The World Wide Web

- Lots of technologies
  - HTML, Java, Flash, ASP, CGI, HTTP
The World Wide Web

- Lots of technologies
  - HTML, Java, Flash, ASP, CGI, HTTP
- Can roughly divide into three groups
  - Content formats: HTML, Java, Flash
  - Dynamic content frameworks: ASP, CGI
  - Transfer protocols: HTTP

- Our focus
  - Transfer protocols and related technology
  - E.g. HTTP, not HTML
Outline

- HTTP Intro
- HTTP Improvements
- Web Caching
- Content Distribution Networks

HTTP Intro

- Original Purpose of HTTP: a way to share scientific publications with colleagues
HTTP Intro

- Original Purpose of HTTP: a way to share scientific publications with colleagues
- Usage today
  - Publishing content (personal web pages, news sites, etc.)
  - Transactions (online stores, banks, etc.)
- Major Functions of HTTP
  - A way to request information
  - A way to submit information (e.g. “I want to order the networking textbook.”)
HTTP Basics

- HTTP layered over bidirectional byte stream
  - Almost always TCP
- Interaction
  - Client sends request to server, followed by response from server to client
  - Requests/responses are encoded in text
- Stateless
  - Server maintains no information about past client requests

HTTP Request

<table>
<thead>
<tr>
<th>method</th>
<th>sp</th>
<th>URL</th>
<th>sp</th>
<th>version</th>
<th>cr</th>
<th>lf</th>
</tr>
</thead>
<tbody>
<tr>
<td>header field name</td>
<td>:</td>
<td>value</td>
<td>cr</td>
<td>lf</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Entity Body
HTTP Request

- Request line: specifies what we want to do
  - Method
    - GET – return URI
    - HEAD – return headers only of GET response
    - POST – send data to the server (forms, etc.)
  - URI
    - E.g. /index.html if connecting directly
    - E.g. http://www.cs.cmu.edu/index.html with a proxy
  - HTTP version

HTTP Request

- Request headers
  - Authorization – authentication info
  - Acceptable document types/encodings
  - From – user email
  - If-Modified-Since
  - Referrer – what caused this page to be requested
  - User-Agent – client software
- Blank-line
- Body
HTTP Request Example

GET / HTTP/1.1
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)
Host: www.cs.cmu.edu
Connection: Keep-Alive

HTTP Response

- Status-line
- Headers
- Blank-line
- Body
HTTP Response

- Status-line
  - HTTP version
  - 3 digit response code
    - 1XX – informational
    - 2XX – success
      - 200 OK
    - 3XX – redirection
      - 301 Moved Permanently
      - 303 Moved Temporarily
      - 304 Not Modified
    - 4XX – client error
      - 404 Not Found
    - 5XX – server error
      - 505 HTTP Version Not Supported
  - Reason phrase

- Headers
  - Location – for redirection
  - Server – server software
  - WWW-Authenticate – request for authentication
  - Allow – list of methods supported (get, head, etc)
  - Content-Encoding – E.g x-gzip
  - Content-Length
  - Content-Type
  - Expires
  - Last-Modified
- Blank-line
- Body
HTTP Response Example

HTTP/1.1 200 OK
Date: Tue, 27 Mar 2001 03:49:38 GMT
Server: Apache/1.3.14 (Unix) (Red-Hat/Linux)
         mod_ssl/2.7.1 OpenSSL/0.9.5a DAV/1.0.2
         PHP/4.0.1pl2 mod_perl/1.24
Last- Modified: Mon, 29 Jan 2001 17:54:18 GMT
ETag: "7a11f-10ed-3a75ae4a"
Accept-Ranges: bytes
Content-Length: 4333
Keep- Alive: timeout=15, max=100
Connection: Keep- Alive
Content-Type: text/html

Embedded Objects

- Web pages include references to “embedded objects” (e.g. images)
- In HTTP, must make a separate request for each object
Embedded Objects and HTTP 0.9/1.0

- One request/response per TCP connection
- Advantage: simple to implement
- Disadvantages
  - Multiple connection setups → three-way handshake each time
    - Several extra round trips added to transfer
  - Multiple slow starts

Single Transfer Example

- Client opens TCP connection
- Client sends HTTP request for HTML
- Client parses HTML
- Client opens TCP connection
- Client sends HTTP request for image
- Image begins to arrive

Server reads from disk

Client

0 RTT
SYN
1 RTT
DAT
ACK
SYN
ACK
DAT
FIN
2 RTT
ACK
SYN
ACK
FIN
3 RTT
DAT
ACK
SYN
ACK
4 RTT
DAT
Problems with HTTP 0.9/1.0

- Short transfers do poorly with TCP
  - TCP stays in slow start
  - Loss recovery is poor when windows are small
- Lots of extra connections
  - Increases server state/processing

Outline

- HTTP Intro
- HTTP Improvements
- Web Caching
- Content Distribution Networks
Performance Improvement: Multiple Connections

- Early browsers (e.g. Mosaic) fetched one object at a time
- Improvement: Use multiple concurrent connections to improve response time
  - First used in Netscape browser
  - Different parts of Web page arrive independently
  - Can grab more of the network bandwidth than other users
- Doesn’t necessarily improve response time
  - TCP loss recovery ends up being timeout dominated because windows are small

Performance Improvement: Persistent Connections

- Multiplex multiple transfers onto one TCP connection
  - Serialized requests \(\rightarrow\) client makes next request only after previous response
- How to demultiplex requests/responses
  - Content-length
    - Must know size of response in advance
    - Doesn’t work for dynamic content
  - Delimiter
    - Server must examine response for delimiter string
    - Limits performance
- In practice
  - Use content-length for static content
  - Close connection for dynamic content
Persistent Connection Example

- Client sends HTTP request for HTML
  - 0 RTT
  - 1 RTT: Client parses HTML
    - 2 RTT: Client sends HTTP request for image
    - 3 RTT: Image begins to arrive

Server
- Server reads from disk

Persistent Connection Performance

- Object response time improves
- Who benefits most: large objects or small?
Persistent Connection Performance

- Object response time improves
- Who benefits most: large objects or small?
  - Benefits greatest for small objects
  - Up to 2x improvement in response time
- Server resource utilization reduced due to fewer connection establishments and fewer active connections
- TCP behavior improved
  - Longer connections help adaptation to available bandwidth
  - Larger congestion window improves loss recovery
Performance Improvement: Pipelined Requests

- Serialized requests: 1 RTT per embedded object
- Can we do better?
Performance Improvement: Pipelined Requests

- Serialized requests: 1 RTT per embedded object
- Can we do better?
- Yes: pipeline the requests
  - GETALL – request HTML document and all embeds
    - Requires server to parse HTML files
    - Doesn’t consider client cached documents
  - GETLIST – request a sequence of documents
    - Implemented by sending multiple GETs

Persistent HTTP

Nonpersistent HTTP issues:
- Requires 2 RTTs per object
- OS must work and allocate host resources for each TCP connection
- But browsers often open parallel TCP connections to fetch referenced objects

Persistent HTTP:
- Server leaves connection open after sending response
- Subsequent HTTP messages between same client/server are sent over connection

Persistent without pipelining:
- Client issues new request only when previous response has been received
- One RTT for each referenced object

Persistent with pipelining:
- Default in HTTP/1.1
- Client sends requests as soon as it encounters a referenced object
- As little as one RTT (overhead) for all the referenced objects
Remaining Problems

• Serialized replies
  • Stall in one reply holds up all other replies
  • Much of the useful information in first few bytes (e.g. progressive JPEG)
  • Possible solution: “packetize” transfer over TCP
    • Could use range requests

Design Critique

• Persistent connections, pipelined requests, and pipelined replies are used by HTTP to work around limitations of TCP
• Application specific solution to transport protocol problems
• Alternate approach:
  • Solve the problem at the transport layer
  • Change TCP so it works well with multiple simultaneous connections
  • More difficult to deploy
  • Better or worse?
Web Caching

- Goals
  - Reduce client response time
  - Reduce (wide-area) network bandwidth usage
  - Reduce load on servers

- Method: Caching
  - Keep a copy of the objects on a “local” disk
  - Always check cache before asking server
    - If object is found in cache, use it
    - Else retrieve it from server
  - Browser caches and proxy caches
Caching Example

Assumptions
• Average object size = 100,000 bits (~12KB)
• Avg. request rate from institution’s browser 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

Possible solution (cont’d)

Possible solution
• Increase bandwidth of access link to, say, 10 Mbps
• Often a costly upgrade

Consequences
• Utilization on LAN = 15%
• Utilization on access link = 15%
• Total delay = Internet delay + access delay + LAN delay
  = 2 sec + msecs + msecs
Caching Example (cont’d)

Alternate Solution: Install cache
- Suppose hit rate is .4

Consequence
- 40% requests will be satisfied almost immediately (say 10 msec)
- 60% requests satisfied by origin server
- Utilization of access link reduced to 60%, resulting in negligible delays
- Weighted average of delays = .6*2 sec + .4*10msecs < 1.3 secs
- Better than upgrading link!

Caching and HTTP

- Caching requires protocol support
  - Need to detect modified objects
  - If-Modified-Since requests to check
    - HTTP 0.9/1.0 used just date
    - HTTP 1.1 has file signature ("ETag") as well
  - When/how often should the original be checked for changes?
    - Check every access?
    - Check each session? Day? Etc?
    - Use Expires header if available
    - Else: use history (Last-Modified)
Example HTTP Cache Check Request

GET / HTTP/1.1
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
If-Modified-Since: Mon, 29 Jan 2001 17:54:18 GMT
If-None-Match: "7a11f-10ed-3a75ae4a"
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)
Host: www.cs.cmu.edu
Connection: Keep-Alive

Example HTTP Cache Check Response

HTTP/1.1 304 Not Modified
Date: Tue, 27 Mar 2001 03:50:51 GMT
Server: Apache/1.3.14 (Unix) (Red-Hat/Linux)
         mod_ssl/2.7.1 OpenSSL/0.9.5a DAV/1.0.2
         PHP/4.0.1pl2 mod_perl/1.24
Connection: Keep-Alive
Keep-Alive: timeout=15, max=100
ETag: "7a11f-10ed-3a75ae4a"
Web Proxy Caches

- User configures 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

Web Proxy Caches

- Caching
  - Provides a centralized coordination point to share information across clients
- Filtering
  - Proxy is only host that can access Internet
  - Security -- Administrators make sure that it is secure
  - Policy -- Filter requests
Web Proxy Caches: Sources for Misses

- Typical hit rates: 30-50%
- Capacity
  - Can only cache a limited set of objects
  - Cache typically on disk
  - In practice: not a significant factor
- Compulsory retrievals
  - First time access to document
  - Non-cacheable documents
- Consistency
  - Document has been updated/expired before reuse

Web Proxy Caches: Challenges

- Over half of all HTTP objects are uncachable – why?
Web Proxy Caches: Challenges

- Over half of all HTTP objects are uncachable – why?

- Many issues -- Not easily solvable
  - Dynamic data → stock prices, scores, web cams
  - CGI scripts → results based on passed parameters
  - SSL → encrypted data is not cacheable
    - Most web clients don’t handle mixed pages well → many generic objects transferred with SSL
  - Hit metering → owner wants to measure # of hits for revenue, etc.

Web Proxy Caches: Hit Rates

- Typical hit rates: 30-50%
- Over half of all HTTP objects are uncachable

- Contradiction?
Web Proxy Caches: Hit Rates

- Typical hit rates: 30-50%
- Over half of all HTTP objects are uncacheable

- Contradiction?
  - No: the cacheable objects are referenced more frequently

Leases

- Only consistency mechanism in HTTP is for clients to poll server for updates
- Should HTTP also support invalidations?
  - Problem: server would have to keep track of many, many clients who may have document
  - Possible solution: leases
- Leases – server promises to provide invalidates for a particular lease duration
- Server can adapt time/duration of lease as needed
  - To number of clients, frequency of page change…
Outline

- HTTP Intro
- HTTP Improvements
- Web Caching
- Content Distribution Networks

Content Distribution Networks (CDNs)

- CDN company installs hundreds of CDN servers throughout Internet
- Content providers pay CDN to distribute content for them
- CDN replicates content in CDN servers

image of CDN distribution network
Challenges for CDNs

• How to replicate content
• Where to place replicated content
• How to find replicated content
• How to direct clients towards replica

• Some of Akamai’s solutions
Akamai: Directing Clients

- Clients fetch html document from primary server
  - E.g. fetch index.html from cnn.com
- URLs for replicated content are replaced in html
  - E.g. `<img src="http://cnn.com/af/x.gif">` replaced with
    `<img src="http://a73.g.akamaitech.net/7/23/cnn.com/af/x.gif">`
- Client is forced to resolve aXYZ.g.akamaitech.net hostname

Akamai: Directing Clients

- Root server gives NS record for akamai.net
- Akamai.net name server returns NS record for g.akamaitech.net
  - Name server chosen to be in region of client’s name server
  - TTL of answer is large
- G.akamaitech.net nameserver chooses amongst servers in region
  - Should try to chose server that has file in cache
  - Which server is that? (location problem)
  - TTL of answer is small
Akamai: Replicating Content

- Akamai only replicates static content
- Modified URL contains original file name
- Content is replicated on-demand
  - Akamai server is asked for content
  - First checks local cache
  - If not in cache, requests file from primary server and caches file

Locating Replicated Content

- Given document XYZ, we need to choose a server to use
- Want a scheme with:
  - Balanced – load is equal across servers
  - Smoothness – few objects move when servers crash or new servers are added
Locating Replicated Content: First Try

- Use modulo arithmetic
- Number servers from 1…n
  - Place document XYZ on server (XYZ mod n)
- Does this scheme have good balance?

- Does this scheme have good smoothness?
Locating Replicated Content: First Try

- Use modulo arithmetic
- Number servers from 1…n
  - Place document XYZ on server (XYZ mod n)
- Does this scheme have good balance?
- Does this scheme have good smoothness?
  - When a server fails, n \rightarrow n-1
  - How many objects must move?

Locating Replicated Content: Consistent Hashing

- Assign each server to a random point on a circle
- Map each object to random position on circle
- Go to server that is closest to the object (going clockwise)
Locating Replicated Content: Consistent Hashing

- Assign each server to a random point on a circle
- Map each object to random position on circle
- Go to server that is closest to the object (going clockwise)
- This scheme gives both balance and smoothness

How Akamai Works: Client’s First Request

- End-user requests cnn.com (content provider)
- cnn.com requests DNS root server
- DNS root server resolves cnn.com to Akamai server
- Akamai server serves foo.jpg
- End-user requests /cnn.com/foo.jpg
- Closest Akamai server serves /cnn.com/foo.jpg
How Akamai Works: Client’s Subsequent Requests

cnn.com (content provider) \rightarrow DNS root server \rightarrow Akamai server

1. End-user
2. Get index.html

7. Akamai high-level DNS server
8. Akamai low-level DNS server
9. Closest Akamai server

10. Get /cnn.com/foo.jpg