Reprinted from the

Proceedings of the Linux Symposium

Volume Two

July 21th–24th, 2004
Ottawa, Ontario
Canada

IPv6 IPsec and Mobile IPv6 implementation of Linux

Kazunori MIYAZAWA
USAGI Project/Yokogawa Electric Corporation
`kazunori@miyazawa.org`

Masahide NAKAMURA
USAGI Project/Hitachi Communication Technologies, Ltd
`masahide_nakamura@hitachi-com.co.jp`

Abstract

USAGI Project [8] has improved Linux IPv6 [1] stack. IPv6 IPsec is one of the products of our efforts. Linux IPsec [6] stack is implemented based on XFRM architecture which is introduced in linux-2.5. We design and implement Mobile IPv6 (MIPv6) [4] Stack on the architecture. MIPv6 uses IPsec for its secure signaling. Accordingly IPv6 IPsec and MIPv6 closely cooperate each other. In this paper we describe the architecture and how they work.

1 Introduction

IPv6 is the next version of an Internet Protocol. The protocol was developed against IPv4 address exhaustion. It was developed for not only spreading address space but improving some features such as plug and play, aggregatable routing architecture, IPsec native support and smooth transition.

IPsec provides security services which are integrity, authentication, anti-replay attacks and confidentiality. Because IPsec is mandatory in IPv6 specification, we must implement IPsec to conform to it.

MIPv6 provides all IPv6 nodes with mobility service which allows nodes to remain reachable while moving around IPv6 networks. To support mobility, We need some signaling architecture to notify movement and deliver mechanisms to assure reachability. Using MIPv6, we can keep routability to mobile node's home link address and deliver a packet to mobile node wherever it is on the network. Because IPv6 is able to process these extension headers natively, we no longer need to arrange foreign agents to all links where mobile node may move to as Mobile IPv4 does, so that IP mobility is easier to be introduce in IPv6 than IPv4.

Linux supported IPsec at version 2.5.47. However it supporting only IPv4 IPsec, we implemented IPsec stack for IPv6. Linux version 2.6 supports IPsec on both IPv6 and IPv4. XFRM architecture and stackable destination were introduced into the kernel for IPsec packet processing [7]. They can be not only for IPsec packet processing, but also general packet processing such as MIPv6. USAGI Project decided to expand the architecture to implement MIPv6.

To develop Linux MIPv6, we cooperate with GO/Core Project [2] which is proven in linux-

2.4.

2 XFRM and stackable destination

XFRM architecture is mainly consist of three structures which are xfrm_policy, xfrm_state and xfrm_tmpl. xfrm_policy corresponds to IPsec policy and xfrm_state to IPsec SA. xfrm_tmpl is intermediate structure between xfrm_policy and xfrm_state. Each IPsec policy and SA database are realized with list of the structures which are also contained hash database.

The kernel provides three interface to configure xfrm structures about IPsec. One is PF_KEY interface which is standard interface to manipulate IPsec database. another is netlink socket interface. The last is socket option interface.

Stackable destination is architecture for efficient outbound packet processing. It is a link list of dst_entry structure which is cached in xfrm_policy. To create stackable destination, the kernel linearly searches xfrm_policy with flow information for a sending packet after routing looking up. After finding xfrm_policy corresponding to the flow information, the kernel searches and gathers xfrm_state from xfrm_state database by xfrm_tmpl in the xfrm_policy. Gathering xfrm_states, the kernel builds up stackable destination and substitutes it into its own member "bundles" to cache it. Additionally xfrm_policy itself is cache in flow_cache. Therefore the kernel only needs to lookup xfrm_policy after second until xfrm_state expired.

3 IPsec

IPsec functionality is consist of packet processing and key exchanging for automatic keying. In the implementation of Linux packet processing runs in the kernel and key exchange is done by a key exchange daemon in user space.

3.1 IPsec database and packet processing

IPsec packet processing is realized with XFRM architecture and stackable destination. Outbound process is explained in previous section. With searching XFRM database and building stackable destination, the kernel gets list of dst_entry structure. To process each function which are ah6_output, esp6_output and ipcomp6_output, the kernel searches insertion point on a packet because a packet is created including IPv6 header and other extension headers before stackable destination process (Figure 1). The insertion point is before upper layer payload, fragmentable destination options header, IPsec header or fragment header. This is not efficient because the kernel searches the insertion point every time when processing one dst_entry.

Inbound process is simpler than outbound process. When packet containing AH or ESP, the kernel finds xfrm_state corresponding to received packet and keep pointers of used xfrm_state in sec_path of skb structure. After process of IP layer, the kernel checks the packet correctly processed with comparing sec_path and xfrm_policy which is searched with flow information of the packet (Figure 2).

3.2 Interface for user and IKEd

Current linux kernel provides users with PF_KEY interface, which however is specified only for IPsec SA interface and it needs some extension to configure IPsec policy. Because this extension is not standardized, there are some different extensions and it prevents compatibility of IKEd. Linux adopts the extension which is compatible with KAME [5] so that racoon is the IKEd for linux. Racoon is originally product of KAME project and its could not compile on Linux. Fortunately

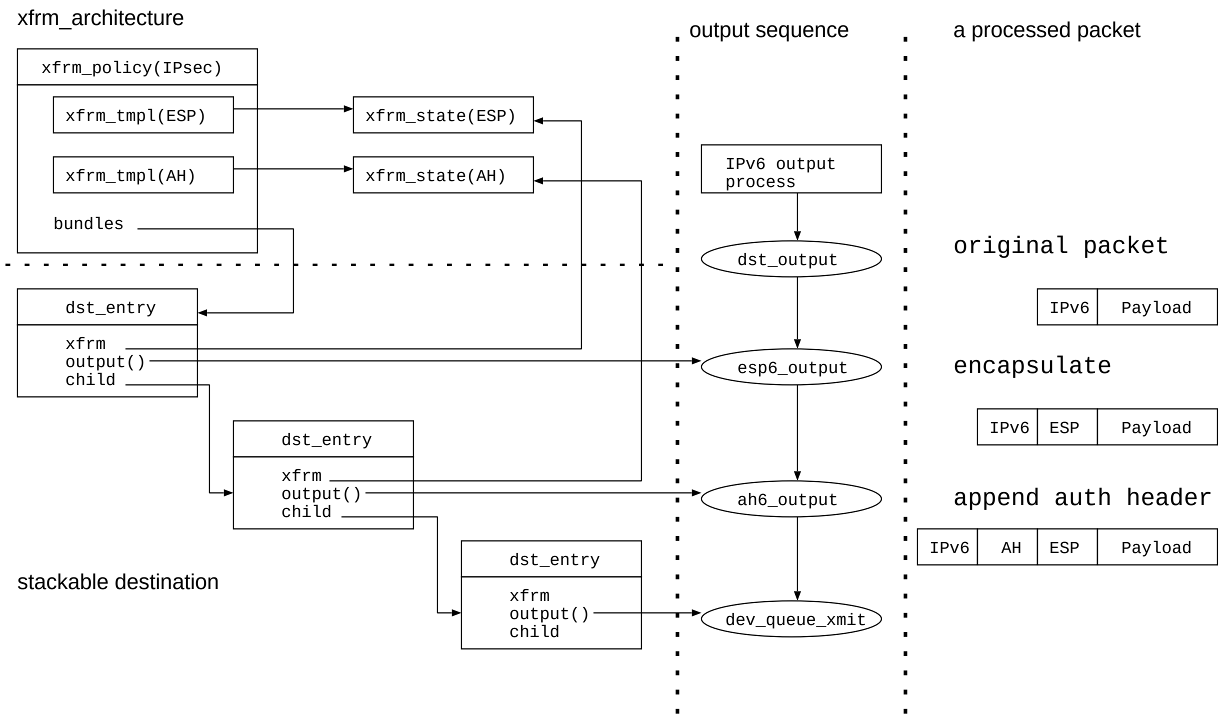

Figure 1: IPsec output process

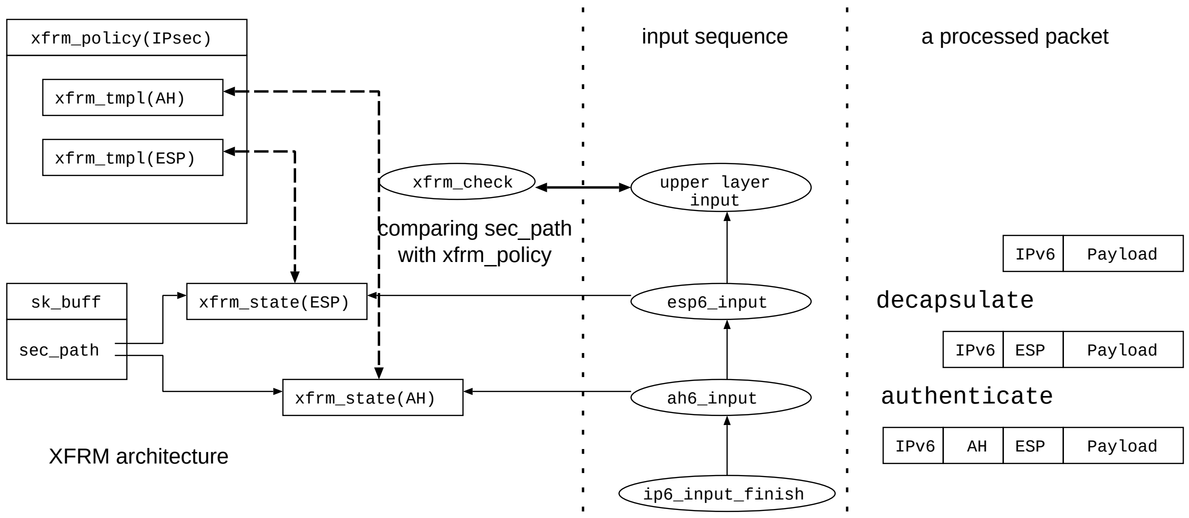

Figure 2: IPsec input process

ported racoon which is provided by ipsec-tools project [3] is available.

4 Mobile IPv6

4.1 Mobile IPv6

In MIPv6, nodes are classified into 3 types. One is a Mobile Node (MN) which moves in the IPv6 Internet bringing its home address (HoA) assigned in a home link which is a base of mobility and in which there is a home agent. Home agent (HA) is another type of node which is a router and manages MN's addresses and supports its signaling and ensures reachability. The other is a correspondent node (CN) which is a node communicating with a MN. CN may be either mobile or stationary.

When MN in a foreign link, it uses a care-of address (CoA) which is the address of a foreign link. MIPv6 accordingly needs to manage relationship between CoA and HoA. A MN sends a packet including HoA in an extension header from CoA.

MIPv6 appends two extension headers and one option for destination options header. Mobility Header (MH) is an extension header for signaling to manage binding cache which is a address list for optimized routing. Type2 routing header (RT2) which is different from routing header in RFC2460 effects destination address in IPv6 header and realizes direct routing according to binding cache. Home Address Option (HAO) is an option carried by destination options header to contain HoA which is an address of a MN in home link and swapped with CoA. HAO effects source address in IPv6 header.

We describe an outline of the procedure taking as an example that MN making binding cache on HA and communicating CN after MN moving to a foreign link (Figure 3). This procedure is divided two steps. First is making IPv6 over IPv6 tunnel between MN and HA (1-4). After this step, HoA of MN becomes routable and MN is able to communicate with all nodes by using HoA via HA through the tunnel. Second is route optimization between MN and CN because MN always communicating via HA (5-8), a packet goes through a superfluous route and communication uses more network resource.

1. MN sends a Binding Update (BU) to HA.
2. HA updates a binding cache and returns Binding Acknowledgment (BA) to MN.
3. MN updates a binding update list.
4. At this time, there is a tunnel between MN and HA.
5. MN sends HoTI to CN through the tunnel and CoTI to CN directly from CoA.
6. CN keeps contents of HoTI and CoTI. CN returns HoT via HA and CoT to CoA.
7. When MN receives HoT and CoT, MN sends BU to CN and updates its own binding list.
8. Then MN and CN have binding between HoA and CoA. They communicate directly with appending HAO and RT2 to packets. They have an optimized route.

4.2 Implementation

We design MIPv6 in Linux consisted with two part. One is packet processing for RT2 and HAO in the kernel and the other is MIPv6 daemon (MIPd) to handle the signaling and manage binding cache and binding update list. It is similar to separation of packet process and IKEd in IPsec.

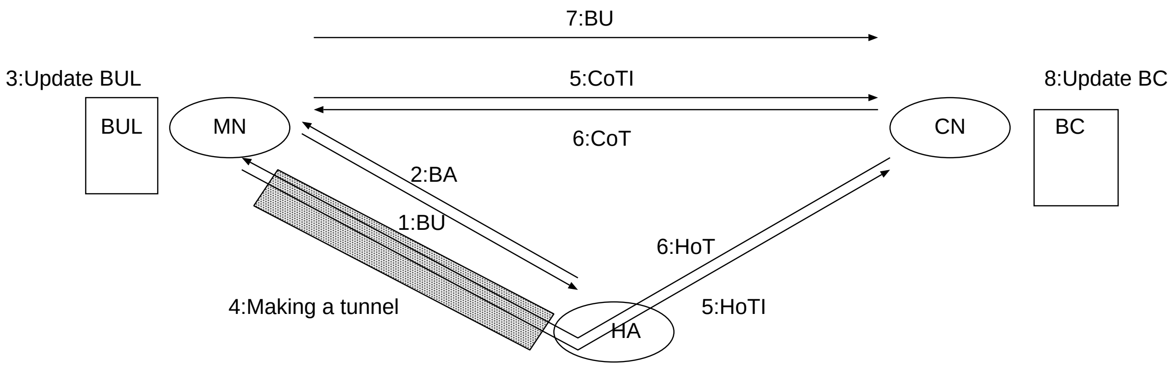

Figure 3: MIPv6 procedure outline

Packet processing for MIPv6 is realized with XFRM and stackable destination architecture, because they are general way to process a packet which matches some selector. Using XFRM, we can avoid to implement duplicate functionality in the kernel. MIPv6 needs to manage a binding cache which specifies an MN address on the network on CN and HA. It also needs to manage a binding update list which is list of sending binding update request for CN on MN. We have two choices to implement this functionality in the kernel or userland. Because we should implement functionalities in userland if it is possible, we consider to basically implement it in userland. Implementing in userland brings us advantages which are easier extension its functionality than implementing in the kernel and reducing the kernel size.

Our MIPd's roles are

- processing a signaling message including an error message
- managing xfrm_policy and xfrm_state of MIPv6 in the kernel through the netlink
- managing binding cache and binding update list
- moving detection and changing CoA when MIPd running on MN

4.3 XFRM operation

In this section, we describe MIPd XFRM operation relating each nodes state with an example which is a phase of binding update to HA and making tunnel for routability. It is called home registration. At first, we initialize MN and HA to send and receive binding message. On MN MIPd sets a xfrm_policy which allows an outbound packet from HoA to HA, proto MH, and type BU with appending HAO and a xfrm_state which appends HOA with CoA to a packet from HoA to HA and including MH of BU. It also set xfrm_policy to receive BA, the policy which allows an inbound packet from HA to HoA including MH of BA with appending RT2 and the inbound xfrm_state which processes RT2. Because MIPd on HA can not expect the source address of BU from MN, it sets a xfrm_policy which allows an inbound packet from Any to HA with MH of BU if it has HAO. It also set xfrm_state which processes HAO included in a packet from ANY to HA with MH of BU. See Figure 6:INITIALIZE.

MIPd on MN sends BU to HA, the packet matches with the xfrm_policy and process with the xfrm_state which appends HAO destination option and swap a source address in IPv6 header with a CoA. HA received the BU from MN. In the kernel the packet matching the xfrm_state, the kernel swaps addresses. Then

MIPd on HA receives BU and updates a binding cache. MIPd configures xfrm_policy and xfrm_state for route optimization with high priority. See Figure 6:Routing Optimization.

At this moment, route optimization is available for all packets between MN and HA. It also sets up a tunnel between MN and HA. After some xfrm_policy and xfrm_state configuration it returns BA with RT2. The kernel of MN receives BA with RT2 and processes it with the inbound xfrm_state and throws up BA packet to MIPd. MIPd on MN updates a binding update list and sets up the tunnel. Each nodes has totally 6 policies at the end of registration.

5 Cooperation of IPsec and MIPv6

MIPv6 uses IPsec for its secure signaling between MN and HA. Our design uses XFRM and stackable destination for both IPsec and MIPv6. MIPv6 needs two kind of IPsec SA one is a transport mode SA which is used for signaling. The other is a tunnel mode SA which is used instead of IPv6 over IPv6 tunnel. We consider two steps to implement MIPv6 with IPsec about IPesc policy and SA management. At first, we implement MIPd to not only manage xfrm_policy and xfrm_state of MIPv6 but also IPsec and a xfrm_policy for MIPv6 holds both MIPv6 and IPsec xfrm_tmpl. This implementation has a couple of issues. One is separation of management of xfrm_policy and xfrm_state of IPsec into MIPv6 and ordinary IPsec. Another issue is interaction between the kernel and IKE daemon. xfrm_policy including a xfrm_tmpls of Mobile IPv6 and IPsec sends a signal for only MIPd. The other is the order of xfrm_policy. When some situation such as configuration done with wrong order, a packet which would be originally applied MIPv6 and IPsec not be applied only IPsec.

For improvement, we will let the kernel hold two xfrm databases and mediate them because it is difficult to manage xfrm_tmpl in a xfrm_policy via userland interface by two management daemons and the xfrm_policies have probably different granularity (Figure 7). In current outbound process, the kernel looks up single xfrm_ policy database and gets a xfrm_policy which includes xfrm_tmpl for IPsec and xfrm_tmpl for MIPv6. However we will change the kernel to separately look up IPsec and MIPv6 xfrm databases and create temporary xfrm_policy which holds xfrm_tmpl gathered from each xfrm_policy. The list of xfrm_tmpl must be serialized as the order of packet processing. For instance, the kernel must put xfrm_state for AH at the end of the list. For inbound process, it is not so difficult, the kernel processes a packet by using xfrm_state which is searched and needs to check sec_path in skb against each xfrm_policy. To make it be efficient, the kernel should use flow_cache for inbound process.

If we could merge two policies correctly, we have another issue. MIPv6 needs two IPsec SA between NM and HA. One is a transport mode SA for signaling and the other is a tunnel mode SA for other packet. Taking outbound SA as an example, a transport mode SA is applied by the policy whose selector is from HoA to HA and protocol MH. On the other hand a tunnel mode SA is applied by the policy whose selector is from HoA to ANY and protocol ANY. The packet should be applied the transport mode SA has possibility to be applied the tunnel mode SA. We can avoid this mismatch by using priority in xfrm_policy.

racoon has a couple of issues as IKE daemon for MIPv6. One is that racoon can not handle multiple peers which have address ANY as peer's address in its configuration. When it behaves as responder on HA, the issue occurs because despite multiple peers being, each configuration has addresses from ANY to HA thus

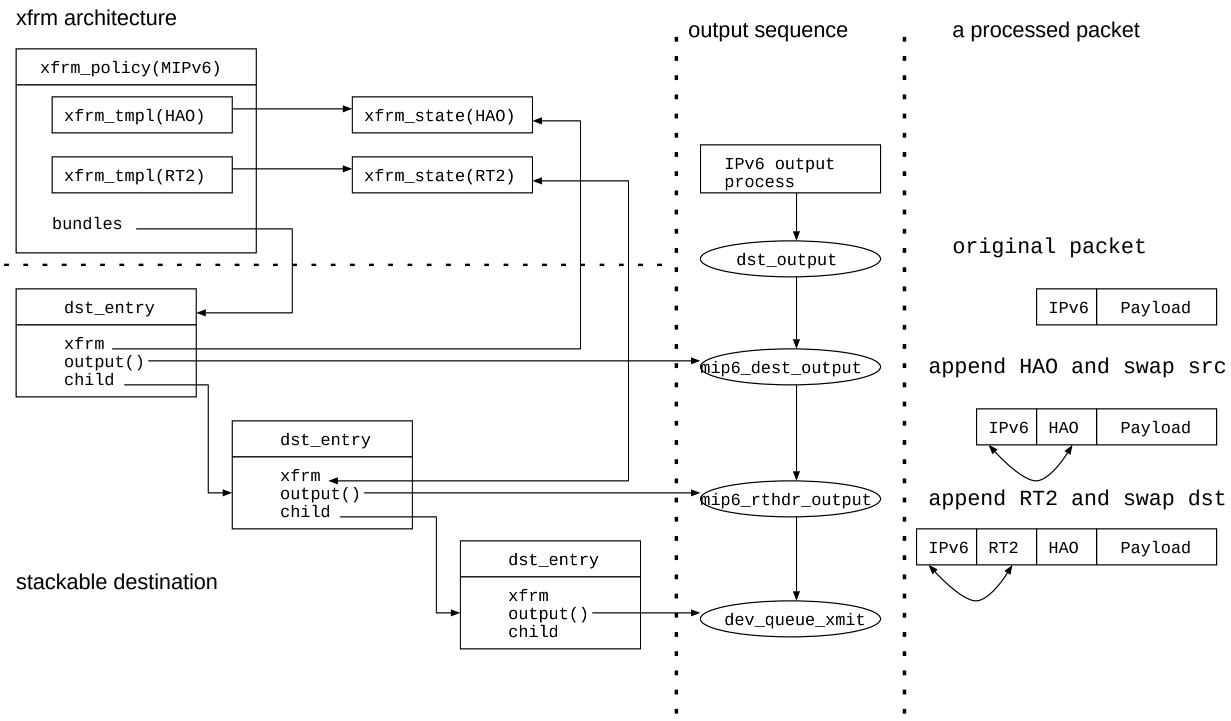

Figure 4: MIPv6 output process

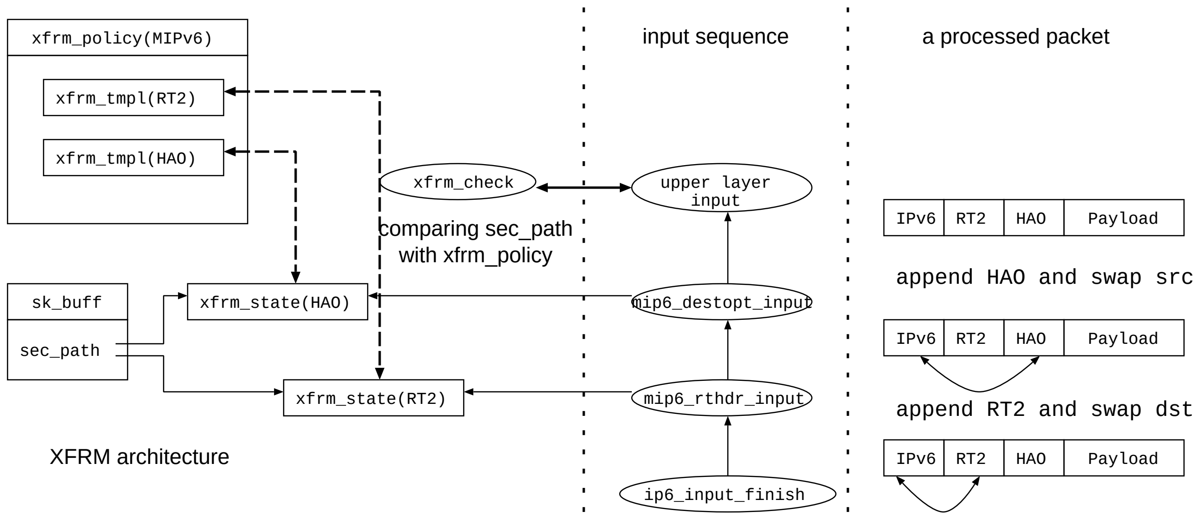

Figure 5: MIPv6 input process

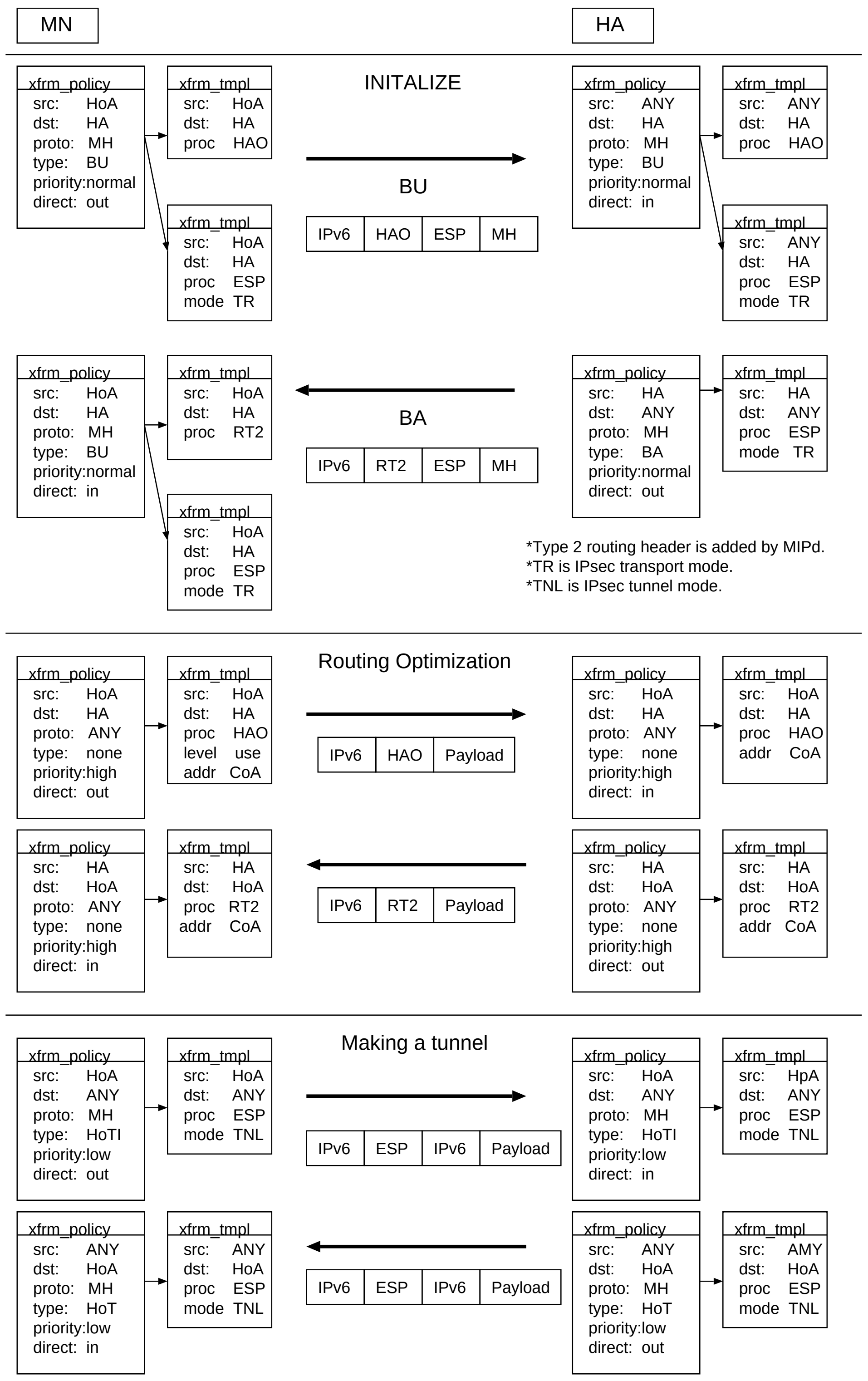
MN
HA
INITALIZE
xfrm_policy src: HoA dst: HA proto: MH type: BU priority:normal direct: out
xfrm_tmpl src: HoA dst: HA proc HAO
xfrm_tmpl src: HoA dst: HA proc ESP mode TR
BU
IPv6 HAO ESP MH
xfrm_policy src: ANY dst: HA proto: MH type: BU priority:normal direct: in
xfrm_tmpl src: ANY dst: HA proc HAO
xfrm_tmpl src: ANY dst: HA proc ESP mode TR
xfrm_policy src: HoA dst: HA proto: MH type: BU priority:normal direct: in
xfrm_tmpl src: HoA dst: HA proc RT2
xfrm_tmpl src: HoA dst: HA proc ESP mode TR
BA
IPv6 RT2 ESP MH
xfrm_policy src: HA dst: ANY proto: MH type: BA priority:normal direct: out
xfrm_tmpl src: HA dst: ANY proc ESP mode TR
*Type 2 routing header is added by MIPd.
*TR is IPsec transport mode.
*TNL is IPsec tunnel mode.
Routing Optimization
xfrm_policy src: HoA dst: HA proto: ANY type: none priority:high direct: out
xfrm_tmpl src: HoA dst: HA proc HAO level use addr CoA
IPv6 HAO Payload
xfrm_policy src: HoA dst: HA proto: ANY type: none priority:high direct: in
xfrm_tmpl src: HoA dst: HA proc HAO addr CoA
xfrm_policy src: HA dst: HoA proto: ANY type: none priority:high direct: in
xfrm_tmpl src: HA dst: HoA proc RT2 addr CoA
IPv6 RT2 Payload
xfrm_policy src: HA dst: HoA proto: ANY type: none priority:high direct: out
xfrm_tmpl src: HA dst: HoA proc RT2 addr CoA
Making a tunnel
xfrm_policy src: HoA dst: ANY proto: MH type: HoTI priority:low direct: out
xfrm_tmpl src: HoA dst: ANY proc ESP mode TNL
IPv6 ESP IPv6 Payload
xfrm_policy src: HoA dst: ANY proto: MH type: HoTI priority:low direct: in
xfrm_tmpl src: HpA dst: ANY proc ESP mode TNL
xfrm_policy src: ANY dst: HoA proto: MH type: HoT priority:low direct: in
xfrm_tmpl src: ANY dst: HoA proc ESP mode TNL
IPv6 ESP IPv6 Payload
xfrm_policy src: ANY dst: HoA proto: MH type: HoT priority:low direct: out
xfrm_tmpl src: AMY dst: HoA proc ESP mode TNL

Figure 6: Binding update procedure to Home Agent

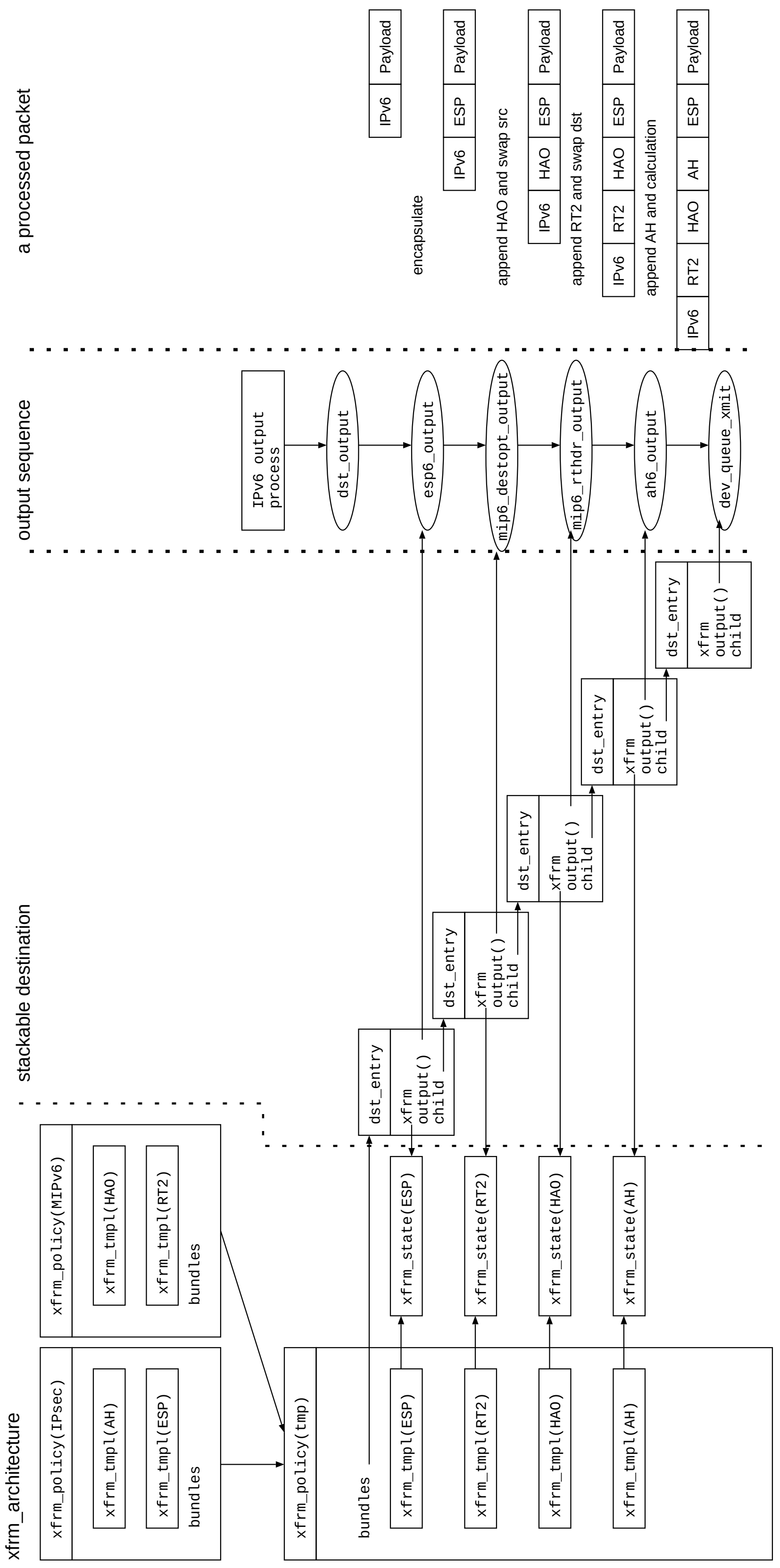

Figure 7: MIPv6 and IPsec output process

racoon can not distinct peer and fails to search proper key. The other issue is update ISAKMP SA end-point address. When MN moves, IKEs on MN and HA need to detect movement in some way and update its ISAKMP SAs because an address of those SAs is CoA. To solve these issues, we will make racoon handle the multiple peers listen netlink socket for the detection and make the kernel notify address changing via netlink socket.

6 Summary

USAGI Project implements IPv6 IPsec and MIPv6 by using XFRM and stackable destination architecture. In this paper we describe our design, implementation and issues. We also describe future design of IPv6 IPsec and MIPv6 which improves flexibility of xfrm configuration.

7 future work

Our future works about MIPv6 are

- implement our new design
- make racoon support MIPv6
- NEMO
- Multihome
- vertical hand-over

Additionally we consider that we should improve or change stackable destination itself because stackable destination runs after building a packet. Thus, IPv6 packet processing is not efficient itself because an IPv6 packet has some extension header and the order of headers is not always same as the order of process so that every process searches correct point on a packet from the head. We should improve its packet processing with keeping xfrm architecture and cache mechanism.

References

[1] S. Deering and R. Hinden. Internet Protocol, Version 6 Specification. RFC2460, December 1998.

[2] GO/Core Project. MIPL Mobile IPv6 for Linux. `http://www.mobile-ipv6.org`.

[3] IPsec Tools. IPsec Tools Web Page. `http://www.ipsec-tools.sourceforge.net/`.

[4] D. Johnson, C. Perkins, and J. Arkko. Mobility Support in IPv6. Work in Progress, June 2003.

[5] KAME Project. KAME Project Web Page. `http://www.kame.net`.

[6] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. RFC2401, November 1998.

[7] Kazunori Miyazawa, Hideaki Yoshifuji, and Yuji Sekiya. Linux IPv6 Networking—Past, Present, and Future. In *Proceedings of the Linux Symposium*, Ottawa, July 2003.

[8] USAGI Project. USAGI Project Web Page. `http://www.linux-ipv6.org`.