Web Proxies and Caching
CS 360 Internet Programming

Daniel Zappala

Brigham Young University
Computer Science Department
Proxy

- an intermediary program that acts as both a server and a client for the purpose of forwarding requests
- accepts requests from other clients and handles them either internally or by passing them on to other servers
- examples
  - caching responses
  - anonymizing requests
  - filtering content
Transparent Proxies

- **transparent proxy**: does not modify the request other than superficially
  - caches place identifying information in headers
  - required by HTTP/1.1
- **non-transparent proxy**: may modify the request or response
  - anonymize request
  - filter content
  - compress response
Anonymizing Requests

- anonymizing proxy hides IP address of client (client is not necessarily in the same organization)
  - may not hide the User-Agent header
  - may not drop cookies

- onion routing: setup a sequence of proxies along an unpredictable path, using encryption at each step
  - prevents eavesdropping
  - prevents traffic analysis
  - http://tor.eff.org/
Filtering Content

- examine application-level HTTP messages to block access to certain content
  - examine URL in the GET and compare to a blacklist of websites
  - compare URL against a list of banned keywords: **anonymizing searches** often blocked
  - examine response and compare to a list of banned keywords
- BYU CS department uses DansGuardian dansguardian.org
Interception Proxies

- all web traffic is diverted to the proxy, regardless of user preference

1. Router must examine TCP header on all packets
2. A TCP packet going to port 80 is diverted to the proxy
3. Proxy must accept packets for any destination address going to port 80
4. Proxy then performs its functions – caching, filtering

- Breaks the rules and layering of IP, but so do firewalls

- A reality for most major campuses and organizations
Web Caching

- may return a cached object rather than contacting the origin web server
- need a cache consistency protocol - check whether objects in cache are up-to-date
- need a cache replacement algorithm - determine which objects to save when the cache is full
- hit rate determined by cache replacement algorithm, workload (object popularity, object size)
Reverse Proxies

- a cache that sits in front of web server
  - provide access to a server behind a firewall
  - centralize security concerns at one server
  - balance load among a set of back-end servers
  - provide one URL space for many different web sites
- can use Apache as either forward or reverse proxy
Caching Benefits for Users

- faster download - web cache is usually on the local network, where there is more available bandwidth
- lower latency - shorter propagation delay for closer servers
- less congestion - fewer users sharing bandwidth, local networks are usually over-provisioned
Caching Benefits for Web Servers

- lower load on the server - can handle more users
- lower cost since it uses less bandwidth
- **but some servers want all the traffic because they receive revenue for ads on the site**
- solutions: no caching on ads, pay per click-through on ad instead of per visit, survey users like with traditional media
Caching Benefits for the Network

- **the network as a whole**
  - less traffic traversing the Internet, since it stays on local networks
  - reduces congestion - lower delay, lower packet loss
  - improves throughput - faster transfer times

- **Internet Service Providers**
  - each ISP pays its upstream provider based on its access link speed (bits per second) or the actual amount of traffic sent over the link
  - big incentive to provide web caches for their users - reduces the amount of traffic on the access link, which reduces their overall cost
What is Cacheable?

- **Expires** header
  - date after which the response is considered stale and must be revalidated
  - cache does not need to revalidate item each time it has a cache hit
- **ETag** header
  - tag specific to a resource
  - decouples cache validation from expiration times, since clocks are not synchronized
  - cache uses **If-Match** header to check if the cached item is the same
- **Vary** header
  - lists fields that may vary in responses (e.g. language)
  - cache must check that these fields are the same in the request and the cached response
Server Control over Caching

- **Cache-Control** header specifies directives that MUST be obeyed by a cache regardless of its own algorithms.

- Restrictions on what is cacheable:
  - **public**: item MAY be cached even if normally not cacheable (e.g. responses that have an Authorize field).
  - **private**: item MUST NOT be cached (intended for one user).
  - **no-cache**: MUST NOT be returned by a cache without validation.

- Restrictions on what may be stored:
  - **no-store**: cache MUST NOT store any part of the request or response.
Browser Control over Caching

- expiration mechanism
  - **max-age**: maximum age client wants from cache
  - **max-stale**: gives maximum staleness client wants from cache
  - **min-fresh**: client wants a response that will still be fresh for a minimum amount of time

- cache revalidation and reload
  - **end-to-end reload**: user wants item from origin server, caches MUST not return a cached copy
  - **only-if-cached**: user wants item if cached, otherwise an error
  - **must-revalidate**: server says response may be cached, but must be revalidated once it is stale
  - **proxy-revalidate**: does not apply to user’s browser cache
Caching Decisions

- check to see if requested object is in cache
- check if client headers allow item to be returned
- perform **cache coherence** checks
- perform **cache replacement** if needed
Cache Coherency

- cache must ensure that what is in the cache is consistent with what the server stores
- validating
  - `If-Modified-Since`: using Date
  - `If-Match`: using ETag
- when should the server validate?
  - use a TTL to indicate how much longer the cached response will be valid
  - based on `Expires, max-age` directive, or heuristic that examines last modification time and frequency of requests
- see Squid Cache FAQ for details on Squid coherence algorithm
Cache Replacement Algorithm

- many important factors
- **access history**: keep objects that are frequently accessed
- **expiration time**: remove objects that will expire soon
- **time since last modification**: keep objects that do not change frequently
- **cost of fetching the resource**: keep in cache if it was expensive to fetch
- **cost of storing the resource**: removing large objects frees a lot of space, but they are expensive to retain
Common Cache Replacement Algorithms

- **Least Recently Used (LRU)**
  - mark objects with time of last access
  - evict object that is least recently accessed
  - old and proven in many areas of CS
  - studies show it is not the best for web caching

- **Least Frequently Used (LFU)**
  - mark objects with how frequently accessed in a given period of time
  - evict object that is least frequently used

- **Size of Object (SIZE):** evict largest object

- **Hyper-G:** first LFU, then LRU, then largest

- **Greedy-Dual Size**
  - compute a utility value for each object
  - evict object with lowest utility
  - utility uses cost of fetching, size, age
Cache Replacement Lessons

- *memory is cheap*: create a really large cache
- *lots of traffic isn’t cacheable*
- *most algorithms are good enough*
- Squid uses LRU, Greedy-Dual Size, LFU with Dynamic Aging
Motivation

- problem: bandwidth bottleneck at a server
- example
  - 100 Mbps connection
  - 100,000 KB typical web page size (with embedded content)
  - 125 requests per second
- buy more bandwidth
  - 1 Gbps (10 x more bandwidth)
  - 1250 requests per second
- hard to scale
- even harder with video
Steve Souders (Yahoo, Google): 80% of web page download time is spent fetching embedded images and scripts
want to avoid latency caused by long paths
Content Delivery Network

- replicate content at many caches, typically at the edge of the network
- use domain name of the CDN in your web pages
- client requests routed to a “nearby” server, generally through DNS, reducing loss and delay
- started with Akamai: [IEEE Internet Computing paper](#)
- see [Amazon CloudFront](#)