Network Working Group   Glenn Fowler
INTERNET-DRAFT   AT&T Labs Research
Intended Status: Informational   Landon Curt Noll
                                                       Cisco Systems
                                                      Kiem-Phong Vo
                                                   AT&T Labs Research
                                                     Donald Eastlake
                                                  Huawei Technologies
Expires: October 5, 2013   April 6, 2013

The FNV Non-Cryptographic Hash Algorithm
<draft-eastlake-fnv-05.txt>

Abstract

   FNV (Fowler/Noll/Vo) is a fast, non-cryptographic hash algorithm with
   good dispersion. The purpose of this document is to make information
   on FNV and open source code performing FNV conveniently available to
   the Internet community.

Status of This Memo

Table of Contents

[1](#). **Introduction**

   The FNV hash algorithm is based on an idea sent as reviewer comments
   to the [[IEEE](#)] POSIX P1003.2 committee by Glenn Fowler and Phong Vo in
   1991. In a subsequent ballot round Landon Curt Noll suggested an
   improvement on their algorithm. Some people tried this hash and found
   that it worked rather well. In an EMail message to Landon, they named
   it the "Fowler/Noll/Vo" or FNV hash. [[FNV](#)]

   FNV hashes are designed to be fast while maintaining a low collision
   rate. The high dispersion of the FNV hashes makes them well suited
   for hashing nearly identical strings such as URLs, hostnames,
   filenames, text, IP addresses, etc. Their speed allows one to quickly
   hash lots of data while maintaining a reasonably low collision rate.
   However, they are generally not suitable for cryptographic use. (See
   [Section 6.1](#).)

   The FNV hash is widely used, for example in DNS servers, the Twitter
   service, database indexing hashes, major web search / indexing
   engines, netnews history file Message-ID lookup functions, anti-spam
   filters, a spellchecker programmed in Ada 95, flatassembler's open
   source x86 assembler - user-defined symbol hashtree, non-
   cryptographic file fingerprints, computing Unique IDs in DASM (DTN
   Applications for Symbian Mobile-phones), Microsoft's hash_map
   implementation for VC++ 2005, the realpath cache in PHP 5.x
   (php-5.2.3/TSRM/tsrm_virtual_cwd.c), and many other uses.

   A study has recommended FNV in connetion with the IPv6 Flow Label
   field [[IPv6flow](#)].

   FNV hash algorithms and source code have been released into the
   public domain. The authors of the FNV algorithm took deliberate steps
   to disclose the algorithm in a public forum soon after it was
   invented. More than a year passed after this public disclosure and
   the authors deliberately took no steps to patent the FNV algorithm.
   Therefore, it is safe to say that the FNV authors have no patent
   claims on the FNV algorithm as published.

   If you use an FNV function in an application, you are kindly
   requested to send an EMail about it to: fnv-mail@asthe.com

2. FNV Basics

This document focuses on the FNV-1a function whose pseudo-code is as
follows:

```
hash = offset_basis
for each octet_of_data to be hashed
        hash = hash xor octet_of_data
        hash = hash * FNV_Prime
return hash
```

In the pseudo-code above, hash is a power-of-two number of bits (32,
64, ... 1024) and offset_basis and FNV_Prime depend on the size of
hash.

The FNV-1 algorithm is the same, including the values of offset_basis
and FNV_Prime, except that the order of the two lines with the "xor"
and multiply operations are reversed. Operational experience
indicates better hash dispersion for small amounts of data with
FNV-1a. FNV-0 is the same as FNV-1 but with offset_basis set to zero.
FNV-1a is suggested for general use.

2.1 FNV Primes

The theory behind FNV_Prime's is beyond the scope of this document
but the basic property to look for is how an FNV_Prime would impact
dispersion. Now, consider any n-bit FNV hash where n is >= 32 and
also a power of 2. For each such an n-bit FNV hash, an FNV_Prime p is
defined as:

When s is an integer and $4 < s < 11$, then FNV_Prime is the
smallest prime p of the form:

$$256**int((5 + 2^s)/12) + 2**8 + b$$

where b is an integer such that:

$$0 < b < 2**8$$
The number of one-bits in b is 4 or 5

and where p mod $(2**40 - 2**24 - 1) > (2**24 + 2**8 + 2**7)$.

Experimentally, FNV_Primes matching the above constraints tend to
have better dispersion properties. They improve the polynomial
feedback characteristic when an FNV_Prime multiplies an intermediate
hash value. As such, the hash values produced are more scattered
throughout the n-bit hash space.

The case where s < 5 is not considered because the resulting hash quality is too low. Such small hashes can, if desired, be derived from a 32 bit FNV hash by XOR folding (see [Section 3](#)). The case where s > 10 is not considered because of the doubtful utility of such large FNV hashes and because the criteria for such large FNV_Primes is more complex, due to the sparsity of such large primes, and would needlessly clutter the criteria given above.

Per the above constraints, an FNV_Prime should have only 6 or 7 one-bits in it. Therefore, some compilers may seek to improve the performance of a multiplication with an FNV_Prime by replacing the multiplication with shifts and adds. However, note that the performance of this substitution is highly hardware-dependent and should be done with care. FNV_Primes were selected primarily for the quality of resulting hash function, not for compiler optimization.

[2.2](#) FNV offset_basis

The offset_basis values for the n-bit FNV-1a algorithms are computed by applying the n-bit FNV-0 algorithm to the 32 octets representing the following character string in [[ASCII](#)]:

        chongo <Landon Curt Noll> /\../\

The \'s in the above string are not C-style escape characters. In C-string notation, these 32 octets are:

        "chongo <Landon Curt Noll> /\\../\\"

[2.3](#) FNV Endianism

For persistent storage or interoperability between different hardware platforms, an FNV hash shall be represented in the little endian format. That is, the FNV hash will be stored in an array hash[N] with N bytes such that its integer value can be retrieved as follows:

```
unsigned char   hash[N];
for ( i = N-1, value = 0; i >= 0; --i )
    value = value << 8 + hash[i];
```

Of course, when FNV hashes are used in a single process or a group of processes sharing memory on processors with compatible endian-ness, the natural endianness of those processors can be used regardless of its type, little, big, or some other exotic form.

[3](#). **Other Hash Sizes and XOR Folding**

   Many hash uses require a hash that is not one of the FNV sizes for
   which constants are provided in [Section 4](#). If a larger hash size is
   needed, please contact the authors of this document.

   Most hash applications make use of a hash that is a fixed size binary
   field. Assume that k bits of hash are desired and k is less than 1024
   but not one of the sizes for which constants are provided in [Section
   4](#). The recommended technique is to take the smallest FNV hash of size
   S, where S is larger than k, and calculate the desired hash using xor
   folding as shown below. The final bit masking operation is logically
   unnecessarily if the size of hash is exactly the number of desired
   bits.

```
temp = FNV_S ( data-to-be-hashed )
hash = ( temp xor temp>>k ) bitwise-and ( 2**k - 1 )
```

   Hash functions are a trade-off between speed and strength. For
   example, a somewhat stronger hash may be obtained for exact FNV sizes
   by calculating an FNV twice as long as the desired output (S = 2*k)
   and performing such data folding using a k equal to the size of the
   desired output. However, if a much stronger hash, for example one
   suitable for cryptographic applications, is wanted, algorithms
   designed for that purpose, such as those in [[RFC6234](#)], should be
   used.

   If it is desired to obtain a hash result that is a value between 0
   and max, where max is a not a power of two, simply choose an FNV hash
   size S such that 2**S > max. Then calculate the following:

```
FNV_S mod ( max+1 )
```

   The resulting remainder will be in the range desired but will suffer
   from a bias against large values with the bias being larger if 2**S
   is only a little bigger than max. If this bias is acceptable, no
   further processing is needed. If this bias is unacceptable, it can be
   avoided by retrying for certain high values of hash, as follows,
   before applying the mod operation above:

```
X = ( int( ( 2**S - 1 ) / ( max+1 ) ) ) * ( max+1 )
while ( hash >= X )
      hash = ( hash * FNV_Prime ) + offset_basis
```

[4](#). **FNV Constants**

   The FNV Primes are as follows:

     32 bit FNV_Prime = 2**24 + 2**8 + 0x93 = 16,777,619
                                            = 0x01000193

     64 bit FNV_Prime = 2**40 + 2**8 + 0xB3 = 1,099,511,628,211
                                            = 0x00000100 000001B3

    128 bit FNV_Prime = 2**88 + 2**8 + 0x3B =
                                    309,485,009,821,345,068,724,781,371
                          = 0x00000000 01000000 00000000 0000013B

    256 bit FNV_Prime = 2**168 + 2**8 + 0x63 =
    374,144,419,156,711,147,060,143,317,175,368,453,031,918,731,002,211 =
    0x0000000000000000 0000010000000000 0000000000000000 0000000000000163

    512 bit FNV_Prime = 2**344 + 2**8 + 0x57 = 35,
    835,915,874,844,867,368,919,076,489,095,108,449,946,327,955,754,392,
    558,399,825,615,420,669,938,882,575,126,094,039,892,345,713,852,759 =
    0x0000000000000000 0000000000000000 0000000001000000 0000000000000000
      0000000000000000 0000000000000000 0000000000000000 0000000000000157

    1024 bit FNV_Prime = 2**680 + 2**8 + 0x8D = 5,
    016,456,510,113,118,655,434,598,811,035,278,955,030,765,345,404,790,
    744,303,017,523,831,112,055,108,147,451,509,157,692,220,295,382,716,
    162,651,878,526,895,249,385,292,291,816,524,375,083,746,691,371,804,
    094,271,873,160,484,737,966,720,260,389,217,684,476,157,468,082,573 =
    0x0000000000000000 0000000000000000 0000000000000000 0000000000000000
      0000000000000000 0000010000000000 0000000000000000 0000000000000000
      0000000000000000 0000000000000000 0000000000000000 0000000000000000
      0000000000000000 0000000000000000 0000000000000000 000000000000018D

   The FNV offset_basis values are as follows:

     32 bit offset_basis = 2,166,136,261 = 0x811C9DC5

     64 bit offset_basis = 14695981039346656037 = 0xCBF29CE4 84222325

    128 bit offset_basis = 144066263297769815596495629667062367629 =
                              0x6C62272E 07BB0142 62B82175 6295C58D

    256 bit offset_basis = 100,029,257,958,052,580,907,070,968,
    620,625,704,837,092,796,014,241,193,945,225,284,501,741,471,925,557 =
    0xDD268DBCAAC55036 2D98C384C4E576CC C8B1536847B6BBB3 1023B4C8CAEE0535

```
 512 bit offset_basis = 9,
659,303,129,496,669,498,009,435,400,716,310,466,090,418,745,672,637,
896,108,374,329,434,462,657,994,582,932,197,716,438,449,813,051,892,
206,539,805,784,495,328,239,340,083,876,191,928,701,583,869,517,785 =
0xB86DB0B1171F4416 DCA1E50F309990AC AC87D059C9000000 0000000000000D21
  E948F68A34C192F6 2EA79BC942DBE7CE 182036415F56E34B AC982AAC4AFE9FD9


1024 bit offset_basis =  14,197,795,064,947,621,068,722,070,641,403,
218,320,880,622,795,441,933,960,878,474,914,617,582,723,252,296,732,
303,717,722,150,864,096,521,202,355,549,365,628,174,669,108,571,814,
760,471,015,076,148,029,755,969,804,077,320,157,692,458,563,003,215,
304,957,150,157,403,644,460,363,550,505,412,711,285,966,361,610,267,
868,082,893,823,963,790,439,336,411,086,884,584,107,735,010,676,915 =
0x0000000000000000 005F7A76758ECC4D 32E56D5A591028B7 4B29FC4223FDADA1
  6C3BF34EDA3674DA 9A21D90000000000 0000000000000000 0000000000000000
  0000000000000000 0000000000000000 0000000000000000 000000000004C6D7
  EB6E73802734510A 555F256CC005AE55 6BDE8CC9C6A93B21 AFF4B16C71EE90B3
```

[5](). **The Source Code**

   The following sub-sections are intended, in later versions, to
   include reference C source code and a test driver for FNV-1a.

[5.1]() **FNV C Headers**

   TBD

[5.2]() **FNV C Code**

   TBD

[5.3]() **FNV Test Code**

   TBD

[6]. Security Considerations

   This document is intended to provide convenient open source access by
   the Internet community to the FNV non-cryptographic hash. No
   assertion of suitability for cryptographic applications is made for
   the FNV hash algorithms.

[6.1] Why is FNV Non-Cryptographic?

   A full discussion of cryptographic hash requirements and strength is
   beyond the scope of this document. However, here are three
   characteristics of FNV that would generally be considered to make it
   non-cryptographic:

   1. Work Factor - To make brute force inversion hard, a cryptographic
      hash should be computationally expensive, especially for a general
      purpose processor. But FNV is designed to be very inexpensive on a
      general-purpose processor. (See Appendix A.)

   2. Sticky State - A cryptographic hash should not have a state in
      which it can stick for a plausible input pattern. But, in the very
      unlikely event that the FNV hash variable becomes zero and the
      input is a sequence of zeros, the hash variable will remain at
      zero until there is a non-zero input byte and the final hash value
      will be unaffected by the length of that sequence of zero input
      bytes. Of course, for the common case of fixed length input, this
      would not be significant because the number of non-zero bytes
      would vary inversely with the number of zero bytes and for some
      types of input runs of zeros do not occur. Furthermore, the
      inclusion of even a little unpredictable input may be sufficient
      to stop an adversary from inducing a zero hash variable.

   3. Diffusion - Every output bit of a cryptographic hash should be an
      equally complex function of every input bit. But it is easy to see
      that the least significant bit of a direct FNV hash is the XOR of
      the least significant bits of every input byte and does not depend
      on any other input bit. While more complex, the second least
      significant bit of an FNV hash has a similar weakness. If these
      properties are considered a problem, they can be easily fixed by
      XOR folding (see Section 3).

   Nevertheless, none of the above have proven to be a problem in actual
   practice for the many applications of FNV.

[7](#). **IANA Considerations**

   This document requires no IANA Actions. RFC Editor Note: please
   delete this section before publication.

[8](#). **Acknowledgements**

   The contributions of the following are gratefully acknowledged:

      Frank Ellermann, Bob Moskowitz, and Stefan Santesson.

[9](). References

Below are the normative and informative references for this document.

[9.1]() Normative References

[ASCII] - American National Standards Institute (formerly United
      States of America Standards Institute), "USA Code for
      Information Interchange", ANSI X3.4-1968, 1968. ANSI X3.4-1968
      has been replaced by newer versions with slight modifications,
      but the 1968 version remains definitive for the Internet.

[9.2]() Informative References

[FNV] - FNV web site:
      http://www.isthe.com/chongo/tech/comp/fnv/index.html

[IEEE] - http://www.ieee.org

[IPv6flow] - https://researchspace.auckland.ac.nz/bitstream/handle/
      2292/13240/flowhashRep.pdf

[RFC3174] - Eastlake 3rd, D. and P. Jones, "US Secure Hash Algorithm
      1 (SHA1)", RFC 3174, September 2001.

[RFC6194] - Polk, T., Chen, L., Turner, S., and P. Hoffman, "Security
      Considerations for the SHA-0 and SHA-1 Message-Digest
      Algorithms", RFC 6194, March 2011.

[RFC6234] - Eastlake 3rd, D. and T. Hansen, "US Secure Hash
      Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, May
      2011.

Appendix A: Work Comparison with SHA-1

    This section provides a simplistic rough comparison of the level of
    effort required per input byte to compute FNV-1a and SHA-1 [RFC3174].

    Ignoring transfer of control and conditional tests and equating all
    logical and arithmetic operations, FNV requires 2 operations per
    byte, an XOR and a multiply.

    SHA-1 is a relatively weak cryptographic hash producing a 160-bit
    hash. It that has been partially broken [RFC6194]. It is actually
    designed to accept a bit vector input although almost all computer
    uses apply it to an integer number of bytes. It processes blocks of
    512 bits (64 bytes) and we estimate the effort involved in SHA-1
    processing a full block. Ignoring SHA-1 initial set up, transfer of
    control, and conditional tests, but counting all logical and
    arithmetic operations, including counting indexing as an addition,
    SHA-1 requires 1,744 operations per 64 bytes block or 27.25
    operations per byte. So by this rough measure, it is a little over 13
    times the effort of FNV for large amounts of data. However, FNV is
    commonly used for small inputs. Using the above method, for inputs of
    N bytes, where N is <= 55 so SHA-1 will take one block (SHA-1
    includes padding and an 8-byte length at the end of the data in the
    last block), the ratio of the effort for SHA-1 to the effort for FNV
    will be 872/N. For example, with an 8 byte input, SHA-1 will take 109
    times as much effort as FNV.

    Stronger cryptographic functions than SHA-1 generally have an even
    high work factor.

Appendix B: Previous IETF Reference to FNV

   FNV-1a was referenced in draft-ietf-tls-cached-info-08.txt that has
   since expired. It was later decided that it would be better to use a
   cryptographic hash for that application.

   Below is the Jave code for FNV64 from that TLS draft include by the
   kind permission of the author:

```java
 /**
  * Java code sample, implementing 64 bit FNV-1a
  * By Stefan Santesson
  */

import java.math.BigInteger;

public class FNV {


   static public BigInteger getFNV1aToByte(byte[] inp) {

       BigInteger m = new BigInteger("2").pow(64);
       BigInteger fnvPrime = new BigInteger("1099511628211");
       BigInteger fnvOffsetBasis =
               new BigInteger("14695981039346656037");

       BigInteger digest = fnvOffsetBasis;

       for (byte b : inp) {
           digest = digest.xor(BigInteger.valueOf((int) b & 255));
           digest = digest.multiply(fnvPrime).mod(m);
       }
       return digest;

   }
}
```

Appendix C: A Few Test Vectors

    Below are a few test vectors in the form of ASCII strings and their
    FNV32 and FNV64 hashes using the FNV-1a algorithm.

    Strings without null (zero byte) termination:

    String   FNV32   FNV64
     ""   0x811c9dc5   0xcbf29ce484222325
     "a"   0xe40c292c   0xaf63dc4c8601ec8c
     "foobar"   0xbf9cf968   0x85944171f73967e8

    Strings including null (zero byte) termination:

    String   FNV32   FNV64
     ""   0x050c5d1f   0xaf63bd4c8601b7df
     "a"   0x2b24d044   0x089be207b544f1e4
     "foobar"   0x0c1c9eb8   0x34531ca7168b8f38

Appendix Z: Change Summary

    RFC Editor Note: Please delete this appendix on publication.

From -00 to -01

    1. Add Security Considerations section on why FNV is non-
       cryptographic.

    2. Add Appendix A on a work factor comparison with SHA-1.

    3. Add Appendix B concerning previous IETF draft referenced to FNV.

    4. Minor editorial changes.

From -01 to -02

    1. Correct FNV_Prime determination criteria and add note as to why s
       < 5 and s > 10 are not considered.

    2. Add acknowledgements list.

    3. Add a couple of references.

    4. Minor editorial changes.

From -02 to -03

    1. Replace direct reference to US-ASCII standard with reference to
       RFC 20.

    2. Update dates and verion number.

    3. Minor editing changes.

From -03 to -04

    1. Change reference to RFC 20 back to a reference to the ANSI 1968
       ASCII standard.

    2. Minor addition to Section 6, point 3.

    3. Update dates and version number.

    4. Minor editing changes.

From -04 to -05

    1. Add Twitter as a use example and IPv6 flow hash study reference.

    2. Update dates and version number.

Author's Address

    Glenn Fowler
    AT&T Labs Research
    180 Park Avenue
    Florham Park, NJ 07932 USA

    Email: gsf@research.att.com
    URL: http://www.research.att.com/~gsf/

    Landon Curt Noll
    Cisco Systems
    170 West Tasman Drive
    San Jose, CA 95134 USA

    Telephone: +1-408-424-1102
    Email: fnv-rfc-mail@asthe.com
    URL: http://www.isthe.com/chongo/index.html

    Kiem-Phong Vo
    AT&T Labs Research
    180 Park Avenue
    Florham Park, NJ 07932 USA

    Email: kpv@research.att.com
    URL: http://www.research.att.com/info/kpv/

    Donald Eastlake
    Huawei Technologies
    155 Beaver Street
    Milford, MA 01757 USA

    Telephone: +1-508-333-2270
    EMail: d3e3e3@gmail.com

Copyright, Disclaimer, and Additional IPR Provisions