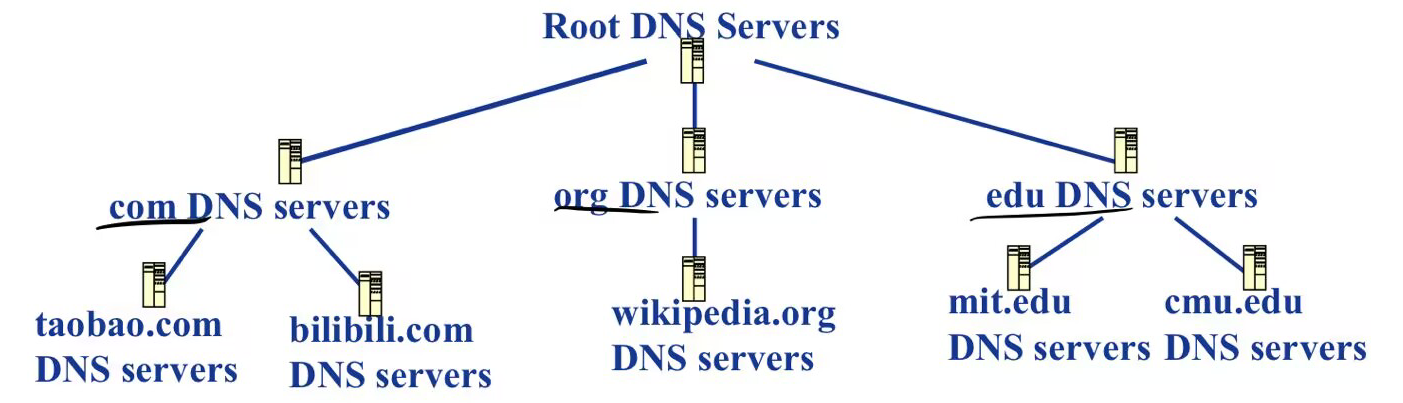
# 5.计算机网络-应用层

1. 网络应用体系结构
   1. 客户/服务器结构（Client-Server、C/S）
      1. 服务器：
         1. 7×24小时提供服务
         2. 永久性可访问地址/域名
         3. 利用大量服务器实现可扩展性（服务器农场）
      2. 客户
         1. 与服务器通信，使用服务器提供的服务
         2. 间歇性接入网络
         3. 可能使用动态IP地址
         4. 不会与其他客户直接通信
   2. 纯点对点结构（P2P）
      1. 没有永远在线的服务器
      2. 任意端系统之间可以直接通讯
      3. 节点间歇性接入网络，且可能改变IP地址
      4. 优点：高度可伸缩
      5. 缺点：管理难度大
   3. 混合结构：结合两者优点，规避缺点
      1. e.g. Napster：文件传输使用P2P，文件搜索采用C/S
         1. 每个结点向中央服务器登记自己的内容
         2. 每个结点向中央服务器提交查询请求，查找感兴趣的内容，找到后与最近的有该内容的主机直接通讯传输文件
2. 网络应用通信原理
   1. 进程间通信：网络应用的基础
      1. 客户机进程：发起通信的进程
      2. 服务器进程：等待通信请求的进程
      3. 同一主机上运行的进程之间通信依赖于进程间通信机制，由操作系统提供
      4. 不同主机上运行的进程通信：消息交换（报文交换）
      5. 套接字（Socket）：进程间通信利用Socket发送/接收消息实现，类似于寄信
         1. 传输基础设施向进程提供API：传输协议的选择、参数的设置
      6. 如何寻址进程：IP地址+端口号
         1. IP地址：寻址主机
         2. 端口号：寻址主机上的进程（见传输层）
      7. 应用层协议：公开协议（RFC定义的允许互操作的协议）/私有协议（多数P2P文件共享应用）
         1. 消息类型：请求消息/响应消息
         2. 消息语法格式：消息中有哪些字段，每个字段如何描述
         3. 字段的语义：字段中信息的含义
         4. 规则：进程何时、如何发送/响应消息
      8. 网络应用对传输服务的需求：
         1. 数据丢失/可靠性
            1. 某些网络应用容忍一定的数据丢失：网络电话
            2. 某些则需要100%可靠的数据传输：文件传输
         2. 时间/延迟
            1. 有些应用要求低时延：网络游戏
            2. 有些则对速度要求并不一定很高，如文件传输、email、web
         3. 带宽
            1. 有些应用需要带宽达到最低要求：长/短视频、网络游戏
            2. 有些应用可以适应任何带宽：弹性应用，如email
      9. Internet提供的传输服务：TCP和UDP
         1. 基于TCP的典型应用：email、远程终端访问、Web、文件传输等
         2. 多数情况下基于UDP（有时也用TCP）的典型应用：流媒体、网络电话
3. 域名解析系统（DNS）：Domain Name System
   1. Internet上主机/路由器的识别问题
      1. IP地址
      2. 域名：[www.4399.com](http://www.4399.com/)
   2. 域名解析系统DNS：
      1. 多层命名服务器构成的分布式数据库，实现域名和IP地址的映射
      2. 应用层协议：完成名字的解析
         1. Internet核心功能：由应用层协议实现
         2. 网络边界复杂
      3. DNS服务
         1. 域名向IP地址的翻译（最基本）
         2. 主机别名
         3. 邮件服务器别名
         4. 负载均衡：Web服务器
            1. 为同一个域名提供多个服务器地址，动态调整其相对顺序，实现各机负载均衡
      4. DNS是多层而不是集中式：单点失败、流量问题、距离问题、维护性问题
      5. 域名层次结构：域名从后往前是层次从上到下
         1. 根
         2. 顶级域名：com、net、org、edu、gov、cn…
         3. 二级域名：sina、google、baidu、edu、4399…
         4. 三级域名：www、hit、mail、pku…
         5. 四级域名：www、mail…
      6. 分布式层次式数据库（与域名层次结构相对应）



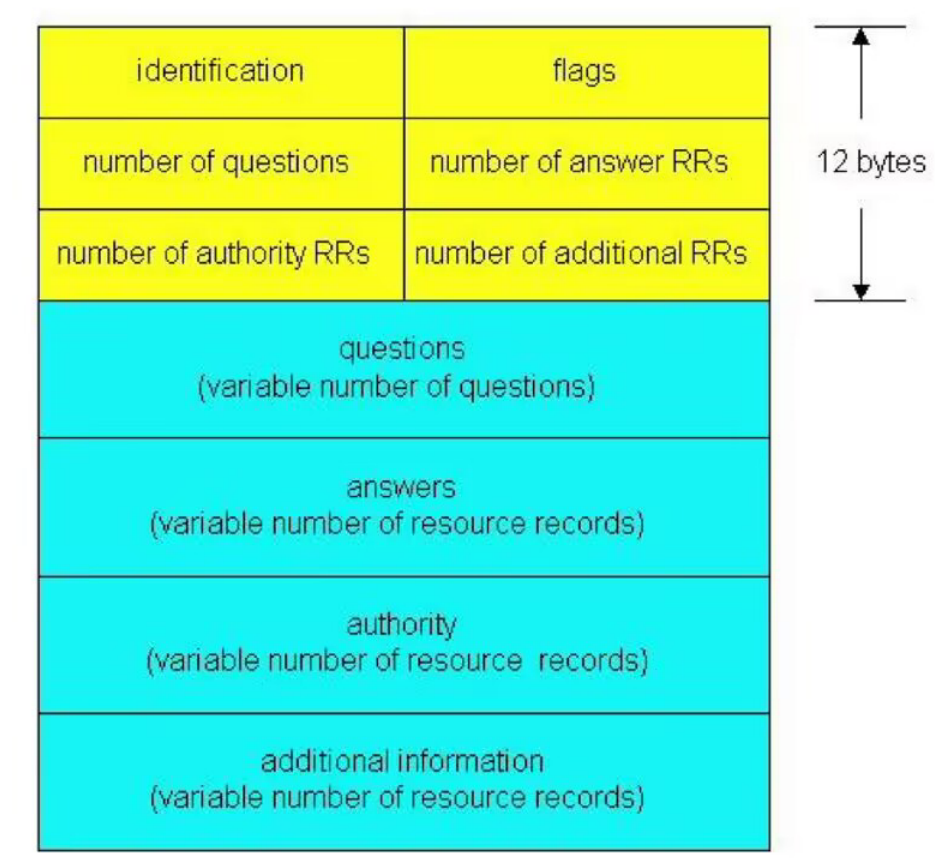
* + - 1. 客户端想要查询[www.4399.com](http://www.4399.com/)的IP：
         1. 客户端查询根服务器，找到com域名解析服务器
         2. 客户端查询com域名解析服务器，找到[4399.com](http://4399.com/)域名服务器
         3. 客户查询[4399.com](http://4399.com/)域名解析服务器，找到[www.4399.com](http://www.4399.com/)的IP地址
    1. 本地域名解析服务器
       1. 不严格属于层级体系
       2. 每个ISP至少有一个本地域名服务器
          1. 默认域名服务器
       3. 主机进行DNS查询时，查询会首先被发到本地域名服务器
          1. 本地域名服务器会作为代理（proxy），将查询转发给层级式域名解析服务器系统
    2. DNS根域名服务器
       1. 本地域名服务器无法解析域名时，访问根域名服务器
       2. 根域名服务器
          1. 如果不知道映射，查询顶级域名服务器或权威域名服务器
          2. 获得映射，向本地域名服务器返回映射
       3. 顶级域名服务器（TLD）和权威域名服务器
          1. 顶级域名服务器：负责com、org、net、edu等顶级域名和cn、uk、fr等国家级域名
          2. 权威域名服务器：组织的域名解析服务器，提供组织内部服务器的解析服务（自治的局域系统）
       4. DNS查询过程：递归查询和迭代查询
          1. 递归查询：本地域名服务器只发起一次通讯，每一层服务器“帮忙找”，找到后逐级返回
          2. 迭代查询：本地域名服务器需要发起多次查询，按顺序到各域名服务器“自己找”
       5. DNS记录缓存和更新
          1. 域名解析服务器获得域名-IP映射，会缓存这一映射，一段时间后缓存条目失效
          2. 记录的更新/通知机制：RFC2136、DNS UPDATE
          3. 本地域名服务器一般会缓存顶级域名服务器的映射

故实际上根域名服务器不常被访问

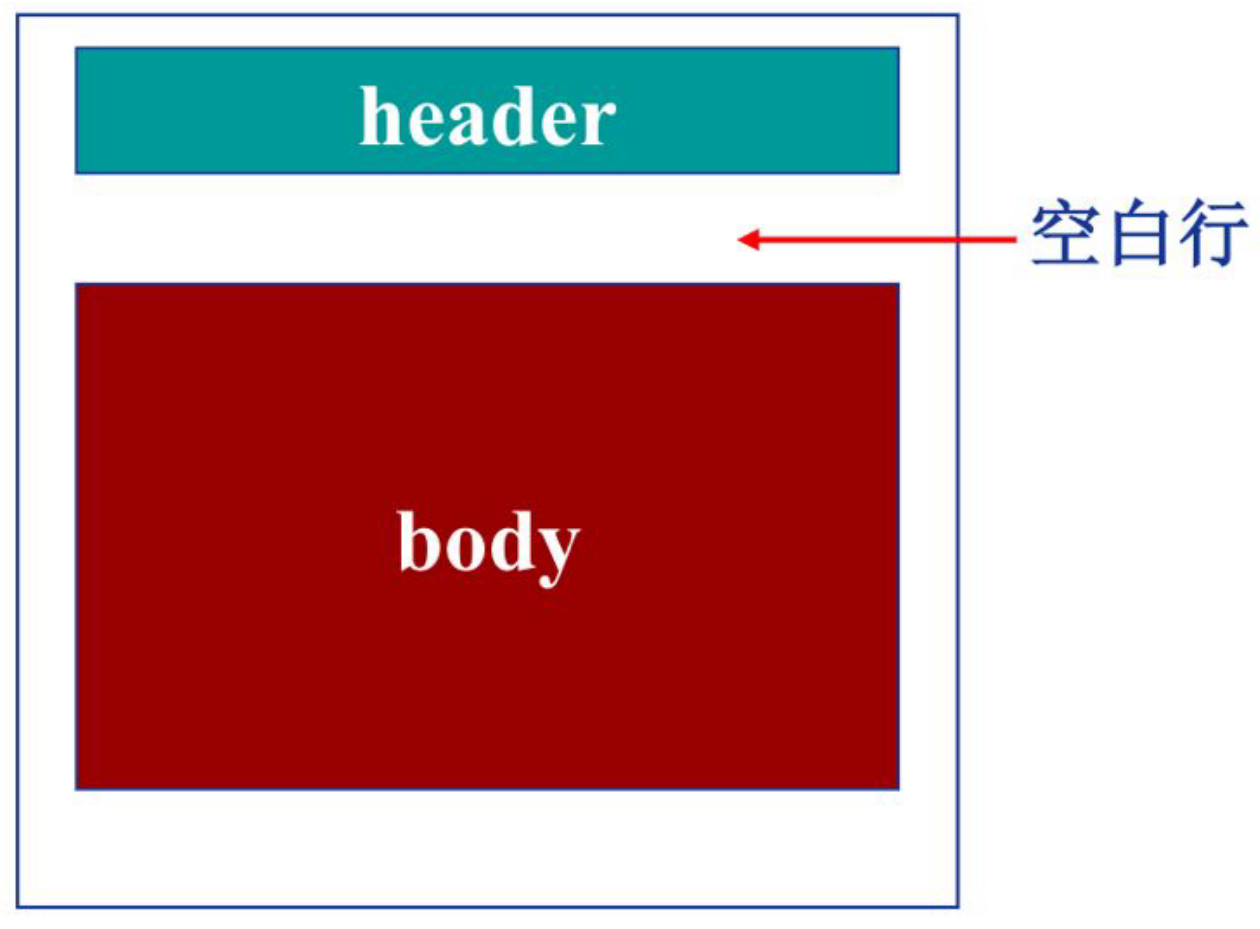
* + - 1. DNS记录：（name，value，type，ttl）
         1. Type=A：name：主机域名，value：主机IP地址
         2. Type=NS：name：域（[edu.cn](http://edu.cn/)），value：该域权威域名解析服务器的主机域名
         3. Type=CNAME：name：某一真实域名的别名，value：真实域名
         4. Type=MX：name：邮件服务器别名，value：邮件服务器名
      2. DNS协议与消息
         1. DNS协议：查询（query）和回复（reply消息），消息格式相同
         2. 消息头部

Identification：16位查询编号，回复使用相同的编号

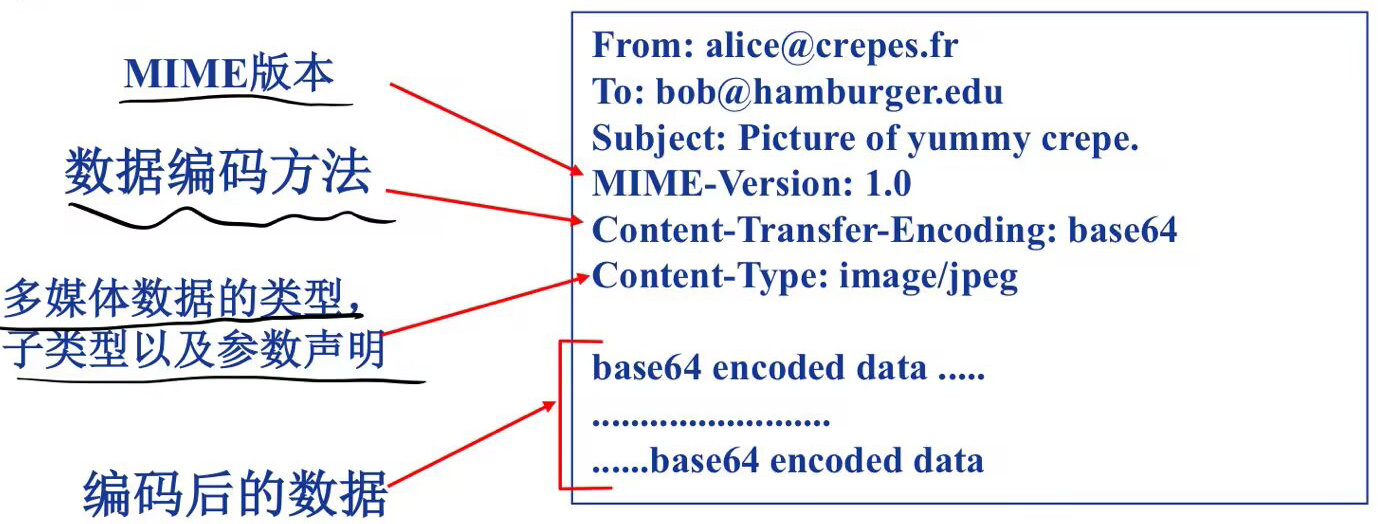
Flags：查询或回复，期望递归，递归可用，权威回答



1. FTP应用
   1. FTP：文件传输协议，向远程主机上传输文件或从远程主机接收文件，典型的C/S模式应用
   2. FTP控制连接与数据连接分开：FTP客户端与FTP服务器通过端口21联系，使用TCP作为传输协议
      1. 客户端通过控制连接获得身份确认、发送命令浏览远程目录
      2. 收到一个文件传输命令时，服务器打开一个到客户端的TCP连接；传输完成后，服务器关闭连接
      3. 收到另一个文件传输命令时，服务器会打开第二个数据连接用来传输另外一个文件
      4. 控制连接是带外传送
      5. FTP服务器维护用户的状态信息：当前路径、用户账户与控制连接对应
   3. FTP命令、响应：在控制连接上以ASCII文本方式传送
2. Email应用
   1. Email应用的构成：邮件客户端、邮件服务器、SMTP协议
      1. 邮件客户端（用户代理）：读、写Email消息；与服务器交互，收、发Email消息
      2. 邮件服务器：用户不能7×24h在线，故需要服务器功能
         1. 邮箱：存储发给该用户的email
         2. 消息队列：存储等待发送的email
      3. SMTP协议：邮件服务器之间传递消息使用的协议；其中客户端为邮件发送方，服务器为接收方
   2. SMTP协议：基于TCP可靠传输的Email消息传输协议，使用端口25
      1. 传输过程的三个阶段：握手、消息传输、关闭
      2. 命令/响应交互模式：命令使用ASCII文本，响应使用状态代码和语句
      3. Email消息只能包含7位ASCII码
   3. Email消息格式：
      1. 头部行：To、From、Subject
      2. 消息体：消息本身，只能是ASCII字符

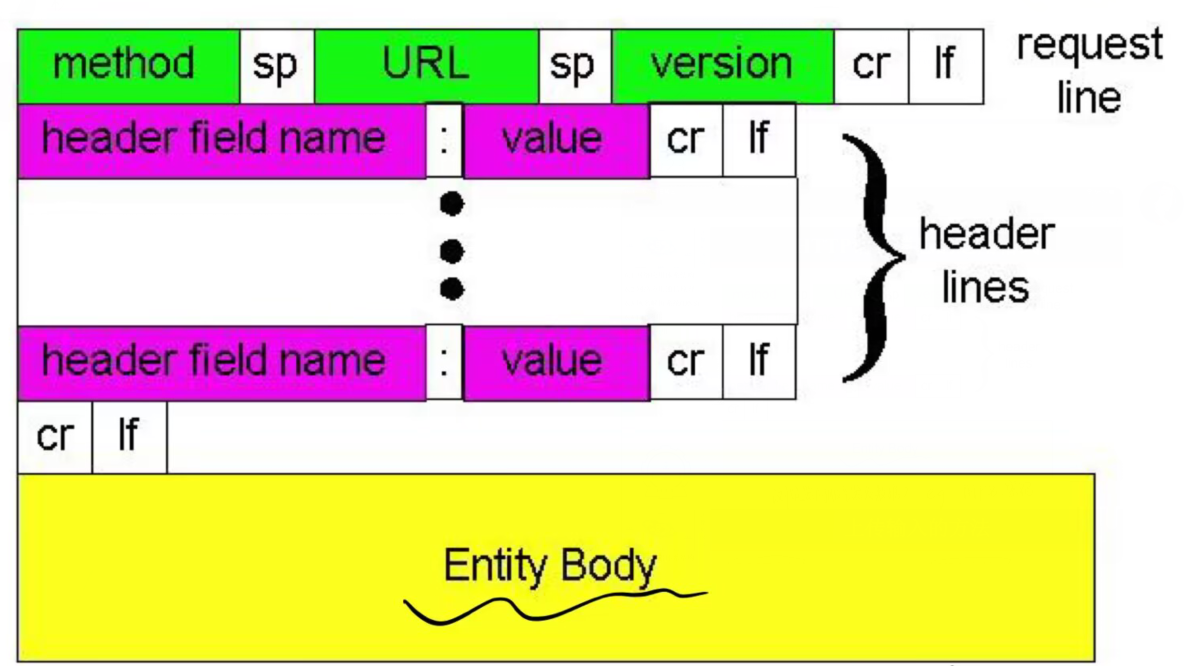


* + 1. 多媒体邮件扩展（MINE）：通过在邮件头部增加额外的行声明MINE的内容类型



* 1. 邮件访问协议：从服务器获取邮件
     1. POP：认证/授权（客户端 服务器）和下载
        1. 认证过程：客户端命令（User、Pass）+服务器响应（+OK，-ERR）
        2. 事务阶段：List（列出消息数量）、Retr（用编号获取消息）、Dele（删除消息）、Quit
        3. “下载并删除”模式：客户端下载后服务器删除邮件，用户更换客户端后无法重读
        4. “下载并保持”模式：客户端下载后服务器不删除邮件，不同客户端都可以保留消息拷贝
        5. POP3是无状态协议：服务器不维护用户状态
     2. IMAP：更新，更加复杂，更多功能，能够操纵服务器上存储的信息
        1. 所有消息统一保存在服务器
        2. 允许用户利用文件夹组织信息
        3. 支持跨会话的用户状态：文件夹的名字、文件夹与消息ID之间的映射等
     3. HTTP：Web Mail的传输协议，也可看作是邮件访问协议

1. Web应用
   1. WWW：Web与Web间互相链接的集合
   2. Web：包含多个对象
      1. 对象：HTML文件、JPEG图片、视频文件、动态脚本等
      2. 基本HTML文件：包含对其他对象引用的链接
      3. 对象寻址：
         1. URL（统一资源定位器）
         2. Scheme：//host:port/path 协议：//主机/路径 若协议不指出则默认为http
            1. <https://www.4399.com/flash/zmhj.htm>
   3. HTTP协议：超文本传输协议（HyperText Transfer Protocol）
      1. C/S结构：客户为Browser，负责请求、接收、展示Web对象；服务器为WebServer，响应客户请求，发送Web对象
      2. HTTP版本：HTTP1.0、HTTP1.1
      3. 使用TCP传输服务：服务器在80端口等待客户（浏览器）发起请求（即创建Socket）
      4. 无状态：服务器不维护任何有关客户端过去所发送请求的信息
      5. HTTP有两类消息：HTTP请求消息、HTTP响应消息
         1. HTTP请求信息：ASCII码，请求行+首部行+额外的回车换行符（表示首部结束）+消息体



* + - 1. HTTP响应消息：ASCII码，状态行+首部行+额外的回车换行符+数据（如请求的HTML文件）
         1. 响应状态代码（即状态行）：200 OK、301 Moved Permanently、304 Not Modified、400 Bad Request、404 Not Found、505 HTTP Version Not Supported…
    1. 上传输入方法（e.g.登录时候向服务器传用户名密码）
       1. POST方法：在请求消息的消息体中上传客户端的输入（e.g.网页填表格）
       2. URL方法：使用GET方法，输入信息通过request行的URL字段上传（一般用于信息少）
          1. e.g.百度搜索哈工大深圳时的HTTP消息：<https://www.baidu.com/s?wd=>哈工大深圳…
    2. HTTP不同版本
       1. HTTP1.0：
          1. 支持GET、POST、HEAD（告诉服务器只返回HTTP头部即可，一般用于测试）
          2. 非持久性连接：每个TCP连接传输最多一个对象，发送响应后服务器关闭TCP连接
          3. 响应时间为2RTT+响应消息传输时间（发起TCP连接、完成HTTP请求各需要1个RTT）

每个对象都要2个RTT，时间性能可能差，且操作系统要为每个TCO连接开销资源

浏览器可以开启多个并行连接获取网页所需对象，但可能给服务器带来过重负载

* + - 1. HTTP1.1：
         1. 支持GET、POST、HEAD、PUT、DELETE（上传到或URL路径删除指定的文件）
         2. 持久连接：每个TCP连接允许传输多个对象，发送响应后服务器保持TCP连接打开

非流水持久连接：客户端只有收到前一个响应后才发送新的请求

每个被引用的对象耗时1个RTT

流水持久连接：HTTP1.1的默认选项，客户端只要遇到一个引用对象就尽快发出请求，等待收到响应

理想情况下收到所有引用对象共耗时1RTT

* + - * 1. 默认FCFS，所有晚来的小对象要等待先来的大对象完成响应传输，平均响应时间长
      1. HTTP/2：HTTP1.1连续请求的响应时间优化，减少多对象请求的时延（响应时间）
         1. 将对象分割为帧，不同对象的帧交替传输
         2. 与HTTP1.1相比，方法、状态码、多数首部行没变
         3. 对象传输顺序基于客户定义的对象优先级
         4. 推送客户未请求的对象
         5. 仍然使用单一的TCP连接，丢包恢复会暂停对象传输
      2. HTTP/3：HTTP/2的实现优化
         1. 基于UDP的应用层协议，提高HTTP性能，部署在Google许多应用上
         2. 在UDP之上实现应用，将传输层功能移到应用层：HTTP/3：QUIC

应用层实现连接建立、错误控制、拥塞控制等功能

连接建立：可靠性、拥塞控制、身份验证、加密、在1个RTT建立状态

差错控制和拥塞控制：采用类似TCP的丢失检测和拥塞控制算法

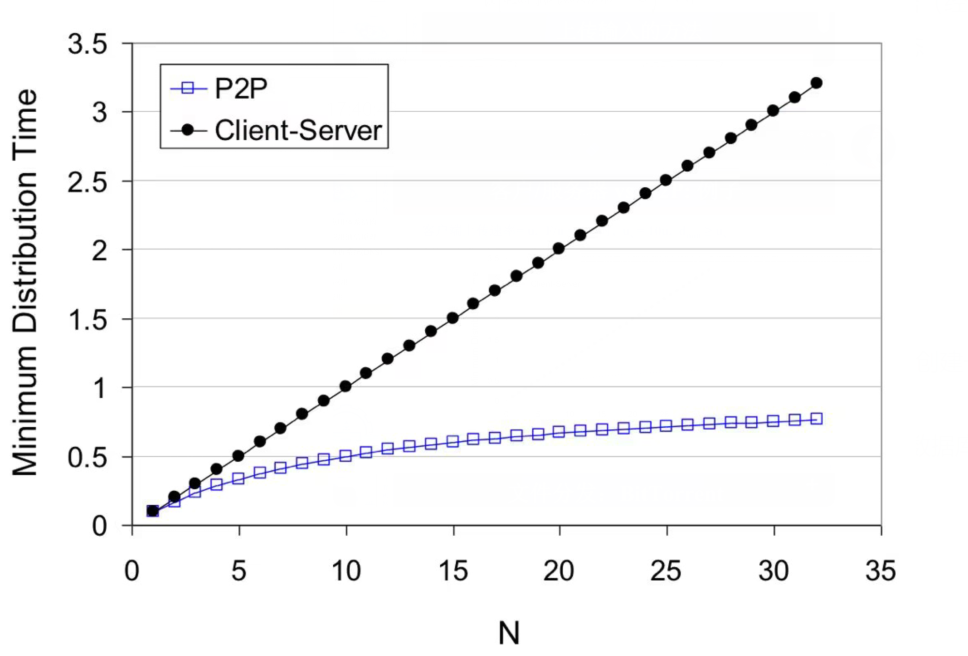
* + - * 1. 多个应用级流复用一个QUIC连接，提高并行性，无HOL阻塞
    1. Web缓存/代理服务器：在不访问源服务器的前提下满足客户端的HTTP请求
       1. 缩短客户请求的响应时间，减少机构/组织的流量，在大范围内实现有效内容分发
       2. 工作过程
          1. 用户设定浏览器通过缓存进行Web访问
          2. 浏览器向缓存/代理服务器发送所有的HTTP请求

若请求对象在缓存中，缓存返回该对象

否则，缓存服务器向原始服务器发送HTTP请求获取对象，将对象返回给客户端并保存

* + - 1. 缓存服务器既充当客户端，也充当服务器，一般由ISP架设
    1. 条件性GET方法：保持缓存与服务器内容版本一致
       1. 缓存：在HTTP请求消息中声明所持有版本的日期
       2. 服务器：
          1. 若缓存的版本是最新的，则响应消息中不包含对象：304 Not Modified；
          2. 否则返回最新版本的对象：200 OK + data
    2. HTTP vs SMTP
       1. 共同点：
          1. 都使用命令/响应交互模式
          2. 命令和状态码都是ASCII码
       2. HTTP：
          1. 拉式（pull）：由客户端从服务器拉取信息，由接收者发起TCP连接
          2. 每个对象封装在独立的响应消息中
          3. 持久性连接，也可使用非持久性连接
       3. SMTP：
          1. 推式（push）：由客户端向服务器推送邮件信息，由发送者发起TCP连接
          2. 多个对象在多个部分构成的消息中发送
          3. 持久性连接
          4. 消息必须由7位ASCII码构成
          5. 服务器利用 CRLF.CRLF确定消息的结束
  1. Cookie：Web的功能优化
     1. 某些网站为了辨别用户身份、进行session跟踪而存储在用户本地终端上的数据（经常经过加密）
     2. 动机：很多应用需要服务器需要掌握客户端状态，如网上购物，但HTTP无状态
     3. Cookie的组件
        1. HTTP**响应**消息的cookie头部行
        2. HTTP**请求**消息的cookie头部行
        3. 保存在**客户端主机**上的cookie文件，由浏览器管理
        4. **Web服务器端**的后台数据库
     4. 工作过程：
        1. 客户端发送HTTP请求，服务器根据请求消息中头部行判断是新用户，为其创建一个ID，封装到响应消息的头部行中并发送响应，同时为用户在数据库中创建条目
        2. 客户端发送带有cookieID的HTTP请求，服务器返回响应并根据其请求修改数据库相应的条目
     5. 作用：身份认证、购物车、推荐、Web email…
     6. 问题：隐私问题

1. P2P应用
   1. 文件分发：C/S vs P2P
      1. 使用C/S架构将一个大小为F的文件分发给N个客户的时间随N线性增加
      2. 使用P2P架构将一个大小为F的文件分发给N个客户的时间随着N线性增加，然而每个对等体提供了新的上传带宽



* 1. BitTorrent：典型的P2P文件分发应用
     1. torrent（洪流）：交换同一个文件的文件块的节点组，动态变化——节点可能加入或离开
     2. tracer（追踪器）：追踪参与torrent的节点
     3. 将文件划分为256KB的文件块chunk
     4. 节点加入torrent：
        1. 向tracer注册以获得节点清单，同时与某些节点（一般是邻居）建立连接
        2. 初始没有文件块，但会慢慢积累
     5. 下载的同时，节点需要向其他节点上传文件块chunk
     6. 一旦节点获得完整的文件，它可以（自私地）离开或（无私地）留下
     7. 获取文件块
        1. 给定任一时刻，不同的节点持有文件的不同文件块chunk集合
        2. 节点会定期查询每个邻居持有的文件块列表并发送请求获取文件块chunk，稀缺优先
     8. 发送文件块：一报还一报（tit-for-tat）
        1. 节点**每10秒重新评估**正在向其发送文件块chunk速率最快的top4，向他们发送文件块chunk
        2. **每30秒随机选择**一个**其他**节点，向其发送文件块：“优化疏通”
           1. 新选择的节点后来也可能加入top4
     9. 上传速率高，则能找到更好的交易伙伴，更快地获取文件
  2. P2P：搜索信息
     1. P2P系统的索引：信息到节点位置（IP地址+端口号）的**映射**
     2. 文件共享（电驴）：利用索引**动态跟踪**节点共享的文件位置，节点需告知索引他拥有的文件
     3. 即时消息（QQ）：索引负责将用户名映射到位置，用户开启即时消息时需通知索引它的位置
     4. **集中式索引**：部分分布式与集中式结合
        1. 节点加入时，通知中央服务器其**IP地址**和**内容**
        2. 节点在中央服务器中查找文件和位置
        3. 向查找到的位置请求文件（最近节点优先）
        4. Napster的最早设计
        5. 内容和文件传输是分布式的，但内容定位是高度集中式的，会带来集中式的经典问题
     5. **泛洪式查询**：完全分布式
        1. 每个结点对它共享的文件进行索引且**只对它共享的文件**进行索引
        2. 覆盖网络：节点之间有TCP连接则构成一个边，所有的活动节点和边构成覆盖网络
        3. 查询消息通过已有的TCP连接发送
           1. **节点转发查询消息**
           2. 查询命中则利用反向路径发回查询节点
           3. 网络负担大，加入节点也复杂
     6. **层次式覆盖网络**：介于集中式索引和泛洪式查询之间的方法
        1. 节点要么是超级节点，要么被分配一个超级节点
        2. 节点和超级节点间维持TCP连接，某些超级节点对之间维持TCP连接
        3. 每个超级节点负责跟踪子节点的内容
        4. 超级节点间采用分布式的泛洪式查询，超级节点和子节点之间采用集中式索引
  3. Skype：P2P案例应用
     1. 本质上是P2P的，用户/节点对之间直接通信
     2. 采用私有应用层协议和层次式覆盖网络
     3. 索引分布在超级节点上，负责维护用户名和IP地址间的映射
  4. 分布式哈希表（DHT）：一种分布式P2P数据库
     1. 数据库存储（key，value）对，一个节点通过key对DHT进行查询、插入
     2. （key，value）对分布在数百万个节点上，如何分配？
        1. 将每个key用某种**哈希**方法转换为整数，为每个结点分配一个整数ID
        2. 将（key，value）对存放到ID与其key距离最近的结点中
        3. 可以将n位数的标识符分配给节点，要求每个key是同一范围内的整数
        4. 规定：“最近”值是key的直接继任者，可以考虑环形
     3. 环形DHT：
        1. 每个节点只知道直接的后继和前驱，定义“最近”为最接近的后继
        2. 当有N个结点时，查询的平均复杂度是O（N）
     4. 带捷径的环形DHT：
        1. 每个节点跟踪直接前驱、后继、**捷径**的IP地址
        2. 可以设计与O（logN）节点建立捷径，使得每次查询为O（logN）
        3. 每个节点多存信息，空间换时间
     5. 节点离开
        1. 每个节点知道其后继节点的地址，**定期ping其两个后继节点**检查其是否存活
        2. 若直接后继离开，选择其下一个后继节点作为新的后继节点

1. Socket编程（略）