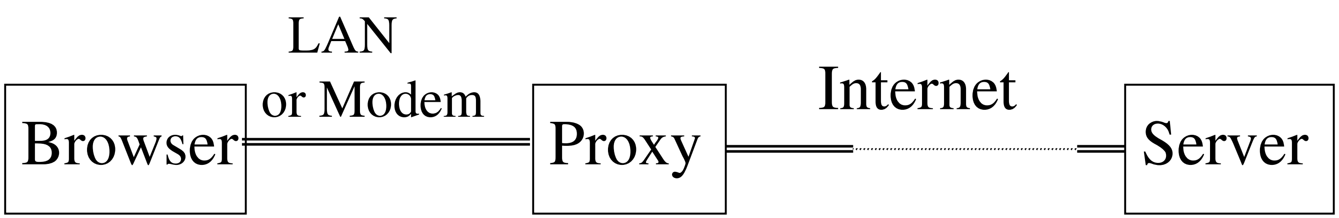
Web Caching: Scalability, Functionality and Performance Issues

Pei Cao
Computer Sciences Department
University of Wisconsin-Madison

Why Web Caching

- Part of WWW's job is to disseminate information; Caching is natural



  - Reduce client latency
  - Reduce Internet traffic
  - Reduce server load

Protocol Support for Caching

- General headers: “cache-control” and “pragma”
- Request headers: “if-modified-since”, “if-matched”, “if-not-matched”, etc.
- Response headers: “expires”, “last-modified”

Success of Proxy Caching

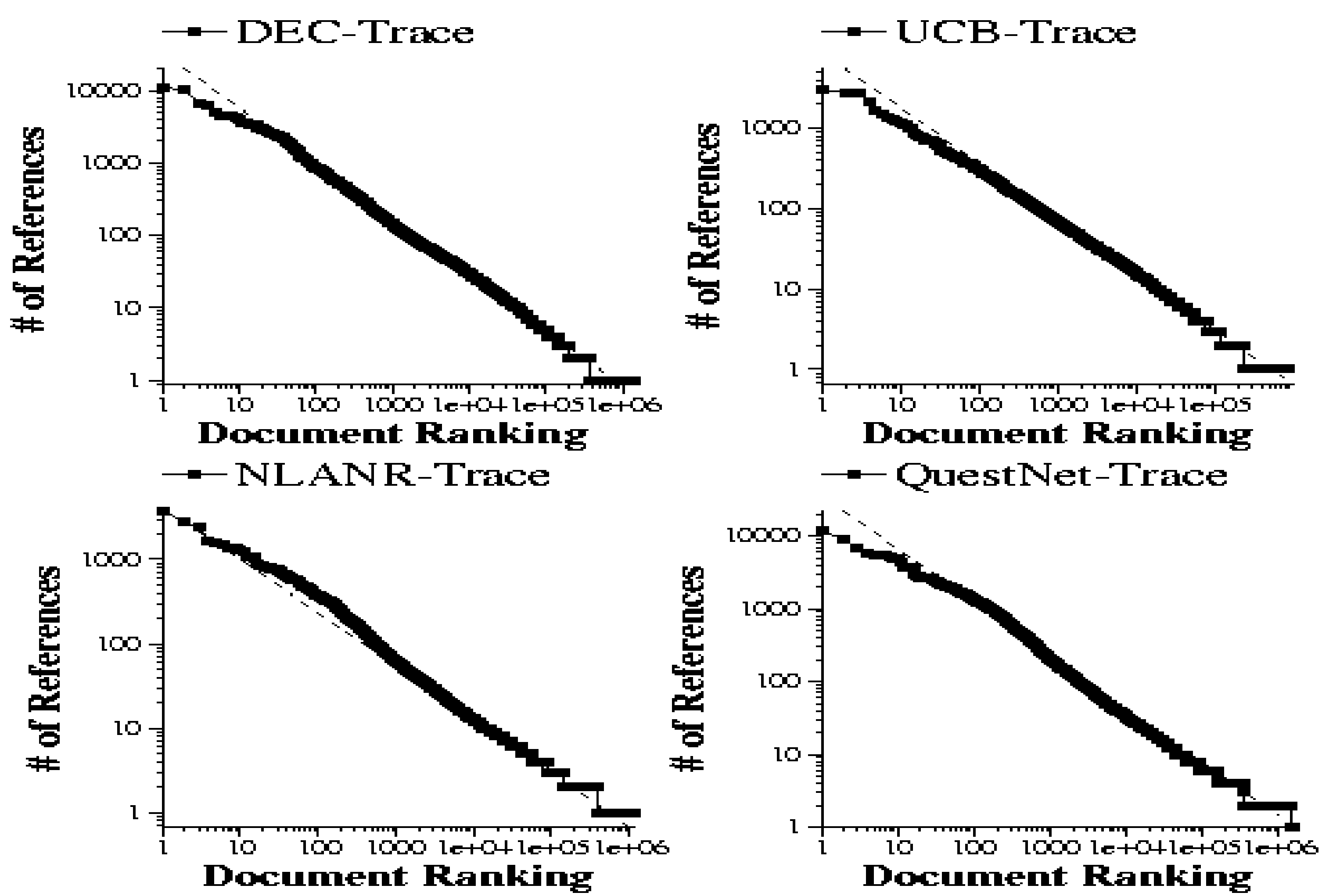
- Caching Web proxies are widely deployed
  - Main reason: cuts down monthly Internet access cost
- Typical hit ratios
  - Cache size: 8GB ~ 22GB (use disks)
  - Object hit ratio: 30% ~ 60%
  - Traffic reduction: 20% ~ 45%
- Several products and vendors
  - Squid

Challenges

- Increase cache hit ratio beyond 50%
- Dynamic documents are on the rise
- Reduce overhead for consistency management
- Performance of proxies

Characteristics of Web Accesses Seen by the Proxy

- Excellent temporal locality
  - The probability that a document will be referenced k requests since the last reference is approximately proportional to $1/k$
- In a homogenous user community, as more users are put through a cache, the cache's hit ratio grows approximately proportional to log(#-of-users)


DEC-Trace
# of References
10000
1000
100
10
1
1
10
100
1000
1e+04
1e+05
1e+06
Document Ranking
UCB-Trace
# of References
1000
100
10
1
1
10
100
1000
1e+04
1e+05
Document Ranking
NLANR-Trace
# of References
1000
100
10
1
1
10
100
1000
1e+04
1e+05
1e+06
Document Ranking
QuestNet-Trace
# of References
10000
1000
100
10
1
1
10
100
1000
1e+04
1e+05
1e+06
Document Ranking

Zipf-Like Distribution

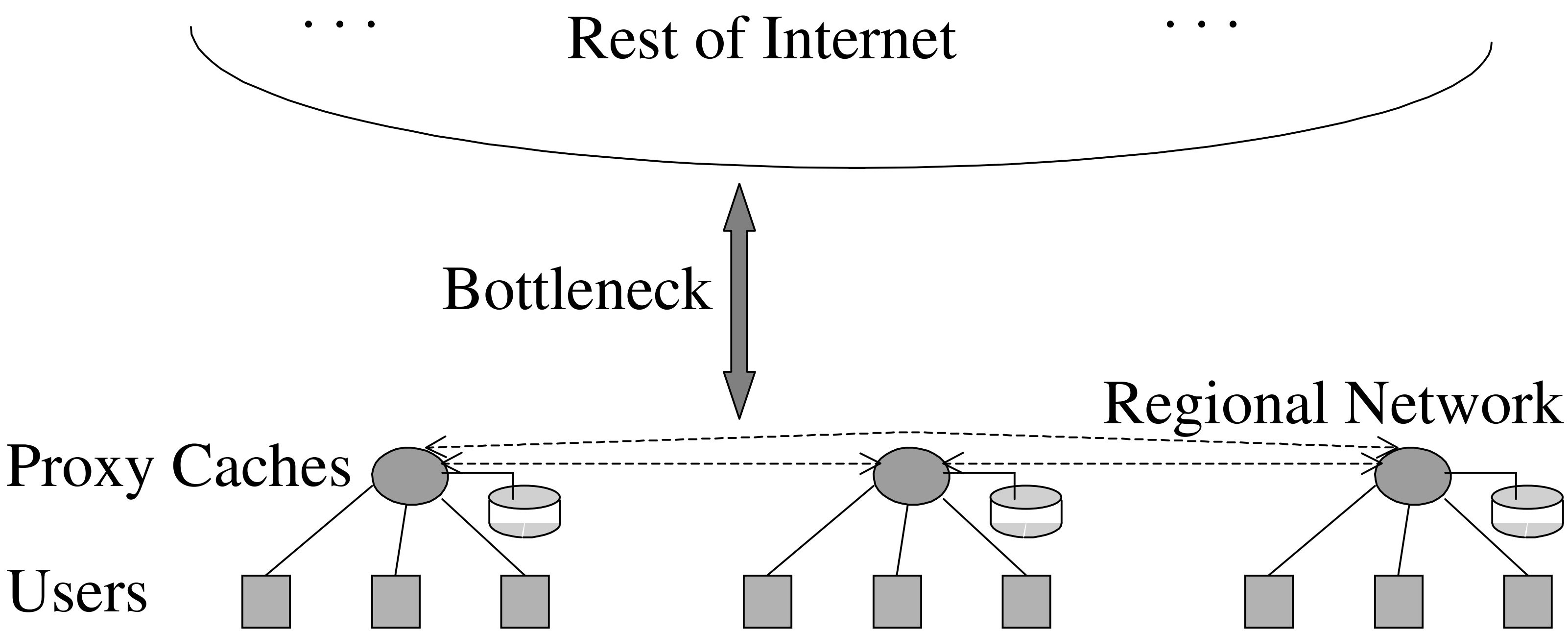
- Probability of reference P(k) ~ $1/k^{a}$
- a ranges from 0.5 to 1
  - DEC: 0.77, UCB: 0.69, Questnet: 0.73, NLANR: 0.64

⇨ A large body of "lukewarm" documents
  - top 1% accounts for 20-35% of accesses
  - top 10% accounts for 45-55% of accesses
  - top 40% accounts for 70-80% of acccesses

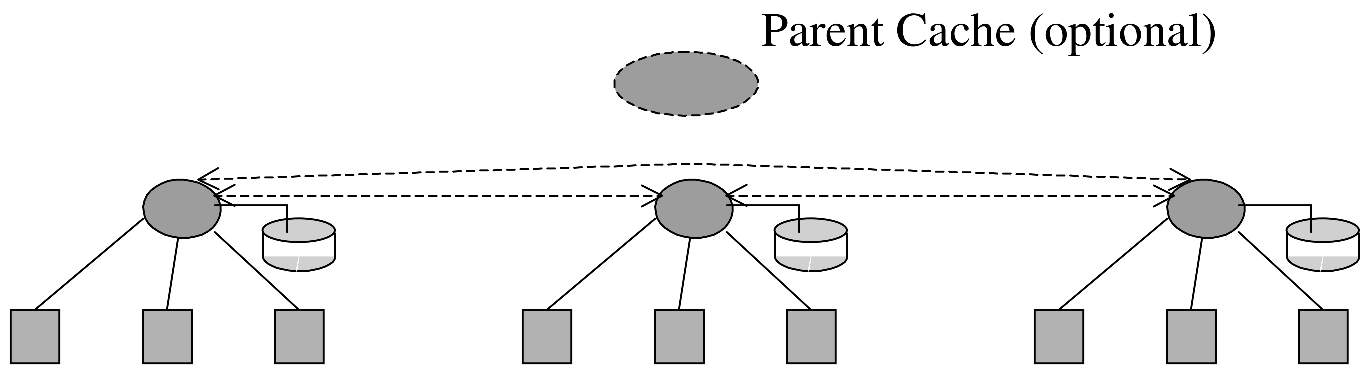
⇨ Hit ratio goes up as # of requests increases

Summary Cache: A Scalable Wide-Area Cache Sharing Protocol

Cache Sharing



Cache Sharing via ICP



– When one proxy has a cache miss, send queries to all siblings (and parents): “do you have the URL?”
– Whoever responds first with “Yes”, send a request to fetch the file
– If no “Yes” response within certain time limit, send request to Web server

Overhead of ICP

- #_of_queries = (#_of_proxies * average_miss_ratio) * #_of_proxies
- Experiments
  - **4** Squid proxies running on dual-processor 64MB SPARC20s linked with 100BaseT Ethernet links
  - Workloads: traces and synthetic benchmarks
- Compared with no cache sharing, ICP:
  - increases total network packets to each proxy by 8-29%
  - increases CPU overhead by 13-32%
  - increases user latency by 2-12%

Alternatives to ICP

- Force all users to go through the same cache or the same array of caches
  - Difficult in a wide-area environment
- Central directory server
  - Directory server can be a bottleneck

Ideally, one wants a protocol:

- keeps the total cache hit ratio high
- minimizes inter-proxy traffic
- scales to a large number of proxies

Summary Cache

- Basic idea:
  - Let each proxy keep a directory of what URLs are cached in every other proxy, and use the directory as a filter to reduce number of queries
- Problem 1: keeping the directory up to date
  - Solution: delay and batch the updates => directory can be slightly out of date
- Problem 2: DRAM requirement
  - Solution: compress the directory => imprecise, but inclusive directory

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

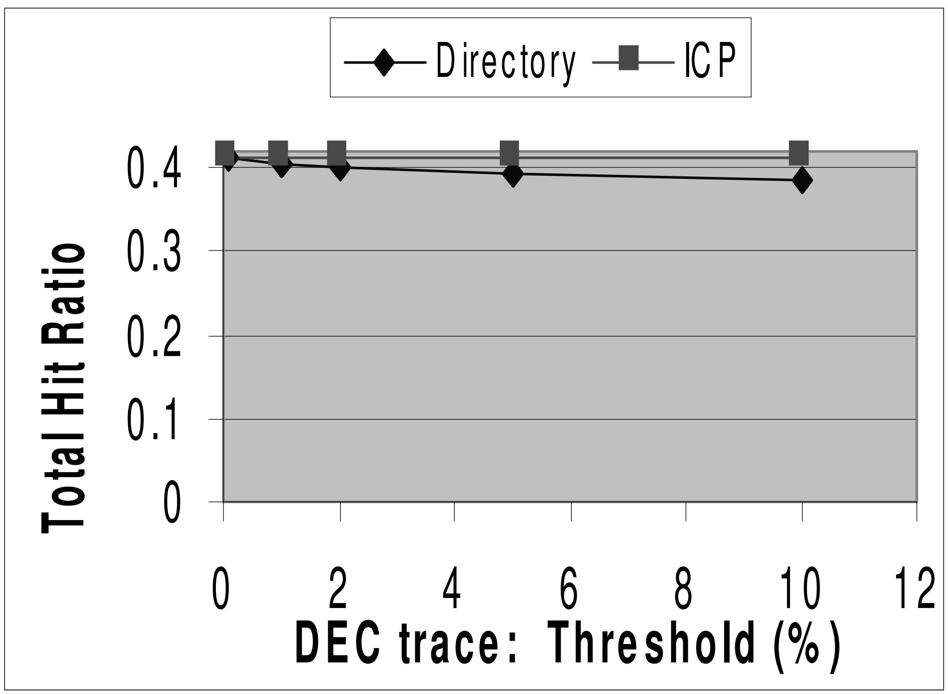
    Effect: lower total cache hit ratio
  - false hits: r is not cached at B, but A thought it is

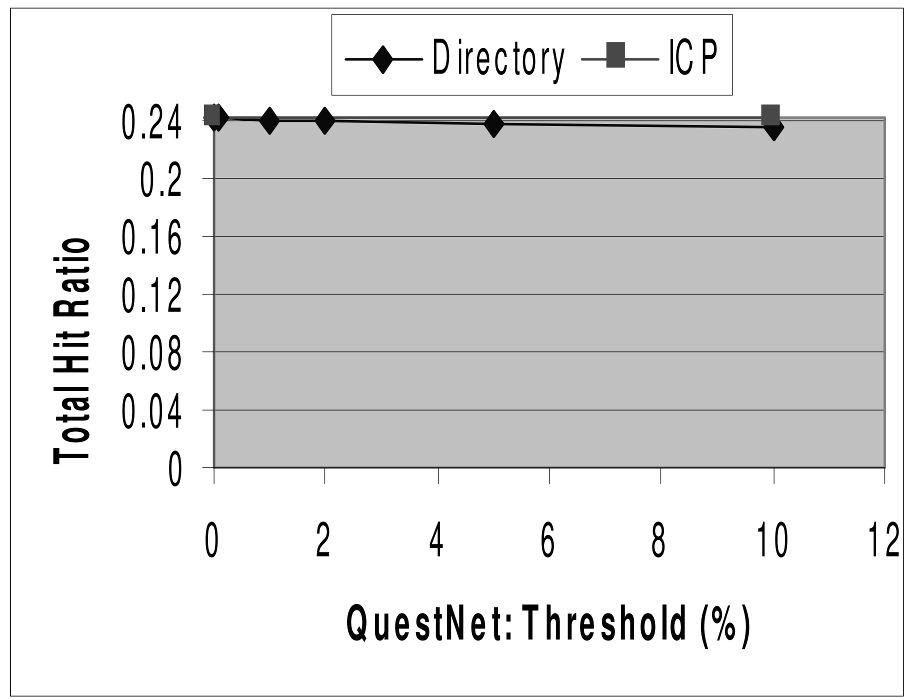
    Effect: wasted query messages
  - stale hits: r is cached at B, but B's copy is stale

    Effect: wasted query messages

Effect of Delay in Directory Updates

- Method: delay the updates until a certain percentage of the cached documents are "new"



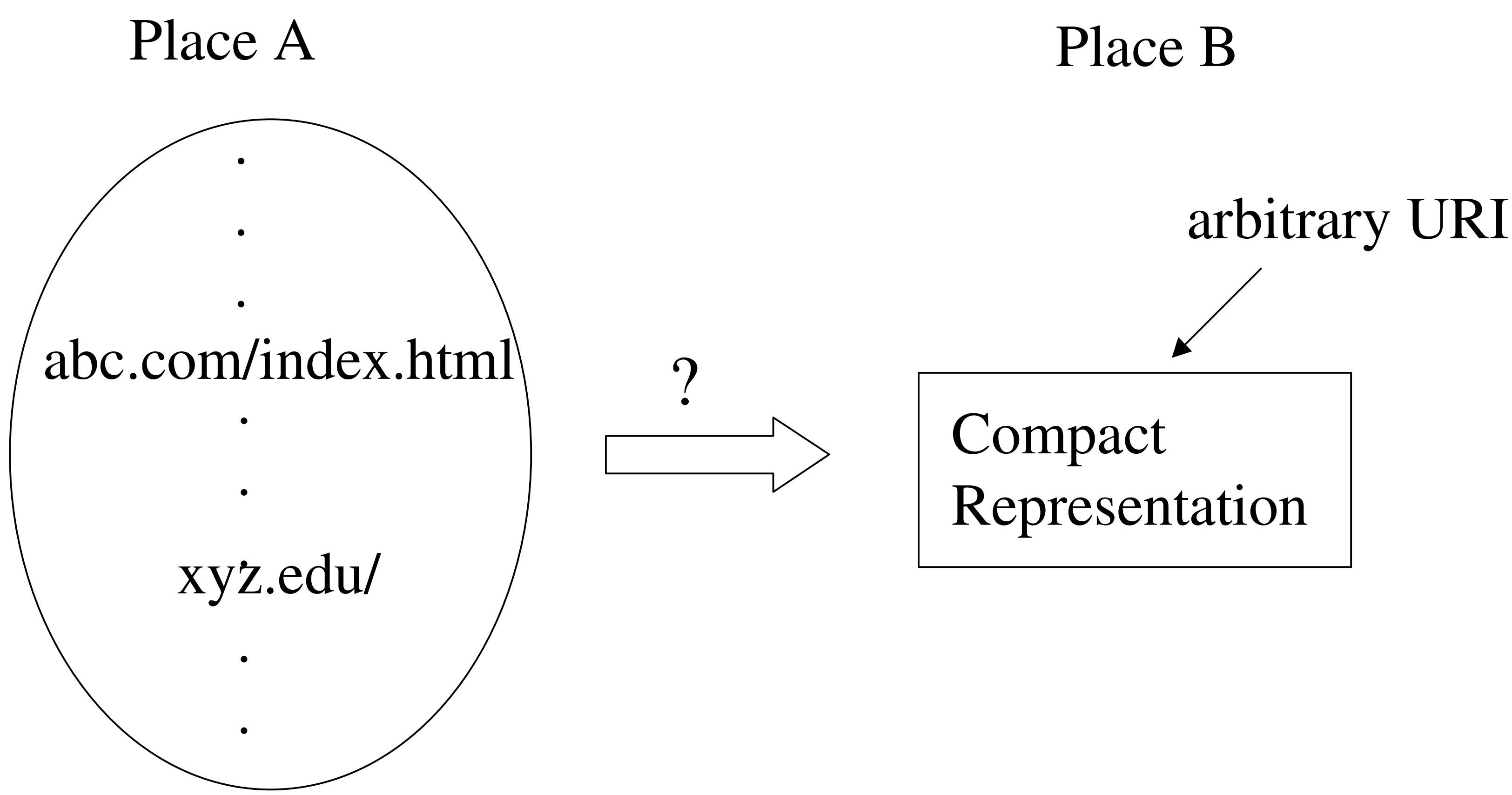


Compressing the Directories

- Requirements:
  - Inclusive
  - Low false positives
  - Concise

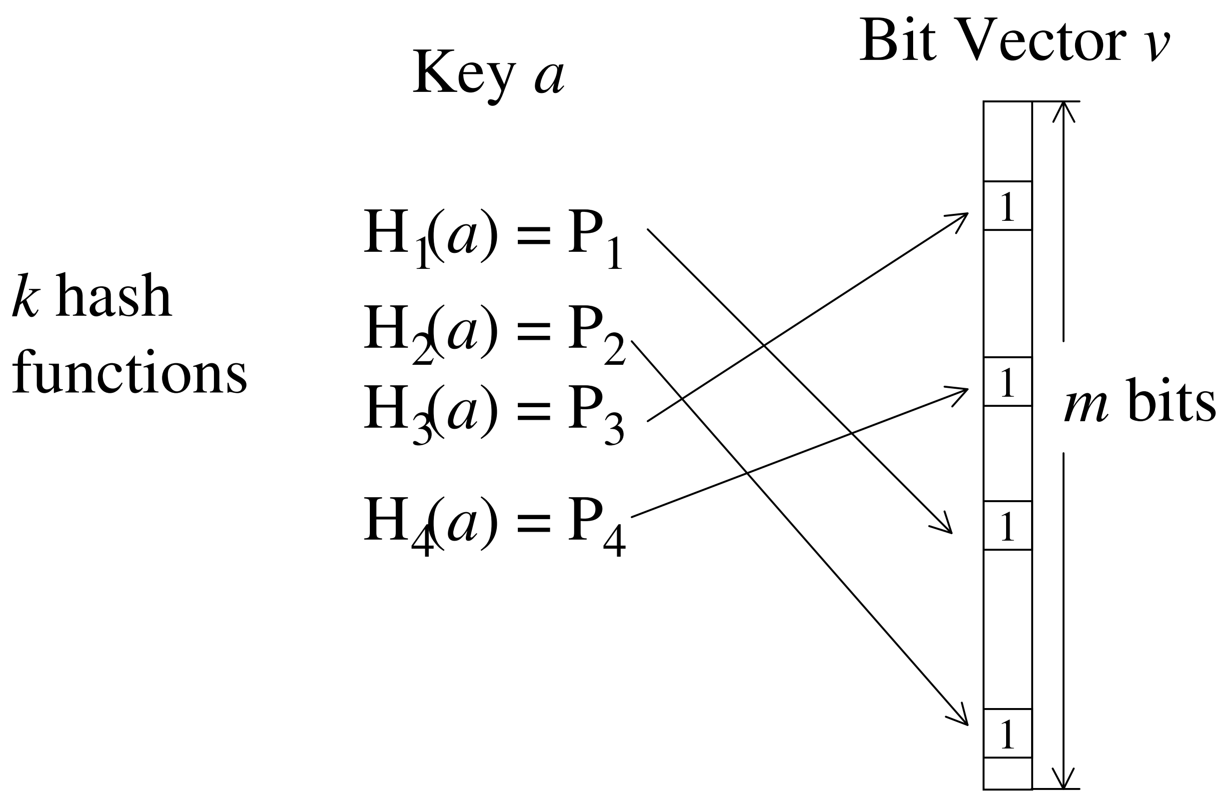
  we call the compressed directories "summaries"
- First try: use server URLs only
  - Problem: too many false hits, leading to too many messages between proxies

The Problem


Place A
Place B
arbitrary URI
abc.com/index.html
xyz.edu/
?
Compact
Representation

Bloom Filters

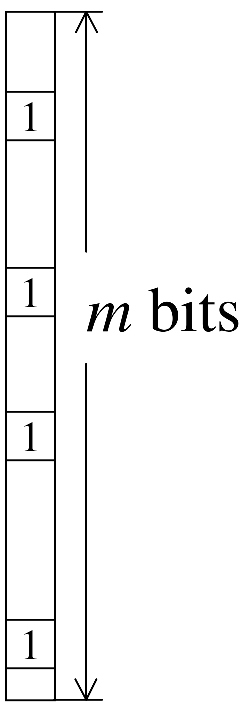
- Support membership test for a set of keys



Bloom Filters: the Math

- Given n keys, how to choose m and k?

Bit Vector v



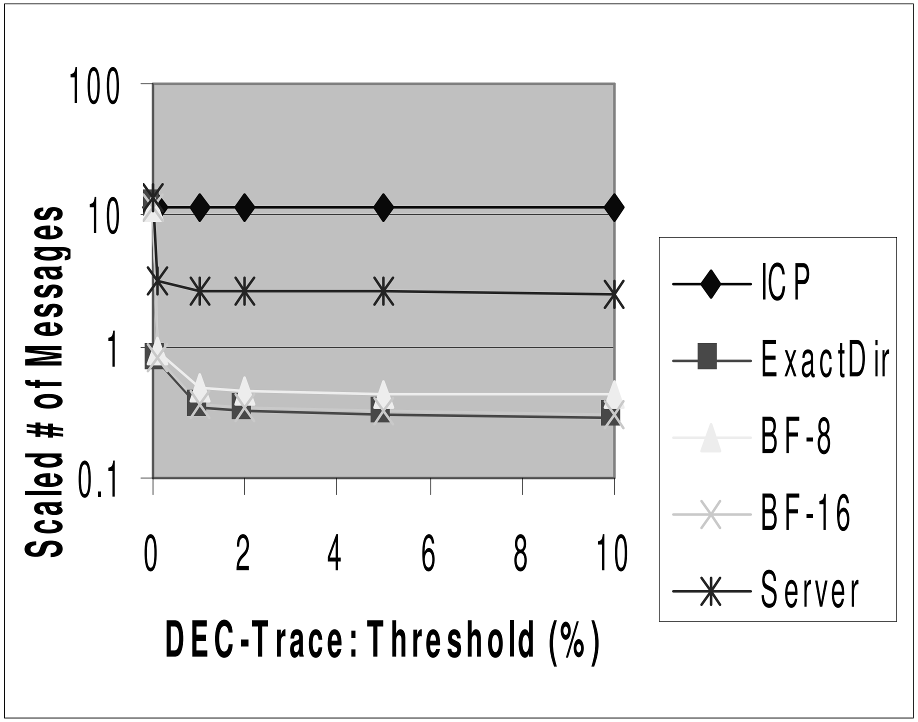
- Suppose m is fixed (>2n), choose k:
  k is optimal when exactly half of the bits are 0
  => optimal $k = \ln(2) * m/n$

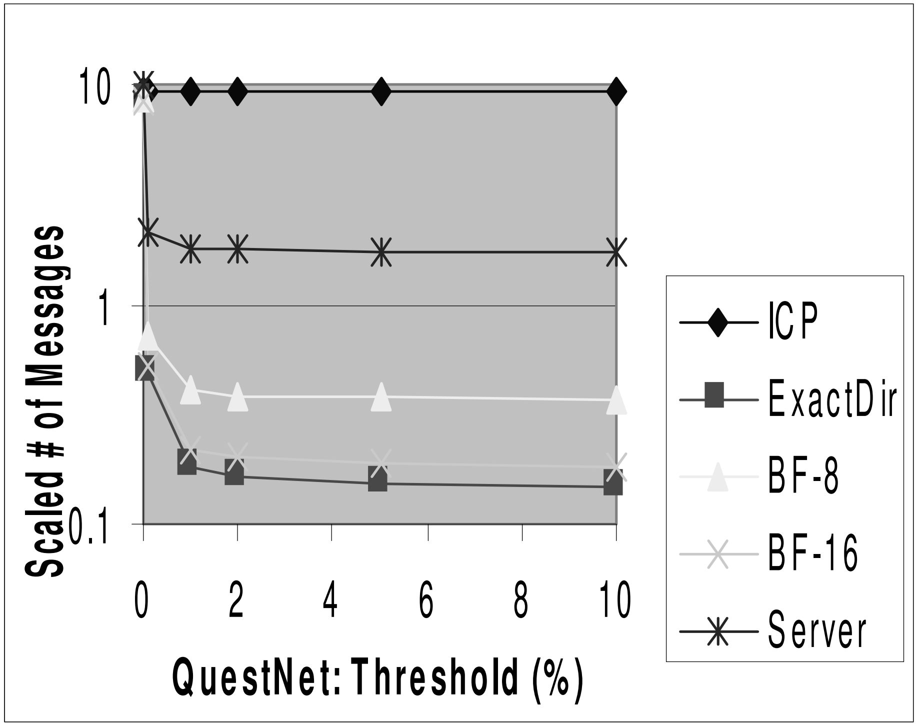
- False positive ratio under optimal k is $(1/2)^k$
  => false positive ratio = $(1/2)^{\ln 2 * m/n} = (0.62)^{m/n}$

Bloom Filters: the Practice

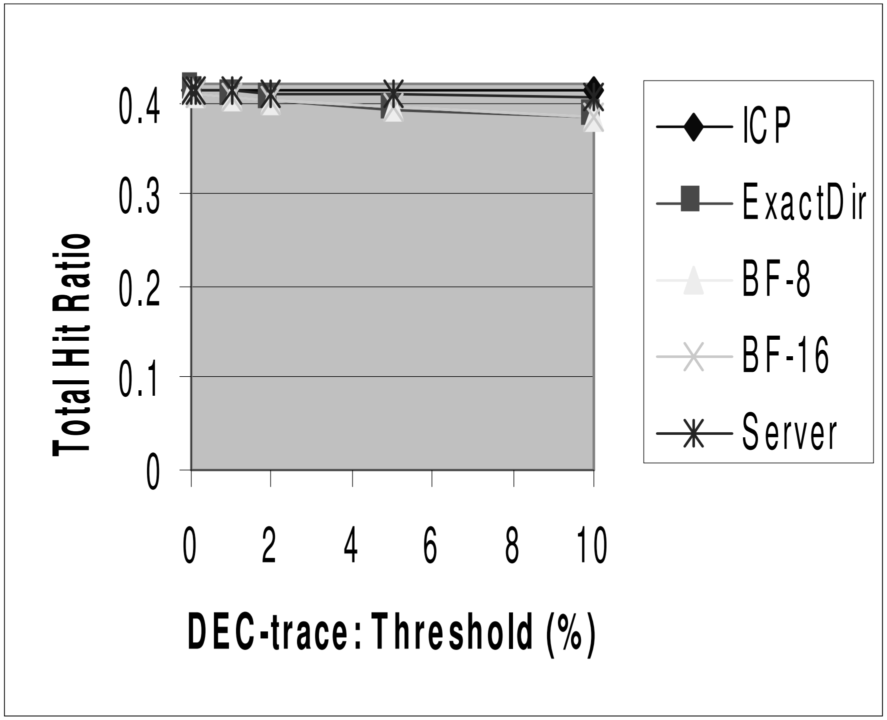
- Choosing hash functions
  - bits from MD5 signatures of URLs
- Maintaining the summary
  - the proxy maintains an array of counters
  - for each bit, the counter records how many times the bit is set to 1
- Updating the summary
  - either the whole bit array or the positions of changed bits (delta encoding)

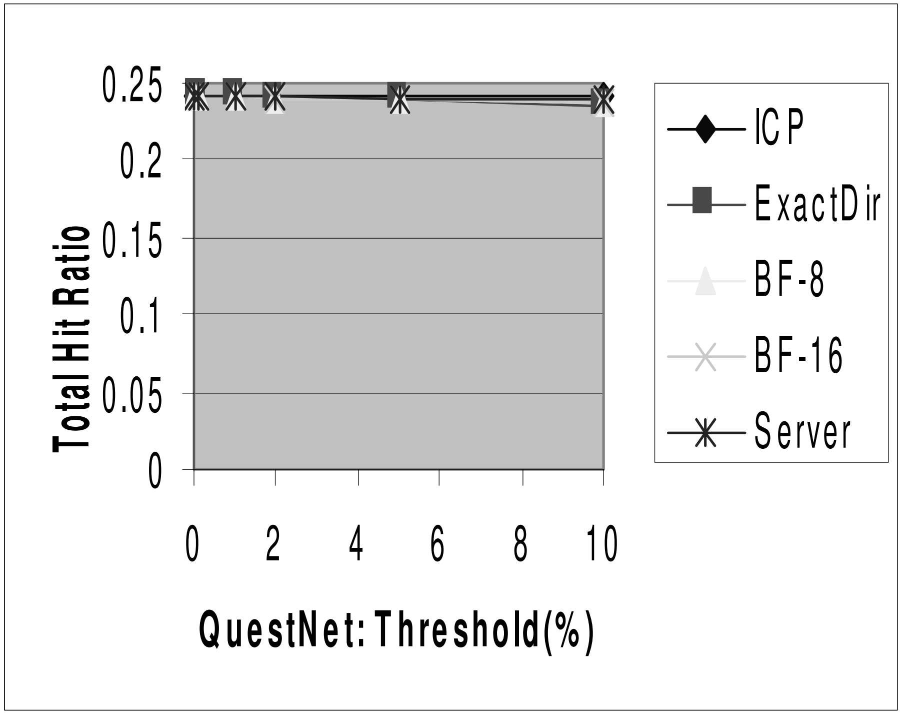
Result: Inter-Proxy Traffic





Result: Total Hit Ratio





Enhancing ICP with Summary Cache

- Prototype implemented in Squid 1.1.14
- Repeating the 4-proxy experiments, the new ICP:
  - Reduces UDP messages by a factor of 12 to 50 compared with the old ICP
  - Little increase in network packets over no cache sharing
  - increase CPU time by 2 - 7%
  - reduce user latency up to 4% with remote cache hits

Sum it up...

- Summary cache enables caches to share contents with low overheads over wide area
- An alternative implementation called "Cache Digest" is in Squid 1.2.0
- Many other applications of bloom filters
- Technical report version available at:

  http://www.cs.wisc.edu/~cao/papers/summary-cache/

Active Cache: A Framework for Caching Dynamic Contents

Motivations

- Web servers desire to track accesses to its contents and customize contents for each user
  - 1/3 of requests carry cookies
  - 23% of requests carry '?' (queries)
  - Popular pages carry rotating ad banners
  - Many customize pages via cgi-bin
- Web caches "hide" accesses from servers

⇨ Current practice: cache busting

Solution: “Active” Caching

- Web pages as objects!
  - Each page is content plus code (provided by Web server)

Cache applet: mobile code that is

- attached to a Web page
- executed at a caching proxy
- for the purpose of processing a cached copy to service a user request
- written in Java for practical reasons

Basic Protocol

Web server: marks a URL as having an associated cache applet

Caching proxy: promises that upon every user request to the URL

- either the request is sent to the Web server
- or the cache applet is invoked and executed successfully

=> resource management flexibility at proxies

Facilities Provided by the Proxy

- Invocation of cache applet upon user request of the URL
- Storage of objects created and used by the applet
  - the objects encapsulate states carried from one invocation of the applet to the next
  - the special “log” object encapsulates information to be returned to the Web server
- Information on which URLs are cached
- Network communication with the Web server

Applet-Proxy Interface

- Server attaches an applet to a URL through an new entity header in the response header
  - "CacheApplet: codebase = base_dir, applet = app"
- Each applet must implement
  - the method FromCache(HTTP_Request, ClientAddr, Cached_File, Result_File)
  - which returns:
    - 0: meaning "return to client cached_file"
    - 1: meaning "return to client Result_file"
    - -1: meaning "send the request to Web server"

Example: Logging User Accesses

- logapplet: logging user accesses to the URL
  - open the log object: file named “logapplet.log”
  - lock the file
  - append to the file: current time, client name and IP address, and the HTTP request from the user
  - unlock the file
  - close the file
  - return 0

Example: Client Specific Web Pages

- csnd: client-specific news distribution
  - find the preference of the client
    - either directly from the cookie
    - or from looking up the cookie in a database object obtained from the server
  - find the data items; fetch the missing data items from the server
  - assemble the new Web page
  - return 1

Security Policy

- What applets can do
  - query for the existence of files
  - create, read/write files
  - communicate with the original server
- Security model: each Web server is a security entity
  - applets from the same Web server have read access to any information belonging to the server
  - each applet only has write accesses (including lock access) to its own objects or the Web page that it is attached to
  - applets from different servers cannot collaborate
- Threats: denial of service, illegal access to information

Security Mechanisms

- Rely on Java security, but harden it with a set of external constraints
  - ActiveProxy class implements all proxy methods; Java run-time system enforces that all I/O operations are through ActiveProxy
  - **Absence** of thread/process packages, Java.io packages and other utilities
  - **Static inspection** of applet code: making sure that the applet does not call any method that manipulates the class hierarchy

Resource Accounting

- For each applet invocation, keep track of:
  - CPU time consumed
  - virtual memory consumed
  - bytes written to and read from files
  - number of files (objects) created
  - bytes sent/received over the network
  - total storage size devoted to the applet
- Put limits on each category of resources

Prototype Implementation

- Modifications of CERN httpd
  - CERN httpd forks a new process for each user request
  - If the entity header of the cached response indicates an applet, invoke the applet
    - Each applet’s invocation is in a separate process
    - Implication: resource accounting is easier, but passing states among invocations has to use files
  - ActiveProxy class is implemented as native methods

Performance of the Prototype

- Platform: Webstone 2.0, running Solaris 2.6 on x86, using Sun's JDK 1.1.5 (no JITs)
  - Ratio of response time applet/no applet

Proxy	1 Client	10 Clients	20 Clients
null	1.47	1.75	1.73
log	2.16	2.31	2.24
ads	1.4	2.44	3.23
apc	2.5	2.73	2.72
ssi	3.81	4.00	3.8
cswp*	1.06	2.04	2.23

* running on Sparc 20

  - Performance problems in existing applets: string operations and file reads

Limitations of the Current Design

- No transactional guarantee or session semantics at the proxy
  - Tradeoff between the desire to offload the Web server and the desire to keep the proxy autonomous
- No real-time guarantee at the proxy
- No support yet for servers to query the state of applet invocations at the proxy
  - "log" object is a minimal support

Summary

- Zipf-like distribution of Web accesses
- Summary Cache
  - Bloom filters
- Active Cache
- More info:

http://www.cs.wisc.edu/~cao/wisweb.html