- **第 7 章　Internet 协议**
  传输层

- **7.2.1 概述**
- ■ **传输层保证端到端可靠的传输。**
- ■ **传输层只存在于通信子网以外的主机中。因通信子网所提供的服务不同。为了能使通信子网的用户得到一个统一的通信服务，就需要设置一个传输层。**
- ■ **它弥补了通信子网提供的服务的差异和不足（分组丢失，分组重复，数据被破坏，分组到达顺序混乱，只对 IP 数据报首部进行校验）。**
- ■ **传输层向高层用户屏蔽了下面通信子网的细节。使得高层用户看见的就好像在两个传输层实体之间有一条端到端的、可靠的、全双工通信通路。**
- **1. 网络层的问题：**
- 网络的性能与用户的要求存在差异。
- **（1）用户要求高速传输，而网络的吞吐量、传输速率和传输延迟等性能不能满足；**
- **（2）用户要求较低的传输费用，对于传输延时要求不高，而网络的吞吐量、传输速率和传输延迟很好，但费用太高，不能满足。**
- **（3）网络的传输差错率不能满足用户的要求；**
- **（4）网络层的分组长度不一定适配用户数据的长度；**
- **（5）网络的数据流量不一定能满足用户的要求。**
- **2. 传输层的目的**
- 为用户(应用进程)提供**可靠的、透明的、有效**的数据传输，使高层用户在相互通信时不必关心通信子网的细节 。
- ■ 可靠：传输层处理并隔离低层的错误。
- ■ 透明：高层用户不涉及点对点间通信的任何细节。
- ■ 有效：全双工、尽量高效。
- **3. 提供的服务**
- **5. 传输服务原语**
- (2) 基本原语功能（传输工作过程）
- (2) 基本原语功能（传输工作过程）
- 6. Windows 的传输原语
- 7. 传输层位置
- 8. TCP/IP 体系中的传输层协议
- **传输层为相互通信的应用进程提供了逻辑通信**
- **应用进程之间的通信**
- 两个主机进行通信实际上就是两个主机中的**应用进程**互相通信。
- 应用进程之间的通信又称为**端到端**的通信。
- 传输层的一个很重要的功能就是**复用**和**分用**。应用层不同进程的报文通过不同的端口向下交到传输层，再往下就共用网络层提供的服务。
- “传输层提供应用进程间的**逻辑通信**”。
- “**逻辑通信**”的意思是：传输层之间的通信好像是沿水平方向传送数据。但事实上这两个传输层之间并没有一条水平方向的物理连接。
- **传输层与网络层的主要区别：**P249-250 谢(4), P181-谢(5),
- **(1)传输层为应用进程之间提供端到端的逻辑通信，而网络层是为主机之间提供逻辑通信。**
- **(2) 传输层还要对收到的报文进行差错检测，而网络层中，IP 数据报首部中的检测和字段，只检测首部是否出现差错而不检查数据部分。**
- **(3) 传输层需要有两种不同的运输协议，即面向连接的 TCP 和无连接的 UDP，而网络层无法同时实现这两种协议。**
- **9. 传输协议的要素**
- ■ **传输协议在实现传输服务时要解决差错控制，分组顺序，流量控制及其它问题。**
- **传输层协议和数据链路层协议的差异：**
- ■ **传输层的环境是两个主机通过多个网络进行通信，这就使传输层比数据链路层的环境复杂的多。由于分组在网络的各个结点都进行存储转发，因此会产生时延，另外网络同时存在多条线路，且连接的数目经常动态地变化，因此流量控制和拥塞控制也较为复杂。**

- **10.传输层功能:**
- 采用的技术手段主要有：
- （1）分流/合流技术：使得具有低吞吐量、低速率和高传输延迟的网络可以支持用户高速传输数据的要求；
- （2）复用/解复用技术和拼接/分割技术：使得具有高吞吐量、高速率和低传输延迟、且高费用的网络可以支持用户低传输成本的要求；
- （3）分段/合段技术：使得传输有限长度用户数据（分组）的网络可以支持用户的无限长数据的传输。
- （4）差错检测和恢复技术：使得差错率较高的网络可以支持用户高可靠性的数据传输要求
- （5）流量控制技术：对连续传输的协议数据单元个数进行限制，避免网络拥塞。
- 7.2.2 传输层的两个协议
- **1. TCP 协议和 UDP 协议的工作方式** P250－谢（4）图 7－5
- ●传输地址称为传输服务访问点 **TSAP**。
- ●两个对等传输实体在通信时传送的数据单位叫作传输协议数据单元 **TPDU (Transport Protocol Data Unit)**。

●**UDP** 在传送数据之前不需要先建立连接。收到 **UDP** 报文后，不需要任何确认。在某些情况下 **UDP** 是一种最有效的工作方式。

●**TCP** 提供面向连接的服务。在传送数据之前必须先建立连接，数据传送结束后要释放连接。**TCP** 不提供广播和多播。由于 **TCP** 要提供可靠的、面向连接的运输服务，因此增加了许多的开销。使协议数据单元的首部增大很多，而且占用许多的处理机资源。

- 还要强调两点:

● 传输层的 **UDP** 用户数据报与网际层的 **IP** 数据报有很大区别。**IP** 数据报要经过互连网中许多路由器的存储转发，但 **UDP** 用户数据报是在传输层的端到端抽象的逻辑信道中传送的。

● **TCP** 报文段是在传输层抽象的端到端逻辑信道中传送，这种信道是可靠的全双工信道。但这样的信道却不知道究竟经过了哪些路由器，而这些路由器也根本不知道上面的传输层是否建立了 **TCP** 连接。

- **7.2.3 端口与套接字** P182－183 谢(5)，P238－黄
- **1. 概念**
- ■ 端口：是一个 **16** 位的地址 **0~65535**。端口就是传输层服务访问点 **TSAP**。
- ■ 端口的用途：用来区分不同的应用进程。
- ■ 端口号只具有本地意义：端口号只是为了标志本计算机应用层中的各进程。在因特网中不同计算机的相同端口号是没有联系的。
- ■ 应用层的各种进程是通过相应的端口才能与传输实体进行交互。应用层的源进程将报文发送给传输层的某个端口，而应用层的目的进程从端口接收报文。
- **2. TSAP 与 NSAP 的对应关系**
- (1) 识别通信的双方——传输地址（TSAP）
- (2) TSAP 与 NSAP 之间的映射
- 1:1 映射——一个 TSAP 映射到唯一一个 NSAP
- 1:*n* 映射——一个 TSAP 映射到多个 NSAP
- *n*:1 映射——多个 TSAP 映射到一个 NSAP，实现多路复用。
- **3. 获知对方 TSAP 的方法**
- **方案一：**每个应用进程(服务)都有固定不变的 TSAP， 并且公之于众，让网络上所有的进程都知道。只适用于少数固定的服务，并且让这些服务一直处于等待接收连接请求状态的情况。
- **方案二：初始连接协议方案**，也是 UNIX 系统所采用的方案。
- **基本思想**：提供服务的机器运行一个进程服务器，它监听一系列端口(TSAP)，等待连接请求。当用户的连接请求到达后，便将用户的请求及所建立的连接一并转交给实际提供此服务的服务器(如同因特网的搜索引擎)。
- **4. 端口的分类：三类端口**
- 熟知端口： 0~1023。
- 注册端口号：1024~49151。使用这个范围

的端口号必须在 IANA(Internet Assigned Numbers Authority) 登记，以防止重复。

- 客户端口号或短暂端口号，数值为49152~65535，留给客户进程选择暂时使用。当服务器进程收到客户进程的报文时，就知道了客户进程所使用的动态端口号。通信结束后，这个端口号可供其他客户进程以后使用。
- 常用的熟知端口号
- **5.端口在进程之间的通信中所起的作用 ：**
- **TCP 与 UDP 的复用和分用** P251－谢(4)
- **6. 插口的概念 P252－253 谢（4），P239－黄**
- **因为主机都是独立地分配自己的端口号。所以在通信中，端口号必须和主机的 IP 地址一起使用。**
- **一个连接由它的两个端点来标识，这里的端点叫插口或套接字（socket）。插口：IP 地址+端口，共 48 位。**
- **如（131.6.23.13，1500）**
- **插口和端口、IP 地址的关系是：**
- **7.2.4 UDP 协议**
- **1.UDP 概述**
- **UDP 只在 IP 的数据报服务之上增加了很少一点功能，这就是端口的功能，即可使应用进程进行复用和分用及差错检测的功能。**
- **UDP 的优点：**P253－谢（4）
- ■ 发送数据之前不需要建立连接。
- ■ UDP 的主机不需要维持复杂的连接状态表。
- ■ UDP 用户数据报只有 8 个字节的首部开销。
- ■ 网络出现的拥塞不会使源主机的发送速率降低。这对某些实时应用是很重要的。
- **使用 UDP 和 TCP 协议的各种应用和应用层协议**见 P253－谢（4）表 7－1 ，P182－谢（5）表 5－1
- UDP 的主要特点
- UDP 是无连接的
- UDP 使用尽力而为交付，即不保证可靠交付，同时也不使用拥塞控制。
- UDP 支持一对一、一对多、多对一和多对多的交互通信。
- UDP 的首部开销小，只有 8 个字节。
- UDP 是面向报文的。UDP 没有拥塞控制，很适合多媒体通信的要求。
- UDP 是面向报文的
- 面向报文的 UDP
- 发送方 UDP 对接收应用程序的报文，在添加首部后就向下交付 IP 层。
- UDP 对应用层的交下来的报文，既不合并，也不拆分，保留报文的边界。
- 接收方 UDP 对 IP 层来的 UDP 用户数据报，在去除首部后就原封不动地交付上层的应用进程，一次交付一个完整的报文。
- 应用程序必须选择合适大小的报文。

2. 使用 UDP 的应用层协议

DNS,TFTP,RIP,BOOTP,DHCP,SNMP,NFS,
IP 电话

3. UDP 适用应用

(1)不太关心数据丢失。如传输视频或多媒体流数据。
(2)每次发送很少量数据。
(3)有自己的全套差错控制机制。
(4)实时性要求较高、差错控制要求不高。

- **4. UDP 用户数据报格式**
- **UDP 用户数据报分首部字段和数据字段。**
- **首部和伪首部的格式：**P185－谢（5），P251－黄
- **UDP 计算检验和的方法：P256－谢（4）, P186－谢（5）**
- **小结：**
- **（1）UDP 与 TCP 的差异：**UDP 直接利用 IP 协议进行 UDP 数据报的传输，因此 UDP 提供的是无连接、不可靠的数据报投递服务。
- **（2）UDP 的使用场合：**UDP 常用于数据量较少的数据传输，例如：域名系统中域名地址/IP 地址的映射请求和应答（Named），Ping 、BOOTP、TFTP 等应用。
- **（3）使用 UDP 协议的好处：**在少量数

据的传输时，使用 UDP 协议传输信息流，可以减少 TCP 连接的过程，提高工作效率。

- **（4）UDP 协议的不足：**当使用 UDP 协议传输信息流时，用户应用程序必须负责解决数据报排序，差错确认等问题。
- **在多媒体应用中，常用 TCP 支持数据传输，UDP 支持音频/视频传输。**
- **7.2.5 TCP 协议**
- **1.** TCP 最主要的特点：
- ■ TCP 是面向连接的。
- ■ 每一条 TCP 连接只能有两个端点(endpoint)，每一条
- TCP 连接只能是点对点的（一对一），不支持广播
- 通信。
- ■ TCP 提供可靠交付的服务。
- ■ TCP 提供全双工通信。
- ■ 面向字节流。
- ■ TCP 保证数据传输可靠、按序、无丢失、无重复的一
- 些机制。
- TCP 面向流的概念
- 特殊方法
- TCP 以报文段（segment）的形式交换数据
- TCP 根据对方给出的窗口值和当前网络拥塞的程度来决定一个报文段的长度（UDP 发送的报文长度是应用进程给出的）。
- TCP 可把太长的数据块划分短一些再传送。TCP 也可等待积累有足够多的字节后再构成报文段发送出去。
- 2. TCP 的连接
- 每一条 TCP 连接有两个端点。
- TCP 连接的端点叫做套接字(socket)或插口。
- Socket（套接字）＝ IP 地址＋端口
- 3. 可靠传输的工作原理
- (1) 停止等待协议
- 要点：
- 在发送完一个分组后，必须暂时保留已发送的分组的副本。
- 分组和确认分组都必须进行编号。
- 超时计时器的重传时间应当比数据在分组传输的平均往返时间更长一些。
- 确认丢失和确认迟到
- 可靠通信的实现
- 使用上述的确认和重传机制，可以在不可靠的传输网络上实现可靠的通信。
- 这种可靠传输协议常称为自动重传请求 ARQ (Automatic Repeat reQuest)。
- ARQ 表明重传的请求是自动进行的。接收方不需要请求发送方重传某个出错的分组 。
- 4.信道利用率
- 停止等待协议的优点是简单，但缺点是信道利用率太低。
- 信道的利用率 U
- 5.流水线传输
- 发送方可连续发送多个分组，不必每发完一个分组就停顿下来等待对方的确认。
- 由于信道上一直有数据不间断地传送，这种传输方式可获得很高的信道利用率。
- 6.连续 ARQ 协议
- 累积确认
- 接收方一般采用累积确认的方式。即不必对收到的分组逐个发送确认，而是对按序到达的最后一个分组发送确认，这样就表示：到这个分组为止的所有分组都已正确收到了。
- 累积确认优点是：容易实现，即使确认丢失也不必重传。缺点是：不能向发送方反映出接收方已经正确收到的所有分组的信息。
- Go-back-N（回退 N）
- 如果发送方发送了前 5 个分组，而中间的第 3 个分组丢失了。这时接收方只能对前两个分组发出确认。发送方无法知道后面三个分组的下落，而只好把后面的三个分组都再重传一次。
- 这就叫做 Go-back-N（回退 N），表示需要再退回来重传已发送过的 N 个分组。
- 可见当通信线路质量不好时，连续 ARQ 协议会带来负面的影响。
- 7. TCP 可靠通信的思想
- TCP 连接的每一端都必须设有两个窗口——一个发送窗口和一个接收窗口。
- TCP 的可靠传输机制用字节的序号进行控制。TCP 所有的确认都是基于序号

而不是基于报文段。

- TCP 两端的四个窗口经常处于动态变化之中。
- TCP 连接的往返时间 RTT 也不是固定不变的。需要使用特定的算法估算较为合理的重传时间。
- **7.2.6 TCP 报文格式** P193－195 谢(5)，P239－黄
- **TCP 报文分为首部和数据两部分。**
- **TCP 报文首部的前 20 个字节是固定的，后面有 4N 字节是根据需要而增加。**
- 例:主机 A 向主机 B 连续发送了两个 TCP 报文段,其序号分别是 70 和 100
- 第一个报文段携带了多少字节的数据?
- 主机 B 收到第一个报文段后发回的确认中的确认号是多少?
- 如果主机 B 收到第二个报文段后发回的确认中的确认号是 180,试问 A 发送的第二个报文段中的数据有多少字节?
- 如果 A 发送的第一个报文段丢失,B 收到第二个报文段后向 A 发送的确认的确认号是多少?
- 窗口字段 —— 占 2 字节。窗口字段用来控制对方发送的数据量，单位为字节。TCP 连接的一端根据设置的缓存空间大小确定自己的接收窗口大小，然后通知对方以确定对方的发送窗口的上限。
- TCP 的伪首部
- 其他选项
- 窗口扩大选项 ——占 3 字节，其中有一个字节表示移位值 S。新的窗口值等于 TCP 首部中的窗口位数增大到(16 + S)，相当于把窗口值向左移动 S 位后获得实际的窗口大小。
- 时间戳选项——占 10 字节，其中最主要的字段时间戳值字段（4 字节）和时间戳回送回答字段（4 字节）。
- 选择确认选项。
- 其他选项
- **TCP 协议服务模型**
- ① 面向连接：socket(=IP 地址＋port)间建立连接
- ② 点对点通信，不支持广播通信
- ③ 高可靠性
- ④ 全双工
- ⑤ 字节流方式
- ⑥ 可靠的连接建立与释放
- ⑦ 提供紧急数据传送功能
- ⑧ TCP 实体以报文段（segment）的形式交换数据
- 7.2.7 以字节为单位的滑动窗口
- 发送缓存
- 接收缓存
- 发送缓存与接收缓存的作用
- 发送缓存用来暂时存放：
  - 发送应用程序传送给发送方 TCP 准备发送的数据；
  - TCP 已发送出但尚未收到确认的数据。
- 接收缓存用来暂时存放：
  - 按序到达的、但尚未被接收应用程序读取的数据；
  - 不按序到达的数据。
- TCP 的特别约定(三点)
- A 的发送窗口并不总是和 B 的接收窗口**一样大**（因为有一定的时间滞后）。
- TCP 标准没有规定对**不按序**到达的数据应如何处理。通常是先临时存放在接收窗口中，等到字节流中所缺少的字节收到后，再按序交付上层的应用进程。
- TCP 要求接收方必须有**累积确认**的功能，这样可以减小传输开销。

7.2.8 TCP 建立与释放连接机制

- 传输连接就有三个阶段，即：连接建立、数据传送和连接释放。运输连接的管理就是使运输连接的建立和释放都能正常地进行。
- 连接建立过程中要解决以下三个问题：
  - 要使每一方能够确知对方的存在。
  - 要允许双方协商一些参数（如最大报文段长度，最大窗口大小，服务质量等）。
  - 能够对传输实体资源（如缓存大小，连接表中的项目等）进行分配。
- 客户服务器方式
- TCP 连接的建立都是采用客户服务器方式。
- 主动发起连接建立的应用进程叫做客户(client)。

- 被动等待连接建立的应用进程叫做服务器(server)。
- 用三次握手建立 TCP 连接
- 用三次握手建立 TCP 连接
- 用三次握手建立 TCP 连接的各状态 P216-谢(5)
- 初始序号的确定 P242-黄
- 可选择 0 或 1
- 实际：设定 ISN 计数器，初始值为 0，每 4 微秒加 1，直到记满 32 位后归 0，这一过程需要 4 个多小时。任何时候建立 TCP 连接时，都选择当前 ISN 计时器的值作为初始序号值。
- 三次握手机制的问题 P243-黄
- 安全问题
  假冒 A （猜出 y+1 序号）
- 半连接
  消耗 TCP 的连接数
- A 必须等待 2MSL 的时间
- 第一，为了保证 A 发送的最后一个 ACK 报文段能够到达 B。
- 第二，防止 “已失效的连接请求报文段”出现在本连接中。A 在发送完最后一个 ACK 报文段后，再经过时间 2MSL，就可以使本连接持续的时间内所产生的所有报文段，都从网络中消失。这样就可以使下一个新的连接中不会出现这种旧的连接请求报文段。
- 7.2.9 TCP 定时管理机制
- TCP 每发送一个报文段，就对这个报文段设置一次计时器。
- 计时器到但还没有收到确认，就要重传这一报文段。
- 问题：超时时间的设定
- 往返时延的方差很大
- 由于 TCP 的下层是一个互联网环境，IP 数据报所选择的路由变化很大。因而传输层的往返时间的方差也很大。
- 加权平均往返时间
- TCP 保留了 RTT 的一个加权平均往返时间 RTT_S（这又称为平滑的往返时间）。
- 第一次测量到 RTT 样本时，RTT_S 值就取为所测量到的 RTT 样本值。以后每测量到一个新的 RTT 样本，就按下式重新计算一次 RTT_S：

新的 $RTT_S = (1 - \alpha) \times$ (旧的 RTT_S) $+ \alpha \times$ (新的 RTT 样本)　　(Jacoson 算法)

- $0 \le \alpha < 1$。若 α 很接近于零，表示 RTT 值更新较慢。若选择 α 接近于 1，则表示 RTT 值更新较快。
- RFC 2988 推荐的 α 值为 1/8。
- 超时重传时间 RTO (RetransmissionTime-Out)
- RTO 应略大于上面得出的加权平均往返时间 RTT_S。
- 最早：2×RTT
- RFC 2988 建议使用下式计算 RTO：
- $RTO = RTT_S + 4 \times RTT_D$
- RTT_D 是 RTT 的偏差的加权平均值。
- RFC 2988 建议的 RTT_D：第一次测量时，RTT_D 值取为测量到的 RTT 样本值的一半。在以后的测量中，则使用下式计算加权平均的 RTT_D：

新的 $RTT_D = (1 - \beta) \times$ (旧的 RTT_D) $+ \beta \times |RTT_S -$ 新的 RTT 样本$|$

- β 是个小于 1 的系数，其推荐值是 1/4。
- 往返时间的测量相当复杂
- TCP 报文段 1 没有收到确认。重传（即报文段 2）后，收到了确认报文段 ACK。
- 如何判定此确认报文段是对原来的报文段 1 的确认，还是对重传的报文段 2 的确认？
- Karn 算法
- 在计算平均往返时间 RTT 时，只要报文段重传了，就不采用其往返时间样本。
- 这样得出的加权平均平均往返时间 RTT_S 和超时重传时间 RTO 就较准确。
- 修正的 Karn 算法
- 报文段每重传一次，就把 RTO 增大一些：

新的 $RTO = \gamma \times$ (旧的 RTO)

- 系数 γ 的典型值是 2 。
- 当不再发生报文段的重传时，才根据报文段的往返时延更新平均往返时延 RTT 和超时重传时间 RTO 的数值。
- 7.2.10 选择确认 SACK

(Selective ACK)

- 接收方收到了和前面的字节流**不连续**的两个字节块。
- 如果这些字节的序号都在接收窗口之内，那么接收方就先收下这些数据，但要把这些信息**准确地告诉发送方，使发送方不要再重复发送这些已收到的数据**。
- 接收到的字节流序号不连续
- RFC 2018 的规定
- 如果要使用选择确认，那么在建立 TCP 连接时，就要在 TCP 首部的选项中加上“允许 SACK”的选项，而双方必须都事先商定好。
- 如果使用选择确认，那么原来首部中的“确认号字段”的用法仍然不变。只是以后在 TCP 报文段的首部中都增加了 SACK 选项，以便报告**收到的不连续的字节块的边界**。
- 由于首部选项的长度最多只有 40 字节，而指明一个边界就要用掉 4 字节，因此在选项中最多只能指明 4 个字节块（32 个字节）的边界信息，加上 1 个字节指明 SACK 选项和 1 个字节选项长度，共 34 个字节。
- **7.2.11　利用滑动窗口实现流量控制**
- ■ **TCP 采用可变发送窗口的方式进行流量控制(滑动窗口协议)。**
- ■ **TCP 的首部窗口字段写入的数值是当前接收窗口的大小。**
- ■ **在通信过程中，接收端根据自己的资源情况，动态调整自己的接收窗口，然后通知发送方，使发送窗口和它的保持一致。**
- **利用可变窗口大小进行流量控制：双方确定的窗口值是 400 ，每一个报文段为 100 字节。**
- 问题 1：死锁问题

问题：接收方应答 0 接收窗口后，发送的非 0 窗口应答报文丢失

对策：持续计数器+探测报文（见下一页）

- **持续计数器+探测报文**
- TCP 为每一个连接设有一个持续计时器。
- 只要 TCP 连接的一方收到对方的零窗口通知，就启动持续计时器。
- 若持续计时器设置的时间到期，就发送一个零窗口探测报文段（仅携带 1 字节的数据），而对方就在确认这个探测报文段时给出了现在的窗口值。
- 若窗口仍然是零，则收到这个报文段的一方就重新设置持续计时器。
- 若窗口不是零，则可以通信。
-

可选发送机制:

- **第一种机制**是 TCP 维持一个变量，它等于最大报文段长度 MSS。只要缓存中存放的数据达到 MSS 字节时，就组装成一个 TCP 报文段发送出去。
- **第二种机制**是由发送方的应用进程指明要求发送报文段，即 TCP 支持的**推送**(push)操作。
- **第三种机制**是发送方的一个**计时器期限到**了，这时就把当前已有的缓存数据装入报文段（但长度不能超过 MSS）发送出去。

问题 2：效率考虑　P204-205 谢(5),P246－黄

问题：每次发送一个字节:需要大量应答。(如 Telnet)

解决：Nagle 算法:发送第一个字节后将后续的字节缓存直到原来的字节被确认，以期收集更多的字节一起发送。

问题 3：傻瓜窗口症状

问题：发送方发送一个大报文，接收方每次只能读取一个字节，每读一个字节通知发送方可发送一个字节。

解决：Clark 算法：禁止**接收方发送 1 字节**的窗口修正。即接收方等待一段时间或者等到接收缓存已有一半空闲的空间。

- 问题 4：利用率问题

问题：发送一个报文段后等待应答(停止-等待)

对策：发送多个报文段后等待应答

- 问题 5：选择性问题

问题：发送多个报文后出现部分报文出错或者丢失

对策：利用 TCP 可选项允许使用选择性确认 SACK

- **7.2.12　TCP 的拥塞控制方法**

1. 拥塞控制的一般原理

- 在某段时间，若对网络中某资源的需求超过了该资源所能提供的可用部分，网

络的性能就要变坏——产生拥塞(congestion)。

- 出现资源拥塞的条件：

对资源需求的总和 > 可用资源

【CPU、缓冲区、线路等】

- 若网络中有许多资源同时产生拥塞，网络的性能就要明显变坏，整个网络的吞吐量将随输入负荷的增大而下降。
- 拥塞控制与流量控制的关系
- 拥塞控制所要做的都有一个前提，就是网络能够承受现有的网络负荷。
- 拥塞控制是一个全局性的过程，涉及到所有的主机、所有的路由器，以及与降低网络传输性能有关的所有因素。——通信子网
- 流量控制往往指在给定的发送端和接收端之间的点对点通信量的控制。
- 流量控制所要做的就是抑制发送端发送数据的速率，以便使接收端来得及接收。——一条路径
- 拥塞控制所起的作用
- 拥塞控制的一般原理
- 拥塞控制是很难设计的，因为它是一个**动态**的（而不是静态的）问题。
- 当前网络正朝着高速化的方向发展，这很容易出现**缓存不够大**而造成分组的丢失。但**分组的丢失**是网络发生拥塞的**征兆**而不是原因。
- 在许多情况下，甚至正是拥塞控制本身成为引起网络性能恶化甚至发生死锁的原因。这点应特别引起重视。
- 拥塞控制原理——开环与闭环原理
- **②闭环原理**
- 7.2.13　TCP 的拥塞控制方法
- 慢开始和拥塞避免
- 快重传和快恢复
- 随即早期检测（RED）
- 1.　慢开始和拥塞避免
- 拥塞窗口 cwnd (congestion window)：可能发生拥塞的数据量。大小取决于网络的拥塞程度，且动态变化。
- 发送窗口：可发送的数据量。
- 接收窗口 rwnd：可接收的数据量。
- 控制拥塞窗口的原则：只要网络没有出现拥塞，拥塞窗口就再增大一些，以便把更多的分组发送出去。但只要网络出现拥塞，拥塞窗口就减小一些，以减少注入到网络中的分组数。
- **发送窗口的上限值**
- 发送方的发送窗口的上限值应当取为接收方窗口 rwnd 和拥塞窗口 cwnd 这两个变量中较小的一个，即应按以下公式确定：
- **发送窗口的上限= Min{接收窗口，拥塞窗口}**
- Min {rwnd, cwnd}
- ■当 rwnd < cwnd 时，是接收方的接收能力限制发送窗口的最大值。
- ■当 cwnd < rwnd 时，则是网络的拥塞限制发送窗口的最大值。
- 慢开始算法的原理
- 在主机刚刚开始发送报文段时可先设置拥塞窗口 cwnd = 1，即设置为一个最大报文段 MSS 的数值。
- 在每收到一个对新的报文段的确认后，将拥塞窗口加 1，即增加一个 MSS 的数值。
- 用这样的方法逐步增大发送端的拥塞窗口 cwnd，可以使分组注入到网络的速率更加合理。
- 传输轮次 (transmission round)
- 使用慢开始算法后，每经过一个传输轮次，拥塞窗口 cwnd 就加倍。
- 一个传输轮次所经历的时间其实就是往返时间 RTT。
- “传输轮次”更加强调：把拥塞窗口 cwnd 所允许发送的报文段都连续发送出去，并收到了对已发送的最后一个字节的确认。
- 例如，拥塞窗口 cwnd = 4，这时的往返时间 RTT 就是发送方连续发送 4 个报文段，并收到这 4 个报文段的确认，总共经历的时间。
- 拥塞避免方法的原理
- 原因：拥塞窗口不可能一直加倍
- 时机：当拥塞窗口大于某个值时生效
- 动作：拥塞窗口按线性增长

- 目的：减少发往网络的数据
- 设置慢开始门限状态变量 ssthresh
- 慢开始门限（阈值） ssthresh 的用法如下：
- 当 cwnd＜ssthresh 时，使用**慢开始算法**。
- 当 cwnd＞ssthresh 时，停止使用慢开始算法而改用**拥塞避免算法**。
- 当 cwnd＝ssthresh 时，既可使用**慢开始算法**，也可使用**拥塞避免算法**。
- 拥塞避免算法的思路是让拥塞窗口 cwnd 缓慢地增大，即每经过一个往返时间 RTT 就把发送方的拥塞窗口 cwnd **加 1**，而不是**加倍**，使拥塞窗口 cwnd 按线性规律缓慢增长。
- 当网络出现拥塞时
- 发送方判断网络出现拥塞（即没有按时收到确认）：
  - 把慢开始门限 ssthresh 设置为出现拥塞时的拥塞窗口值的一半（但不能小于 2）。
  - 把拥塞窗口 cwnd 重新设置为 1，执行慢开始算法。
- 目的：迅速减少主机发送到网络中的分组数，使得发生拥塞的路由器有足够时间把队列中积压的分组处理完毕。
- **慢开始和拥塞避免算法的实现举例**
- **慢开始和拥塞避免算法的实现举例**
- **慢开始和拥塞避免算法的实现举例**
- **慢开始和拥塞避免算法的实现举例**
- **慢开始和拥塞避免算法的实现举例**
- **慢开始和拥塞避免算法的实现举例**
- **慢开始和拥塞避免算法的实现举例**
- **慢开始和拥塞避免算法的实现举例**
- **慢开始和拥塞避免算法的实现举例**
- **慢开始和拥塞避免算法的实现举例**
- **乘法减小(multiplicative decrease)**
- “**乘法减小**”是指不论在慢开始阶段还是拥塞避免阶段，只要出现一次超时（即出现一次网络拥塞），就把慢开始门限值 ssthresh 设置为当前的拥塞窗口值乘以 0.5。
- 当网络频繁出现拥塞时，ssthresh 值就下降得很快，以大大减少注入到网络中的分组数。
- **加法增大(additive increase)**
- “**加法增大**”是指执行**拥塞避免算法后**，当收到对所有报文段的确认就将拥塞窗口 cwnd 增加一个 MSS 大小，使拥塞窗口缓慢增大，以防止网络过早出现拥塞。
- **必须强调指出**
- “拥塞避免”并非指完全能够避免了拥塞。利用以上的措施要完全避免网络拥塞还是不可能的。
- “拥塞避免”是说在拥塞避免阶段把拥塞窗口控制为按线性规律增长，使网络比较不容易出现拥塞。
- **2.快重传和快恢复**
- 快重传算法首先要求接收方**每收到一个失序的报文段后就立即发出重复确认**。这样做可以让发送方及早知道有报文段没有到达接收方。
- 发送方只要一连收到**三个重复确认**就应当立即重传对方尚未收到的报文段。
- 不难看出，快重传并非取消重传计时器，而是在某些情况下可更早地重传丢失的报文段。
- 快重传举例
- **快恢复算法**

(1) 当发送端收到连续三个重复的 ACK 时，就重新设置慢开始门限 ssthresh。

(2) 与慢开始不同之处是拥塞窗口 **cwnd 不是设置为 1**，而是设置为 **ssthresh + 3 × MSS**。

(3) 若收到的重复的 ACK 为 n 个（$n>3$），则将 cwnd 设置为 ssthresh + n × MSS。

(4) 若发送窗口值还容许发送报文段，就按拥塞避免算法继续发送报文段。

(5) 若收到了确认**新**的报文段的 ACK，就将 cwnd 缩小到 ssthresh。

- 从连续收到三个重复的确认
  转入拥塞避免
- 3. 随机早期检测 RED (Random Early Detection)
- 使路由器的队列维持两个参数，即队列长度最小门限 TH_{min} 和最大门限 TH_{max}。
- RED 对每一个到达的数据报都先计算平均队列长度 L_{AV}。
- 若平均队列长度**小于**最小门限 TH_{min}，则将新到达的数据报放入队列进行**排队**。

- 若平均队列长度**超过**最大门限 TH_{max}，则将新到达的数据报**丢弃**。
- 若平均队列长度在最小门限 TH_{min} 和最大门限 TH_{max} **之间**，则按照某一**概率** ***p*** 将新到达的数据报丢弃。
- RED 将路由器的到达队列划分成为三个区域
- 丢弃概率 p 与 TH_{min} 和 TH_{max} 的关系
- 当 $L_{AV} < TH_{min}$ 时，丢弃概率 $p = 0$。
- 当 $L_{AV} > TH_{max}$ 时，丢弃概率 $p = 1$。
- 当 $TH_{min} < L_{AV} < TH_{max}$ 时，$0 < p < 1$ 。

例如，按线性规律变化，从 0 变到 p_{max}。

- 瞬时队列长度和平均队列长度的区别
- 7.2.14 无线 TCP

问题：

- 不稳定
- 性能低

对策：

- 端到端方案：区分拥塞和不稳定导致的丢失
- 分段连接方案：在基站分别连接两端
- 链路层方案：ARQ