Internet and Intranet Protocols and Applications

Lecture 8a: WWW Proxy Servers and Cookies

March 12, 2003

Arthur Goldberg
Computer Science Department
New York University
artg@cs.nyu.edu
Terminology

• **Origin Server**
  – The Web server that hosts the resource

• **Proxy Server**
  – Intermediate server that accepts requests from clients and forwards them to (towards) origin servers, to other proxy servers, or services request from its cache.
  – Acts as server to requesting client, and as client to origin server
Web Caches (proxy server)

Goal: satisfy client request without involving origin server

- configure browser: Web accesses via web cache
- client sends all http requests to web cache
  - if object at web cache, web cache immediately returns object in http response
  - else requests object from origin server, then returns http response to client
Why Web Caching?

Assume: cache is “close” to client

• smaller response time

• decrease traffic to distant servers
  – link out of institutional/local ISP network is often a bottleneck
Web Caching Summary

• Web proxy Servers store copies of documents retrieved from origin servers

• Advantages
  – Improve performance (latency reduction, bandwidth conservation)
  – Advanced access control (intermediate requester in firewalled DMZ, authentication & authorization)
  – Advanced filtering (e.g. detect espionage!)
  – Logging and auditing

• Disadvantages
  – stale (out of date) data
Proxy Server: Basic Operation

- Accept connection request from client
  - establishes new Socket client_sock
- Read HTTP request
- Parse HTTP request
  - reject invalid requests with appropriate response code
- Connect to (towards) requested server
  - establishes new socket serv_sock
- Send original HTTP request to server
  - or to next proxy on path to server
Proxy Server:  *Basic Operation (continued)*

- Read response from Server
  - If time-out server connection, then
    - 504 Gateway Timeout
- Send response to client
- If **Connection: close** header received, close client connection (client_sock)
- What about server connection (serv_sock)?
HTTP 1.1 Cache Control
(Definitions)

• **Freshness** of objects: a document is fresh when:
  – it is first retrieved from an origin server.
  – when the origin server is contacted to make up-to-date check
  – when its *age* does not exceed its *freshness lifetime*

• **Age** of an object
  – time that has elapsed since object was retrieved, or
  – time since last up-to-date check
Age and Freshness Lifetime  
(determining an object’s age)

• Cache servers use **Date:** header in response plus some compensation for latency between response creation and receipt to calculate an initial-age.

• When Cache server sends this object in a response, it adds elapsed time (since object receipt) and initial-age and sends an **Age:** header
Age and Freshness Lifetime
(determining an Object’s Freshness Lifetime)

• **Cache-Control**: header contains `max-age` directive, or

• **Expires**: header in response contains date and time the object becomes “stale”

• Since both of these values come from server, no latency compensation is needed.
HTTP 1.1 Cache Control Directives
(controlling an object’s “cacheability”)
RFC 2616 Section 14.9

- **Cache-Control:** general header is used to specify directives that MUST be obeyed by ALL proxy servers handling the request or response.
- Directives used in Requests
  - `no-cache` an end-to-end revalidation should be preformed
  - `no-store` sensitive information: do not store any part of request or response on disk
  - `max-age=<delta-seconds>` max age acceptable to client
HTTP 1.1 Cache Control Directives
(controlling an object’s “cacheability”)

- Directives used in responses
  
  **public** response is cacheable by any cache (proxy or client)

  **private** response is cacheable by client only

  **no-cache** cache MUST NOT use the response to satisfy a subsequent request (for example, dynamic pages)

  **no-store** response may not be written to disk
Cache Architectures

• Components of a Web proxy cache
  – Network communications
  – Storage mechanism for storing the cache data
  – Mapping mechanism to establish relationship between URLs and their cached copies
  – Format for cached object content and its metadata
Cache Architecture: mapping

• Direct mapping
  – e.g., map URL to a file system path
  – direct mappings are reversible

• Hash mapping
  – compute some unique ID
  – could be file name or index to table
  – not reversible

• Why do we care about reversibility?
Existing Mapping Mechanisms

• Directly mapping URLs to filesystem
  – CERN httpd used a tree map (like DNS tree!)
  – Easy to implement, but not a good performer
    • long pathnames = long inode search
    • garbage collection requires complete traversal of tree
Existing Mapping Mechanisms

• Hashing URLs (Netscape Proxy server uses URL hashing)

• Object location (on disk) based on MD5 hash
  – very fast
  – good distribution of different object types (image, text) across cache
  – disadvantage: cannot compute URLs from hash
Alternative Cache Protocols

• **On-Demand**
  – document does not exist in cache unless it has been requested (at least once) by some client

• **On-Command, or Pre-Fetch**
  – proxy server automatically retrieves documents (or even entire web sites!) at regular intervals
General Purpose Proxy Servers

• **Transparency**
  – users get same response whether connection was direct or to proxy
  – (a non-transparent proxy modifies content in some way)

• **Use is client controlled**
  – client programs (e.g., browsers) can be configured to use (or not use) proxy servers.

• **Origin Server is unaware of proxy server**
  – Origin Server does not have to process request from proxy differently than from other client

• **Example protocols**
  – Ftp, ssh, socks, telnet, SMTP

• **Typical Location**
  – Firewalled DMZ
Other Intermediate Systems

• **Firewall**
  – General term for hardware, software, or combination used to protect internal network from intruders.
  – Uses packet filtering to enforce generic security policies
  – Uses application level proxy servers to enforce protocol-specific policies

• **Packet filter**
  – Control based on something in packet headers (e.g., IP addresses or port numbers)

• **Application level proxy**
  – Control based on knowledge of application level protocol (e.g., SMTP headers or HTTP methods)
HTTP State Management: Cookies

• We said earlier that HTTP is a stateless protocol.
• We also said that stateful protocols can provide improved performance. This feature is usually established by the idea of a “session” between client and server.
• So, cookies enable HTTP sessions.
COOKIES (briefly)

• Cookie protocol - RFC 2109
• A cookie is a token given to a client by a server.
  – Server sends *Set-cookie:* header in response
  – Client associates cookie with issuing server (directory)
• The token is just a file with a simple format (name/value pairs)
• Each cookie has a unique name
Client-server interaction: cookies

- Server sends “cookie” to client in response most

  `Set-cookie: 1678453`

- Client presents cookie in later requests

  `cookie: 1678453`

- Server matches presented-cookie with server-stored info
  - Authentication
  - Remembering user preferences, previous choices
Cookies and Proxies

• HTTP cookies are meant for the end-point entities (client and origin server)
• Cannot be used for state between proxy and end-point
• Why would we need cookies for proxy servers?
A Case for Proxy Cookies

• A common use of cookies is for authentication - so cookie may contain IP Address of client
• In a network of load balancing servers, requests between two endpoints may not follow the same route. This would invalidate the client cookie!
• Proxy cookies might be used to establish proxy credentials.
• Note: proxy cookies do not exist!
• Can you think of other approaches?