SHISA: The Mobile IPv6/NEMO BS Stack Implementation Current Status

Keiichi Shima, Internet Initiative Japan Inc., Japan, keiichi@iijlab.net
Koshiro Mitsuya, Keio University, Japan, mitsuya@sfc.wide.ad.jp
Ryuji Wakikawa, Keio University, Japan, ryuji@sfc.wide.ad.jp
Tsuyoshi Momose, NEC Corporation, Japan, momose@az.jp.nec.com
Keisuke Uehara, Keio University, Japan, kei@wide.ad.jp

Abstract

Mobile IPv6 and Network Mobility Basic Support (NEMO BS) are the IETF standard mobility protocols for IPv6. We implemented these protocols and we call the implementation SHISA. SHISA supports most of the features in these mobility protocol specifications and has high level interoperability with other stacks compliant to the specifications. We are now working towards adapting the SHISA code to fit the latest BSD source tree. In this paper we explain the detailed implementation design of the stack, current status of the porting work and the future plans of our project.

1 Introduction

The rapid growth of the IPv4 Internet raised a concern of the IPv4 address exhaustion. IPv6 was designed as the essential solution of the problem. We are now on the transition period from the IPv4 Internet to the IPv6 Internet. As a result of the transition, a vast number of IPv6 devices connected to the Internet using various communication technologies will appear in the future. The devices will not only be computers and PDAs but also cars, mobile phones, sensor devices and so on. Since many devices will potentially move around changing its point of attachment to the Internet, mobility support for IPv6 is considered necessary. The IETF has discussed the protocol specification and finally standardized two IPv6 mobility protocols, Mobile IPv6 [1] for host mobility and Network Mobility Basic Support (NEMO BS) [2] for network mobility.

When we deploy a protocol, it is one of the efficient ways to provide the protocol stack as open source software. The developers of the protocol stack can get many feedback from worldwide users and can enhance their implementation. We implemented the mobility protocol stack, called *SHISA*[1] [3, 4], that supports both Mobile IPv6 and NEMO BS to provide a full featured mobility stack on top of BSD operating systems as a part of the KAME project activity [5], and released the stack as open source software. After the KAME project concluded in March 2006, we started to adapt the stack to fit the latest BSD tree aiming to merge the mobility code.

This paper presents the current status of our work on IPv6 mobility and future plans. We will provide the basic knowledge of Mobile IPv6 and NEMO BS in Section 2 and discuss the design principle and implementation detail in Section 3 and 4. Section 5 discusses the remaining stuffs to be designed and implemented to give advanced mobility features and also discusses the future plans of our project. Section 6 concludes this paper.

2 Mobile IPv6 and NEMO BS Overview

Mobile IPv6 is a protocol which adds a mobility function to IPv6. Figure 1 illustrates the operation of Mobile IPv6. In Mobile IPv6, a moving node (*Mobile Node*, *MN*) has a permanent fixed address which is called a *Home Address* (*HoA*). HoAs are assigned to the MN from the network to which the MN is originally attached. The network is called a *Home Network*. When the MN moves to other networks than the home network, the MN sends a message to bind its HoA and the address assigned at the foreign network. The message is called a *Binding Update* (*BU*) message. The address at the foreign network is called a *Care-of Address* (*CoA*) and the networks other than the home network are called *Foreign Networks*. The message is sent to a special node, called a *Home Agent* (*HA*) located in the home network. The HA replies to the MN with a *Binding Acknowledgement* (*BA*) message to confirm the request. A bi-directional tunnel between the HA and the CoA of the MN is established after the binding information has been successfully exchanged. All packets sent to the HoA of the MN are routed to the home network by the Internet routing mechanism. The HA intercepts the packets and for-

[1] SHISA was named after a traditional roof ornament in Okinawa Japan, where we had the first design meeting.

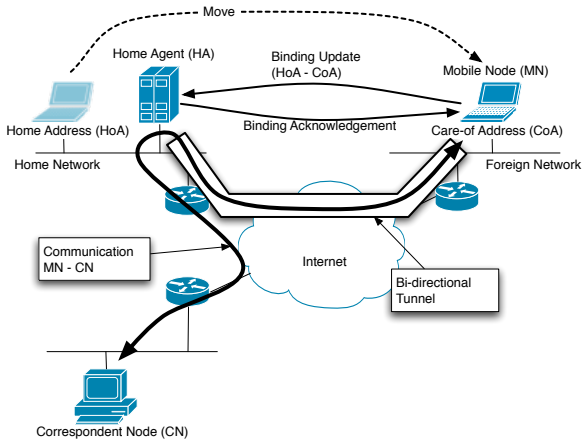

Figure 1: Basic Operation of Mobile IPv6.

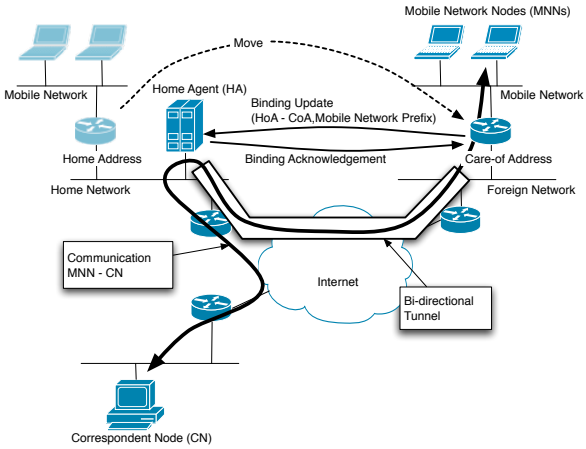

Figure 2: Basic Operation of NEMO BS.

wards them to the MN using the tunnel. Also, the MN sends packets using the tunnel when communicating with other nodes. The communicating nodes (called as *Correspondent Nodes*, *CN*) do not need to care about the location of the MN, since they see the MN as if it is attached to the home network.

In Figure 1, the communication path between the MN and its peer node is redundant since all traffic is forwarded through the HA. Mobile IPv6 allows an MN to optimise the path to an IPv6 node which is aware of the Mobile IPv6 protocol by sending a BU message. When an MN send a BU message to a CN, the MN must perform a simple address ownership verification procedure called *Return Routability* (*RR*). The MN sends two messages (*Home Test Init* (*HoTI*) and *Care-of Test Init* (*CoTI*) messages) to the CN, one from its HoA and the other from its CoA. The CN responds these two messages with *Home Test* (*HoT*) and *Care-of Test* (*CoT*) messages with cookie values. The MN then generates secret information using these two cookies and sends a BU message cryptographically protected with the secret information. Once the CN accepts the BU message, the MN can directly send a packet to the CN from its CoA. To provide the HoA information to the CN, the MN stores its HoA in a Destination Options Header as the *Home Address option* (*HAO*). The option is newly defined in the Mobile IPv6 specification. The CN can also directly send a packet to the MN using the *Routing Header Type 2* (*RTHDR2*), which is a new type of a routing header. This direct path is called a *Route Optimized* (*RO*) path.

NEMO BS is an extension of Mobile IPv6. The basic operation of a moving router (*Mobile Router*, *MR*) is same as that of an MN except the MR has a network (*Mobile Network*) behind it. The network prefix is called a Mobile Network Prefix (*MNP*). A node in the mobile network, which is called a *Mobile Network Node* (*MNN*), can communicate with other nodes as if they are attached to the home network, thanks to the tunneling between the HA and the MR. NEMO BS does not provide the RO feature. Figure 2 depicts the operation of NEMO BS.

3 SHISA Design

Mobile IPv6 and NEMO BS are layered between the Network Layer and Transport Layer. The first version of our mobility stack, known as the KAME Mobile IPv6 stack, was implemented as a part of the kernel as other Network Layer and Transport Layer protocols.

When the Mobile IPv6 specification was published as an RFC, we were considering to extend the features of the mobility stack. We thought it would not be a good idea to keep all mobility functions in the kernel, considering its extensibility and maintainability[2]. We redesigned the entire stack and moved most of the protocol functions to user space. In the process of redesign, we also referred the basic design of another Mobile IPv6 stack (SFCMIP [7]) for BSD that was being developed at Keio University. The remaining functions in the kernel was packet forwarding processing. All the mobility signal processing and binding information management processing were moved to user space. The design gives us the following benefits.

- Easy development and maintenance: Since the signaling processing of Mobile IPv6 and NEMO

[2]There was a separate project that provided a NEMO BS implementation [6] based on the KAME Mobile IPv6 stack, which was also implemented in the kernel.

BS is complicated, it is better to implement it in user space. We can develop and debug the complex part of the protocol easier than doing it in the kernel, without reducing packet forwarding performance.

- Extensibility for additional features: Developing user space programs is easier than the kernel programming in most cases and for most users. Moving the core mobility implementation from the kernel to user space will encourage third party developers to add new features.
- Minimum modification of the kernel code: When considering to merge the developed code into BSD trees, the smaller amount of kernel modification is the better. Moving signaling part to the user space reduces the amount of kernel modification.

In the user space, we also divided the entire stack into 6 pieces as follows.

- MN functions
- MR functions
- HA functions
- RO responder functions
- Movement detection functions
- NEMO BS tunnel setup functions

The design allows users to chose only necessary components when they build mobility aware nodes. For example, if one wants to build an MN that does not act as an RO responder, he can disable it. The design also allows to replace components with their own implementation. Especially, the ability to replace the movement detection mechanism is useful when deploying mobility services in a specific network infrastructure that supports a good movement detection mechanism, such as the Layer 2 movement notification scheme. In that case, the system integrator can create a special movement detection program, keeping other signaling processing code untouched.

The components, including the kernel, communicate each other through a newly designed socket domain dedicated to mobility information exchange. When a user program put some information (e.g. binding information) to the kernel, this socket domain is used. The socket domain is used to exchange such information between user space programs too. It can also be used as a notification mechanism from the kernel to user space programs. When the kernel has to notify information that can only be retrieved inside the kernel, such as extension header processing errors or tunneled packet input events, the kernel writes the event information to the socket domain so that all the listening programs of the socket in user space can receive the event data.

4 Implementation

SHISA was originally developed on top of the KAME IPv6 stack [9] for NetBSD 2.0 and FreeBSD 5.4. We ported SHISA to the NetBSD-current tree as the first step of porting effort. There are two reasons why we chose NetBSD as the first platform for the porting work. The first reason is that it supports various kinds of architectures. The mobility functions are useful especially when it is integrated to a moving entities such as PDAs and cars or trains, and so on. They usually use an architecture that runs with limited resources. NetBSD supports many such architectures that is suitable for embedded use, and we wanted to realize such small devices using our code. The other reason is the difference between the KAME tree (that was based on NetBSD 2.0) and the latest NetBSD is relatively small compared to other BSD variants that KAME supported. This makes it easier to port the SHISA code from KAME to NetBSD-current.

Figure 3 shows the relationship of the SHISA modules. The objects with solid lines are newly implemented modules. The dotted line objects exist in the original BSD system and the shaded ones of them have been modified for the SHISA system.

There are 6 user space programs; **mnd**, **babymdd**, **cnd**, **mrd**, **nemonetd** and **had**. Each program handles, the MN signaling messages, the movement detection procedure, the RO responder signaling messages, the MR signaling messages, the tunnel setup procedure for NEMO BS, and the HA signaling messages respectively. The binding database that corresponds the HoA and CoA of an MN is maintained by **mnd** and **mrd** on the MN/MR side, and by **cnd** and **had** on the CN/HA side. The subset information of the databases that is necessary for the packet input and output processes in the kernel is injected by these programs using the Mobility socket discussed in Section 4.1.

The communication interface used between the kernel and the user space programs, and between the user space programs is provided by the newly implemented Mobility socket domain (AF_MOBILITY). The mechanism and message formats used in the domain are similar to the Routing socket [10]. Unlike the Routing socket, we use this socket to exchange

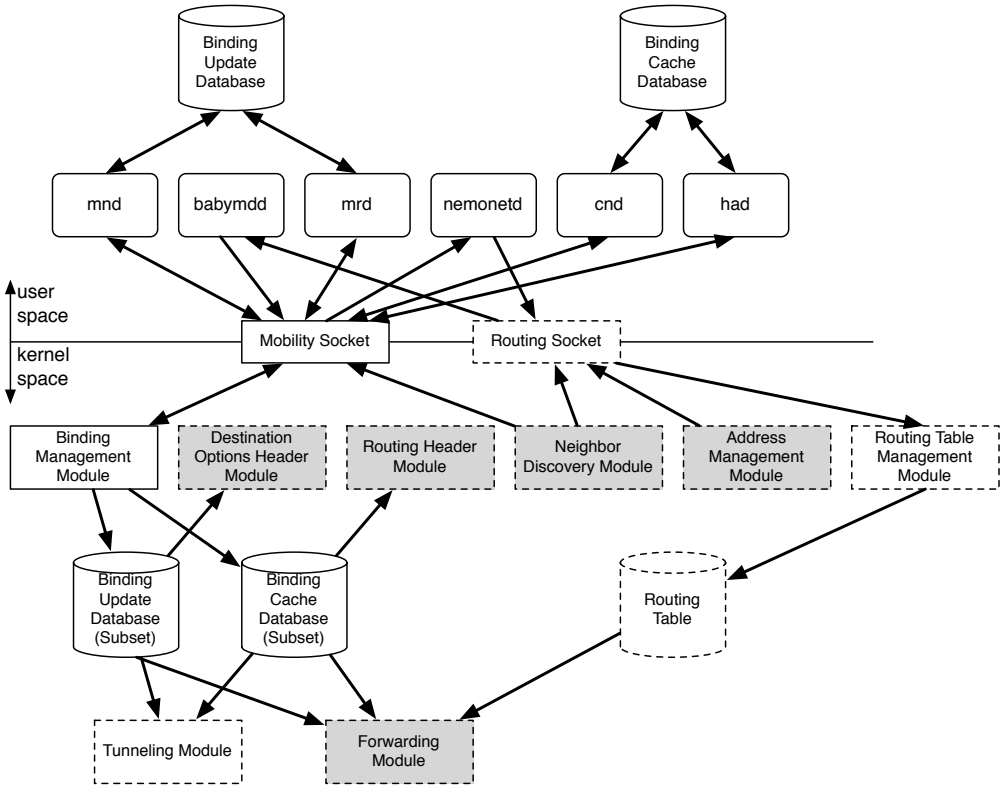

Figure 3: The relationship of the SHISA modules.

mobility related information even between user space programs. For example, the movement detection program (**babymdd**) uses this socket to notify other programs of movement events through the socket when it considers the node is attached to a new network. The socket is also used as a broadcasting channel from the kernel to user space programs. For example, when the kernel of a mobile host receives a tunneled packet from a correspondent node, it notifies the **mnd** program of the fact so that it can start the route optimization procedure. Such information is not usually available from the user space.

We created two new pseudo interfaces, `mip` and `mtun`. The `mip` interface represents the home network of an MN/MR and keeps HoAs of the node. If an MN/MR has more than one home network, the node will have multiple `mip` interfaces. The `mtun` interface is used as a tunnel interface between an MR and its HA. The interface is basically a copy of the `gif` interface with some extension to keep the next-hop information of the interface. The `mtun` interfaces are controlled by the **nemonetd** program based on the signaling messages exchanged between an MR and an HA by monitoring the Mobility socket. The `mip` and `mtun` interfaces are discussed in Section 4.2 and 4.3 respectively.

Mobile IPv6 and NEMO BS extended IPv6 extension headers. These protocols use a new destination option (HAO) and a new routing header type (RTHDR2). The processing code is implemented by extending the existing extension header processing code in the kernel, because these headers cannot be handled in user space. The normal packets, that are not mobility signal messages, are automatically processed based on the binding information stored in the kernel by the extended processing code. The signaling packets are sent and received by the user space programs using the socket API specified in RFC4584 [11].

4.1 Mobility Socket: AF_MOBILITY

The Mobility socket [8] is implemented as a variant of the raw sockets. The usage of this socket is similar to that of the Routing socket. The mobility socket can be opened as follows.

```
s = socket(AF_MOBILITY, SOCK_RAW, 0);
```

At the this moment, there are 12 message types as shown in Table 1.

Type	Description
NODETYPE_INFO	Set or reset the operation mode (MN, MR, HA or CN).
BC_ADD	Add a binding cache entry.
BC_REMOVE	Remove a binding cache entry.
BC_FLUSH	Remove all binding cache entries.
BUL_ADD	Add a binding update list entry.
BUL_REMOVE	Remove a binding update list entry.
BUL_FLUSH	Remove all binding update list entries.
MD_INFO	A hint message that indicates the movement of an MN.
HOME_HINT	A hint message from the kernel that notifies returning home of an MN from the kernel.
RR_HINT	A hint message from the kernel that indicates receiving or sending a bi-directional packet.
BE_HINT	A hint message from the kernel that an error message has to be sent due to protocol processing error in the kernel.
DAD	Request the kernel to perform the DAD (Duplicate Address Detection) procedure for a specific address.

Table 1: The Mobility socket message types.

The NODETYPE_INFO message enables (or disables) mobility functions in the kernel. The user space programs issue this message to enable (or disable) specific mobility processing code in the kernel, for example, the **mnd** program issues this message to enable mobile node functions in the kernel such as the binding update list management and the extension header processing. The BC_* messages are used by the **had** and **cnd** programs to add or remove binding cache entries in the kernel. The BUL_* messages are used for binding update list entries by the **mnd** and **mrd** programs similarly. The MD_INFO message is issued by the **babymdd** program to notify the node movement of the **mnd** or **mrd** program. As discussed earlier, any system integrator can prepare their specific movement detection program that issues the MD_INFO message for better or optimized performance of the node movement. The *_HINT messages are issued by the kernel to notify the events that cannot be obtained in user space of user space programs. The HOME_HINT message is issued when an MN/MR returns home by comparing received prefix information in a Router Advertisement message and the configured home network prefix. When receiving this message, the MN/MR stops mobility functions.

Flag	Description
IN6_IFF_HOME	The address is an HoA.
IN6_IFF_DEREGISTERING	The address is being de-registered.

Table 2: The address flags used by an HoA.

The address information in these messages are stored in the form of the sockaddr structure so that any kind of address family can utilize this socket mechanism. IPv6 is the only supported address family at this moment.

4.2 The mip Interface and Home Address

The mip interface represents the home network of an MN/MR. This interface is used to keep the HoAs of an MN/MR when the node is in foreign networks. The HoAs are assigned to the physical network interface attached to the home network of the MN/MR while it is at home. However the physical interface is used to attach to a foreign network when the node leaves from the home network. In this case, the HoAs are moved from the physical interface to the mip interface.

The address assigned as an HoA has special flags as shown in Table 2. All HoAs have the IFF_HOME flag. The IFF_HOME flag is used by the source address selection procedure to prefer an HoA as a source address. The IFF_DEREGISTERING flag is used in the returning home procedure. The IFF_DEREGISTERING flag is added while an MN/MR is performing de-registration procedure of its HoA when it returns to home. Until the procedure has successfully completed, the HoA is not valid and is not used for communication.

4.3 The mtun Interface

The mtun interface is used when the NEMO BS function is used. This interface is used by an MR and an HA to create an unnumbered tunnel between them. The physical endpoint address of the tunnel is the CoA of the MR and the HA's address. On the HA, the traffic addressed to the mobile network of the MR is sent to the mtun interface established between it and the MR that manages the mobile network. On the MR, the mtun interface is set as the default route of the outgoing packets. All packets generated by the MR or the nodes in the mobile network of the MR will be tunneled to the HA.

Since the mtun interface is used as the default route on the MR, the loop condition occurs if we do not specify the next hop router when sending tunneled packets. Figure 4 shows the situation. When the MR

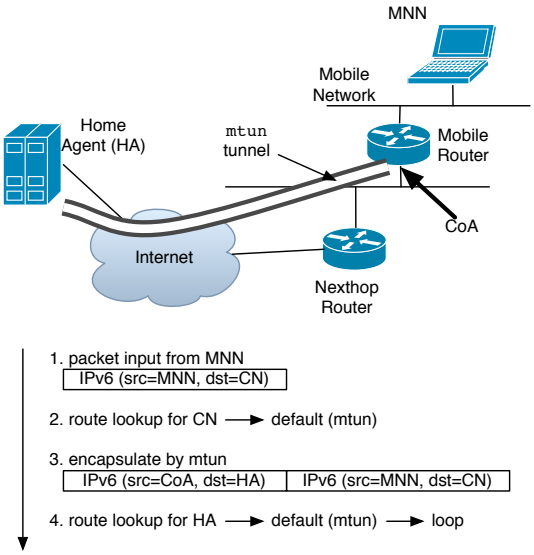

Figure 4: The loop of a tunneled packet.

sends a packet to the default route that is the `mtun` tunnel, the MR creates an encapsulated packet whose outer source is its CoA and the outer destination is the HA. The output function of the encapsulated packet will try to send it based on the routing table and it will try to send it to the default route again.

To avoid this problem, the `mtun` interface keeps the next hop router's address in its interface structure. The information is retrieved from the default router list managed by the kernel. The **nemonetd** program checks the default router list and picks up one of them that are attached to the same network as the CoA of the MR. The next hop information is stored by the **nemonetd** program using I/O control message of the `mtun` interface. When sending an encapsulated packet, the output function of the `mtun` interface will add the next hop information as an IPv6 packet option.

As we have mentioned already, the `mtun` interface is originally copied from the `gif` interface. The only difference is the next hop information storing mechanism and output mechanism using the information.

4.4 Sending and Receiving Signaling Messages

All the signaling messages are processed by the user space programs. The signaling messages are carried by the Mobility Header which is introduced by the Mobile IPv6 specification. Although the header is defined as one of the IPv6 extension headers, it is treated as a final header at this moment. Therefore, there is no following upper layer or other extension headers after a Mobility Header. To support this header, we implemented a simple input validation routine in the kernel and used the raw IPv6 packet delivery mechanism. The Mobility Header processing function is added using the protocol switch mechanism. When a packet whose last header is a Mobility Header (protocol number 135) is input, then the `mip6_input()` function is called.

As shown in the following code fragment, the `mip6_input()` function performs the validation check of the input packet and calls the raw IPv6 input function (`rip6_input()`) to deliver the packet to applications. The application with the Mobility socket will receive all Mobility Header messages.

```
int
mip6_input(mp, offp, proto)
    struct mbuf **mp;
    int *offp, proto;
{
    validation of the input packet.

    /* deliver the packet using Raw IPv6
       interface. */
    return (rip6_input(mp, offp ,proto));
}
```

When sending a Mobility Header packet, the same output function as that of the raw IPv6 socket (`rip6_output()`) is used.

4.5 Extension Header Processing

The Mobile IPv6 specification defines a new destination option, the HAO option, to carry the HoA of an MN to an HA or a CN, and the RTHDR2 to deliver packets to an MN directly from an HA or a CN. We simply extended the existing code to support these new messages, since both Destination Options Header and Routing Header processing code had been already implemented as a basic IPv6 feature in NetBSD.

The input processing code of the HAO option is implemented in the `dest6_input()` function. The function checks an HAO option and related binding cache entry of the HoA included in the HAO option. If the cache entry exists, the source address of the input packet and the HoA are swapped. The transport layer and above layer will process the HoA as the source of the packet. Note that this operation is a kind of source spoofing operation and we need to verify the operation is safe. The existence of the binding cache entry is used for the validation.

The exception of the swapping is a BU message. A BU message has an HAO option to request a peer node to create a binding cache entry that binds the

HoA in the HAO option and the CoA stored in the source address field of the IPv6 header. When a node receives a BU message first time, there is no binding cache entry, and we cannot rely on the cache existence to validate the message. In the Mobile IPv6 specification, it is specified that a BU message is protected by some cryptographic mechanisms. When an MN sends a BU message to its HA, the message is protected by the IPsec mechanism. When an MN sends a BU message to a CN, the message is protected by the secret created through the RR procedure. If a bogus MN tries to send a BU message to a victim HA, then the message will be dropped during the IPsec header processing. If a bogus MN tries to send a BU message to a CN, then the message will be delivered to the **cnd** program because it does not have any IPsec headers. The **cnd** program checks if the message is protected by the secret created by the RR procedure, and drop it if it is not protected. Once the message is accepted, the **had** or **cnd** program creates a new binding cache entry for the message. The following packets with an HAO option will be accepted.

The input processing of the RTHDR2 is implemented in the `route6_input()` function. The function calls the `rthdr2_input()` function when the type number of the input routing header is 2. The RTHDR2 includes the HoA of an MN. The basic procedure is same as that of the Type 1 Routing Header (RTHDR1). The destination address of the input IPv6 header and the address in the Routing Header are swapped. Unlike the RTHDR1, the RTHDR2 only include one address and the address must be an HoA. The `rthdr2_input()` validates the RTHDR2 and swaps the addresses if it is valid. Since the original IPv6 destination address (which is the CoA of an MN) and the address in the RTHDR2 (the HoA of the MN) both belong to the same MN, a peer node can send a packet directly to the MN without using the tunnel established between the MN and its HA.

The output processing of these headers is handled by the `ip6_output()` function. Since the mobility functions are transparent to all the applications, the packet passed to the `ip6_output()` function does not have any mobility related data, except signaling packets that are handled in the user space programs and have extension headers specified by the user space programs. The `ip6_output()` function checks binding update list entries and binding cache entries at the beginning of the packet processing, and inserts a HAO and/or a RTHDR2 if there is a binding entry related to the addresses of the outgoing IPv6 packet. For example, if the packet's source address is the HoA of an MN and the MN has a valid binding update list entry of the HoA, then a HAO option, that includes the CoA of the MN stored in the binding update list entry, is created and inserted to the outgoing packet. Similarly, a RTHDR2 is also inserted if there is a valid binding cache entry that is related to the destination address of the outgoing packet.

4.6 Tunneling

When an MN/MR sends packets to CNs, or when an MR forwards packets from its mobile network to the nodes outside, the nodes encapsulate packets to its HA using the tunnel established between them. The same operation is performed in the reverse direction. In the SHISA stack, we use two different encapsulating mechanisms for tunneling. One is the mechanism for packets sent/delivered to a moving node itself, the other is for packets sent/delivered to the nodes in a mobile network.

In fact, these two tunnels have the same function. The reason why we have two different tunnels is that the stack has been build step-by-step based on the previous KAME Mobile IPv6 design. In KAME Mobile IPv6 that did not support NEMO BS, the tunneling was implemented as a part of packet processing in the kernel. SHISA re-used the design as a Mobile IPv6 tunneling mechanism. When we started implementing NEMO BS in SHISA, we chose to use a specific tunnel interface (the `mtun` interface) as a tunneling mechanism for mobile network nodes, so that the `nemonetd` programs can easily control the tunnel endpoints. We do not think the current design is the best and keep discussing to revise the design. Section 5 mentions this topic further.

Figure 5 shows the output flow of tunneling packets on an MN/MR. When an MN sends a tunneled packet, the `mip6_tunnel_output()` function is used. For the forwarding packets from the mobile network of an MR, the `mtun_output()` which is the output function of the `mtun` interface is used instead.

Figure 6 shows the input flow of tunneling packets on an MN/MR. Similar to the output case, the tunneled packets sent to the moving node itself is processed by the special input function `mip6_tunnel_input()`. For the forwarding packets to the mobile network nodes, the `mtun_input()` function handles tunneled packets.

4.7 Intercepting Packets

Thanks to the backward compatibility of Mobile IPv6, all IPv6 nodes can communicate with an MN/MR or nodes inside the mobile network of the MR. In this case, all packets sent to the moving entities are routed to the home network of them. The HA of these mov-

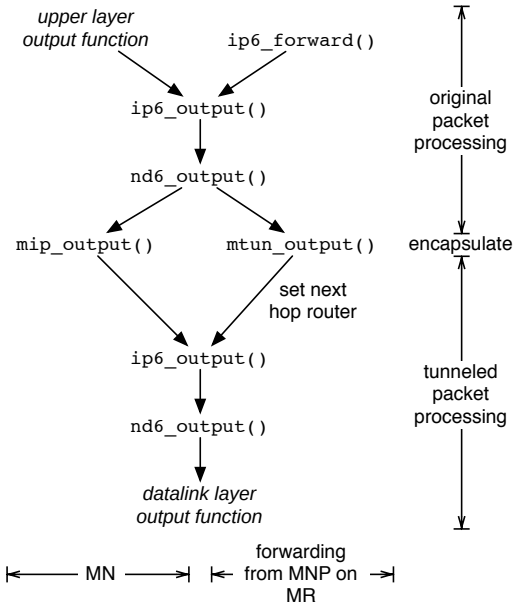

Figure 5: The output flow of a tunneled packet on an MN/MR.

ing entities have to intercept the packets and forward them properly.

When an HA intercepts packets sent to the HoA of an MN or MR, the HA uses proxy Neighbor Discovery mechanism. The proxy is started after the HA receives a valid BU message for registration from the MN/MR, and is stopped when it receives a de-registration BU message. The intercepted packets are forwarded using the tunneling mechanism. In contrast to the packets sent to HoAs, the packets sent to the mobile network of the MR are processed by the normal forwarding mechanism. The HA has a routing entry for the MNP whose outgoing interface is set to the tunnel interface. Figure 7 shows the flow.

The input processing of the tunnel packets at an HA is a simple forwarding processing. The only difference is that the packets originated by an MN/MR itself are input by the special input function `mip6_tunnel_input()`. Figure 8 shows the flow. The `mip6_tunnel_input()` function is defined as a part of the protocol switch structure for Mobile IPv6 as shown in Figure 9. The protocol switch structure is used internally in the kernel when the Mobile IPv6 function is enabled.

4.8 Movement Detection

Movement detection is also performed in user space in the SHISA stack. At this moment, we are providing

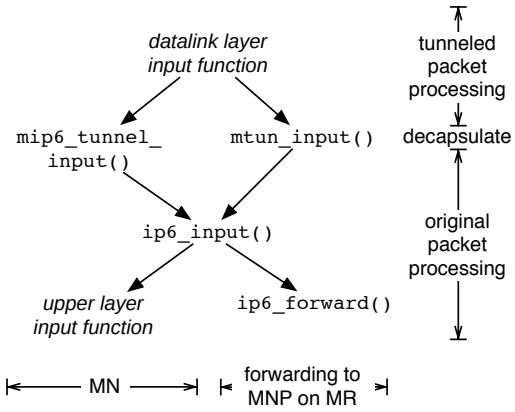

Figure 6: The input flow of a tunneled packet on an MN/MR.

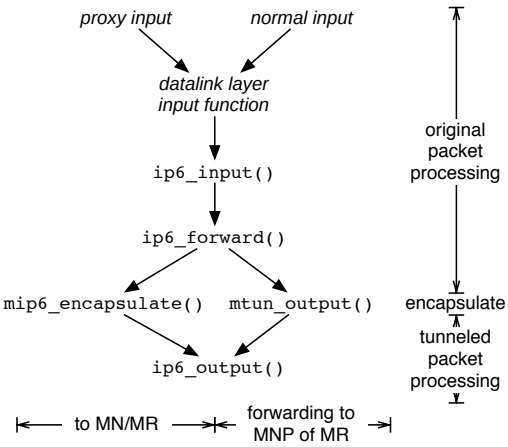

Figure 7: The output flow of a tunneled packet on an HA.

a simple detection program `babymdd` as a sample code for developers of more enhanced detection program. The `babymdd` programs detects the node movement based on the validity of the CoA currently assigned. In the BSD Operating Systems, all IPv6 addresses have a special flag called *DETACHED* that is proposed in [12]. The flag means that the address is valid but the router that advertised the prefix of the address is unreachable. This implies that an MN/MR once received prefix information and formed an address from the prefix, but left the network.

The `babymdd` program sends a Router Solicitation message when the status of the network interfaces used to connect the node to the Internet is changed from 'down' to 'up'. If the node leaves and attaches to a new network, then the old routers will become unreachable by the Neighbor Unreachability Detection

```
struct ip6protosw mip6_tunnel_protosw =
{ SOCK_RAW,     &inet6domain,   IPPROTO_IPV6,   PR_ATOMIC|PR_ADDR,
  mip6_tunnel_input, rip6_output, 0,            rip6_ctloutput,
  rip6_usrreq,
  0,            0,              0,              0,
};
```

Figure 9: The tunneled packet protocol switch entry.

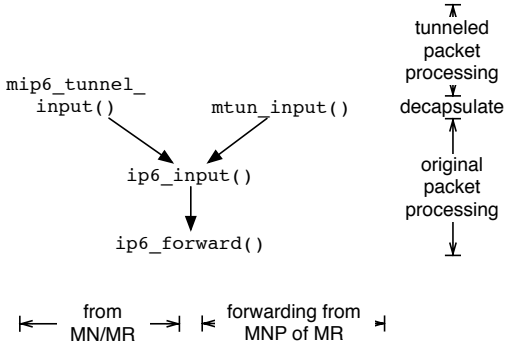

Figure 8: The input flow of a tunneled packet on an HA.

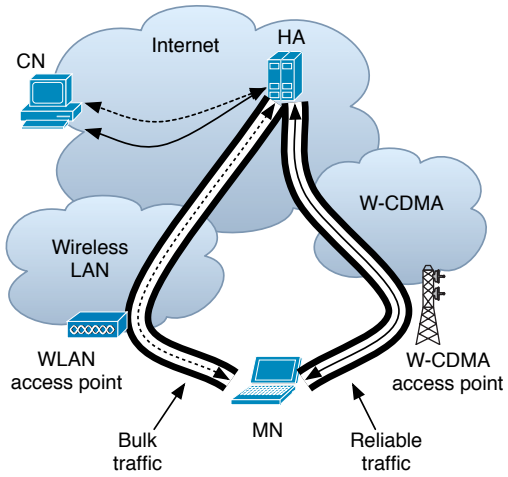

Figure 10: The usage scenario of multiple network interfaces simultaneously.

(NUD) mechanism. As a result the corresponding addresses formed from the prefix advertised by these old routers will become detached. If the CoA is one of these detached addresses, the `babymdd` program will search other appropriate address and inform the `mnd` or `mrd` program of the new CoA by the `MD_INFO` Mobility socket message.

5 Discussion

SHISA provides full functional Mobile IPv6 and NEMO BS implementation. We have conformed its interoperability with other implementations through a couple of interoperability test events. However we still need to develop the SHISA stack in order to support extensions of Mobile IPv6/NEMO BS, which are currently discussed in the IETF, and to support more operating platforms. In this section, we explain some of these remaining stuffs.

5.1 Multiple Tunnel Mechanisms

As discussed in Section 4.6, the SHISA stack is currently providing two different tunneling mechanisms related to mobility function for the same purpose, one for Mobile IPv6 and the other for NEMO BS. When a node is acting as an MR, it can also work as an MN. However the packet tunneled to its HA goes to the `mtun` interface when it is an MR, and goes to the internal in-kernel tunnel function if it is an MN. This causes not only the code duplication problem but also cases functional restrictions.

The IETF MONAMI6 WG is standardizing the mechanism to utilize multiple network interfaces at the same time on Mobile IPv6 and NEMO BS. For example, if a mobile device has a wireless LAN interface and a W-CDMA interface, it might want to utilize both of them according to the local traffic policy. The mobile device can use the wireless LAN interface as a cheap bulk data transfer interface and use the W-CDMA as a reliable interface (Figure 10). Recently, as many mobile devices often have multiple communication interfaces, utilizing them simultaneously is urgent matter for Mobile IP and NEMO. The Multiple Care-of Addresses Registration (MCoA) mechanism [13] proposed at the MONAMI6 WG provides a method to register more than one CoA at the same time.

The current SHISA implementation supports MCoA for NEMO BS as described below. The tunnel mechanism of NEMO BS is implemented as the `mtun` interface as discussed in Section 4.3. The current de-

sign of the MCoA mechanism in SHISA is to define the same number of `mtun` interfaces as the number of physical network interfaces, and to bind each physical interface to a `mtun` interface. The default route of an MR is set to one of the `mtun` interfaces (for example, `mtun0`) and packet flow is distributed using a packet filter mechanism, such as IP Filter [14] or PF [15]. If a wireless LAN interface `wi0` is bound to the `mtun0` interface, and a W-CDMA interface `ppp0` is bound to the `mtun1` interface, then we may use the rules described in Figure 11 to distribute traffic. With these rules, all traffic except the SSH traffic is sent to the `mtun0` interface which is bound to the wireless LAN interface. This mechanism cannot be used with the Mobile IPv6 case of the SHISA implementation, because the `mtun` interface is not used in Mobile IPv6.

We once tried to solve this problem using the PF mechanism for Mobile IPv6 too. In the trial, we stopped using the special in-kernel tunnel mechanism and passed all the Mobile IPv6 traffic to the `mtun` interface. It worked with one network interface, however we noticed that we would have a problem when we use multiple network interfaces and the RO communication.

The stack has multiple binding update list or cache entries when the node registers multiple CoAs to its HA. When the RO is used, the source address of the packet (which is one of the CoAs of the MN) must be decided based on the local flow distribution policy. That means, we need two policy judgement points for the essentially same traffic, one in the CoA selection part, and the other in the packet filtering part.

We are now designing a new tunnel mechanism for mobility functions. In the idea, the moving node always outputs packets to a special tunnel interface bound to the home network of the node (similar to the `mip` interface). In the output function, the local traffic distribution policies are applied to the packets and they are redirected to the HA with an encapsulating header with a proper CoA of the node based on the policy. With this procedure, we can put both the CoA selection task and policy application task in the same place that will solve the problem described above. We will verify if this is feasible to implement.

5.2 IPsec Policy Management

As specified in RFC, some of the signaling messages between an MN and an HA must be protected by the IPsec mechanism. In these messages, the HoT and HoTI messages cause IPsec configuration problem when a node returns to home. These messages must be protected by the ESP tunnel mechanism while the node is in a foreign network. In the current implementation, the node has static IPsec tunnel policy entries for these messages. Figure 12 is a sample policy definition for these packets. 135 is the protocol number of the Mobility Header and 1 and 3 represents the HoTI and HoT message types respectively.

```
spdadd HoA ::/0 135 1,0 -P out ipsec
  esp/tunnel/HoA-HA/;
spdadd ::/0 HoA 135 3,0 -P in ipsec
  esp/tunnel/HA-HoA/;
```

Figure 12: IPsec policy entries to protect the HoTI and HoT messages

These tunnels are used only the node is in a foreign network and must not be used at home. This restriction cases a problem. An MN has to de-register its binding information registered in CNs when the MN returns to home. To de-register binding information, the MN needs to perform the RR procedure that requires HoTI/HoT message exchange. The HoTI message sent from the node will match the IPsec policy statically installed on the MN and may be dropped at the tunnel end point (the HA). The HoT messages will come from CNs directly to the MN, because the MN has already returned to home and the HA is not proxying its address anymore. The IPsec policy will discards the incoming HoT message because it is not protected by the IPsec mechanism as required in the policy entry.

To solve this problem, the mobility stack must inactivate all the policy related to the HoTI/HoT messages installed in the kernel. Currently, there is no standard way to inactivate the policy entries, other than removing them. Removing policy entries may work if the node uses IPsec only for Mobile IPv6. However if other communication frameworks are also using IPsec policy database, then removing and adding policy entries may influence the policy matching order, that may result in unexpected IPsec processing. We are considering a new policy management message to activate/inactivate a specific policy entry and planning to implement and test the mechanism.

5.3 Standard Mobility Interface

We have moved all the signal processing code to user space. That means, if the kernel supports packet forwarding mechanisms for Mobile IPv6 and NEMO BS, then we can use the same signal management program on different kernels. We defined a generic mobility information exchange mechanism as the Mobility Socket for this purpose. The messages used in the socket is basically platform independent. Thus, if we can cleanly separate kernel functions and user space

```
pass out route-to mtun0 inet6 from MNP::/64 to any
pass out route-to mtun1 inet6 from MNP::/64 to any port 22
```

Figure 11: The filter rules to distribute mobile network traffic to multiple NEMO BS tunnels

functions, we can develop the kernel and the signal processing program independently. SHISA now runs only on BSD operating systems that support the Mobility Socket and in-kernel mobility functions, however it can run on other operating systems if they provide the Mobility Socket interface and equivalent functions in their kernel.

One obvious missing feature of the current Mobility Socket implementation is a message filtering mechanism. Currently, all the messages sent by mobility entities are delivered to all the listening sockets regardless of its necessity. However, some Mobility Socket messages are meaningless to some of the mobility entities. For example, the HA module may not want to receive any messages related to MN/MR functions. Suppressing unnecessary messages will alleviate the exhaustion of the socket buffer when there are many messages.

We once submitted the basic specification (not including the filtering mechanism) of the Mobility Socket at the IETF, but more than one year has passed since the draft expired. We may need to resume the standardization work when we have gotten clear understanding of the roles in the kernel and user space though the development.

5.4 IKE Interaction

At this moment, the SHISA stack works with IPsec security associations (SAs) manually configured. The manual operation works only with a small number of nodes and it is not scalable. The essential solution is the Internet Key Exchange (IKE) protocol that provides a dynamic SA generation mechanism. IKE creates a pair of SA between two nodes, however the constructed SA is based on the addresses used by the IKE procedure. This causes a problem in mobility environment. In the Mobile IPv6 (and NEMO BS) case, the communication is originated from the HoA of the MN/MR. Because the HoA cannot be used for communication, the MN/MR cannot start the IKE procedure using their HoA.

The solution is to use CoAs for IKE communication and creates SAs for HoA during the IKE negotiation[3]. How to use IKE with Mobile IPv6 is further described in [16, 17]. Unfortunately, most of the current IKE programs are not aware of Mobile IPv6. To provide the dynamic keying feature to our stack and encourage mobility technology deployment, we are now working with the Racoon2 project [18] that is developing an open source IKE implementation. The interaction mechanism between a mobility stack and an IKE program is proposed in [19]. The proposal defines an optional data structure to provide CoA and HoA information to an IKE program of an MN, when it needs to start the SA negotiation process. We are planning to join the discussion of the standardization process of the proposal and to provide the implementation.

[3]The code contributed by Francis Dupont exists waiting to be merged to the SHISA code.

5.5 Porting to Other Platforms

The KAME version of the SHISA stack supported both NetBSD 2.0 and FreeBSD 5.4. Unfortunately we could not support OpenBSD mainly because lack of developers using OpenBSD in our team, but it could be supported potentially. As explained, we are now focusing on NetBSD-current. Once we have completed the porting work to NetBSD-current, we will work on FreeBSD-current. The work will be harder than the NetBSD work, since the difference of the kernel code between FreeBSD 5.4 and FreeBSD-current is bigger than that of NetBSD. In addition, recent FreeBSD introduced the fine-grained locking mechanism for better performance in a multi-processor environment. The IPv6 code and mobility related code in the kernel does not have support for the fine-grained locking mechanism. The adaptation will need some additional development and more test to stabilize. The OpenBSD port is planned after the FreeBSD port.

It is possible to port SHISA to other platforms than BSDs if they support kernel modifications as described in Section 5.3. In fact, there is a port to the Darwin operating system as announced in the Darwin IPv6 developers mailing list.

6 Conclusion

We developed the Mobile IPv6 and NEMO BS protocol stack on the KAME platform. We are now porting it to the latest BSD distributions. We have started to port it to NetBSD as the first step and will try to work on other platforms based on the progress of the current work. The stack provides most of the specified features. It is confirmed interoperable with many other independently developed Mobile IPv6 and NEMO BS stacks, and it works stably. We are now focusing to

refine the code, especially the kernel code, to make the quality high enough to be merged to the NetBSD main tree.

Although we have completed the implementation of the basic mobility functions, we still have many things to do to support advanced mobility features under standardization in the IETF. We continue to work on supporting these advanced functions to provide more useful mobility stack to various BSD operating systems.

Acknowledgement

The authors would like to thank the WIDE Project for the support on our development activity.

References

[1] David B. Johnson, Charles E. Perkins, and Jari Arkko. Mobility Support in IPv6. Technical Report RFC3775, IETF, June 2004.

[2] Vijay Devarapalli, Ryuji Wakikawa, Alexandru Petrescu, and Pascal Thubert. Network Mobility (NEMO) Basic Support Protocol. Technical Report RFC3963, IETF, January 2005.

[3] WIDE project. SHISA, February 2007. http://www.mobileip.jp/.

[4] Keiichi Shima, Ryuji Wakikawa, Koshiro Mitsuya, Tsuyoshi Momose, and Keisuke Uehara. SHISA: The IPv6 Mobility Framework for BSD Operating Systems. In *IPv6 Today – Technology and Deployment (IPv6TD'06)*. International Academy Research and Industry Association, IEEE Computer Society, August 2006.

[5] WIDE project. KAME Working Group, March 2006. http://www.kame.net/.

[6] Koshiro Mitsuya. ATLANTIS: NEMO Basic Support Implementation, January 2005. http://www.nautilus6.org/implementation/atlantis.html.

[7] Ryuji Wakikawa, Susumu Koshiba, Keisuke Uehara, and Jun Murai. Multiple Network Interfaces Support by Policy-Based Routing on Mobile IPv6. In *2002 International Conference on Wireless Networks (ICWN'02)*, July 2002.

[8] Tsuyoshi Momose, Keiichi Shima, and Anti Tuominen. The application interface to exchange mobility information with Mobility subsystem (Mobility Socket, AF_MOBILITY). Technical Report draft-momose-mip6-mipsock-00, IETF, June 2005.

[9] Tatuya Jinmei, Kazuhiko Yamamoto, Jun-ichiro Hagino, Shoichi Sakane, Hiroshi Esaki, and Jun Murai. The IPv6 Software Platform for BSD. *IEICE Transactions on Communications*, E86-B(2):464–471, February 2003.

[10] Keith Sklower. A Tree-based Packet Routing Table for Berkeley UNIX. In *Proceedings of the Winter 1991 USENIX Conference*, pages 93–103. USENIX Association, January 1991.

[11] Samita Chakrabarti and Erik Nordmark. Extension to Socket API for Mobile IPv6. Technical Report RFC4584, IETF, July 2006.

[12] Tatuya Jinmei, Jun-ichiro Ito, and Munechika Sumikawa. Efficient Use of IPv6 Auto-Configuration in a Mobile Environment. In *The 7th Research Reporting Session*. Information Processing Society of Japan, SIG Mobile Computing, December 1998.

[13] Ryuji Wakikawa, Thierry Ernst, and Kenichi Nagami. Multiple Care-of Addresses Registration. Technical Report draft-wakikawa-mobileip-multiplecoa-05, IETF, February 2006.

[14] Darren Reed. IP Filter. Web page, February 2007. http://coombs.anu.edu.au/~avalon/.

[15] Daniel Hartmeier. Design and Performance of the OpenBSD Stateful Packet Filter (pf). In *USENIX 2002 Annual Technical Conference*, pages 171–180, June 2002.

[16] Jari Arkko, Vijay Devarapalli, and Francis Dupont. Using IPsec to Protect Mobile IPv6 Signaling Between Mobile Nodes and Home Agents. Technical Report RFC3776, IETF, June 2004.

[17] Vijay Devarapalli and Francis Dupont. Mobile IPv6 Operation with IKEv2 and the revised IPsec Architecture. Technical Report draft-ietf-mip6-ikev2-ipsec-07, IETF, October 2006.

[18] WIDE project. The Racoon2 Project, February 2007. http://www.racoon2.wide.ad.jp/.

[19] Shinta Sugimoto, Francis Dupont, and Masahide Nakamura. PF_KEY Extension as an Interface between Mobile IPv6 and IPsec/IKE. Technical Report draft-sugimoto-mip6-pfkey-migrate-03, IETF, September 2006.