创造一个网络app

你需要写的程序是基于终端的

你不需要为network-core devices写程序//因为一开始设计就是为了end point的快速的开发，不至于买了一个新路由就要重新写程序

因此application layer 是基于end-end的

app architecture只有两种

cliend-server

peer-to-peer(p2p)

**Client-server architecture**

server:

always-on host ，永远打开的终端

永久的IP地址

建造一个数据中心来集成这些server，更方便拓展scale

Client:

与server交互

有时候在线有时候不在线，intermittently connected

动态地址dynamic IP address

client之间不会直接相连

.

**P2P architecture**

没有 server

arbitrary end systems directly communicate 任意终端直接连接

不同end system之间互相提供服务，申请服务

self scalability: 自我伸缩性， 新的Peer带来服务资源也增加了服务需求

peer被间接链接（非永久）并且会动态改变地址，导致管理复杂

Process communicating

**Process:** 进程，终端host内运转的程序

在同一个host里，两个进程的交流使用**Inter-process communication** (是OS描述的)

在不同的host里，是通过交换**message communication**来交流的

client servers：

client process: 激活交流的Process

server process : 等待被呼叫的Process

aside: p2p结构的应用，既有client process又有server process

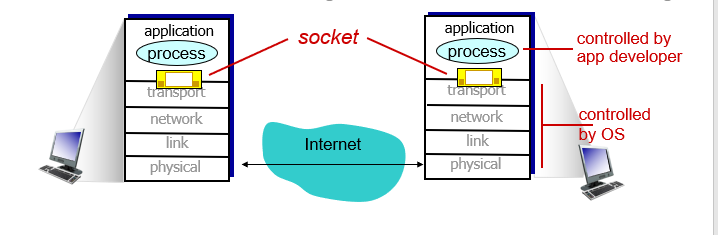
Sockets:

process 接受/发出 信息通过他的 socket（凹槽）

socket类似analogous门一样

sending process 把信息从门推出

sending process 依赖门另一侧的传输设施transport layer(比喻就是管道)来把信息传递给receiving process的socket



socket推出信息以后，OS会控制这个信息，把它通过internet传到另一个电脑，另一个电脑推给socket

也就是说开发者只需要写process

Addressing process

为了接收到信息，process必须有个鉴定器identifier

host设备通常有32bit IP

问题：host的IP地址足够用来分辨process吗

答案：不够，同一个Host可能有很多process

Identifier同时包含IP address与port numbers

IP address是host

port numbers是process

post number的例子：

HTTP server:80， mail server:25

为了把一个Http信息发给某一个server，需要

IP address:128.119.245.12

port number:80

App-layer protocal defines //app层的协议描述的是

信息的类别type：例如

request, 请求

response 回应

信息的格式syntax：

what fields in messages 消息中有哪些字段

how fields are delineated 这些字段是如何被描述的

信息的semantics语义：

field的meaning

信息的规则rule：

when and how发出信息&&接受信息

app-layer这一层的绝大部分协议是开源的，open protocals

以rfc描述

例如HTTP SMTP

还有一些是不告诉你内部代码的protocal : proprietary

例如skype

一个app需要transport layer提供怎样的运输服务？

data integrity数据完整性

有些程序例如文件传输这种，需要百分百的可靠数据转移

其他app例如视频，可以接受些许Loss

timing:

一些app例如DOTA，需要较低延迟

throughput：带宽

一些app例如youtube需要一定的最小带宽来effective

其他app则随意

security

encryption加密

Internet transport protocols service

一共有两种transport protocols

**TCP service:**

**reliable transport**： 接受process与发出process之间的可靠传输

flow control: 发出者不会过快的发出信息，导致接受者overwhelm超负荷

congestion 拥挤control: 当网络负荷超载压制sender

Connection-oriented: 你需要一些时间来在client 与server process之间 connection setup

不能提供的：timing延迟， minimum throughout guarantee 最小带宽担保, security 安全性

**UDP service:**

unreliable data transfer， 在发出与接受process之间是不稳定的信传输

不提供：可靠性，flow control, congestion control, timing, throughoutput guarantee , security, connection setup

less latency更小的延迟

为什么有UDP: 因为 sender 网络拥挤的时候 sender过分被压制， 有可能我们需要让TCP得到信息

cheaper

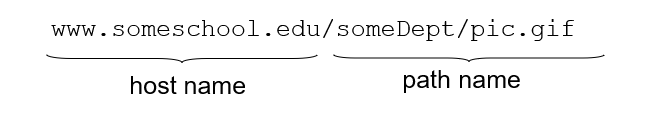
Web and HTTP

一个网页包含了多个objects

object可以是HTML file，图片等等

网页本身构建在base HTML-file上，然后HTML-file里有多个referenced objects

每一个object由一个URL addressed



前面是Host name:在哪个终端

后者是具体位置

HTTP：超文本hypertext传输协议 ，hypertext指的是含有指向其它文本文件链接的文本

他是web的 application layer（第一层）协议

**是client/server model**

client: 浏览器， 浏览器发出http request,收到http response, （基于http 协议）， 并且display对应的Web Object

server: Web server 通过HTTP协议 发出对应object作为对应request的回应

**HTTP使用TCP:**

client会初始化一个TCP链接（也就是创造一个socket）一直连到SERVER, PORT 80

然后server接受了从client来的TCP链接

HTTP信息就可以在浏览器(client)与Web server(server)之间交换了

结束后关闭TCP connection

HTTP是stateless的，server不会保存过去client request的任何信息

HTTP链接：

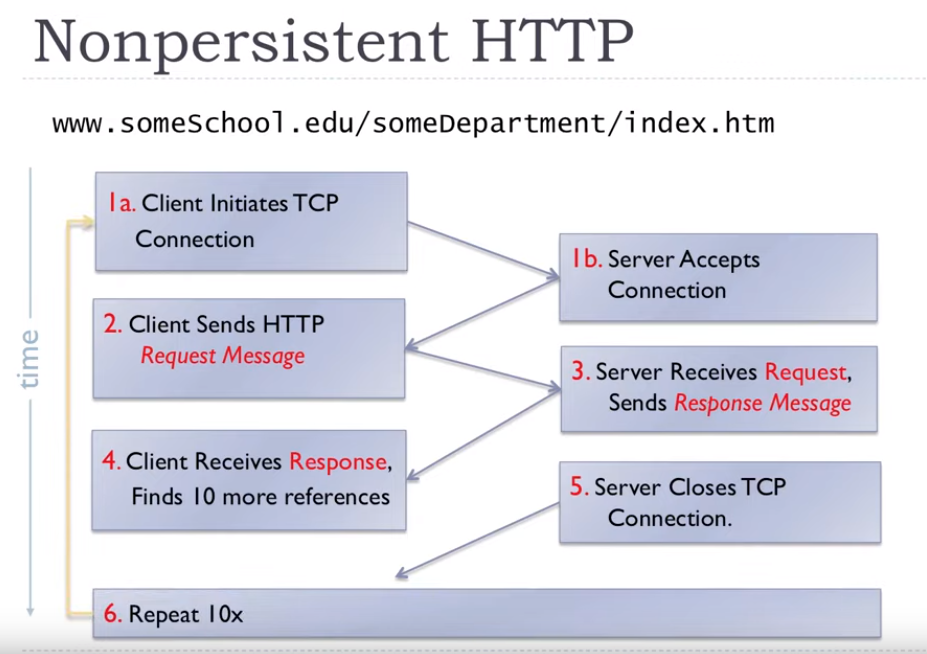
**non-persistent HTTP** 不坚固的HTTP

一次最多一个Object从TCP链接发出来，然后TCP就会关闭

加载多个OBJECT需要多个TCP链接

persistent HTTP

可以通过一个TCP传输多个object



Non Persistent的具体过程

1.Client尝试建立链接

2.Server接受连接

3.Cleint发出HTTP Request Message (只是message)

4.Server 接收到了请求，发出Response Message并包括被需求的object，

5.Client接受了Response找到了10个图片reference (就是对图片声音啥的链接)

6.Server会关闭connection

7.重新进行循环

RTT:一个小数据包从client到server再被传回来的时间

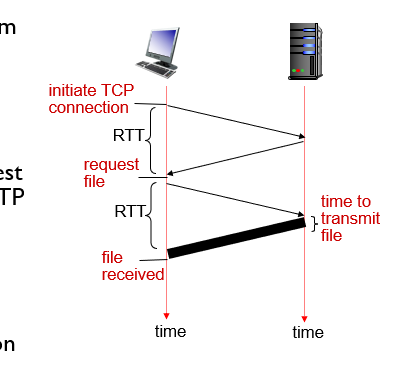
HTTP响应时间：

一个RTT用来建立TCP链接

一个RTT用来发出HTTP request 加上收到文件的时间

然后file transmission既把文件从服务器传到连接上所用的时间

所以一共需要2RTT+FILE TRANSMISSION TIME



non-persistent Http 的问题

2RTT:每个OBJECT需要两个RTT的时间

Overhead:OS过载因为每一个TCP都要建立连接

浏览器经常开平行TCP链接为了取东西

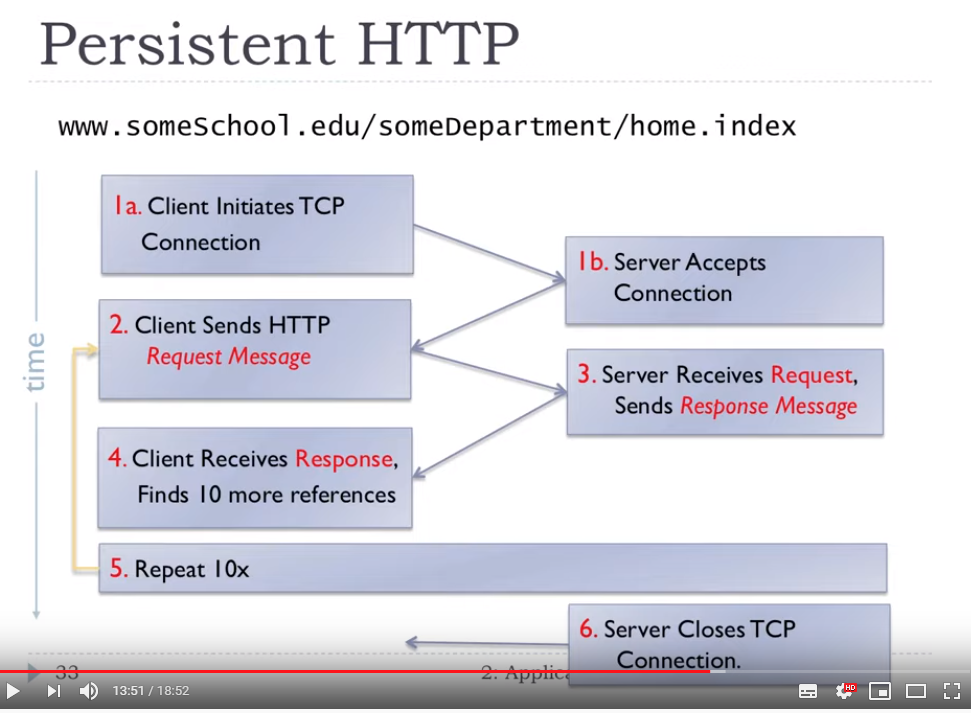
Persistent HTTP

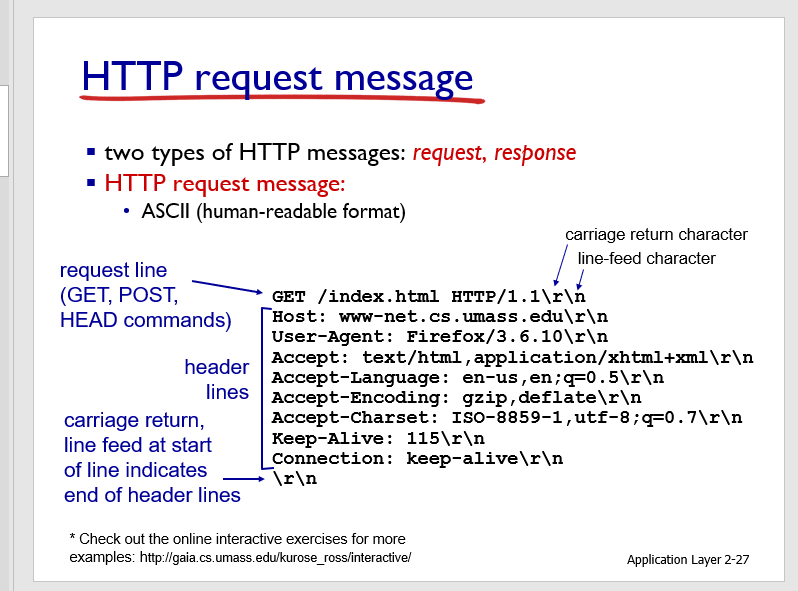
server在发出回应以后就会一直保持连接

之后同一个client/server之间就可以通过这个链接收发信息

client一遇到referenced object就可以发出请求

所有的referenced object都只需要一个RTT时间





HTTP信息一共有两种

request/response

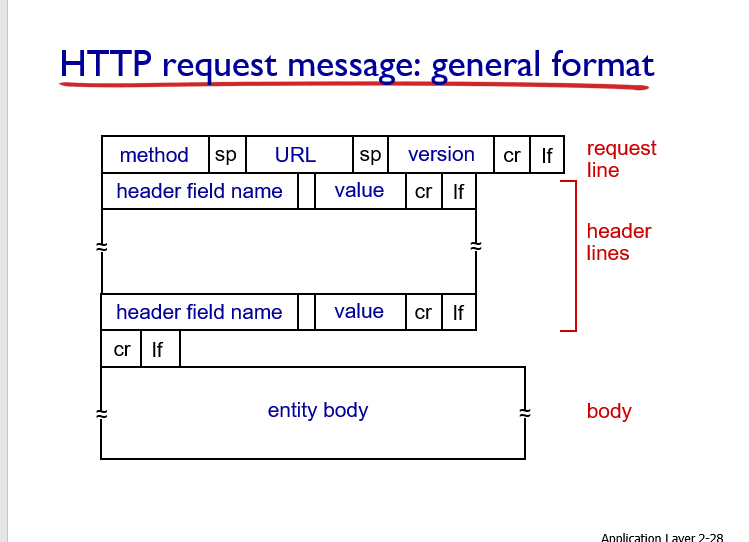
Request message是ASCII码

第一行是request line

get/post/head ， HTTP/1.1是protocol

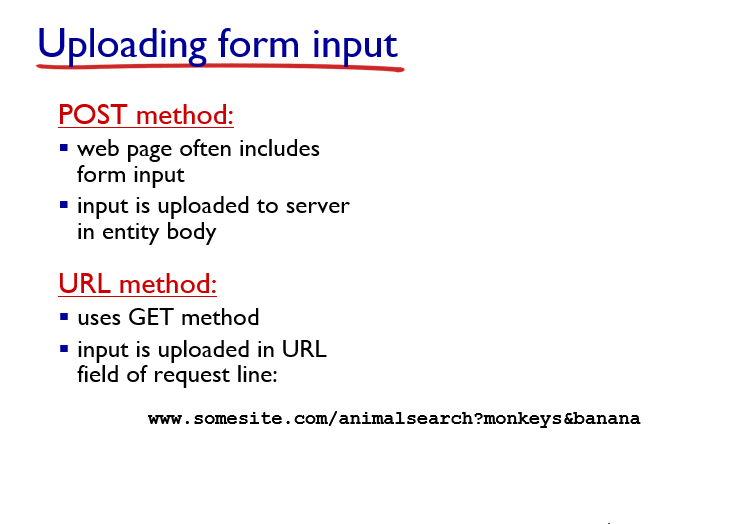
接下来就是headerline，

然后会\r\n，叫做carriage return，告诉你这是end of header lines



sp就是斜杠

lf就是Line feed caracter

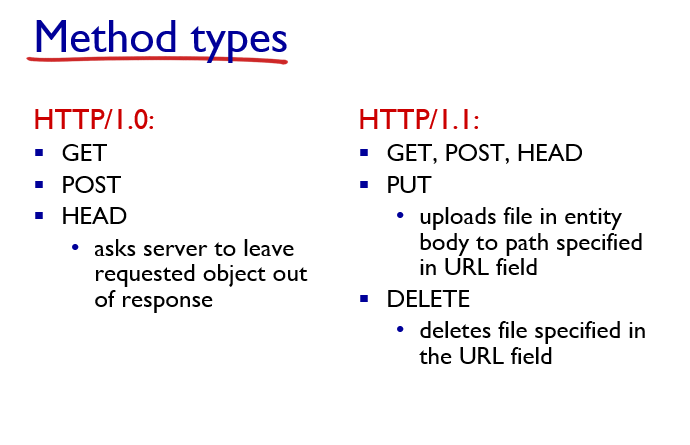


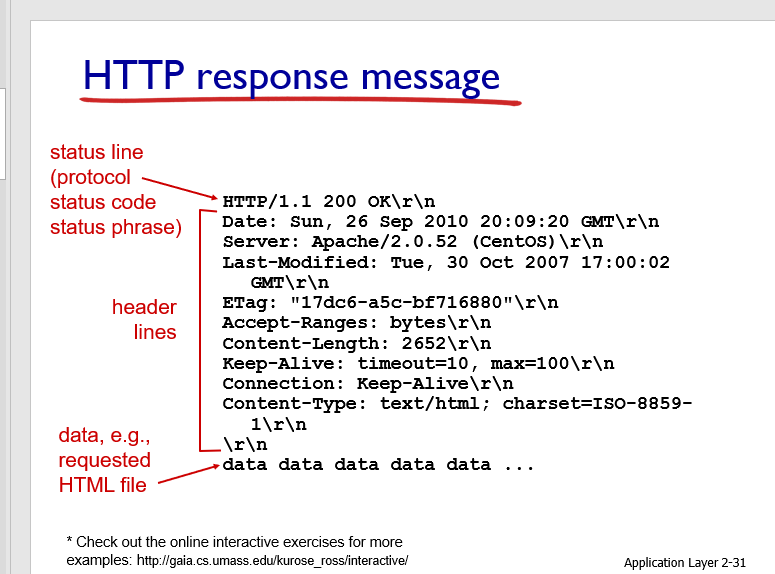
Post method:

POST与GET method都是上传输入信息的方式

POST会在entity body中输入

GET会在URL那栏输入





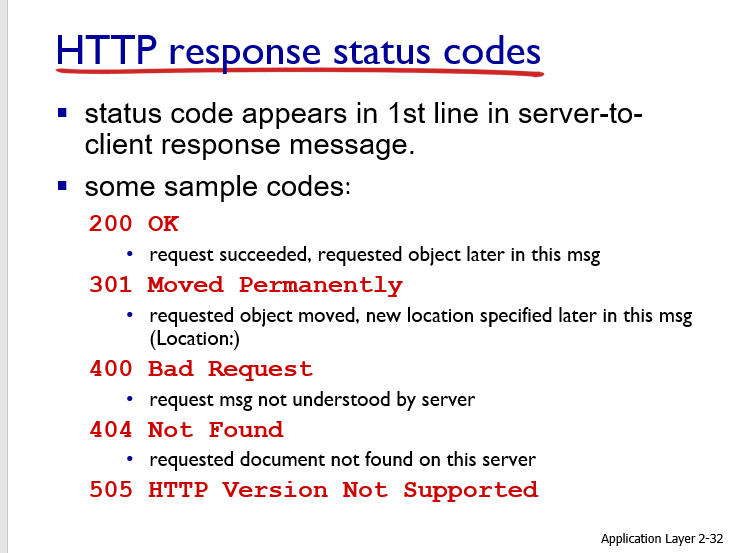
response message

第一行是status line，告诉你用了什么协议

headerlines

空一格

最下面才是data



第一行的数字

200，成功

301，需求的Object永久的被移除了

400：Bad request，无法理解这个request

404: 服务器找不到这个server

505：不支持这个HTTP version

User-server state: cookies

网站用来maintain state的东西，当初始HTTP request arrive，

cookie由四部分构成

1. cookie header line of HTTP response message : 创造独特的ID与entry //网站的response，告诉你要创造怎样的cookie

2. cookie header line in next HTTP request message告诉浏览器要set entry //存储cookie信息，下一次发request直接告诉网站我这次是什么ID

3. cookie file用户的host所存储, 由用户的浏览器控制

4.存储在网站的后端信息

例子：

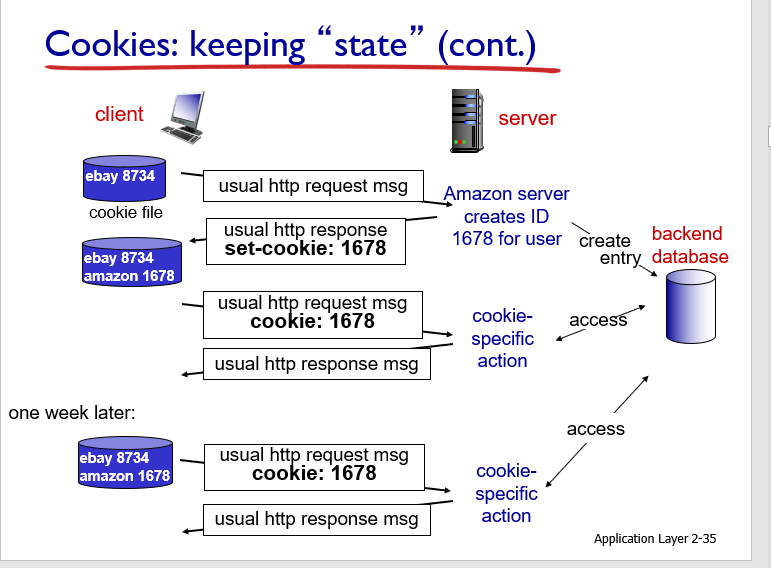
Susan 从PC进入INTERNET

浏览了某一个电子商务网站

当初始HTTP request 到达网站

SITE会创造独立的ID， //第一部分

这个ID的入口entry 对后端数据库



例如Client (一开始存了一个cookie ebay8734 )发送了一个request给msg， amazon server创造了一个 用户ID1678， 并且返回这个response, 这个response会设置CLIENT的cookie，

那么这时用户会长期保存一个cookie 1678, 再发request的时候会带一个cooki，然后会得到常规response

在一星期以后也会得到常规response

Cookie能用来干嘛

authorization授权 //登陆没有

shopping carts， 记录购物车

用户会话状态user session state（网络邮箱）

怎么保持state：

protocol endpoint协议端点， 通过多个处理transaction在sender/receiver 保持state

cookies: 本质上是携带了state的http message

cookies and privacy

cookie 允许permit网站得到你的许多信息

你可能需要提供名字与邮箱给网站

Web caches (proxy server)

网络高速缓存器 （代理服务系统）

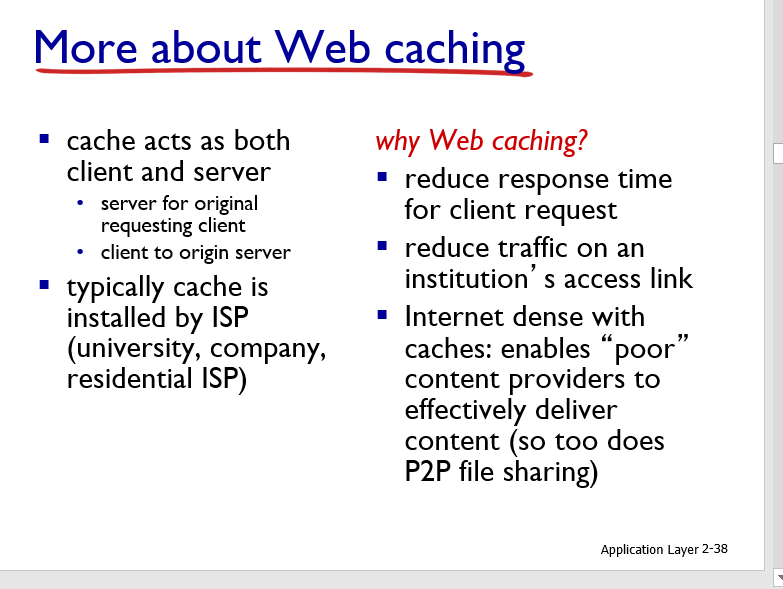
目的

让client request得到response在不与原始origin server 直接接触的情况下

browser把所有HTTP request 送到cache处：

如果是cache里面就有的object， cache return http request

如果cache里面不存在被需求的object: cache 向 origin server发出 requests object， 然后再把object return 给client



cache即是client又是server， 对于用户的初始request来说， 他是server

对于origin server来说，他是client

通常 cache 是通过ISP安装的 （学校，公司，地区型ISP)

为什么要web caching

减少用户得到 reponse的时间

减少origin server的 访问链接（减少了traffic)  
因为internet有充足的（dense with 充满了） caches， 这样没钱的内容提供商也可以迅速的提供内容（p2p file sharing也是）

caching example“

假设平均object的size是100K bit

假设从浏览器到最终原始server的request频率是15/sec

浏览器的平均data rate是1.5mbps， //计算出来的

RTT（发出request到收到收到reply的间隔，是两秒

access link rateL1,54 mbps

那么结果就是

局域网利用率：15％

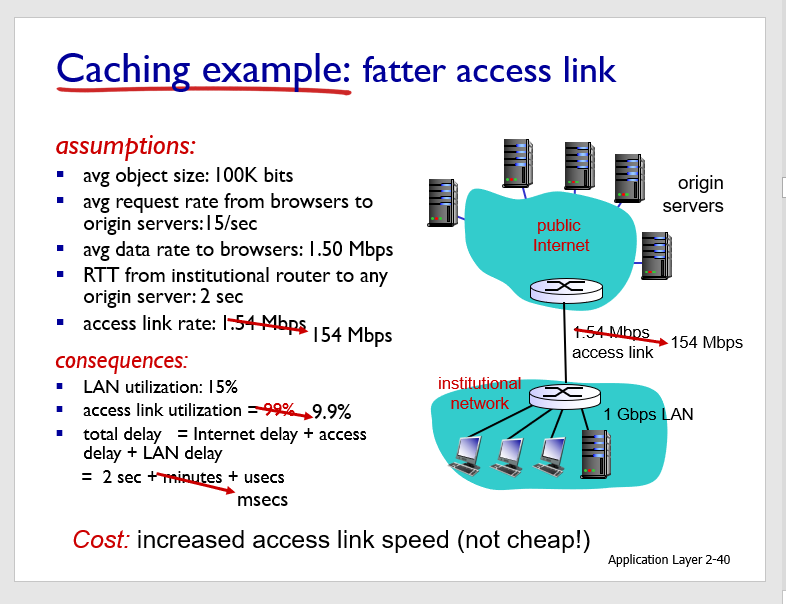
access link utilization=99％， 出入路链 利用率99％太大了

总共的延迟=internet delay+ access delay+lan delay

具体计算， 浏览器平均data rate 1.5mps=100k \*15 /sec

1.54 mbps 是自带的数据

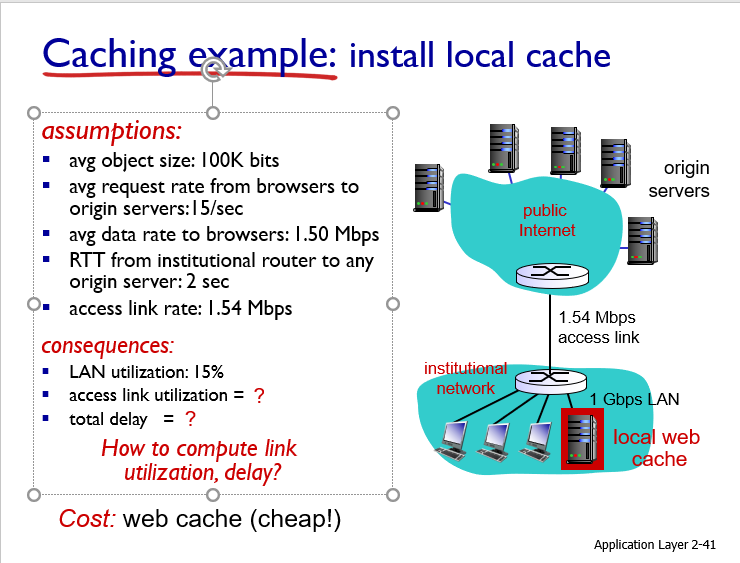
1.5mbps/1.54 mbps数据太高



提高了access link rate负载能力更高

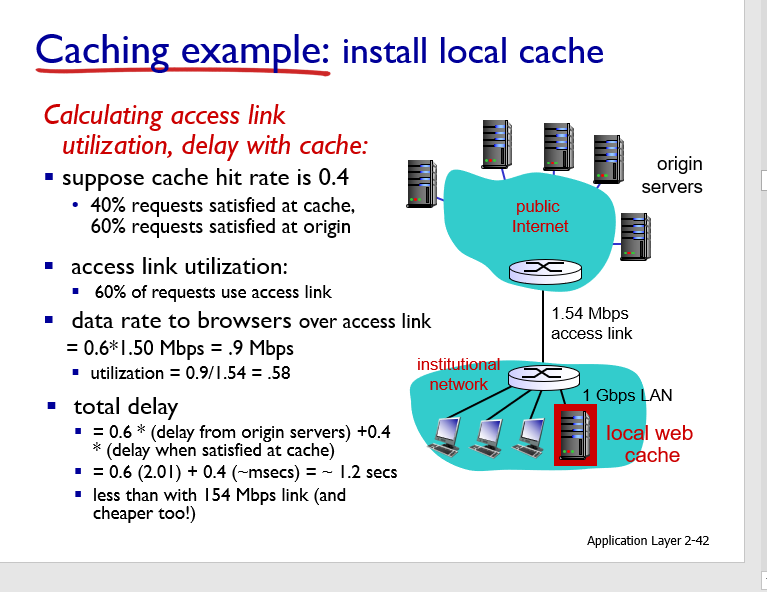
因此排队延迟小

但是代价就是很贵，小公司付不起



解决方案就是用caching

在所有原始条件的情况下 大量降低access link utizilation 以及total delay



cache hit rate //cache有所需object的几率

假设这个几率为0.4， 那么就相当于原始内容供应商的access link utilization只有60％

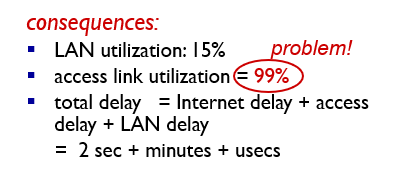
data rate(从浏览器通过access link到原始内容供应商),=原来的1.5mbps \*0.6, =0.9mbps

然后再算

utilization 0.9/1.54=.58 (不是直接.60)

总共的延迟：0.6\* delay from orgin server +0.4 delay when cache

甚至比超贵的方法还好

原来：

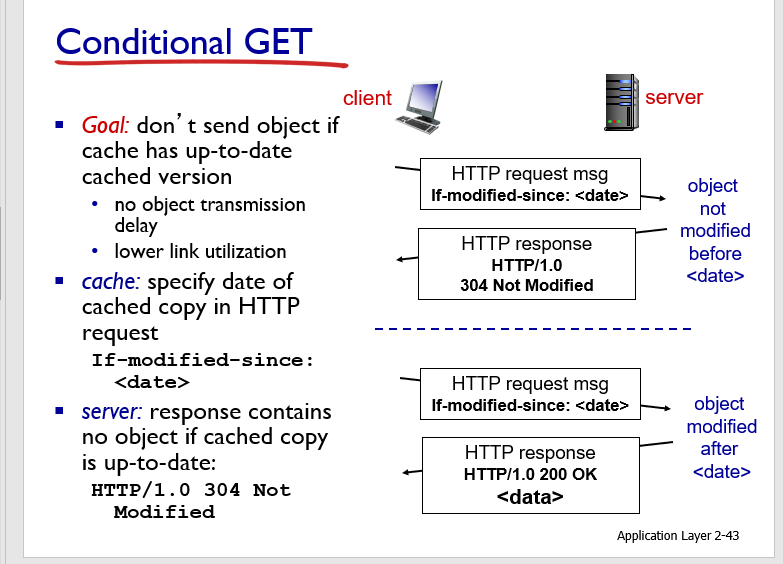
Conditional GET

cache: 在HTTP request中详细说明cached版本的日期，

If-modified-since<date> ,从XX改变过吗

server:如果是最新的，那就发出response,这个response 不包含object

http/1.0 304，没有改变过



如果是200，那就是改变过，要附着OBJECT

3.4 electronic mail

SMTP POP3 IMAP

electronic email 总共有三个主要部分

user agent 用户代理//客户端

mail server 邮箱服务器

simple mail transfer protocol: SMTP 简单的邮件传输协议

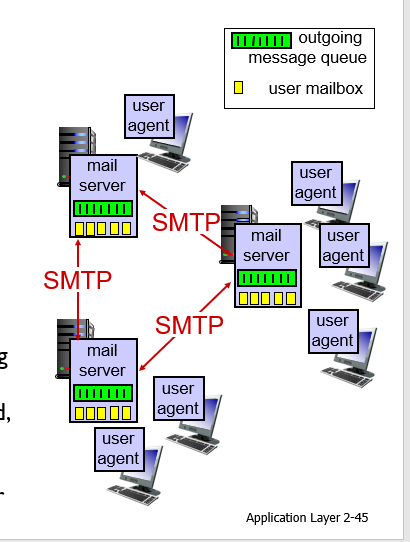
User Agent:

实际上就是邮件阅读者,

程序允许你撰写编辑阅读邮件信息 compose mail撰写邮件

Outlook,Thunderbird等等

发出与接受到的文件都储存在server



看可以看出实际上的运行都是在mail server上的

mail server的构成部分

mailbox保存了用户的incoming message (收件箱)

message queue: 缓存区， 存储了那些将要发出的邮件

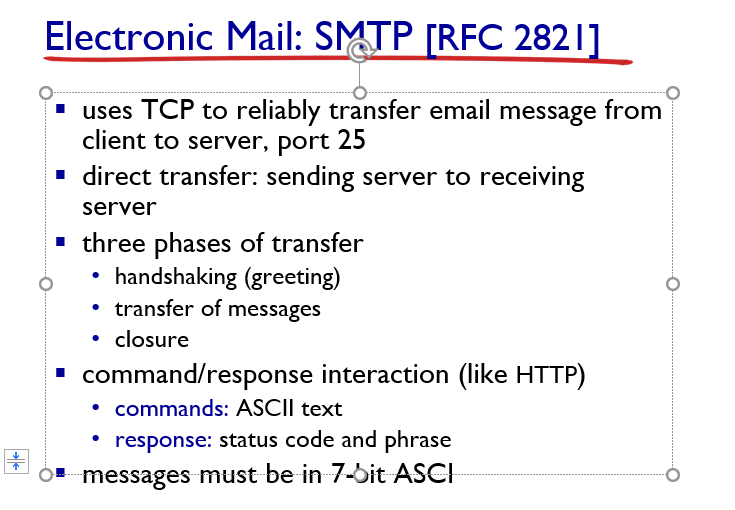
SMTP protocol: 使用TCP ，port 25

在mail server之间来回发送邮件的protocol

client：发出邮件的server

"server" "接收邮件的server

SMTP



是使用TCP来稳定的传输email ，从client到server, port为25

direct transfer:：一条路从sending server 到 receiving server

three phases of transfer

第一阶段:handshaking(建立连接，也叫shaking)

第二阶段：传输信息

第三阶段：closure,关闭

Command/response交互

command指令的形式:ASCII 文本

response:状态码status code 与短语phrase

Scenario剧情，场景: ALICE给bob发email

1.alice使用UA （user agent）撰写邮件（compose message）给bob@someschool.edu

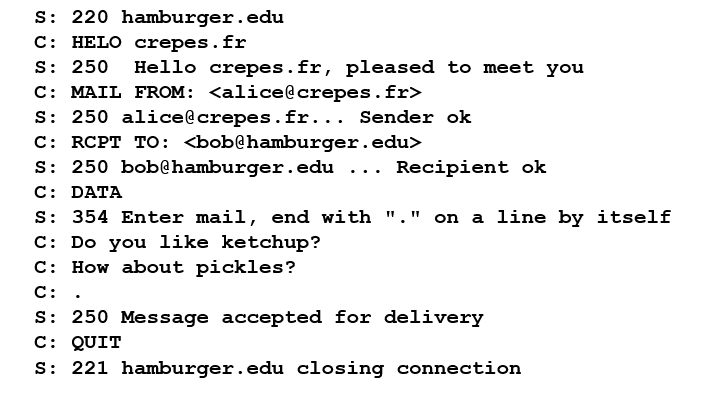
2. Alice的UA把她的邮件发给她的mail server. 邮件暂时存储在message queue

3. SMTP的client side(server分为发出邮件的client与接收邮件的server)打开了与BOB mail server的TCP链接

4. SMTP client side 把alice的邮件通过TCP connection发出

5. bob 的mail server把邮件存放在mailbox

6. bob打开自己的user agent 来阅读信息



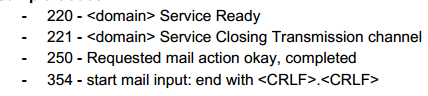
S： server

C: CLIENT

phase1 hand shaking

phase2: exchange of messages

phase3: closure

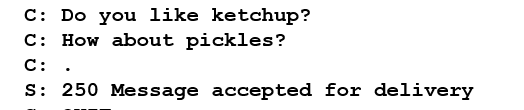


SMTP: final words

smtp 使用的是稳定的链接 （与HTTP不同之处在于，我们用一个链接交换多个信息）

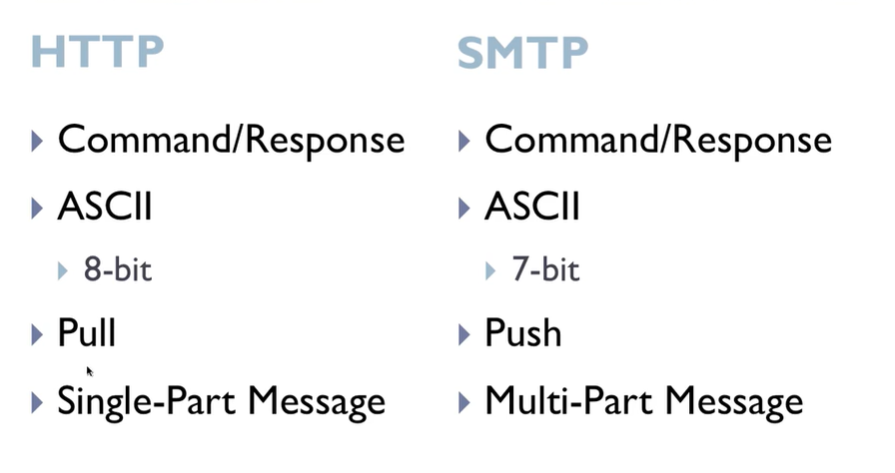
SMTP需要message(header& body)是7-bit ASCII码（而不是8-bit）

SMTP server用CRLF.CRLF来决定message 的end //CRLF就是换行

意思就是

点单独占一行就代表着信息的完结

SMTP与HTTP的比较



都是Command/Response, 用指令 回复啥的

都是ASCII码，但是HTTP是8bit, SMTP是7bit

HTTP 更倾向于pull:发出请求从原站点得到一个object

SMTP更倾向于push: 把文件从client server push 给server server

HTTP支持single-part message:一个connection发一个object

SMTP支持multi-part message:一个connection发多个object

SMTP:交换邮件的所用的Protocol

RFC 822: 标准的邮件格式

header lines:

TO:

From：

Subject://主题

这些To， From, Subject 是与SMTP指令（FROM,RCPT TO）不同的

我们直接空一行

然后写BODY，只能接受ASCII码

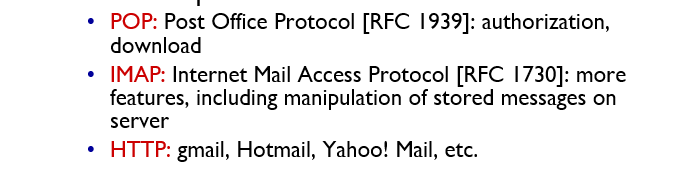
mail access 协议

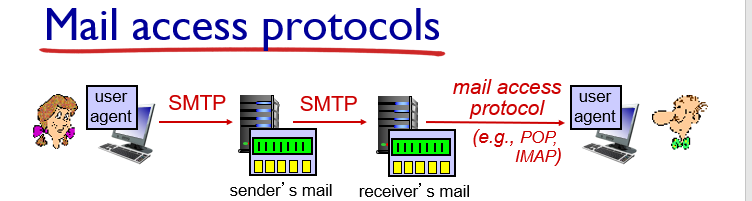
SMTP：发给收件人的server，存储在server所用的协议

然后用户从user agent取回邮件还需要协议，也就是mail access protocol//进入邮箱的协议

分为POP,IMAP,HTTP

POP与IMAP是更老的protocol用RFC描述





注意发件人与server之间仍然是SMTP

只有收件人从他的server mailbox中提取所用的是mail access protocol //pull

POP3协议

authorization phase:授权阶段，也就是输入密码用户名

client的指令有 user输入用户名，pass输入密码

sserver的回复有，OK,ERR//error

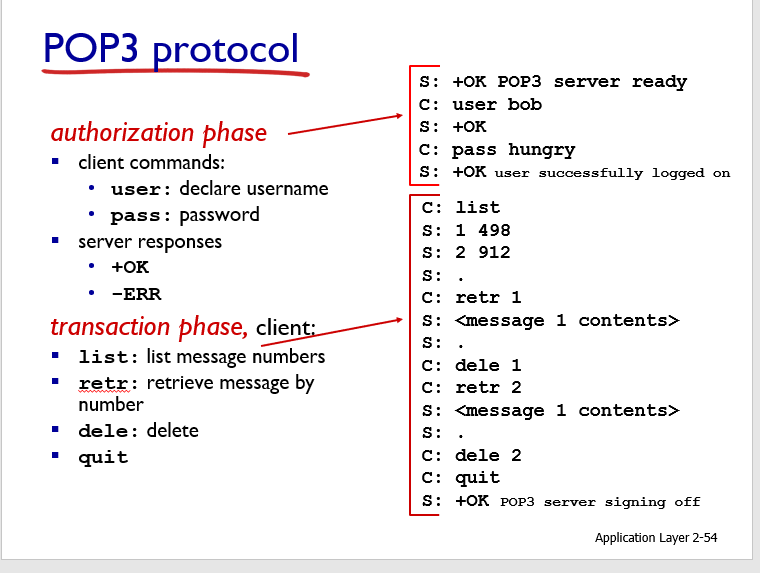
transaction phase: 事务处理阶段

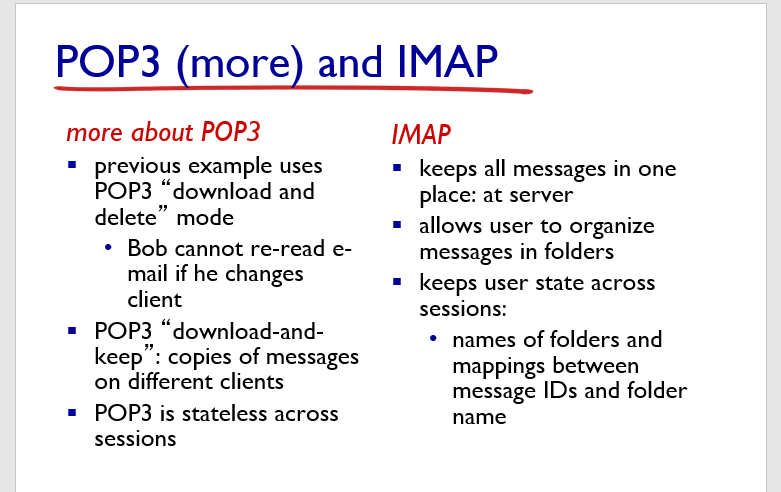
List:列出有哪些邮件，只列出邮件数字

retr 加数字，通过数字提取邮件

dele ：delete

quit离开邮箱，会恢复OK





POP3：

只有两个mode,

1.download and delete:下载下来以后就不留了，如果此时退出就永远看不到了

2.download-and-keep 下载的只是copy

POP3是stateless的

IMAP:

把所有邮件keep在server处

允许用户管理邮箱里的信息

保持state，folder的名字…..

DNS： domain name system //域名系统

人类的identifier识别：SSN,NAME,PASSPORT

怎么识别网络的主机host与router地址呢

两组name

IP address(32 bit)： computer更好识别的identifier，computer-readable

domain name/hostname：www.yahoo.com,是Human-readable

DNS: Domain Name System

将IP address 与 domain联系起来

DNS三要素：**distributed, hierarchal, database**

distributed hierarchy database

distributed 意思是碎片化的，分散的

这个database是以hierarchy的形式安装的//hierarchy层级的，意思是他并不是完全无序分散，而是按照**name servers名称服务器**的形式分类分层. database中不仅存放了ip address,也存放了结构信息

Application-layer protocol

hosts, name servers与resolve names解析域名互动（地址，名字翻译）

这个协议是internet的核心，是application-layer-protocol

**A DNS server is a computer server that contains a database of public IP addresses and their associated hostnames**

DNS的服务以及结构

DNS的服务 //

hostname到IP address的翻译

host aliasing主机别名：canonical主域名, alias names别名

mail server aliasing邮件服务器别名

load distribution负载分配：

为什么不让DNS 中心化centralize 而使用分散式

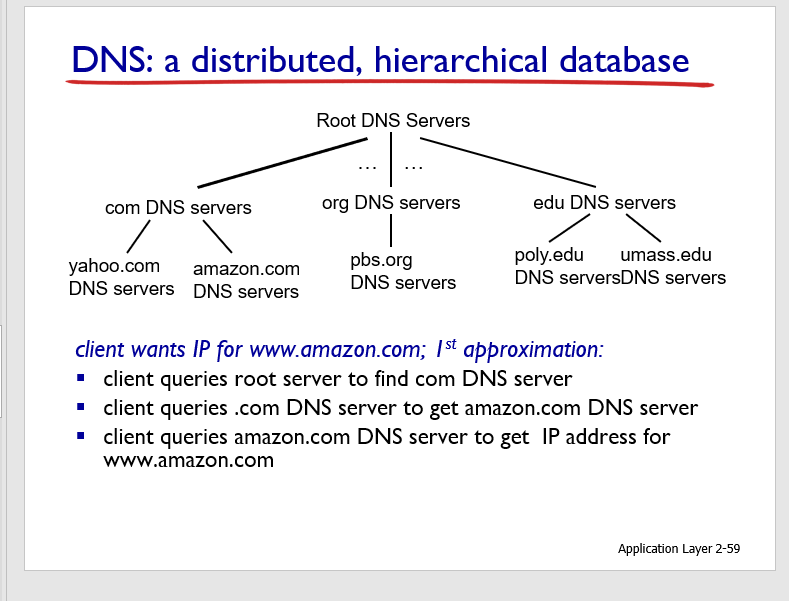
1.Single point of failure: 如果一个DNS database炸了那别人就没法用了

2.traffic volume:让一个database支持所有人的DNS系统，超负荷了，所以用分散式负载分配

3.distance centralized database: 如果服务器在美国，美国人可能延迟低，别的人延迟高

4.maintenance:这个database hold 所有信息，难以维护

**doesn't scale:最重要的一条，我们没法快速度的拓展这个数据库**



你想得到 amazon.com的IP address

首先你请求root dns server，找到COM server(里面存着所有以com结尾的网址)

然后你再请求.com DNS server,找到amazon.com DNS server

最后你请求amazon.com 得到www.amazon.com 的IP address

所以总的来说有三层

第一层:root name servers

第二层：TLD

第三层，authoritative name server

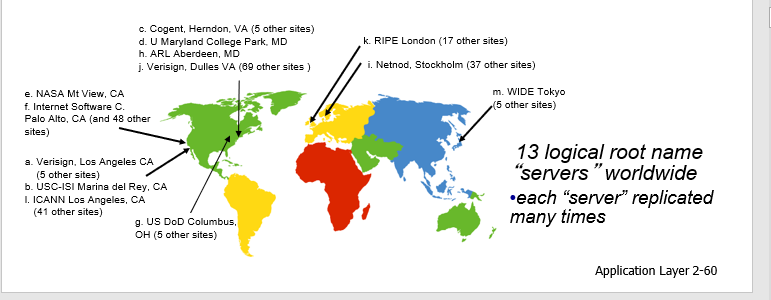
DNS: root name servers 根域名服务器

被local name server （最基础的server）所连接，不能解析域名

如果name mapping不知道，联系authoritative name server有权威的域名服务器（不知道域名对应的ID，他不会自己分析，而是向高阶服务器要答案）

gets mapping，成功对应

returns mapping to local name server，把对应的IP发还给local name server



全世界有13个root name server

TLD: authoritative servers

top-level domain servers:

负责切割各种大类的domain例如.com.org.net

或者top-level country domain国家.ca.jp.uk

Authoritative DNS servers:

组织（例如amazon）自己的DNS server，对request的name 提供最终的IP mapping。

可以通过organization或service provider来保持

Local DNS name server：

严格来的说并不属于DNS hierarchy//三层结构

每个ISP（地区ISP，公司ISP，大学ISP）都有一个local name server，也叫做default name server

当host终端想要查询（query）dns,这个query被送往local dns server处

这个local server有local cache，或许保存着最近的name-to-address translation，（就是name到IP address,）但有可能过期

如果没有，就想代理人（proxy）一样，把query发放给hierarchy层级结构

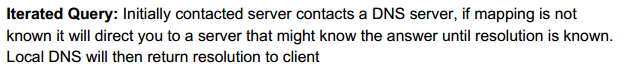
DNS name resolution example //resolution解决，分辨

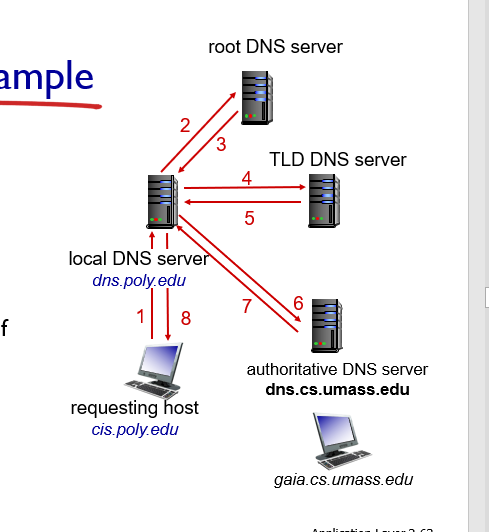
对于Query处理一共有两种方案： //query 查询

iterated query

Recursive query

Iterated query:





你是cis.poly.edu的终端，想要知道gaia.cs.umass.edu的 ip address

1.你先给你的local dns server发出这个query

2.然后你的local dns server发query给root dns server,问他edu的TLD server在哪，

3.得到地址以后

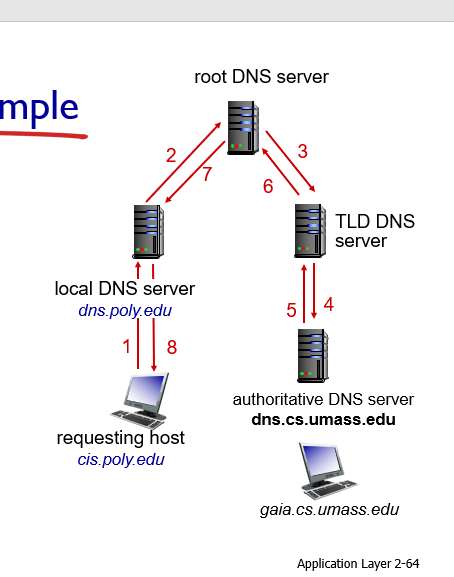
4.再问edu TLD server，umass.edu 的地址在哪，

5.得到具体地址

6.再把query发给authoritative dns server，

7.得到具体IP，

8.发还给requesting host



recursive query:

这个就比较简单，一路向上就完事儿

把name resolution的burden一路上传



区别在于recursive query给上层的内置链接的server load

而iterative是给local dns server load

caching, updating records

任何name server成功配对ip address与name，都会cache缓存这个mapping

//因为有的问题一天被问好多次，例如google.com这种，没必要一路上传

timeout: cache entries会失效在某一时间后（TTL）

TLD server的地址通常在local name server那里缓存//这样就没必要去问root name servers

cached entries 有可能过时，如果有人个改变了他的IP address， internet不能即使知道，直到所有的TTLs过期

DNS records

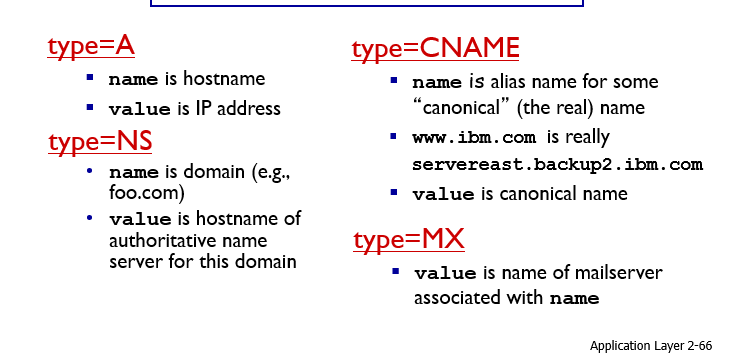
DNS: 离散数据库存储着RR

RR：resource records 资源记录



TTL是time to live 也就是说这个数据的持续时间

而name value取决于type



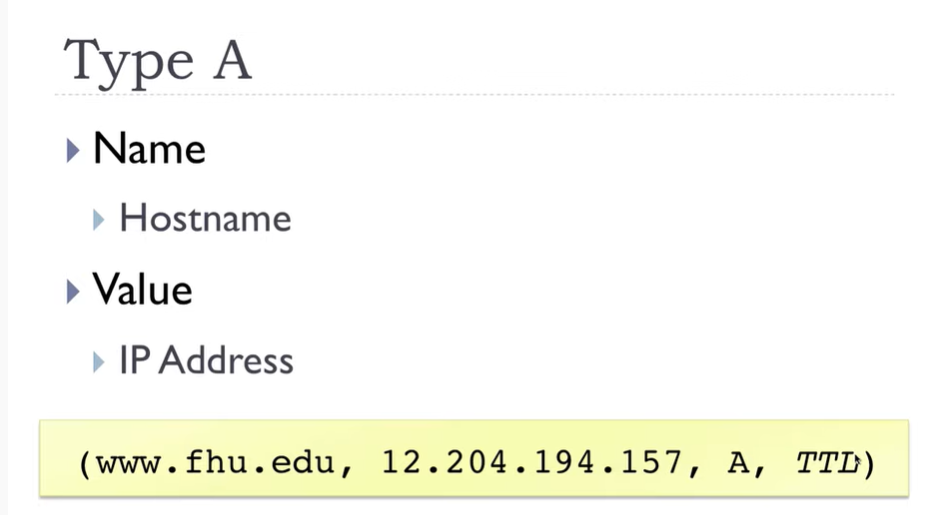
如果type是A， name是hostname（authoritative server 的host name）, value是IP地址 //针对host

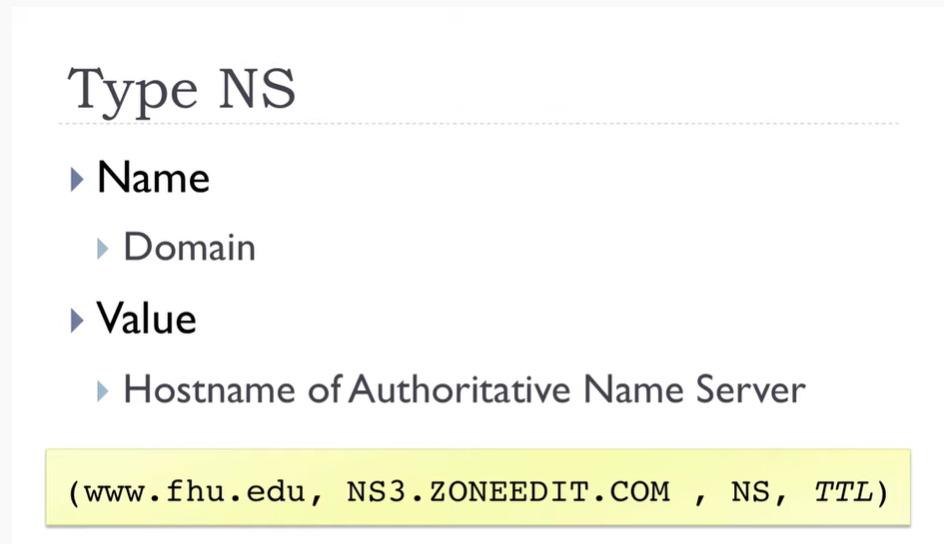
如果type=NS, name是domain name一个网址名字，value是它对应的authoritative server的hostname主机名字

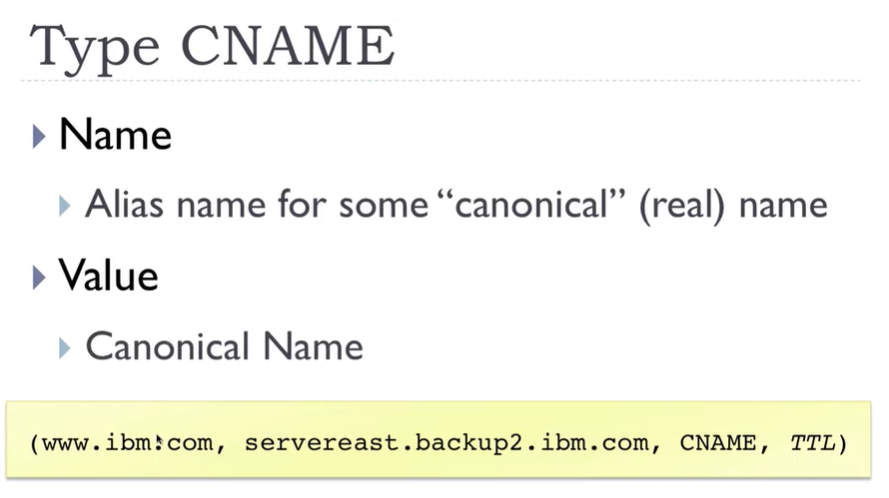
如果type=cname，name是真实canonical名字的alias， 例如 ibm。com的真实域名其实是servereast.backup2.ibm.com(真实域名太丑了，用一个alias代替他)，value是真是域名

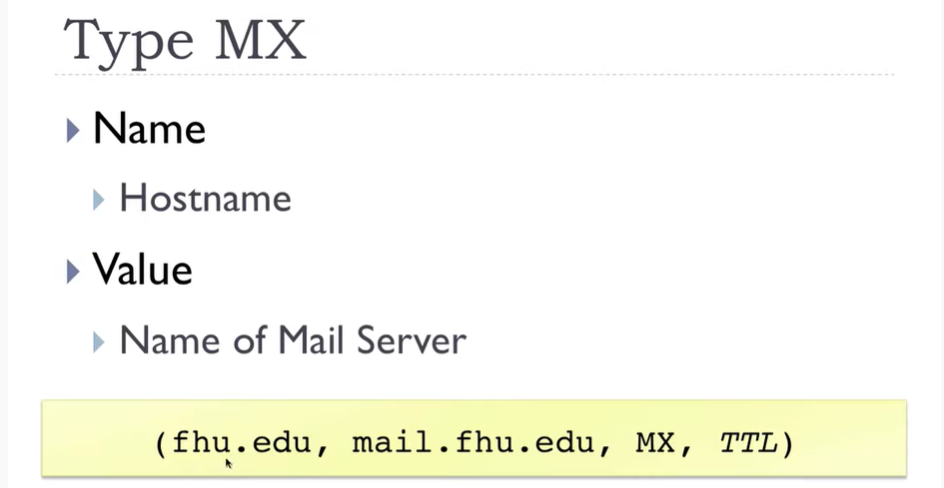
type=MX

name是domain网址，value是这个 mail server的名字

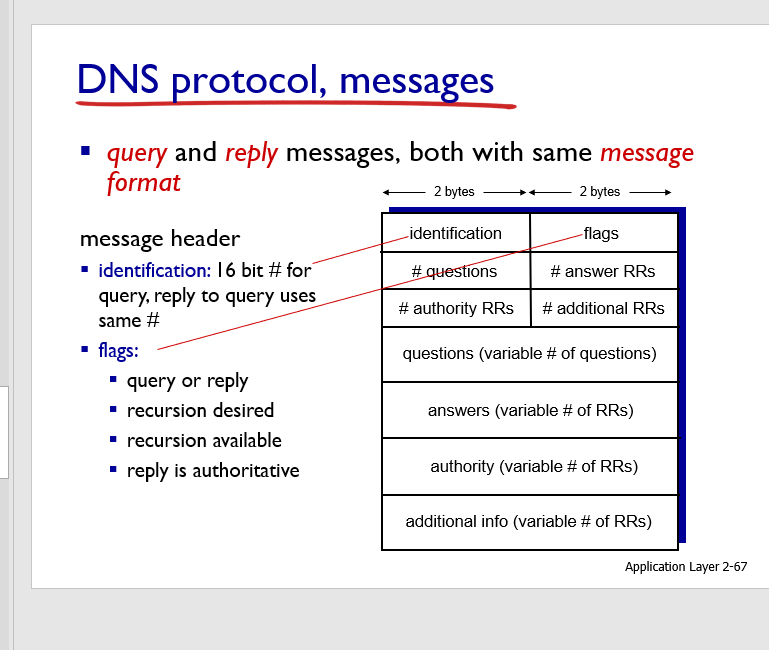








DNS协议发的信息

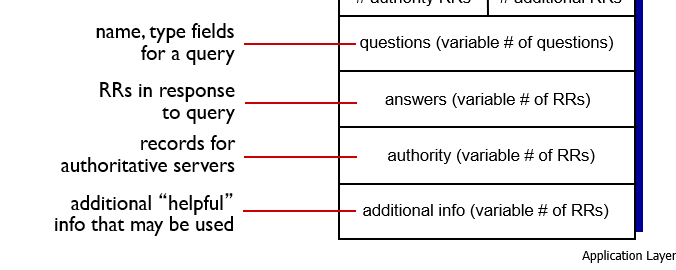


几个要点

query与reply都是同样的格式message format

identification：16bit ，reply与query用同样的数字

flag:告诉你是query还是reply， 是否需要recursion



下面四个就是对应信息

inserting records into dns

如果你想要注册一个网站，你需要注入一些信息给dns

例如你想创造一个叫utopia的网站

首先你要在DNS registrar域名注册商 那里登记这个domain name networkutopia.com

需要提供两个RR

**(networkutopia.com, dns1.networkutopia.com, NS)**

**(dns1.networkutopia.com, 212.212.212.1, A)**

NS的那个，域名，以及其对应的authoritative name server的host名字

A的那个，实际host name//也就是我们自己建立的authoritative name server，以及对应的IP address

P2P application

no always-on server

arbitrary任意end systems

直接directly相连

intermittently Connections间歇的相连

changing IP address一直变动的IP address

例子：BT下载，直播

File distribution文件分发

问题：把一个size为F的文件从一个服务器分发给N个peer需要多少时间

Peer指的是同级的Host

Us：server 上传带宽

Ui: peer upload带宽

di: peer download带宽

Client Server

server transmission:

必须依次的sequentially 发出或上传N份副本

如果发出一个·copy的时间是F/Us

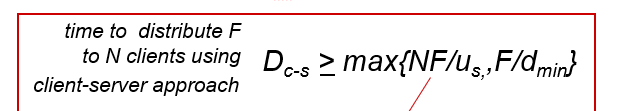
发出N份就是NF/Us

client： 每一个client必须下载这个file copy

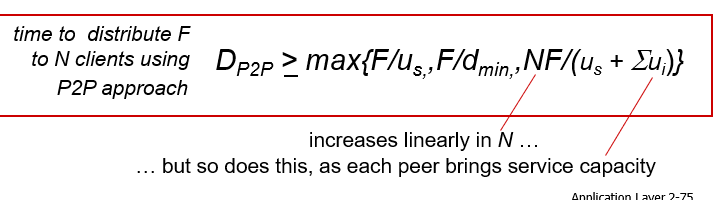
假设下载的最慢的人用的速度为dmin

那么下载时最慢的人要用F/dmin

那么把F大的文件发给N个·用户的时间是

取决于括号内最大值，而且NF/us随着用户越来越多，线性变大

而P2P



server

只要上传一份copy //然后下载到的人会上传，下一个人再下载他的，随着下载的人越来越多，只要Upload一部分就行

F/Us

client:每个client都要下载

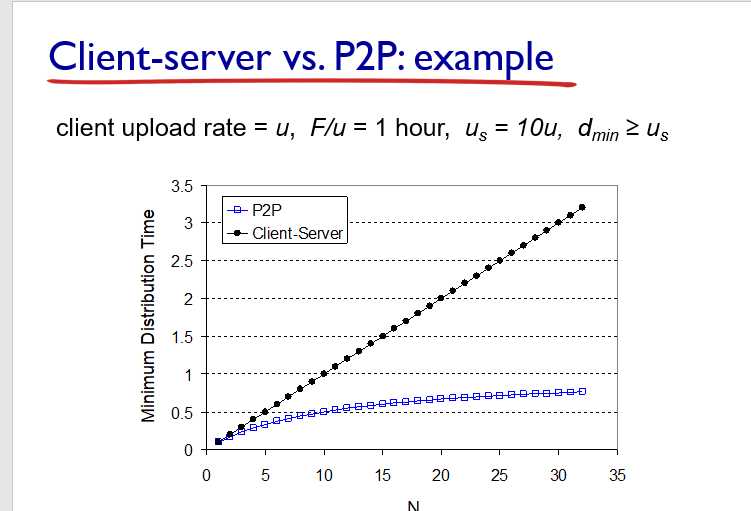
最慢的下载时间F/dmin

aggregate data

其实如果N个人下载，还是相当于上传了N次，所以总共要上传的bit为NF，

但是区别在于速度=

us是第一次原始上传，Eui是每个client上传速度的总和



随着用户越来越多，Client-Server所需时间线性增长，而P2P平缓

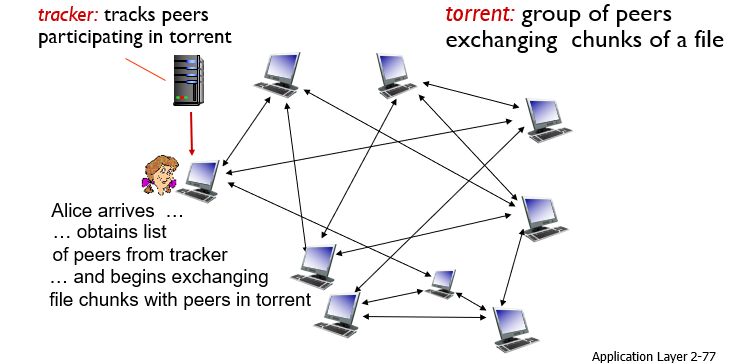
BitTorrent//torrent 奔流

把文件分成256kb的chunks小块

在torrent流中的peer都会发送接受file chunks

tracker追踪者:追踪流中的peer

torrent:一组peer，这组peer交换file的chunks



alice想要下载，从tracker处得到peer list，然后与peers互换chunks

一开始加入torrent的peer:

初始没有chunk，但是随着时间累积chunk

在tracker处登记，同时也拿到list of peers，与peers相连接

当下载的时候，peer也会上传chunk给别的peer

churn:搅动，peer能加入这个torrent自然也能退出torrent

一旦peer完成了整个file，他可以自私的退出或者利他的留在torrent中

BitTorrent: Requesting, sending file chunks

requesting chunks

在任意时间，不同的peer有不同的file chunks子集

间歇性的periodically，alice会向每一个peer索要他们拥有的**list** of chunks

alice从peers索要他没有的**chunks ，**最稀有的（拥有的人少的） 优先

Sending chunks:tit-for-tat

alice把她的chunks发给四个peers——当前给他chunk频率最高的四个人

其他peer则不会从ALICE处收到chunk

每十秒重新评估一次top4

optimistically unchokes（积极疏通）: 同时每30秒随意选择一个peer给他们发chunk

新选择的peer包括top 4

例子：

ALICE optimistically unchokes了bob

alice变成了bob的top4之一，bob也给她发送chunk

boob变成了Alice的top4之一

Video Streaming and CDN（content distribution networks）

video traffic:播放视频占据了主要的当前网络带宽

问题之一：heterogeneity，不均匀的，每个用户网速不一样

解决方案：distributed分布式, application-level infrastructure基础设施

video是一串稳定频率的图片

解决方案：使用图像内部与之间的redundancy（重复的数据）来减少需要改变的bit数量

两种方案

spatial coding：假设一个图片是紫色的，不直接把所有N个像素点发送，而是只发送两个值，value(purple),重复的数量（N）

temporal coding: 与其发送整个画面，只发送与上一帧不同的像素点

CBR: constant bit rate 恒定比特率， 视频编码速度恒定

VBR: variable bit rate:变化比特率 视频编码速度取决于spatial coding ,temporal coding的改变量

流媒体：DASH

dynamic, adaptive streaming over http

http的动态自适应流

server:

把video file切割成多个chunk

每个chunk被存储编码成不同rate

manifest file: 提供不同chunk的url链接

client:

周期性的检查server-to-client 带宽

咨询manifest file， 一次request一个chunk

选择现有带宽下最大的码率的chunk

DASH这个模式

client 来决定：

什么时候需求chunk

什么码率encoding rate

where to request chunk（从URL服务器选择离client近的或者有更高码率得URL）

Content distribution networks

怎么把直播内容同时播给好多人一起看

第一选择：超级大的mega-server

为啥不行

他出现错误直播就炸

很贵

离得远的人没法看

最主要的是：这个答案不能scale缩放

第二选择：复制多份copy地理均匀的放在多个site上(CDN)

enter deep: 把CDN servers推入多个access network

bring home: pop到离得近的access network

CDN:把content的多个copy存储在CDN nodes

订阅者从CDN request content就会从近的 node

Socket programming:

两种socket type 对应两种transport services

UDP: 不可靠的datagram数据段

TCP: 可靠的 byte stream

UDP socket programming

client与server之间没有connection

在发送数据之前没有handshaking

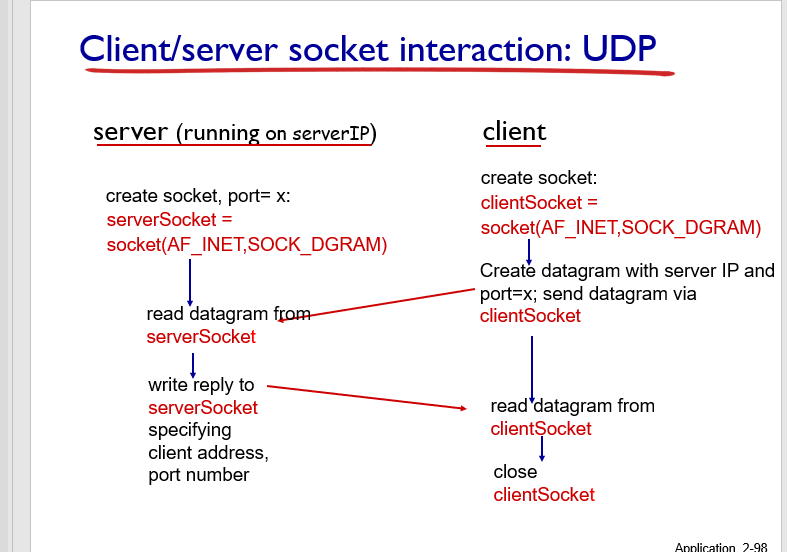
sender明显的把IP目的地址以及port number附着在每个packet上

receiver从received packet 抽离sender的IP地址以及port number

传输的数据或许会丢失或者顺序出错

应用角度

UDP 提供了不稳定的传输，传递datagrams(一组bytes) 在client 与 server之间



server：

创建socket, port=x

client创建socket

创建datagram（参数是server的port与IP）

然后通过clientSocket把datagram发出去

server从serverSocket读到datagram

写下reply，指定client address,port# 并通过serverSocket发出

client从clientSocket读到datagram

client关掉clientSocket

TCP socket programming

client必须先联系server

因此server首先一开始是running状态

server也必须提前创建socket来欢迎client

client 怎样联系server

创造TCP socket ,指定server process的IP address与port number

当client 创造socket的时候，client TCP会建立与server TCP 的connection

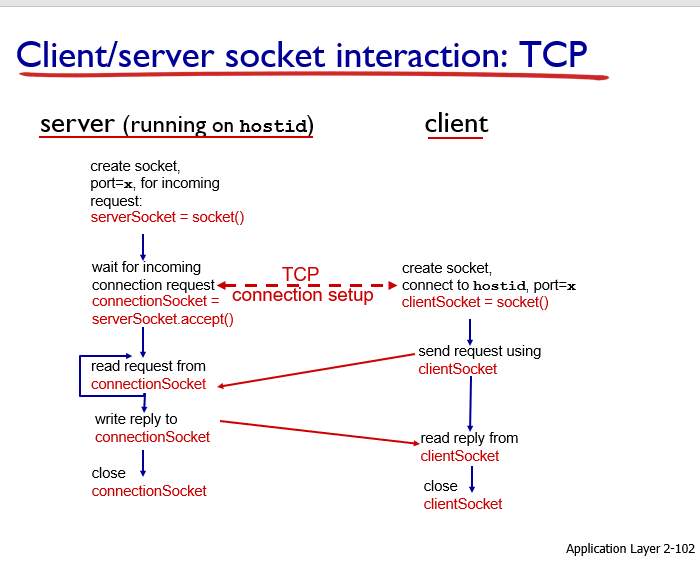
当被client所联系，server TCP会创造一个新socket 来与这个特定的client交流

允许server与多个client交流

source port numbers被用来区分client

application观点

TCP提供了可靠的管道在client与server之间



server创造socket,port # 为了迎接request

等待 连接请求

这时client创建socket， 指定server process的hostid与port#

创建的一瞬间就会发出请求

然后serverSockket accept

这样就创建好了TCP 链接

client可以不停用clientSocket发出请求

connection Socket读这些请求

并且做出一次统一的回应通过connectionSocket，关闭

clientSocket收到回复，关闭