Linux IPv6 Packet Forwarding

Speaker: Whai-En Chen

2008/05/06
Implementation of IPv6 in Linux Kernel

TCP/UDP/RAW
- ip6_input_finish()
- NF_IP6_LOCAL_IN
  - ip6_mc_input()
  - ip6_input()
  - skb->dst->input(skb)
    - skb->dst->output(skb)
      - ip6_route_input()
      - ip6_rcv_finish()
      - NF_IP6_PRE_ROUTING
        - ip6_rcv()
  - NF_IP6_FORWARD
    - ip6_forward()
    - skb->dst->output(skb)
      - ip6_forward_finish()
      - ip6_output()
      - NF_IP6_POST_ROUTING
      - ip6_output_finish()

TCP
- ip6_xmit()
- NF_IP6_LOCAL_OUT
  - ip6_output()
  - ip6_output_finish()
Forwarding Path

- Step 1. ip6_rcv()
- Step 2. ip6_rcv_finish()
- Step 3. ip6_forward()
- Step 4. ip6_forward_finish()
- Step 5. ip6_output()
- Step 6. ip6_output_finish()
ip6_rcv()

- net_rx_action (net/core/dev.c)
- ip6_rcv() (net/ipv6/ip6_input.c)
- Check the error, store the incoming device interface index and increase the SNMP counter
  - Check packet (DST MAC, shared Buffer, Length)
  - SNMP: increase the count of received packets
  - skb->cb->iif = dev->ifindex; store the incoming device interface index in control buffer since we can not refer to it later.
  - Check packet length
  - Check IP version (if (hdr->version != 6))
  - process packet excluded jumbo payload
  - Process Hop-by-hop option header
- If error happens, release the buffer.
- No error: Call ip6_rcv_finish() via NF_HOOK()
ip6_rcv_finish()

- Decide route for the incoming packet
  - Call ip6_route_input(skb)
  - The function looks up routing table and obtain destination information.
    ( net/ipv6/ip6_fib.c::fib6_lookup() )

- Dispatch the packet to different functions
  - return skb->dst->input(skb)
  - dst->input may be ip6_mc_input(), ip6_input(), ip6_forward(), or ip6_pkt_discard()
  - dst->output may be ip6_output() or ip6_pkt_discard()
Construction of Routing Table

- Routing entries include addresses (prefix length=128) and prefixes (prefix length<128).
- Address (net/ipv6/addrconf.c)
  - Including unspecified address, loopback address, link-local address, and global address (site-local address are almost not used now).
  - Unspecified address is default in routing table (root node).
  - When loopback address is configured, it will be added into the routing table directly. (init_loopback())
  - Duplicate Address Detection must be performed before link-local address and global address being assigned to an interface (including addresses configured manually).
  - If DAD succeed, now the address is valid and the relating entry will be added into routing table. (addrconf_dad_completed())
Construction of Routing Table (cnt.)

- Prefix
  - Default: Multicast route (FF00::) and Link-Local route (FE80::)
    - (addrconf_add_mroute() and addrconf_add_lroute())
  - When the *Duplicate Address Detection* (DAD) procedure starts, the prefix of the IPv6 address will be added into routing table. (addrconf_dad_start())
Linux Routing Tree for IPv6

Bit 0
- rt6i_dst.addr = 0
- rt6i_dst.plen = 0
- input = ip6_pkt_discard
- output = ip6_pkt_discard

Bit 8
- rt6i_dst.addr = FF00::
- rt6i_dst.plen = 8
- input = ip6_mc_input
- output = ip6_output

Bit 10
- rt6i_dst.addr = FE80::<EUI64>
- rt6i_dst.plen = 10
- input = ip6_forward
- output = ip6_output

Bit 128
- rt6i_dst.addr = FE80::<EUI64>
- rt6i_dst.plen = 128
- input = ip6_input
- output = ip6_output
**fib6_lookup()**

- net/ipv6/ip6_fib.c::fib6_lookup()
- **Input**: root node (ip6_route_table generally), destination address and source address.
- **Output**: the corresponding fib6_node

- How to look up the routing table?
fib6_lookup()

- If the destination address matches FE80::<EUI64>
  - skb->dst->input=ip6_input
  - skb->dst->output=ip6_output
- Else if the destination address’s first 10 bits matches FE80::<EUI64>
  - skb->dst->input=ip6_forward
  - skb->dst->output=ip6_output
- Else if the destination address’s first 8 bits matches FF00::<EUI64>
  - skb->dst->input=ip6_mc_input
  - skb->dst->output=ip6_output
- Else (no match)
  - skb->dst->input=ip6_pkt_discard
  - skb->dst->output=ip6_pkt_discard

(net/ipv6/ip6_fib.c :: fib6_lookup() )
Some Tips (for Window XP)

- How to get the information about interfaces and routing table?
  - start > run > cmd
  - IPv4:
    1. route print
    2. netsh > interface > ip (type “?” to look up available commend)
  - IPv6:
    1. ipv6 install > ipv6 -v rt
    2. netsh > interface > ipv6
start > run > cmd > ipv6 -v rt

C:\Documents and Settings\Administrator>ipv6 -v rt
*:0 -> 3/2002:836b:213c:1:e0:8f08:f020:8 pref 1if+1156=1157 life 7200s/1800s, publish, no aging (manual)
*:0 -> 3/2002:c00c:cef3:1::1 pref 1if+1219=1220 life 7200s/1800s, publish, no aging (manual)
*2002::/16 -> 3 pref 1if+1000=1001 life 7200s/1800s, publish, no aging (manual)
2001:f18:113:93::/64 -> 4 pref 8 life 2591897s (autoconf)
::/0 -> 4/fe80::20e:38ff:fefe:7300 pref 256 life 1697s (autoconf)
*fe80::5efe:140.113.93.237/128 -> 2/fe80::5efe:140.113.93.237 pref 1if+4=5 life infinite (system)
*2002:8c71:5ded::8c71:5ded/128 -> 3/2002:8c71:5ded::8c71:5ded pref 1if+4=5 life infinite (system)

ff00::/8 -> 4 pref 8 life infinite (system)
fe80::250:fcff:fe65:c948/128 -> 4/fe80::250:fcff:fe65:c948 pref 4 life infinite(system)

::1/128 -> 1/::1 pref 4 life infinite (system)
ff00::/8 -> 1 pref 8 life infinite (system)
fe80::1/128 -> 1/fe80::1 pref 4 life infinite (system)
## Practical Windows XP Routing Table

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Interface</th>
<th>Metric</th>
<th>Lifetime</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>::/0</td>
<td>4</td>
<td>256</td>
<td>1697s</td>
<td>autoconf</td>
</tr>
<tr>
<td>2001:f18:113:93::/64</td>
<td>4</td>
<td>8</td>
<td>2591897s</td>
<td>autoconf</td>
</tr>
<tr>
<td>ff00::/8</td>
<td>1,4</td>
<td>8</td>
<td>Infinite</td>
<td>system</td>
</tr>
<tr>
<td>fe80:: &lt;EUI64&gt;/128</td>
<td>4</td>
<td>4</td>
<td>Infinite</td>
<td>system</td>
</tr>
<tr>
<td>fe80::/10</td>
<td>none</td>
<td>none</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>::1/128</td>
<td>1</td>
<td>4</td>
<td>Infinite</td>
<td>system</td>
</tr>
<tr>
<td>fe80::1/128</td>
<td>1</td>
<td>4</td>
<td>Infinite</td>
<td>system</td>
</tr>
</tbody>
</table>
# Linux Routing Table

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Metric</th>
<th>Lifetime</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>::/0</td>
<td>~0U</td>
<td>infinite</td>
<td>Unspecified Address (sys)</td>
</tr>
<tr>
<td>2001:f18:113:93::&lt;EUI 64&gt;</td>
<td>null</td>
<td>infinite</td>
<td>Global Address (autorconf)</td>
</tr>
<tr>
<td>2001:f18:113:93::/64</td>
<td>256</td>
<td>finite</td>
<td>Global Prefix (autorconf)</td>
</tr>
<tr>
<td>ff00::/8</td>
<td>256</td>
<td>infinite</td>
<td>Multicast Prefix (sys)</td>
</tr>
<tr>
<td>fe80:: &lt;EUI 64&gt;</td>
<td>null</td>
<td>infinite</td>
<td>Link-Local Address (sys)</td>
</tr>
<tr>
<td>fe80::/10</td>
<td>256</td>
<td>infinite</td>
<td>Link-Local Prefix (sys)</td>
</tr>
<tr>
<td>::1/128</td>
<td>null</td>
<td>infinite</td>
<td>Loopback Address (sys)</td>
</tr>
<tr>
<td>fe80::1/128</td>
<td>null</td>
<td>infinite</td>
<td>Link-Local Address (sys)</td>
</tr>
</tbody>
</table>

- Source Address is not used completely.
ip6_input_finish()

- The packet has been confirmed to deliver to upper layer.
- So do some preparations before delivering it.
  - Parse extension headers and options, but skip hop-by-hop options header since it has been already parsed.
  - Check packet capacity.
  - Sub the checksum of Network Layer for TCP.
- delivery to transport layer protocols such as TCP, UDP, ICMPv6, ...etc) or raw socket.
- If raw socket and transport protocols are all not found, send a ICMPv6 parameter problem message back.
- Note: This function has been modified after V2.5.59.
ip6_forward()

- in net/ipv6/ip6_output.c
- Check the device configuration (whether can forward packet or not).
- Check the packet and forward the Router Alert Option to upper layer
- Check Hop-limit value (should > 1)
- Redirect the packet with the same incoming and outgoing devices
- Drop the Multicast, Loop-back and Link-local packets
- Check packet length (should <Path MTU)
- Check buffer’s length
- Decrement hop-limit
- Call ip6_forward_finish() via NF_HOOK()
ip6_forward()

How did a host know the link-layer address of other host on the same link?

- IPv4: ARP
- IPv6: Neighbor Discovery

When a host does not know another host’s MAC address, it must send a NS message with the target address of the other host first.

The other host received the message and responded with a NA message.

The original host received the NA message and know the mac address of the other host from the options of NA message.
ip6_xmit() (for TCP)

- If there are requirements for filling the extension header, adjust headroom for extension header, IPv6 header and hardware header (Ethernet header).
- Fill the IPv6 extension headers and IPv6 header respectively.
- If the packet buffer’s size is too big, send a ICMPv6 Packet Too Big message.
- Else, call ip6_maybe_reroute() via NF_HOOK().
Some processes about netfilter.

If the host does not configure netfilter, call skb->dst->output(skb) directly.
ip6_output()

- ip6_forward_finish calls skb->dst->output(skb)
  - the same as skb->dst->input(skb)
  - call ip6_output() generally
- Check the packet is Multicast (from upper layer) or not.
- If the multicast is link-local or node-local Multicast address, clone the skb, call ip6_dev_loopback_xmit() via NF_HOOK() and clean the skb.
- ip6_dev_loopback_xmit() calls netif_rx() which passes the packet to upper layer again.
- Else, call ip6_output_finish() via HF_HOOK().
ip6_output_finish()

- Fill the MAC header
  - `memcpy(skb->data - 16, hh->hh_data, 16);`
- Call `hh->hh_output(skb);`
- If neighbor’s MAC address has been resolved, kernel will call `neigh_connect()` which sets `hh->output(skb)` as `dev_queue_xmit()`, preparing to send out the packet.
- Else, kernel will call `neigh_suspect()` which sets `hh->output(skb)` as `neigh_resolve_output()` and starts to resolve neighbor’s MAC address. Then the kernel will call `hh->hh_output(skb)` to send out the packet.
Neighbor Discovery

- neigh_resolve_output( ) calls neigh_event_send( )
- Neigh_event_send nested calls __neigh_event_send( )
- __neigh_event_send( ) will call ndisc_solicite( ) which will send an ICMPv6 neighbor solicitation message out to inquire neighbor’s mac address.
- The neighbor receives the message will return an ICMPv6 neighbor advertisement message with its mac address.
- Then the host can update its neighbor table according information inside the NA message.
Neighbor Discovery

- ndisc_solicite()
- ndisc_send_ns()
- icmp6_rcv()
- ndisc_rcv()
- ndisc_rcv_na()
- neigh_update()

ICMPv6 Neighbor Solicitation Message

ICMPv6 Neighbor Advertisement Message