第二章 应用层

1 网络应用层的结构体系

1.1 研发网络应用程序的核心

- 写出能够运行在**不同端系统**并通过网络彼**此通信的程序**
- 没有应用程序运行在**网络核心设备**上

1.2 客户机/服务器体系结构

1.3 服务器

- 总是打开的主机
- 具有固定的、众所周知的IP地址

1.4 客户机

- 同服务器端通信
- 可间断性的同服务器连接
- 可拥有动态IP地址
- 客户机之间不直接通信

1.5 P2P体系结构

- 没有总是打开的服务器
- 任意一对主机直接相互通信

- 等对方间歇连接并且可以改变IP地址

  自扩展性高 & 难以管理

1.6 混合体系结构

1.6.1 Napster

- 文件直接在对方之间交换
- 文件搜索通过服务器
  - 服务中性记录对等方内容
  - 对等方查询服务中心来决定要求的文件位置

1.6.2 即时通信

- 两个聊天用户之间是P2P
- 注册、查询通过服务器

2 Web应用和HTTP协议

- 网页：**HTML文件+若干引用对象构成**，每个对象被一个**URL**（Uniform Resource Locator）寻址

2.1 HTTP概述

- 超文本传输协议（Hypertext Transfer Protocol）

- Web**应用层**协议

2.2 使用TCP

1. 客户初始化一个与HTTP服务器**80端口**的TCP连接
2. **HTTP服务器**接受来自客户的TCP连接请求，建立连接。
3. **Browser**和**Web服务器**交换HTTP消息
4. 结束响应

   HTTP是无状态协议，HTTP服务器不维护客户先前的状态信息

2.3 HTTP连接

2.3.1 非持久HTTP连接

- 每个TCP连接上只传送一个对象，下载多个对象需要建立多个TCP连接

2.3.2 持久HTTP连接

- 一个TCP连接上可以传送多个对象

2.4 响应时间模型

2.4.1 往返时间RTT（Round-Trip Time）

- 一个小分组从客户主机到服务器再到客户主机所花费的时

间

- 1个RTT用于建立TCP连接
- 1个RTT用HTTP请求/响应消息的交互

Total = 2RTT + transmittime

2.4.2 各种连接的耗时

2.4.2.1 非持久HTTP连接

- 每个对象需要两个RTT
- OS必须为每个TCP连接分配主机资源
- 大量客户并发TCP连接形成服务器的严重负担

2.4.2.2 持久HTTP连接

- 服务器发送响应消息之后保持连接

2.4.2.3 不带流水线的持久HTTP连接

- 客户先前响应消息收到才发出醒的请求消息
- 每个引用对象经历1个RTT

2.4.2.4 带流水线的持久HTTP连接

- 客户遇到一个引用对象就发送请求信息
- 所有应用对象只经历1个RTT

2.4.3 HTTP报文格式

2.4.3.1 报文格式

- 请求报文 request

- 响应报文 response
- ASCII文本

2.4.3.2 方法类型

- HTTP 1.0
  - GET
  - POST
  - HEAD（服务器接受请求时，对报文进行响应，但不返回请求对象）
- HTTP 1.1
  - GET
  - POST
  - HEAD
  - PUT（文件被上载到URL指定的路径）
  - DELETE（删除URL指定文件）

2.5 用户与服务器交互：Cookies

2.5.1 四个重要的方面

- Cookie头部在HTTP的响**应消息**中
- Cookie头部行在HTTP**请求消息**中
- Cookie文件保存在用户主机被**用户浏览器**管理
- Cookie也保存在**Web站点的后端数据库**

2.6 Web缓存（代理服务器）

2.6.1 所有HTTP请求指向缓存

- 对象在缓存中，缓存器返回对象
- 否则向其实服务器发起请求，接受对象**转发**给客户机
- 缓存既是**服务器**又是**客户机**

2.6.2 Web缓存的用处

- 减少对客户机请求**响应的时间**
- 减少内部网络与接入链路上的**通信量**
- 能从整体上大大降低因特网上的**Web流量**

3 文件传输协议FTP：两种连接

3.1 控制连接 & 数据连接

- FTP客户端首先发起建立一个与FTP服务器**端口号21**之间的**TCP控制连接**，指定TCP作为**传输层协议**
- 获得身份认证
- 发送命令控制远程主机
- 服务收到文件传输命令，在**服务器端口20**创建一个与客户的**TCP数据连接**
- 一个文件传输后，服务器结束这个**TCP数据连接**

3.2 FTP数据连接的建立方式

3.2.1 主动模式

- 客户端发送**PORT命令**
- 服务器更具PORT命令**指定的客户度端地址**和**端口**发起数据连接

3.2.2 被动模式

- 客户端发起**PASV命令**
- **服务器返回**监听的地址和端口号
- 客户端发起**数据连接**

3.3 FTP常见命令和应答

- **USER username**
- **PASS password**
- **LIST** 返回远程目录文件列表
- **RETER filename** 获取文件
- **STOR filename** 存放一个文件到远程主机

4 电子邮件：组成及其使用的协议

4.1 用户代理 User Agents

- 允许用户阅读、回复、转发、保存、编辑邮件消息
- 发送消息到服务器
- 从服务器接收消息

- 运行邮件协议

4.2 邮件服务器

- **邮箱**：存放用户接受的邮件消息
- **外出报文队列** outgoing message queue
- **运行邮件协议**

4.3 SMTP：Simple Mail Transfer Protocol

- 客户使用TCP来可靠传输邮件消息到**服务器端口号25**
- 直接传输：**发送服务器**到**接受服务器**
- **传输三阶段**
  - 握手
  - 邮件消息传输
  - 结束
- 命令/应答交互
  - **命令**：ASCII文本格式
  - **应答**：状态码及其短语
- 邮件消息必须是**7-bit ASCII**

4.4 SMTP & HTTP

- **HTTP**：拉协议
- **SMTP**：推协议
- 都有ASCII命令/应答交互，状态码

4.5 邮件消息格式

- Header
  - To
  - From
  - Subject
- Blankline
- Body

4.6 邮件访问协议

- SMTP：**递送/存储**邮件消息到**接受者邮件服务器**
- 邮件访问协议：从服务器**获取邮件消息**

4.6.1 POP：Post Office Protocol

- **身份认证**并**下载邮件消息**

4.6.2 IMAP：Internet Message Access Protocol

- **更多**功能特征
- 允许用户像对待**本地邮箱**那样**远程操纵**邮箱的邮件

4.6.3 HTTP

- Hotmail
- Yahool

5 DNS的功能和实现

- Domain Name System

5.1 提供功能

- 主机名到IP地址的转换
- 主机别名
- 邮件服务器别名
- 负载分配

5.2 DNS构造

- 分散式数据库
  - 一个有分层DNS服务器实现的分布式数据库
- 为什么不集中DNS?
  - 单点故障
  - 巨大访问量
  - 远距离集中式诗句哭
  - 维护
  - 不可扩展
- 应用层协议
  - DNS服务器实现域名转换

5.3 DNS服务器等级

- 根名字服务器 Root name servers
  - 负责记录顶级域名服务器信息

- 顶级域名服务器 top-level domain servers
  - 负责顶级域名和所有国家的顶级域名
- 权威DNS服务器 authoritative DNS servers
- 本地DNS服务器 Local DNS name server

5.4 DNS查询方法

- 递归查询 recursive query
- 迭代查询 iterated query

5.5 DNS记录

RR格式：（name，value，type，ttl）

Type = A

name = 主机名
value = IP地址

Type = CNAME

name = 主机别名
value = 真实的规范主机名

Type = NS

name = 域名
value = 该权威域名服务器的主机名

Type = MX

name = 邮件服务器的主机别名
value = 邮件服务器的真实规范主机名

6 P2P文件共享原理和实现技术

6.1 P2P构架

- 没有总是在线的服务器
- 任意端系统之间直接通信
- 对等方之间可以间断连接并可以改变IP地址

6.2 P2P：集中式目录

- 当对等方启动时，通知服务器
    - IP地址
    - 可供共享的对象名称
- 问题
    - 单点故障
    - 性能瓶颈
    - 侵犯版权

6.2 查询洪范：Gnutella

- 全分布，无集中式服务器
- 公共域协议
- 许多Gnutella客户机实现Gnutella协议

6.2.1 加入对等方

- 必须先发想在Gnutella网络 中的其他对等方，使用**对等方列表**
- X试图在该列表上的对等方建立一条TCP连接
- X向Y发送一个**Ping报文**
- 所有的对等方就收Ping报文并响应一个**Pong报文**
- X收到许多Pong报文，然后能同某些其他对等方**建立TCP连接**

6.2.2 对等方离开

- 主动离开
- 断网

6.3 KazaA

- 那个对等方要不**被指派为组长**，或者被**指派给一个组长**
- **对等方和组长之**间建立TCp连接
- **组长之间**建立TCp连接
- 组长维护子对等方之间的共享内容

6 TCP 套接字编程

6.1 Socket套接字编程

- 套接字是一个主机**本读应用程序**所创建的，为操作系统所控制的**接口**
- 应用进程通过这个接口。使用**传输层提供服务**，跨网络**同其他进程**进行通信

6.2 用TCP进行套接字编程

- 客户必须**初始联系服务器**
    - 服务器进程必须**先运行**
    - 服务器进程必须创建套接字来**迎接客户**的初始联系联系
- 客户如何初始联系服务器
    - 创建客户**本地TCP Socket**
    - 指定服务器进程的**IP地址**和**端口号**
    - 一旦客户创建套接字，客户TCP就发起**3次握手**并建立与服务器的**TCP连接**

6.3 流的概念

- 流是流入或者流出进程的**一串字符**
- **输入流**被加在进程的某个输入源上
- **输出流**被加在进程的某个输出源上

6.4 编程示例

```
from socket import *
serverName = 'localhost'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))
sentence = input('Input lowercase sentence:')
clientSocket.send(sentence.encode('utf-8'))
modifiedSentence = clientSocket.recv(1024)
print ('From Server:', modifiedSentence.decode('
utf-8'))
clientSocket.close()
```

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('localhost', serverPort))
serverSocket.listen(1)
Print('The server is ready to receive')
while 1:
     connectionSocket, addr = serverSocket.accep
t()
     sentence = connectionSocket.recv(1024)
     capitalizedSentence = sentence.upper()
     connectionSocket.send(capitalizedSentence)
     connectionSocket.close()
```

7 UDP套接字编程

- UDP客户和服务器之间**没有连接**
  - 没有初始**握手阶段**
  - 发送方明确将接收方的**地址**和**端口号**加入**每个分组**
  - 服务器必须从接受的分组中**析取**发送方进程的**IP地址**和**端口号**
- UDP发送数据可能被**乱序**收到或者**丢失**

7.1 编程举例

```
from socket import *
serverName = 'localhost'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_DGRAM)
message = input('Input lowercase sentence:')
clientSocket.sendto(message.encode('utf-8'), (se
rverName, serverPort))
modifiedMessage, serverAddress = clientSocket.re
cvfrom(2048)
print('From Server:', modifiedMessage.decode('ut
f-8'))
clientSocket.close()
```

```
from socket import *
    serverPort = 12000
    serverSocket = socket(AF_INET, SOCK_DGRAM)
    serverSocket.bind(('localhost', serverPort))
    print ("The server is ready to receive")
    while 1:
        message, clientAddress = serverSocket.re
cvfrom(2048)
```

```
        modifiedMessage = message.upper()
        serverSocket.sendto(modifiedMessage, cli
entAddress)
```