哈工大计算机网络Week1-网络应用

2.1网络应用的体系结构

特点

由不同的部分构成，由网络连接组合应用场景

应采取什么结构

C/S结构 客户机/服务器

P2P

混合结构

C/S结构 客户机/服务器

- 服务器
  - **持续提供服务**
  - **固定的访问地址/域**
  - **利用大量服务器实现可扩展性**
- 客户机
  - 与服务器通信，使用服务器提供的服务
  - 间歇性接入网络
  - 可能使用动态IP地址
  - 客户机之间无通信能力

如何定义服务器和客户机？会话的发起方是客户，会话开始前的等待方是服务器。

P2P

点对点服务，去中心化体系结构。**中心服务器的依赖很低，但是仍然由中心服务器。**纯P2P中，资源索引是维护在每个节点上的，查找信息时需要在网络中进行广播，查询被一层层节点扩散，直到查找到有效信息。

- 描述
  - 没有永远在线的服务器
  - 任意端系统/节点之间可以直接通讯
  - 节点间歇性接入网络
  - 节点可能改变IP地址
- **自拓展性**
  - 新加入的对等方又为整个网络增强服务能力。
- 挑战
  - ISP友好：目前的ISP是非对称的，不适合P2P架构
  - 安全性：高度分布和开发带来安全问题
  - 激励：鼓励用户来自愿为应用提供带宽、存储、计算

CS vs P2P

从服务及时、传输质量稳定、资源查找速度、管理成本、网络安全、客户端节点间交流能力、网络闲散资源利用率

- CS>P2P
  - CS更安全，隐私不容易泄露，因为客户机间不可见，客户机只能访问由服务器暴露的经其他客户机允许的其他客户机的信息，服务器能轻松的设置安全机制。P2P则不能，个人计算机安全等级较服务器来说低很多。可能带来垃圾信息或者匿名失效。
  - CS服务随时支持，P2P则需要提供服务的节点在线才行。
  - CS服务稳定，因为服务器通常是性能强大的设备群。而P2P受制于服务器节点，稳定性不可预见。
  - CS中心式结构便于集中管理资源以及客户端，P2P结构松散不便于管理资源。
  - CS文件查找速度快，P2P需要将查找命令在P2P网络中层层传播，比较慢
- P2P>CS
  - P2P能充分利用节点资源，提高限制资源利用率。CS无法充分利用客户机的能力。使得整个网络负担过多集中在服务器，客户机性能冗余。
  - P2P管理成本低，虽然难以有效管理但是去中心化的结构使得管理成本低。CS则要求中心服务器的管理软硬件要求高。

- 结构自由。CS结构限制较大。
- P2P网络分布广泛，能够大范围提供服务。C-S距离太远延迟高，每个服务器辐射的范围有限，需要采用分布式多服务器来提供服务。

混合结构

能否将两种结构混合在一起使用？ 混合能够利用两者的优点同时规避两者的缺点吗？

目前的混合结构主要有2种：

- **中心化拓扑P2P**（例如Napster）
  - 文件传输使用P2P结构
  - 文件的搜索采用C/S结构——集中式
    - 每个节点向中央服务器登记自己的内容
    - 每个节点向中央服务器提交查询请求，查找感兴趣的内容
- **半分布式拓扑结构P2P**
  - 节点分成索引节点和普通节点，索引节点负责为临近普通节点提供服务。
  - 各个索引节点之间采用去中心化结构
- **全分布式结构化拓扑的P2P网络**主要是采用分布式散列表（Distributed Hash Table, 简写成DHT）技术来组织网络中的结点。DHT是一个由广域范围大量结点共同维护的巨大散列表。散列表被分割成不连续的块，每个结点被分配给一个属于自己的散列块，并成为这个散列块的管理者。通过加密散列函数，一个对象的名字或关键词被映射为128位或160位的散列值。
  - 即对目录进行了分割，使得目录被结构化组织

思考题目

为每种体系结构找出5种以上的网络应用：

从多个方面/角度对比三种体系结构的优缺点

2.2网络应用的基本原理

网络应用进程通信

进程间通信

- 进程
  - 主机上运行的程序
  - **客户机进程**: 发起通信的进程
  - **服务器进程**: 等待通信请求的进程
- 同一主机上运行的进程之间如何通信？
  - 进程间通信机制
  - 操作系统提供

- 不同主机上运行的进程间如何通信？
    - 消息交换

采用P2P架构的应用是否存在客户机进程/服务器进程之分？

也是有的，只是角色转换非常自由。

套接字: Socket

- Socket（套接字、插座）并不是一种协议，而是一种抽象API，将传输层TCP/UDP协议封装，作为一种门面模式提供应用层抽象的连接服务。Socket实质上提供了进程通信的端点。
- Socket=（所用协议，本地IP地址，本地进程端口号），唯一，由本地操作系统分配
- Socket面向C/S设计。
- C/S连接建立过程
    - 服务器监听：是**服务器端套接字**并不定位具体的**客户端套接字**，而是处于等待连接的状态，实时监控网络状态。
    - 客户端请求：是指由**客户端的套接字**提出连接请求，要连接的目标是**服务器端的套接字**。为此，客户端的套接字必须首先描述它要连接的服务器的套接字，指出服务器端套接字的地址和端口号，然后就向服务器端套接字提出连接请求。
    - 连接确认：是指当服务器端套接字监听到或者说接收到客户端套接字的连接请求，它就响应客户端套接字的请求，建立一个新的线程，把服务器端套接字的描述发给客户端，一旦客户端确认了此描述，连接就建立好了**。而服务器端套接字继续处于监听状态，继续接收其他客户端套接字的连接请求。**

寻址

不同主机上的进程间通信，那么每个进程必须拥有标识符

- 如何寻址主机？——IP地址
    - Q: 主机有了IP地址后，是否足以定位进程？
    - **A: 否。IP地址只能定位主机，同一主机上可能同时有多个进程需要通信。**
- **端口号/Port number**
    - 为主机上**每个需要通信的进程**分配一个端口号
    - **HTTP Server: 80**
    - **Mail Server：25**
- 进程的标识符
    - **IP地址+端口号**

应用层协议

网络应用需遵循应用层协议,但是不仅仅有应用层协议

- 公开协议（由RFC定义）
    - DNS
    - FTP
    - SMTP
    - HTTP
    - Telnet（远程登陆服务）

- 私有协议
    - 多数P2P文件共享应用
- 应用层数据传输依赖传输层协议：
    - tcp
    - udp

应用层协议的内容

信息类型、信息语法、信息语义、信息规则

- 消息的类型(**type**)
    - 请求消息
    - 响应消息
- 消息的语法(**syntax**)/格式
    - 消息中有哪些字段(**field**)？
    - 每个字段如何描述
- 字段的语义(**semantics**)
    - 字段中信息的含义
- 规则(**rules**)
    - 进程**何时**发送/响应消息
    - 进程**如何**发送/响应消息

网络应用的需求与传输层服务

网络应用对传输服务的需求

- **数据丢失**(data loss)/可靠性(reliability)
    - 某些网络应用能够容忍一定的数据丢失：网络电话
    - 某些网络应用要求100%可靠的数据传输：文件传输，telnet
- 时间(timing)/**延迟**(delay)
    - 有些应用只有在延迟足够低时才“有效”
    - 网络电话/网络游戏
- **带宽**(bandwidth)
    - 某些应用只有在带宽达到最低要求时才“有效”：网络视频
    - 某些应用能够适应任何带宽——弹性应用：email
- **延迟和带宽的不同**：
    - 带宽是bps，是单位时间交换数据量，延迟是数据发送到接受所用时间，和带宽不是一个概念。
    - 带宽小于发送量会导致延迟上升。带宽小于发送量仍然有延迟，此时延迟是数据从源发送到目的所需时间

Internet提供的传输服务

- TCP服务，传输控制协议（英语：Transmission Control Protocol，縮寫為**TCP**）

    - 面向连接: 客户机/服务器进程间需要建立连接。
    - 可**靠的数据传输：准确、完整、有序。注意没有带宽和延迟！**
        - 准确：保证数据包没有失真
            - 校验码
        - 完整：降低丢包率
            - 接受反馈：接受发接受到数据必须告知发送方，否则发送方会重发，保证分组完整
            - 流量控制: 发送方不会发送速度过快，超过接收方的处理能力
            - 拥塞控制: 当网络负载过重时能够限制发送方的发送速度
        - 有序：保证分组重组顺序正确
            - 为分组标记序号。接收方使用序号，不重收，不乱序。
    - **不提供时间/延迟保障**
    - **不提供最小带宽保障**
    - 握手三次
        - A申请建立
        - B回复批准
        - A回复收到批准
    - 挥手四次
        - A申请结束
        - B回复收到申请，并发送尚未传输完成的数据
        - B批准申请
        - A回复收到批准
- UDP服务，用户数据报协议（英语：User Datagram Protocol，縮寫為**UDP**），又稱使用者資料包協定
    - 无连接，单纯的发送出去
    - 不可靠的数据传输（单纯的发送出去，什么保障都没有）
    - 不提供
        - 可靠性保障
        - 流量控制
        - 拥塞控制
        - 延迟保障
        - 带宽保障
    - 正是由与udp如此低级，他只能支持最基本的功能，但是它提供了掌控数据传输的自由性。可以让应用自由设置发挥
- 应用场景：
    - 稳定>速度：tcp
        - HTTP、FTP、POP、SMTP
    - 速度>稳定：udp
        - 视频、语音通话

课后练习

- 盘点你计算机上的所有网络应用，制作一个清单，包括网络应用的名字、功能、协议等。

- 基于上述清单，制作表格，分析这些网络应用对传输服务的需求。 分析这些网络应用所使用的传输服务是TCP还是UDP。

2.3Web应用

web应用概述

Web与HTTP

- web万维网是基于HTTP的Inernet因特网**网络服务**。
- Internet将资源组织起来形成网络，web的功能是为用户提供**资源索引展示服务**。
- web将资源以**URL定位**，**使用超文本和超链接将资源连接起**来，是一个**由许多互相链接的超文本组成的系统**。
- **超文本**是包含超链接的文本，允许从当前阅读位置直接切换到超链接所指向的文本、图像、音视频、文件等等。超链接就是指向其他资源的连接。
- 网页：超文本（如html）经浏览器解析渲染展现为网页
- 网页互相链接：（准确的说是网页的源相互连接）
- 网页(Web Page)包含多个对象(objects)
    - 基本HTML文件：包含对其他对象引用的链接
    - 对象：HTML文件、JPEG图片、视频文件、动态脚本等
- **对象的寻址(addressing)**
    - URL(Uniform Resoure Locator)：统一资源定位器 RFC1738
        - 格式： Scheme://host:port/path
        - 例如：http://www.hit.edu.cn/header/article.txt

HTTP协议概述

HTTP，原名超文本传输协议，HyperText Transfer Protocol，是web万维网遵守的协议之一。功能是：建立web服务器和本地客户端（通常是浏览器）之间的数据交换。

- **端口号：80**
- **http网络结构：C/S结构**
    - 客户—Browser：请求、接收、展示Web对象
    - 服务器—Web Server：响应客户的请求，发送对象
- HTTP版本：
    - 1.0： RFC 1945
    - 1.1： RFC 2068
- **http传输层协议：TCP传输协议**
    - 服务器在80端口等待客户的请求
    - 三次握手后，客户端浏览器建立起到服务器的TCP连接

    - 浏览器(HTTP客户端)与Web服务器(HTTP服务器)交换HTTP消息
    - 四次握手，关闭TCP连接
- **无状态(stateless)**，即无记录
    - 服务器不记录任何有关客户端过去所发请求的信息
        - 有状态的协议**更复杂**:
            - 需维护状态(历史信息)
            - 如果客户或服务器失效，会产生状态的不一致，**解决这种不一致代价高**
    - web通过cookie、session技术来实现状态存储功能。

HTTP连接

HTTP连接的两种类型

- **非持久性连接(NonpersistentHTTP)**
    - **每个TCP连接最多允许传输一个对象**
    - HTTP 1.0版本（早期）使用非持久性连接
- **持久性连接(Persistent HTTP)**
    - **每个TCP连接允许传输多个对象**
    - HTTP 1.1版本默认使用持久性连接

非持久性连接

流程

假定用户在浏览器中输入URLwww.someSchool.edu/someDepartment/home.index（包含文本和指向10个jpeg图片的链接）

1. 服务器等待TCP连接中
2. 客户端使用socket申请tcp连接。
3. 三次握手完成建立，在第三次握手同时传输Http请求信息。
4. HTTP服务器收到请求消息，解析，产生包含所需要对象的响应消息，并通过套接字发给客户端。HTTP服务器通知tcp连接断开。（不过客户端收到响应后才真正断开）
5. 当客户端收到响应，tcp连接断开，解析html文件，显示html文件，发现有10个指向jpeg对象的超连接。
6. 为每个jpeg对象重复步骤1-5。而不能接着上次的TCP连接继续交流。

时延

- **单位：RTT(Round Trip Time)**
    - 从客户端发送一个**很小**的数据包到服务器**并返回**所经历的时间
- **响应时间(Response time)**
    - tcp三次握手前两次：1RTT
    - tcp三次握手第三次+发送HTTP请求消息：0.5RTT，这里之所以不是1RTT是因为tcp三次握手第三次+发送HTTP请求消息同时发送。而不是等但三次握手完成再发送请求。
    - HTTP接受请求--HTTP响应消息的前几个字节到达：0.5个RTT
    - 响应消息中所含的文件/对象传输时间
    - **Total=2RTT +文件发送时间**
- 延时如何改进：

- 对于非持久连接，通过并行连接改进延迟

非持久性连接的问题

- **每个对象需要2个RTT**：1+0.5+0.5
- 操作系统需要为每个TCP连接**开销资源(overhead)**，同时刻创建大量的连接。
- 浏览器会怎么做？
  - 打开多个并行的TCP连接以获取网页所需对象
  - 给服务器端note造成恶劣的影响，**高密度tcp请求耗尽资源。**

持久性HTTP

- 持久性连接
  - **发送响应后，服务器保持TCP连接的打开**
  - 后续的HTTP消息可以通过这个连接发送
- **无流水(pipelining)的持久性连接**
  - **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
- **带有流水机制的持久性连接**
  - HTTP 1.1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**
  - 理想情况下，收到所有的引用对象只需耗时约1个RTT
- 这里计算延迟都是忽略了连接建立时间的1RTT！！只考虑申请发送+响应的1RTT！

HTTP消息格式

- HTTP协议有两类消息
  - 请求消息(request)
  - 响应消息(response)

HTTP请求消息

请求消息 使用ASCII书写：人直接可读

GET /somedir/page.html HTTP/1.1 请求行： 方法+url+http版本,注意url只有后面的索引部分

Host: www.someschool.edu 头部行

User-agent: Mozilla/4.0 头部行

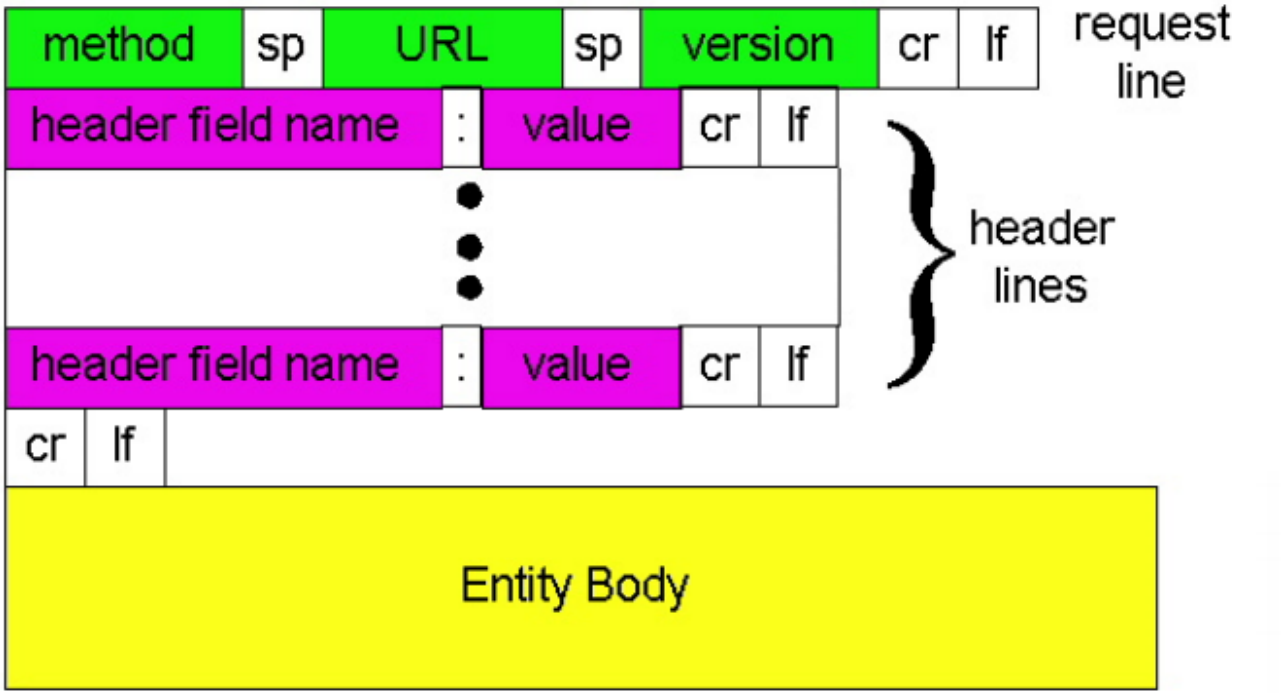
Connection: close 头部行

Accept-language:fr 头部行

Cookies 头部行

空行（非常重要，讲请求头和消息体分开）

消息体，一般包括要填的表单



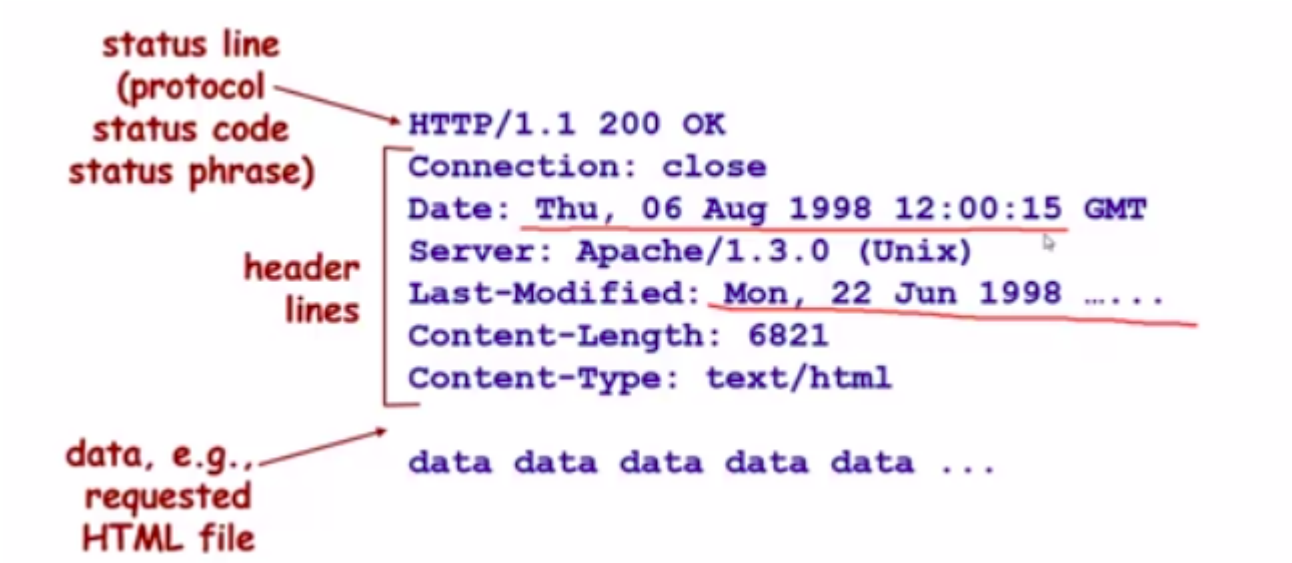
上传附加输入信息的方法

- POST方法
  - 网页经常需要填写表格(form)
  - 在请求消息的消**息体(entity body)**中上传客户端的输入，entity body在上述的name：value部分之后
- GET方法
  - 使用URL传输，在url后link信息
  - 输入信息通过request行的URL字段上传

方法的类型

- HTTP/1.0
  - GET
  - POST
  - HEAD
    - 请Server不要将所请求的对象放入响应消息，常常用来做测试
- HTTP/1.1
  - GET, POST, HEAD
  - PUT
    - 将**消息体**中的文件上传到URL字段所指定的路径
  - DELETE
    - 删除URL字段所指定的文件

HTTP响应消息



第一行叫状态行

date是生成该响应消息的时间，last-modified是请求的服务器资源（比如某个文件）最后的修改时间。

响应头下一个空行与响应正文分割，响应正文就是申请的内容

HTTP响应状态代码

 响应消息的第一行

 示例 200 OK 301 Moved Permanently（资源已被永久改变位置） 400 Bad Request 404 Not Found 505 HTTP Version Not Supported

Cookie技术

为什么需要Cookie？

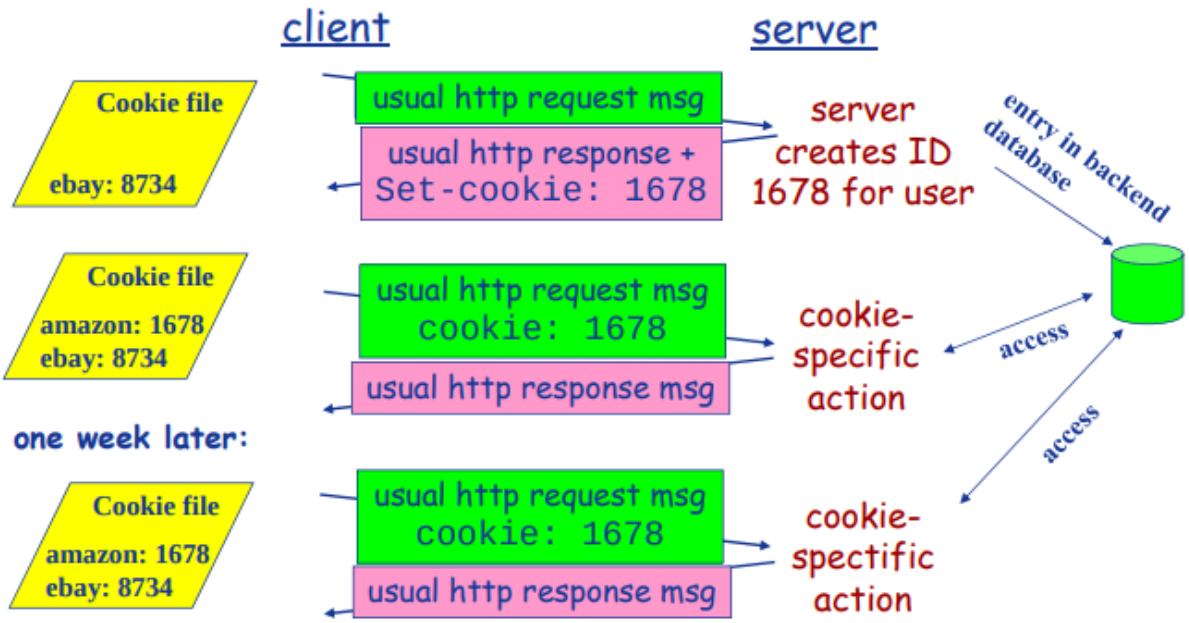
- **HTTP协议无状态，但是很多应用需要服务器掌握客户端的状态**，如网上购物，如何实现？

Cookie技术

- Cookie技术
  - 某些网站为了**辨别用户身份、进行session跟踪**而**储存在用户本地终端**上的数据（通常经过加密）。
  - RFC6265
- **Cookie的组件**
  - HTTP响应消息的cookie头部行
  - HTTP请求消息的cookie头部行
  - 保存在客户端主机上的cookie文件，由浏览器管理
  - **Web服务器端的后台数据库**
- Cookie和session
  - cookie存储在客户端，持续时间较长

- session在服务端，持续时间较短，一般只在会话期间和会话结束后一小段时间内存在。

Cookie的原理



//TODO ################################

技术细节没有涉及到，此处要补充，包括session技术，以及cookie验证机制

Cookie的作用

- Cookie能够用于：
  - 身份认证
  - 购物车
  - 推荐
  - Web e-mail
- 隐私问题
  - cookie虽然经加密但不是没有破解可能，但是客户端安全性不高，一旦cookie被恶意获取，隐私很可能泄露
  - 网站获应用要求用户提打开cookie功能才能正常访问，暗中记录用户访问信息。

思考题

1. Cookie能够怎样被用于收集隐私？
2. 能够收集哪些隐私？
3. 你在上网的时候感觉到自己的隐私
4. 被严重侵犯吗？

Web缓存/代理服务器技术

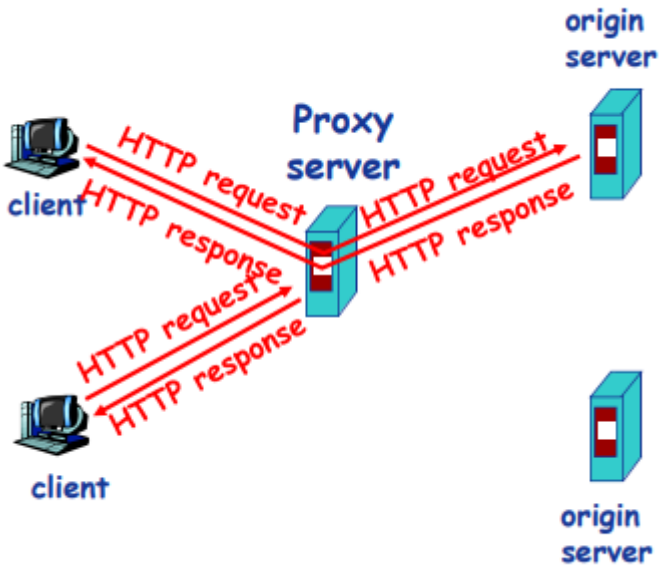
功能

在不访问服务器的前提下满足客户端的HTTP请求。

为什么要发明这种技术？

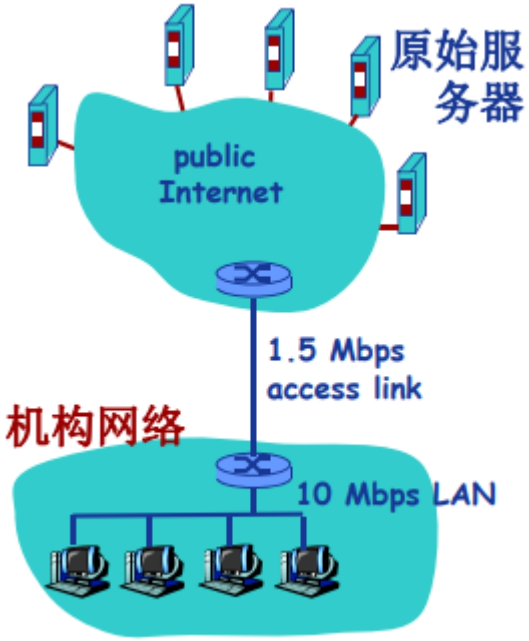
- 缩短客户请求的响应时间
- 减少机构/组织的流量
- 在大范围内(Internet)实现有效的内容分发

Web缓存/代理服务器



- 用户设定浏览器通过缓存进行Web访问
- 浏览器向缓存/代理服务器发送所有的HTTP请求
  - 如果所请求对象在缓存中，缓存返回对象
  - 否则，缓存服务器向原始服务器发送HTTP请求，获取对象，然后返回给客户端并存该对象
- 缓存既充当客户端，也充当服务器
- 一般由ISP(Internet服务提供商)架设

Web缓存示例



- 假定：
  - 对象的平均大小=100,000比特
  - 机构网络中的浏览器平均每秒有15个到原始服务器的请求
  - 从机构路由器到原始服务器的往返延迟=2秒
- 网络性能分析：
  - 局域网(LAN)的利用率=1.5M/10M=15%
  - 接入互联网的链路的利用率=100%
  - 总的延迟=互联网上的延迟+访问延迟+局域网延迟=2秒+几分钟+几微秒

- 解决方案1：
  - **提升互联网接入带宽**=10Mbps
- 网络性能分析：
  - 局域网(LAN)的利用率=15%
  - 接入互联网的链路的利用率=15%
  - 总的延迟=互联网上的延迟+访问延迟+局域网延迟=2秒+几微秒+几微秒
- 问题：
  - **成本太高**

- 解决方案2：
  - 安装Web缓存
  - **假定缓存命中率是0.4**
- 网络性能分析：
  - 40%的请求立刻得到满足
  - 60%的请求通过原始服务器满足
  - 接入互联网的链路的利用率下降到60%，**从而其延迟可以忽略不计**，例如10微秒
  - 总的平均延迟=互联网上的延迟+访问延迟+局域网延迟=0.6×2.01秒+0.4×min微秒<1.4秒

- 问题：
  - 缓存和远端服务器资源是否一致？或者版本是否满足用户要求？

web缓存是如何降低延时的？

注意web不是直接降低的。而是通过缓存，使得命中的部分能直接本地解决，不需要向远方的服务器进行传输。而对于未命中的部分，是因为命中部分减少了局域网和因特网之间的流量，改善了拥塞程度而使得排队延迟降低的。

条件性GET方法

解决了缓存和远端服务器资源是否一致？或者版本是否满足用户要求？的问题。

最重要：条件get是代理服务器向原始服务器发送！

- 目标：
  - 代理服务器向远端服务其的get请求进行设计，告知远端服务器该代理所持资源版本，远端服务器检查后，若是最新版则不发送请求的对象。
- 缓存：
  - 在HTTP请求消息中声明所持有版本的日期
  - If-modified-since:（告诉远端，如果在**日期之后远端资源更改，则发送所请求对象）
- 服务器：
  - 如果缓存的版本是最新的，则响应消息中不包含对象
  - HTTP/1.0 304 Not Modified

课后作业

检索文献，分析、总结Web技术近 年来有哪些新进展？其关键思想和 技术是什么？

2.4Email应用

Email应用

Email应用的构成

- Email应用的构成组件
  - 邮件客户端(user agent)：外围
    - 读、写Email消息
    - 与服务器交互，收、发Email消息
    - Outlook, Foxmail, Thunderbird
    - Web客户端
  - 邮件服务器：核心
    - 邮箱：存储发给该用户的Email
    - **消息队列(message queue)：存储等待发送的Email**（其存在的主要意义）
  - SMTP协议(Simple Mail TransferProtocol)，简单邮件传输协议

- 邮件服务器之间传递消息所使用的协议
- 客户端：发送消息的**服务器**
- 服务器：接收消息的服务器

实现异步发送，自动重试，在线缓存，延时发送等等功能。

SMTP协议: RFC 2821

SMTP是一个**“推”的协议，**它不允许根据需要从远程服务器上“拉”来消息。要做到这点，邮件客户端必须使用POP3或IMAP。

- 使用TCP进行email消息的可靠传输
- **端口25**
- 传输过程的三个阶段
  - tcp握手
  - 消息的传输
  - tcp挥手
- 命令/响应交互模式
  - 命令(command): ASCII文本
  - 响应(response): 状态代码和语句
- **Email消息只能包含7位ASCII码，因为email系统开发是非常早期的时代**
- **stmp一般不使用中间服务器来转发邮件，发送方服务器-接受方服务器是通过传输层直接发送的，即使两者离得非常之远！**

与HTTP对比

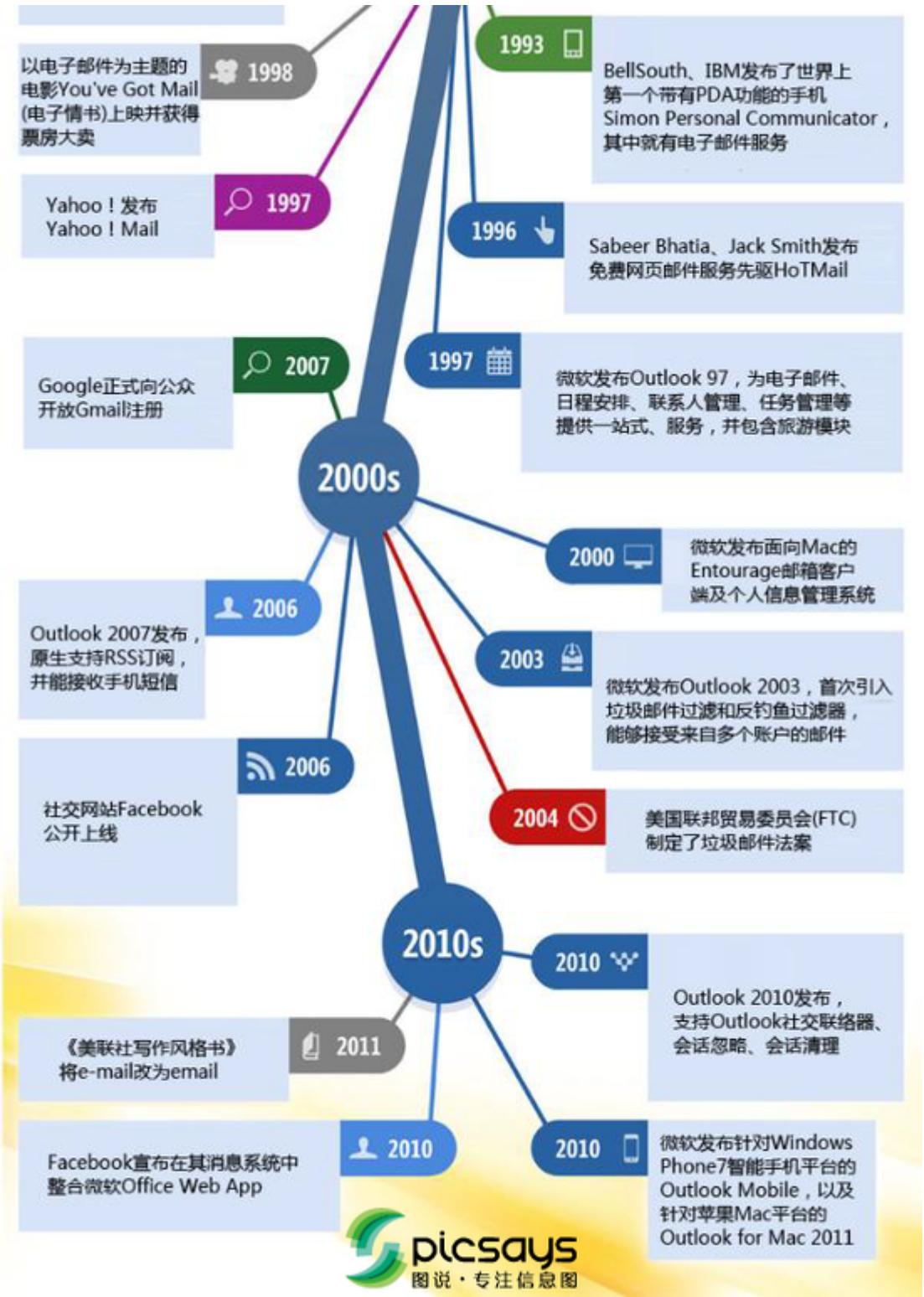
- **HTTP: 拉式(pull)，SMTP: 推式(push)**
- **http不必要是Ascii**
- **HTTP: 每个对象封装在独立的响应消息中**
- **SMTP: 多个对象在由多个部分构成的消息中发送**

动手尝试SMTP交互

- telnet servername 25
- 服务器返回代码220
- 输入以下命令与SMTP服务器交互
  - **HELO：发送方服务器向接收方服务器申请连接**
  - **MAIL FROM**
  - **RCPT TO**
  - **DATA：内容**
  - **QUIT**

思考题

Email作为互联网上的古老应用，从 出现至今经过了什么样的演变过程？ 站在今天的角度看，Email应用有哪 些缺点和不足？请查阅资料，给出你 的见解。



Email消息格式与POP3协议

Email消息格式

- SMTP：email消息的传输/交换协议
- RFC 822：文本消息格式标准

    - **头部行(header)**
        - **To**
        - **From**
        - **Subject（主题）**
    - **空行**
    - **消息体(body)**
        - **消息本身（只能是ASCII字符）**

Email消息格式：多媒体扩展

- MIME：多媒体邮件扩展 RFC 2045, 2056
    - 通过在邮件头部增加额外的行以声明MIME的内容类型

邮件访问协议（不同与STMP协议）

邮件访问协议：从服务器获取邮件。 Mail Access Protocol

EMAIL应用使用多个协议完成功能

- POP3: Post Office Protocol [RFC 1939]
    - 认证/授权(客户端<-->服务器)和下载
- IMAP: Internet Mail Access Protocol [RFC 1730]
    - 更多功能
    - 更加复杂
    - 能够操纵服务器上存储的消息
    - 趋势
- HTTP：163, QQ Mail等。也可以当作一种邮件访问协议

POP3协议

- 认证过程（两次握手）
    - 客户端命令
        - User：声明用户名
        - ass: 声明密码
    - 服务器响应
        - +OK
        - -ERR
- 事务阶段
    - List：列出消息数量
    - Retr：用编号获取消息（这个是本阶段完成）
    - Dele: 删除消息（在更新阶段才被真正删除，这里只是标记）
    - Quit
- 更新阶段
    - quit后，在事物阶段被标记删除的文件被真正删除

- “下载并删除”模式，下载到本地，删除服务器上的信息。POP协议仅支持该模式。
    - 用户如果换了客户端软件，无法重读该邮件
- “下载并保持”模式：不同客户端都可以保留消息的拷贝。**POP3支持，POP不支持**
- **POP3是无状态的**。客户端的动作不能保存到服务器上。比如删除、移动文件

IMAP协议

- 有状态协议，**支持CS双向通信**
    - 所有消息统一保存在一个地方：服务器
    - 允许用户利用文件夹组织消息，**删除移动文件服务器会同步**
- IMAP支持跨会话(Session)的用户状态:
    - 文件夹的名字
    - 文件夹与消息ID之间的映射等
- IMAP更好地支持了从多个不同设备中随时访问新邮件。
- IMAP提供的**摘要浏览功能**可以让你在阅读完所有的邮件到达时间、主题、发件人、大小等信息后才作出是否下载的决定。

Email流程示例

对host：a、b，a发SMTP送邮件m给a的邮件服务器A，A遵循STMP将m发送给b的邮件服务器B，**b使用access protocol来获取B上的信件**

课后练习

请查阅资料，比较IMAP与POP3的 不同，并调研主流Email服务对 IMAP协议的支持情况。

1、IMAP提供Webmail 与电子邮件客户端之间的**双向通信，客户端收取的邮件仍然保留在服务器上，同时在客户端上的操作都会反馈到服务器上**（如：删除邮件，标记已读等，服务器上的邮件也会做相应的动作。所以无论从浏览器登录邮箱或者客户端软件登录邮箱，看到的邮件以及状态都是一致的。）。**而POP3在客户端的操作不会反馈到服务器上。**

2、I**MAP更好地支持了从多个不同设备中随时访问新邮件。**

3、IMAP提供的**摘要浏览功能**可以让你在阅读完所有的邮件到达时间、主题、发件人、大小等信息后才作出是否下载的决定。

4、POP3需要下载未阅读的**邮件，IMAP可以不用把所有的邮件全部下载，而是通过客户端直接对服务器上的邮件进行操**作。所有通过IMAP传输的数据都会被加密，从而保证通信的安全性。

5、IMAP 整体上为用户带来更为便捷和可靠的体验。**POP3 更易丢失邮件或多次下载相同的邮件。**

2.5DNS服务

DNS概述

DNS：Domain Name System

- Internet上主机/路由器的识别问题
  - IP地址
  - 域名：www.hit.edu.cn。是为了方便记忆IP，是面向人类的，IP地址是面向计算机的。
- 域名解析系统DNS：解决问题：域名和IP地址之间如何映射？
  - **多层**命名服务器构成的**分布式**数据库
  - DNS本身也是一个应用层协议
    - • DNS提供Internet核心功能，但是在**应用层使用用应用层协议实现**
    - • 网络边界复杂
- DNS服务具体内容
  - 域名向IP地址的翻译
  - 主机别名
  - 邮件服务器别名
  - 负载均衡：Web服务器，例如轮换web服务器排名
- 问题：为什么不使用集中式的DNS？
  - 单点失败问题
  - 流量问题
  - 距离问题
  - 维护性问题

分布式层次式数据库

以迭代查询为例：

- 客户端想要查询www.amazon.com的IP
  - 先向本地DNS申请查询
    - 本地dns查看缓存，有则返回无则继续查询
  - 本地dns查询**根服务器**，找到**com域名解**析服务器
  - 本地dns**查询com域名**解析服务器，**找到amazon.com域名**解析服务器
  - 本地dns查询**amazon.com域名解**析服务器，获**得www.amazon.com的IP**地址
  - 本地dns缓存www.amazon.com的IP，并发送结果给客户端

DNS根域名服务器

- **根域名服务器**（root name server）是互联网域名解析系统（DNS）中**最高级别**的域名服务器，负责返回**顶级域的权威域名服务器地址。**

  > 在根域名服务器中虽然没有每个域名的具体信息，但储存了负责每个域（如.com,.xyz,.cn,.ren,.top等）的解析的域名服务器

- 本地域名解析服务器无法解析域名时，有一个递归访问过程，起点是根域名解析器
- 全球由13个root dns，中国没有。

> 由于DNS和某些协议（未分片的用户数据报协议（UDP）数据包在IPv4内的最大有效大小为512字节）的共同限制，根域名服务器地址的数量被限制为13个。幸运的是，采用任播技术架设镜像服务器可解决该问题，并使得实际运行的根域名服务器数量大大增加。截至2017年11月，**全球共有800台根域名服务器在运行。**

TLD和权威域名解析服务器

- 顶级域名服务器(TLD, top-level domain): 负责com, org, net,edu等顶级域名和国家顶级域名，例如cn, uk, fr等
    - Network Solutions维护com顶级域名服务器
    - Educause维护edu顶级域名服务器
- 权威(Authoritative)域名服务器：权处于DNS服务端的一套系统，该系统保存了某个响应域名的权威信息。权威DNS即通俗上“这个域名我说了算”的服务器。
    - 域名所有者（通常不是个人而是一个组织）负责维护

本地域名解析服务器

- 不严格属于层级体系
- 每个ISP有一个本地域名服务器
    - 默认域名解析服务器
- **当主机进行DNS查询时，查询被发送到本地域名服务器，由本地域名服务器先查本地缓存再递归查询**
    - **作为代理(proxy)，将查询转发给（层级式）域名解析服务器系统**

DNS查询示例

- 完整的递归DNS查询流程需要DNS服务器**从根域名“.”服务器、顶级域名服务器“.com”、一级域名服务器“taobao.com”一级一级递归查下来最终找到权威服务器取得结果，并返回给客户**，同时将取得的结果根据域名设置的**TTL时间**，**缓存**在自己的系统当中，以便下次使用。
- 迭代查询
    - 查询失败，则被查询服务器返回域名解析服务器的名字
    - “我不认识这个域名，但是你可以问这服务器”
    - 重试工作由查询者承担
- 递归查询
    - 将域名解析的任务交给所联系的服务器
    - 重试工作由被查询者承担，形成一个查询链路，每个查询者仅发送一次信息

DNS记录缓存和更新

- 只要域名解析服务器获得域名—IP映射，即缓存这一映射
    - 一段时间过后，缓存条目失效（删除）
    - **本地域名服务器一般会缓存顶级域名服务器的映射，因此根域名服务器不经常被访问**
- 记录的更新/通知机制
    - RFC 2136
    - Dynamic Updates in the Domain Name System (DNS UPDATE)

思考题

我国没有根域名服务器，是否会影响我国的网络安全，会有什么影响。请思考并查阅资料，回答该问题。

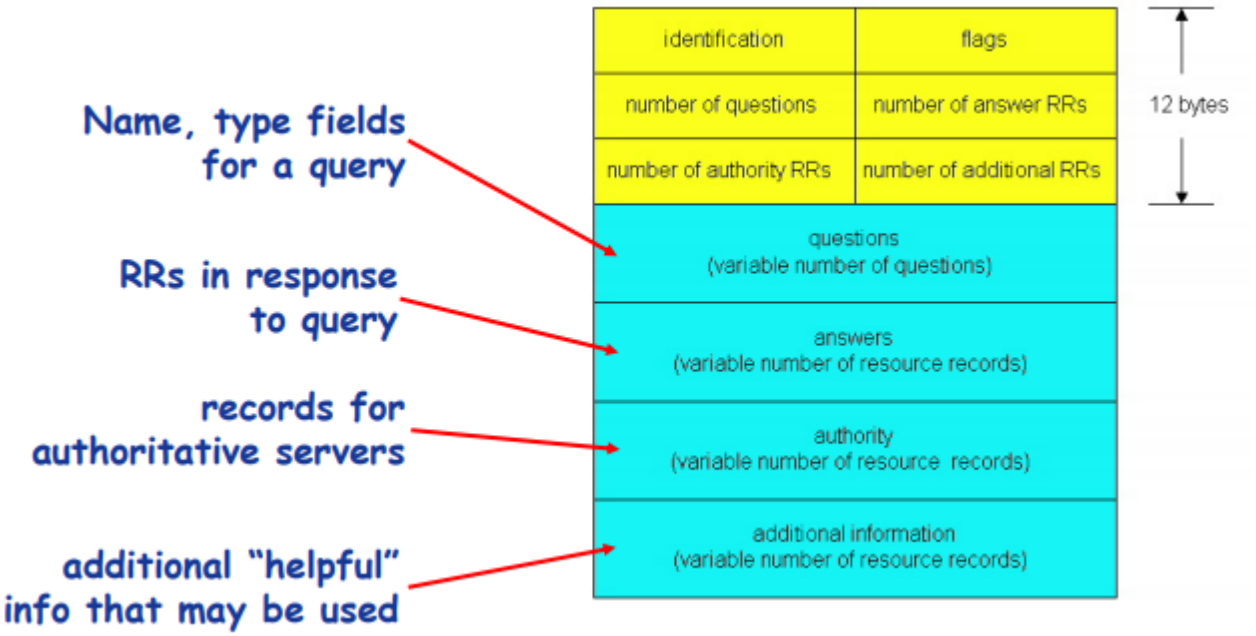
DNS记录和消息格式

DNS记录

- 资源记录(RR, resource records)
- RR format：（name，value，type，ttl）
- Type（从使用角度理解谁是name 谁是value）
  - Type=A（主机类）
    - Name: 主机域名
    - Value: IP地址
  - Type=NS（DNS）：解析服务器记录。用来表明由哪台服务器对该域名进行解析。这里的NS记录只对子域名生效。
    - Name: 域
    - Value: 该域权威域名解析服务器的主机域名
  - Type=CNAME（别名类）：别名记录。这种记录允许您将多个名字映射到另外一个域名。
    - Name: 某一真实域名的别名
    - Value: 真实域名
  - Type=MX（邮件服务类）
    - Value是与name相对应的邮件服务器

DNS协议与消息

- DNS协议：
  - 查询(query)和回复(reply消息)
  - 消息格式相同
- 消息头部
  - Identification: 16位查询编号，回复使用相同的编号
- flags：标志位
  - • 查询或回复
  - • 期望递归
  - • 递归可用
  - • 权威回答



如何注册域名？

- 例子：你刚刚创建了一个公司 “Network Utopia”
- 在域名管理机构(如Network Solutions)注册域名networkutopia.com
  - 向域名管理机构提供你的权威域名解析服务器的名字和IP地址
  - 域名管理机构向com顶级域名解析服务器中插入两条记录,分别是你域名权威解析器的ns记录和该权威解析器的A记录
- 在权威域名解析服务器中为www.networkuptopia.com加入Type A记录，为networkutopia.com加入Type MX记录

TCP？UDP？？

> DNS同时占用UDP和TCP端口53是公认的，这种单个应用协议同时使用两种传输协议的情况在TCP/IP栈也算是个另类。但很少有人知道DNS分别在什么情况下使用这两种协议。 先简单介绍下TCP与UDP。 TCP是一种面向连接的协议，提供可靠的数据传输，一般服务质量要求比较高的情况，使用这个协议。UDP---用户数据报协议，是一种无连接的传输层协议，提供面向事务的简单不可靠信息传送服务。 TCP与UDP的区别： UDP和TCP协议的**主要区别是两者在如何实现信息的可靠传递方面不同。**TCP协议中包含了专门的传递保证机制，当数据接收方收到发送方传来的信息时，会自动向发送方发出确认消息；发送方只有在接收到该确认消息之后才继续传送其它信息，否则将一直等待直到收到确认信息为止。 与TCP不同，UDP协议并不提供数据传送的保证机制。如果在从发送方到接收方的传递过程中出现数据报的丢失，协议本身并不能做出任何检测或提示。因此，通常人们把UDP协议称为不可靠的传输协议。。**不同于TCP，UDP并不能确保数据的发送和接收顺序。事实上，UDP协议的这种乱序性基本上很少出现，通常只会在网络非常拥挤的情况下才有可能发生。既然UDP是一种不可靠的网络协议，那么还有什么使用价值或必要呢？**其实不然，在有些情况下UDP协议可能会变得非常有用。**因为UDP具有TCP所望尘莫及的速度优势。**虽然TCP协议中植入了各种安全保障功能，但是在实际执行的过程中会占用大量的系统开销，无疑使速度受到严重的影响。反观UDP由于排除了信息可靠传递机制，将安全和排序等功能移交给上层应用来完成，极大降低了执行时间，使速度得到了保证。

> **DNS在进行区域传输的时候使用TCP协议，其它时候则使用UDP协议；** DNS的规范规定了2种类型的DNS服务器，一个叫主DNS服务器，一个叫辅助DNS服务器。在一个区中主DNS服务器从自己本机的数据文件中读取该区的DNS数据信息，而辅助DNS服务器则从区的主DNS服务器中读取该区的DNS数据信息。当一个辅助DNS服务器启动时，它需要与主DNS服务器通信，并加载数据信息，这就叫做区传送（zone transfer）。
>
> 为什么既使用TCP又使用UDP？ **首先了解一下TCP与UDP传送字节的长度限制：** UDP报文的最大长度为512字节，而TCP则允许报文长度超过512字节。当DNS查询超过512字节时，协议的TC标志出现删除标志，这时则使用TCP发送。**通常传统的UDP报文一般不会大于512字节。**
>
> **区域传送时使用TCP，主要有一下两点考虑：** 1.**辅域名服务器会定时（一般时3小时）向主域名服务器进行查询**以便了解数据是否有变动。如有变动，则会执行一次区域传送，进行数据同步。区域传送将使用TCP而不是UDP，**因为数据同步传送的数据量比一个请求和应答的数据量要多得多，不能乱序，或者丢包或者失真。**
>
> **域名解析时使用UDP协议：** 客户端向DNS服务器查询域名，一般返回的内容都不超过512字节，用UDP传输即可。不用经过TCP三次握手，这样DNS服务器负载更低，响应更快。**虽然从理论上说，客户端也可以指定向DNS服务器查询的时候使用TCP，但事实上，很多DNS服务器进行配置的时候，仅支持UDP查询包。**

思考题

请查阅有关资料，找出那些在应用层实现的Internet核心服务，调研它们的协议、消息格式。