# Aliasing on the WWW
## Prevalence and Performance Implications


Terence Kelly
U. Michigan EECS
Ann Arbor, MI

Jeffrey C. Mogul
Compaq WRL
Palo Alto, CA



11th International World Wide Web Conference
Honolulu, Hawaii 9 May 2002



$Id: aliasing_slides.tex,v 1.8 2002/04/24 07:33:13 tpkelly Exp $

# What Is Web Aliasing?

Aliasing: *multiple names for the same thing*

Aliasing in the Web:

- “Things” of interest: HTTP reply payloads
- Static view: two URLs “point to” same payload
- Dynamic view: two *transactions*, with different URLs, have same reply payload

# Motivation to study Web aliasing

- Aliasing increases cache miss rates
  - At both proxies and clients
  - Causes redundant data transfers
- Previous crawler-based (static) studies:
  - Broder *et al.* similarity study: 18–41% of reachable payloads are aliased
  - Shivakumar & Garcia-Molina: 36%

# Goals of our research

Look at *dynamic* prevalence of Web aliasing:

- How much aliasing in *transactions*?
  - # of payloads aliased
  - # of transactions w/ aliased payloads
  - # of aliased bytes transferred
- Look for correlations with other attributes
- Measure redundant transfers in conventional cache hierarchies
- How can we eliminate redundant transfers?

# Outline of talk

- Motivation
- Terms and example
- Methodology and traces
- Prevalence of aliasing
- Correlates of aliasing
- Performance implications
- Solutions

# Terms

**Payload:** "Entity body" of HTTP reply

**Aliased payload:** Accessed via $\geq 2$ URLs

- I.e., payloads are bit-for-bit identical

**Payload Hit:** Payload comes from cache

- Note: "304 Not Modified" *is* a payload hit

**Payload Miss:** Must fetch payload via network

# Example reference stream

| | URL | Payload | Reason for cache miss |
|---|---|---|---|
| *1* | A | 1 | new payload |
| *2* | A | 2 | new payload |
| *3* | A | 1 | resource A is modified |
| *4* | B | 1 | payload 1 is aliased |

- In conventional Web cache, *all* are misses
- Transfers #3 and #4 are *redundant*
- Aliasing not sole cause of redundant xfers

# Methodology

- Analyze real users' accesses
  traces include anonymized
  - URLs
  - payload digests (using MD5)
- Simulate behavior of:
  - browser/proxy cache hierarchy
  - cacheless & infinite-cache browser
- Tabulate redundant payload transfers

# Anonymized Traces

- All traces made at *non-caching* proxies
  - So: all payloads came from origin server
- WebTV trace:
  - Cache-busting proxy (no client caching!)
  - Sept. 2000
- Compaq trace:
  - Clients did use caching
  - Jan–Mar 1999

# Trace characteristics

| | WebTV | Compaq |
|---|---|---|
| Days | 16 | 90 |
| Clients | 37 K | 22 K |
| URLs | 32 M | 20 M |
| Payloads | 36 M | 31 M |
| Transactions | 326 M | 79 M |
| Working Set | 596 GB | 501 GB |
| Bytes transferred | 1,838 GB | 841 GB |

*Among the largest and most detailed traces used in Web-related research.*

# Prevalence of Aliasing

WebTV: aliased payloads account for

- 5% of unique payloads
- 54% of transactions
- 36% of bytes transferred

Only 10% of transactions involve modified resources.

*Aliasing is more prevalent than resource modification by several measures.*

# Correlates of Aliasing

- Aliased payloads are smaller:

| | **Median unique payload** | **Median transfer** |
|---|---|---|
| non-aliased | 5.5 KB | 2.5 KB |
| aliased | 3.1 KB | 1.3 KB |

- GIF both popular & heavily aliased

  *45% of transfers carry aliased GIFs!*

- Are Web authoring tools to blame?

# Content Naming & Caching

Conventional caches:

- Indexed by URL
- Store (at most) one payload per URL

But: $(URL, payload)$ binding in traces not 1:1

So: cache *could* see redundant xfers due to

- Aliasing: $\geq 2$ URLs bind to 1 payload
- Modification: 1 URL binds to $\geq 2$ payloads
  - Redundant if payloads are $(1, ..., 2, ..., 1)$
  - e.g., ad rotation

# Performance Implications

- *What price do we pay?*
- Simulate URL-indexed browser/proxy cache hierarchy
  - Payload miss rate
  - % redundant transfers
- Do not model redundant transfers due to faulty metadata, silly cache management.

# Payload Miss Rates

| | Payload miss rate (%) | % redundant xfers |
|---|---|---|
| WebTV ∞-cache clients | 29.5 | 9.8 |
| WebTV proxy (warm) | | |
| cacheless clients | 12.9 | 23.1 |
| ∞-cache clients | 46.3 | 22.8 |
| Compaq proxy (warm) | 44.9 | 18.5 |

*Client cache size has little effect on % redundant at proxy!*

# Causes of Redundant Transfers

Our results consider interplay between URL-indexed caching & content naming (aliasing, resource modification)

Other causes can include:

- Finite caches (capacity misses)
- Silly caches: e.g., evict-upon-expire
- Silly metadata: e.g., changing Etags

# Eliminating Redundant Transfers

"Duplicate Transfer Detection" (DTD)

- Cache retains old payloads indefinitely
- Index cache also by *payload digest*
- Server sends digest *before* payload
- Cache looks for entry w/ same digest
- Don't transfer payload if already cached

*Never receive same payload twice.*

Devil is in the details (details are future work!)

# Other Possible Solutions

Educate site designers/implementors:

- 1:1 URL-payload mapping where possible (CDNs do this already)
- Eliminate within-site aliasing

If Web authoring tools are to blame:

- Serve "clip art" images from one site/CDN
- Bundle clip art with browsers

# Summary

- Aliasing happens:
  54% of transfers carry aliased payloads
- Redundant transfers happen:
  10% at browser, 22% at proxy
- Avoidable causes include:
  - Content-naming practices
  - combined with URL-indexed caching
- Comprehensive solution: DTD

# Credits

- Traces: WebTV Networks, Compaq
- Computers: Compaq, U-M & UCSD supercomputer centers
- Web mystery explainer: Mikhail Mikhailov

# Backup slides

# Conventional & DTD Caches

URL-indexed cache

```
if cache[URL] == correct payload
    payload_hit++
else
    payload_miss++
    send URL
    receive payload
    cache[URL] := payload
```

DTD cache

```
if u_cache[URL] == correct payload
    payload_hit++
else
    send URL
    receive payload digest
    if d_cache[digest] == correct payload
        payload_hit++
        send “don’t bother”
    else
        payload_miss++
        send “proceed”
        receive payload
        d_cache[digest] := payload
        u_cache[URL] := payload
```

# Details: Duplicate Transfer Detection

- Hop-by-hop HTTP extension
- Cache every payload forever
- Index cache using *payload digest*
- Before receiving payload, check cache using digest from sender

*Note: No special treatment for "dynamic" content. A payload is a payload.*

# DTD Implementation I: “Proceed” Model

- Server sends payload digest only
- Client says “proceed” if not in cache
- No redundant bytes ever sent
- Extra RTT for payload misses

# DTD Implementation II: “Abort” Model

- Server sends digest + full payload
- Client says “abort” if cached
- No additional client latency
- Some redundant bytes sent