



# Generating Locally Unique point-2-point N\_Port\_IDs The Caffè Corretto version

Landon Curt Noll  
chongo@cisco.com

T11/10-137v1

Image Credit:  
Flickr user duegnazio  
Creative Commons License



Coffee & Sea © annia316  
Permission to use with attribution  
<http://www.flickr.com/photos/annia316/487151804/>

# Generating Locally Unique N\_Port\_IDs for point-2-point

- Step 0: Initialize the 12-octet state
  - Initialize to the concatenation of {N\_Port\_Name, ENode Mac}
- 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 ID matches an a recent ID, return to Step 1
  - Compare against past 10 saved IDs from output of Step 2
- Step 4: If trial ID is in use, return to Step 1
  - According to the VN2VN protocol
- Step 5: Use trial ID as our N\_Port\_ID

Image Credit:  
Flickr user [Blackout14\\*](#)  
Creative Commons License



# What changed from T11/10-137v0?

## New material in red

- Step 0: Initialize the **12-octet** state
  - Initialize to the concatenation of {N\_Port\_Name, **ENode Mac**}
- 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 ID matches an a recent ID, return to Step 1**
  - **Compare against past 10 saved IDs from output of Step 2**
- Step 4: If trial ID is in use, return to Step 1
  - According to the VN2VN protocol
- Step 5: Use trial ID as our N\_Port\_ID

Image Credit:  
Flickr user ~~Blackout~~14\*  
Creative Commons License



# The FNV 1a hash



Image Credit:  
Flickr user miheco  
Creative Commons License

- 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>
- *Draft RFC for FNV hash in progress*

- 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: 12 8-bit XORs + 12 32-bit multiplies  
or: 12 8-bit XORs + 60 32-bit shifts & adds

# FNV 1a hash function - Straight forward coding

## 12 8-bit XORs + 12 32-bit multiplies



Image Credit:  
www.gewoonslechtwerk.nl

```
u_int32_t
fnv_hash_state(union stuff *state)
{
    u_int32_t hash = FNV_32_BASIS; /* FNV 1a hash state */
    size_t i;

    /* compute 32 bit FNV 1a hash of the 12 octet state */
    for (i=0; i < STATE_LEN; ++i) {
        hash = (hash ^ state->byte[i]) * FNV_32_PRIME;
    }

    /* return the 32 bit FNV 1a hash */
    return 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

Image Credit:  
Flickr user benben  
Creative Commons License

```
u_int16_t
generate_tentative_LUID( union stuff *state )
{
    u_int32_t hash;          /* 32 bit FNV 1a hash of state */
    u_int16_t tentative_id; /* tentative LUID */
    int regen;              /* 1 ==> generate another LUID */
    u_int32_t i;

    /*
     * generate a tentative LUID not returned recently
     */
    do {
        /*
         * generate a tentative LUID that is NOT reserved [1, 0xFFFF]
         */
        do {
            /* tweak state */
            ++state->data.b32;

            /* generate 32-bit FNV 1a hash value */
            hash = fnv_hash_state( state );

            /* XOR fold 32 bits into 16 bits */
            tentative_id = (u_int16_t) (hash ^ (hash>>16));

        } while (tentative_id == 0 || tentative_id == 0xFFFF);
    }
```



## Step 3: If ID matches an a recent ID, return to Step 1

```
/*
 * scan our memory of recent tentative LUIDs, reject if match found
 */
for (regen=0, i=0; i < idmem_indx.beyond; ++i) {
    if (tentative_id == idmem[i]) {
        regen = 1;
        break;
    }
}

/*
 * if tentative LUID is not recent, remember it for next time
 */
if (regen == 0) {
    /*
     * remember our tentative ID advance next, OK to cycle back to 0
     */
    idmem[idmem_indx.next++] = tentative_id;

    /* if we grew our table, remember our new beyond slot */
    if (idmem_indx.beyond < ID_MEMORY) {
        ++idmem_indx.beyond;
    }
}
} while (regen != 0);

/* return a tentative LUID */
return tentative_id;
};
```



Image Credit: flickr - Creative Commons - originalsignal.com

# Step 4: If trial ID is in use, return to Step 1

## Step 5: 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 */
u_int16_t port_id; /* a potential N_Port_ID */

/* Step 0: Initialize */
state.data.b64 = N_Port_Name;
state.data.b32 = (u_int32_t) enode_mac;

/* 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 );
}
```

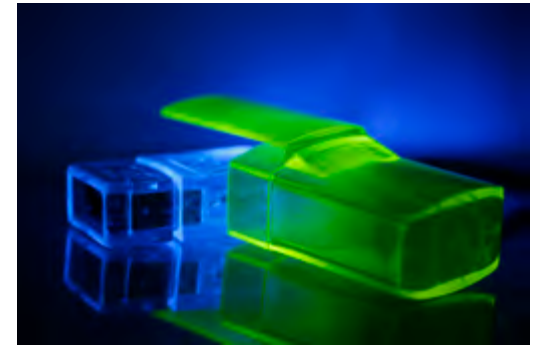


Image Credit:  
Flickr user jepoirrier  
Creative Commons License

# FNV 1a hash alternative without the multiply 12 8-bit XORs + 60 32-bit shifts & additions

Image Credit:  
www.gelderlander.nl



```
u_int32_t
fnv_hash_state (union stuff *state)
{
    u_int32_t hash = FNV_32_BASIS; /* FNV 1a hash state */
    size_t i;

    /* compute 32 bit FNV 1a hash of the 12 octet 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);
    }

    /* return the 32 bit FNV 1a hash */
    return 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;
}
```

# 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
```



B2 Image Credit:  
Flickr user James Gordon  
Creative Commons License

# Pseudo C header stuff

## Things only an ANSI C coder cares about

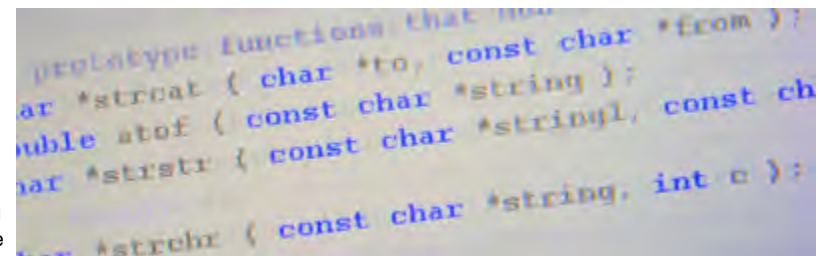
```
#define FNV_32_PRIME 16777619U    /* 32 bit FNV prime */
#define FNV_32_BASIS 2166136261U /* 32 bit FNV basis */

/* The state we FNV 1a hash to form N_Port IDs */
struct state_data {
    u_int64_t b64; /* initialized to N_Port_Name */
    u_int32_t b32; /* low order bits of ENode MAC */
};

#define STATE_LEN sizeof(struct state_data)

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

Image Credit:  
Flickr user QualityFrog  
Creative Commons License



# Locally Unique N\_Port\_ID generation simulation

---

- Simulated networks: 100 000
- Generate random {N\_Port\_Name, ENode Mac} values
  - Track how the saved ID table grows
  - Add N\_Ports to each network until we need 10 network retries
- Simulated networks with an average of 28 399 N\_Ports
- Total N\_Ports simulated: 2 839 878 902



Image Credit:  
Flickr user zen  
Creative Commons License

# Example of a simulated network

- after 330 N\_Ports, an N\_Port required 1 loops
- after 3514 N\_Ports, an N\_Port required 2 loops
- after 4505 N\_Ports, an N\_Port required 3 loops
- after 7950 N\_Ports, an N\_Port required 4 loops
- after 10587 N\_Ports, an N\_Port required 5 loops
- after 19367 N\_Ports, an N\_Port required 6 loops
- after 22368 N\_Ports, an N\_Port required 7 loops
- after 22873 N\_Ports, an N\_Port required 8 loops
- after 27697 N\_Ports, an N\_Port required 9 loops
- after 29279 N\_Ports, an N\_Port required 10 loops



Image Credit:  
Nathan Abels  
[nathanabels.blogspot.com](http://nathanabels.blogspot.com)



# Simulation results

- 100 000 Networks, 2 839 878 902 N\_Ports

<b>Max loops required</b>	<b>Min N_Ports to reach max loops</b>	<b>Ave N_Ports <math>\pm</math> Std. Deviation to reach max loops</b>	<b>Max N_Ports to reach max loops</b>
1	1	320 $\pm$ 167	1245
2	70	2095 $\pm$ 759	5272
3	150	5242 $\pm$ 1474	11094
4	777	9030 $\pm$ 2063	15999
5	2157	12897 $\pm$ 2486	21136
6	3625	16598 $\pm$ 2780	25403
7	4590	19999 $\pm$ 2958	29555
8	6767	23095 $\pm$ 3061	31453
9	7975	25881 $\pm$ 3118	31770
10	8371	28398 $\pm$ 3136	31979

# EOT

---

- Questions?
- Public domain simulation code available on request
  - Send EMail to: `chongo@cisco.com`



SeaTac airport sign, © Wake Rockett  
Permission to share unmodified with attribution  
<http://www.flickr.com/photos/waderockett/244675267/>