Upcoming IPv6 Features -Linux IPv6 2013-

http://www.linux-ipv6.org/materials/201305-LCJ2013-IPv6/

YOSHIFUJI Hideaki

<yoshfuji@linux-ipv6.org>
Twitter: yoshfuji

Table of Contents

- Who am I?
- Recent IPv6 Topics
- Upcoming / Target features
- Discussion

Who am I?

YOSHIFUJI Hideaki (吉藤 英明)

- Affiliation
  - Keio University
  - WIDE Project
  - USAGI Project
    - IPv6 development on Linux
- Co-maintainer of Linux Networking [IPv4/IPv6]

Recent IPv6 Topics

- IPv6 Netfilter NPT (3.7, fixed in 3.8)
- IPv6 over Firewire (3.10)
- Tokenized ID (3.10)

Netfilter NPT

- IPv6-to-IPv6 Network Prefix Translation (RFC6296)
- A stateless IPv6-to-IPv6 Network Prefix Translation (NPTv6) function, designed to provide address independence to the edge network.
- Mapping is selected so that the node does not need to adjust checksum in upper layer protocols.
- Maybe useful to provide access to global internet via wireless network on your laptop.

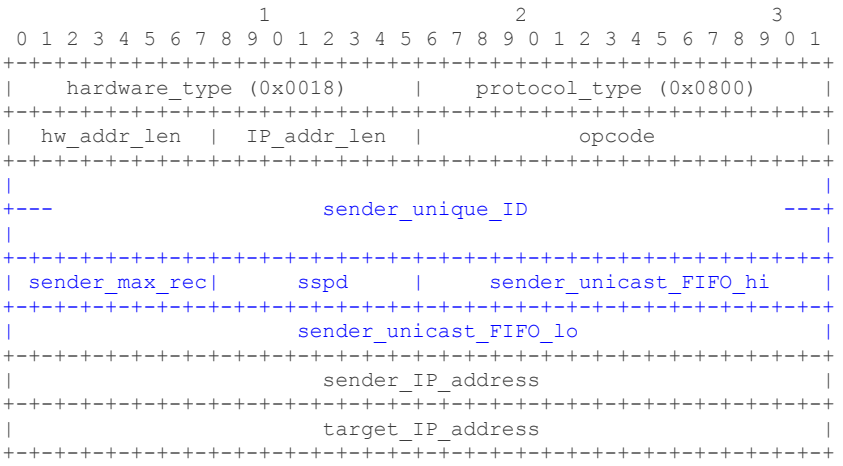
IPv6 over Firewire

- One day, someone asked a question about allocating memory for Neighbor Discovery message for IPv6, to implement IPv6 over Firewire.
- I was curious why he needs such a thing...
- I looked into driver/firewire/net.c (which implements IPv4 over Firewire (RFC2734) and found that it parsed/modified packets.
- BAD, BAD, BAD!!!
  - IPv6 NDP is more flexible (or say, complex) than IPv4 ARP.
    - NDP has strict state machine; it is more difficult to maintain it in the driver layer.
    - Drivers should be simple as possible
  - IPsec (or SEND) will not work.
  - I could not not believe that IPv6 over “media” specification requires such nasty design...

Overview of IPv4 over Firewire: Address Resolution

- Driver maintains EUI-64 => Node-ID mapping (which is volatile)
- Driver looks into packet, and convert packet format.
- Difference between standard ARP and 1394ARP
  - Additional field other that “unique ID”
    - max_rec, sspd and unicast FIFO address
  - No “target” hardware address field
    - It always act as if it were all zero

- 1394 ARP (RFC2734) format



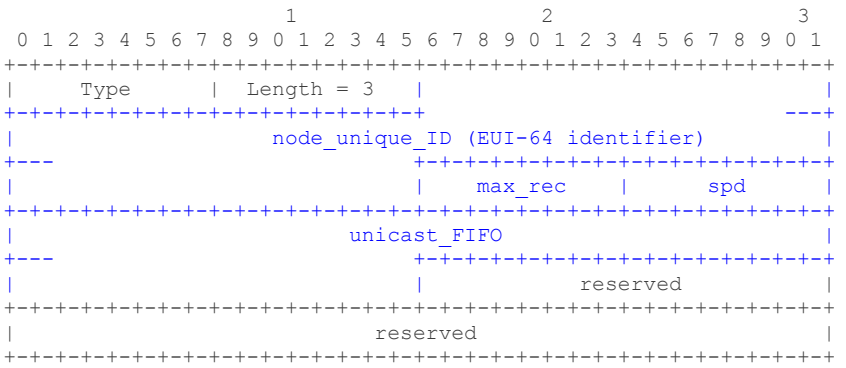
Overview of IPv4 over Firewire: Multicast

- Limited multicast support
  - ARP and IPv4 Multicast packets are sent as GASP (Global Asynchronous Stream Packet)
  - No L2 multicast support
    - “L3” address should be negotiated via MCAP (Multicast Channel Allocation Protocol) in full implementation.

Overview of IPv6 over Firewire: Address Resolution

- No special packet format
  - In IPv6 NDP, “target” and “source” are options.
- Additional data in these options
- “Reserved” are set to zero because of alignment requirement.

- IPv6 NDP: Source/Target Link-layer Option (RFC3146)



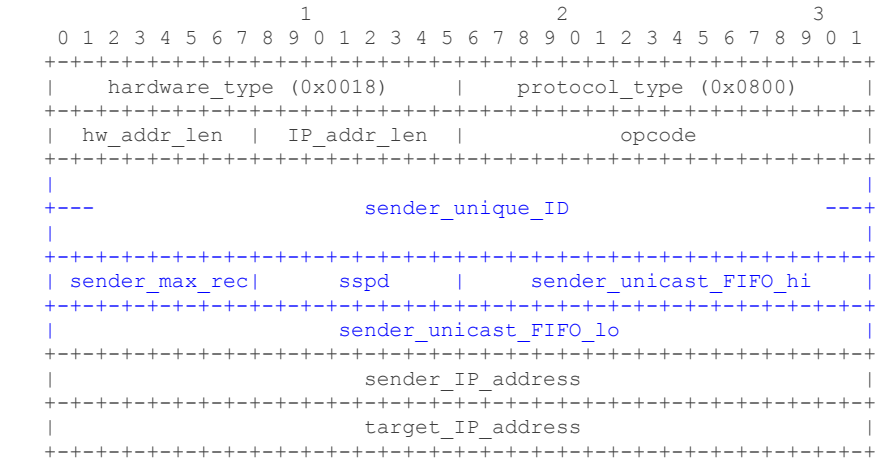
Overview of IPv6 over Firewire: Multicast

- NDP messages are sent to L3 multicast
- Specification explicitly says that they must be sent via GASP.
- Other multicast should be negotiated via MCAP
  - It uses L3 address
  - IGMP layer tells drivers “L2 multicast address” only; not L3 address.
  - 16 bytes does NOT fit into original 64bit-long “uniq_id” field.

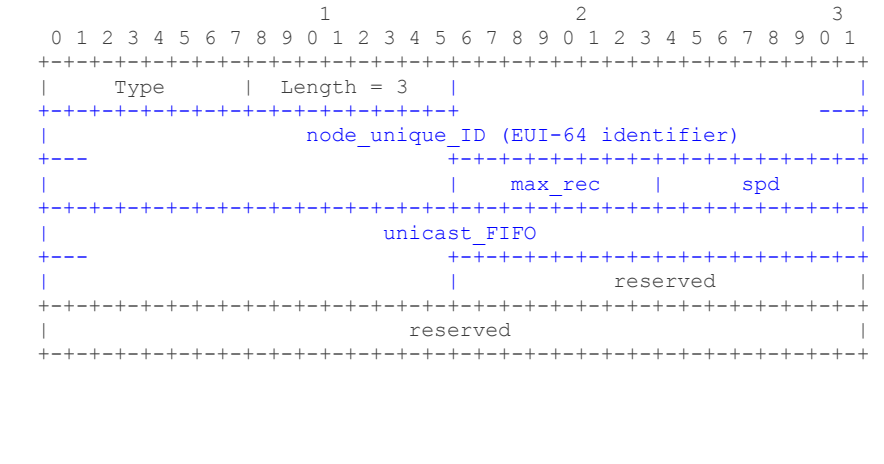
IPv4/IPv6 over Firewire, Reworked

- I strongly proposed to change the “hardware address” format to include other additional data.
  - max_rec, sspd and unicast_fifo
  - IPv4 (1394ARP) and IPv6 (NDP) shares the format.

IPv4 (1394 ARP): RFC2734



IPv6 (NDP): RFC3146



IPv4/IPv6 over Firewire, Reworked (cont'ed)

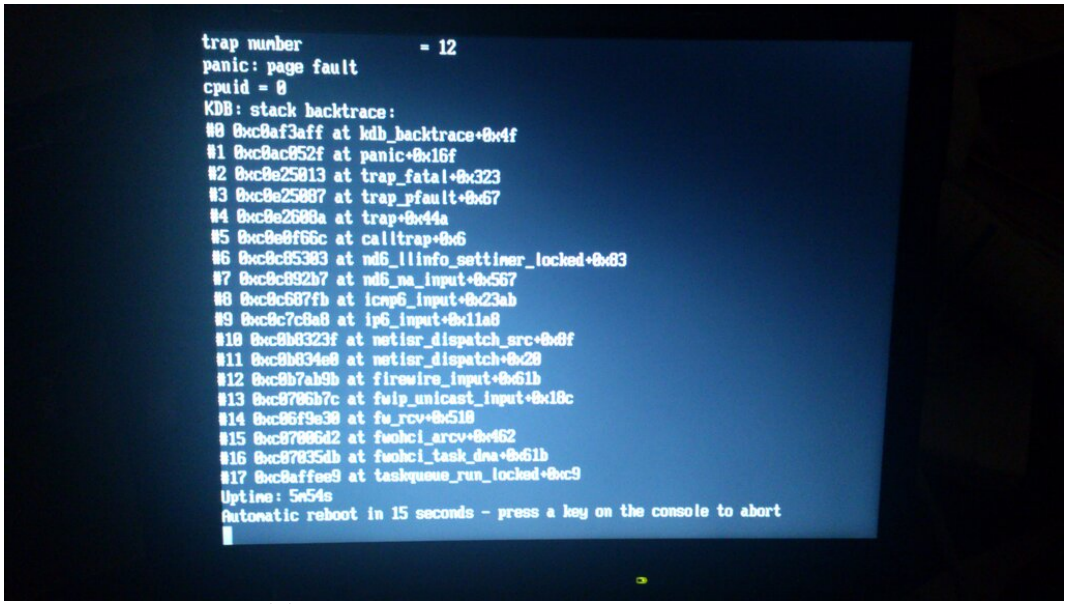
- Changing the L2 hardware address format allows us to have direct map between IPv6 multicast addresses and L2 addresses.
  - If “multicast” bit (the lowest bit of the first octet) is on, send the packet via GASP
    - IPv6 multicast address is in ff00::/8, whose lowest bit of the first octet is set.
    - IPv4 can be mappe d as 01:00:00:...:00:xx:xx:xx:xx, for example.
  - Otherwise, it is L2 unicast; use the additional information embedded in the destination (virtual) hardware address to send.
- Note: All multicast adresses are mapped as L2 broadcast (ff:ff:...:ff) so far.
  - In fact, MCAP is almost broken. Broadcast context is an expensive resource to allocate and it is not always available. Unless all of receivers can allocate the resource, sender cannot send multicast packet for that address via that channel.

Interoperability with MacOS

- FAILED
  - He sends ICMPv6 with wrong checksum and he does not accept NDP messages from Linux.
  - If Linux ignores the checksum (by setting CHECKSUM_UNNECESSARY in skb), Linux can receive his packet. Once he sends NS (or NA), Linux can send ping him without receiving NDP messages.
    - Source link-layer option in NS/NA from MacOS
    - MSG_CONFIRM in ping6

Interoperability with FreeBSD

- FAILED
  - FreeBSD carshes when trying to ping other side; e.g. ping6 fe80::1%fwip0
  - http://www.freebsd.org/cgi/query-pr.cgi?pr=176596



NetBSD?

- FAILED
  - It seems to support IPv6 over Firewire, but no packet is sent or received...

IPv6 over Firewire: Conclusion

- IPv6 over Firewire works on Linux
  - MacOS, FreeBSD, NetBSD: broken.
  - Solaris and OpenBSD: untested
  - Windows: unsupported
- I love Linux, which seems to be maintained better.

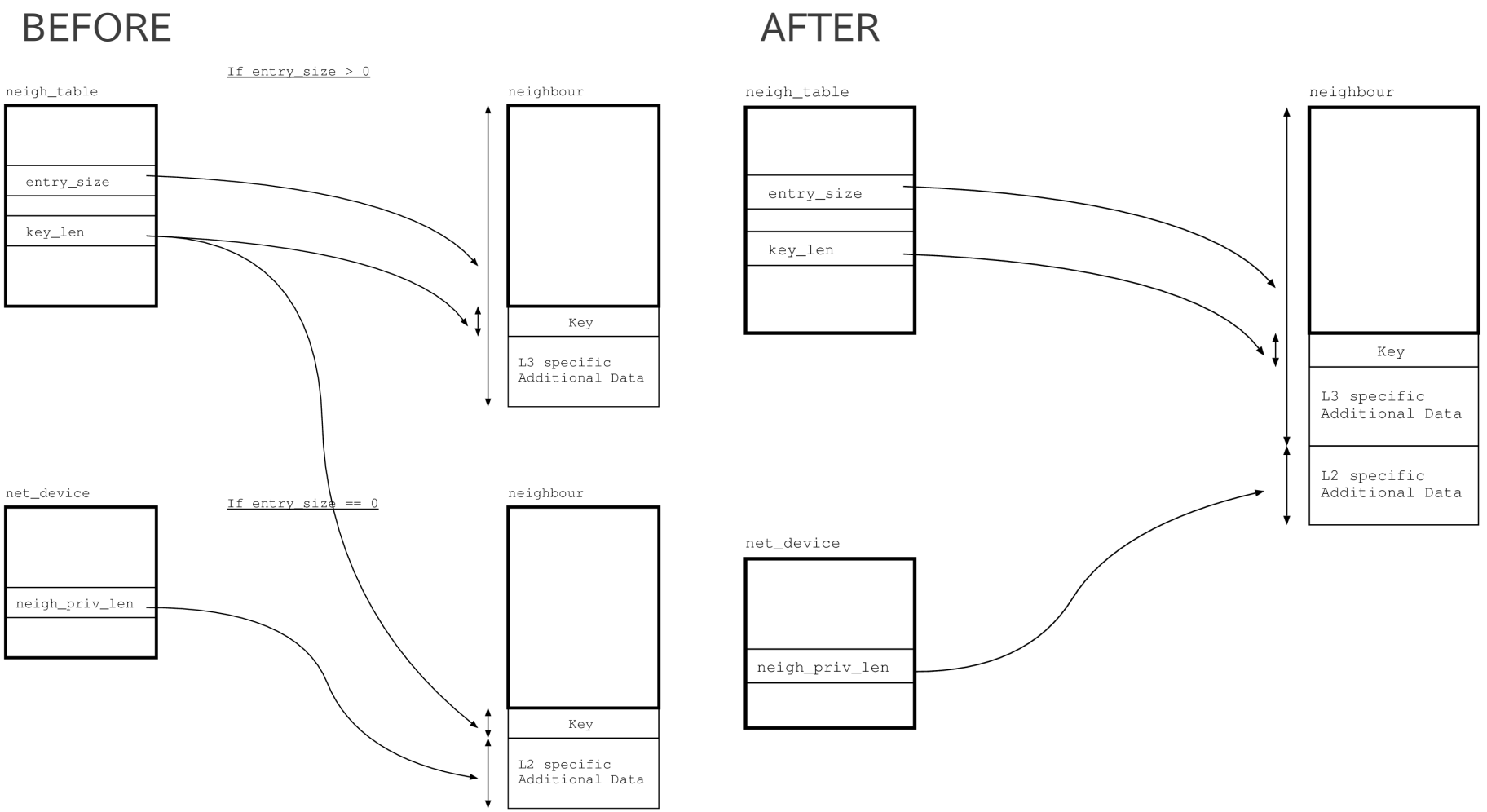
Firewire vs SEND

- Generally, driver should be simple as possible.
  - State management should be minimized.
- Even if the driver needs to maintain per-neighbor state (e.g. node-id), it may be worth to try to have such information attached to each NCE (Neighbor Cache Entry).
- At the same time, I was thinking about implementation of SEND (Securing Neighbor Discovery), which should maintain additional L3-specific neighbor information.

Separation of L2/L3-specific Neighbor Data

- Linux has unified data structure for neighbors for various L2/L3 such as:
  - L2: Ethernet, Firewire, ATM, …
  - L3: IPv4, IPv6, DECnet, …
- In addition to basic state/mapping between L2/L3, it can hold additional data specific to L2 or L3.
- Because the only users (ATM and DECnet) are exclusive (DECnet over ATM is not possible), the code assumes that L2/L3 specific data are not used at the same time.
- Problems
  - SEND (Securing Neighbour Discovery) may want to have IPv6-specific data (timestamp, key etc.)
  - Firewire or other L2 may want to have L2-specific data
  - Even without these, we always calculated size of entry size when allocating new NCE.

NCE Illustrated



Tokenized ID

- Tokenised IPv6 Identifiers <draft-chown-6man-tokenised-ipv6-identifiers-02>
- Manual “Interface Identifier”
  - e.g. Token = ::1234
  - Prefix in RA: 2001:db8::/64

    => 2001:db8::1234

Upcoming Features / Fixes

- Features
  - SEND (Securing Neighbor Discovery)
    - RFC3971, RFC3972
  - “Zero” checksum in UDP
    - RFC6936
  - Enhancement of DAD and NDP
- Improvements
  - Use of Workqueue
    - Address management
  - Conformance

SEND (Securing Neighbor Discovery)

- Designed to counter the threats to NDP (described in RFC3756)
- Applicable in environments where physical security on the link is not assured (e.g. wireless) and attacks on NDP are a concern.
- Key Idea: CGA (Cryptographically Generated Adreess)
  - RSA
- Several implementations are available, but they seem patchy.

Handing of NDP on Linux Systems

- NS/NA, RS/RA, Redirect
  - NS/NA and Redirect messages: processed by kernel
  - RA: sent by daemon, processed by kernel and daemon
  - RS: sent by kernel, processed by both of kernel and the daemon

Options

- Native (+socket)
  - Both in NDP and in the daemon
  - Or with extensions to socket interface
    - Sender side: appends required options.
    - Receiver side: provides verification information via CMSG
- XFRM
  - How to tell application the status of verification?
  - NDP message are not always “dropped” even if the messages are not verified; they are treated as “unverified” messages.
- Netfilter
  - Similar to XFRM
- Hmm...

UDP with ZERO Checksum

- RFC6936: Applicability Statement for the Use of IPv6 UDP Datagrams with Zero Checksums
  - Since IPv6 does not have L3 checksum in IP header, checksum has been mandatory for UDP.
  - In limited situations, UDP checksum might be redundant; e.g. encapsulating UDP.
  - The RFC adds knobs to send and/or receive UDP datagrams with zero checksums, if UDP w/ ZERO checksum is suppored.

Receiving of UDP with ZERO Checksums

- IPv4: CHECKSUM_UNNECESSARY is set.
- IPv6: packet is dropped immediately.
- Actual verification is done in recvmsg(2).
- We can check of new socket option and checksum of the datagram in recvmsg(2).
  - It costs some CPU cycles...

Enhancements of DAD/NDP

- Enhanced Duplicate Address Detection <draft-ietf-6man-enchanced-dad-03>
- Neighbor Unreachability Detection is too impatient <draft-ietf-6man-impatient-nud-06>
- A Method of Generating Stable Privacy-Enhanced Addresses with IPv6 Stateless Address Autoconfiguration (SLLAC) <draft-ietf-6man-stable-privacy-addresses>
- Packet loss resiliency for Router Solicitations <draft-ietf-6man-resilent-rs-01>
- Security Implications of IPv6 Fragmentation with IPv6 Neighbor Discovery <draft-ietf-6man-nd-extension-headers-04>

Workqueue

- IPv6 stack does not use workqueue/delayed work.
- Possible areas: address validation
  - like IPv4
  - Plus: temporary address management

Conformance

- Since we (or maybe I) have not actively tested recent kernel, conformance level of IPv6 stack seems worse than before...
- Some NDP functions seem broken at least
- Test infrastructure is being rebuilt.

Conclusion

- New features
  - Stateless NAT
  - IPv6 over Firewire
  - Tokenized ID
- Upcoming Features in my mind
  - SEND
  - UDP with ZERO Checksum
  - Use of Workqueue, least interactions between resources.
  - Conformance should be revisited.
  - And more...?
- I love Linux :-)

Conclusion

- New features
  - Stateless NAT
  - IPv6 over Firewire
  - Tokenized ID
- Upcoming Features in my mind
  - SEND
  - UDP with ZERO Checksum
  - Use of Workqueue, least interactions between resources.
  - Conformance should be revisited.
  - And more...?
- I love Linux :-)

Anything else?

- Free discussion