15-744: Computer Networking

L-6 Application Networking

Application Networking

- Web caching hierarchies
- Content distribution networks
- Assigned reading
  - [CT90] Architectural Consideration for a New Generation of Protocols
  - [FCAB98] Summary Cache: A Scalable Wide-Area Cache Sharing Protocol
  - [K+99] Web Caching with Consistent Hashing

Overview

- HTTP
- ALF
- Web caching
- Content distribution networks

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

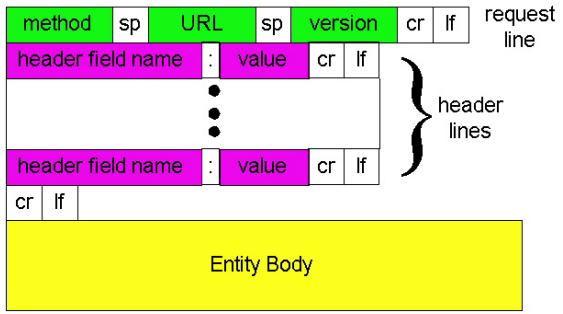
HTTP Request

- Request line
  - Method
    - GET – return URI
    - HEAD – return headers only of GET response
    - POST – send data to the server (forms, etc.)
  - URL (relative)
    - E.g., /index.html
  - HTTP version

HTTP Request (cont.)

- 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



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.intel-iris.net

Connection: Keep-Alive

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

HTTP Response (cont.)

- 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
…..

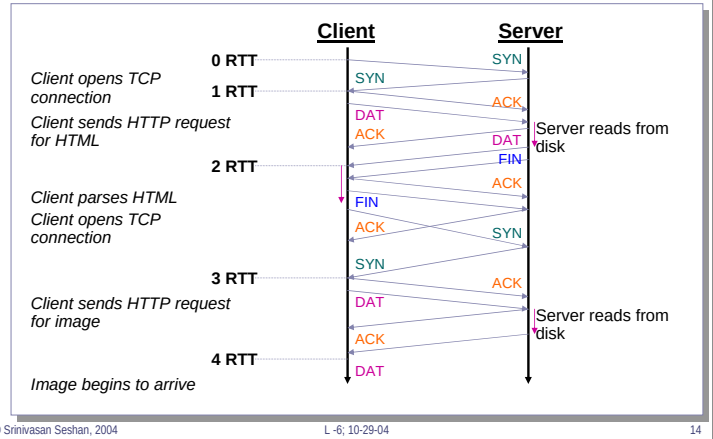
Typical Workload (Web Pages)

- Multiple (typically small) objects per page
- File sizes
  - Why different than request sizes?
  - Also heavy-tailed
    - Pareto distribution for tail
    - Lognormal for body of distribution
- Embedded references
  - Number of embedded objects = pareto – $p(x) = ak^a x^{-(a+1)}$

HTTP 0.9/1.0

- One request/response per TCP connection
  - 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



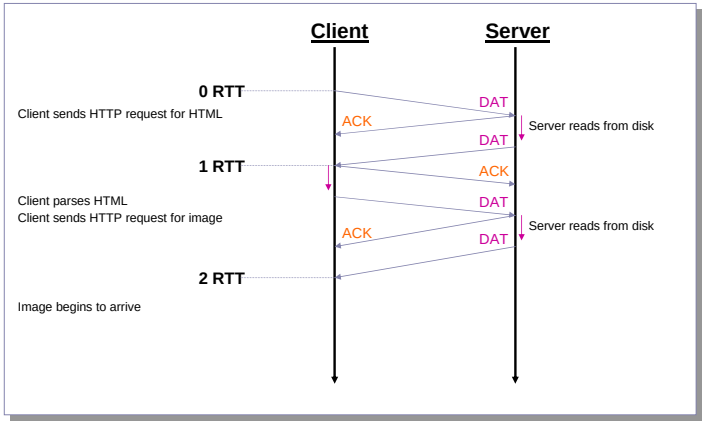
More Problems

- Short transfers are hard on TCP
  - Stuck in slow start
  - Loss recovery is poor when windows are small
- Lots of extra connections
  - Increases server state/processing
- Server also forced to keep TIME_WAIT connection state
  - Why must server keep these?
  - Tends to be an order of magnitude greater than # of active connections, why?

Netscape Solution

- Netscape
  - Use multiple concurrent connections to improve response time
    - 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
- Persistent connections [PM95]
  - Multiplex multiple transfers onto one TCP connection
    - Serialize transfers → client makes next request only after previous response

Persistent Connection Example



Persistent Connection Solution

- Serialized requests do not improve response time
- Pipelining requests
  - Getall – request HTML document and all embeds
    - Requires server to parse HTML files
    - Doesn't consider client cached documents
  - Getlist – request a set of documents
    - Implemented as a simple set of GETs
- Prefetching
  - Must carefully balance impact of unused data transfers
  - Not widely used due to poor hit rates

Persistent Connection Performance

- 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

Remaining Problems

- Application specific solution to transport protocol problems
- Stall in transfer of one object prevents delivery of others
- Serialized transmission
  - Much of the useful information in first few bytes
  - Can "packetize" transfer over TCP
    - HTTP 1.1 recommends using range requests
    - MUX protocol provides similar generic solution
- Solve the problem at the transport layer
  - Tcp-Int/CM/TCP control block interdependence

Overview

- HTTP
- ALF
- Web caching
- Content distribution networks

Integrated Layer Processing (ILP)

- Layering is convenient for architecture but not for implementations
- Combining data manipulation operations across layers provides gains
  - E.g. copy and checksum combined provides 90Mbps vs. 60Mbps separated
- Protocol design must be done carefully to enable ILP
- Data manipulation more expensive than control
  - Presentation overhead/application-specific processing >> other data manipulation processing
  - Target for ILP optimization

Application Lever Framing (ALF)

- Objective: enable application to process data ASAP
- Application response to loss
  - Retransmit (TCP applications)
  - Ignore (UDP applications)
  - Recompute/send new data (clever application)
- Expose unit of application processing (ADU) to protocol stack
  - Lower protocol layers should maintain framing

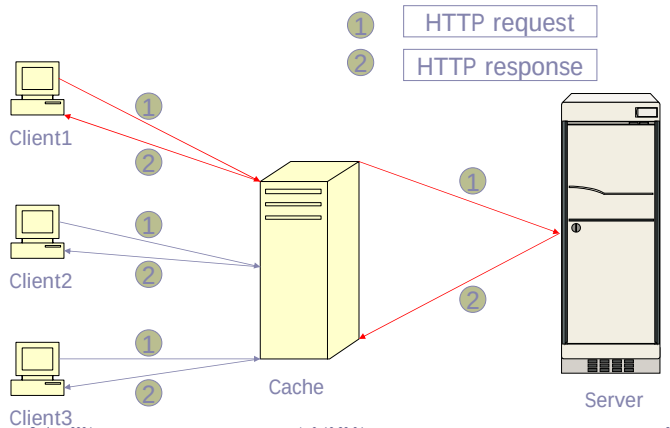
Application Data Units Requirements

- ADUs can be processed in any order
- Naming of ADUs should help identify position in stream
- Size
  - Enough to process independently
  - Impact on loss recovery
    - ADU becomes unit of loss recovery
    - What if size is very large?

Overview

- HTTP
- ALF
- Web caching
- Content distribution networks

Web caching



Why web caching?

- Client-server architecture is inherently unscalable
  - Proxies: a level of indirection
- Reduce client response time
  - Direct and indirect effect
  - Less load on the server:
    - Server does not have to over-provision for *slashdot* effect
- Reduce network bandwidth usage
  - Wide area vs. local area use
  - These two objectives are often in conflict
    - May do exhaustive local search to avoid using wide area bandwidth
    - Prefetching uses extra bandwidth to reduce client response time

HTTP support for caching

- Conditional requests (IMS)
- Servers can set expires and max-age
- Request indirection: application level routing
- Range requests, entity tag
- Cache-control header
  - Requests: min-fresh, max-stale, no-transform
  - Responses: must-revalidate, public, private, no-cache

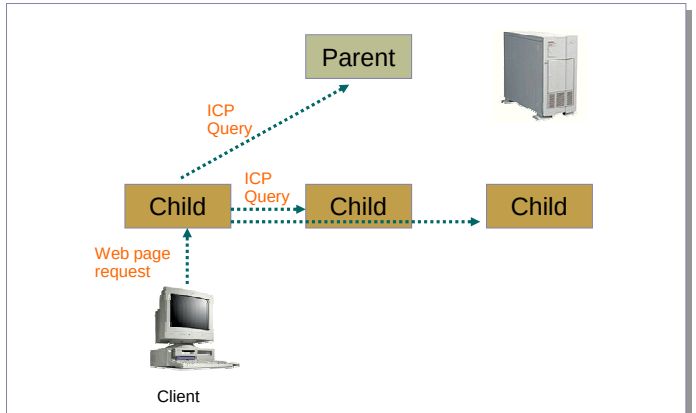
Cache Hierarchies

- Use hierarchy to scale a proxy
  - Why?
    - Larger population = higher hit rate (less compulsory misses)
    - Larger effective cache size
  - Why is population for single proxy limited?
    - Performance, administration, policy, etc.
- NLANR cache hierarchy
  - Most popular
  - 9 top level caches
  - Internet Cache Protocol based (ICP)
  - Squid/Harvest proxy
- How to locate content?

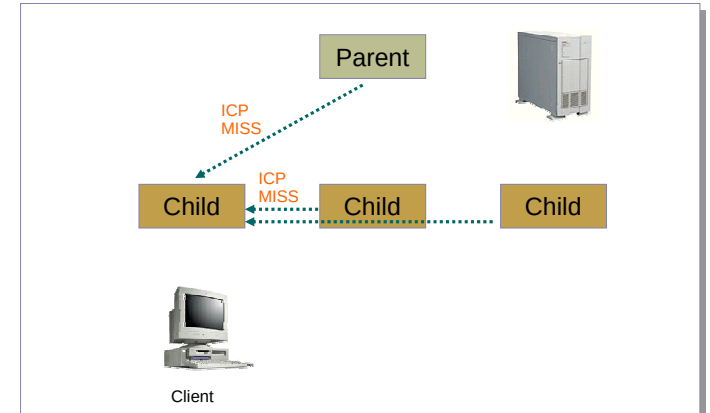
ICP (Internet cache protocol)

- Simple protocol to query another cache for content
- Uses UDP – why?
- ICP message contents
  - Type – query, hit, hit_obj, miss
  - Other – identifier, URL, version, sender address
  - Special message types used with UDP echo port
    - Used to probe server or “dumb cache”
- Query and then wait till time-out (2 sec)
- Transfers between caches still done using HTTP

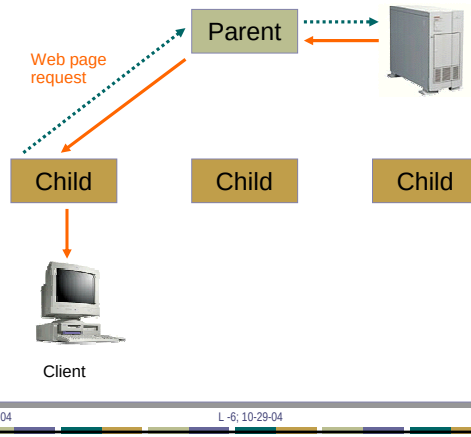
Squid



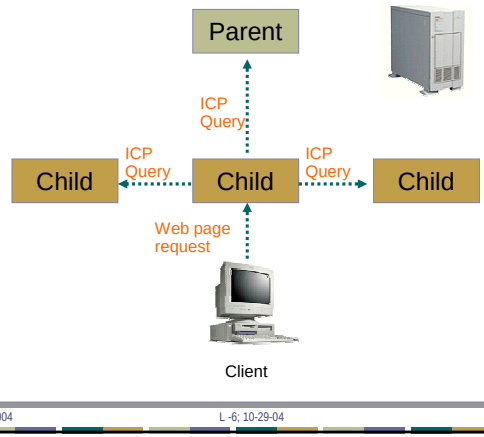
Squid



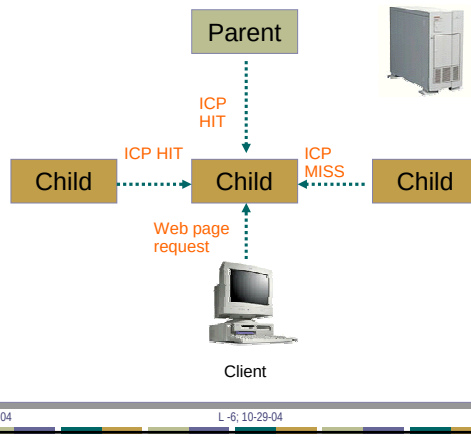
Squid



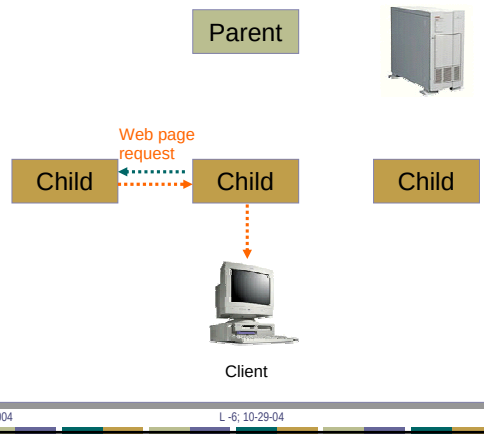
Squid



Squid



Squid



Optimal Cache Mesh Behavior

- Ideally, want the cache mesh to behave as a single cache with equivalent capacity and processing capability
- ICP: many copies of popular objects created – capacity wasted
- More than one hop needed for searching object
- Locate content – ***how***?

Hinting

- Have proxies store content as well as metadata about contents of other proxies (hints)
  - Minimizes number of hops through mesh
  - Size of hint cache is a concern – size of key vs. size of document
- Having hints can help consistency
  - Makes it possible to push updated documents or invalidations to other caches
- How to keep hints up-to-date?
  - Not critical – incorrect hint results in extra lookups, not incorrect behavior
  - Can batch updates to peers

Summary Cache

- Primary innovation – use of compact representation of cache contents
  - Typical cache has 80 GB of space and 8KB objects → 10 M objects
  - Using 16byte MD5 → 160 MB per peer
  - Solution: Bloom filters
- Delayed propagation of hints
  - Waits until threshold %age of cached documents are not in summary
  - Perhaps should have looked at %age of false hits?

Errors tolerated

- Suppose A and B share caches, A has a request for URL r that misses in A,
  - *false misses*: r is cached at B, but A didn't know
    Effect: lower total cache hit ratio
  - *false hits (false +ves)*: r is not cached at B, but A thought it is
    Effect: wasted query messages
  - *stale hits*: r is cached at B, but B's copy is stale
    Effect: wasted query messages

Bloom Filters

- Proxy contents summarize as a M bit value
- Each page stored contributes k hash values in range [1..M]
  - Bits corresponding to the k hashes set in summary
- Check for page = if all k hash bits corresponding to a page are set in summary, it is likely that proxy has summary
- Tradeoff → false positives
  - Larger M reduces false positives
  - What should M be? 8-16 * number of pages seems to work well
  - What about k? Is related to (M/number of pages) → 4 works for above M

Problems with caching

- Over 50% of all HTTP objects are uncacheable.
- Sources:
  - Dynamic data → stock prices, frequently updated content
  - 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
  - Cookies → results may be based on passed data
  - Hit metering → owner wants to measure # of hits for revenue, etc, so, cache busting

Overview

- HTTP
- ALF
- Web caching
- Content distribution networks

CDN

- Replicate content on many servers
- Challenges
  - How to replicate content
  - Where to replicate content
  - How to find replicated content
  - How to choose among known replicas
  - How to direct clients towards replica
    - DNS, HTTP 304 response, anycast, etc.
- Akamai

Server Selection

- Service is replicated in many places in network
- How to direct clients to a particular server?
  - As part of routing → anycast, cluster load balancing
  - As part of application → HTTP redirect
  - As part of naming → DNS
- Which server?
  - Lowest load → to balance load on servers
  - Best performance → to improve client performance
    - Based on Geography? RTT? Throughput? Load?
  - Any alive node → to provide fault tolerance

Routing Based

- Anycast
  - Give service a single IP address
  - Each node implementing service advertises route to address
  - Packets get routed from client to "closest" service node
    - Closest is defined by routing metrics
    - May not mirror performance/application needs
  - What about the stability of routes?

Routing Based

- Cluster load balancing
  - Router in front of cluster of nodes directs packets to server
  - Can only look at global address (L3 switching)
  - Often want to do this on a connection by connection basis – why?
    - Forces router to keep per connection state
    - L4 switching – transport headers, port numbers
  - How to choose server
    - Easiest to decide based on arrival of first packet in exchange
    - Primarily based on local load
    - Can be based on later packets (e.g. HTTP Get request) but makes system more complex

L-7 switching

- Interpret requests, content-aware switches
- Have to do the initial hand-shake
- Different proxies for different content-types
- Load balancing vs locality
- Locality means all requests (even to a popular object) serviced by a single proxy
- Caching alleviates the above problem. Why?

Application Based

- HTTP supports simple way to indicate that Web page has moved
- Server gets Get request from client
  - Decides which server is best suited for particular client and object
  - Returns HTTP redirect to that server
- Can make informed application specific decision
- May introduce additional overhead → multiple connection setup, name lookups, etc.
- While good solution in general HTTP Redirect has some design flaws – especially with current browsers?

Naming Based

- Client does name lookup for service
- Name server chooses appropriate server address
- What information can it base decision on?
  - Server load/location → must be collected
  - Name service client
    - Typically the local name server for client
- Round-robin
  - Randomly choose replica
  - Avoid hot-spots
- [Semi-]static metrics
  - Geography
  - Route metrics
  - How well would these work?

How Akamai Works

- 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

How Akamai Works

- How is content replicated?
- Akamai only replicates static content
  - Serves about 7% of the Internet traffic !
- Modified name contains original file
- Akamai server is asked for content
  - First checks local cache
  - If not in cache, requests file from primary server and caches file

How Akamai Works

- 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 is large
- G.akamaitech.net nameserver choses server in region
  - Should try to chose server that has file in cache - How to choose?
  - Uses aXYZ name and consistent hash
  - TTL is small

Hashing

- Advantages
  - Let the CDN nodes are numbered 1..m
  - Client uses a ***good*** hash function to map a URL to 1..m
  - Say hash (url) = *x*, so, client fetches content from node *x*
  - No duplication – not being fault tolerant.
  - One hop access
  - Any problems?
    - What happens if a node goes down?
    - What happens if a node comes back up?
    - What if different nodes have different views?

Robust hashing

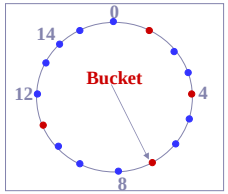
- Let 90 documents, node 1..9, node 10 which was dead is alive again
- % of documents in the wrong node?
  - 10, 19-20, 28-30, 37-40, 46-50, 55-60, 64-70, 73-80, 82-90
  - *Disruption coefficient* = ½
  - Unacceptable, use consistent hashing – idea behind Akamai!

Consistent Hash

- "view" = subset of all hash buckets that are visible
- Desired features
  - Balanced – in any one view, load is equal across buckets
  - Smoothness – little impact on hash bucket contents when buckets are added/removed
  - Spread – small set of hash buckets that may hold an object regardless of views
  - Load – across all views # of objects assigned to hash bucket is small

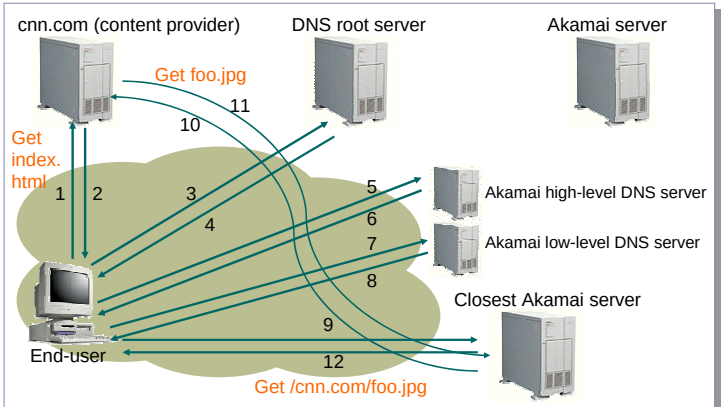
Consistent Hash – Example

- Construction
  - Assign each of C hash buckets to random points on mod 2^n circle, where, hash key size = *n*.
  - Map object to random position on circle
  - Hash of object = closest clockwise bucket

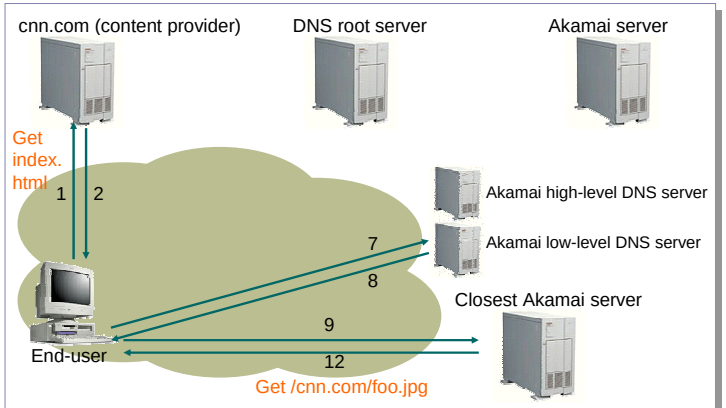


- Smoothness → addition of bucket does not cause much movement between existing buckets
- Spread & Load → small set of buckets that lie near object
- Balance → no bucket is responsible for large number of objects

How Akamai Works



Akamai – Subsequent Requests



Next Lecture: DNS, P2P & P2P Apps

- Readings
  - [JBSM01] DNS Performance and the Effectiveness of Caching
  - [BLR04] A Layered Naming Architecture for the Internet
  - [S+01] Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications
  - [G+03] The Impact of DHT Routing Geometry on Resilience and Proximity

How to Mark End of HTTP Message?

- Size of message → Content-Length
  - Must know size of transfer in advance
- Delimiter → MIME-style Content-Type
  - Server must “escape” delimiter in content
- Close connection
  - Only server can do this

Persistent Connection Solution

- Multiplex multiple transfers onto one TCP connection
  - Serialize transfers → client makes next request only after previous response
- How to demultiplex requests/responses
  - Content-length and delimiter → same problems as before
  - Block-based transmission – send in multiple length delimited blocks
  - Store-and-forward – wait for entire response and then use content-length
  - PM95 solution – use existing methods and close connection otherwise

Cookies: Keeping “state”

Many major Web sites use cookies

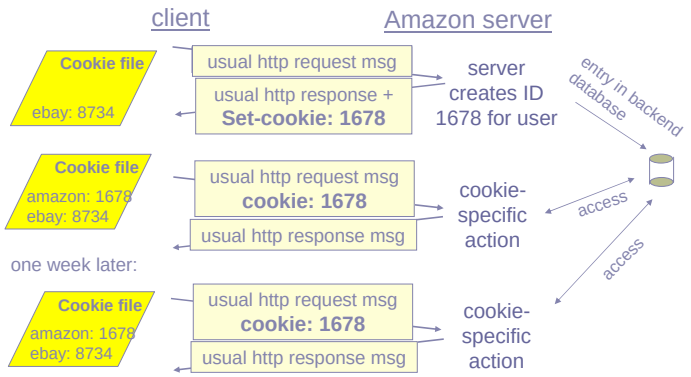
Four components:

1) Cookie header line in the HTTP response message
2) Cookie header line in HTTP request message
3) Cookie file kept on user’s host and 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.)



Web Caching - advantages

- Also used for security
  - Proxy is only host that can access Internet
  - Administrators make sure that it is secure
- Performance
  - How many clients can a single proxy handle?
- Caching
  - Provides a centralized coordination point to share information across clients
- How to index
  - Early caches used file system to find file
  - Metadata now kept in memory on most caches

Obscure Caching Advantages

- Connection caching [Feldmann 1999]
  - HTTP – small objects, overhead in setting up connection
  - Multiplex multiple requests over single persistent HTTP connection
  - Proxy maintains persistent HTTP connections to clients and servers
- Split TCP connection
  - TCP throughput increases as RTT decreases

Cache consistency - 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, etc
- Proxies make leases scalable

Proxies – cache misses

- Capacity
  - How large a cache is necessary or equivalent to infinite
  - On disk vs. in memory → typically on disk
- Compulsory
  - First time access to document (large caches)
  - Non-cacheable documents
    - CGI-scripts
    - Personalized documents (cookies, etc)
    - Encrypted data (SSL)
- Consistency
  - Document has been updated/expired before reuse
- Conflict → no such issue

ICP vs HTTP

- Why not just use HTTP to query other caches?
- ICP is lightweight – positive and negative
  - Makes it easy to process quickly
  - HTTP has many functions that are not supported by ICP
  - Extra RTT (2 sec) for any proxy-proxy transfer
  - Does not scale to large number of peers

Hierarchy Problems – Population Size

- How does population size affect hit rate?
- Critical to understand usefulness of hierarchy or placement of caches
- Issues: frequency of access vs. frequency of change (ignore working set size → infinite cache)
- UW/Msoft measurement → hit rate rises quickly to about 5000 people and very slowly beyond that
- Proxies/Hierarchies don't make much sense for populations > 5000
  - Single proxies can easily handle such populations
  - Hierarchies only make sense for policy/administrative reasons

Problems – Common Interests

- Do different communities have different interests?
  - I.e. do CS and English majors access same pages? IBM and Pepsi workers?
- Has some impact → UW departments have about 5% higher hit rate than randomly chosen UW groups
  - Many common interests remain
- Is this true in general? UW students have more in common than IBM & Pepsi workers
- Some related observations
  - Geographic caching – server traces have shown that there is geographic locality to interest
  - UW & MS hierarchy performance is bad – could be due to size or interests?

Other Caching Problems

- Aborted transfers
  - Many proxies transfer entire document even though client has stopped → eliminates saving of bandwidth
- Client misconfiguration
  - Many clients have either absurdly small caches or no cache
- Session –
  - HTTP: stateless
  - Not much interesting things can be done
  - Sessions needed for e-commerce