议不担心数据丢失，也不关注 TCP 从网络的数据丢失和乱序故障中恢复的细节

网页由基本 HTML 文件和引用对象构成。 HTML 文件通过对象 URL 引用对象，其自身也算是对象。一个对象是一个文件(HTML JPEG)

**[持久连接与非持久连接]**

客户在 80 端口发起一个到服务器 xxx 的 TCP 连接（双方分别有一个套接字)→客户经套接字向服务器发 HTTP 请求报文→服务器经套接字接受报文并封装对象发送响应报文→断开 TCP

Round-Trip Time：一个小分组从客户发送到服务器再返回客户的时间

• HTTP/1.0：非持久连接，一个连接只传输一个请求报文和响应报文

非并行单个对象：建立连接 RTT + 请求传输 RTT + 对象传输时间 T

M 并行 n 对象： (2 RTT + T)\* ceil(n/M)

• HTTP/1.1：持久连接，所有的请求及其响应经相同 TCP 连接发送。

流水线（默认）：建立连接RTT + 解析所有对象RTT + 发送请求RTT

非流水线 n 对象：建立连接 RTT + n \* RTT

**注意一个网页带 n 个引用对象，总共有 n+1 个对象**

同一服务器同一客户的多个页面可以使用一个连接。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

**[cookie]：**允许站点对用户跟踪，以将内容与用户身份联系，或限制用户的访问。可在 HTTP 上建立有状态会话层。带来隐私问题

1 响应报文 cookie 首部行 2 请求报文 cookie 首部行

3 用户系统保存 cookie 文件 4 后端数据库<set-cookie 识别码>

存储位置：服-后端数据库，返回 ID 给客户；客- cookie 文件中

**[web 缓存器/代理服务器]**

既是客户端又是服务器，保存最近请求过的对象的副本。减少响应时间、减少通信量、减少 web 流量,提升应用性能。用户可以设置浏览器：所有 HTTP 请求首先发往 web 缓存。若命中则缓存返回对象，否则从原始服务器获取对象并缓存在本地，发送给客户

• 条件 Get 方法：web 缓存确认缓存对象为最新的手段。

在 HTTP 请求报文中使用 GET 方法，增加一个 If-Modified-Since 首部行。若无更新返回 304，且不包含对象，否则返回 200 OK 以及新对象

**2.3 文件传输协议 FTP （File Transfer Protocol)，20/21 端口**

FTP 采用两个并行的 TCP 连接传输文件：

• 控制连接(21)：贯穿整个用户会话期间，采用命令/响应交互(非报文)

• 数据连接（20)：一条连接仅传一个文件，随文件传输结束而关闭

[**分开控制、数据连接原因]**：

• 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（可以随时终止）

用关闭数据连接的方式结束传输：允许动态创建文件

**[FTP 是有状态服务]：**FTP 服务器必须在整个会话期间保留用户的状态。这限制了 FTP 同时维持的会话总数

## 1.4 分组交换的延迟

衡量互联网性能的主要指标：延迟、吞吐量、丢包率

**[分组交换延迟的来源] 不同情况下主要延迟来源不确定**

• 节点处理延迟（读取首部信息、查 bit 错、确定输出链路，μs)

• 排队延迟（在缓存处等待传输，变化范围最大)

• 传输延迟（将分组移动到链路，μs～ms，取决于链路带宽）

• 传播延迟（分组在链路上传播，μs～ms，取决于链路长度）

**[一些概念]**

• 最大吞吐量：路由器能够转发分组的最大速率

• 流量强度：La/R（R 链路带宽，L 分组大小，a 到达速率）

上式应 ≤ 1，越接近 1 延迟越高，但不会趋向于无穷(因为缓冲满了之后直接丢包，不会引入更高的排队延迟)

**[端到端延迟]**

分组传输路径上所有结点的延迟之和，同时包括了端系统、应用程序延迟等。对端到端延迟高度敏感的：网络电话、视频会议等实时交互性应用；中度敏感的：网页浏览等在线交互应用

**[端到端吞吐量]**

• 定义：单位时间内向接收端成功交付的数据量 bps

• 瓶颈链路：瓶颈链路的带宽限制了端到端吞吐量(取最小值)

• 影响因素：数据流过的链路的传输速率、沿着路径的传输速率、干扰流量。特别的，若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

## 2.4 电子邮件系统

三部分组成：用户代理、邮件服务器、简单邮件传输协议 SMTP

用户代理：允许阅读回复转发保存撰写报文；邮件服务器：存放邮件

• 邮件服务器上运行着系统守护进程，用于在服务器间传输邮件，并及时将收到的邮件放入用户信箱（报文传输代理 MTA）@后是名字

**[SMTP] 使用 TCP 协议，端口号 25.发送方客户端 & 接收方服务端**

• **持续连接**。每台邮件服务器上既运行 SMTP 客户端也运行 SMTP 服务端。SMTP 不使用中间邮件服务器。报文由 7bit ASCII 文本组成

• 可以在一条 TCP 连接上传输多个报文，也可以在一个方向的报文传输结束后，向另一个方向上传输报文

建立服务器间 TCP 连接→应用层握手→发送报文→关闭连接

• 如果接收方的邮件服务器没有开机，那么报文会保留在发送方的邮件服务器(报文队列)上并等待进行新的尝试

**[SMTP 与 HTTP 对比] 二者都是文件传输协议，均为持续连接**

• SMTP 是一个**推**协议：发送邮件服务器推到接受邮件服务器。HTTP 主要是一个**拉**协议：用户从 web 服务器拉取。**推送也可以用 HTTP**

• SMTP 只能使用按 7 比特ASCII 码进行编码，FTP、HTTP 无此限制。

• SMTP使用“.”表示报文结束（FTP：关闭连接；HTTP：使用长度域）

**[邮件报文格式与多用途因特网邮件扩展协议 MIME]**

MIME 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

**[邮件访问协议] POP3（无状态）IMAP（有状态）HTTP（无状态)**

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。SMTP 将邮件推向邮件服务器，但不能从邮件服务器把邮件拉下来到用户代理。 为此引入邮件访问协议将邮件服务器上的报文传到本地 PC

• POP3 端口110 TCP 连接，特许<发送用户名与口令>→事务处理<取回删除阅读报文>→更新（发出 quit 命令后)。通常有<下载并删除>(不能重复读)模式和<下载并保留>方式。POP3 服务器保留了一些状态信息，但不会在会话过程中携带信息，这简化了协议的实现。

• IMAP 将报文与一个文件夹一一对应，可以移动阅读删除查询邮件、创建文件夹，**维护用户状态信息**，用户代理可获取报文组的部分。

• HTTP：用户代理就是浏览器。代理与服务器之间采用 HTTP 通信。邮件服务器之间采用 SMTP 通信

• HTTP 采用报文交互，SMTP 和 FTP 采用命令/响应交互

**2.5 P2P 应用：自扩展——每个对等方既是消费者，也是分发者**

集中式目录的问题：单点失效、目录服务器称为性能瓶颈、版权问题

**[文件分发]**

分发时间：所有 N 个对等方都得到文件 F 副本所需要的时间。假定服务器带宽 u\_s，客户 i 带宽为 d\_i

• 客户-服务器架构：max(NF/u\_s, F/min(d\_i))

• P2P 架构： max(F/u\_s, F/min(d\_i),NF/(u\_s+Su\_i))

*共计 NF 的文件必须在网络中被上传，网络中总上传带宽为 u\_s+Su\_i*

**[案例 BitTorrent]**

tracker: 跟踪洪流中的对等方；|Torrent: 参与特定文件分发的对等方集合；|对等方加入洪流：向跟踪器注册，获得对等方列表，尝试与每个对等方建立 TCP 连接

**操作：**

• 不同对等方拥有不同的数据块

• 每个对等方每 30s 随机选择邻居发送自己的请求<最稀罕优先原则：优先获得邻居中拷贝数量最少的数据块>

• 自己响应当前向自己发送数据最快的 4 个邻居的请求（每 10s 重新评估，每 30s 随机另选一个对等方响应请求）

这样对等方总能趋于找到彼此协调的速率下载

问题：现实中存在仅索取而不提供的结点；以发送数据快慢作为指标并不公平：对等方的带宽不相同

## 2.6 DNS ：因特网的目录服务

• 一个由分层的 DNS 服务器实现的分布式数据库

• 一个使得主机能查询数据库的应用层协议(不是普通的用户应用)

主机的标识方法：主机名(字符串-人类)、IP 地址(32bit-机器)

**[DNS 提供的服务] 应用层协议、缓存机制**

• 允许拥有复杂主机名的主机具有一个或多个别名，提供与主机别名对应的规范主机名及 IP 地址，迁移服务不需要修改主机名。

• 提供邮件服务器的规范主机名及 IP 地址，允许用域名作为邮件服务器别名，邮件与 web 同名。

• 允许一个规范主机名对应一组 IP 地址，多个邮件服务器同名，将服务请求分配到功能相同的一组服务器上。

DNS 报文长度 < 512B 用 UDP，> 使用 TCP，未知长度先 UDP，端口 53

**[分布式 DNS]**

## Why not集中式 DNS？单点失效；流量集中-单个 DNS 服务器需处理全部查询；响应时间长-远距离的集中式数据库；需要维护庞大的数据库

• 层次结构：根 DNS(13个)、顶级域 DNS(TLD)<组织域、国家域、反向域，知道所有二级子域名服务器地址>、权威 DNS <提供机构内部服务器名字映射、提供一个主域名服务器、一个或多个辅助域名服务器>

本地 DNS 服务器不属于层次结构，仅起代理作用：向层次结构发送 DNS 查询报文，将结果缓存（一段时间丢弃)并返回。通常会缓存 TLD 服务器地址，因而很少访问根服务器。缓存可以改善延迟，降低流量

DNS 查询是 DNS 服务器间通讯。迭代（8 次)/递归（8 次)

**[域名空间]**

因特网被划分为 200 多个顶级域，且可以被继续划分。域：名字树只用任何一棵子树；标记：树都每一个结点都有一个不超过 63 字符的标记(总根节点为空)；域名：从根节点到该结点的标记序列

Web 服务器：[www.bulabula； Ftp](http://www.bulabula；Ftp) 服务器：ftp.bulabula

**[工作机制] <黑箱>**

应用程序将主机名传给 DNS 解析器，解析器向本地 DNS 服务器发送包含主机名的请求，本地 DNS 根据缓存情况向层次结构查询，将结果返回给解析器，解析器将包含主机名与 IP 的回答发给应用程序

• 层次结构中的查询为迭代查询：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

• 主机的查询为递归查询：发送一条查询消息，经过一系列路径来到本地 DNS 服务器，最后得到一条准确结果

**[DNS 资源记录](RR)：四元组(name,type,TTL<记录的生存时间>,value)**

Type=A：Name 是主机名，Value 是对应的 IP 地址

Type=NS：Name：域，Value：域的权威 DNS 服务器主机名

Type=CNAME：Name：别名，Value 是对应规范名

Type=MX：Name：别名，Value：对应邮件服务器规范名

• 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

**插入记录**：向注册登记机构提供权威 DNS 的名字和 IP，NS 和 A 输入TLD，并确保用于 Web 服务器的 A 和邮件服务器是的 MX 输入到权威 DNS。

**第二章补充**

• FTP 常用命令：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）

**第三章 传输层**

## 3.3 可靠数据传输 (RDT) 原理：分组无比特损坏丢失，且按序接收

• rdt\_send()：上层实体调用，将数据放入 rdt

• udt\_send()：rdt 调用，将分组交给下层实体

• rdt\_rcv()：下层实体调用，将分组交给 rdt

• deliver\_data()：rdt 调用，将数据交给上层实体

**[RDT]：针对不同信道条件而构造的不同协议，因为下层协议未必可靠**

• RDT1.0：底层信道完全可靠，发送能力 < = 接收能力

数据到来之后直接发送/接收即可，无需额外操作

• RDT2.0：可能存在比特翻转。使用自动重传请求协议 ARQ

发送方发送后等待反馈，若收到 NAK 则重传，若收到 ACK 则继续发

接收方收到分组后检错，若有误则发送 NAK，否则发送 ACK

**停等协议**：收到 ACK 并离开该状态才继续获取数据

存在的缺点：ACK/NAK 本身也可能受损

• RDT2.1：在 2.0 基础上为分组添加序号 0/1。接收方发现序号相同（冗余）分组直接丢弃，收到期望序号分组发送 ACK

• RDT2.2：用 ACK0/ACK1 代替 NAK。在 2.1 的基础上：

接收方仅对正确接收到分组发送相应编号 ACK，对于出错分组重发编号最近的一次 ACK

发送方仅在收到期望 ACK 后发送下一分组，否则直接重发分组

• RDT3.0：可能存在比特翻转、丢包：引入定时器：超时后重传

发送方发送分组后启动定时器，在 2.2 的基础上，如果计时器超时，则视作丢包，发送方重新发送该分组

存在的缺点：链路利用率太低。有效传输时间只有 (L/R)/(RTT+L/R)

改进思路：引入流水线思想，允许发送方有多个已发送未确认的分组。可增加序号范围，双方缓存多个分组。

**[回退 N 步 GoBackN(GBN)]：实际使用更多的流水线可靠传输协议**

***发送方：最多允许存在 N 个已发送未确认的分组***

[0，base-1]：已发送已确认

[base，next-1]：已发送未确认，对最早的分组使用**计时器**

[next，base+N-1]：未发送待发送，每收到一个 base 包就发送 next

[base+N，+∞]：未发送，且超出 N 所以不可发送

• 发送窗口 = 已发送未确认序号 + 未发送可用序号 = N，每收到一个期望的包（base）则在发送下一包后将窗口右移一位

• 若定时器超时，则重传发送窗口内的所有分组

• 累积确认： ACKq 表明序号至 q 的分组均已被正确收到

***接收方***

只对正确收到的序号连续的一系列分组中最高的回 ACK。若分组未按序到达，则直接丢弃并重发最近的一次 ACK

窗口长度 N 与序号空间大小 k 的关系： N ≤ k-1

**[选择重传（SR）]**

引入单独确认，缓存失序分组，避免不必要的重传

***发送方***

•已发送未确认与已发送已确认的包是**交错排列**的

•收到第一个已发送未确认的 ACK 后，窗口右移至下一个已发送未确认的包处（新的发送窗口下边界）。

***接收方***

• 一旦收到就发 ACK。若失序，缓存。若收到的是基序号，滑动接收窗口，交付从 n 开始的若干连续分组。如果收到基序号前的冗余分组，说明发送方超时或上一 ACK 丢包，发送 ACK（冗余）。

• 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]不能重叠。所以 N <= 序号空间的一半。

• 每个发送的分组需要一个定时器，以便被单独重发

窗口长度 N 与序号空间大小 k 的关系： N ≤ floor(k/2)

## 3.4 UDP：无连接、不可靠的运输

**[UDP 的实现的功能]**

多路复用和解复用（最基础的）、差错检测（但不尝试恢复）

不提供：可靠/按序交付、延迟/带宽保证，不进行握手

**[UDP 检查和] 若无则将 Checksum 置零**

发送方将报文视作 16b 组成的序列，对其进行二进制加法运算，进位加到最低位。最后对结果取反。计算时从伪头开始，Checksum 视作 0。接收方进行相同运算，结果相同则**认为无错**（其实可能有错）

**[UDP 的优点]**

• 发送速度快（不限制发送速率：没有拥塞控制和流量控制；无须建立连接）；协议简单；分组报头开销小（UDP 8B，TCP 20B)

• 适合应用：容忍丢包但对延迟敏感（流媒体）；单次请求即可（DNS）.不适合：文件传输(需完全可靠)

## 3.2 多路复用与解复用

• 多路复用：将多个套接字标识收集并封装成报文段，传给网络层

• 解复用：根据报文段中的套接字标识，将数据交付到正确的套接字

**[套接字与端口号] <套接字：网络与进程之间传递数据的门户>**

主机中每个套接字应分配一个唯一的标识，报文段中有特殊字段指示要交付的套接字。发送方传输层需在报文段中包含目的套接字标识。接收方传输层需将报文段中的目的套接字标识与本地套接字标识进行匹配，将报文段交付到正确的套接字。

端口号为 16 bit 整数，其中 0-1023 由公共域协议使用(众所周知的端口号)。报文段的前两字段会携带源、目的端口号

端口号的分配：客户端通常调用分配函数自动分配，服务器使用指定端口号

• UDP 套接字的标识为<目的 IP 地址,目的端口号> 二元组。服务器使用单个 UDP 套接字服务所有客户。对于相同的 IP 和端口则通过相同目的套接字连接到相同目的进程。若为源 IP 和端口则直接返回

• TCP 连接套接字为<源 IP 地址,源端口号,目的 IP 地址,目的端口号> 四元组。TCP 欢迎套接字位于 12000 端口，客户创建一个套接字与其通信，随后服务器建立一个仅接收来自该客户的连接套接字。

连接套接字与进程并非总是一一对应

## 3.1 概述和运输层服务

在应用程序看来，传输层提供了进程间逻辑通信。忽略通信过程，不同主机上的进程可以认为它们是直接相连的。具体怎么传， 由网络层提供终端（主机）之间的逻辑通信

**[因特网的网络服务]**

• 网络层提供尽力而为的服务：网络层尽最大努力在主机间交付分组，但不做任何担保：不保证交付，不保证按序交付，不保证数据完整。

• 传输层**不能**提供的服务：延迟保证，带宽保证

• 传输层可以提供的服务：数据交付，差错检查，保证可靠按序的交付：TCP; 不保证：UDP

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（主机间交付扩展到进程间交付）

## 3.5 TCP：面向连接的运输

**[TCP 的特性]**

点到点（单个发送方与单个接收方进程间通信）、全双工（同时双向）、面向连接（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）保存在端系统中）、可靠有序 、流水式发送、流量控制、发送缓存(上层数据存入缓存整体发送)、接收缓存(收到正确按序的分组后先放入接收缓存，等待接收方自主读取)、提供累计确认、无边界

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该连接分配任何缓存或变量）

• 最大传输单元 MTU：链路层帧最大长度 1500 字节

• 最大报文段长度 MSS：MTU-TCP/IP 首部=1460 字节

**[往返时间估计] RTT 是实时变化的，为此需要引入估计**

• (瞬时)SampleRTT：测量某报文从发出到收到确认报文之间的时间，不会被已重传的报文段计算（发送方无法确定 ACK 是对第一次还是超时重传后的报文段的确认）

• (指数加权移动平均 EWMA)EstimatedRTT：EstimatedRTT=（1-α)EstimatedRTT+αSampleRTT。一般取α=0.125，初值设为 SRTT

问题：瞬时 RTT 与平均 RTT 有较大的偏差，需要引入安全距离

• 估算 SampleRTT 偏离 EstimatedRTT 的程度 <DevRTT>：

DevRTT=(1-β)DevRTT+β|SampleRTT-EstimatedRTT|,β一般为 0.25

**超时间隔管理(定时器补偿)：**引入 TimeoutInterval = EstimatedRTT + 4·DevRTT，初始为 1，超时翻倍，直到成功发送为止。在收到 ACK 或收到上层数据时按照公式更新（Karn 算法）

**[TCP 可靠数据传输机制]：像一个带累计确认的 SR**

发送方采用流水线方式发送报文，采用重传恢复丢失的报文段，接收方采用累积确认进行响应

***三个主要事件：***

• 从上层应用接收数据：创建并发送 TCP 报文段。若当前没有定时器启动（无已发送未确认报文），则为其启动一个定时器

• 定时器超时：重传最小序号的已发送未确认报文段并重启定时器

• 收到 ACK：若确认序号大于基序号，则移动发送窗口，终止定时器，若有则为当前最早已发送未确认的报文段启动定时器

***简化协议的接收方（累积确认)：***

• 仅在正确、按序收到报文段后更新确认序号并发 ACK，否则重复最近的一次 ACK，并缓存失序的报文段（避免重复发送 ACK）

***简化协议的发送方（流水式发送）：***

• 流水式发送，仅对最早未确认的报文段使用一个定时器，并在超时候重发最早未确认的报文段（避免超时长度过小导致大量重发）

***改进：接收方推迟确认***

• 为减少通信量，接收方可在收到若干报文段后，仅发送一个累积确认报文。TCP 规定：推迟确认的时间最多为 0.5s，接收方至少每隔一个报文段就采用正常确认，且对于失序或填充间隙低端的报文段立即发送确认。

缺点：延迟太大时导致不必要的重传；RTT 估计不准确

***改进：发送方快速重传-利用重复 ACK 检测丢包***

仅通过超时触发重传的问题：超时间隔较长，增加了延迟

• 大多数情况下下层协议(IP)能够按序交付分组，重复 ACK 极有可能是因为丢包。故当收到对同一序号的三次重复确认时，立即重发该报文段。

**3.6 TCP 流量控制<目标：接收方>**

引入原因：为了消除发送方发送过快、接收方读取过慢而导致接收缓存溢出的可能性。发送方应调节速度使接收缓存不溢出

**[流量控制服务]**

• **为什么 GBN 与 SR 没有？**因为正确且按序的分组会被迅速交给上层，而不会占用缓存，因此不会出现接收缓存溢出的情况

• **为什么 UDP 没有？**因为 UDP 不保证交付，接收缓存溢出没有违反其服务承诺

**[接收窗口]**

发送方维护接收窗口 rwnd（动态)，接收方用于给发送方通告缓存可用空间。引入变量 LastByteRcvd：放入缓存的最后一个字节、LastByteRead：从缓存读出的最后一个字节、LastByteSent：发送的最后一个字节、LastByteAcked：确认的最后一个字节。则有：

LastByteRcvd – LastByteRead <= RcvBuffer

rwnd = RcvBuffer-[LastByteRcvd - LastByteRead]

LastByteSent-LastByteAcked <= rwnd

接收窗口为 0 时，发送方立即停止发送。此时发送方启动定时器，不断发送零窗口探测报文段（序号为上一段中最后一个字节的序号），从接收方的响应中获知窗口大小。若接受窗口依然为 0 则重启定时器继续等待

**[糊涂窗口综合症]**

**问题：**当发送速度很快，消费速度很慢时，大量带宽被浪费在微小窗口通告与探测报文上

**接收方启发式策略：**当接收窗口达到最大缓冲的一半或最大段长度时接收方才发送更新的窗口大小

**发送方启发式策略：**发送方积累足够多的数据才统一发送。如何确定等待时间？ Nagle 算法 [在新建连接上，数据到来时立即发送；之后当数据量达到一个最大段长度，或期待的 ACK 到来时统一发送数据] 优点：常规情况下不会降低吞吐量，适应网络延时、MSS 长度及应用速度的各种组合

**3.7 TCP 连接管理**

基本目标：双方同意建立连接，且知道对方同意，并保证双方连接的参数初始化保持一致（MSS、序列号等）

**[建立连接的三次握手]**

两次握手的局限性：服务器收到请求后没有向客户确认，可能会因为重传而产生半开连接，从而消耗服务器资源

握手过程：

• 客户 TCP 先向服务端 TCP 发送连接请求，不包含应用层数据，设置 SYN=1，并随机选择起始序号 client\_isn （不从 1 开始是为了防止新旧连接产生干扰，可以采用基于时钟的起始序号选择法）

• 服务器分配 TCP 缓存与变量，发送 ACK=client\_isn+1，设置SYN=1，并随机化起始序号 server\_isn。不包含数据

• 客户端分配 TCP 缓存与变量，发送 ACK=server\_isn+1，序列号为 client\_isn+1， 可以包含数据。之后发送的报文中 SYN==0

**[中断连接的四次握手](可以合并成 3 次)**

双方进程的任何一个都可以终止连接。不妨假设客户端发起终止

• 客户端向服务器发送序列号为 x， FIN=1 的关闭报文段，并等待服务器确认。此时客户端不可继续发送数据，但可以接收数据

• 服务器收到后，向客户端发送序列号为 y, ACK=x+1 的响应报文段，确认客户端请求并启动计时器。此时服务器仍可发送数据

• 等待一段时间后（超过设定值没有通信），服务器发送 FIN=1 的关闭报文段，等待客户确认。此时服务器不可发送数据

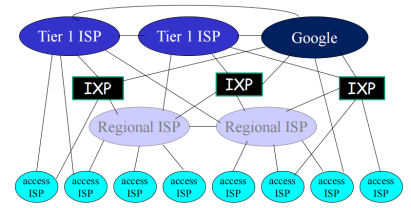
• 客户端向服务器发送对应 ACK，等待一段时间（2 倍最大生命周期）后关闭连接，所有资源被释放。若此时 ACK 丢失则重传 ACK

*实际过程中第 2、3 次报文段可以合并*

**[SYN 洪泛攻击]**

问题：建立的第二次握手时服务器直接分配了资源，在等待 30s-120s 后若没有收到第三次握手则释放。攻击者可以发送大量 SYN 报文并不发送 ACK，被造成服务器资源大量占用

一种解决方案：将服务器分配资源的时间放在第三次握手之后。第二次握手时服务器发送一个特殊序列号，收到 ACK 后再进行检验，以确保连接的安全性与同一性。但这样加大了服务器的负担



今天的因特网结构

# 额外补充

## [TLD 服务器与权威服务器]

顶级域(TLD)服务器：每个 TLD 服务器负责一个顶级域，知道其所有二级子域的域名服务器地址

权威 DNS 服务器：机构的 DNS 服务器，提供机构内服务器(如Web，mail)的主机名-IP 地址映射；提供一个主域名服务器、一个或多个辅助域名服务器；可由机构维护，也可委托 ISP 维护

**[因特网生态系统]**

接入 ISP、区域 ISP、第一层 ISP

• 存在点 PoP：低层 ISP 接入高层 ISP 的地方

• 多宿：一个 ISP 可以与两个或多个上层 ISP 连接

• 对等：相同层次上的一对 ISP 直接相连，不进行结算

• 因特网交换点 IXP：多个 ISP 在这里共同对等

**[DNS]**

DNS 是因特网的核心功能，但是一个应用层服务：使用客户-服务器模式运行在端系统之间；利用传输层协议传输报文；使用者不是用户，而是其它应用程序

**[顶级域]**

## 顶级域分为组织域、国家域和反向域三种。

• 组织域：美国国内及一些国际组织使用

• 国家域：使用二字符的国家代码，每个国家对应一个

• 反向域：域名为 arpa，用来把一个 IP 地址映射为名字

**[socket API 与传输层服务]**

## 应用通过 socket API 可以调用两种传输服务：不可靠的数据报服务（UDP 协议）和可靠的字节流服务 （TCP 协议）

***UDP***

报文传输服务。由于没有建立管道，应用程序发送每个报文必须给出远程进程地址。服务器使用一个进程和一个套接字为所有客户服务，一次请求-响应完成一次服务

## *TCP*

字节流传输服务。由于建立了管道，应用程序只需向套接字中写入字节序列，不需指出远程进程地址。服务器为每个客户单独生成一个套接字和一个新进程，允许双方长时间通信

**[发送窗口与接收窗口]**

## *发送窗口*

限制已发送未确认的分组数目。当收到基序号(最早已发送未确认)的 ACK 时，滑动发送窗口

GBN：已发送已确认和已发送未确认的序号不交织（采用累积确认）

SR：已发送已确认和已发送未确认的序号可能交织在一起

## *接收窗口*

限制可以接收的分组数目。接收窗口之外的分组被直接丢弃

GBN：接收窗口只包含当前期待接收的分组序号

SR：接收窗口包含以下三种序号：期待但未收到、失序（已缓存）且已确认、可接受

**[GBN、SR、TCP]**

## *Go-Back-N*

接收方：

• 不缓存失序的分组、使用累积确认、对失序分组发送重复 ACK

发送方：

• 超时后重传从基序号开始的所有分组仅维护基序号和下一个序号

***SR***

接收方：

• 缓存失序的分组、单独确认每个正确收到的分组

发送方：

• 仅重传未被确认的分组

## *TCP*

接收方：

• 缓存失序的报文段、使用累积确认、对失序分组发送重复 ACK

发送方：

• 超时后仅重传最早未确认的报文段、仅维护基序号和下一个序号

## *修改的 TCP [RFC2018]*

接收方：

• 缓存失序的报文段、在 SACK 选项头中给出收到的非连续数据块的上下边界

发送方：

• 仅重传接收方缺失的数据

**[可靠传输协议的设计过程]**

1. 不可靠信道有哪些出错类型：⽐特错误，丢包

2. 如何发现错误：接收方通过检错码发现出错包，利⽤定时器发现丢失包

3. 采取什么恢复措施：重传出错或丢失的包

4. 恢复措施引⼊的问题：ACK出错、ACK丢失或超时设置过短，导致接收端出现冗余分组

5. 如何解决冗余分组的问题：给分组加上序号，序号多⻓

6. 将以上措施汇总，画出各种情形下的时间线图（正常，分组出错，ACK出错，分组丢失，ACK丢失，过早超时）

7. 归纳发送方、接收方的事件类型及采取的动作，画出 FSM

**[仅使用 NAK 的传输策略]**

• 若偶尔发送数据，仅用 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才发现，丢失的包不能及时重传导致时延大大提升。

• 若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 ，直接持续发送降低了传输时延。

# 田野班小测题

**下面哪种不是 ISP 的连接方式 （a)**

1. 双方连接内容提供商网络实现连接
2. 通过客户-提供商方式连接
3. 双方接入因特网交换点（IXP）实现连接
4. 双方通过对等（Peering）方式连接

**以下那些协议工作在主机/3 层路由器/2 层交换机上?**

主机：应-物/ 路由器：网-物/ 交换机：链-物

**带宽/吞吐率区别：**

• 带宽：单位时间能够传输的最大数据量

• 吞吐量：单位时间能够成功实现的最高传输速率

**进程 A 想发数据到另一台主机上的 B 进程，如何识别 B?**

需要 IP 地址+端口号

**若访问百度，哪个 DNS 服务器你一定不会遇到?**

百度的本地 DNS 服务器

## 3.8 TCP 拥塞控制<目标：网络>

**[拥塞控制原理]**

拥塞：大量分组短时间进入网络，超出了承载能力，路由器中分组延迟加大。为此需要降低进入网络的分组数量

• 端到端拥塞控制：端系统通过观察丢包与延迟推断发生拥塞，进而降低发送速率，网络不提供显式反馈。【TCP 采用，IP 无反馈】

• 网络辅助的拥塞控制：发生拥塞的路由器向发送端/接收端提供直接反馈，指示拥塞程度或给定发送速率。可以选择直接阻塞分组，也可以将信息塞入报文发给接收方，再发给发送方

**[TCP 拥塞控制的三个关键问题]**

**如何感知拥塞？**利用丢包。超时或收到 3 个冗余 ACK 即视作丢包

**如何限制发送速率？**采用拥塞窗口 CongWin 限制已发送未确认的数量。发送方 LastByteSent-LastByteAcked <= CongWin

• 可以粗略认为发送速率 = CongWin/RTT 字节/秒

**如何改变发送速率？*AIMD*** [乘性减：每检测一个丢包事件 CongWin 减半，最小为 MSS；加性增：若没有丢包，每 RTT CongWin 增加 MSS]

***慢启动*** [初始 CongWin 为 RTT，然后每收到一个 ACK 即增加 MSS，直至丢包。若收到 3 个冗余 ACK，则 CongWin 降为一半，采用 AIMD；若超时，则 CongWin 降为 1，重新开始慢启动，直至达到超时时 CongWin 的一半，随后采用 AIMD 调节]

**[TCP 拥塞控制的实现]**

• 当CongWin < Threshold时，发送方处于慢启动阶段, 拥塞窗口指数增长；当CongWin >= Threshold时，发送方处于拥塞避免阶段， 拥塞窗口线性增长。

• 当收到3个冗余ACK 时, Threshold = CongWin/2，CongWin = Threshold + 3MSS（快速恢复）；当超时发生时，Threshold=CongWin/2，CongWin =1 MSS

**[TCP 吞吐量]**

令丢包时CongWin=W，则发送速率为 W/RTT，发生丢包后速率为 W/2RTT

可近似认为平均传输速率为 0.5(W/RTT+W/2RTT) = 0.75W/RTT

TCP 连接的平均吞吐量为 1.22 MSS/(RTT sqrt(L))，其中 L 为丢包率

**[TCP 公平性]**

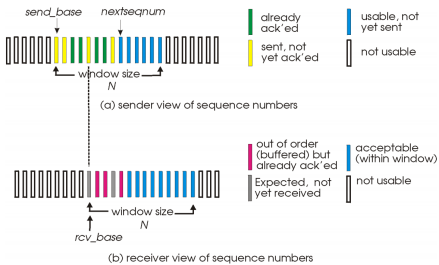
目标：如果 n 条 TCP 连接共享带宽为 R 的瓶颈链路，则每条链路具有平均速度 R/n

考虑 n=2 的情形，未拥塞时以 k=1 斜率增长，拥塞时来到原点连线的中点。如此往复最终总会来到平衡线附近

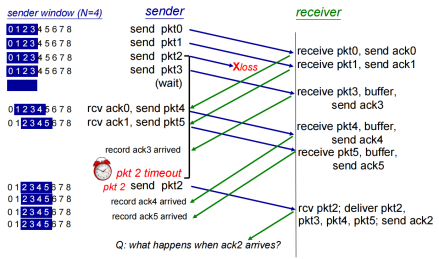
**若连接的参数(RTT，MSS)不同，则不能保证公平性**

• 并行 TCP 连接不能保证公平性：建立的连接越多，分配到的带宽也就越多

• 多媒体应用通常不使用 TCP：不希望拥塞控制限制发送速率。然而 UDP 在拥塞发生时无法感知，影响网络整体性能，且抢占了 TCP 让出的带宽。所以后续引入 DCCP = UDP + 拥塞控制



SR 的发送方与接收方序号空间



一个 SR 示例

# 补充

**TCP 如何发送紧急数据?**

• 紧急标志位 URG 置 1

• 紧急数据置于 TCP 段数据（载荷)前部

• 紧急指针指向紧急数据的最后一个字节

**TCP 接收方何种情形需要立即进行确认?**

• 连续两个段按序到达，且前一个未确认

• 收到失序段（序号比期望的序号大)

• 收到丢失段

• 收到重复段

**Alice 从她的终端登陆到公司的文件服务器上下载了 4 个文件。请问 Alice 的终端和公司的文件服务器之间总共建立了几条 TCP 连接？这些 TCP 连接分别用来传输什么?**

共 5 条连接。一条控制连接，用于传输命令和响应。四条数据连接,每条连接用于传输一个文件。

**在某个时刻，Alice 的邮件服务器和 Bob 的邮件服务器之间需要交换一批邮件， 这两个邮件服务器之间需建立几条 TCP 连接?**

一条 TCP 连接（SMTP）

**以下哪些应用层协议可能会被用来传输一个邮件报文：HTTP，FTP， SMTP， POP， DNS**

HTTP， SMTP， POP

**Alice 向Bob 发送了一封邮件，Bob 阅读了这封邮件。这封邮件在 Alice 的用户代理、Bob 的用户代理、Alice 的邮件服务器、Bob 的邮件服务器中停留过。请按顺序列出这封邮件经过的地方，并指出经过邮件服务器时是进入发送队列还是进入信箱。**

邮件从 Alice 的用户代理到达 Alice 的邮件服务器，进入发送队列；然后到达 Bob 的邮件服务器，进入 Bob 的邮箱；最后到达 Bob 的用户代理。

**如果 TCP 服务器支持 n 个并发连接，每个连接来自不同的客户机主机， TCP 服务器将需要多少个套接字？这些套接字是怎么分配的?**

服务器需要（n+1）个套接字。一个套接字用于监听来自客户的连接请求；其余 n 个套接字，每个用于和一个客户进程进行通信。

**判断题**

• 假设主机 A 通过一条 TCP 连接向主机 B 发送一个大文件，如果某个报文段的序号为 m，则其后续报文段的序号必定是 m+1 （×）

• 假设主机 A 通过一条 TCP 连接向主机 B 发送一个序号为 38、包含 4 个数据字节的报文段，则主机 B 对该报文段的确认号必定是 42（×）

• 假设主机 A 通过一条 TCP 连接向主机 B 发送一个大文件，主机 A 已发送但未被确认的字节数不会超过接收缓存的大小（√）

• 在 TCP 连接的持续过程中 TCP 头中的 rwnd 不会变化 （×）

**发送方 TCP 的基序号 SendBase 和接收方缓存中的LastByteRcvd 之间的关系为 (A)**

（A）LastByteRcvd ≧ SendBase-1

（B）LastByteRcvd ≧ SendBase

（C）不能确定

**假设发送方 TCP 收到了确认序号 y（表示 y 之前的字节均已正确收到），则 y 与接收方缓存中的 LastByteRcvd 之间的关系为（B）**

1. LastByteRcvd = y-1 （B）LastByteRcvd ≧ y-1 （C）不能确定

**假设主机 A 在一条 TCP 连接上发送了一大批数据，然后在 t1 时刻变得空闲(因为没有更多的数据需要发送)。在相对较长的一段时间空闲后，在 t2 时刻又有一大批数据需要发送。你认为此时主机 A 应当使用 t1 时刻的 CongWin 和 Threshold， 还是应当使用慢启动发送数据？为什么？**

答：应使用慢启动发送数据。可知 t1 时刻的 CongWin 和 Threshold 可能较大。经过了相对较长的一段时间后，网络状态可能发生了变化，此时应使用慢启动逐渐提高发送速度，以免一下子发送大量数据引起网络拥塞。

**写出至少三种接入网技术。对于每一种接入网技术，指出使用的传输媒体是什么。**

接入网技术： DSL， HFC，以太网， WiFi， 3G， …… （写出三种即可）DSL 使用双绞线， HFC 使用光纤和同轴电缆，以太网使用双绞线或光纤， WiFi 和 3G 使用电磁波

**假设主机A通过一条TCP连接向主机B发送两个紧接着的TCP报文段。若第一个报文段的序号为 80，第二个报文段的序号为 120。请问**：

1）第一个报文段中有多少数据？ 120 - 80 = 40 bytes

2）假设第一个报文段丢失，而第二个报文段到达主机 B。那么在主机 B 发往主机 A 的确认报文中，确认号应该是多少？ 80

**无连接分组交换与面向连接(虚电路)分组交换的区别？**

• 分组格式:前者完全源、目的地址；后者虚电路号

• 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，转发基于索引查找路由表

• 可靠性、顺序性：前者无；后者有

• 建立、维护连接：前者无；后者有

**假定要传送的报文共有 x 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 bps；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 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：***

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

**总时延 D(c)=3kd+x/b**

***分组交换：***

单个分组传输时间：(p+h)/b、第 1 跳传输时间：(x/p)((p+h)/b) (其中 x/p 为分组个数) 、每 1 跳增加 1 个分组的传输时间，总传输时间为 (x/p)((p+h)/b)+(k-1)((p+h)/b)、排队时间：kq 传播时间：kd

**总时延 D(p)=x/p\*(p+h)/b+(k-1)\*(p+h)/b+kd+kq**

**双绞线由两根具有绝缘保护层的铜导线按一定密度互相绞在一起组成，这样可以** 降低信号干扰的程度

**第三章补充**

**[传输层与网络层的关系]**

**网络层：**提供主机之间的逻辑通信

**传输层：**提供进程之间的逻辑通信， 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）

• 传输层也能提供某些服务（可靠的数据传输，加密）

**[许多应用更适合 UDP 的原因]**

• 关于何时，发送什么数据的控制应用可以更为精细（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

**[创建 TCP 套接字的过程]**

• 服务器在 port=x 创建一个欢迎套接字

• 客户 A 创建一个与欢迎套接字通信的客户套接字（假设自动分配端口号 y）

• 服务器在收到客户 A 的连接请求后创建一个连接套接字。该连接套接字只与客户 A 的套接字通信，即只接收具有以下四元组的报文段

源 IP 地址 = 客户 A 的 IP 地址

源端口号 = y

目的 IP 地址 = 服务器的 IP 地址

目的端口号 = x

不同的客户进程与服务器上不同的连接套接字对应

## [TCP 协议在减少重传方面的设计]

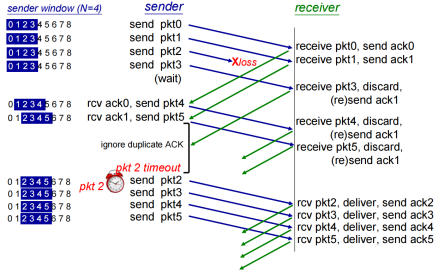
• 利用流水式发送和累积确认，可避免重发某些丢失了 ACK 的报文段

• 只使用一个重传定时器，可避免超时值过小时大量报文段的重发

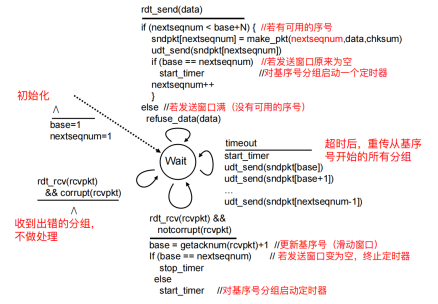
## [定时器补偿]

简单忽略重传报文段的问题：重传意味着超时设置偏小了，需加大。但若简单忽略重传报文段（不估计、不更新 RTT），则超时设也不会更新，形成反复重传的局面。

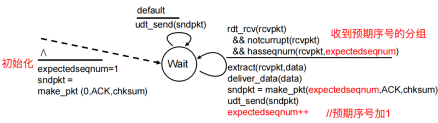
解决方法：采用定时器补偿策略，发送方每重传一个报文段，超时值就增大一倍。若连续发生超时事件，超时值呈指数增长（至一个规定的上限值）。



一个 GBN 示例



GBN 发送方 FSM



GBN 接收方 FSM

**报文格式**

**[HTTP 报文格式]**

HTTP 请求报文： HEAD 让服务器响应但不返回内容，PUT 上传

请求行 <方法 + URL + HTTP 版本> 方法:GET POST HEAD PUT DELETE

首部行 <主机 + 连接类型 + 用户代理 + 语言>

实体体 <仅在 POST 方法时使用，发送用户向搜索引擎提供的关键字表单>

用表单生成的请求报文不是必须使用 POST，也可以放在 GET 中

HTTP 响应报文：

状态行 <协议版本 + 状态码 + 相应状态信息>

200-OK、301-MovedPermanently、400-BadRequest、404-NotFound、505-HTTPVersionNotSupport

首部行 <主机+连接类型+上次修改+报文类型+报文长度+日期>

实体体 包含了所请求的对象本身

**[DNS 报文格式]**

• 首部区域(12B)，包含 16b 标识符，16b 标志来标记报文是查询（0）还是回答、回答者是权威（1）还是非权威、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）

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

**[UDP 报文格式]**

• 伪头（12 B）：32b 源 IP 地址 + 32b 目的 IP + 8b 全 0 + 8b 协议编码 + 16b UDP 报文段长度

• 报头（8 B）：16b 源端口号 + 16b 目的端口号 + 16b UDP 报文段长度 + 16b 检查和

• 有效载荷：来自应用层的数据

**[TCP 报文格式] （1 行 4 Bytes）**

• 源/目的端口号（2B）：多路复用/分解

• 序号：首字节在字节流中的序号，非报文段序号

• 确认号：期望收到的下一字节序号，采用累计确认

• 首部长度(4b)：32bits 为单位的首部长度 通常是 20B

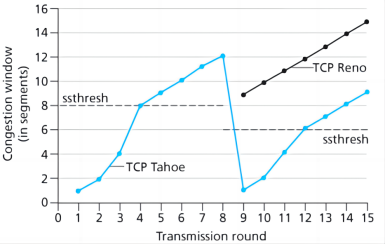
• 标志位(6b)：URG：紧急数据指针有效、ACK 确认收到、PSH：立即交给上层、RST：不接受连接、SYN：建立连接、FIN：结束连接

• 接受窗口字段（2B）：用于流量控制

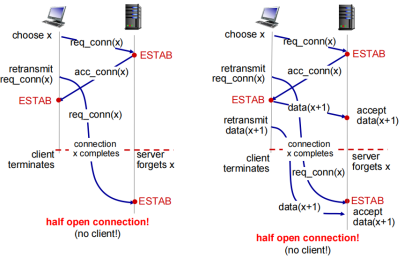
• 检验和、紧急数据指针（指向最后一个字节)

• 选项字段：最大段长度(MSS)：不包括TCP首部，基于路径MTU设置、窗口比例因子：实际窗口为 win-size \* 2^win-scale、选择确认(允许接收端指出缺失的数据字节)

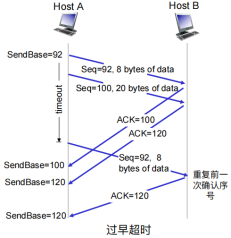
• 数据段



经典的 TCP 慢启动与 AIMD 策略



TCP 2 次握手失败的情形



对于左侧：仅使用一个定时器保证了第一次报文段没有重发；

对于右侧：采用流水式发送避免了重发第一个报文段，采用累计确认避免了重发第二个报文段。

