概述

2019年7月10日 星期三 00:19

OSI参考模型（7层）

2019年7月10日 星期三　　22:45

	分层名称	功　能	每层功能概览
7	应用层	针对特定应用的协议。	针对每个应用的协议 电子邮件 ↔ 电子邮件协议 远程登录 ↔ 远程登录协议 文件传输 ↔ 文件传输协议
6	表示层	设备固有数据格式和网络标准数据格式的转换。	网络标准格式 接收不同表现形式的信息，如文字流、图像、声音等
5	会话层	通信管理。负责建立和断开通信连接（数据流动的逻辑通路）。 管理传输层以下的分层。	何时建立连接，何时断开连接以及保持多久的连接？
4	传输层	管理两个节点之间的数据传输。负责可靠传输（确保数据被可靠地传送到目标地址）。	是否有数据丢失？
3	网络层	地址管理与路由选择。	经过哪个路由传递到目标地址？
2	数据链路层	互连设备之间传送和识别数据帧。	0101 数据帧与比特流之间的转换 分段转发
1	物理层	以“0”、“1”代表电压的高低、灯光的闪灭。 界定连接器和网线的规格。	0101 → 0101 比特流与电子信号之间的切换 连接器与网线的规格

- 应用层
  - 处理将要发送的信息和接收到的信息
  - 以及应用层面的其他处理（例：邮件应用）
  - 首部：标明内容和收发地址
- 表示层
  - 从特定数据格式转换成网络通用的标准数据格式
  - 首部：包含编码格式
- 会话层
  - 决定采用何种连接方式，数据的分割等相关处理
  - 首部：数据传送顺序相关的信息
- 传输层

    - 在两个主机之间创建逻辑上的通信连接
        - 目标为最终地址
        - 只在通信起点终点进行处理，不在路由器上处理
    - 进行建立连接或断开连接的处理
    - 对数据的传输进行确认，重发的处理，保证数据传输的可靠性
- 网络层
    - 在网络相互连接的环境中，将数据从发送端发送到接收端
        - 目标可以是多个网络通过路由器连接而成的某个地址
        - 负责寻址和路由选择
    - 传输层和网络层的关系
        - TCPIP中，不是网络层而是传输层负责确保传输数据的正确性
        - 网络层与传呼层相互协作来确保数据能够可靠的传输
- 数据链路层
    - 通过传输介质在设备之间进行数据处理和传输
    - 网络层和数据链路层关系
        - 都是基于目标地址将数据发送出去
        - 但是网络层负责将整个数据发送给最终目标地址
        - 数据链路层只负责发送一个分段内的数据
- 物理层
    - 将数据的0和1转换为物理信号进行传输
    - 基于MAC地址（物理地址/硬件地址）进行传输

TCP/IP模型（4层）

2019年7月13日 星期六 22:39

- 网络接口层（包括HTTP DNS）
- 传输层（包括TCP UDP）
- 网络层（包括IP ARP）
- 应用层（包括MAC）

TCP/IP

2019年7月10日 星期三 22:44

TCP和UDP区别和使用场景

2019年7月13日 星期六 22:49

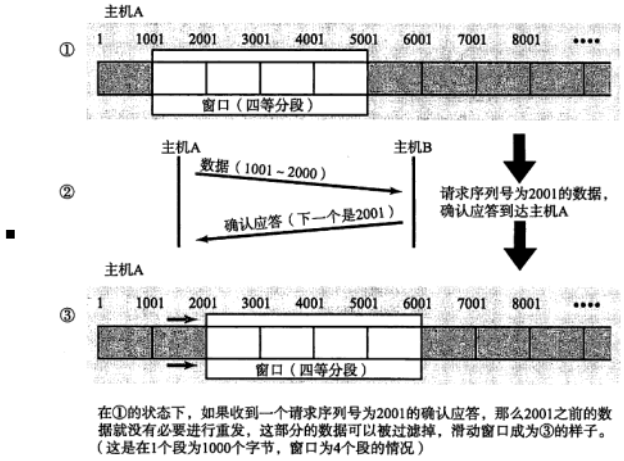
- 都是传输层协议，但具有不同特性
- 区别

	TCP	UDP
可靠性	可靠	不可靠
连接性	面向连接	无连接
有序性	有序	无序
传输速度	慢	块
拥塞控制	慢开始，拥塞避免，快重传，快恢复	无
流量控制	滑动窗口	无

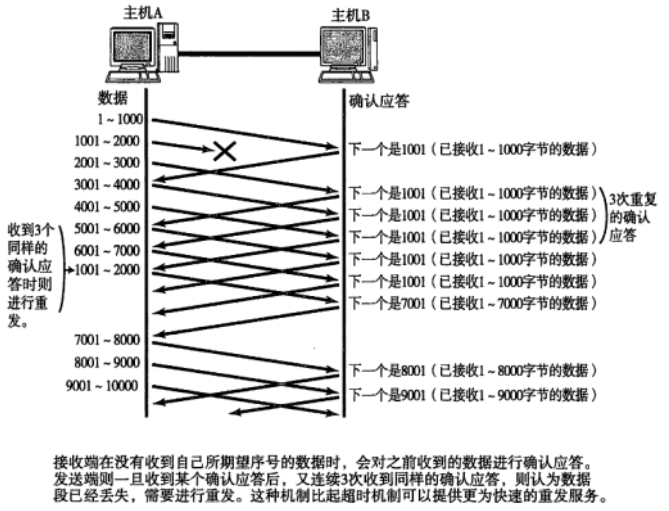
- 什么时候用TCP
    - 对网络通讯质量有高要求的时候
- 什么时候用UDP
    - 对网络通讯质量要求不高，但对速度要求高

*TCP是如何保证可靠性的

2019年7月15日 星期一 12:39

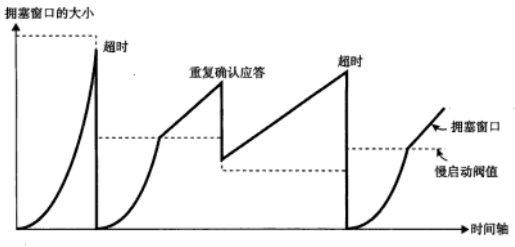
- 序列号，确认应答，超时重传
  - 数据到达服务器后，服务器端需要发出一个确认应答（ACK），表示已经收到该数据段，并且确认序号会说明它下一次需要接收的数据序列号
  - 如果发送迟迟未收到确认应答，那么可能是发送的数据丢失，也可能是确认应答丢失，这时客户端会在等待一段时间后进行重传
- 窗口控制
  - 目的：提高传输速度
  - 做了什么
    - 在一个窗口大小内，不用一定要等到应答才能发送下一段数据
    - 窗口大小就是无需确认就可以继续发送的数据大小的最大值
  - 如果不使用窗口控制，每一个没收到确认应答的数据都要重发
  - 滑动窗口
    - 

在①的状态下，如果收到一个请求序列号为2001的确认应答，那么2001之前的数据就没有必要进行重发，这部分的数据可以被过滤掉，滑动窗口成为③的样子。（这是在1个段为1000个字节，窗口为4个段的情况）
    - 收到确认应答的情况下，将窗口华东到确认应答中的序列号的位置
    - 这样可以顺序地将多个段同时发送提高速度
- 高速重发控制

- 

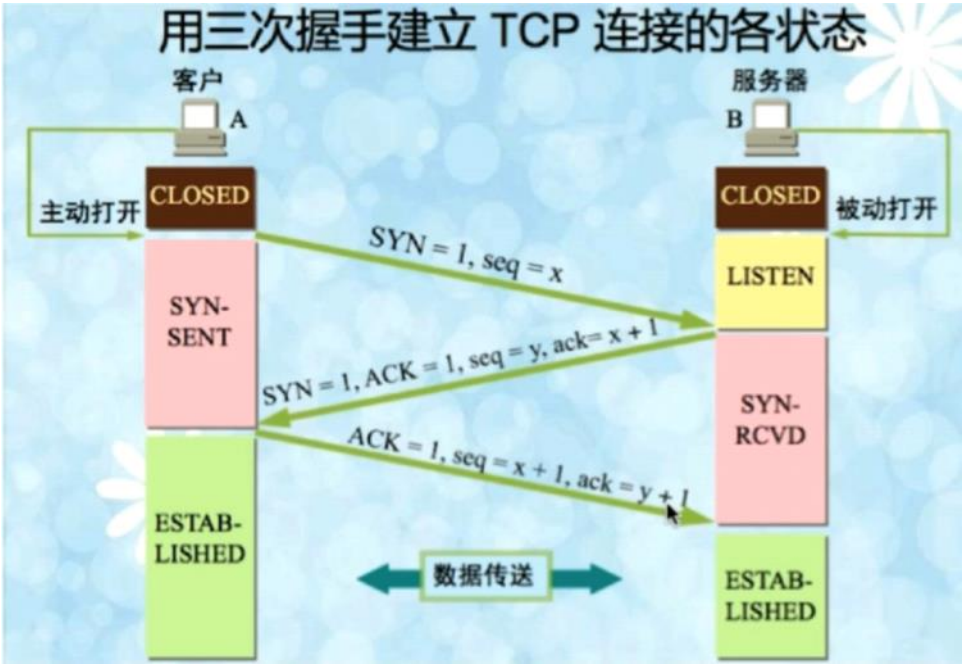
  - 如图所示
  - 当发送数据的某一段（如图中1001-2000）报文丢失后，发送端会持续收到序号为1001的确认应答（服务器端想要客户端给它重发这一段）
  - 所以，在窗口比较大，又出现这样的报文段丢失的情况下，同一个序号的确认应答会被不断重复的发送
  - 而客户端如果连续3次收到同一个确认应答，就会立即发送该确认应答所对应的数据
  - 比超时重传更高效
- 拥塞控制
  - 为什么需要拥塞控制
    - 当通信刚开始的时候客户端就发送一个大的数据，又可能导致网络的瘫痪
  - 什么是拥塞控制
    - 通过慢启动算法，计算拥塞窗口的大小，对发送数据量进行控制
    - 慢启动算法
      - 拥塞窗口定义函数，一般初始值设为一倍数据段（1MSS）
      - 之后每收到一次确认应答，拥塞窗口扩大2倍，呈指数增长
      - 但为了避免窗口过大导致网络瘫痪，设置一个阈值，超过这个阈值窗口就不再指数上升，而是加法增加
      - 如果发生超时重传，阈值就会变为当前窗口大小的一半，窗口大小初始化为1MSS，重新进入慢启动过程
      - 如果发生高速重发控制，阈值会变为当前窗口大小的一半，窗口大小设置为该阈值+3MSS

□


拥塞窗口的大小
超时
重复确认应答
超时
拥塞窗口
慢启动阀值
时间轴

*三次握手

2019年7月13日 星期六　　22:59

- 什么是三次握手
  - TCP是面向连接的，所以无论哪一方向另一方发送数据之前，都必须先在双方之间建立一条连接
  - TCP协议就是通过三次握手进行连接的初始化的
  - 目的
    - 同步连接双方的序列号和确认号
    - 交换TCP窗口大小信息
- 三次握手具体怎么做的
  - 

  - 第一次握手
    - 客户端发送连接请求报文段，SYN为1，SEQ设为x
    - 然后客户端进入SYN_SEND状态，等待服务器端的确认
  - 第二次握手
    - 服务器收到客户端发来的SYN报文段，需要对这个报文段进行确认
    - 设置ACK为x+1，将自己的那份SYN请求信息的SYN值设为1，SEQ设为y
    - 服务器将上述所有信息放到一个SYN+ACK报文段，发给客户端
    - 服务器进入SYN_RECV状态
  - 第三次握手
    - 客户端收到SYN+ACK报文段
    - 将ACK设为y+1，SEQ为z，并把这个报文段发给服务器
    - 完成后，客户端和服务器都进入ESTABLISHED状态，三次握手完成
- 为什么要三次握手
  - 三次握手都干了
    - 客户端向服务器端发送连接请求

        - 服务器端对收到的客户端的报文段进行确认
        - 客户端再次对服务器端的确认进行确认
    - 原因：为了防止失效的连接请求报文段突然又传送到服务器端，从而产生错误
        - 失效的连接请求报文段
            - 客户端发出的连接请求没有收到服务器端的确认
            - 于是经过一段时间后，客户端又重新向服务器端发送连接请求
            - 这一次建立成功，完成连接传输
        - 为什么会产生错误
            - 第一次客户端发送的连接请求其实没有丢失，而是因为网络节点延迟导致到达服务器端慢了
            - 于是服务器以为客户端又发起了新连接，并同意了这个迟到的连接请求，向客户端发回确认
            - 但客户端因为通过重发的连接请求完成了连接，根本不理会
            - 服务器端就一直在等待客户端确认，导致服务器端资源浪费

SYN FLOOD攻击原理

2019年7月14日 星期日 23:30

- 利用TCP协议缺陷
- 发送大量伪造的TCP连接请求，从而使被攻击方资源耗尽（CPU过载）
- 具体过程
  - 在第一次握手中客户端若是发送SYN报文后突然掉线，服务器会重发SYN+ACK报文并等待一段时间，这段时间成为SYN Timeout（一般在2分钟以内）
  - 就是攻击者大量模拟这种情况，服务器为了维护一个非常大的半连接列表而消耗过多资源
  - 导致栈溢出崩溃，或者正常用户的请求无法得到响应
- 解决方法
  - 缩短SYN Timeout时间
  - 设置Cookie
    - 若短时间内收到来自某一IP的重复SYN报文，则之后的报文会被丢弃

*四次挥手

2019年7月14日 星期日 23:36

- 挥手的目的：终止连接
- 具体怎么做的
  - 

  - 第一次挥手
    - 客户端发送一个FIN，SEQ=u到服务器
    - 只是用来声明我们要关闭客户端到服务器端的数据传送了，从服务器端到客户端的数据传输依然可用
  - 第二次h挥手
    - 服务器端在收到FIN之后，发送一个ACK=u+1的包给对方
      - 此处有等待
  - 第三次挥手
    - 服务器端发送一个FIN，且SEQ=r用来关闭服务器端到客户端的数据传送
    - 也就是告诉客户端，我的数据发完了，以后不会再给你发了
      - 此处有等待
  - 第四次挥手
    - 客户端收到FIN后，发送一个ACK=r+1的包到服务器端
    - 四次挥手完成
- 为什么挥手有四次
  - 因为TCP的连接是全双工模式的，也就是说
    - 当一方发送FIN报文后，只是代表这一方已经没有数据要发送了，但依然还可以接收数据
    - 另一方回复ACK报文的意思也只是我知道你不会再发给我数据了，但依然可以接收数据
  - 所以一个方向的连接需要2次挥手，双向就是4次
- 几种状态的解释

    - FIN_WAIT1
        - 客户端发送了FIN报文后进入，表示等待对方回应
    - FIN_WAIT2
        - 半关闭状态
            - 客户端还有接收数据的能力，但没有了发送数据的能力
    - CLOSE_WAIT
        - 一般服务器端收到FIN包后会立即回复ACK包表示已收到
        - 但如果服务器端还有剩余数据要发送就会进入CLOSE_WAIT状态用于发送剩余数据
    - TIME_WAIT
        - 客户端发送最后一个ACK后，等待一段时间关闭，这段时间就是TIME_WAIT
        - 一般持续2倍的MSL（报文生存最大周期），所以又称为2MSL
        - 为什么需要这一状态
            - 因为这时候不确定ACK报文服务器端是都能收的到
            - 保持连接的话如果对方没收到还会重发ACK
- 服务器出现大量CLOSE_WAIT
    - 原因
        - 客户端发送请求关闭，服务器端还有很多没有发送完，没有及时关闭连接

HTTP

2019年7月15日 星期一 15:17

HTTP基础

2019年7月10日 星期三 22:44

- HTTP（超文本传输协议）
  - 传输数据到本地浏览器的协议
  - 基于应用层的面向对象的协议，工作在应用层，在传输层中由TCP协议负责为其提供服务
  - 端口号80
  - 浏览器根据从服务器得到的HTTP响应体中得到报文头，响应头和信息体（HTML正文等），之后将HTML文件解析并呈现在浏览器上
  - 浏览器作为HTTP客户端通过URL向HTTP服务器端（WEB服务器）发送所有请求，WEB服务器根据接收到的请求，向客户端发送响应信息
- HTTP特点
  - 简单快速
    - 客户端向服务器请求服务时，只需要传送请求方法和路径
    - 常用请求方法
      - GET，HEAD，POST
      - GET，POST区别
        - 报文层面
          - GET请求信息放在URL
          - POST放在报文体
        - 数据库层面
          - GET符合安全性
        - 其他层面
          - GET可以被缓存
          - POST不可以
      - 每种方法中客户端与服务器联系的类型不同
    - 使得HTTP服务器程序规模小，通信速度快
  - 灵活
    - HTTP允许传输任意类型的数据对象
    - 类型由Content-Type加以标记
  - 无连接
    - 指的是 限制每次连接只处理一个请求
    - 服务器处理完客户的请求，并受到客户的应答后，即断开连接
    - 节省传输时间
  - 无状态
    - HTTP是无状态协议
    - 无状态指的是 协议对于事务处理没有记忆能力
    - 缺少状态意味着如果后续处理需要前面的信息，就必须重传，这样就可能导致每次连接传送的数据量增大

        - 另一方面，在服务器不需要先前信息时它的应答就会比较快
    - 支持B/S和C/S
    - 基于TCP协议

HTTP工作流程

2019年7月15日 星期一 15:17

- HTTP工作流程
  - 预备知识
    - HTTP协议定义了客户端如何从服务器请求页面，以及服务器如何把页面传送给客户端
    - HTTP采用请求/响应模型
      - 客户端向服务器发送一个请求报文，请求报文包含
        - 请求的方法
        - URL
        - 协议版本
        - 请求头部和请求数据
      - 服务器以一个状态行作为响应，响应的内容包括
        - 协议的版本
        - 成功或错误代码
        - 服务器信息
        - 响应头部和响应数据

①. 客户端连接到服务器
   - 一个HTTP客户端（通常是浏览器），与服务器的HTTP端口（80）建立一个TCP套接字连接

②. 发送HTTP请求
   - 通过TCP套接字，客户端向服务器发送一个文本的请求报文
   - 请求报文包含：请求行，请求头部，空行和请求数据4部分

③. 服务器接受请求并返回HTTP响应
   - 服务器解析请求，定位请求资源
   - 服务器将资源副本写到TCP套接字，由客户端读取
   - 响应包含：状态行，响应头部，空行，响应数据4部分

④. 断开连接

⑤. 客户端浏览器解析HTML内容

在浏览器输入网址后的执行过程

2019年7月15日 星期一 15:23

①. 浏览器向DNS服务器请求解析该URL中的域名所对应的IP地址
②. 解析出IP地址后，根据IP地址和端口80，和服务器建立TCP连接
③. 浏览器发出读取文件（URL中域名后面部分对应的文件）的HTTP请求，该请求报文作为TCP三次握手的第三个报文的数据发送给服务器
④. 服务器对浏览器请求做出响应，并把对应的HTML文本发送给浏览器
⑤. 断开TCP连接
⑥. 浏览器将HTML文本解析并显示内容

HTTPS

2019年7月16日 星期二 22:22

- 具体工作流程省略
- HTTPS特点
  - 加密传输数据确保安全型
- 优点
  - 相比http更加安全
  - 需要认证用户和服务器，确保数据发送准确
- 缺点
  - 成本高
    - 需要购买证书
    - 需要的服务器配置高
  - SSL握手阶段延时高（因为平白无故多加了一个SSL握手）

HTTP和HTTPS的区别

2019年7月16日 星期二 22:17

- 区别

HTTP	HTTPS
传输数据无加密	加密后的（更安全）
不需要	在TCP三次握手之后，还需要进行SSL握手
不需要	需要服务器端申请证书，浏览器端安装对应证书
协议端口80	协议端口443

- 区别总结
    - HTTPS是HTTP的安全版本（主要通过SSL来实现安全性）