Using Signatures to Improve URL Routing

Zornitza Genova  
Kenneth J. Christensen  
Department of Computer Science and Engineering  
University of South Florida  
Tampa, Florida 33620  
{zgenova, christen}@csee.usf.edu

This material is based upon work funded by the National Science Foundation under grant no. 9875177

Topics

- Introduction - distributed content in the Web
- Description of URL routers
- Digesting of URL lists
- Evaluation of digesting
- Summary and future work
**Introduction**

- Content is replicated and distributed in the Web
  - Same file can be found in many content sources in the Web
    - Origin server, temporary servers, caches, etc.

- Replication and distribution improves user response time by
  - Reducing server loads
  - Reducing network loads
  - Reducing distance content must travel

- Content Distribution Networks (CDNs) are commercially supported
  - For example, Akamai

---

**Introduction continued**

- Content Distribution Network

![Diagram of Content Distribution Network]

Same content found in multiple places.
Mechanisms for “URL routing”
- DNS redirection [4]
- TCP connection spoofing or splicing [13]
- HTTP redirection

Each mechanism has its trade-offs
- DNS redirection has poor load balancing properties
- TCP connection splicing/spoofing is resource intensive
- HTTP redirection requires a double round-trip delay and uses CPU resources of a content source

Introduction continued

Key challenge is...

To determine which content source is the “best” for a given URL request from a given client.

A CDN must be able to forward requests (to the best source)
- A “URL router” front ends content sources
- Or, is built in to the content source

Distributed caches forward missed requests
- Microsoft CARP for proxy caches
- SQUID ICP for transparent caches
Description of a URL router

• Operation of a URL router (DNS redirection not shown)...

1) Establishes TCP connection with client.

2) Receives and parses HTTP request from client.

3) Looks-up the requested URL in a routing table and determines the hostname of the best source.

4) Spoofs or splices the connection with the source if the best source is local to URL router.

5) Sends the client an HTTP redirection message containing new URL of source if the best source is remote to URL router.

Description of a URL router continued

• Operation of a URL router...

1 = Spoofing or splicing
2 = HTTP redirection

HTTP request to origin server site

Reverse cache

Origin site

Distributed server

Clients

1 = URL router
**Description of a URL router continued**

- Structure of a URL routing table...

<table>
<thead>
<tr>
<th>URL 1</th>
<th>Loc 1 (state), loc 2 (state), ... loc M₁ (state)</th>
</tr>
</thead>
<tbody>
<tr>
<td>URL 2</td>
<td>Loc 1 (state), loc 2 (state), ... loc M₂ (state)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>URL N</td>
<td>Loc 1 (state), loc 2 (state), ... loc Mₙ (state)</td>
</tr>
</tbody>
</table>

Select best source based on state (and location of client)

Routing tables must be updated/shared frequently

Our problem!

**Digesting of URL lists**

- URL routing tables are very large

- Typical content source (cache or server) has...
  - Thousands to millions of files
  - Each file represented by a URL of about 50 bytes length

- Want to reduce the size of these URL lists
  - To more efficiently transfer across the Internet
  - To reduce routing look-up effort

A compressed URL list is a **URL digest**
**Digesting of URL lists continued**

- Bloom filters used in Summary Cache [7]
  - Hash a URL string into 128 bits or four 32-bit values
  - Modulo the four 32-bit values by size of filter (array)
  - Resulting four values are indexes into array

- Bloom filter is probabilistic
  - False positives if non-unique hashes (but no false miss)
  - Results in a "routing collision" in the context of URLs

- MD5 is CPU intensive to compute
- "Goodness" of MD5 not needed when reduced for Bloom filter
- Bloom filter may be difficult to share and update

**Digesting of URL lists continued**

- Incremental CRC19 digests used in Adaptive Web Caching [15]
  - Compute CRC19 for parts of URL string (separated by "/")
  - Build a tree of CRC19 values
- This method is also probabilistic

- Requires software computation of CRC
- Results in large digests
**Digesting of URL lists continued**

- **"False hits"** can occur in two ways

  1) Due to a non-unique signature in digest
     - (Mis)route to a source which does not contain object
  2) Due to object having been removed in source
     - For example, a cache age-out

- False hits self-correct for the case of caches
  - Cache will request missed item and have it for future

---

**Digesting of URL lists continued**

- **Our idea...**
  - Use **CRC32** for URL signatures

- **CRC32 circuitry is already part of an Ethernet adapter**
  - Serial shift-register with wrapped XOR terms
  - A CRC32 is a modulo-2 division remainder

- Need to calculate a CRC32 over a subfield
  - The subfield is the URL in an HTTP request
    - Let the subfield by \( M \) bits in length

- Wish to use the same existing circuit for,
  - Usual IEEE 802.3 Frame Check Sequence (FCS)
  - URL signatures of HTTP requests
Digesting of URL lists continued

- Define the following,
  - \( P \) is CRC32 generator polynomial
  - \( A_i, i = 1, ..., m \) is a polynomial (bit sequence)

<table>
<thead>
<tr>
<th>Packet header</th>
<th>Subfield</th>
<th>Rest of packet</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A_0 )</td>
<td>( A_2 )</td>
<td>( A_1 )</td>
</tr>
</tbody>
</table>

- We store in a table (for all possible \( M \)) the remainders...

\[
R_M = \text{Rem}\left(\frac{A^M}{P}\right)
\]

Digesting of URL lists continued

- We have the following,

\[
R_A = \text{Rem}\left(\frac{A}{P}\right)
\]

Returned by adapter
- from CRC32 shift register

\[
R_A = \text{Rem}\left(\frac{A}{P}\right)
\]

What we want (CRC32 for subfield)
**Digesting of URL lists continued**

- For

\[ R_A = \text{Rem} \left( \frac{A}{p} \right) \]

we have the following properties

\[
\begin{align*}
\text{Rem} \left( \sum_{i=0}^{m} A_i \cdot 2^i \right) & = \text{Rem} \left( \sum_{i=0}^{m} R_A \cdot 2^i \right) \\
\text{Rem} \left( \prod_{i=0}^{m} A_i \right) & = \text{Rem} \left( \prod_{i=0}^{m} R_A \right)
\end{align*}
\]

**Digesting of URL lists continued**

- We solve for \( R_{A_2} \) as follows...

Let \( A_3 \) be \( A_0 \) shifted left \( M \) bits

Then

\[
R_A = \text{Rem} \left( \frac{A_3 \cdot 2^m}{p} \right) = \text{Rem} \left( \frac{R_{A_A} \cdot R_A}{p} \right)
\]

\[
R_A = \text{Rem} \left( \frac{A_3 - A_1}{p} \right) = \text{Rem} \left( \frac{R_{A_A} \cdot R_A}{p} \right)
\]

32-bit multiply
### Digesting of URL lists continued

- Probability of a false hit using CRC32 signatures
  - $\kappa = 32$ for CRC32

\[
\Pr[\text{collision in a set of } N \text{ URLs}] = 1 - \prod_{i=1}^{N} \frac{\left(2^{\kappa} - i + 1\right)}{2^{\kappa}}
\]

\[
\Pr[\text{collision for a given URL}] = 1 - \left(\frac{2^{\kappa} - 1}{2^{\kappa}}\right)^{N-1}
\]

\[
E[\text{number of collisions}] = N \left(1 - \left(\frac{2^{\kappa} - 1}{2^{\kappa}}\right)^{N-1}\right)
\]

---

### Evaluation of digesting

- Obtain URL lists from caches and servers
- We evaluate...
  1) Reduction in size of URL list
  2) Probability of a false hit
  3) CPU resources required to generate digest
  4) Reduction in CPU resources for signature look-up in hash table
Evaluation of digesting continued

- URL list stored as full URL string or as CRC32 signatures
  - Stored in a chained hash table
- We study the relationship of hash table size to look-up time
  - For both full URL and CRC32 signatures

Testbed was 866-Mhz Dell Pentium-3, 128M RAM, Win2K

Evaluation of digesting continued

- Summary statistics for URL lists

<table>
<thead>
<tr>
<th>Access list name</th>
<th>Number of accesses</th>
<th>Number of URLs</th>
<th>Mean URL length</th>
<th>URL list size (full URL)</th>
<th>URL list size (CRC32)</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.peak.org">www.peak.org</a> [18]</td>
<td>16,374</td>
<td>70</td>
<td>23.93 bytes</td>
<td>1,675 bytes</td>
<td>280 bytes</td>
</tr>
<tr>
<td>SDMA [22]</td>
<td>41,941</td>
<td>153</td>
<td>33.76</td>
<td>5,165</td>
<td>612</td>
</tr>
<tr>
<td>UVA [25]</td>
<td>318,899</td>
<td>45,816</td>
<td>44.91</td>
<td>2,057,625</td>
<td>183,264</td>
</tr>
<tr>
<td>NLANR [17]</td>
<td>944,028</td>
<td>504,967</td>
<td>58.44</td>
<td>29,510,135</td>
<td>2,019,868</td>
</tr>
<tr>
<td>UC Berkeley [16]</td>
<td>1,791,349</td>
<td>149,344</td>
<td>41.87</td>
<td>6,253,716</td>
<td>597,376</td>
</tr>
<tr>
<td>mcs.net [14]</td>
<td>1,862,070</td>
<td>75,361</td>
<td>29.87</td>
<td>2,250,829</td>
<td>301,444</td>
</tr>
<tr>
<td>hyperreal.org [3]</td>
<td>4,080,590</td>
<td>86,338</td>
<td>89.17</td>
<td>7,698,337</td>
<td>345,352</td>
</tr>
<tr>
<td>CA*netII [23]</td>
<td>4,642,861</td>
<td>2,552,045</td>
<td>57.83</td>
<td>147,573,556</td>
<td>10,208,184</td>
</tr>
<tr>
<td>USF CSEE [6]</td>
<td>8,819,454</td>
<td>49,029</td>
<td>51.84</td>
<td>2,541,483</td>
<td>196,116</td>
</tr>
</tbody>
</table>
Evaluation of digesting continued

• CRC32 URL digest collision probability

<table>
<thead>
<tr>
<th>Access list name</th>
<th>Collisions measured</th>
<th>Expected value (Eq.3)</th>
<th>$Pr[collision]$ measured</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.peak.org">www.peak.org</a></td>
<td>0</td>
<td>0</td>
<td>0.0000000</td>
</tr>
<tr>
<td>SDMA</td>
<td>0</td>
<td>0</td>
<td>0.0000000</td>
</tr>
<tr>
<td>UVA</td>
<td>0</td>
<td>1</td>
<td>0.0000000</td>
</tr>
<tr>
<td>NLANR</td>
<td>68</td>
<td>59</td>
<td>0.0001347</td>
</tr>
<tr>
<td>UC Berkeley</td>
<td>2</td>
<td>5</td>
<td>0.0000134</td>
</tr>
<tr>
<td>mcs.net</td>
<td>0</td>
<td>1</td>
<td>0.0000000</td>
</tr>
<tr>
<td>hyperreal.org</td>
<td>2</td>
<td>2</td>
<td>0.0000463</td>
</tr>
<tr>
<td>CA*netII</td>
<td>1558</td>
<td>1516</td>
<td>0.0006105</td>
</tr>
<tr>
<td>USF CSEE</td>
<td>2</td>
<td>1</td>
<td>0.0000408</td>
</tr>
</tbody>
</table>

Measured and expected are very close.

Evaluation of digesting continued

• CRC32 URL digest CPU time

<table>
<thead>
<tr>
<th>Access list name</th>
<th>Time for URL list</th>
<th>Time per URL</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.peak.org">www.peak.org</a></td>
<td>&lt;10 msec</td>
<td>-</td>
</tr>
<tr>
<td>SDMA</td>
<td>&lt;10</td>
<td>-</td>
</tr>
<tr>
<td>UVA</td>
<td>40</td>
<td>0.8730 μsec</td>
</tr>
<tr>
<td>NLANR</td>
<td>460</td>
<td>0.9109</td>
</tr>
<tr>
<td>UC Berkeley</td>
<td>100</td>
<td>0.6695</td>
</tr>
<tr>
<td>mcs.net</td>
<td>40</td>
<td>0.5307</td>
</tr>
<tr>
<td>hyperreal.org</td>
<td>120</td>
<td>1.3897</td>
</tr>
<tr>
<td>CA*netII</td>
<td>2390</td>
<td>0.9368</td>
</tr>
<tr>
<td>USF CSEE</td>
<td>40</td>
<td>0.8158</td>
</tr>
</tbody>
</table>

10ms precision for CPU time
Evaluation of digesting continued

• Hash table size look-up time for 50K look-ups

![Graph showing look-up time for Full and Compressed (CRC32) URL](image)

6x reduction for $H=12$

Evaluation of digesting continued

• TCP connection overhead is an issue

• TCP connection must be established before HTTP request sent

• Connections per second capability is much less than look-up
  - Future work to address this
Summary and future work

- URL routers are needed to automatically distribute load
  - Between contents servers in a cluster
  - Between content courses "Internet wide"

- Routing tables are large in size – can be "compressed"
  - We proposed CRC32 signatures to replace URLs

- Showed that CRC32 digests reduce
  - URL list size
  - Look-up time in a URL router

- Showed that CRC32 signatures can be computed w/ existing circuit

- Showed that collisions (false hits) are very rare