Free from Using Zone Identifier for IPv6 Link-Local

<draft-kitamura-ipv6-zoneid-free-03.txt>

Hiroshi KITAMURA

NEC Corporation

kitamura@da.jp.nec.com

Analysis of the problems of using Zone-ID

Problem 1:

Zone-ID information is **hard to tell** for **normal end users**.

Problem 2:

Zone-ID is **node-local** info. and **cannot be shared with others**.

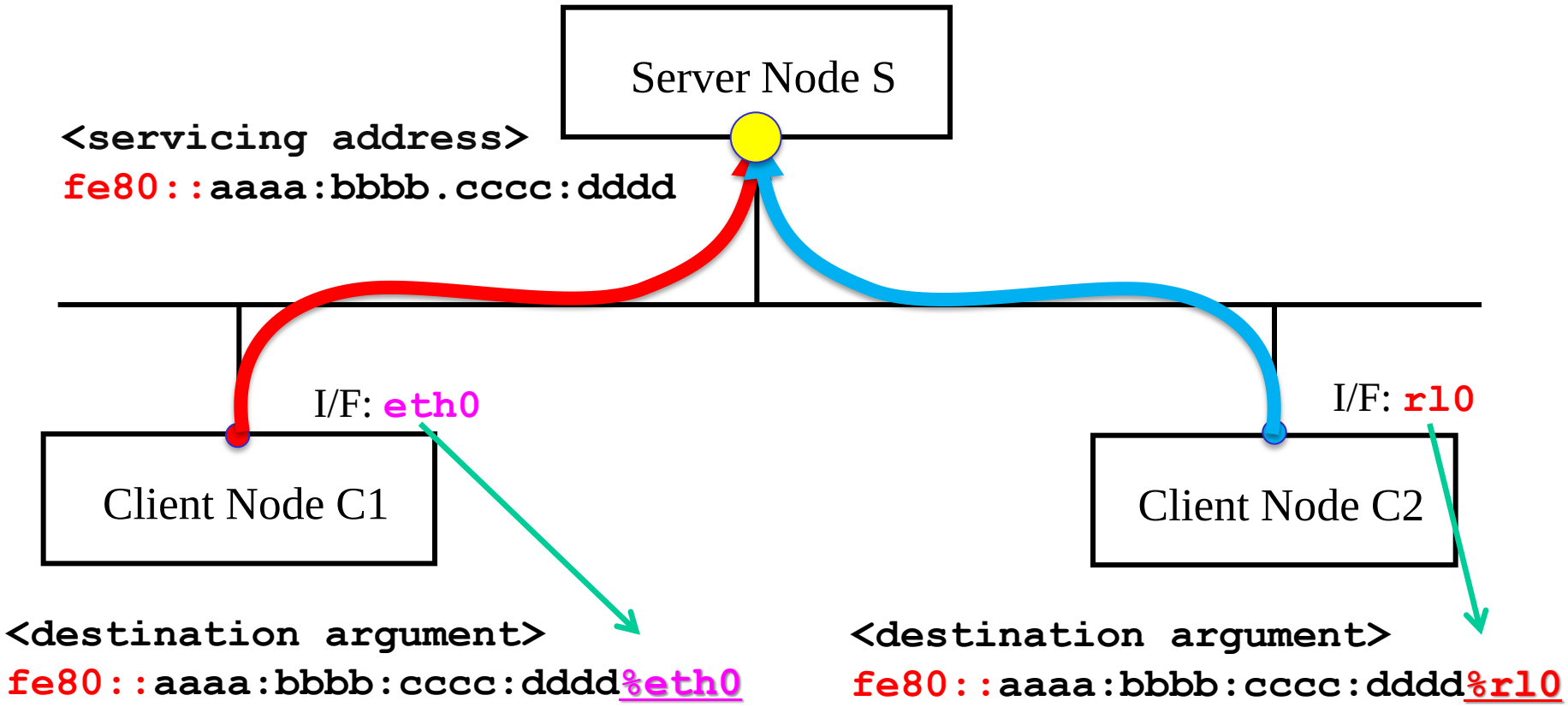
Problem 3:

Zone-ID depends on node's situation and **can be changed**.

Problem 4:

"<Address>%<Zone-ID>" representation (with %<Zone-ID>) is **quite different** from traditional **"<Address>"** only representation (without %<Zone-ID>).

Problem: on providing services via Link-Local Address



The provided **services are the same**,
but Client Nodes can **NOT use the same** arguments.

Goals of "**Zone-ID Free**" function and what will be done

Current Style **with** %<Zone-ID>

```
> ping6 <Link-Local_Address_A>%<Zone-ID_X>
> ping6 <Link-Local_Address_B>%<Zone-ID_Y>

> ssh   <Link-Local_Address_A>%<Zone-ID_X>
> ssh   <Link-Local_Address_B>%<Zone-ID_Y>
```

Goal Communication Style **without** %<Zone-ID>

```
> ping6 <Link-Local_Address_A>
> ping6 <Link-Local_Address_B>

> ssh   <Link-Local_Address_A>
> ssh   <Link-Local_Address_B>
```

What "**Zone-ID Free**" function should do is:

Without %<Zone-ID> info. at argument,

Go on communication programs (without causing errors / stopping)

Where is Zone-ID specification?

Basic Socket API [RFC3493] defines
"**sockaddr_in6**" structure as follows:

```
struct sockaddr_in6 {
  sa_family_t sin6_family;   /* AF_INET6 */
  in_port_t   sin6_port;     /* transport layer port # */
  uint32_t    sin6_flowinfo; /* IPv6 flow information */
  struct in6_addr sin6_addr; /* IPv6 address */
  uint32_t    sin6_scope_id; /* set of interfaces for a scope */
 };
```

"**sin6_scope_id**" member definition is
directly related with **Zone-ID** information.

(Advanced Socket API [RFC3542] also defines Zone-ID related specification.
Since targets of this document are simple usages of normal end users,
Advanced Socket API is not discussed here.)

How Zone-ID info. dealt with in the current impl.

Zone-ID (at Application) => `sin6_scope_id` (at kernel)

If Zone-ID info. is not provided by end users,

`sin6_scope_id` filled with **0** (special value).

When the kernel meets `sin6_scope_id` **=0** situation:

Reactions are categorized into two cases:

1: Non Link-Local (such as **Global**) Address Case:

The kernel proceeds and applications **go on**.

2: **Link-Local** Address Case:

The kernel can NOT proceed the operations.

Applications receive **ERROR** return value from the kernel,

and applications can **NOT continue** and **STOP**.

Design and Implementation of "**Zone-ID Free**" function

In order to avoid ERROR and STOP situation,
`sin6_scope_id` value should be resolved **by the kernel**

Two types to resolving:

One is **NO probing type**

without issuing probes, `sin6_scope_id` is resolved.
(**dominant** at normal end users environment)

The other is **Probing type**

with issuing probes, `sin6_scope_id` is resolved.

NO probing type (dominant)

There are three cases:

- **One** Case:
  Number of available interface of a node is **One**.
- **Self** Case:
  Target address is issuer's **Self** address.
  (loopback type communication)
- **Filled** Case:
  Neighbor Cache entry has already been **Filled**.
  (N.C. entry is ***filled by some operations***
  before Socket functions are called.)

From a view point that ***no probes are needed***,
these cases can be called "**trivial**" type.

Probing type “**Zone-ID Learning**”

This is the **last resort**.

If the situation does not match any of no probing cases (One/Self/Filled), “`sin6_scope_id` (Zone-ID)” is resolved (learned) by issuing probes.

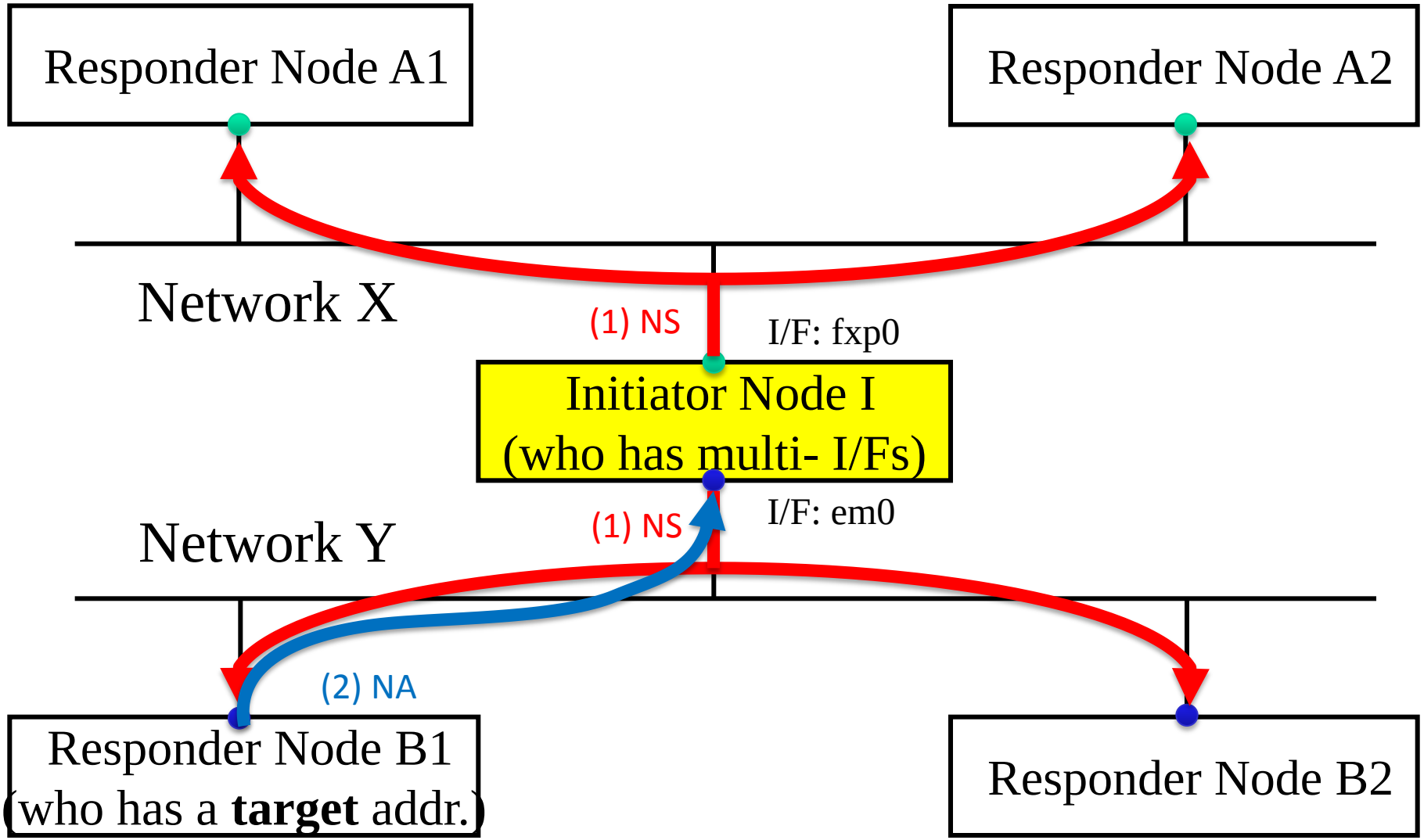
This method is called "**Zone-ID Learning**“

NS (Neighbor Solicitation) is used for a probe.

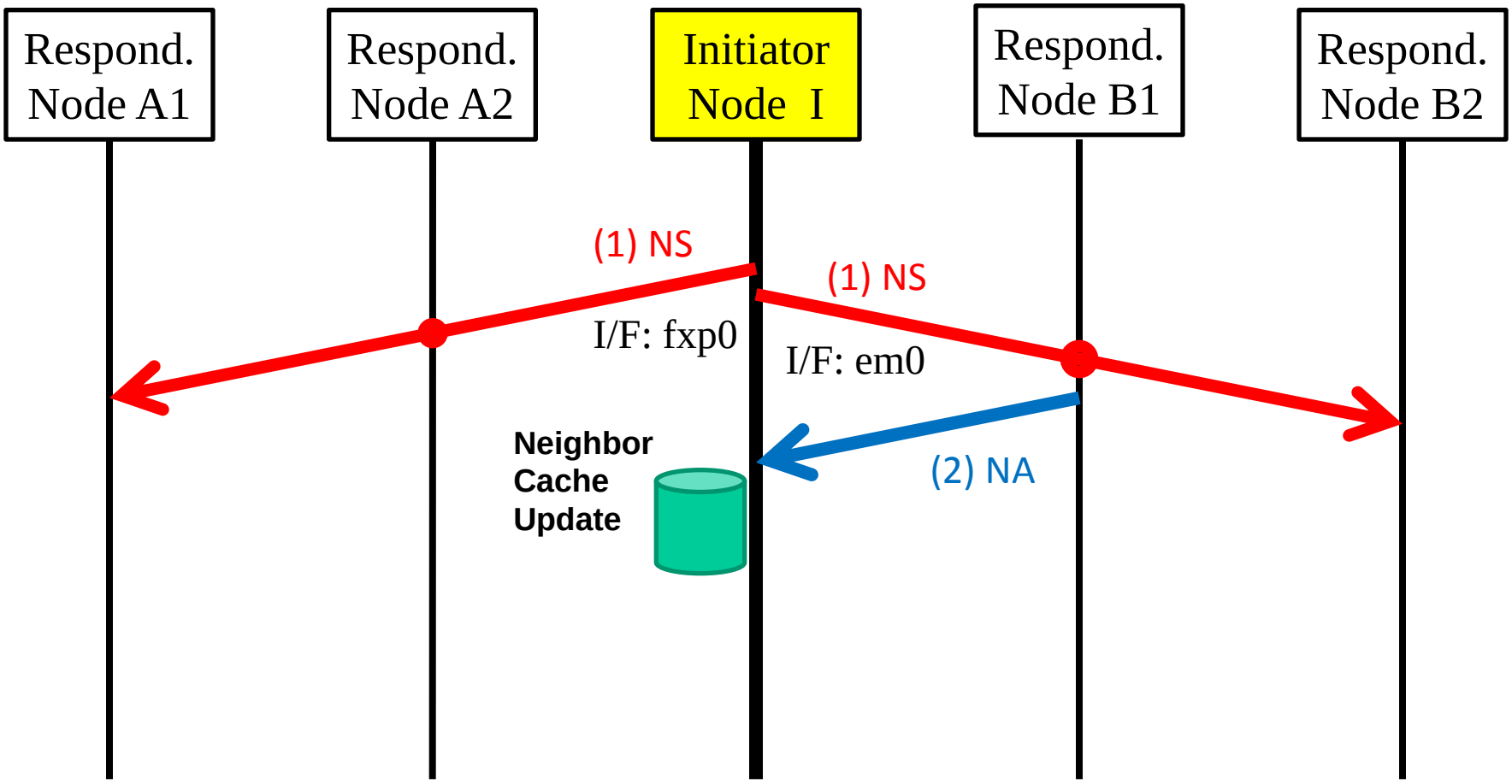
At **normal end user** environment, Zone-ID information is usually obtained by either of One/Self/Filled Cases.

It is **very rare** to execute "Zone-ID Leaning" method.

Zone-ID Learning (send multiple NS probes)



Zone-ID Learning (send multiple NS probes) Typical Sequence Chart



Considerations on "**Zone-ID Learning**"

There are three cases to reply to multiple-probes.

1. **No Reply**:
   Clear situation. There are no difficulties.
2. **Single Reply**:
   Most comfortable and typical situation.
   Zone-ID is resolved. There are no difficulties.
3. **Multiple Replies**:

   Rarely happened. If appropriate Zone-ID is needed, **users can fall back** to the Zone-ID accompanied method.

Since the "Zone-ID Free" functions are **upper compatible**, there is no difficulties to fall back to the current method.

Target Socket functions and Implementation

Socket functions that returns **ERROR** value when they are called with `sin6_scope_id` = 0 **must be improved**.

The following 5 types of Socket functions are actual targets.

1. **TCP connect()**
2. **UDP sendto()/sendmsg()**
3. **UDP connect()**
4. **TCP bind()**
5. **UDP bind()**

Only with modifying the kernel implementation,
socket functions do **NOT return ERROR** value.
So it is not necessary to modify userland/library implementation.

Implementation

Currently, above described "Zone-ID Free" functions have been implemented and verified
under the following OS environment.

- OS: **Linux** (Ubuntu 13.04)
- Kernel: **3.8.13** (Ubuntu 13.04) and **3.8.8** Vanilla kernel
- Code size (kernel patch) : less than **1k line**

Summary

This I-D describes:

- How "Zone-ID Free" functions work
- How **end users are released** from using nuisance Zone-ID

The "Zone-ID Free" functions are:

– Upper compatible with the current usage of link-local addresses

– Harmless to the existing communications.

End users' network connecting topology is generally simple.

Even if their nodes are equipped multiple interfaces,

they use only **one interface** to connect to network at one time

It is **dominant** that Zone-ID is resolved without issuing probes.

For the rare cases, a new technology "Zone-ID Learning" that issues multiple probes to resolve `sin6_scope_id` is introduced.

Discussions

Please let us know your comments.

Acknowledgment:

A part of these works are supported by the program: **SCOPE** (Strategic Information and Communications R&D Promotion Programme) operated by Ministry of Internal Affairs and Communications of JAPAN.

Reserved slides are started from here.

Socket functions and Implementation (1/3)

1. **TCP connect()**
2. **UDP sendto()/sendmsg()**

Since goals of above 2 types of Socket functions is to issues IP packets, it is easy to associate that they can become trigger functions to issue multiple NS probing messages (when correspondent neighbor cache entry is empty). IP packets are issued soon after NS/NA sequences are finished.

Socket functions and Implementation (2/3)

3. UDP connect()

Since goal of UDP connect() functions is NOT to issues IP packets, it is not easy to associate that UDP connect() functions can become trigger functions to issue NS probing messages. However, in order not to return Error value and to resolve "**`sin6_scope_id`**" value, multiple NS probing messages can be issued here.

Socket functions and Implementation (3/3)

4. **TCP bind()**
5. **UDP bind()**

Since an argument of bind() function must be Self address, "**`sin6_scope_id`**" value is solved by no probing (Self Case).