



Generating pseudo-random N_Port_IDs

It is as easy as FNV!

Landon Curt Noll T11/10-137v0
chongo@cisco.com

Generating pseudo-random N_Port_IDs

- Step 0: Initialize the 10-octet state
 - Initialize to the concatenation of {N_Port_Name, 23209}
- Step 1: Form trial ID from FNV 1a hash of tweaked state
 - Tweak the state, 32 bit FNV 1a hash, then XOR fold into 16 bits
- Step 2: If trial ID is a reserved ID, return to Step 1
 - Avoid reserved IDs 0x00000 and 0x00FFFF
- Step 3: If trial ID is in use, return to Step 1
 - Using the proposed VN2NV LUID method
- Step 4: Use trial ID as our N_Port_ID

The FNV 1a hash

- Fowler-Noll-Vo hash

- Widely used public domain algorithm since 1991 in some:
 - DNS servers, DB libs (mdbm, hash_map, ...), innd, search engines, parallel kernel object managers, EMail servers, anti-spam filters, NFS implementations, Video games, Mobile phone IDs, PHP, Twitter, ...
- <http://www.isthe.com/chongo/tech/comp/fnv/index.html>

- High dispersion

- A slight change in data will produce a very different hash value

- Low CPU impact

- 2 ops per octet
- For N_Port_ID method: 10 8-bit XORs + 10 32-bit multiplies
or: 10 8-bit XORs + 50 32-bit shifts & adds

FNV 1a hash function - Straight forward coding

10 8-bit XORs + 10 32-bit multiplies

```
id
fnv_hash_state( union stuff *state )
{
    u_int32_t hash = FNV_32_BASIS;    /* hash state */

    /* compute 32 bit FNV 1a hash of state */
    hash = (hash^state->octet[0]) * FNV_32_PRIME;
    hash = (hash^state->octet[1]) * FNV_32_PRIME;
        . . .
    hash = (hash^state->octet[9]) * FNV_32_PRIME;

    /* XOR fold 32 bits into 16 bits */
    hash = (hash&0xFFFF) ^ (hash>>16);

    /* return ID with only the lower 16 bits set */
    return (id)hash;
};
```

Step 1: Form trial ID from FNV 1a hash of tweaked state

Step 2: If trial ID is a reserved ID, return to Step 1

```
id
generate_trial_ID( union stuff *state )
{
    id trial_id;      /* potential N_Port_ID */

    /* generate an N_Port_ID that is NOT reserved */
    do {
        /* tweak state */
        ++state->data.b64;
        /* generate ID (16 bit value with upper bits set to 0) */
        trial_id = fnv_hash_state( state );
        /* save low 16 bits of ID for next time */
        state->data.b16 = (u_int16_t) trial_id;
    } while (trial_id == 0 || trial_id == 0xFFFF);

    /* return potential N_Port_ID */
    return trial_id;
};
```

Step 3: If trial ID is in use, return to Step 1

Step 4: Use trial ID as our N_Port_ID

```
#define PATIENCE 10 /* max retries to find a free N_Port_ID */
union stuff state; /* hash state */
int attempt = 0; /* attempts to find a free N_Port_ID */
id port_id; /* a potential N_Port_ID */

/* Step 0: Initialize */
state.data.b64 = N_Port_Name;
state.data.b16 = 23209; /* one of a my favorite primes :-) */

/* look for an unused N_PORT_ID */
do {
    port_id = generate_trial_ID( &state );
} while (attempt++ < PATIENCE && id_in_use( port_id ));

/* abort if we ran out of patience */
if (attempt > PATIENCE) {
    error_notification(E_RAN_OUT_OF_PATIENCE);
}

/* use our N_Port_ID */
} else {
    set_N_Port_ID( port_id );
}
```

FNV 1a hash alternative without the multiply 10 8-bit XORs + 50 32-bit shifts & additions

```
id
fnv_hash_state( union stuff *state )
{
    u_int32_t hash = FNV_32_BASIS;  /* hash value */
    size_t i;

    /* compute 32 bit FNV 1a hash of state */
    for (i=0; i < STATE_LEN; ++i) {
        hash ^= (u_int32_t) state->octet[i];
        hash += (hash<<1) + (hash<<4) + (hash<<7) +
                (hash<<8) + (hash<<24);
    }

    /* XOR fold 32 bits into 16 bits */
    hash = (hash&0xFFFF) ^ (hash>>16);

    /* return ID with only the lower 16 bits set */
    return (id)hash;
};
```

FNV 1a hash function alternative coded as a looping function

```
u_int32_t
fnv_32a_buf(void *buf, size_t len, u_int32_t hash)
{
    u_int8_t *bp = (u_int8_t *)buf; /* start of buffer */
    u_int8_t char *be = bp + len; /* beyond end of buffer */

    /* FNV-1a hash each octet in the buffer */
    while (bp < be) {
        hash ^= (u_int32_t)*bp++;
        hash *= FNV_32_PRIME;
    }

    /* return our new hash value */
    return hash;
}

id
fnv_hash_state( union stuff *state )
{
    u_int32_t hash; /* 32 bit FNV hash value */

    hash = fnv_32a_buf(state->octet, STATE_LEN, FNV_32_BASIS); /* compute FNV1a hash of state */
    hash = (hash&0xFFFF) ^ (hash>>16); /* XOR fold 32 bits into 16 bits */

    return (id)hash; /* return ID with only the lower 16 bits set */
};
```

FNV 1a hash function alternative coded in x86 assembler

- On an Intel Core 2 Duo E6600 (2.4 GHz)
 - 36 octets of code can FNV 1a hash data at 575 mb/second

```
; u_int32_t fast_fnv_32a_buf(void *buf, size_t len, u_int32_t hash)
```

```
fast_fnv_32a_buf:
```

```
    push    ebx
    push    esi
    push    edi
    mov     esi, [esp + 10h] ;buffer
    mov     ecx, [esp + 14h] ;length
    mov     eax, [esp + 18h] ;basis
    mov     edi, 01000193h   ;fnv_32_prime
    xor     ebx, ebx
nexta:
    mov     bl, [esi]
    xor     eax, ebx
    mul     edi
    inc     esi
    dec     ecx
    jnz     nexta
    pop     edi
    pop     esi
    pop     ebx
    retn   0ch
```

Pseudo C header stuff

Things only an ANSI C coder cares about

```
#define FNV_32_PRIME 16777619U    /* 32 bit FNV prime */
#define FNV_32_BASIS 2166136261U /* 32 bit FNV basis */

typedef u_int24_t id;           /* 24 bit N_Port_ID */

/* The state we FNV 1a hash to name N_Port IDs */
struct state_data {
    u_int64_t b64; /* initialized to N_Port_Name */
    u_int16_t b16; /* previous trial N_Port_ID */
};

#define STATE_LEN sizeof(struct state_data)

union stuff {
    struct state_data data; /* state elements */
    u_int8_t octet[STATE_LEN]; /* octets to hash */
};
```

EOT

- Questions?
- Public domain simulation code available on request
 - Send EMail to:

chongo@cisco.com