Computer Networks

Chapter 7: Application Layer (2/3)

(Version February 27, 2023)

Xiao Mingzhong

CIST, Beijing Normal University

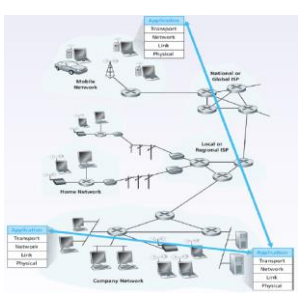
Review

- 从E2E的传输抽象
- 到
- 使用TCP/UDP的Socket 程序设计
  - Tcp/udp是个什么样的服务
  - 如何使用它们? Socket程序设计.
- Now, 是琢磨应用及其协议的时候了.

应用层

- Domain Name System(DNS)
- Electronic Mail*
- FTP*
- World Wide Web(WWW)
- Multimedia*

- Peer-to-Peer应用
  - Napster,Maze
  - Gnutella,BitTorrent
  - DHT(Chord,Kadmelia)

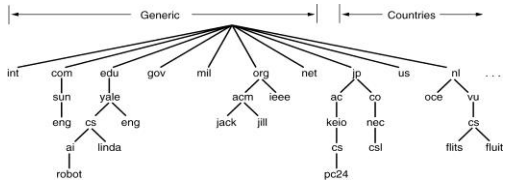


DNS 重要性

- Motivation(动机)
  - 通过网络地址直接访问资源，地址记忆难
  - 机器名与机器地址分离的需要
- Host.txt解决策略
  - 某机器记录所有主机的IP地址及其对应名字
  - 网络主机周期下载此文件到本地，以实现<名,址>转换
- 存在扩展性问题
  - DNS解决方案

域名系统(DNS)

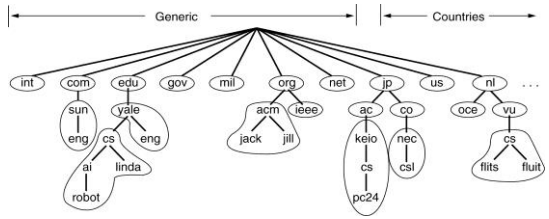
- Basic idea: 每台主机都有一个全球唯一的名字跟它的IP地址关联，由DNS系统来提供按名字查找IP地址的服务. 即：提交一个主机名,该系统返回该主机的IP地址. 咋命名?(层次命名法，以net.bnu.edu.cn为例)



- 从叶子节点向上到达树根，一路所遇节点名串联起来就是该主机的名字，称为域名,用"."分隔. 一个域在域名空间树中就是一棵子树.
- 每个域应该部署有相应的名字服务器(name server)，来负责存储和管理其下游名字跟IP地址的对应关系.这种对应关系通常以文件形式存储，专业术语资源记录(resource records)

DNS名字服务器

- 层次表达的名字空间被划分成一系列非重叠的区域(zones)，每个区域由一个或多个 name servers管理，该NS由其上一个NS管理:



- Note: 之所以多个服务器，备份作用. "从"服务器通常需要跟"主"服务器保持数据一致. "主"服务器上的信息由域名管理者维护.
- 名字解析器(resolver) 负责向名字服务发起DNS请求. A resolver 通常以若干库例程(library routines)的方式被"链接"(linked)进应用程序中. 如:gethostbyname(...);

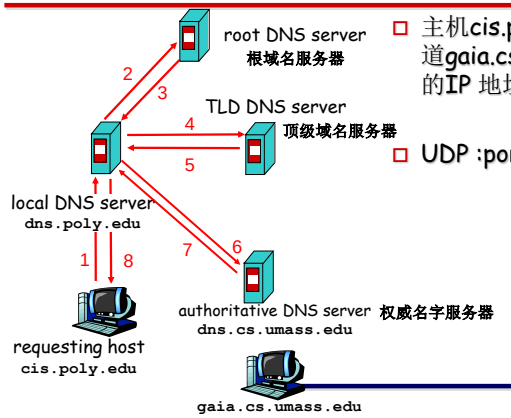
资源记录(Resource records)

Type	Associated entity	Description
SOA	Zone	Holds information on the represented zone
A	Host	Contains an IP address of the host this node represents
MX	Domain	Refers to a mail server to handle mail addressed to this node
SRV	Domain	Refers to a server handling a specific service
NS	Zone	Refers to a name server that implements the represented zone
CNAME	Node	Symbolic link with the primary name of the represented node
PTR	Host	Contains the canonical name of a host
HINFO	Host	Holds information on the host this node represents
TXT	Any kind	Contains any entity-specific information considered useful

Resource Record (Example)

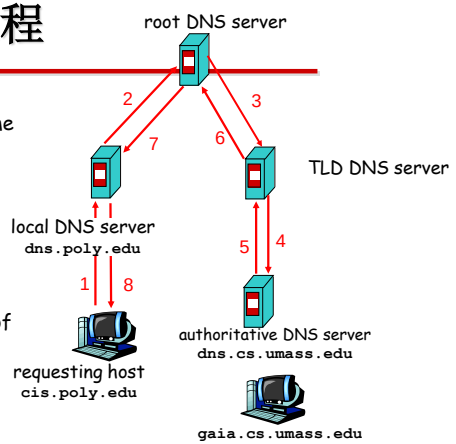
```
; Authoritative data for cs.vu.nl
cs.vu.nl.       86400   IN  SOA     star boss (952771,7200,7200,2419200,86400)
cs.vu.nl.       86400   IN  TXT     "Divisie Wiskunde en Informatica."
cs.vu.nl.       86400   IN  TXT     "Vrije Universiteit Amsterdam."
cs.vu.nl.       86400   IN  MX      1 zephyr.cs.vu.nl.
cs.vu.nl.       86400   IN  MX      2 top.cs.vu.nl.

flits.cs.vu.nl. 86400   IN  HINFO   Sun Unix
flits.cs.vu.nl. 86400   IN  A       130.37.16.112
flits.cs.vu.nl. 86400   IN  A       192.31.231.165
flits.cs.vu.nl. 86400   IN  MX      1 flits.cs.vu.nl.
flits.cs.vu.nl. 86400   IN  MX      2 zephyr.cs.vu.nl.
flits.cs.vu.nl. 86400   IN  MX      3 top.cs.vu.nl.
www.cs.vu.nl.   86400   IN  CNAME   star.cs.vu.nl
ftp.cs.vu.nl.   86400   IN  CNAME   zephyr.cs.vu.nl

rowboat                 IN  A       130.37.56.201
                        IN  MX      1 rowboat
                        IN  MX      2 zephyr
                        IN  HINFO   Sun Unix

little-sister           IN  A       130.37.62.23
                        IN  HINFO   Mac MacOS

laserjet                IN  A       192.31.231.216
                        IN  HINFO   "HP Laserjet IIISi" Proprietary
```

DNS 的迭代名字解析过程



- 主机cis.poly.edu 想知道gaia.cs.umass.edu 的IP 地址
- UDP :port 53

递归解析过程



recursive query:

- puts burden of name resolution on contacted name server
- heavy load?

iterated query:

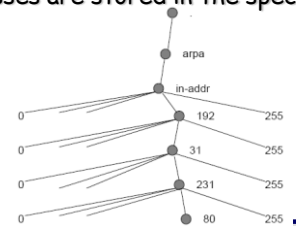
- contacted server replies with name of server to contact
- "I don't know this name, but ask this server"

DNS Address Resolution(地址解析)

- Guess what: you can also find the name of a host when given its address:

```
% host  192.31.231.80
% 80.231.31.192.in-addr.arpa domain name pointer veldersschuit.cs.vu.nl.
```

- Solution: IP addresses are stored in the special in-addr.arpa domain:



Round robin DNS(轮回DNS)

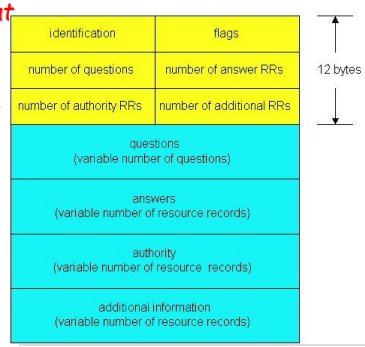
- 为地理上分布的多个服务器进行负载平衡用.
- 原理是针对DNS查询返回的不是一个IP地址,而是几个.
- e.g., executing host www.google.com returns
  - www.google.com is an alias for www.l.google.com.
  - www.l.google.com has address 66.249.91.99
  - www.l.google.com has address 66.249.91.104
  - www.l.google.com has address 66.249.91.103
  - www.l.google.com had address 66.249.91.147
- Each DNS reply has associated a TTL(生存期), telling how long it can be cached.
- 某些情况下,返回的IP地址跟请求者的地理位置有关. (e.g., Akamai) 出于就近获得服务的目的,前面是负载平衡目的.
- Lab tools: nslookup & dig

DNS protocol messages

DNS protocol : *query* and *reply* messages, both with same *message format*

msg header

- identification: 16 bit # for query, reply to query uses same #
- flags:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative



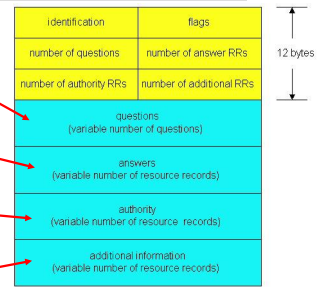
DNS protocol, messages

Name, type fields for a query

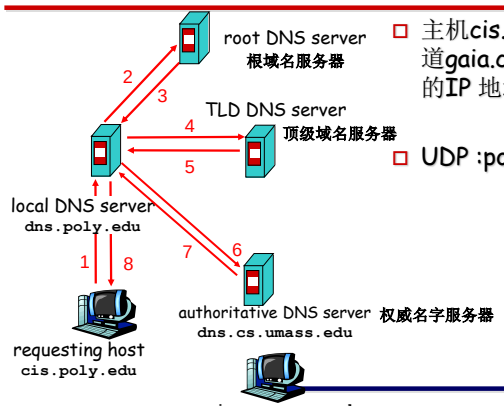
RRs in response to query

records for authoritative servers

additional "helpful" info that may be used



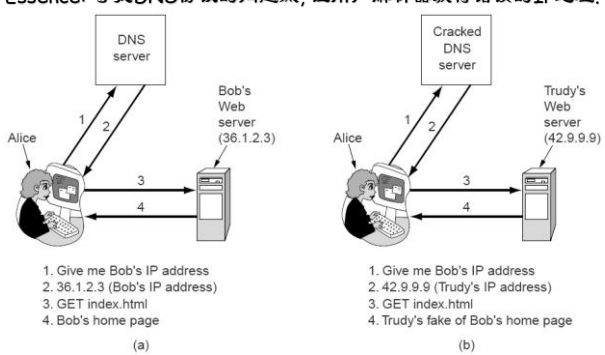
DNS 的迭代名字解析过程(回顾)



- 主机cis.poly.edu 想知道gaia.cs.umass.edu 的IP 地址
- UDP :port 53

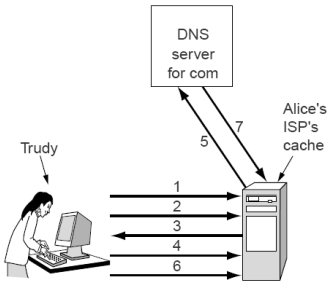
DNS 攻击例

- Essence: 寻找DNS协议的问题点，让用户解析器获得错误的IP地址.



Steps:

- Observation: DNS 根据UDP包中 的"identification" 未关联"请求和应答".Trudy能发起DNS欺骗:
  - 假定: DNS cache初始为空(or cached entry已经过期)



1. Look up foobar.trudy-the-intruder.com (to force it into the ISP's cache)
2. Look up www.trudy-the-intruder.com (to get the ISP's next sequence number)
3. Request for www.trudy-the-intruder.com (Carrying the ISP's next sequence number, n)
4. Quick like a bunny, look up bob.com (to force the ISP to query the com server in step 5)
5. Legitimate query for bob.com with seq = n+1
6. Trudy's forged answer: Bob is 42.9.9.9, seq = n+1
7. Real answer (rejected, too late)

Web App. and HTTP Pro.

First some jargon

- 一个Web page 由若干 objects 构成
- Object 可能是HTML file, JPEG image, Java applet, audio file, Link, …
- Web page的objects用HTML语言来"标记"
- Each object 用一个 URL 标识, 它是可寻址的
- Example URL:

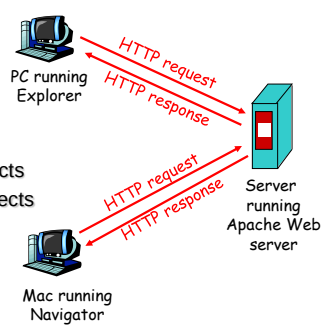
www.someschool.edu/someDept/pic.gif

host name | path name

HTTP overview

HTTP: hypertext transfer protocol 超文本传输协议

- Web's 应用层协议
- client/server model
  - *client:* browser that requests, receives, "displays" Web objects
  - *server:* Web server sends objects in response to requests
- HTTP 1.0: RFC 1945
- HTTP 1.1: RFC 2615



HTTP overview (continued)

Uses TCP:

- Client发起对服务器80端口的TCP connection请求
- Server接受TCP connection请求
- HTTP messages (应用层协议消息) exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed

HTTP is "stateless"无状态的

- server 不维护client的请求史

aside

Protocols that maintain "state" are complex!

- past history (state) must be maintained
- if server/client crashes, their views of "state" may be inconsistent, must be reconciled

HTTP connections

Nonpersistent HTTP
非持续的http连接

- At most one object is sent over a TCP connection. Web页一个对象一个TCP连接。
- HTTP/1.0 uses nonpersistent HTTP

Persistent HTTP
持续的http连接

- Multiple objects can be sent over single TCP connection between client and server.一个TCP连接上可收发多个对象。
- HTTP/1.1 uses persistent connections in default mode

Nonpersistent HTTP

Suppose user enters URL
`www.someSchool.edu/someDepartment/home.index`

(contains text, references to 10 jpeg images)

1a. HTTP client 发起TCP连接 to HTTP server (process) at www.someSchool.edu on port 80

1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80. "accepts" 连接, 并通知client

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index

3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket

time

Nonpersistent HTTP (cont.)

4. HTTP server closes TCP connection.

5. HTTP client 收到包含html 文件的应答消息, Brower显示 html. Client 解析html file, 发现10个对jpeg objects的引用

6. Steps 1-5 repeated for each of 10 jpeg objects

time

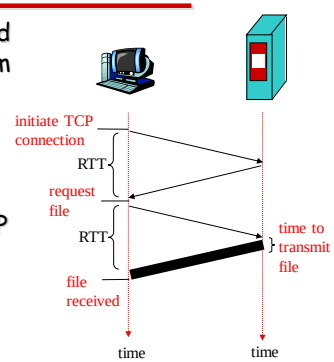
Response time modeling响应时间模型

Definition of RRT: time to send a small packet to travel from client to server and back.

Response time:

- 一个RTT用于发起TCP连接
- one RTT for HTTP request and first few bytes of HTTP response to return
- file transmission time

total = 2RTT+transmit time



持续HTTP

非持续HTTP分析:

- 要求2 RTTs per object
- 每次TCP连接都会引发OS开销
- browser often open parallel TCP connections to fetch referenced objects

持续HTTP

- 服务在响应请求后,不立即关闭连接
- subsequent HTTP messages between same client/server sent over open connection

无管道型持续http:

- 仅当前面的请求被响应, client才会发起新的请求.
- one RTT for each referenced object

管道型持续http:

- default in HTTP/1.1
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects

HTTP request message

- two types of HTTP messages: *request, response*
- HTTP request message:
  - ASCII (human-readable format)

request line (GET, POST, HEAD commands)

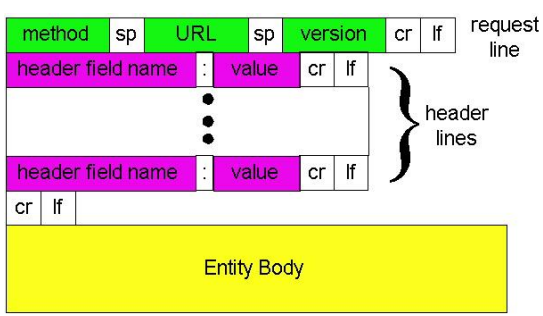
```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

header lines

Carriage return, line feed indicates end of message

(extra carriage return, line feed)

HTTP request message: general format



Uploading form input

Post method:

- Web page often includes form input
- Input is uploaded to server in entity body

URL method:

- Uses GET method
- Input is uploaded in URL field of request line:

```
www.somesite.com/animalsearch?monkeys&banana
```

Method types

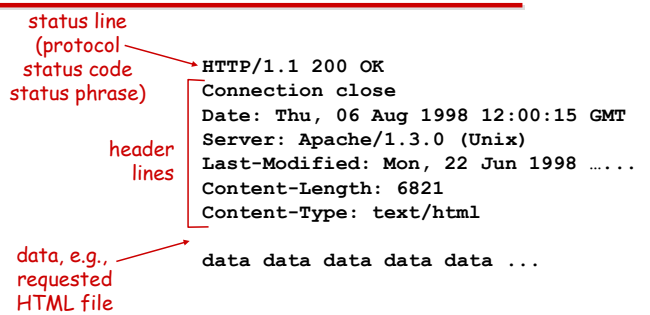
HTTP/1.0

- GET
- POST
- HEAD
  - is similar to GET, but asks server to leave requested object out of response

HTTP/1.1

- GET, POST, HEAD
- PUT
  - uploads file in entity body to path specified in URL field
- DELETE
  - deletes file specified in the URL field

HTTP response message



HTTP response status codes(应答状态码)

In first line in server->client response message.
A few sample codes:

`200 OK`
- request succeeded, requested object later in this message

`301 Moved Permanently`
- requested object moved, new location specified later in this message (Location:)

`400 Bad Request`
- request message not understood by server

`404 Not Found`
- requested document not found on this server

`505 HTTP Version Not Supported`

Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

```
telnet cis.poly.edu 80
```

Opens TCP connection to port 80 (default HTTP server port) at cis.poly.edu. Anything typed in sent to port 80 at cis.poly.edu

2. Type in a GET HTTP request:

```
GET /~ross/ HTTP/1.1
Host: cis.poly.edu
```

By typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server

3. Look at response message sent by HTTP server!

User-server state: cookies

Many major Web sites use cookies

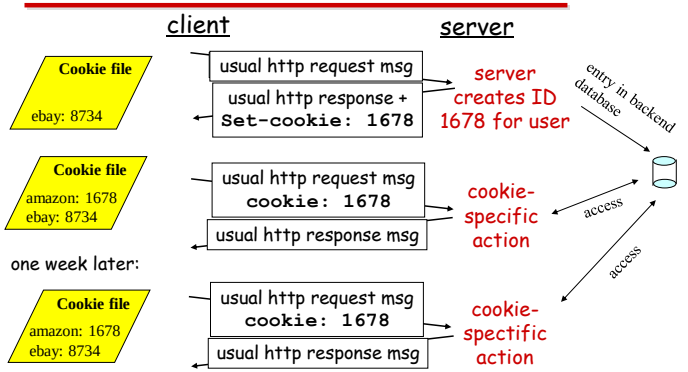
Four components:

1) cookie header line of HTTP *response* message
2) cookie header line in HTTP *request* message
3) cookie file kept on user's host, managed by user's browser
4) back-end database at Web site

Example:

- Susan access Internet always from same PC
- She visits a specific e-commerce site for first time
- When initial HTTP requests arrives at site, site creates a unique ID and creates an entry in backend database for ID

Cookies: keeping "state" (cont.)



Cookies (continued)

What cookies can bring:

- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

aside

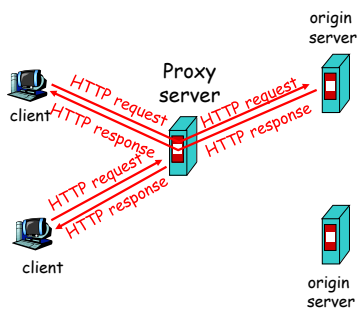
Cookies and privacy:

- cookies permit sites to learn a lot about you
- you may supply name and e-mail to sites
- search engines use redirection & cookies to learn yet more
- advertising companies obtain info across sites

Web caches (web proxy server)

Goal: satisfy client request without involving origin server

- user sets browser: Web accesses via cache
- browser sends all HTTP requests to cache
  - object in cache: cache returns object
  - else cache requests object from origin server, then returns object to client



More about Web caching

- Cache acts as both client and server
- Typically cache is installed by ISP (university, company, residential ISP)

Why Web caching?

- Reduce response time for client request.
- Reduce traffic on an institution's access link.
- Internet dense with caches enables "poor" content providers to effectively deliver content (but so does P2P file sharing)

Caching example

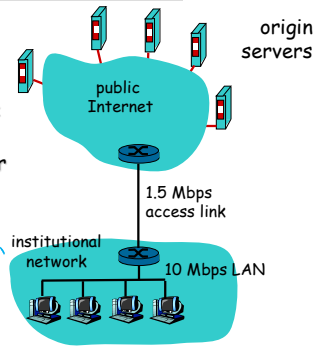
Assumptions

- average object size = 100,000 bits
- avg. request rate from institution's browsers to origin servers = 15/sec
- delay from institutional router to any origin server and back to router = 2 sec

Consequences

- utilization on LAN = 15%
- utilization on access link = 100%
- total delay = Internet delay + access delay + LAN delay

= 2 sec + minutes + milliseconds



Caching example (cont)

Possible solution

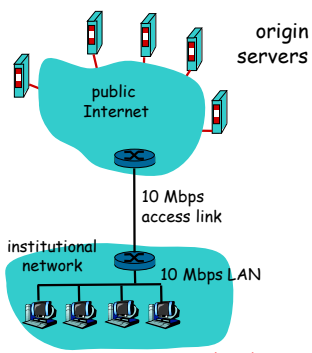
- increase bandwidth of access link to, say, 10 Mbps

Consequences

- utilization on LAN = 15%
- utilization on access link = 15%
- Total delay = Internet delay + access delay + LAN delay

= 2 sec + msecs + msecs

- often a costly upgrade



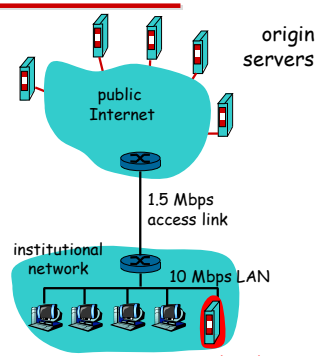
Caching example (cont)

Install cache

- suppose hit rate is .4

Consequence

- 40% requests will be satisfied almost immediately
- 60% requests satisfied by origin server
- utilization of access link reduced to 60%, resulting in negligible delays (say 10 msec)
- total avg delay = Internet delay + access delay + LAN delay = .6*(2.01) secs + .4*milliseconds < 1.4 secs

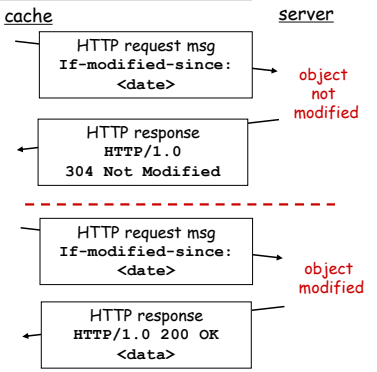


Conditional GET

- Goal: don't send object if cache has up-to-date cached version
- cache: specify date of cached copy in HTTP request

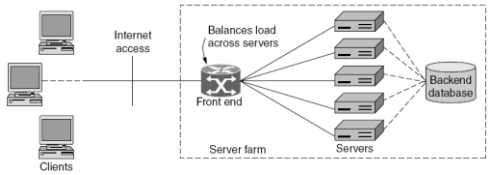
  `If-modified-since: <date>`
- server: response contains no object if cached copy is up-to-date:

  `HTTP/1.0 304 Not Modified`



Server Farm

- Issue: No matter how much bandwidth one machine has, it can only serve so many Web requests before the load is too great. Solution: Server farm



1) Each server must have a copy of the Web site;
2) The servers are connected to a common back-end database;
3) How to make up the server farm look like a single logical Web site to client?

To make the servers appear to be one Web site

- DNS solution
  - When a DNS request is made for the web URL, the DNS server returns a rotating list of the IP addresses of the servers;
  - Each client tries one IP address, typically the first on the list.
- Front end solution
  - To broadcast all of the incoming requests, each server answers them by agreement
    - The front end is maybe a switch or a router (maybe a NAT).
  - The front end maps them to a server
    - The front end need inspect the IP,TCP, or HTTP headers of packers, and notice the servers' state (maybe a host)
    - Load balance: round-robin

Content Delivery Networks

- Basic idea: Install a bunch of servers across the internet and simply replicate web pages on those servers. Be sure to redirect clients to the nearest replica server.

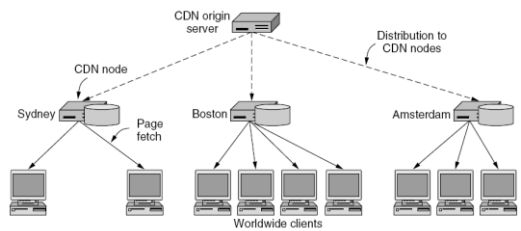
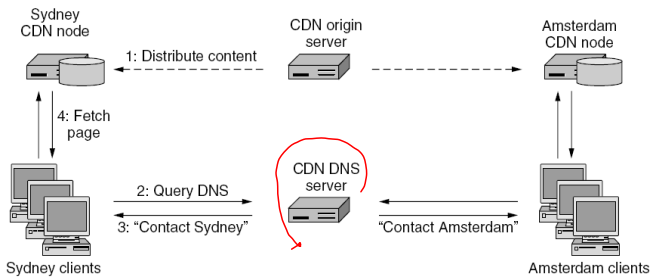


Figure 7-67. CDN distribution tree.

- CDN VS WebProxy = Push vs Pull, public vs private,...

Akamai: Using DNS to redirect…



- CDN DNS server looks at the IP address of the client making the request and returns the IP address of the CDN node that is nearest the client. (nearest=f(distance,load))

Summary

Our study of network apps now complete!

- Application architectures
  - client-server
  - P2P
  - hybrid
- application service requirements:
  - reliability, bandwidth, delay
- Internet transport service model
  - connection-oriented, reliable: TCP
  - unreliable, datagrams: UDP
- specific protocols:
  - HTTP
  - DNS
- socket programming

Summary

Most importantly: learned about *protocols*

- typical request/reply message exchange:
  - client requests info or service
  - server responds with data, status code
- message formats:
  - headers: fields giving info about data
  - data: info being communicated

- **control vs. data msgs**
  - **in-band, out-of-band**
- **centralized vs. decentralized**
- **stateless vs. stateful**
- **reliable vs. unreliable msg transfer**
- **"complexity at network edge"**