今日干饭背诵（1.7）

Chap8 端到端访问与传输层

传输服务

【背诵】

1、**传输实体**

定义：完成传输层功能的硬软件

功能：利用网络层的服务向上层提供 ①有效且可靠 / ②best effort 的服务（分别对应TCP/UDP）

2、**传输服务**

分类 ——

- 面向连接：含 连接建立、数据传输、连接释放 三个阶段
- 无连接

相关方 ——

- 传输服务提供者：1~4层，应该是物理层、数据链路层、网络层、传输层?
- 传输服务用户：4层以上，应该是应用层?

3、**传输服务原语** 功能：传输用户（应用程序）通过传输服务原语访问传输服务

- 举例：一个C/S模式的原语可以有Listen、Connect、Send、Receive、Disconnect

4、**TPDU**: Transport PDU

TPDU：（一个非官方的理解）传输层的实体发给对等实体的信息单元

> 回忆：chap2
>
> - PDU是协议数据单元，PDU=上层SDU+PCI

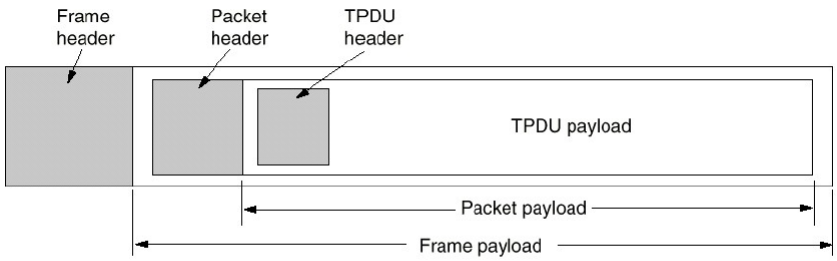


Fig. 6-4. Nesting of TPDUs, packets, and frames.

【概念】

引入传输层的原因：

- （优化点到点）消除网络层的不可靠性
- （实现端到端）提供端到端的可靠的、与实际使用的网络无关的数据传输

> 端到端：源端主机的**进程** -> 目的端主机的**进程**
>
> 【区分】网络层是点到点（主机之间），传输层是端到端（主机的进程之间）

简单连接管理

【概念】

拆除连接的方式：不对称、对称（P8有一个状态图举例，有空可以看看）

【其他】

一个例子：Berkeley Sockets

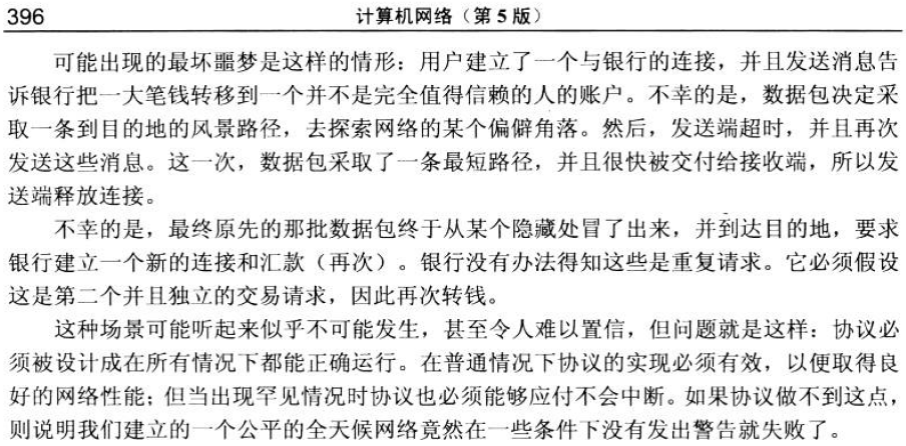
- 原语：Socket、Bind、Listen、Accept、Connect、Send、Receive、Close
- 数据传输 - 全双工；连接释放 - 对称
- 应用举例：P10-11

（复杂）连接管理：三次握手

【背诵】

传输层建立连接的复杂性原因：网络可能丢失、重复包，特别是**延迟重复包**的存在。

- 延迟重复包的个人理解：A发了包1给B，可能由于①包1迟迟没有到达B / ②B的ack1尚未到达A，使得A超时重发包2，这个时候包2就是延迟重复包。
- 《计算机网络》上的解释：



396 计算机网络（第5版）

可能出现的最坏噩梦是这样的情形：用户建立了一个与银行的连接，并且发送消息告诉银行把一大笔钱转移到一个并不是完全值得信赖的人的账户。不幸的是，数据包决定采取一条到目的地的风景路径，去探索网络的某个偏僻角落。然后，发送端超时，并且再次发送这些消息。这一次，数据包采取了一条最短路径，并且很快被交付给接收端，所以发送端释放连接。

不幸的是，最终原先的那批数据包终于从某个隐藏处冒了出来，并到达目的地，要求银行建立一个新的连接和汇款（再次）。银行没有办法得知这些是重复请求。它必须假设这是第二个并且独立的交易请求，因此再次转钱。

这种场景可能听起来似乎不可能发生，甚至令人难以置信，但问题就是这样：协议必须被设计成在所有情况下都能正确运行。在普通情况下协议的实现必须有效，以便取得良好的网络性能；但当出现罕见情况时协议也必须能够应付不会中断。如果协议做不到这点，则说明我们建立的一个公平的全天候网络竟然在一些条件下没有发出警告就失败了。

解决延迟重复包的关键：**丢弃**过时的包

【概念】

建立连接：三次握手

1. A发出序号为 X 的CR TPDU（Connection Request）
2. B发出序号为 **Y** 的CC TPDU（Connection Confirm） & 确认A的序号 X 的CR TPDU
3. A发出序号为 X+1 的第一个数据TPDU & 确认B的序号为 **Y** 的CR TPDU

> 【注意】P20的图 (a) 有错，课件P19指出DATA(seq=x, ACK=y)应该是seq=x+1
>
> 一些可以解决的情况举例：
>
> - 延迟重复CR：当B收到的是一个step1延迟重复包的时候，回送的TPDU含seq=y, ACK=x（step2指明对方身份），A可以发现这是延迟重复包然后REJECT，不进入step3
> - 延迟重复CR+ACK
> - CR：当B收到了一个step1延迟重复包的时候，回送的TPDU含seq=y, ACK=x_1，A可以从x_1序号发现是延迟重复包然后REJECT，不进入step3；

    - ACK：此后B又收到了step3延迟重复包seq=x_2, ACK=z，B可以从z序号发现这不是自己回送的最新seq=y，判定是延迟重复包，不进入step2。

释放连接：三次握手+定时器

- 原因：（DR = Disconnection Request）
    - 非对称式释放可能丢失数据（Data和DR交叉）
    - 对称式存在两军问题（你ack了我的ack）
- 以下不同的情况都能正常释放：（表明绝大多数时候是成功的）
    1. 正常的三次握手：
        - A发DR给B并启动timer
        - B回送DR给A并启动timer（A收到DR就释放）
        - A发送ACK确认B的DR（B收到ACK就释放）
    2. 最后的A发的ACK丢了，B的timer超时从而释放
    3. B回送的DR丢了，A的timer超时重发DR给B，直到某次B正常回送DR没有丢，归约到case 1 ~ 2
    4. B回送的DR丢了，A的timer超时重发DR也丢了，后续一直丢（没有正常），双方timeout N次后认为整个连接断开，双方都会断开并向上层报错

【其他】

缓存：

- 发送方：对每个连接单独缓存它发出的所有TPDU，以便错误时进行重传
- 接受方：可做可不做
- 设计：固定大小（按最大包长度，会有浪费） / 可变大小 / 缓存池（时间长了有碎片）

传输层寻址

【背诵】

TSAP：传输服务访问点

> 回忆：chap2
>
> - SAP是服务访问点，层间服务在接口SAP上进行
> - 因此TSAP在上层应用层和下层传输层之间，如下图

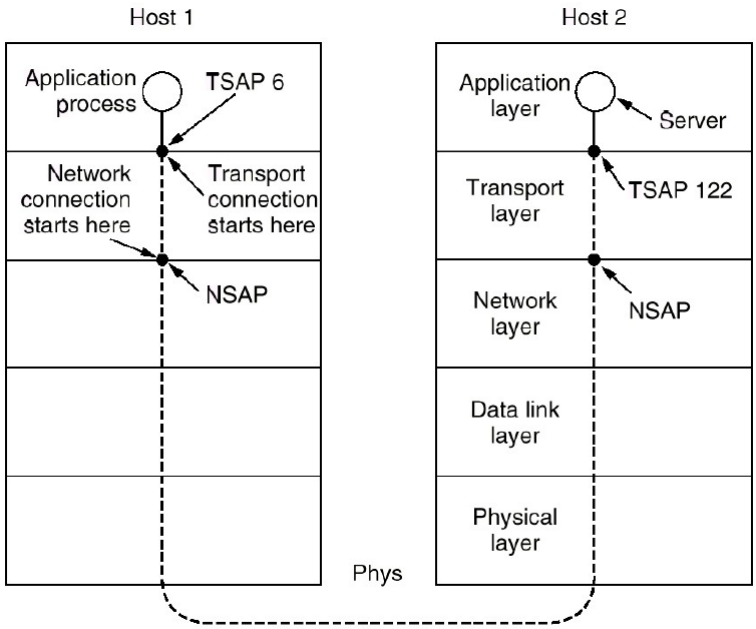


Fig. 6-8. TSAPs, NSAPs, and connections.

编址方法：将**应用进程**与**TSAP**相连；互联网中TSAP为 `(IP address, local port)`

远方client程序要想将自己的应用进程与server相连，需要**获得server程序的TSAP**，方法有：

1. **预先约定**，例如telnet是 `(IP address, port=23)`，FTP的port为21
2. 从**名称服务器**（name server）或目录服务器（directory server）获得
    - 名称 / 目录服务器（其TSAP是预先约定，众所周知的）工作过程：
        1. client进程与名称服务器建立连接，发送服务名称，获得服务进程的TSAP，释放与名称服务器的连接（类似DNS解析过程）
        2. client进程可以与服务进程建立连接了！
    - 进程服务器：当server程序很多时使用，同时在多个端口上监听

流控：TCP的窗口管理机制

【概念】

可变滑动窗口协议：传输层实现流控的方式。接收方根据自己的实际缓存情况，向发送方发出ACK，控制发送方的发送窗口（注意ACK应该周期性发出，避免一个TPDU丢失导致死锁）

【大题】

见P27，但是感觉滑动窗口更可能考网络层的，见2020网原课堂习题汇总。

互联网传输协议

TCP（Transmission Control Protocol）：面向连接、可靠、端到端、基于**字节流**

> 回忆：chap2数据链路层协议
>
> - HDLC、X.25 LAPB协议：面向比特
> - PPP协议：面向字符

UDP（User Data Protocol）：无连接、端到端、基于**消息流**

TCP

【背诵】

- 应用程序访问TCP服务：通过sender、receiver之间创建套接字
- 表示法：
  - 套接字地址用`(IP address, port)`表示，256以下的端口号被标准服务保留

    > 【回忆】TSAP也是这样的表示法

  - 一条连接用`(socket1, socket2)`表示，是点到点的全双工通道

    > 【注】因此TCP流的标识是个五元组，（src 地址, src 端口, dst 地址, dst 端口, 协议）

- 特点：
  - TCP**不支持组播、广播**
  - TCP基于字节流，消息的边界无法保留
  - 对于应用程序发来的数据，可以立即发送，也可以缓存一段时间再发（因为header比较长），可以用各种标记位标识数据发送要求（如PUSH、URGENT）
  - 使用滑动窗口协议，按byte分配序号
  - TPDU称为段（segment），段的大小要满足
    - 网络层：IP包对payload的长度限制（65536字节）
    - 数据链路层：最大传输单元（MTU）的限制（e.g. 以太网MTU为1500字节）

【概念】

- 解决的主要问题：
  - 连接管理
    - 建立连接：三次握手（P35）

      1. A->B: <seq=x=100>，<ctl=SYN>
      2. B->A: <seq=y=300>，<ack=x+1=101>，<ctl=SYN, ACK>
      3. A->B: <seq=x+1=101>，<ack=y+1=301>，<ctl=ACK>
      4. A->B：<seq=x+1=101>，<ack=y+1=301>，<ctl=ACK>，<data>（实际上4不是握手部分了，只是它和之前描述的三次握手模型略有差别，不同点在于data段留在下一次发了，不是ack&data一起)

      【注意】第三次握手是单独的确认段，之后发送的数据段序号和第三次握手相同

    - 释放连接：三次握手 + 定时器（**对称**方式释放，P39）
  - 可靠传输：滑动窗口
    - 【注意】窗口大小为0时有两个例外可以发送TCP段，是**URGENT数据** 和**防止死锁发送1字节的TCP段**
    - 改进传输层性能的策略（重要，见下）
  - 流控制（可变滑动窗口）和拥塞控制（慢启动+拥塞避免）（流控在前面，拥塞控制在后面）

UDP

【背诵】

应用范围：（对应UDP的特点）

- best effort的不可靠传输
    - 延迟小 —— 流媒体（可以容忍丢包，速率敏感）
    - 支持组播&广播 —— RIP（路由表周期性广播，而TCP不支持组播/广播）
    - 无连接 —— DNS（避免TCP连接建立延迟）
    - 无拥塞控制 —— SNMP（避免TCP在拥塞控制的时候把网管的包优化了，牛牛岂是你想吃就吃）
- 基于UDP的可靠传输：应用程序自己定义错误恢复

UDP头的帧格式：src/dst端口，UDP length，checksum。

TCP：改进传输层性能的各种策略

【概念】

1. 发送方缓存上层的数据，形成一个比较大的段再发
2. 在不允许捎带确认（ack和data一起发，变成滑动窗口大题中常见的Ixy，见2020网原课堂小测题）的时候，receiver延迟发送ack段
3. Nagle算法：（避免多次发送小包）
    1. 传输实体每收到上层（应用程序）发来的一个字节，就发出第一个字节并缓存所有其后的字节，直至收到对第一个字节的确认
    2. 然后将已缓存的所有字节组成一段发出，继续对以后收到的字节缓存，直至收到对这段的确认
4. Clarke算法：（解决傻窗口症状）
    - 傻窗口症状：当应用程序一次从传输层实体读出一个字节时，传输层实体会产生一个一字节的窗口更新段（即ack，这个传输层实体作为receiver会回送一个ack以改变sender的发送窗口），但这个时候被ack更新了的sender只能发送一个字节
    - 解决办法：限制收方只有在具备**一半的空缓存**或**最大段长的空缓存**时，才产生一个窗口更新段（ack）

TCP拥塞控制：慢启动和拥塞避免算法

【概念】

拥塞的两种情况：

1. 网络快，receiver缓存小 —— 只需要做流控，采用可变滑动窗口 / 设置静态的最大可接收窗口
2. 网络慢，receiver缓存够 —— 要做拥塞控制，采用拥塞窗口（按照慢启动+拥塞避免算法）

【注意】TCP发送的段长取二者的最小值

> 回忆：chap7网络层拥塞控制算法（拥塞控制主要在网络层和传输层TCP进行）
>
> - 拥塞控制基本方法
>   - 开环控制：提前设计
>   - 闭环控制：基于反馈，实时监测
> - 策略（预防-开环 => 反应-闭环）
>   - 网络供给：设备增加
>   - 流量感知的路由：提高利用率

    - 准入控制：降低负载，拒绝新连接的建立（开环控制）（广泛用于虚电路子网中的拥塞控制）
        - 流说明：通常采用的描述符是**漏桶 / 令牌桶**，用于流量整形，这样流说明中就包含了数据流平均速率和突发性这两个参数
    - 流量调节：参与源端的反馈循环，通知src减缓速度（闭环控制）
        - **抑制分组 / 逐跳抑制分组**是其中的两种方法，可用于数据报 & 虚电路子网
    - 负载丢弃：努力失败，只能丢弃（闭环控制）
        - **随机早期检测**（RED，Random Early Detection）是其中一种方法，为了确定何时开始丢弃数据包，路由器要维护一个运行队列长度的平均值。当某条链路上的平均队列长度超过某个阈值时，该链路就被认为即将拥塞，因此路由器随机丢弃一小部分数据包。随机丢弃的数据包使得快速sender发现丢包的可能性更大。（见《计算机网络》第5版，P325）

TCP拥塞控制：

- 维护一个拥塞窗口，窗口大小是任何时候sender可以往网络发送的字节数（但实际题目中通常是字节 / 最大段长，看题目随机应变好了）
- 【注意】TCP采用的是**AIMD**拥塞控制策略，公平性收敛&有效性收敛。

【大题】

慢启动和拥塞避免算法：非常重要，见2020网原课堂习题的例题

慢启动算法的基本过程：

1. 拥塞窗口大小初始化为TCP连接的最大发送段长MSS
2. 发出一个最大段长的TCP段，若得到确认，拥塞窗口大小+=一个最大段长
3. 每次发出 $\frac{\text{拥塞窗口大小}}{\text{最大段长}}$ 个最大段长的TCP段，凡有一个最大段长得到确认，拥塞窗口大小+=一个最大段长。

    > 【注意】存在累计确认的情况，一个ack帧可能确认了n个最大段长，那么拥塞窗口大小 += n个最大段长

4. 重复第3步，直到发生超时事件 或者 拥塞窗口大小>阈值

慢启动算法之后，进入拥塞避免算法，其基本过程为：（这个阶段还会发包，受拥塞窗口大小限制）

- 若上一步没有超时，而是因拥塞窗口大小>阈值而退出慢启动，则拥塞窗口开始线性增长，直到发生超时（Additive Increase）
    - 凡有 $\frac{\text{拥塞窗口大小}}{\text{最大段长}}$ 个最大段长得到确认，拥塞窗口大小+=一个最大段长
- 若发生超时，则将阈值设置为当前拥塞窗口大小的 $\frac{1}{2}$ ，拥塞窗口重新设置为一个最大段长，重新进入慢启动算法（Multiplicative Decrease）

伪代码如下：

```
// slow start
initialize: Congwin = 1
for (each segment ACKed)
    Congwin++
until (loss event OR CongWin >= threshold)

// slow start is over (if caused by Congwin > threshold)
```

```
// congestion avoidance
until (loss event) {
    for (every w segments ACKed)
        Congwin++
}
threshold = Congwin/2
Congwin = 1
goto slowstart
```

【概念】

检测丢包：发生了①计时器timeout / ②收到3个重复确认ack

快速重传算法：（个人认为就是把拥塞避免算法的until timeout替代为until ack×3）

- 连续收到**3个重复确认**后，不再等待计时器超时，而是
- 将阈值设置为当前拥塞窗口大小的一半，拥塞窗口重新设置为一个最大段长，进入慢启动算法

【其它】

TCP拥塞控制分析、系统模型和拥塞控制函数（MIAD、AIAD、MIMD、AIMD）：有空可以看看课件，感觉乱七八糟的