Computer Network - Application Layer(HIT)

Author: Haihan Gao

General Discription

网络应用体系结构

网络应用服务需求

Internet传输服务层模型

特定网络应用与协议

Socket编程

网络应用的体系结构

网络应用的特点

网络应用的体系结构

- 客户机-服务器结构(Client-Server)
- 点对点结构(Peer to Peer)
- 混合结构(Hybrid)

Client-server

服务器：永久开机 域名不变，永久性访问 大量并发服务实现课扩展性

客户机：使用动态IP地址 不会与其它客户机直接通信

eg Web(服务器软件-浏览器软件 HTTP请求-HTTP响应)

P2P

- 没有永远在线的服务器
- 任意节点可以直接通信
- 节点间歇性的接入网络
- 节点IP地址可以改变
- 优点：高度可伸缩
- 缺点：难于管理

Hybrid(napster)

- 文件的传输P2P
- 文件的搜索:CS
    - 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
    - 避免服务器成为性能的瓶颈

网络应用进程通信-网络应用的核心是两台物理机上进行通信

如何通信：消息交换（报文交换）

客户机进程：发起通信请求 服务器进程：等待通信请求的进程

Socket：操作系统对网络协议栈的抽象

- 进程之间通信利用socket发送-接收消息
- 操作系统向进程提供网络服务API
    - 选择传输协议
    - 参数设置

进程的寻址

- 每个进程拥有表示符
- 寻址主机：IP地址可以唯一标识主机
- 寻址每个主机上的进程：端口号，为主机上每一个需要通信的进程提供端口号
- 进程的标识符：IP地址+端口号

应用层协议

- 网络应用遵循应用层协议
- 公开协议：RFC-允许互操作 eg HTTP，SMTP
- 私有协议：P2P共享应用

应用层协议的内容

- 消息的类型
    - 请求消息
    - 响应消息
- 消息的格式
    - 消息的字段
    - 每个字段如何描述
- 消息的语义
- 消息处理的规则

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/可靠性
- 时间/延迟
- 带宽 某些网络应用需要带宽达到最低要求 某些对于带宽是弹性的
- 安全

Internet提供的传输服务

TCP(e-mail Web)

- 面向连接：通信之前需要建立连接
- 可靠的数据传输：避免底层的不可靠
- 流量控制：防止超过接收方的处理能力
- 拥塞控制：网络负担过重限制发送
- 不提供时间保障和最小带宽保障

UDP(Internet telephone)

- 无连接
- 不可靠的数据传输

Web应用

构成

- 网页
  - 包含多个对象(HTML文件，JPEG图片)
  - 基本HTML文件，包含对其它对象的链接
  - 对象的寻址(URL统一资源定位地址) 协议:hostname+pathname
- 网页互相连接

HTTP协议

- C/S结构
  - 客户机-brower
  - 服务器-Web server
- 传输层协议 TCP
  - 服务器使用80端口
  - 浏览器发起到服务器的TCP连接，创建socket
  - 服务器接收来自浏览器的TCP连接
  - 浏览器与web服务器交换HTTP消息
  - 关闭TCP连接
- 无状态
  - 服务器不维护客户端过去发送的请求信息
  - 有状态协议复杂，存在同步的问题

HTTP连接的类型

非持久性连接

- 输入URL
- 向服务器80端口发送TCP连接请求
- 服务器在80端口等待TCP连接请求，接收连接并告知客户端
- HTTP客户端将请求消息（包含URL）通过TCP连接的套接字发出，消息所含的URL表明客户端需要对象
- HTTP服务器收到请求消息解析，产生包含所需对象的响应信息，并通过套接字发送给客户端
- HTTP服务器关闭TCP连接(非持久性协议，一次仅发送一个对象)
- HTTP服务器收到响应消息，解析HTML，发现指向其它对象的超链接，重复上述过程

响应时间分析

RTT(Round trip time)

- 发起建立TCP连接，一个RTT
- 发送HTTP请求消息到HTTP响应消息的前几个字节到达，1个RTT
- TOTAL=2*RTT+文件发送

持久性HTTP

- 持久性连接的问题
  - 每个对象需要2个RTT
  - 为每个TCP连接开销资源
  - 并行TCP获取网页对象，给服务器端带来负担
- 持久性连接
  - 发送响应之后，服务器保持TCP连接
- 无流水
  - 客户端收到前一个响应后才发送新的请求
  - 每个引用对象消耗1个RTT
- 流水
  - 收到所有对象只需要1个RTT

HTTP消息格式

请求消息(ASCII)

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language: fr

```

Get：request line 包含URL

head line

body

上传输入的方法

- POST 网页填写的表格传给消息体
- URL方法 使用GET方法 输入信息通过request字段的URL输入
- HEAD方法，测试

PUT方法

- 将消息体中的文件上传到URL字段指定的路径

Delete

- 删除URL字段指定的文件

HTTP响应消息

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon,22 Jun 1998
Content-Length: 6821
Content-Type: text/html

data data
```

status line(protocol/status code/status phrase)

header line(data:send time last-modified:last modified time)

status code:404(not found)

Cookie技术

掌握user-agent的状态

What is Cookie

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

Cookie组件

- HTTP响应消息相应的cookie头部行
- HTTP请求消息相应的cookie头部行
- 保存在客户机主机上的cookie文件，由浏览器管理
- Web服务器端的后台数据库

Cookie的原理

- 常规HTTP请求消息
- 服务器为客户创建ID，将其IP地址放到数据库中
- 响应消息加入头部行，返回ID
- 用户加入访问的IP和对应的cookie到cookie文件中
- 常规访问服务器，头部行中加入cookie
- 服务器查询cookie，从服务器数据库中查询对应的内容

Application of Cookie

- 身份认证
- 购物车
- 隐私问题

缓存/代理服务器技术

概念：不访问服务器的前提下满足客户端的HTTP请求

提升性能

- 缩短客户请求的响应时间
- 减少机构、组织的流量
- 在大范围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
    - 如果在缓存中，直接返回
    - 否则，缓存服务器向原始服务器发送HTTP请求，获取对象，然后返回客户端并保存在本地
    - 原理：局域网带宽大与局域网接到广域网的带宽

条件GET方法

- 目标：
  - 如果缓存有最新的版本，则不需要发送请求对象
- 缓存
  - 在HTTP请求消息中声明所持有版本的日期
  - if-modified-since
- 服务器
  - 如果缓存中版本是最新的，则响应消息中不包含对象
  - HTTP/1.0 304 Not Modified

Email应用

Email应用的构成

- 邮件客户端
  - 读写Email消息
  - 与服务器交互，交换email
- 邮件服务器
  - 邮箱：存储发给用户的email
  - 消息队列：存储等待发送的email
- SMTP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
- 可以保证主机关闭时仍然能收到邮件

SMTP协议

- TCP进行Email消息的可靠传输
- 端口25
- 握手
- 消息传递
- 关闭
- 命令/响应交互模式(HTTP采用请求/响应模式)
  - 命令：ASCII文本
  - 响应：状态代码和语句
- 异步应用，发送方发送和接受方接收不用同时
- 采用持久性连接
- 消息由7为ASCII码构成
- 利用CRLF.CRLF标识消息的结束

Email消息格式

- header 头部行
  - To
  - From
  - Subject

- body 消息体
    - 消息本身
    - 只能是ASCII字符
    - 多媒体邮件格式扩展
    - 文件头部增加格外的行声明是否含有多媒体，如何解码，类型

邮件访问协议：从服务器获取邮件

POP协议

- 认证/授权
    - 客户端命令
        - user
        - password
    - 服务器响应
- 下载
    - 下载并删除?
    - 下载并保持
- 无状态协议

IMAP协议

- 更多功能，更加复杂
- 操纵服务器上存储的信息
- 所有的消息统一保存在一个地方
- 允许利用文件夹组织消息
- 支持跨会话的协议

DNS

解决的是Internet上主机/路由器的识别问题

- IP地址
- 域名
- 域名和IP地址之间的映射，域名翻译为IP地址

域名解析系统DNS

- 多层命名服务器构成的分布式数据库
- 应用层协议，完成名字的解析
    - Internet核心功能，用应用层协议实现(Why?)
    - 网络边界复杂

DNS service

- 域名向IP地址的翻译
- 主机别名
- 邮件服务器别名
- 复杂均衡，Web服务器
- 分布式，不采用集中式
    - 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

- 第一层：根服务器
- 第二层：顶级域名服务器 com DNS server
- 第三层：权威域名服务器
- eg 查询www.amazon.com
    - 客户端查询根服务器，找到com域名解析服务器
    - 客户端查询com域名解析服务器，找到Amazon.com域名解析服务器
    - 客户端查询amazon.com域名解析服务器，找到www.amazon.com的IP地址

DNS根域名服务器

- 本地域名解析服务器无法解析域名时，访问根域名服务器
- 根域名服务器
    - 知道映射，返回
    - 不知道映射，访问权威域名服务器

顶级域名服务器

权威域名服务器-组织负责维护

本地域名解析服务器

- 不属于层级体系
- 每个ISP都有一个本地域名服务器
- 主机进行DNS查询，查询被发送给本地域名服务器
    - 作为代理proxy，将查询转发给层级域名服务器
- 迭代查询
    - 被查询服务器返回域名解析服务器名字
- 递归查询
    - 将域名解析服务器的任务交给所联系的服务器

DNS记录的缓存和更新

- 只要域名解析服务器获得域名-IP映射，即缓存
    - 一段时间之后，缓存条目失效
    - 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会被经常访问

DNS记录和消息格式

资源记录

RR format:name value type ttl

- Type=A
  - name:主机名
  - value:IP地址
- Type=NS
  - Name：域
  - Value：域的域名解析服务器的主机域名
- Type=CNAME
- Type=MX

DNS协议与消息

DNS协议

- 查询/回复协议
- 消息格式相同

消息头部

P2P应用

应用：原理与文件分发

BitTorrent

Tracker：跟踪参与torrent的节点

Torrent：交换同一个文件块的节点组

- 文件划分为256KB的chunk
- 节点加入torrent
  - 没有chunk，但是会逐渐积累
  - 向tracker注册并获得节点清单，与某些节点建立连接
- 下载的同时，节点需要向其它节点上传chunk
- 节点可以加入或离开
- 一旦节点获得完整的文件，它可能离开或者留下

获取chunk

- 给定任意时刻，不同的节点持有文件的不同chunk集合
- 节点定期查询每个邻居所持有的chunk列表
- 节点发送请求，请求获取缺失的chunk
  - 稀缺优先，所缺失的chunk可以提供的节点较少，则优先提供这些缺失的chunk

发送chunk

- 向n个邻居发送chunk：正在向其发送chunk，而且是速率最快的n个
  - 每10s重新评估top n
- 每30s重新选择一个其它节点，向其发送chunk
  - 新选择的节点可能加入top n

P2P应用：索引技术

搜索信息P2P

- 信息到节点位置IP地址+端口号的映射
- 文件共享
- 即时消息

集中式索引

- 节点加入时，通知中央服务器
  - IP地址
  - 内容
- 节点到中央服务器上查询
- 节点向节点传输文件

分布式索引(Query flooding)

- 每个节点对它共享的文件进行索引，而且只对它共享的文件进行索引
- 覆盖网络
  - 节点X和Y之前如果有TCP链接，那么构成一个边
  - 所有活动节点和边构成覆盖网络
  - 节点的一般邻居数少于6个
  - 利用graph检索
  - 任何收到查询消息的节点将会传递查询消息，如果查询利用，那就利用反向路径发回查询节点
  - 可能导致网络拥塞

层次式覆盖网络

Socket编程(略)