传输层

1. 传输层提供的服务和传输协议的要素

1.1. 传输层的功能

- 传输层提供应用进程之间逻辑通信（ 端到端**通信** ），网络层提供的是**主机到主机**的通信
- 复用和分用:
  - 复用：发送方**不同的应用进程**可以用**同一个传输层协议**传送数据。
  - 分用：接收方的传输层在剥去报文首部后能够把这些数据正确交付到应用进程。
- 传输层对收到的报文进行**差错检测**
- 提供两种不同的传输协议：**面向连接**的 TCP 和**无连接**的 UDP
- 传输层向高层屏蔽了低层网络核心的细节，如**网络拓扑，路由协议**等。它使得进程看见的是两个传输层实体之间的一条**端到端的逻辑通信信道**，信道对上层的表现因传输层协议的不同而不同。

1.2. 传输层的寻址与端口

1.2.1. 端口的作用

- 端口是传输层**服务访问点（TSAP）**，用于标识主机中的某个**应用进程**
- 其能够让**应用层**的各种应用进程将数据通过端口向下交付给**传输层**
- 也能够让**传输层**将报文段中的数据通过端口向上交付给**应用层**

> 数据链路层的SAP是**MAC地址**，网络层的SAP是**IP地址**，传输层的SAP是**端口号**

1.2.2. 端口号

- 应用程序通过**端口号**进行标识，长度为 $16bit$
- 端口号只具有**本地意义**，即端口号只标识本计算机应用层的进程。
- 端口号分类：
  - 服务器端:
    - 熟知端口号：0-1023，用于**TCP/IP最重要的一些应用程序**
    - 登记端口号：1024-49151，也叫注册端口号，用于**没有熟知端口号的应用程序**
  - 客户端:
    - 临时端口号：49152-65535，也叫动态端口号，用于**客户进程运行时才会动态选择**

1.2.3. 套接字 *

- IP**区分不同主机**，端口号**区分不同应用进程**，套接字**区分不同通信端点**。

$$ 套接字Socket=（IP地址:端口号）$$

（协议、源地址、目的地址、源端口号、目的端口号）唯一标识一次**传输层通信**

1.3. 无连接服务与面向连接服务

- 面向连接服务：通信前必须**建立连接**，通信过程中，整个连接的情况被实时**监控和管理**，通信结束后必须**释放连接**。

- **无连接服务**：需要通信时，直接将信息发送到网络中，让信息在网上**尽力而为地**向目的地传送。
- **TCP** 提供 **面向连接服务**，增加许多开销，诸如**确认、流量控制、计时器**及**连接管理**。适用**可靠性更高**的场合，如**FTP，HTTP**等。
- **UDP** 采用 **无连接服务**，在IP之上只提供**多路复用**和**错误检查**。执行速度更快，实时性好，如**TFTP（小文件传输协议），DNS，SNMP和RTP（实时传输协议）**。
- **IP数据报** 在网络层要经过路由器存储转发， **UDP数据报** 在传输层的端到端的逻辑信道传输，其封装成IP数据报在网络层传输时，UDP数据报的信息**对路由不可见**。
- **TCP报文段** 在传输层抽象逻辑信道传输，**对路由器不可见**；在传输层采用TCP不影响网络层提供**无连接服务**

1.4. 传输协议的要素

1.4.1. 寻址

传输服务访问点TSAP、网络服务访问点NSAP

- 远方客户程序如何获得服务程序的TSAP
  - 方法1：预先约定、广为人知的，象telnet是（IP地址，端口23）
  - 方法2：从名字服务器或目录服务器获得TSAP

1.4.2. 建立连接和释放连接

- 三次握手方案解决了由于网络层会丢失、存储和重复包带来的问题。
- 释放连接的两种方法：非对称释放（一方释放后整个连接就断开了）、对称释放

1.4.3. 差错控制

数据链路层的差错检测功能提供了点到点的链路传输的可靠，但无法保证端到端（源节点和目的节点之间）的传输的可靠性。

- 传输层实体必须缓存所有连接发出的TPDU，而且为每个连接单独做缓存，以便用于错误情况下的重传。接收方的传输层实体也可以不做缓存。
- 缓存区的设计有三种：链状固定大小缓冲区、链状可变大小缓冲区、循环缓冲区

1.4.4. 流量控制

传输层利用可变滑动窗口协议来实现流控，传输层的可变滑动窗口协议实现，通常是指发送的发送窗口大小是由接收方根据自己的实际缓存情况给出的。

1.4.5. 复用/分用

从多个应用进程获取数据, 用首部(便于随后的分用)封装数据

1.4.6. 崩溃恢复

从网络和路由器中恢复：如果网络层提供数据报服务，传输实体留有丢失的TPDU的副本；如果网络层提供面向连接的服务，重建新的虚电路，重发丢失的TPDU 即从第N层崩溃中恢复只能由第N+1层来完成，并且只能由第N+1层留有足够状态信息的情况下才能完成

2. UDP协议

2.1. UDP数据报

2.1.1. 概述

- UDP**是无连接**的**传输层协议**，它为应用层提供了一种**不可靠的数据传输服务**。
- 在**IP数据报**上增加最基本的服务：**复用和分用**，**差错检测**
- 优点
  - 无需建立连接，不会引入**建立连接时延**。
  - 无连接状态，不用维护连接状态，不用跟踪控制参数，能支持更多的客户机。
  - 分组首部开销小，只有**8B**。
  - 应用层能更好控制发送的**数据**和**发送时间**。UDP**没有拥塞控制**，故网络中的拥塞不会影响**主机发送效率**
  - 支持**一对一，一对多，多对一和多对多**的交互通信。
- UDP常用于一次性传输较少数据的网络应用，如 DNS，SNMP，以及**多媒体应用**。
- UDP不保证可靠交付，不意味着应用对数据要求是不可靠的。所有维护可靠性的工作可由用户在应用层完成。
- UDP是**面向报文**的，UDP对应用层交下来的报文，不进行任何拆分，也不会对报文进行合并，而是**直接将应用层交下来的报文封装**成UDP数据报，然后交给IP层。对于IP层交上来的UDP数据报，去除首部后，直接交给应用层。因此**报文**不可分割，是UDP数据报处理的**最小单位**。

2.1.2. UDP首部格式 *

- UDP首部**8B**，包括**源端口号**，**目的端口号**，**长度**，**校验和**。
  - 源端口：**2B**，源端口号，需要对方回信时选用，否则全0
  - 目的端口号：**2B**，目的端口号，终点交付报文时必须使用
  - 长度：**2B**，UDP首部和UDP数据的总长度，单位**字节**，最小值为**8B**，即仅有首部
  - 校验和：**2B**，检验UDP首部和UDP数据的正确性，如果为0，表示不检验
- 传输层将UDP数据报根据不同端口上交给不同应用进程，即 UDP分用
- 如果接收方UDP发现端口号不正确，则丢弃报文，并且由ICMP发送**差错报文**给发送方

2.2. UDP校验

- 在UDP数据报前添加 12B伪首部，得到临时数据报。**伪首部**不向上或向下传递，只是为了计算 校验和 。
- 源IP地址 **(4B)**，目的IP地址 **(4B)**，0 **(1B)**，UDP协议号17 **(1B)**，UDP长度 **(2B)**，共**12B**。
- 发送方把**全0**放入**校验和**并添加**伪首部**，然后把UDP视为**16位字串**的拼接。若数据部分不是偶数个字节，则末尾添加**1个全0字节**（只添加不发送）。
- 按 二进制反码 计算这些16位字串的和，将 **和的反码** 写入校验和字段并发送。
- 接收方把收到的UDP**加上伪首部**，按**二进制反码**求16位字串的和。若结果位**全1**，则表明无差错，否则**丢弃报文**。

3. TCP协议 *

3.1. TCP协议特点 *

- TCP**是面向连接**的**传输层协议**，其连接是**逻辑链接**，主要解决传输**可靠**，**有序**，**无丢失**和**不重复问题**。
- 每条TCP连接只能有**两个端点**，每一条TCP连接只能是**端到端**的（进程对进程）。

- TCP提供可靠交付的服务，保证传输的数据**无差错**，**不丢失**，**不重复**，且**按序**到达。
- TCP提供**全双工通信**，允许通信双方的应用进程在**任何时候**都能发送数据。所以TCP连接的两端都设有**发送缓存**和**接收缓存**，用来临时存放双方通信的数据。
    - **发送缓存**：
        - 发送应用程序传送给发送方TCP准备发送的数据
        - TCP已发送但尚未收到确认的数据
    - **接收缓存**：
        - 按序到达的数据，但尚未被接收应用程序读取的数据
        - 不按序到达的数据
- TCP是**面向字节流**的，其将应用程序传下来的数据视为一连串**无结构的字节流**
- UDP报文长度由发送的**应用进程**决定，TCP报文长度根据**接收方的窗口值**和**网络拥塞程度**决定。

3.2. TCP报文段 *

- TCP报文段既可以**运载数据**，也可以**建立连接**，**释放连接**和**应答**
- TCP报文字段：
    - **源端口和目的端口**：各**2B**，传输层和应用层的服务接口，**复用和分用**功能依靠端口号实现
    - **序号**：**4B**，**32位**的序号，用来标识本报文段所发送的数据的第一个字节的序号
    - **确认号**：**4B**，期望收到对方下一个报文段的第一个数据。若确认号为N，则表明到N-1为止的数据已经正确收到
    - **数据偏移**：**4位**，指TCP报文段数据起始处距离TCP报文起始处多远，单位是**4B**，即**32位**，最大值为**60B**，即**15个32位**，即**60B**
    - **保留位**：**6位**，保留位，目前没有使用
    - **紧急位URG**：**1位**，**紧急指针字段**是否有效，尽快传输紧急数据
    - **确认位ACK**：**1位**，**确认号字段**是否有效，TCP规定建立连接后，所有报文段都必须把ACK置1
    - **推送位PSH**：**1位**，接收方TCP是否**立即**将收到的数据交给应用程序，而不是等到缓存填满再向上交付。
    - **复位位RST**：**1位**，表示TCP连接出现严重差错，必须**释放连接**，然后**重新建立连接**
    - **同步位SYN**：**1位**，表示连接请求（ACK=0）或连接接受（ACK=1）报文段
    - **终止位FIN**：**1位**，表示**释放连接**。
    - **窗口**：**2B**，**16位**，允许对方发送的数据量。
    - **校验和**：**2B**，校验和字段检验范围包括**首部和数据**两部分。与UDP一样，要在TCP报文段前加上**12B的伪首部**
    - **紧急指针**：**2B**，只有在**URG=1**时才有意义，指出本报文段中紧急数据共多少字节。
    - **选项**：长度可变。TCP最初只规定了一种选项，即 **MSS最大报文段长度**，表示数据字段的最大长度。
    - **填充**：为了使整个首部长度是 **4B** 的整数倍

3.3. TCP连接管理 *

- TCP连接的建立和释放需要经过**三次握手**和**四次挥手**，**TCP连接是全双工的**，即**双方都可以同时发送数据**。
- TCP解决的三个问题：使双方确知对方存在；允许双方协商参数；能够对运输实体资源进行分配。
- TCP连接的端口是**套接字**，每条TCP连接都唯一地被通信的两个端点（套接字）确定。
- TCP连接的建立采用 **C/S** 模式，主动发起建立连接的应用进程称为 **客户端Client** ，被动等待建立连接的应用进程称为 **服务器Server** 。

3.3.1. TCP连接的建立 *

- **三次握手**：建立连接时，客户端和服务器端需要**发送SYN**，**SYN+ACK**，**ACK**三个报文段。
- **建立连接前**：服务器处于 `LISTEN`**收听**状态。
- **第一次握手**：客户端发送**连接请求**报文段，**SYN=1**，**Seq=x**（初始序号），**ACK=0**，客户端进入 `SYN_SENT`**同步已发送** 状态，等待服务器确认。
- **第二次握手**：服务器收到连接请求报文段后，如果同意建立连接，则会发送一个**连接确认报文段**，**SYN=1**，**ACK=1**，**Seq=y**，**ack(确认号)=x+1**，服务器进入 `SYN_RCVD`**同步收到** 状态。
- **第三次握手**：客户端收到连接确认报文段后，还要向服务器发送一个**确认报文段**，**ACK=1**，**Seq=x+1**，**ack=y+1**，客户端进入 `ESTABLISHED`**已建立** 状态，完成TCP连接的建立。

> 第二次握手**不能携带数据**，也要消耗一个序号；第三次握手如果**不携带数据则不消耗序号**。
>
> 服务器端的资源是**第二次握手**时分配的，客户端资源是**第三次握手**时分配的。服务器容易收到SYN洪泛攻击。

3.3.2. TCP连接的释放 *

- 参与TCP连接的两个进程中**任何一个**都能终止连接。

> 以下假设**客户端先挥手**。

- **四次挥手**：释放连接时，客户端和服务器端需要**发送FIN**，**ACK**，**FIN**，**ACK**四个报文段。
- **第一次挥手**：客户端向TCP发送**连接释放**报文段，**FIN=1**，**Seq=u**，客户端进入 `FIN_WAIT_1`**终止等待1** 状态，停止发送数据，但**对方可以继续发送数据**。
- **第二次挥手**：服务器收到**连接释放**报文段后，发送一个**确认报文段**，**ACK=1**，**Seq=v**，**ack=u+1**，服务器进入 `CLOSE_WAIT`**关闭等待** 状态，此时TCP连接处于**半关闭状态**。
- **期间**：客户端到服务器端的连接已经释放，**但服务器到客户端的连接还没有释放**，服务器发送的数据客户端仍需接收。
- **第三次挥手**：服务器向客户端发送**连接释放**报文段，**ACK=1**，**FIN=1**，**Seq=w**，**ack=u+1**，服务器进入 `LAST_ACK`**最后确认** 状态。
- **第四次挥手**：客户端收到**连接释放**报文段后，发送一个**确认报文段**，**ACK=1**，**Seq=u+1**，**ack=w+1**，客户端进入 `TIME_WAIT`**时间等待** 状态，等待可能出现的**对方的最后一个ACK**。在 `2MSL`**最长报文段寿命**后，客户端进入 `CLOSED`**关闭** 状态，完成TCP连接的释放。

3.4. TCP可靠传输

- TCP使用了**校验**，**序号**，**确认**和**重传**等机制来达到目的。其中TCP的**校验机制**与UDP校验一样。

3.4.1. 序号

- 序号建立在**字节流**的基础上，TCP首部的序号保证数据能**有序**交给应用层。

3.4.2. 确认

- TCP的确认机制是**接收方**向**发送方**发送确认信息，确认信息中包含了**期望收到的下一个字节的序号**。
- 发送方缓存区会存储**已发送未收到确认**的报文段，以便需要时**重传**。
- TCP默认使用 **累积确认**。即只确认数据流中**至第一个丢失字节为止**的字节。

3.4.3. 重传

- 两种事件会导致TCP重传： **超时** 和 **冗余ACK**。

- **超时**
  - TCP发送一个报文段后，对其设置**计时器**，如果到重传时间后还未收到确认，则重传这一报文段。
  - 由于传输层往返时延**方差很大**，TCP采用**自适应算法**。
  - **往返时间**`RTT`：一个报文发出和收到确认的**时间差**。
  - TCP随着测量 RTT 样本值的变化，根据加权平均算法，算出平均往返时延 RTT_{S}，又称**平滑往返时延**。

$$ RTT_{S-New} = \alpha RTT + (1-\alpha)RTT_{S-Old} $$

- **超时重传时间** RTO 应略大于 RTT_{S}。

$$ RTO = RTT_{S} + 4 \times RTT_{D} $$

- RTT_{D} 是 RTT 的偏差加权平均值，其与 RTT_{S} 和新的 RTT 样本之差有关。第一次测量时，RTT_{D} 为测量到的 RTT 样本值的**一半**。在以后的测量中，使用下面的公式计算 RTT_{D}。

$$ RTT_{D-New} = (1-\beta) RTT_{D-Old} + \beta |RTT_{S} - RTT| $$

- 一般 $\beta \lt 1$，推荐值是 0.25
-
- **冗余确认**
  - 每当**比期望序号大**的失序报文到达时，就发送一个 **冗余ACK**，指明**下一个期待字节**的序号。
  - 当发送方收到同一个报文段的 **三次冗余ACK**后，就可认为跟在这个被确认报文段后的报文段已经丢失，于是 **快速重传** 这个报文段。
- 设置超时值
- TCP确认的二义性

3.5. TCP流量控制 *

- 流量控制的目的是消除发送方和接收方之间的**数据传输速率不匹配**的问题。
- TCP采用基于 **滑动窗口协议** 的流量控制机制。
- 接收方根据自己接收缓存的大小，动态调整发送方的发送窗口大小，这称为**接收窗口**`rwnd`，单位**字节**。
- 发送方根据当前网络**拥塞程度**的估计而确定的窗口值，称为**拥塞窗口**`cwnd`，其大小与**网络带宽**和**时延**密切相关。

3.6. TCP拥塞控制 *

- **描述**：
  - 拥塞控制防止**过多数据注入网络**，保证网络中路由器或链路不致过载。
  - 拥塞控制让网络能够**承受现有网络负荷**，是**全局过程**，涉及所有主机和路由器。
  - 流量控制往往是指**点对点的通信量的控制**，是端到端的问题，它要做的是**抑制发送端发送数据的速率**，使得接收端能够及时地处理接收到的数据。
- **维护两个窗口**:
  - **接收窗口**`rwnd`：反应**接收方容量**，**接收方**根据TCP首部窗口字段通知发送方。
  - **拥塞窗口**`cwnd`：反应**网络当前容量**，**发送方**根据网络拥塞程度动态调整。
  - **发送窗口上限值**：上述两个之间最小的一个。

$$ 发送窗口上限值 = min(rwnd, cwnd) $$

- **四种算法**：**慢开始**，**拥塞避免**，**快重传**，**快恢复**。

3.6.1. 慢开始 *

- 令**cwnd=1**，即一个最大报文段长度MSS，每收到一个报文段的确认后，**将cwnd加一**。宏观上，每经过一个往返时间RTT，**cwnd就加倍**，直至增加到一个规定的**慢开始门限ssthresh（阈值）**，然后采用 **拥塞避免算法** 。
- **慢开始的慢** 指 **开始慢**，即开始设置**cwnd=1**。

3.6.2. 拥塞避免算法 *

- 每经过一个往返时延RTT就把发送方的拥塞窗口 **cwnd加一** 。使拥塞窗口cwnd按**线性规律缓慢增长**。
- 算法归纳如下：
  - $cwnd \lt ssthresh$，使用 **慢开始算法**
  - $cwnd \gt ssthresh$，停止使用慢开始算法而改用 **拥塞避免算法**
  - $cwnd=ssthresh$，既可以用**慢开始算法**，又可以用 **拥塞避免算法(通常做法)**

3.6.3. 网络拥塞处理 *

- 在上述两个阶段，如果发送方判断**网络出现拥塞**，即出现 **超时** 情况，则把慢开始门限 $ssthresh$ 设置为当前 $cwnd$ 的 **一半** （不能小于2）。然后将 $cwnd$ 置为 **1** ，并重新开始 **慢开始算法** 。
- 目的是**迅速减少**主机发送到网络中的分组数，使得拥塞路由器有足够时间把队列中积压的分组处理完。
- 拥塞避免**无法完全避免拥塞**，只能使网络不同意出现拥塞

3.6.4. 快重传 *

- 发送方连续收到 **三个冗余ACK** 后，不必等待超时重传定时器到期，就**立即重传**丢失的报文段。

3.6.5. 快恢复 *

- 发送方连续收到 **三个冗余ACK** 时，将 $ssthresh$ 设置为当前 $cwnd$ 的 **一半** ，并将 $cwnd$ 置为 $ssthresh$ **当前数值**，然后进入 **拥塞避免算法** 。
- 发送方预防网络发生拥塞，而由于可以收到连续的报文，故网络**没有发生严重拥塞**，所以**跳过了慢开始算法**，直接进入 **拥塞避免算法**，称为 **快恢复**。