Lecture

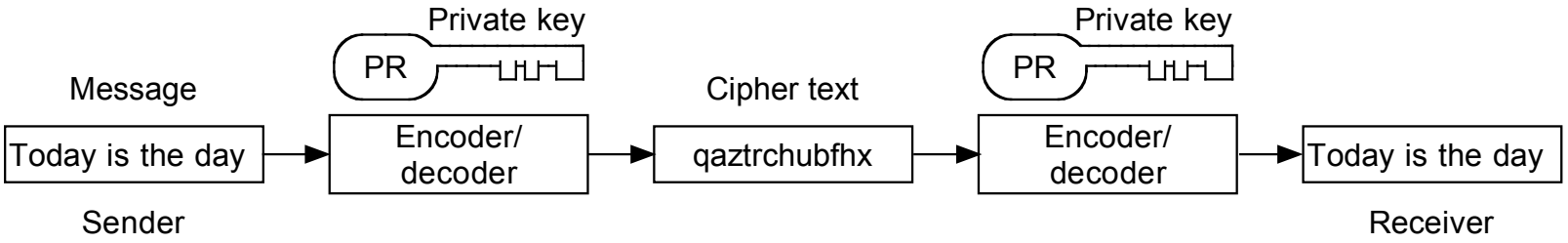
Secure Web Communication
and
Web Server Performance

3 Diverse Topics for Today

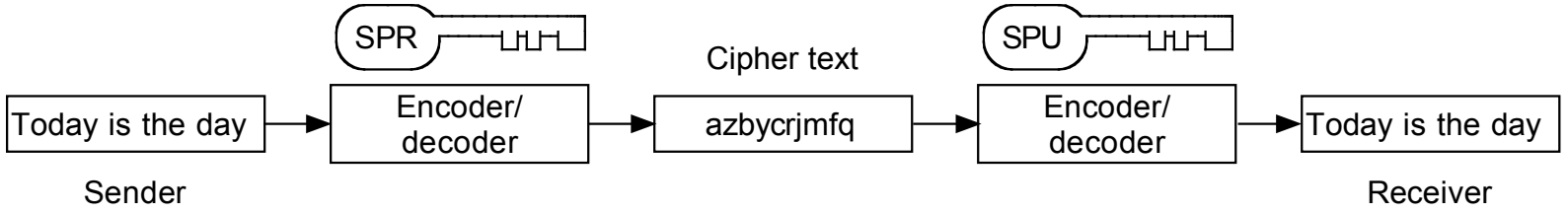
1. **Secure web communication**
   - Public Key Cryptography
   - Public and private key encryption
   - Digital Certificates and Certifying Authorities
   - Secure Sockets Layer Protocol (SSL) and https
2. **Web Server Performance**
   - Popular platforms
   - Web Server Farms
   - Load Balancing
     - Switches
     - DNS redirection
3. **Web Server as Proxy Server**
   - Caching
   - Using Apache as a proxy server

Public vs. Private key Cryptography

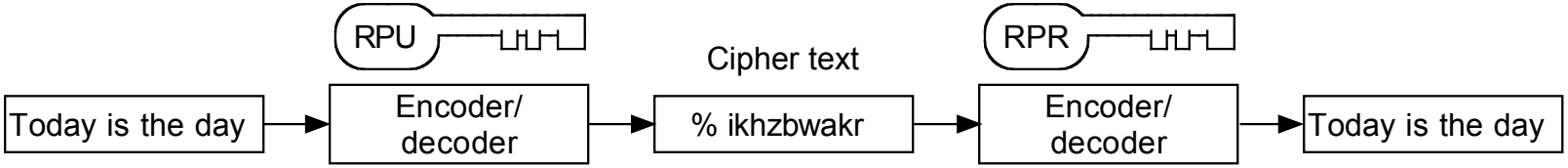
Private Key Encryption: sender/receiver share private key



Public Key Encryption: for authentication



Receiver has private and public keys: for privacy



RPR = receiver private key
RPU = receiver public key

SPR = sender private key
SPU = sender public key

RSA Public Key Encryption

- **RSA** is one of the first practical public-key cryptosystems and is widely used for secure data transmission
- A user of RSA creates two large prime numbers and then publishes one of them as his public key
  - the numbers are typically 1024 digits or more
  - there are a few more steps involved
- The most popular algorithm for public key encryption is the RSA algorithm (***R***ivest, ***S***hamir ***A***dleman)
- Determining the private key from the public key involves factoring very large numbers
  - ***but*** there is no efficient algorithm for factoring large numbers
- Though current wisdom is that 1024-bit keys are unbreakable Certificate Authorities use 2048-bit keys

Cryptographic Hash Functions

- A *hash function* or *hash algorithm* is a function that maps a domain of values into a range of numbers.
- Given a data item X (X could be a word or a file), H is called a *cryptographic* hash function if it is computationally infeasible to find another data item Y, not equal to X, such that the hash value H(X) is equal to the hash value H(Y).
  - H(X) is called the ***message digest*** or ***digital signature*** of X under the hashing algorithm H.
- Two well known cryptographic hash functions are MD5 and SHA

Bulk Cipher Methods

- public/private key encryption methods are not suitable for general purposes, e.g.
  - the RSA method can only encrypt blocks of data which are 11 bytes less than the key size; and each decryption involves complex mathematical calculations
- therefore, secure communication on the web uses a combination of public key encryption and conventional one-way ciphers
- a ***bulk cipher*** is one in which the same keys are used to encrypt and decrypt the data; they are fast and can encrypt files of any size
- some sample bulk ciphers: RC2, RC4-40, RC4-56, DES40-CBC

Making Sure a Message Has Not Been Altered

- A ***message digest*** is the number produced by applying a cryptographic hash function to a message
  - The message digest is included along with the message; here are the steps that are followed:

1. Sender produces a message digest using a known hashing algorithm
2. Message digest is encrypted and sent with the message
3. Receiver decrypts the digest and then computes the message digest from the actual message to make sure they are identical

- For greater security, the message can also be encrypted
- Systems combining public key cryptography and message digests are called ***digital signatures***

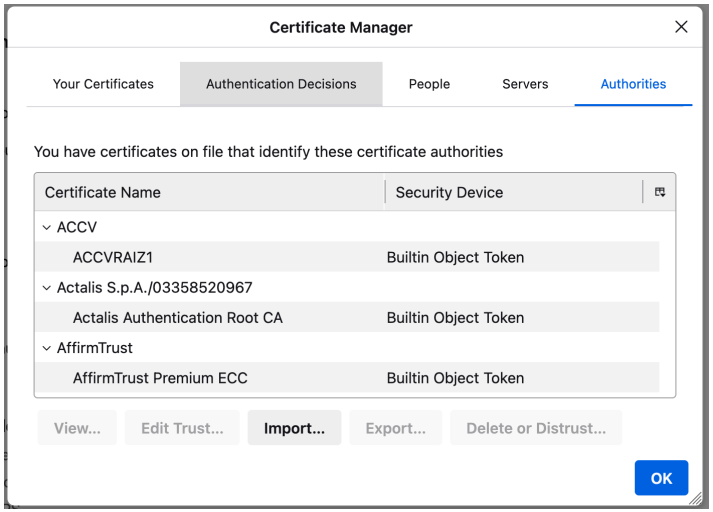
Certificate Authority

- Encryption of data does not solve the entire problem; **how do we guarantee that the organizations that we are dealing with are legitimate?**
- A certificate authority (CA) is an organization that both parties involved in a secure communication, trust
- the role of the CA is to verify the identity of an entity (client/server)
- once the CA verifies the entity, it issues a digitally signed electronic certificate
  - it is signed with the CA's private key
- Web browsers are usually pre-configured with a list of CAs that are trusted
  - In Firefox Settings…, Privacy & Security, Security, Certificates, View Certificates...
  - In Chrome select Settings…, Privacy and Security, Security, Advanced, Certificates Managed by Chrome, Manage certificates, root certificates trusted by Chrome …

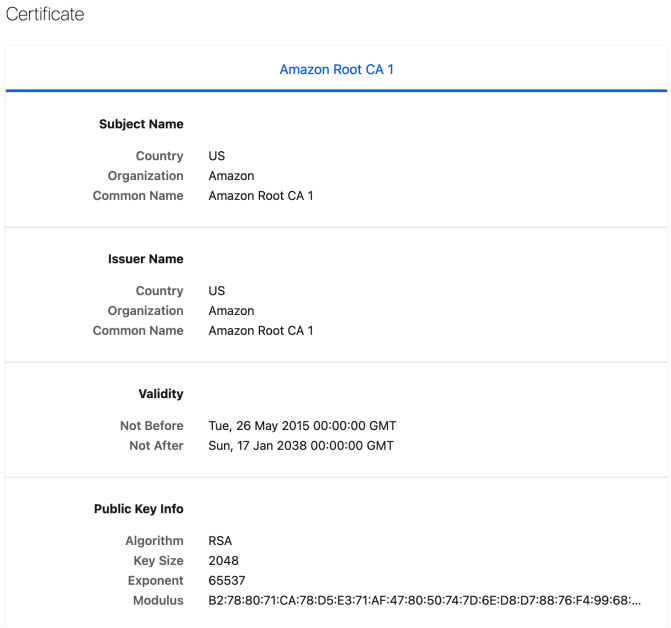
Trusted Root CAs (Firefox)

- A *certificate authority* dispenses public and private keys

authorities, including Comodo, Verisign →



Actual certificate →



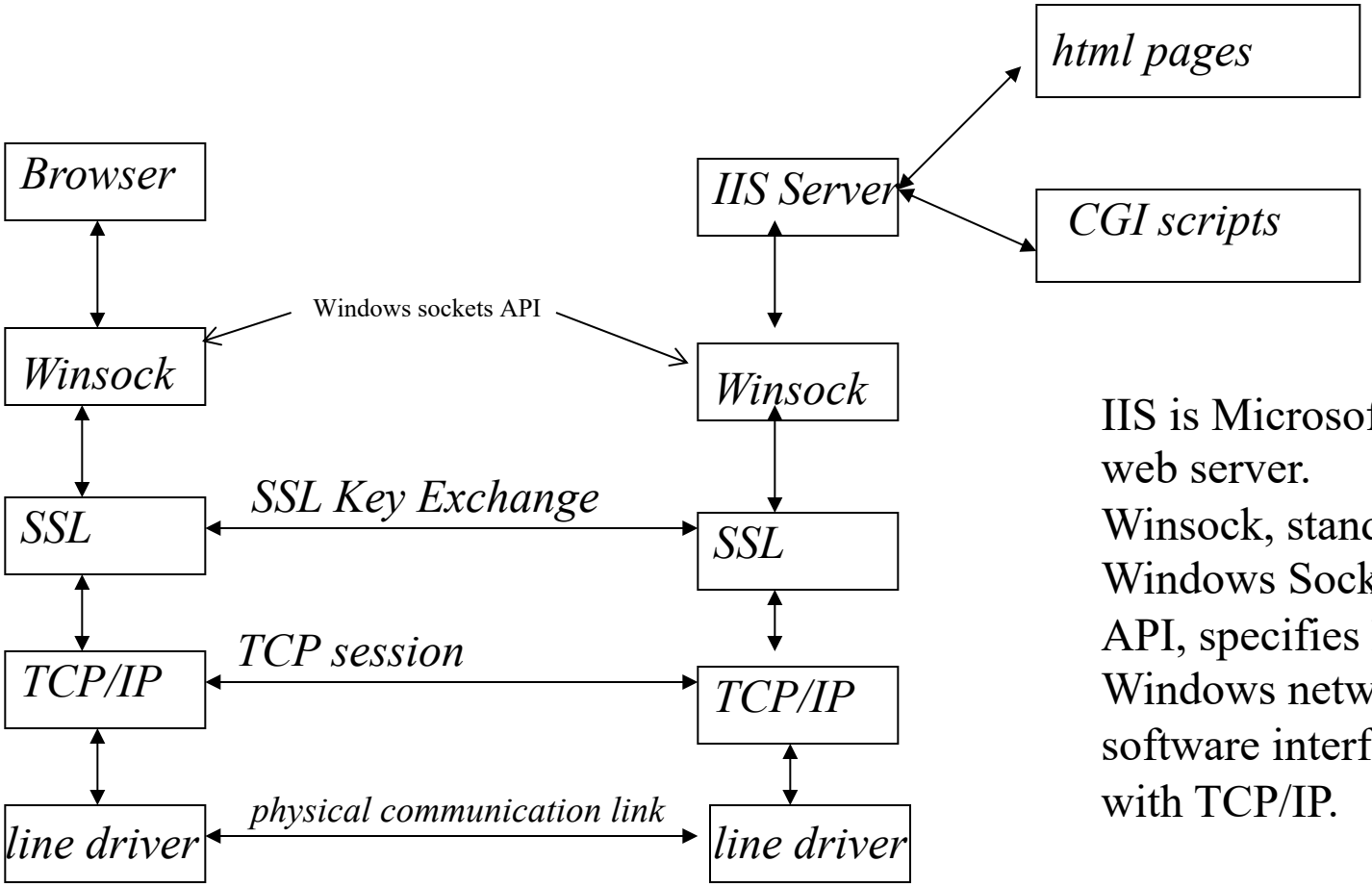
Introduction to SSL

- Given public key encryption, cryptographic hashing, message digests and digital certificates, how are these put together to produce secure electronic commerce?
- The answer: SSL, is a protocol for establishing an encrypted link between server and client, that uses **authentication** and **encryption** of transactional data
- Netscape was the original designer of SSL
- SSL is occasionally called TLS, Transport Layer Security protocol
- SSL is transparent to users, except for the http**s** (rather than http) that appears
- the SSL protocol fits between the TCP layer and the HTTP layer
  - therefore, SSL can be used to encrypt other application-level protocols such as FTP and NNTP
- SSL is supported by all major browsers

Introduction to SSL

- Secure Sockets Layer (SSL) provides end-to-end security between client and server
- ***authentication*** of both parties is done using digital certificates
- ***privacy*** is maintained using encryption
- ***message integrity*** is accomplished using message digests
- SSL for HTTP is referred to as HTTPS and operates on **port 443**

SSL Graphical View



IIS is Microsoft's web server.
Winsock, stands for Windows Sockets API, specifies how Windows network software interfaces with TCP/IP.

How SSL Works – Detailed Steps

1. Client contacts a secure server using https and asks that the server identify itself
2. Server sends client its digital certificate including the server's public key, thereby verifying that the server is who he says he is
3. Client checks the certificate root against its list of trusted CAs and that the certificate is unexpired; if all is OK, client generates a session key, encrypts it with server's public key and sends the cipher back to the server
4. Server decrypts the symmetric session key using its private key and sends back an acknowledgement, and a secure channel is established
5. Client and server now exchange data that is always encrypted
6. Before each data transfer, a message digest calculated from the data is sent on the secure channel
7. Receiving party uses the message digest to guarantee the data has not been modified

digicert.com

- A vendor that offers Extended validation (EV) certificates
- Validated companies are guaranteed to use encryption for all password/credit card transactions
- digicert activates a green address bar whenever an SSL session is established with a merchant's or issuing bank's EV-validated site
- Participating companies include Facebook, GitHub, etc.

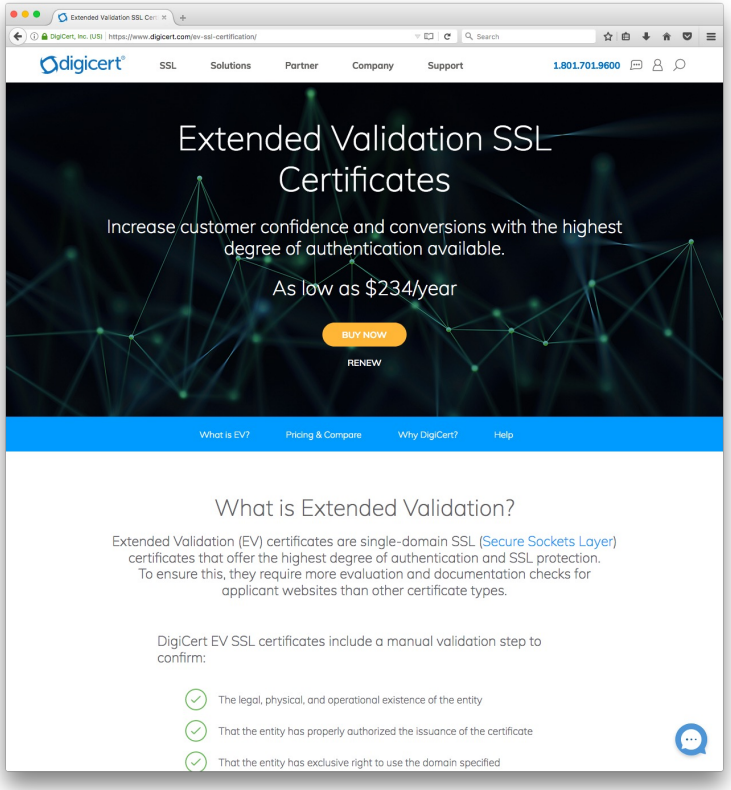
https://www.digicert.com/tls-ssl/basic-tls-ssl-certificates

- You can buy EV certificates here:

https://comodosslstore.com/ssl-types/ev

- You can buy cheap certificates here:

https://www.positivessl.com/

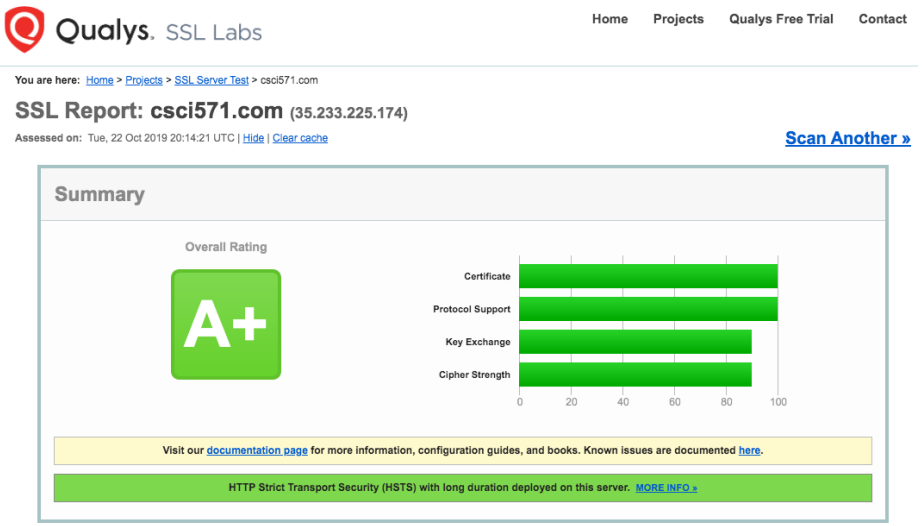


References

- SSL Protocol Version 3.0 Specification:
  - http://www.mozilla.org/projects/security/pki/nss/ssl/draft302.txt
- TLS Protocol Version 1.0 Specification:
  - http://www.ietf.org/rfc/rfc2246.txt
- Extensions to TLS:
  - Kerberos: http://www.ietf.org/rfc/rfc2712.txt
  - TLS with HTTP/1.1: http://www.ietf.org/rfc/rfc2817.txt
  - HTTP over TLS: http://www.ietf.org/rfc/rfc2818.txt
  - AES Ciphersuites for TLS: http://www.ietf.org/rfc/rfc3268.txt

Secure csci571.com

- **https:csci571.com** is secured by PositiveSSL certificate: https://comodosslstore.com/positivessl.aspx
- Much cheaper than direct from PositiveSSL
- Apache 2.4, servicing csci571.com obtained the **highest overall rating, A+**, from SSL Labs **https://www.ssllabs.com/ssltest/**
  - Turning off **compromised SSL protocols** (all except TLS 1.2)
  - Enabling HTTP Strict Transport Security (**HSTS**)
  - Prioritizing **ECDHE** (Elliptic Curve Diffie-Hellman Ephemeral) cipher suite.



Analyzing Web Server Performance

Selecting a Web Platform

- Capacity – what capacity is needed from the server, databases, applications
- Cost/investment – what are the initial costs and the continuing costs?
- Maintenance – who will perform it; how complex
- Security – a strategy is needed
- Development support – are there staff to support application development

Popular Platforms

- Microsoft
  - Windows Server 2012 or 2016, Microsoft Nano Server
  - Internet Information Services (IIS) version 8 or 10
  - MS SQL Server
  - Active Server Pages or ASP.NET applications
  - Develop with VB, COM, C++, C#, or HTML/CSS/JS
  - PHP Manager for IIS 7
  - Available on Cloud
- Linux
  - Ubuntu / Red Hat / SuSe or any other distribution
  - Apache / Nginx web servers
  - Oracle mySQL
  - PHP, Python, NodeJS
  - Java servlets (Tomcat)
  - Available on Cloud
- UNIX
  - Oracle Solaris
  - Oracle WebLogic
  - IBM WebSphere

Estimating Server Performance Requirements

- **What is the time required to deliver a request to the server?**
  - is client authenticated or anonymous, the former is much more costly
  - how much data does the client send
  - the bandwidth of the slowest link between client and server can determine the time
- **What is the time required to obtain the result?**
  - is the result a static HTML page
    - is it in the cache, how big is the file
  - is the result a dynamic page: is the application that creates it a CGI script, an .NET program, etc.
- **What is the time to deliver the result?**
  - the amount of data sent by the server
  - bandwidth of the client-server link
  - number of simultaneous requests

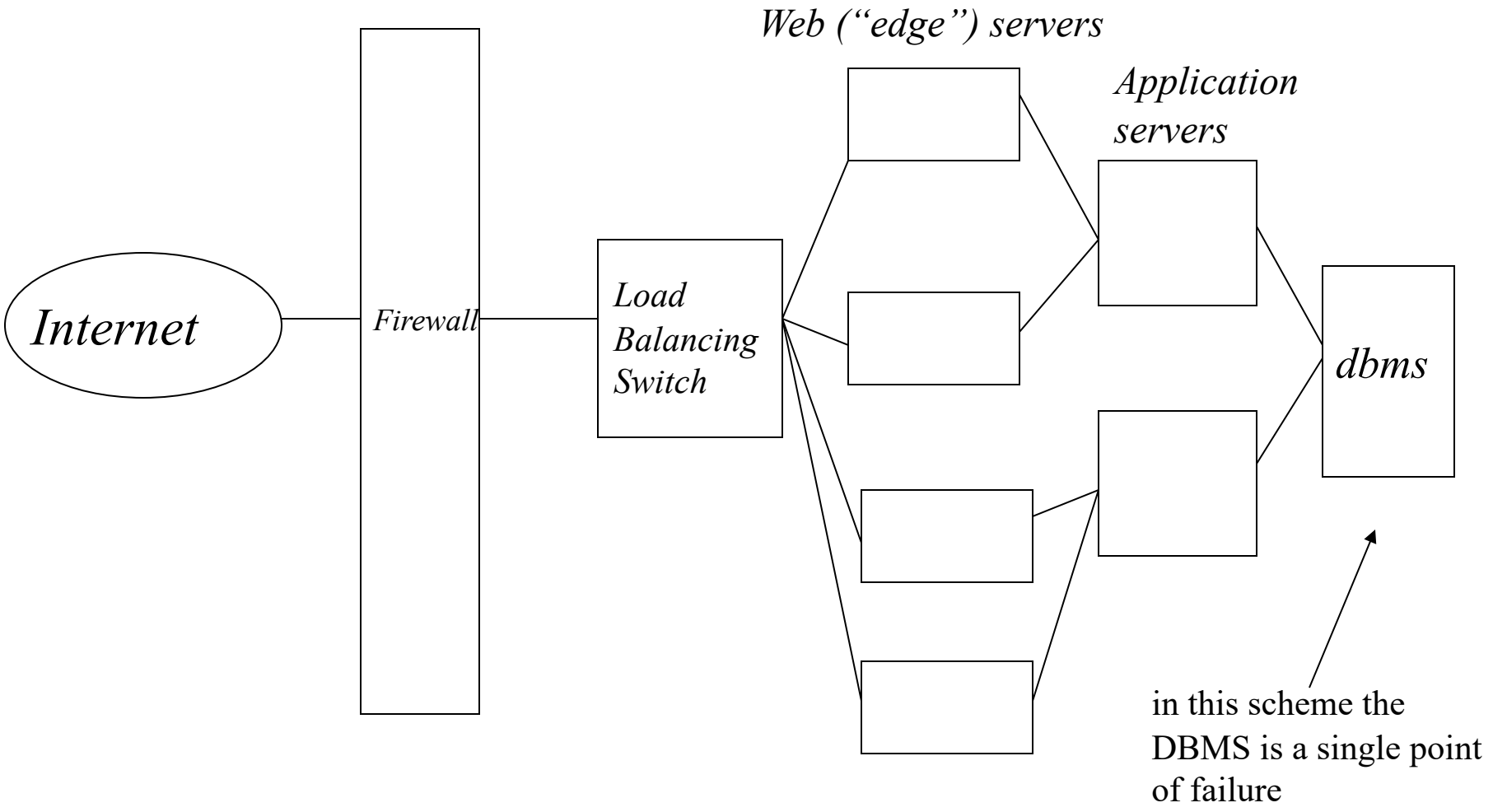
Estimating Server Performance Requirements

- to estimate what performance is needed, estimate
  - the number of clients that will connect each second, *connections per sec*
  - the number of bytes the client will send to the server during each transaction
  - the number of bytes the server will send to the client during each transaction, *bytes transferred*
  - how much of the link the web server is allowed to use
- multiply the number of clients connecting each second by the total number of bytes transferred during each connection and divide the result by the percentage of the link the web server is allowed to use
- E.g., 120 clients connect twice each minute implies 240 connections per minute or 4 per second. If each client sends 1Kbyte of data and receives 15Kbytes back, then the server needs to support 64Kbytes per second or 512 Kbitsps or roughly .5megabits per second.
- **General rule:** the link should have at least twice the bandwidth as the average above

Web Server Farms

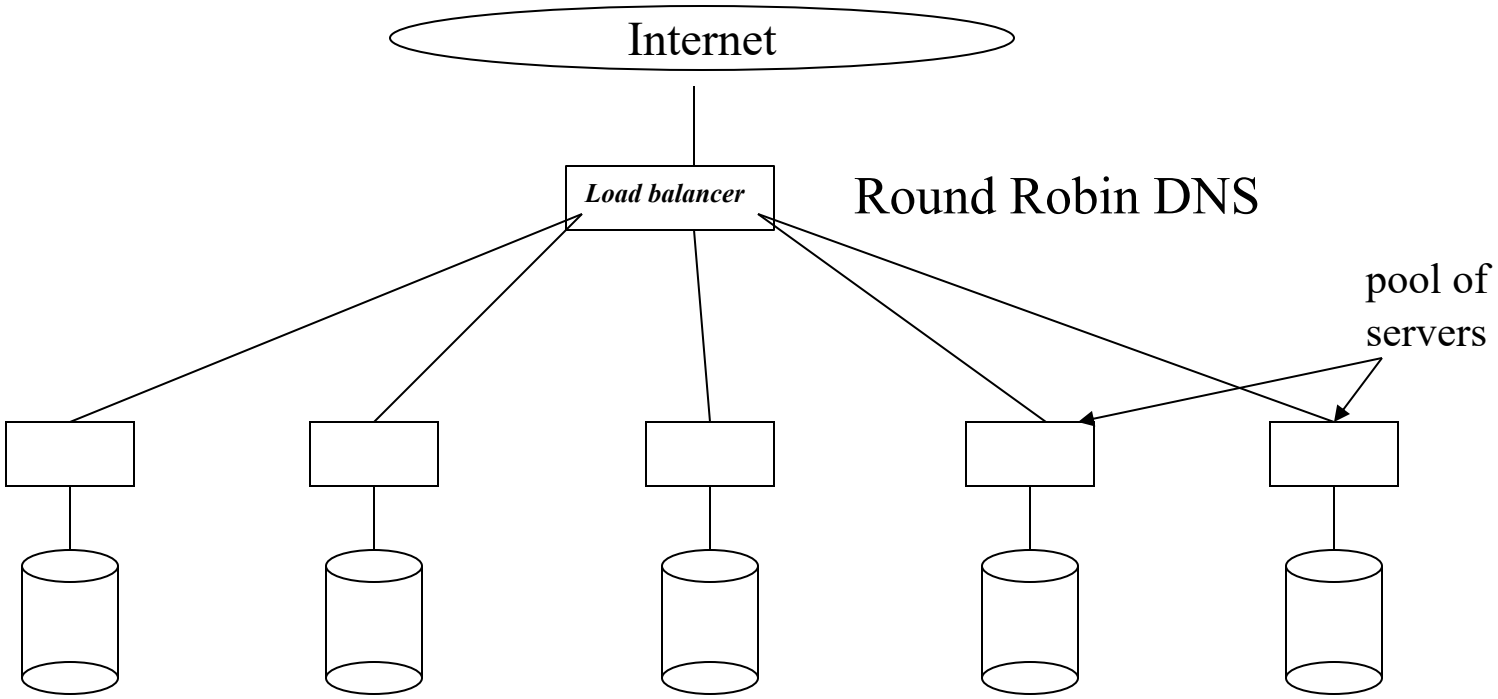
- A web server farm consists of **multiple server** machines and **load balancing** hardware that distributes web requests across the servers
  - this provides incremental scalability and high availability at a low cost.
  - For example, a farm of four independent servers, each with 95 percent availability has an overall availability of 99.999 percent.
- Generally, Web servers have a relatively small amount of static data, and an application server (behind the firewall) provides dynamic content from information stored in a database or exiting applications
- However, if storage is shared across all servers, then there is a single point of failure
  - mirroring is one possible solution using a DBMS feature to push new data to all servers

One Possible Web Farm Architecture



Why Load Balancing is Needed

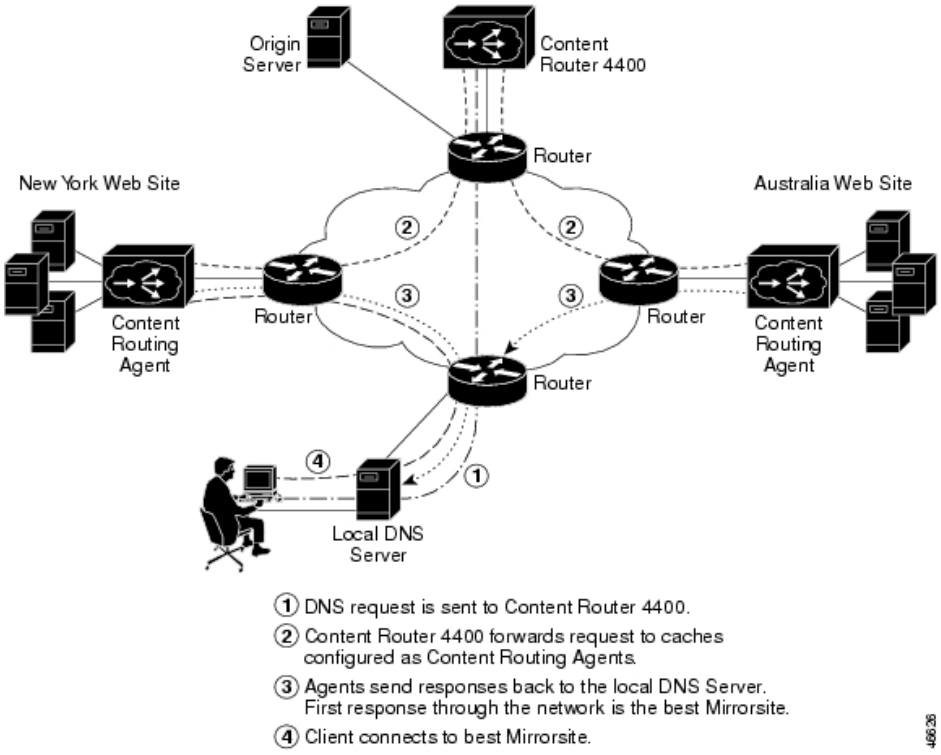
- In the scheme below
  - a pool of server machines appear as a single host and client requests are routed alternately to each one
  - However, in the event of a failure, the load balancer can detect the failure and route the request elsewhere
  - Persistent data must be fully replicated; all nodes are identical



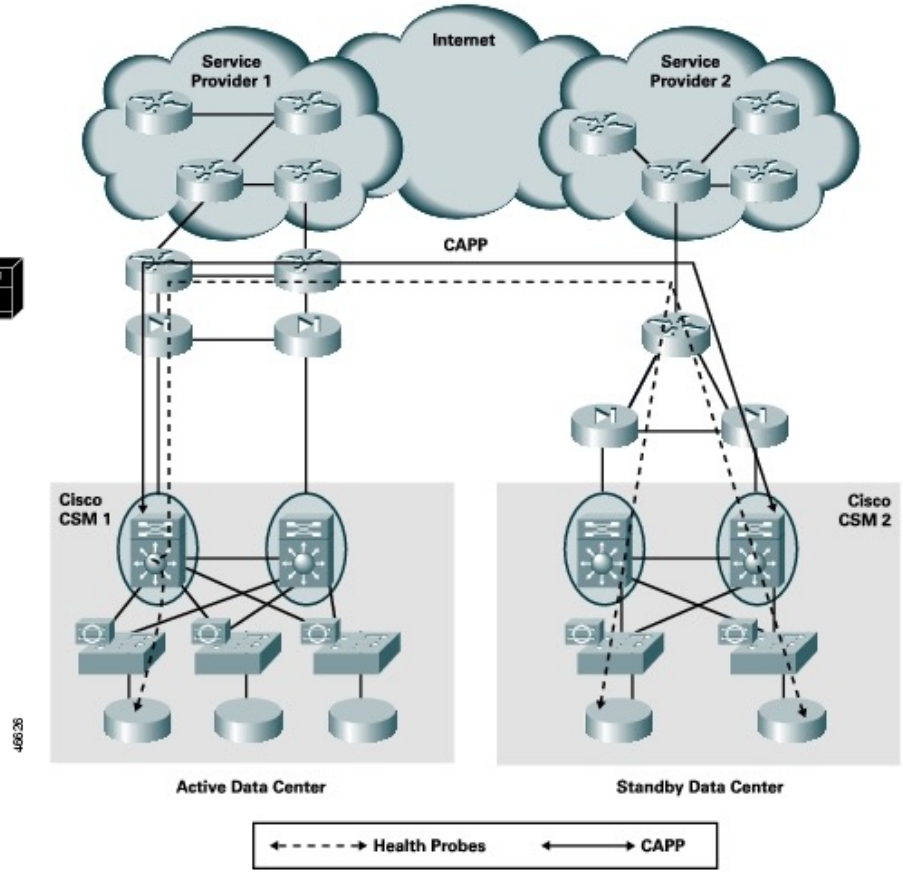
Example Load Balancing Switch

- Load balancing hardware exists to prevent requests going to servers that have failed; e.g.
  - Cisco Content Switching Module, Cisco Catalyst 6500; RADware appDirector
- continually monitors servers for application availability and database connectivity
- routes traffic only to available applications and ensures that traffic is not directed to a failed server or application.
- can perform an HTTP redirect to a different location upon failure of a real or virtual IP address.
- Supports Cookie, HTTP redirect, and Secure Sockets Layer (SSL) session ID persistence features guaranteeing that a specific client gets the correct content, regardless of Internet mega-proxies and shopping-cart application design.

Two Scenarios for Using Cisco Content Router Load Balancing Switch



Supporting multiple sites around the world



Supporting standby data center

Another Approach to Load Balancing: DNS Redirection

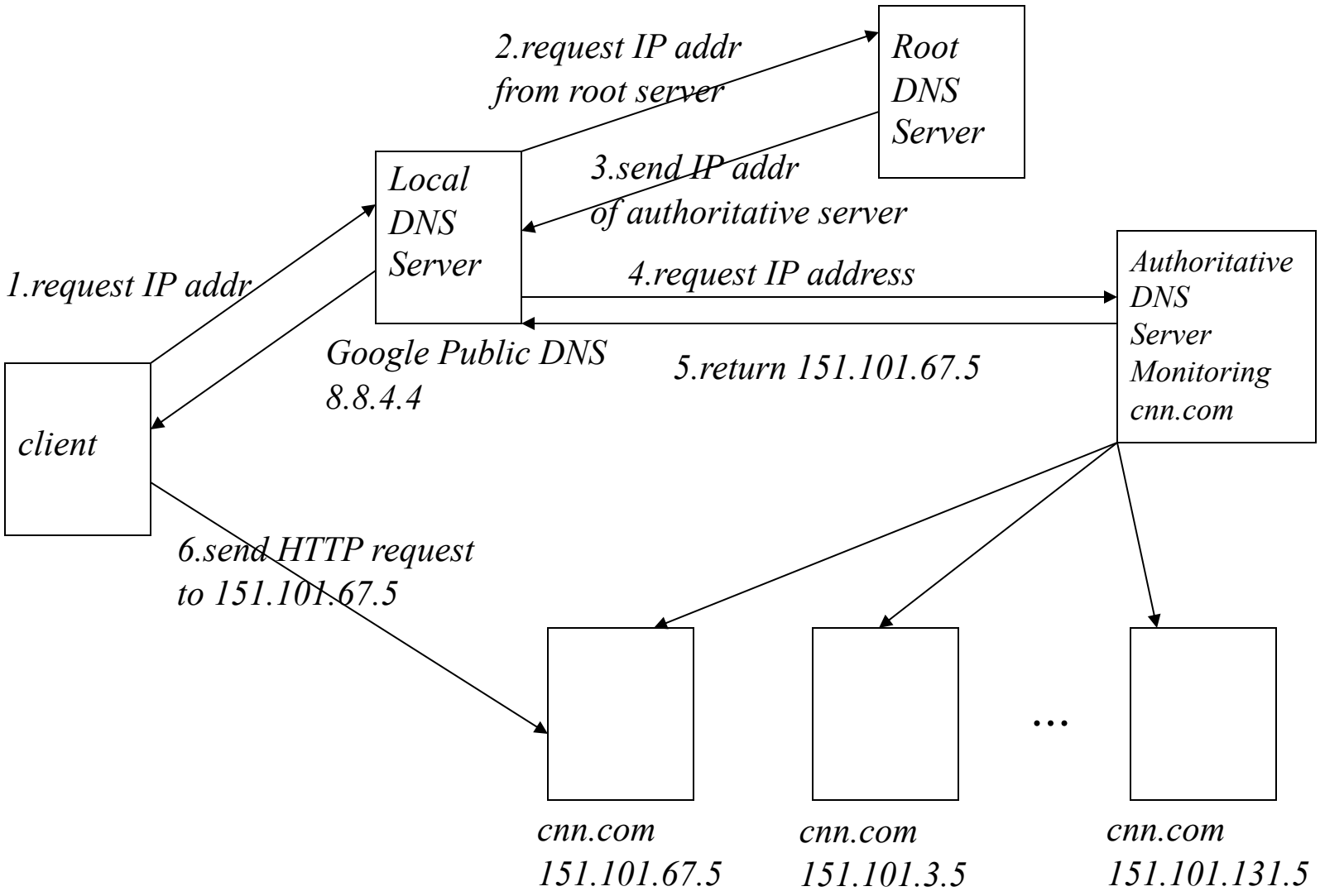
- A browser begins by resolving a domain name into an IP address
- A DNS resolver may be on the browser machine, on the client network, or a remote DNS server
- DNS resolvers can be configured to return one of a set of IP addresses, e.g.

```
$ nslookup cnn.com
Server: 10.202.0.153
Address: 10.202.0.153#53

Non-authoritative answer:
Name: cnn.com
Address: 151.101.67.5
Name: cnn.com
Address: 151.101.3.5
Name: cnn.com
Address: 151.101.195.5
Name: cnn.com
Address: 151.101.131.5
```

(note: the last two sets of digits change over time)

DNS Redirection



DNS Redirection

- This form of load balancing has problems because
  - web browsers will cache the IP address for a given domain
  - some operating systems cache IP addresses for given domains
- However, some DNS servers use algorithms other than round robin, e.g.
  - load-balancing: they check the load on many web servers and send the request to the least loaded
  - proximity-routing: they send the request to the nearest server, when the servers are geographically distributed
  - fault-masking: check for down web servers and avoid them

Web Server Performance Testing

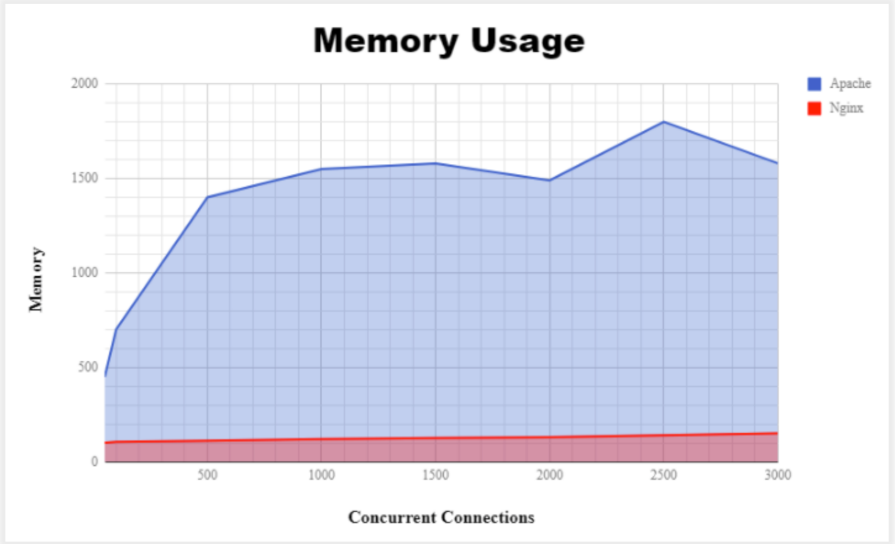
Web Server Benchmarking

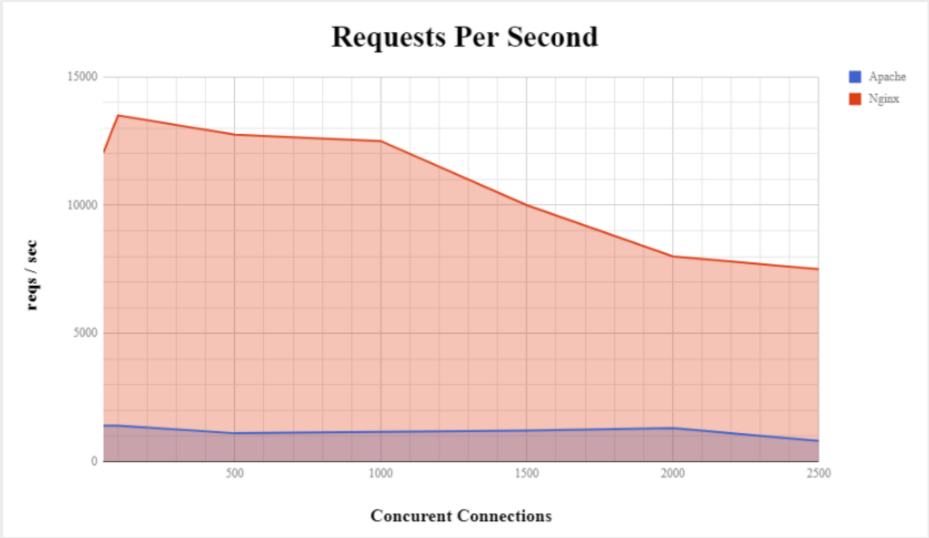
- **Web server benchmarking** is the process of estimating a web server performance in order to find if the server can serve sufficiently high workload.
- The performance is usually measured in terms of:
  - ***Number of requests*** that can be served per second (depending on the type of request, etc.);
  - ***Latency response time*** in milliseconds for each new connection or request;
  - ***Throughput*** in bytes per second (depending on file size, cached or not cached content, available network bandwidth, etc.).
- The measurements must be performed under a varying load of clients and requests per client.

Other Web Servers: Lighttpd, nginx

- **lighttpd** (pronounced "lighty") [www.lighttpd.net] is an open-source web server optimized for speed-critical environments
  - The low memory footprint, small CPU load and speed optimizations make lighttpd suitable for servers that are suffering load problems, or for serving static media separately from dynamic content.
  - lighttpd is free software/open source, and is distributed under the BSD license
  - Lighttpd is used by a number of high-traffic websites
  - See: https://www.lighttpd.net/
- **Nginx** (pronounced "Engine-X") [nginx.org] is an open-source Web server and a reverse proxy server for HTTP, SMTP, POP3 and IMAP protocols, with a strong focus on high concurrency, performance and low memory usage. It is licensed under a BSD-like license
  - Nginx uses an asynchronous event-driven approach to handling requests, instead of the Apache HTTP Server model that defaults to a threaded or process-oriented approach. Nginx's event-driven approach can provide more predictable performance under high loads
  - According to Netcraft's August 2021 Web Server Survey, Nginx was found to be the **first most widely used** web server across all domains

Comparisons of Nginx, Apache Memory Usage and Requests Per Second

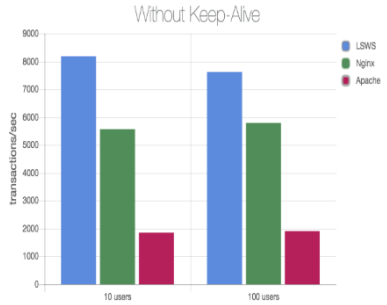


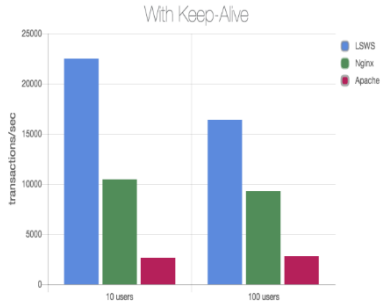


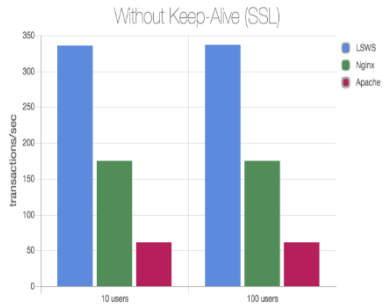
https://help.dreamhost.com/hc/en-us/articles/215945987-Web-server-performance-comparison

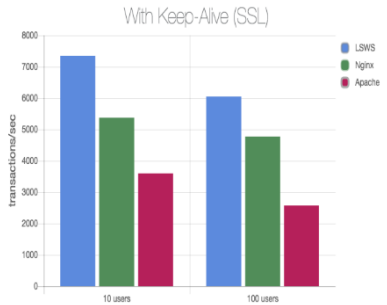
LSWS vs. nginx vs. Apache

Small Static File Benchmark

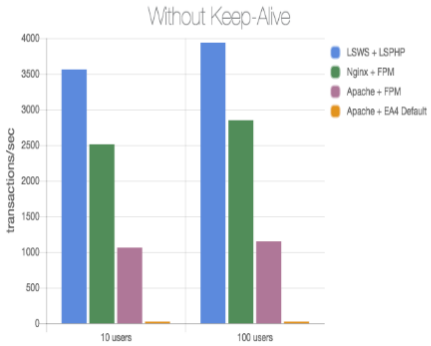


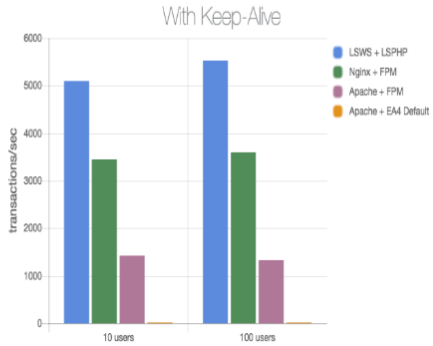






PHP Hello World Benchmark





http://www.litespeedtech.com/benchmarks/small-static-file
(LSWS = LiteSpeed Web Server)

Apache Benchmarking (not required)

- ab is a tool for benchmarking your Apache Hypertext Transfer Protocol (HTTP) server.
- It is included in your apache installation; in the bin directory
- Among other things, it shows how many requests per second your Apache installation is capable of serving.
- It is a command line program; its description can be found at

  http://httpd.apache.org/docs/2.2/programs/ab.html

Exercise to run ab (not required)

1. Note down the server load using the uptime command

```
cs-server.usc.edu(16): uptime
9:21am  up 609 day(s), 21:29,  2 users,  load average: 28.68, 30.12, 29.96
```

2. Create a static (small) html page as follows, snkpage.html, and place it in the document root

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html> <head> <title>Webserver test</title> </head>
<body> This is a webserver test page. </body> </html>
```

3. Connect to the bin directory and run ab

```
cs-server.usc.edu(1):  ./ab -n 1000 -c 5 http://cs-server.usc.edu:7890/snkpage.html
```

This sends 1000 requests to the server with currency (c = 5) number of multiple requests at a time

4. To save the output for 50,000 requests in a comma separated file

```
$ ab -k -n 50000 -c   2   -e   apache2r1.cvs   http://cs-server.usc.edu:7890/snkpage.html
```

Sample Output of ab (not required)

```
This is ApacheBench, Version 2.3
    <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus
    Technology Ltd,
    http://www.zeustech.net/
Licensed to The Apache Software
    Foundation,
    http://www.apache.org/

Benchmarking cs-server.usc.edu (be
    patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
Completed 1000 requests
Finished 1000 requests


Server Software:
    Apache/2.2.22
Server Hostname:       cs-
    server.usc.edu
Server Port:                7890

Document Path:
    /snkpage.html
Document Length:        168 bytes
```

Concurrency Level: 5
Time taken for tests: 0.746 seconds
Complete requests: 1000
Failed requests: 0
Write errors: 0
Total transferred: 430000 bytes
HTML transferred: 168000 bytes
Requests per second: 1341.04 [#/sec] (mean)
Time per request: 3.728 [ms] (mean)
Time per request: 0.746 [ms] (mean, across all concurrent requests)
Transfer rate: 563.13 [Kbytes/sec] received

Connection Times (ms)

	min	mean[+/-sd]	median	max
Connect:	0	0 0.6	0	10
Processing:	1	3 1.9	3	26
Waiting:	1	3 1.9	3	25
Total:	2	4 2.0	3	26

Percentage of the requests served within a certain time (ms)

50%	3
66%	3
75%	4
80%	4
90%	4
95%	5
98%	7
99%	12
100%	26 (longest request)

Improving Apache Web Server Performance (1)

- **Additional RAM**, faster CPU always helps!
- Load **only the required modules**; this reduces the memory footprint
- The **HostnameLookups** directive enables DNS lookup so that hostnames are logged instead of the IP address, which adds latency to every request;
  - If hostnames are desired, use the Apache utility logresolve (which ships with Apache) to resolve IP addresses of an entire log file before analyzing the log file; e.g.
  - logresolve < access.log > access.log.resolved
- If **AllowOverride** is not set to None, then Apache will attempt to open .htaccess file in each directory that it visits; enable .htaccess for only those directories that contain it
- Do not set **MaxClients** too low as that will cause requests to be queued and possibly lost; do not set MaxClients too high as that will cause the server to start swapping and response time will degrade;
  - Let MaxClients = (Total RAM dedicated to Web Server)/Max child process size
- **Tune MinSpareServers and MaxSpareServers** so that Apache need not spawn more than 4 child processes per second; when this happens an error is placed in the error_log file
- HTTP compression can be enabled using mod_gzip or mod_deflate; the payload is compressed only if the browser requests it

Improving Apache Web Server Performance (2)

- **mod_fastcgi** is a general FastCGI module, which uses FastCGI rather than normal CGI to connect to the CGI scripts.
  - With normal CGI, the webserver communicates with the CGI script through environment variables, and the client browser with the CGI script through its standard input.
  - With FastCGI, each script acts as a daemon, being started once and handling multiple requests. Instead of environment variables, the server passes all information about the request through standard input, allowing FastCGI scripts to even be executed on different servers, over extra TCP connections.
  - See www.fastcgi.com
- Use "**direct modules**": mod_perl, **mod_php5**, mod_python, etc
- For several scripting languages (including Perl, Python, PHP, Tcl and Ruby) there are separate interpreter modules that give the language more control over Apache, as well as a performance boost.
- In general, this is done by using specific knowledge about the language, and by keeping the language's Interpreter or Virtual Machine hanging around, passing it scripts as they get invoked. This avoids the execution overhead, and in some languages the compiling phase.

Improving Apache Web Server Performance (3)

- Use two versions of Apache, one "tiny version" to serve static content and forward requests for dynamic content to the "larger version"; request forwarding is done by using mod_proxy and rewrite_module. An example follows.
- Suppose there is a lightweight Apache server listening to port 80 and the heavyweight Apache server listening on port 8088. Then the following configuration in the lightweight Apache can be used to forward all request except request for images to the heavyweight server.

```
ProxyPassReverse / http://%{HTTP_HOST}:8088/
RewriteEngine on
RewriteCond %{REQUEST_URI} !.*\.(gif|png|jpg)$
RewriteRule ^/(.*) http://%{HTTP_HOST}:8088/$1 [P]
```

- Alternatively use **Nginx as "reverse" proxy** (tier-1) and Apache as App Server (tier-2). (many cloud services provide this configuration by default)

Topic 3: Web Server as a Proxy Server

- Definition of ***Proxy Server***: An intermediary server that accepts requests from clients and either forwards them or services the request from its own cache.
- Implications
  - A proxy server is a web server
  - A proxy server acts as a server to the client that makes a request, and it acts as a client to the servers it connects to
  - A proxy server can monitor all documents provided by server responses
  - A proxy server can monitor all client requests to other servers
  - On **client-side**, a called a "**forward proxy**":
    - https://en.wikipedia.org/wiki/Proxy_server#Forward_proxy

HTTP Protocol and Proxy Servers

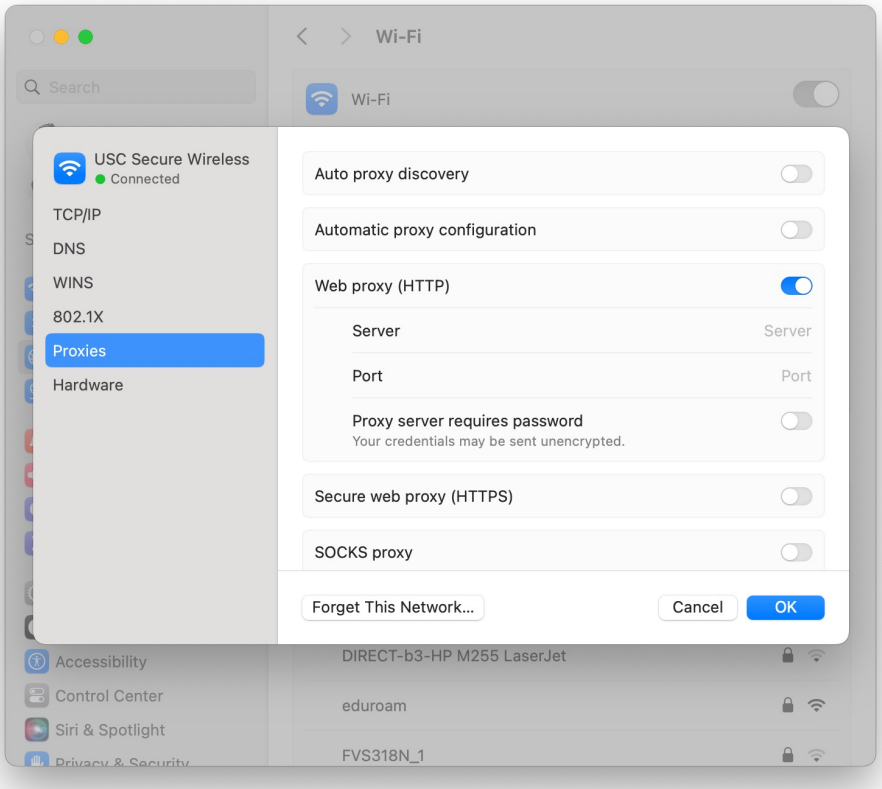
- The protocol between client and proxy server, or between proxy servers is HTTP, even if the request is for ftp, telnet, mailto, etc.
- e.g., an ftp request is sent as an HTTP request to a proxy server as:

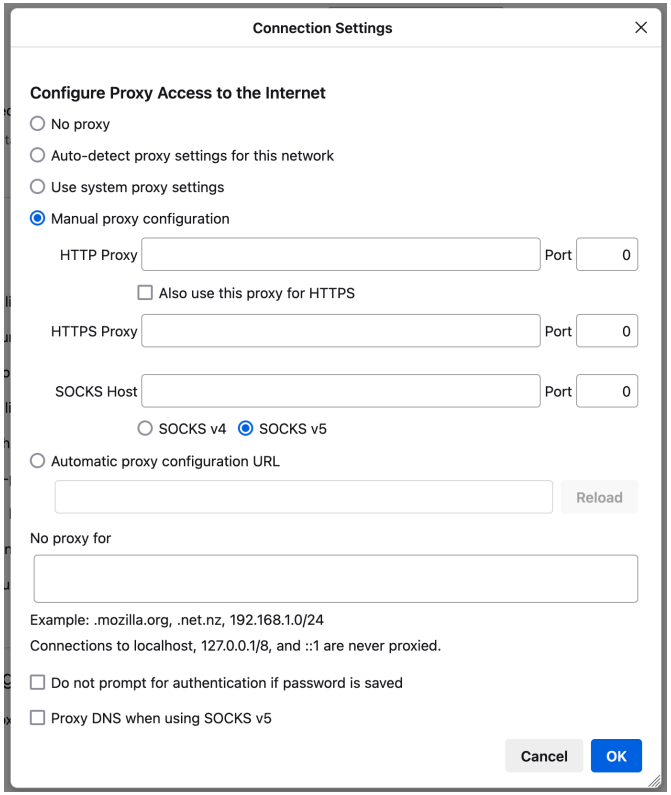
```
GET ftp://ftp.usc.edu/pub/softeng.txt HTTP/1.0
User-agent: Mozilla/4.0
Accept: */*
```

- only the last proxy server has to speak the ftp protocol to the ftp server, in this case ftp.usc.edu

Pointing to a "forward" Proxy in Your Browser

- To configure your browser to point to a proxy server
  - In Firefox click on Settings… | General | Network Settings | Use system proxy settings
  - In Chrome click Settings… |System | Open your computer proxy settings | Proxies (on macOS) | Web Proxy





Why Use a ("forward") Proxy Server

- Runs on client side
- to **prevent access** to restricted sites
  - e.g., Facebook and Twitter ("**blacklisting**")
- to **control access** to a restricted site
  - the proxy server can request name/password
- to enhance security by **controlling** which application-level **protocols** are permitted
- to improve performance by maintaining a **cache**
- to modify content before delivery to the client, e.g., reducing image sizes for display on TV sets.
- to act as an **anonymizer,** removing identifying information from HTTP messages (see proxy list at https://free-proxy-list.net/)

Example of Anonymizing Proxy

- Client issues:

```
GET /something/file.html HTTP/1.0
Date:Sun, 01 Oct 2000 23:25:17 GMT
User-agent: Mozilla/4.75 (Win98; U)
From: joe@hardware.com
Referer: http://www.irs.gov/tax-audits.html
Cookie: profile="football,lite beer"
Cookie: income-bracket="30K-45K"
```

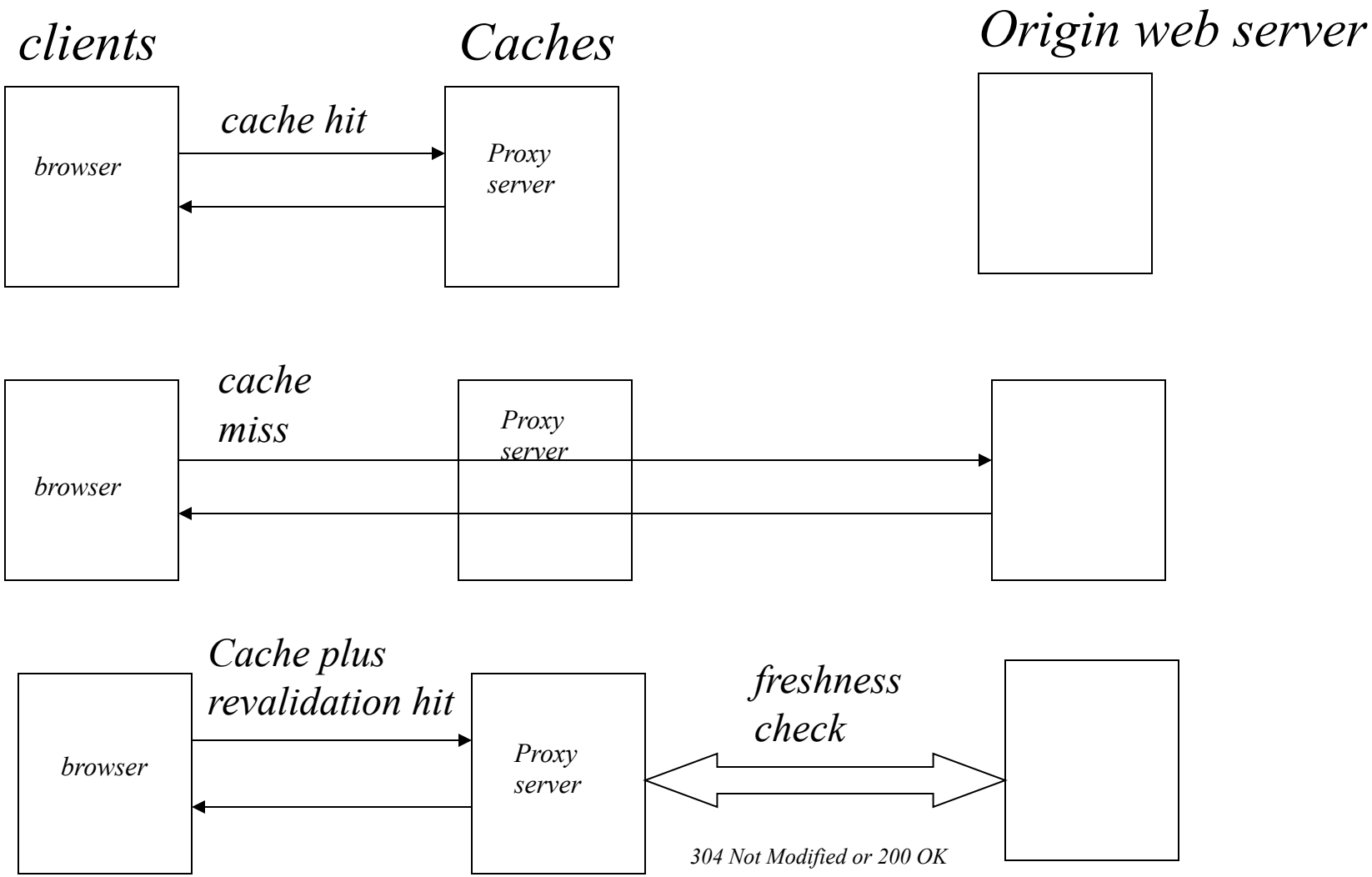
- The anonymized message is

```
GET /something/file.html HTTP/1.0
Date:Sun, 01 Oct 2000 23:25:17 GMT
User-agent: Mozilla/4.75 (Win98; U)
```

Caching and Revalidation

- Every Web object changes over time, so each Web object has a useful life, or "freshness."
  - Caches must determine whether or not their copy of an object is still "fresh," or whether they need to retrieve a new copy from the origin server.
- Caching and revalidation refer to the process of storing copies of documents retrieved by the proxy server to local storage (disk) or memory, so it is readily available
- Caching improves performance, reduces latency, and saves network bandwidth
- Potential disadvantage is the risk of receiving stale data
- HTTP/1.1 added the concept of **conditional GET**
  - retrieve a document based upon whether it has been modified since the last access
  - an up-to-date check that returns “304 not modified” means the cache copy can be used
  - successful revalidations are faster than cache misses and nearly identical to cache misses

Revalidations



Validation

- Validation is used by servers and caches to communicate when an object has changed.
  - caches avoid having to download the entire object when they already have a copy locally, but they're not sure if it's still fresh.
- The most common validator is the ***Last-Modified*** time.
- If a validator is not present, and there isn't any freshness information (***Expires*** or ***Cache-Control***) available, most caches will not store an object at all.
- HTTP 1.1 introduced a new kind of validator called the ***ETag***.
  - ETags are unique identifiers that are generated by the server and changed every time the object does.
- Almost all caches use Last-Modified times and E-Tags in determining if an object is fresh.
- Most modern Web servers will generate both ETag and Last-Modified validators for static content automatically

Browsers and Proxies Use the last_modified Date

- Retrieve a document myfile.html if it has been modified since the last access, E.g., a response:

HTTP/1.0 200 OK

Server: Apache 1.3.20

Date: Wed, 30 Dec 2008 09:23:43 GMT

Last-modified: Tue, 29 Dec 2008 10:11:12 GMT

Content-type: text/html

Content-length: 3456

- a conditional Get uses the timestamp from Last-modified header and sends it along in a If-Modified-Since:

GET /htdocs/myfile.html HTTP/1.0

User-agent: Mozilla/4.7

Accept: text/html, image/*

If-modified-since: Tue, 29 Dec 2008 10:11:12 GMT

Web Server's Respond to Conditional GET Requests

- if the document is unchanged, server responds

HTTP/1.0 304 Not Modified

Server: Apache 1.3.1

Date: Thu, 31 Dec 2008 14:15:22 GMT

- if the document has changed the server responds

HTTP/1.0 200 OK

Server: Apache 1.3.1

Date: Thu, 31 Dec 2008 14:15:22 GMT

Content-type: text/html

Content-length: 6745

< document follows>

How Web Caches Work

1. If the object's headers tell the cache not to keep the object, it won't
2. If the object is authenticated or secure, it won't be cached.
3. A cached object is considered *fresh* (that is, able to be sent to a client without checking with the origin server) if:
    1. It has an expiry time or other age-controlling directive set and is still within the fresh period.
    2. If a browser cache has already seen the object and has been set to check once a session.
    3. If a proxy cache has seen the object recently, and it was modified relatively long ago.
4. Fresh documents are served directly from the cache, without checking with the origin server.
5. If an object is stale, the origin server will be asked to *validate* the object or tell the cache whether the copy that it has is still good.

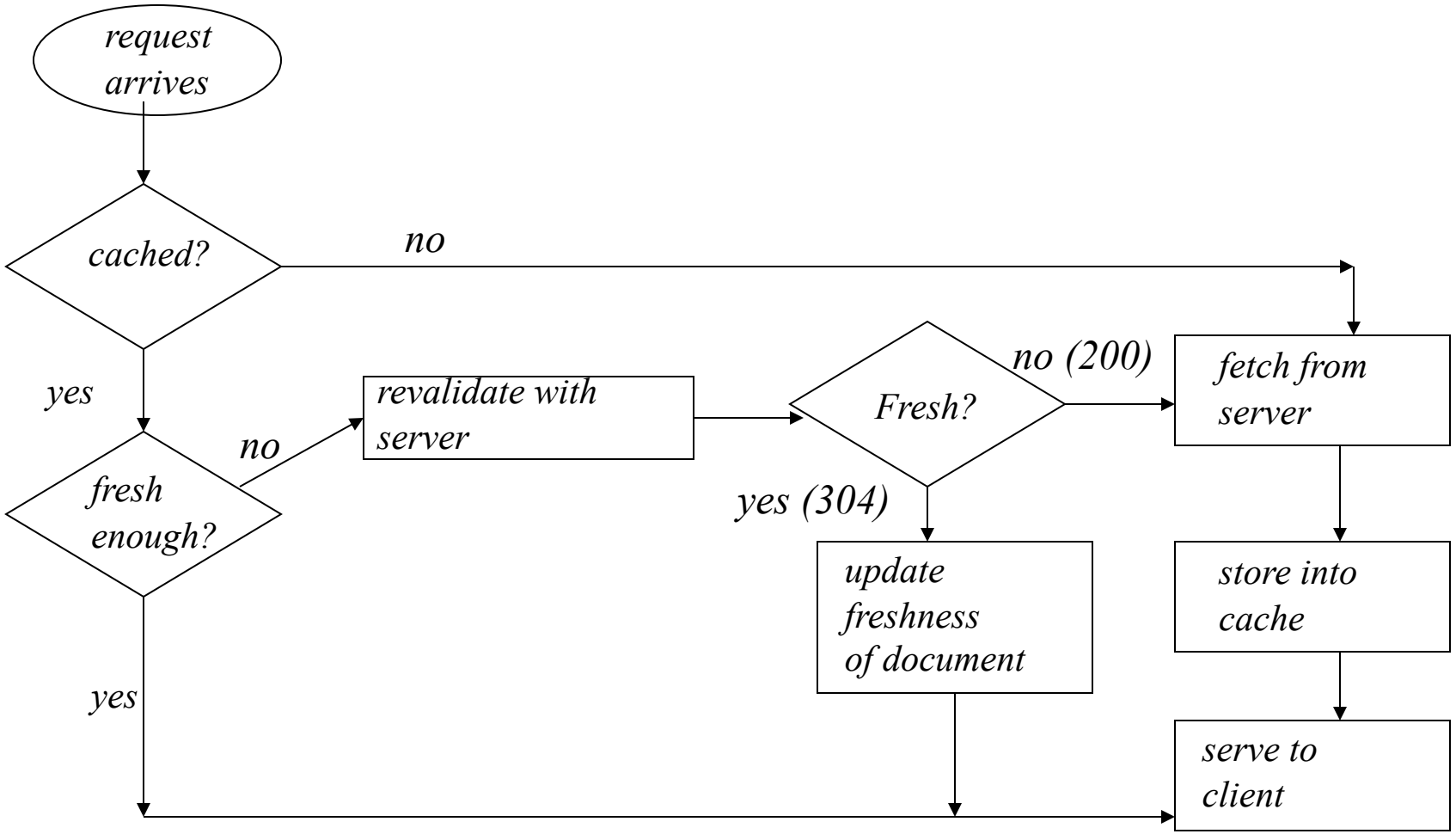
Proxies Need to Know When to Download a Fresh Version of a Page

- The cache must determine the freshness of each object
- As just shown, a cache can pass a "get-if-modified" request to the server each time an object is requested.
  - When a cache receives a request for an object that it has already stored, it sends a "get-if-modified" request. **If the cache's modification date for that object is older that the server's,** the cache retrieves a new copy of the object.
- Use freshness data (such as the time expiration data on the object's header under HTTP 1.1) to evaluate a stored object, and then retrieve a fresh copy of an object when its freshness expires.
- Apply heuristics to judge the life expectancy of each object based on the elapsed time since the object was last modified.

Freshness Heuristics

- **When the proxy retrieves an object from the server**, the cache notes the Last Modified date on the object, and then assumes that the object has an additional useful life that is a fixed percentage (10%, for example) of the time already elapsed since the last modification.
  - For example, if an object was last modified 10 days before the cache fetches it, the cache assumes that the object will grow stale in one more day, and the cache itself satisfies requests for that object for that day.
  - When the freshness period elapses, the cache will return to the server to revalidate the object's freshness and obtain a new copy if the object has changed.
- Heuristics sometimes results in stale files being sent from the cache, because the 10% additional freshness allowance sometimes proves to be too generous. On the other hand, heuristics can also result in unnecessary traffic between the cache and the server when the cache sends "get-if-modified" requests that prove unnecessary.

Cache GET Request Flowchart



Guaranteeing Freshness

- Using conditional GET for all requests guarantees that stale data is never returned, but is very wasteful as most requests will return “not modified”
- thus, servers use heuristics
  - the age of the document at the time of last retrieval; if a document is fairly old, its unlikely to change soon; e.g., image files
  - dynamic pages, e.g., weather reports, usually include

  Expires: 0

  Expires: now

  Expires: Thu, 01 Jan 1970 00:00:00 GMT

  Expires: current time

Cache Control Response Header

- Pragma: no-cache, does not work in most browsers, deprecated in favor of cache-control command

cache-control = public, responses from this server may be stored without restriction

cache-control = "max-age=43,200" the document should be considered stale after 43,200 seconds

cache-control = "must revalidate" means the caching service must revalidate the document after it becomes stale from the originating server or report an error. It must not pass on the stale version

cache-control = "no-cache" means responses can be cached, but the cached copy may not be reused for subsequent requests without revalidating the cached copy with the originating server

cache-control = "no store" means this response may not be cached on nonvolatile storage

HTML Meta Tags vs. HTTP Headers

- HTML authors can put tags in a document's <HEAD> section that describe its attributes. These *Meta tags* are often used in the belief that they can mark a document as uncacheable or expire it at a certain time.
- <META http-equiv="Expires" content="Tue, 20 Aug 2005 14:25:27 GMT">
- <META HTTP-EQUIV="Pragma" CONTENT="no-cache">
- <META HTTP-EQUIV="Refresh" CONTENT="3;URL=http://www.some.org/some.html">
- <META HTTP-EQUIV="Set-Cookie" CONTENT="cookievalue=xxx;expires=Friday, 30-Dec-05 23:59:59 GMT; path=/">
- Meta tags are easy to use but **aren't very effective**. They are never honored by proxy caches, and they are honored by browser caches only together with HTTP Headers.
- For a more detailed explanation see

http://www.htmlgoodies.com/beyond/reference/article.php/3472881

HTTP Headers

- *HTTP headers* give you a **lot of control** over how both browser caches and proxies handle your objects.
  - They can't be seen in the HTML and are usually automatically generated by the Web server. However, you can control them to some degree, depending on the server you use.
- *HTTP headers* are sent by the server in front of the HTML, and only seen by the browser and any intermediate caches.
- A typical HTTP 1.1 response headers might look like this:

```
HTTP/1.1 200 OK
Date: Fri, 30 Oct 1998 13:19:41 GMT
Server: Apache/1.3.3 (Unix)
Cache-Control: max-age=3600, must-revalidate
Expires: Fri, 30 Oct 1998 14:19:41 GMT
Last-Modified: Mon, 29 Jun 1998 02:28:12 GMT
ETag: "3e86-410-3596fbbc"
Content-Length: 1040
Content-Type: text/html
```

HTTP Headers

- Set up is done using directives of the specific server
- In Apache 2.2+ the module **mod_expires** lists the directives that can be used to set this up:
  - http://httpd.apache.org/docs/2.2/mod/mod_expires.html
  - Directives can be located in the httpd.conf file or in .htaccess files in specific subdirectories
  - the server will satisfy directory requests with the cache controls ETag and LastModified, if IndexOptions includes the TrackModified directive.
- With IIS, directives are part of ASP code:
  - http://support.microsoft.com/default.aspx?scid=kb;en-us;234067

```
<% Response.CacheControl = "no-cache" %>
<% Response.AddHeader "Pragma", "no-cache" %>
<% Response.Expires = -1 %>
```