计算机网络

1. 基础知识

路由器、交换机

交换机是目前局域网内主机通信的解决方案。
它能够进行MAC地址学习，记录在其MAC表中。主机通信时只将消息发送给对应主机。
在交换机之前的同轴电缆或者集线器等都是广播，然后丢包的形式

路由器用来在不同的网段之间发送消息。首先会发送至网关，网关地址一般在路由器上。然后路由器根据路由表进行通信。
路由表中记录了静态路由（手动配置下一跳）、直连设备（直接连在路由器上的网段）、动态路由（通过路由选择协议自动获取路由信息）

静态路由、动态路由

静态路由：
管理员手动添加路由信息，适用于小规模网络
动态路由：
路由器通过路由选择协议（比如 RIP、OSPF）自动获取路由信息，适用于大规模网络。
它的原理是路由器之间根据动态协议适时地交换路由信息

局域网、以太网、无线局域网

局域网：
一般是范围在几百米到十几公里内的计算机所构成的计算机网络。
常用于公司、家庭、学校、医院、机关、一幢大楼等
以太网：
以太网是局域网中使用最广泛的网络技术。
使用了 CSMA/CD（载波侦听、多路访问、冲突检测） 的网络可以称为是以太网，它传输的是以太网帧。
制定了一些规范如物理层的连线、接口等等
无线局域网：
无线局域网对比局域网。局域网是使用物理连线将各设备连接起来。而无线局域网则是使用无线通信技术。其他基本类似。

DNS、CDN、VPN、NAT

DNS：域名系统。它的主要作用就是将域名解析为IP地址。
域名查找大部分工作是根据自己配置的DNS服务器（即本地DNS服务器，这个本地是相对的概念，意为离你最近的DNS服务器）
比如：缓存、从根DNS服务器开始，一级一级查找最终的域名IP，都是通过本地DNS服务器完成
CDN：利用最靠近用户的服务器，最快地将资源传递给用户。
比如很多JS库都是使用了CDN的。当国内用户和国外用户都加载同一个域名的CDN时。实际上提供资源的服务器并不是同一个，CDN运营商再各大城市部署了机房，付费后即可享受其服务。
VPN：在公共网络上建立专用网络，进行加密通讯。
它的原理就是在VPN客户端和VPN服务端之间进行加密通信。用户先安装VPN客户端，使用客户端访问公司内网，先到达公司部署的VPN服务器，服务器解密后再传输给内网地址。
NAT：私网 IP 访问 Internet 需要进行 NAT 转换为公网 IP， 这一步可以由路由器来完成。
它有三种方式：静态转换、动态转换。这两种转换均为一对一。
使用最广泛的是PAT，采用端口多路复用，路由器通过端口号标记不同主机。可实现多个私网IP对一个公网IP

- MAC、IPv4、IPv6、端口

  MAC：网卡带有，共6字节，前3字节为OUI（组织唯一标识符），由IEEE分配给产商。后3字节为网络接口标识符，由厂商自行分配。
  当 48 位全为 1 时，即 FF-FF-FF-FF-FF-FF，代表广播地址
  IPv4：平时称的IP，如果不特殊说明，一般就是指IPv4。
  IP由网络标识（网络ID）和主机标识（主机ID）两部分构成。
  子网掩码二进制为1的位，即代表网络ID位数。更加科学的方式是：子网掩码 &（按位与） IP 地址。网络ID相同的主机即认为在同一网段。
  主机ID为0，代表主机所在网段、主机ID为1，代表广播
  A类地址（0开头）：1-126
  B类地址（10开头）：128~191.0~255
  C类地址（110开头）：192~223.0~255.0~255
  D类地址(1110开头)：用于多播（组播）地址。无子网掩码、第一部分224~239
  E类地址(1111开头)：保留为今后使用，第 1 部分240~255
  IPv6：IPv6采用十六进制表示法，区别于IPv4的十进制表示。共16字节，分8组。如
  2001:0db8:86a3:08d3:1319:8a2e:0370:7344
  工作在网络层，传输时首部格式相对IPV4简洁。
  功能上新增的头部：Flow Label，表示数据包属于哪个流数据。Next Header（8bit）：下一个头部，用于扩展头部。
  端口：可认为是计算机与外界通讯交流的出口。不同的协议有不同的默认端口。比如熟知的HTTP（80）、HTTPS（443）、FTP（21）、SMTP服务（53） - 用于发送协议
  发现只有应用层协议有默认端口
  当客户端发送数据给服务端时，默认也是会有一个随机端口的。传输层不存储URL（比如TCP，不同URL通过TCP先后发送给服务器时，数据可能是混在一起传输的），是根据客户端的随机端口映射对应的请求的。

- 子网划分、子网掩码、超网

  子网掩码：
  子网掩码是一个32位的2进制数， 其对应网络地址的所有位都置为1，对应于主机地址的所有位置都为0。
  子网掩码必须结合IP地址一起使用，它的主要功能就是将某个IP地址划分成网络地址和主机地址两部分
  子网划分：
  为什么要进行子网划分?
  主要目的是为了合理利用资源。试想下B类地址一个网段能够部署65534台主机，而C类地址只能部署255台。如果要部署500台，没有子网划分，只能使用B类地址，并且存在大量浪费。
  概念：借用主机位作子网位，划分出多个子网
  借用的主机位实际上变成了网络地址，结果就是网络地址位增多、增加的几位就是分配给不同子网段的。 主机位减少，所以子网段可分配的ip地址减少。
  超网
  概念：跟子网反过来，它是将多个连续的网段合并成一个更大的网段。重点连续的，不连续是不可以的。
  将子网掩码向左移动，即可实现超网，网络 ID 位数减小，主机 ID 位数增多。可分配的 IP 地址就会增加。

- 2. 参考模型

- 7层（OSI）

  由国际标准化组织ISO组织定义，为网络模型标准，偏理论。
  7层分别为物理层、数据链路层、网络层、传输层、会话层、表示层和应用层
  其中会话层、表示层和应用层常被其他模型合并为一层，称为应用层。这里介绍下会话层、表示层和应用层之间的区别，其他几层在五层模型中介绍
  会话层的主要功能是负责维护两个节点之间的传输联接。即管理会话（建立会话、终止会话、会话采用的通信方式（全双工or半双工）等等）。
  表示层：应用程序之间传输信息的表示方法（如传输格式和语法、数据加解密，数据压缩等）
  应用层直接为应用程序提供服务。它的作用第一实现多个系统应用进程相互通信、第二完成业务处理服务。
  可以看到会话建立、传输信息以及业务处理可以认为都是应用的概念。都需要研发人员应对不同场景进行设计和实现。所以TCP/IP模型才会将他们合并为一层

- 4层（TCP/IP四层模型）

  TCP/IP四层模型是由世界各个网络编程人员、软件开发人员实践后得出的
  它包括 网络接口层（物理层和数据链路层） 、网络层、传输层、应用层

- 5层（TCP/IP五层模型）

  - 应用层、报文

    概念：
    应用层顾名思义是与应用程序直接进行交互的。

    它所做的概括起来就是两点：
    将应用程序的信息进行加工(产物为“报文段”)，并传递给传输层
    将传输层传过来的信息进行加工(产物还是“报文段”)，并传递给应用程序

    协议：
    其主要协议包含：
    HTTP、HTTPS、FTP、SMTP、DNS、DHCP等

  - 传输层、段

    概念：
    网络层只是把分组（不清楚话的可以说信息）发送到目的主机，但是真正的通信并不是在主机而是主机中的进程。

    传输层提供了进程间的逻辑通信。

    协议：
    传输层有两个协议：UDP和TCP，这两个后面详细介绍

    补充：
    一个端口只能被一个进程占用，一个进程可以占用多个端口；一个端口就是一个数据通道

- 网络层、包

  概念：网络层的目的是实现两个端系统之间的数据透明传送。端系统即主机。它的基础功能就是寻址和路由选择（路由器的路线选择）。
  说白了，网络层识别IP地址，能够将信息送到正确的主机。与之对应的传输层，它则根据端口号将数据传输到对应的应用程序（或者说进程）上。

  网络层最终的协议就是IP协议，所以需要详细分析IP协议
  总体分为首部+数据
  数据很多时候是由传输层传递下来的数据段
  所以重点分析下首部
  包含：版本（V4/V6）、首部长度、区分服务、总长度、标识（区分是否为同一个数据包，在数据包分片时应用）、标识（Flags，包含允许分片、是否还有片）、片偏移（该片是从数据包的哪里开始的）、TTL（路由器跳数）、协议、首部校验和、

- 数据链路层、帧

  概念：数据链路层定义了在单个链路上如何传输数据。主要作用是负责解决两个直接相邻节点之间的通信
  链路： 两个相邻节点间的物理线路，中间没有其他交换节点。比方说路由器和直连它的交换机之间的连线就是一段链路。
  数据链路：在一条链路上传输数据时，需要有对应的通信协议来控制数据的传输（如路由器和路由器之间的链路一般是通过PPP协议传输。交换机和主机之间一般是CSMA/CD）
  其传输的基本单位为帧
  数据链路层3个主要工作：封装成帧、透明传输、差错校验
  封装成帧：将比特流组合成以太帧进行传送。
  帧的格式：帧首部（含帧开始符-SOH） | 帧的数据部分 | 帧尾部（含帧结束符-EOT）
  帧的数据部分一般为网络层传递下来的IP数据包，但如果直接以数据层协议传输，则是由数据链路层拼装而成。它的大小不可大于MTU（以太网帧的MTU为1500字节）

  透明传输：数据部分一旦出现 SOH、EOH，就需要进行转义。否则被误当做开始或结束帧，数据就不对了。
  之所以称为透明传输就是因为你发送的原始数据和接收到的数据之间可能在协议内转义了，这并不被外部感知。

  差错校验：检验传输数据的完整性和正确性。
  帧校验序列字段（FCS）：根据帧的数据部分和帧首部（去除帧开始符）计算得出
  过程：
  帧尾部包含 FCS。帧发送时会先计算得到 FCS，一起传输给接收方。
  接收方接到数据后，也会计算一遍 FCS，如果两者一致，代表数据传输 ok，如果不一致网卡就会丢弃。
  数据链路层2个主要协议
  CSMA/CD协议
  该协议全称翻译为 载波侦听、多路访问/冲突检测
  载波侦听：侦听信道上是否有信号正在传输
  多路访问：就是连接着的设备都可以发送信号
  冲突检测: 如果信号传输过程中产生了冲突,则会被弹回。冲突检测就是检测接收的信号是来自其他设备，还是自身发出信号的回弹。
  使用了 CSMA/CD 的网络可以称为是以太网（Ethernet），它传输的是以太网帧
  PPP（点对点协议）
  它和以太网帧不同，点对点信道不需要源MAC和目标MAC
  以太网帧使用最广泛的格式为Ethernet V2标准，它的首部包含目标MAC、源MAC和网络类型。
  以太网帧的数据长度是有大小限制的，最大是因为数据部分限制MTU（1500）。最小帧规定为64字节，可根据首部和尾部计算出数据部分最小46

- 物理层、比特流

  概念：物理层定义了接口标准、线缆标准、传输速率、传输方式等，基本单位为比特流。
  可以用两个词概括这一层：信号和介质。因为物理层主要关心如何传输信号，而信号的传输离不开传输介质
  信号有数字信号、模拟信号、光信号等
  介质：可以是网线、集线器/交换机、电话线、光纤等等。这些介质中如果传输的信号类型不一样，比如需要将网线传输的信号输送到电话线传输，就需要先由调制解调器将数字信号转为模拟信号。

- 总结：各层作用串联

▼ 5. 其他协议

- ARP、RARP、DHCP、ICMP

  ARP：是根据IP地址获取MAC地址的一个TCP/IP协议
  通过广播的方式获取目标IP的MAC地址
  ARP欺骗：正常情况下，主机接收到ARP协议消息，发现消息中携带的目标IP不是自己，则会丢包。ARP欺骗时发现目标IP不是自己，依然返回响应给源主机，并且错误告知目标IP的MAC地址。
  RARP：逆地址解析协议
  与ARP作用相反，根据MAC地址获取IP地址
  目前已被DHCP所取代
  DHCP
  DCHP是用来获取动态IP的。根据MAC地址分配IP
  一台新的主机部署好后，动态获取IP就是通过DHCP协议像DHCP服务器请求，DHCP服务器会从IP地址池中挑选可用的ip，出租给这台主机。
  现在的路由器基本上充当了DHCP服务器的角色
  ICMP
  ICMP的主要用于在主机与路由器之间传递控制信息。比如目的是否可达、TTL值是否过期等等
  当我们PING一台主机时，就是使用ICMP协议
  ICMP 的错误信息总是包括了源数据并返回给发送者

- FTP

  FTP概念：
  FTP是基于TCP的应用层协议，主要用来做文件传输

  FTP 有 2 种连接模式：主动（Active）和被动（Passive）。不管是哪种模式，都需要客户端和服务器建立 2 个连接（最大特点）。
  控制连接：用于传输状态信息（命令，cmd）
  数据连接：用于传输文件和目录信息（data）z

  主动模式：服务端主动发送文件数据给客户端，数据端口固定为20
  客户端打开命令端口N（N>1024）,与服务端命令端口21建立连接
  客户端开启N+1数据端口监听
  开启后客户端命令端口发送Port命令给服务端，告知服 务端客户端数据端口号，并且已准备好接收数据了
  服务端打开数据端口20，与客户端建立数据连接

  被动模式：服务端等待着客户端发送数据，数据端口随机
  客户端的命令端口 N 连接服务器的命令端口 21
  客户端命令端口发送PASV命令给服务端，代表本次连接为被动模式
  服务端接收到后，随机打开一个端口作为数据端口，监听客户端数据，并通过命令端口告知客户端端口号
  双方建立数据连接

- Websocket

  概念：
  基于TCP的全双工通信的应用层协议。在H5中定义。

  Websocket需要先建立连接（应用层建立连接），在进行通信，所以它是一种有状态的协议，所以后续的通信可以省略部分状态信息。很像TCP，TCP做了一层应用层的封装。

  作用：
  Websocket的出现主要就是为了解决服务器需要主动推送消息的场景，使用HTTP轮询低效且浪费资源

  WebSocket 需要借助 HTTP 协议来建立连接，webScoket基本只在web中使用，而web传统上一直使用HTTP，所以Webscoet看做是HTTP协议的升级，所以借助HTTP协议建立连接

  连接过程：
  客户端（浏览器）主动发出握手请求，有如下几个请求头
  Connection：Upgrade，表示希望连接升级
  Upgrad：WebSocket，表示希望升级到 WebSocket 协议
  Sec-WebSocket-Version，表示支持的 Websocket 版本，目前普遍使用的是13
  Sec-WebSocket-Key：客户端生成的随机字符串，与服务端返回的Sec-WebSocket-Accept对应服务端
  返回101 switching protocol
  响应头Sec-WebSocket-Accept是 对请求头 Sec-WebSocket-Key做了一些处理，摘要计算，base64编码等
  主要目的是为了避免普通HTTP请求被误认为升级为Websocket协议

- 流媒体协议

  概念：
  是指将一连串的多媒体数据压缩后，经过互联网分段发送数据，在互联网上即时传输影音以供观赏的一种技术
  也就是说，流媒体支持传输一部分就可使用一部分

  常见协议：
  RTP（Real-Time Transport Protocol），译为：实时传输协议，RFC3550、RFC3551，基于UDP
  RTCP（Real-Time Transport Control Protocol），译为：实时传输控制协议，RFC3550，基于UDP，使用 RTP 的下一个端口
  RTSP（Real-Time Streaming Protocol）译为：实时流协议，参考 RFC7820，基于 TCP/UDP 的 554 端口
  HLS（HTTP Live Streaming），基于 HTTP 的流媒体网络传输协议，苹果公司出品，参考：RFC 8216

- 即时通讯协议

- XMPP

  XMPP前身是 Jabber，基于TCP。默认端口 5222、5269

  有两个默认端口是因为
  一个端口是服务端与客户端通信的，另一个是客户端和客户端通信的

  XMPP特点为使用 XML 格式进行传输，体积较大、比较成熟的 IM（即时通讯） 协议，开发者接入方便。

- MQTT

  基于 TCP，默认端口 1883、8883

  其特点为
  开销很小，以降低网络流量，信息冗余远小于 XMPP
  不是专门为 IM 设计的协议，很多功能需要自己实现
  很多人认为 MQTT 是最适合物联网（IoT，Internet of Things）的网络协议 -- 就是因为其开销很小，即使在很小的设备甚至单片机上也有较好的性能

- 邮件协议

  - 邮件发送：SMTP

    SMTP为发送邮件常用协议，基于TCP

  - 邮件接收：IMAP、POP

    IMAP和POP为常见的邮件接收协议，都是基于TCP的。他们两者的特点不同

    IMAP特点：
    按需下载
    客户端连接服务器时，获取的是服务器上邮件的基本信息，并不会下载邮件
    等打开邮件时，才开始下载邮件
    客户端的操作（比如删除邮件、移动邮件）会跟服务器同步
    所有客户端始终会看到相同的邮件和相同的文件夹
    可以说IMAP协议的特点是比较符合web开发人员的认知的，因为web也是客户端和服务端同步

    POP特点：
    即刻下载
    客户端连接服务器时，将会从服务器下载所有邮件
    客户端和服务端不同步
    客户端的删除邮件、移动邮件不会跟服务器同步
    每个客户端连接后，相当于获得了自己的点子邮件副本
    即客户端和服务端是不同步的，客户端可以选择再从服务端下载一遍服务器上有的邮件

- 4. 实战工具

  - xshell、wireshark

    wireShark抓包原理是通过底层驱动，拦截网卡上流过的数据

  - Telnet、tcpdump
  - Cisco Packet Tracer
  - sucket 爬虫

- 3. 主要协议

- UDP

  概念：
  UDP 提供无连接、不可靠、不保证顺序的数据传输服务。

  因为其上面的三个特点，所以它的首部格式和大小较TCP都更加简单。UDP首部包含：
  源端口号、目的端口号、UDP长度、UDP校验和
  UDP校验和：
  UDP在计算校验和时，除了首部和数据部分，还会加上伪首部一起计算，增加安全性。

- TCP

  概念：
  TCP提供面向连接、可靠、保证顺序的数据传输服务

  它的首部至少20字节（固定部分），最大可以为60字节（40字节的可选部分）。首部较大是因为TCP的以下特点：
  可靠传输：丢包重传
  流量控制
  拥塞控制
  连接管理

  - 报文格式

    TCP报文：首部 + 数据

    首部：
    包含：以下（一行为32位、2字节）
    源端口、目的端口
    序号（发送数据的首字节编号）
    确认号 （期望收到下一个报文段的序号）
    数据偏移、标志位（Flags）、窗口
    校验和（类UDP）、紧急指针
    可选部分（长度可变，0~40字节）
    字段具体含义：
    数据偏移：数据针对整个TCP报文的偏移量，说白了就是TCP首部长度（但是需要乘以4）
    标志位包含：
    URG：紧急位，为1表示紧急指针有效
    ACK：确认位，为1表示确认号有效
    RST：重置位，此位为1强制释放连接，重新建立连接。
    SYN：同步位，为1表示建立连接
    FIN：结束位，为1表示释放连接
    窗口：告知对方下一次允许发送的数据大小。用来进行流量控制。
    紧急指针：正的偏移量，和序号值相加即为紧急数据最后一个字节的序号。该字节数据优先传输

  - 连接管理

- 三次握手

  一开始客户端处于closed状态，服务器处于listen状态
  第一次握手客户端发送SYN报文，请求建立连接。
  SYN报文发出后客户端进入SYN-SENT（同步已发送）状态，等待服务端的第2次握手。
  服务端在接收到SYN报文后，进入SYN-REVD（同步已接收）状态
  至此第一次握手完成
  第二次握手服务端发送SYN、ACK报文
  客户端接收到服务端返回的ACK报文后，进入ESTABLISHED（连接已建立）状态。
  至此第二次握手完成
  第三次握手客户端发送ACK报文给服务端
  客户端接收到客户端返回的ACK报文后，进入ESTABLISHED（连接已建立）状态。
  至此三次握手完成，可以开始数据通信了。
  思考下：如果只有两次握手会有什么问题？即当服务端返回确认报文后，双方建立连接。如果没有第三次握手，即客户端确认过程。那么有很多情况导致服务端错误地建立了连接。
  比如：第二次握手丢包了，客户端实际没收到，如果服务端直接建立了连接，那么就是一个空连接，永远不会受到客户端发送的数据。
  再比如第一次握手因网络原因迟到了，客户端超时重传，然后经历了建立连接、数据交互和释放连接，此时迟到的第一次握手到达了服务端。 但服务端并不知道是迟到的，以为是客户端再次要求建立连接，所以又返回ACK。 实际上客户端对次ACK不予理睬。如果服务端直接建立连接又是一个空连接等等

- 四次挥手

  假设客户端主动要求断开连接，实际双方都可以发起断开连接请求
  释放连接前双方都处于ESTABLISHED状态
  第一次挥手，客户端发送FIN报文，代表请求断开连接。
  当FIN报文发送后，客户端进入FIN-WAIT-1（终止等待1）状态，表示等待服务端返回ACK报文。服务端接收到FIN报文。第一次挥手完成

  第二次挥手，服务端对接收到的FIN报文返回ACK确认
  服务端一旦发送了ACK报文，其自身就变为了CLOSE-WAIT（关闭等待）状态。在此状态下，服务端会根据自身是否还有数据需要发送给客户端决定，是否发送FIN报文
  当客户端收到了ACK报文后，其状态变为FIN-WAIT-2(终止等待2)，表示等待服务端发送断开连接请求
  至此第二次挥手完成
  CLOSE-WAIT状态下的服务端，如果发现自身已无数据要发送了。就会发送FIN报文给客户端
  当服务端发送FIN报文后，其自身状态变为LASK-ACK（最后确认）状态，即等待客户端最后的确认报文
  客户端收到服务端发送的FIN报文
  至此第三次挥手完成
  客户端返回ACK报文，即同意服务端断开连接。
  客户端一旦发送ACK报文后，其自身就进入了TIME-WAIT状态，此状态下等待2MSL（若无再次收到服务端报文）后，自动进入CLOSED状态
  服务端在接收到了客户端发送的ACK报文后，立即进入CLOSED状态。
  至此四次挥手全部完成
  为什么必须是四次挥手?
  假设只有两次或者三次挥手，会产生很多问题。
  省去第三次挥手：服务端可能还有数据未发送完成。直接返回ACK导致数据丢失。如果强制等待数据发送完成再返回ACK，可能导致超时重传，甚至多次超时重传。不合理。
  省去第四次挥手： 如果服务端返回的FIN报文丢失，服务端此时直接进入CLOSED状态，而客户端并不知道，客户端就不会关闭连接，一直等待服务端的FIN报文，浪费客户端资源

  其实这里说到省去第三次握手的情况，实际情况下还真的有可能发生。只不过并不是服务端不管有没有数据，直接省略。而是如果服务端判断出无数据要发送，直接合并第三次和第四次挥手，发送ACK和FIN报文。 这个在实际应用中是存在的。

- 可靠传输

  可靠传输主要依靠ARQ协议（自动重传请求），它有两种模式：
  停止等待ARQ协议：它的核心即同步发送（发送一个TCP数据后必须收到确认才能发送下一个）和超时重传。

  超时的情况是很多的，比如。。。。。。
  只要出现问题，发送方没有收到确认，发送方就会根据定时器等待，时间到了重发请求。这之前不会发送下一个tcp数据。
  连续ARQ + 滑动窗口协议
  所谓连续即发送方不再同步的一个一个发送tcp数据了，而是连续（可以理解为并发）多条数据。

  最终发多少条是根据窗口大小来决定的。窗口越大，连续发送的tcp数据条数就多。
  在同一个窗口内的数据，接收方只需要返回一次确认即可。
  如果传输时某个包出问题了，那么接收方返回的确认消息只能从丢失前的数据统计。即使丢失后还正确接收到了其他包。
  补充：现在有了选择性确认（SACK）技术，明确告诉发送方接收到了哪些包。发送方则只需重传丢失的包即可。

- 流量控制

  为什么要进行流量控制?
  发送方发送过快，超出接收方的处理能力，如果不加以控制，接收方只能选择丢包，大量丢包造成网络浪费，也会把服务端搞崩溃

  概念：控制发送方的发送速率不要过快，使接收方来得及处理。

  原理：通过确认报文中的窗口字段控制发送方的发送速率。
  如果接收窗口大小为0，代表接收方缓存区已满，发送方就会停止发送数据（定时确认，确认接收方窗口是否不为0了）

- 拥塞控制

  拥塞控制概念：
  拥塞控制即通过调整发送窗口大小控制发送方的数据发送频率。

  目的：
  防止过多数据注入到网络，避免网络中的路由器或链路过载。

  跟流量控制的区别：
  流量控制是端对端的，是在两台设备之间完成的流量控制
  拥塞控制是网络中的所有设备遵循一定规则，来保证整个网络的畅通。
  类比：拥塞控制和交通管理非常类似。交通系统也是内部所有成员遵循交通规则保证交通顺畅。

  拥塞控制的方法（规则）包含：慢开始、拥塞避免、快重传、快恢复。
  慢开始：TCP连接刚建立，一点一点地提速，试探网络的承受能力。
  具体实现：发送窗口的大小 = 拥塞窗口 和 接收窗口 两者的较小值，而接收窗口更多表明服务端处理情况，拥塞窗口则根据当前网络状况动态改变。网络状况是根据接收端丢包情况判断。
  当连接刚建立时，拥塞窗口为1，代表可传一个MSS大小的数据。当接收到一个ACK就将拥塞窗口加1，每轮次拥塞窗口乘以2，呈指数上升。
  拥塞避免：当拥塞窗口大于慢启动阈值ssthresh后，就会进入拥塞避免算法
  拥塞避免算法一个轮次拥塞窗口+1，从指数上升变为线性增加（加法增大）
  当出现丢包，就认为网络出现拥塞，这个时候会进行乘法减小。即将慢启动阈值置为丢包时的拥塞窗口的一半。并且将拥塞窗口重置为1，重新开始慢启动算法（这是旧版本TCP的做法），如今已优化为快恢复算法。
  快重传
  拥塞避免算法根据丢包判断是否进行乘法减小。而丢包有两种判断方式，一种是传统的超时重传，另一种就是快重传
  所谓的快重传即接收方如果收到了失序的分组，就立即发出 丢失包的前一个报文段的 重复确认。告知发送方该报文的下一个报文丢失了。发送方收到后立即重传。
  快恢复
  当发送方连续收到3个重复确认，说明网络出现拥塞，就会执行“乘法减小算法”，将慢开始阈值 设为 拥塞窗口的一般半
  但快恢复在此时不执行慢开始算法，而是直接执行拥塞避免算法，拥塞窗口不置为1，而是置为慢开始阈值

  1、2、3、4如此反复执行，根据网络状况调整发送窗口大小。

- HTTP

  超文本传输协议、是互联网中应用最广泛的应用层协议之一
  设计 HTTP 最初的目的是：提供一种发布和接收 HTML 页面的方法，由 URI 来标识具体的资源
  后面用 HTTP 来传递的数据格式不仅仅是 HTML，应用非常广泛

- 代理

  代理服务器：
  本身不生产内容，处于中间位置转发上下游的请求和响应。

  分为正向代理和反向代理。
  正向代理
  代理的对象是客户端。如浏览器端设置的代理，又比如Fiddler抓包工具，就是在客户端启动了正向代理。
  反向代理
  代理对象是服务端，如Nginx反向代理
  代理的作用一般有以下几点
  隐藏身份-正向代理隐藏客户端身份（如黑客攻击隐藏自身地址），反向代理隐藏服务端身份，从而有助于服务端的安全防护。
  绕过防火墙，俗称的翻墙。
  负载均衡，根据服务器承载能力和负载均衡算法，将请求合理地分配给多台业务服务器
  和代理相关的HTTP头信息包括：
  Via：追加经过的每一台代理服务器的主机名（或域名）
  X-Forwarded-For：追加请求方的 IP 地址，即该代理服务器上一层的IP地址
  X-Real-IP：客户端的真实 IP 地址
  这些头信息是否添加根据代理服务器配置决定

- 缓存

  缓存分为强缓存和协商缓存
  强缓存：不发送请求到服务器，直接读取浏览器本地的缓存，返回200状态码
  协商缓存：强缓存失效（缓存过期），浏览器携带缓存标识像服务器发送请求，由服务器根据缓存标识决定是否使用缓存。
  如果使用了协商缓存，这次请求过后，又会根据过期时间优先使用强缓存。
  具体流程查看 "缓存流程.png"

  刷新对缓存的影响：
  ctrl+f5：强制刷新时，忽略强缓存和协商缓存，直接像服务器请求。忽略浏览器缓存浏览器本身就可以办到，忽略协商缓存是请求头中不加标识字段
  f5普通刷新：会跳过强缓存，但是会检查协商缓存
  在地址栏输入URL：会同时使用使用强缓存和协商缓存

- 报文格式

  HTTP的报文，基本类型是字符串。它的格式有着严格的规范（根据ABNF）

  HTTP-message = start-line *(header-field CRLF)CRLF[message-body]

  start-line = request-line / status-line
  request-line = method SP request-target SP HTTP-version CRLF
  status-line = HTTP-version SP Status-code SP reason-phrase CRLF

  header-field = field-name ":" OWS field-value OWS

- 头信息

- 请求头

  请求头分下类为：浏览器信息、请求本身、缓存相关、跨域相关、通用信息（其他）

  浏览器信息：
  User-Agent：浏览器的身份标识字符串
  Accept：能够接受的响应内容类型（Content-Types）
  Accept-Charset：能够接受的字符集（GB2312,utf-8）
  Accept-Encoding：能够接收的编码方式列表（gzip, deflate）
  Accept-Language：能够接收的响应的内容的自然语言列表

  请求本身：
  Content-Type：请求体的类型 --（一般有请求体时带有，6同理）
  Content-Length：请求体的长度（单位为字节）
  Range：仅请求某个实体的一部分。字节偏移以 0 开始
  格式为：“Range: bytes=start-end”，如果服务器支持则返回206以及片段数据，否则返回200，整个数据。分片下载技术。

  跨域相关：
  Origin：发起一个针对跨域资源共享的请求，即跨域请求时浏览器会带上

  缓存相关：
  cache-control：用来指定在这次的请求/响应链中的所有缓存机制都必须遵守的指令
  If-Modified-Since：资源的最后一次修改时间
  可以认为就是请求该资源时服务端返回的响应头中的Last-Modified
  If-None-Match：资源的唯一标识（单向散列函数计算的hash值）
  可以认为即请求该资源服务端返回的响应投中的ETag
  通用信息：
  Host：服务器的域名、端口号
  Date：发送该消息的日期和时间
  Referer： 翻译为引用，如果是html页面跳转，一般是他的前一个页面，引用自前一个页面。如果是js、css等资源，一般是当前页面。引用自当前页
  Cookie：Cookie可以做浏览器本地缓存，但更多时候是用来做鉴权的。
  Connection：该浏览器想要优先使用的连接类型（keep-alive）

- 响应头

  响应头分下类为：服务器信息、响应本身、缓存相关、跨域相关、通用信息（其他）

  服务器信息：
  Server：服务器名称
  响应本身：
  Content-Type（同请求头）
  Content-Length（同请求头）
  Content-Range：这部分消息是属于完整消息的哪部分
  一般和请求头range配合使用
  缓存相关：
  CaChe-control：向从服务器直到客户端在内的所有缓存机制告知
  Last-Modified：资源的最后一次修改时间
  ETag：资源的唯一标识
  缓存相关还有Pragma、Expires，但那是HTTP/1.0的产物，存在一些问题，建议还是使用Cache-control好
  跨域相关：
  Access-Control-Allow-Origin：指定哪些网站可参与到跨资源共享过程中
  Access-Control-Allow-Headers：指定哪些请求头可通过跨域
  Access-Control-Allow-Methods：指定哪些方法可通过跨域
  通用信息：
  Date：同请求头
  Content-Disposition：可以使客户端下载文件并建议文件名的头部
  Location：用来重定向，或者创建了某个新资源时使用
  set-Cookie：返回一个 Cookie 让客户端去保存
  Connection：针对该连接所预期的选项（同请求头）

- HTTP/2.x

  HTPP/2主要解决的问题：并发请求、请求和响应一对一、头信息重复传输
  所有支持 HTTP/2 的浏览器都强制只使用 TLS(https) 连接。所以可有认为 HTTP/2 只在 https 环境下运行
  基本概念：
  数据流：已建立的连接内的双向字节流，可以承载一条或多条消息
  消息： 与逻辑 HTTP 请求或响应对应，由一系列帧组成
  帧：HTTP/2 通信的最小单位
  来自不同数据流的帧可以交错发送，然后再根据每个帧头的数据标识符重新组装
  HTTP/2所有的通信都在TCP 连接上完成，此连接可以承载任意数量的双向数据流。所以支持并发

  与HTTP1.x的区别：
  数据格式：HTTP1.x传输的报文是字符串、HTTP/2传输的是二进制数据
  HTTP/2的特性：
  多路复用
  所谓多路复用，即多个数据流共用一个 TCP 连接。并且互不干扰。其原理就是消息被分解为帧后交错发送，另一端根据帧头的数据标识重组消息。
  并发请求不会再有性能问题，不必在为 减少HTTP请求做优化
  -- 解决HTTP/1 并发请求需要建立多个TCP连接问题
  头部压缩
  HTTP/2 使用 HPACK 压缩请求头和响应头，可以极大减少头部开销，从而提高性能。
  那么他的原理是双方维护同一张“首部表”，是该首部最近一次出现的值，如果发送的请求首部没有更新，则直接使用首部表内的索引。从而减少了数据传输。
  -- 解决HTTP1.1 头信息重复传输问题
  服务器推送
  服务器可以对一个客户端请求发送多个响应
  比如客户端请求了一个HTML，服务端除了返回HTML页面本身外，还可以将HTML内加载的资源也一并返回给客户端。无需等浏览器加载后，主动发起
  注意：这并不是服务器可以任意推送，只是额外推送。并不是类似webSocket那样的全双工模式
  -- 解决请求和响应一对一，效率偏低问题
  优先级
  HTTP/2 标准允许每个数据流都有一个关联的权重和依赖关系。
  可以向每个数据流分配一个介于 1 至 256 之间的整数，每个数据流与其他数据流之间可以存在显式依赖关系。
  最终构建的就是一颗优先级树，父节点总是优先被分配资源
  HTTP/2存在的问题：
  对头阻塞
  HTTP/2 并发时会将帧交错发送。但是如果再tcp传输过程中丢包了。tcp不会将后续的包向上传递，而是等待丢失的包重传成功后再向上传递——这是TCP承诺的特性：保证顺序。 这就造成了对头阻塞
  因为丢失的包属于数据流A，对数据流B完全无影响。本来可以将数据流B优先接收拼装的。但却发生了阻塞
  握手延迟
  因为 TCP 的 3 次握手以及 TLS 连接，所以就会存在握手延迟。
  因为有这些问题的存在，所以现在还在完善的HTTP/3，尝试解决这些问题

- HTTP/3.x

  之前说到HTTP/2存在对头阻塞和握手延迟的问题。这些问题都是因为TCP协议的特点、面向连接、可靠传输和保证顺序决定的。所以如果要解决这些问题，就需要启用TCP协议

  因此HTTP/3 使用基于 UDP 协议的 QUIC 协议实现。UDP就不存在对头阻塞和握手延迟问题。因为其无连接和不保证顺序。

  那么HTTP/3如果保证可靠传输呢?
  就是通过中间层QUIC协议来保证。Google 研发的QUIC协议工作在UDP和HTTP层之间，用它来保证可靠传输，同时灵活性更强

  HTTP3目前尚未完全稳定和标准化，所以是不推荐使用的

- 其他

  - Web Service

    WebService 是使用固定 XML 格式（SOAP 标准）的 HTTP 接口，一般是政府、公司想要公开自己的数据时使用。该技术很老旧，现在一般都直接使用 HTTP+JSON，简单方便。

  - RESful

    RESTful是一种互联网软件架构设计风格
    定义了一组用于创建 Web 服务的约束， 符合 REST 架构的 Web 服务，称为 RESTful Web 服务

- HTTPS

  概念:
  HTTPS 是在 HTTP 的基础上使用 SSL/TLS 来加密报文（整个个HTTP 报文，不仅限于消息体），对窃听和中间人攻击提供合理的防护

  这里需要注意下下，SSL/TLS并不是HTTPS的专属。其他应用层协议也有应用。比如 FTP+SSL/TLS 变为 FTPS、SMTP变为SMTPS等

  HTTPS的通信过程如下:
  TCP3次握手
  TLS 的连接
  HTTP 请求和 响应
  可以看到和HTTP对比就多了一次TLS的连接。在TLS子节重点分析

  - 摘要算法（不可逆加密）

    加密分为可逆加密和不可逆加密。摘要算法就属于不可逆加密

    不可逆加密：即无法根据加密后的内容反推出源消息。常见的md5、SHA-1、SHA-2（SHA-256、SHA-384、SHA-512,后面跟的数字代表散列长度）

    现在md5和SHA-1已经不安全了。利用枚举方式，被破解的可能性较大

    应用:
    防止数据被篡改：根据对比消息发送前后的散列值
    密码数据库存储：现在很多数据库存储的密码都是存储的散列值，这样即使数据库泄露，也可保证密码的安全
    特点:
    单向性、唯一性、快速、固定长度的散列值（源消息大小任意，最终的散列值长度固定）

- 可逆加密

  概念：
  加密后的密文能够被解密回源消息的，就属于可逆加密。可逆加密势必存在密钥。

  分类：
  可逆加密分为对称加密和非对称加密。他们的区别是加解密的密钥是否相同。相同的为对称加密。

  - 对称加密

    概念：
    在对称加密（又称对称密码）中，加密用的秘钥和解密用的秘钥是相同的

    常见的对称加密算法：
    DES
    DES 是一种将 64bit 明文加密成 64bit 密文的对称加密算法，密钥长度是 56bit。
    目前已经可以在短时间内被破解，不推荐使用
    3DES
    3DES 将 DES 重复 3 次所得到的一种密码算法，也叫做 3 重 DES。
    三重 DES 并不是进行三次 DES 加密。而是进行DES加密、解密、加密。这三次加解密的密钥一般均不相同
    目前市面上还有应用，但处理速度不高，安全性逐渐暴露问题
    AES
    取代 DES 成为新标准的一种对称加密算法
    密钥长度有128/192/256bit三种。
    推荐使用

    密钥配送问题：
    对称加密，最大的问题就是要解决密钥配送问题。一般有如下方案：
    事先共享密钥
    密钥配送中心（KDC）
    非对称加密（重点了解）

  - 非对称加密

    概念：
    在非对称加密（又称公钥密码）中，加密用的秘钥（公钥）和解密用的秘钥（私钥）是不同的。

    基本流程：
    接收方B 产生一对公钥和私钥
    B将公钥发送给 发送方A
    A拿到公钥后，利用公钥加密消息，将密文发送给B
    B利用对应的私钥解密，拿到源消息
    整个过程很安全，因为私钥没有在网上流通，不存在被截取的风险

- 混合密码系统

  对称加密和非对称加密，他们是各有利弊的
  非对称加密更加复杂并且安全
  但是非对称加密，加解密速度慢很多
  所以在实际应用中，大部分情况下是使用混合密码系统，即对称加密和非对称加密共同使用

  它的工作模式大致如下：
  使用对称加密加密消息体，得到消息体密文
  使用非对称加密加密对称加密使用的密钥，得到密钥密文
  将两者一起发送给接收方
  这样数据量比较大的消息体使用对称加密加快速度。而密钥一般是固定位数的，数据量很小，使用非对称加密。既提高了效率，又保证了安全性。普遍使用。

- 数字签名

  数字签名的作用（对比生活中的签名）：
  确认消息的完整性、识别消息是否被篡改
  防止消息发送人否认
  它是如何工作的呢?
  消息发送者A生成一对公钥和私钥
  A将消息通过单向散列函数生成散列值，并使用私钥对其进行加密（这就是签名的行为，私钥加密就是签名了）
  将消息本身和加密后的散列值发送给接收方B
  B在接收到后，也将消息使用相同的单向散列函数生成散列值。并且将A发送的散列值进行解密
  对比两个散列值，若不一致，则消息有问题
  所以数字签名这一整套流程，就是为了验证你收到的消息确实是我发送的，和我发送的时候内容是一样的。

- 证书

  混合密码系统看上去是不是已经比较完美了，但实际上它还存在风险。即中间人攻击。

  过程如下：
  在发送方A和接收方B中间，有一个中间人进行数据伪造。在B将公钥发送给A时，对其进行拦截，并且替换成自己的公钥
  当A发送将公钥加密的数据发送给B时，中间人在使用自己公钥对应的私钥解密。

  正是因为存在这样的风险，所以需要证书

  密码学中的证书，全称叫公钥证书（Public-key Certificate，PKC）。证书中一般包含域名、IP等身份信息以及最重要的公钥信息。然后证书颁发机构对其进行数字签名。

  证书颁发机构不用担心数字签名的公钥被中间人替换或者伪造，因为一般权威机构的公钥已经内置于操作系统或者浏览器里了。所以通过证书就可以保证公钥的合法性。解决中间人攻击。

- 加密总结

  加密的进化史大概为：
  对称加密 => 非对称加密 => 混合密码 => 混合密码 + 证书
  对称加密，快速高效。但密钥事先在网络中传输，极不安全
  非对称加密，私钥个人保存，不在网络流通。相对安 全，但是非对称加密耗时低效
  混合密码系统，消息主题对称加密，对称密钥非对称加密。安全高效
  中间人攻击，伪造公钥甚至伪造数字签名（签名也是事先传输公钥）
  混合密码 + 证书，将加密公钥利用权威机构的私钥进行数字签名。保证公钥合法性。

- SSL、TLS

  SSL是TLS的前身，TLS可以看做是SSL的增强版，但是两者是不兼容的。主要原因是两者的加密算法不同，所以说两者不能共用

  根据RFC文档，SSL已经被弃用，所以现在的服务都推荐使用TLS

  TLS 是工作在传输层和应用层中间，如果非要说它处于哪一层，大部分人认为他处于会话层。HTTP 报文先通过 TLS 处理，然后传递给传输层，通过 tcp 协议传输。
  TLS 的连接主要目的就是交换密钥，而相对复杂的设计就是保证交换的密钥安全。
  TLS的连接过程如下：
  Client Hello（TLS版本、支持加密组件列表、client随机数）
  Server Hello（TLS版本、选择的加密组件、server随机数）
  Certificate（服务器的公钥证书（被 CA 签名过的））
  Server Key Exchange（用以实现密钥交换算法的Server Params）
  Server Hello Done（协商部分结束）
  2、3、4、5都是服务端发送给客户端，可能在一个报文段中一起传输
  Client Key Exchange（用以实现密钥交换算法的Client Params）

  客户端和服务端：利用密钥交换算法（参数先前已经交换得到），计算出随机密钥串pre-master secret

  然后在使用随机秘钥串和两方的随机数生成主密钥（主密钥可衍生出其他密钥）

  至此，双方的加解密基础已有
  Change Cipher Spec
  client指示Server从现在开始发送的消息都是加密过的
  Finished（抓包也称 Encrypted Handshake Message）
  客户端发送：包含连接至今全部报文的整体校验值（摘要），加密之后发送给服务器。验证服务器是否能成功解密
  Change Cipher Spec
  Server指示client从现在开始发送的消息都是加密过的
  Finished（抓包也称 Encrypted Handshake Message）
  同客户端，验证服务端加密内容，客户端是否能解密
  到此为止，客户端服务器都验证加密解密没问题，握手正式结束。后面开始传输加密的 HTTP 请求和响应

- 6. 高频面试题