

# Efficient Summarization of URLs using CRC32 for Implementing URL Switching

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

## Abstract

We investigate methods of using CRC32 for compressing Web URL strings and sharing of URL lists between servers, caches, and URL switches. Using trace-based evaluation, we compare our new CRC32 digesting method against existing Bloom filter and incremental CRC19 methods. Our CRC32 method requires less CPU resources, generates equal or smaller size digests, achieves equal collision rates, and simplifies switching.

## 1. Introduction and background

Globally distributed Web sites allow for identical content to be contained in multiple servers and caches throughout the Internet (Figure 1). In such systems, URL switching is used to forward HTTP requests. Sharing of URL lists enables routing of requests between the cache sites (e.g., [2] and [4]) and building of URL routing tables. To reduce network bandwidth and router table size, compressing of URL lists into digests is of interest. False hits, or routing collisions, will occur when URL lists contain expired entries or the same entry for different content or when a non-existing URL is requested.

In summary cache [2] each URL in a list is hashed into a 128-bit value using an MD5 signature [7], which is then partitioned into four 32-bit quantities further reduced (by modulo division) to become indexes into a Bloom filter [1] digest. A vector of  $m$  bits is initialized to zero and then individual bits are set based on hashes (with a range of 1 to  $m$ ) of set members. Bloom filters are probabilistic, false hits result when unique members have overlapping non-unique hashes. Four indexes are computed for each URL. Each entry in the Bloom filter is a 4-bit counter where the counter is incremented when a URL entry is added and decremented when it is removed. The number of bits in the Bloom filter is the “load factor” and is 8, 16, or 32 times the number of entries in the URL list. MD5 guarantees unique hashes at the expense of being very processor (or hardware) intensive [7].

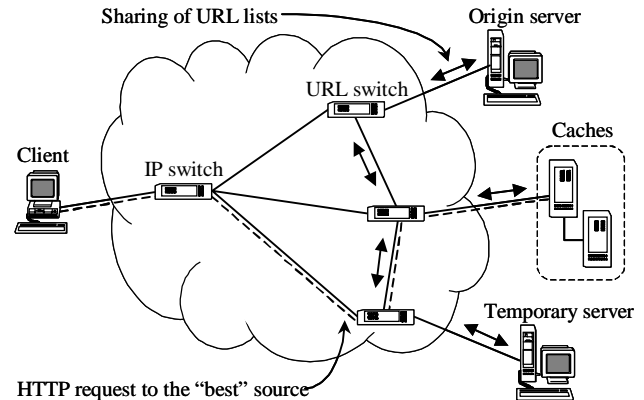


Figure 1. Globally distributed Web infrastructure

We investigate the effectiveness of CRC32 as a method for digesting and compare it to the Bloom filter method of [2] and the incremental CRC19 digest of [4].

## 2. Trace-driven digesting methods evaluation

All experiments were executed on an 866-Mhz Dell Dimension Pentium III PC (running Windows 2000 and using the Borland 5.02 “C” compiler and x86 assembler). The input was a set of URL lists to be compressed into a digest. These lists were obtained from the access logs: CA\*net squid proxy log (list #1) [6], a log from the USF Computer Science and Engineering server (list #2), an NLANR proxy log (list #3) [5], and a server log from Virginia Tech (list #4) [8]. URL lists #3 and #4 are the same as used in Summary cache [2]. See Table 1. The last column is the mean number of parts in a URL.

Table 1. Summary statistics for access logs

	# URLs	Size	URL len	URL parts
List #1	2,552,045	140.75 MB	56.8 B	6
List #2	49,029	1.54	28.8	7
List #3	483,631	16.30	56.0	6
List #4	45,817	1.96	43.9	6

**Table 2. Results for experiments #1 and #2 (CPU time in seconds, size in Mbytes, and collisions as a percentage)**

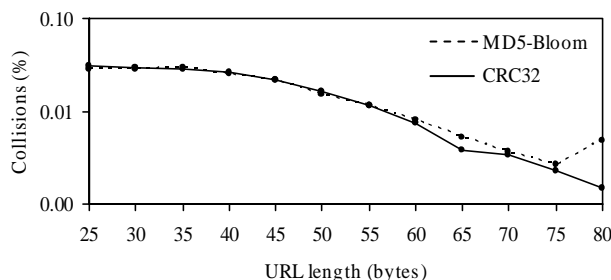
Method (L. Factor)	URL list #1			URL list #2			URL list #3			URL list #4		
	CPU	Size	Coll	CPU	Size	Coll	CPU	Size	Coll	CPU	Size	Coll
MD5-Bloom (8)	89.13	9.74	0.03	1.63	0.19	0.00	16.25	1.84	0.01	1.52	0.17	0.00
CRC32	16.22	9.74	0.03	0.27	0.19	0.00	2.51	1.84	0.01	0.20	0.17	0.00
32-bit checksum	14.85	9.74	0.71	0.24	0.19	0.22	2.29	1.84	0.30	0.20	0.17	0.08
LZ compression	17.35	16.43	0.00	0.23	0.25	0.00	3.32	4.00	0.00	0.27	0.31	0.00
MD5-Bloom (8)	89.13	9.74	0.03	1.63	0.19	0.00	16.25	1.84	0.01	1.52	0.17	0.00
MD5-Bloom (16)	92.37	19.47	0.00	1.71	0.37	0.00	16.99	3.69	0.00	1.59	0.35	0.00
MD5-Bloom (32)	97.40	38.94	0.00	1.84	0.75	0.00	18.20	7.38	0.00	1.70	0.70	0.00

*Experiment #1 – Full comparison.* For the Bloom filter digests, a load factor of 8 was used (smallest load factor in [2]). Table 2 shows results. The CRC32 digest requires about a factor of 6 less CPU time and with no increase in collisions! LZ compression requires less CPU time than Bloom filter and slightly more time than the CRC32 method, but results in larger digest sizes. The 32-bit checksum yields only slightly better CPU time results than CRC32 at an expense of higher collision rates.

*Experiment #2 – MD5-Bloom varying load factor [2].* The last three rows of Table 2 show the results for MD5-Bloom digesting with varying load factor.

*Experiment #3 – Effects of URL length.* A comparison of MD5-Bloom and CRC32 for collision rate as a function of URL length. Only URLs of greater than 25, 30, ..., 80 bytes and load factor 8 were used. Figure 2 shows results for list #1, the other URL lists exhibit a similar trend. As the URL length increases, the collision rate decreases.

*Experiment #4 – Digest size of hash chain method [4].* The digests of the four URL lists were larger than all other methods from experiment #1 with the exceptions of Bloom filter with load factor 32 for all URL lists and LZ and Bloom with load factor 16 for URL list #2. We did not implement the full hash chain tree structure and computed the digest size based on the number of components in a URL and a shared tree structure (with 32 bits to represent a <depth, hash code> pair as proposed in [4]). Hash chaining produces an average of 212% larger digests than CRC32. CRC computation in software is linear with the CRC length, hence the hash chain method will require greater CPU time than the CRC32 method.

**Figure 2. Results from experiment #3**

*Experiment #5 – Direct hash-based look-up.* Multiple digests can be merged into one look-up table. Insertion and deletion of entries is of the same complexity as a switching look-up. CRC32 is already well randomized and a subset of the 32-bits per URL can be used as hash indexes. Table 3 shows the results for URL list #1 (with 2,552,045 entries).

**Table 3. Results for experiment #5 (hash chain length)**

# bits	Unique	2	3	4	5 or more
20	223,768	272,424	220,696	134,523	105,115
24	2,191,565	166,850	8,524	298	4
28	2,528,110	11,902	44	0	0
32	2,550,488	779	0	0	0

### 3. Summary and future work

Compared to other methods CRC32 digesting requires less CPU resources to generate the digest, produces equivalent collision rates, and simplifies switching decisions. Collision-free hashing methods are not sufficient to prevent false hits – more study is needed.

### References

- [1] B. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, Vol. 13, No. 7, pp. 422-426, July 1970.
- [2] L. Fan, P. Cao, J. Almeida, and A. Broder, "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol," *Proceedings of ACM SIGCOMM*, pp. 254-265, 1998.
- [3] Z. Genova and K. Christensen, "Using Signatures to Improve URL Routing," *Proceedings of IEEE International Performance, Computing, and Communications Conference*, April 2002.
- [4] B. Michel, K. Nikoloudakis, P. Reiher, and L. Zhang, "URL Forwarding and Compression in Adaptive Web Caching," *Proceedings of IEEE INFOCOM*, pp. 670-678, March 2000.
- [5] NLANR Sanitized Cache Access Logs (National Science Foundation Grants NCR-9616602 and NCR-9521745), 2000. URL: <ftp://ircache.nlanr.net/Traces/>.
- [6] Sanitized Log Files from the CA\*NetII, 2000. URL: <http://ardnoc41.canet2.net/cache/squid/rawlogs/>
- [7] J. Touch, "Performance Analysis of MD5," *Proceedings of ACM SIGCOMM*, pp. 77-86, September 1995.
- [8] Virginia Tech's Proxy Traces, 2000. URL: <http://www.cs.wisc.edu/~cao/icache/proxytrace.html>.