Network Working Group
Request for Comments: 4584
Category: Informational

S. Chakrabarti
E. Nordmark
Sun Microsystems

July 2006

Extension to Sockets API for Mobile IPv6

Status of This Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Copyright Notice

Copyright (C) The Internet Society (2006).

Abstract

This document describes data structures and API support for Mobile IPv6 as an extension to the Advanced Socket API for IPv6.

Just as the Advanced Sockets API for IPv6 gives access to various extension headers and the ICMPv6 protocol, this document specifies the same level of access for Mobile IPv6 components. It specifies a mechanism for applications to retrieve and set information for Mobility Header messages, Home Address destination options, and Routing Header Type 2 extension headers. It also specifies the common data structures and definitions that might be used by certain advanced Mobile IPv6 socket applications.

Table of Contents

1. Introduction

Mobility Support in IPv6 [2] defines a new Mobility Protocol header, a Home Address destination option and a new Routing Header type. It is expected that Mobile IPv6 user-level implementations and some special applications will need to access and process these IPv6 extension headers. This document is an extension to the existing Advanced Sockets API document [1]; it addresses the Advanced IPv6 Sockets API for these new protocol elements defined by Mobile IPv6.

The applicability of this API mainly targets user-level applications. However, it has also been shown to be useful within some Mobile IPv6 implementations; for instance, where part of the Mobile IPv6 protocol is implemented at user-level and part in the kernel. It is up to any such implementations to architect which part of the Mobile IPv6 and IP Security (IPSec) packet processing should be done at the user-level in order to meet the design needs of the particular platform and operating system.

The target user-level applications for this socket API are believed to be debugging and diagnostic applications and some policy applications that would like to receive copies of protocol information at the application layer.

The packet information and access to the extension headers (Routing header and Destination options) are specified using the "ancillary data" fields that were added to the 4.3BSD Reno sockets API in 1990. The reason is that these ancillary data fields are part of the Posix.1g standard and should therefore be adopted by most vendors. This document is consistent with Advanced Sockets API for IPv6 [1] in structure definitions, header files, and function definitions. Thus, the implementors of this API document are assumed to be familiar with the data structures, data sending and receiving procedures, and the IPv6 extension header access functions described in the Advanced Sockets API for IPv6 [1].

Non-goals

This document does not address application access to either the Authentication Header or the Encapsulating Security Payload header. This document also does not address any API that might be necessary for Mobile Network [4] specific needs. Furthermore, note that this API document excludes discussion on application-level API. It assumes that address selection socket API [5] takes care of selection of care-of address or home address as the source address by the application, when source address selection is required due to the nature of the application.

Providing mobility "awareness" to applications, such as applications' being able to tell whether the host is at home or not, is out of scope for this API.

2. Applicability

This API document can be applied in the following cases:

1. User-level debugging and monitoring tools: This socket API is useful for accessing Mobility Headers, Home Address destination options and Type 2 Routing Headers . For example, mh-ping might be a monitoring tool that can process mobility headers on the receiving side to check binding status.

2. Partial user-level implementation of Mobile IPv6: We assume that some implementations may choose to do the Mobility header processing at user level. In that case, this document recommends implementing at least the handling of Home Address destination options and Type 2 Routing Header in the main IP processing paths in the kernel. The API can then be used to send and receive the Mobility Header packets used for Mobile IPv6 signaling.

3. Complete header processing at the kernel-level: Many implementations of Mobile IPv6 [2] perform processing of Home Address destination options, Type 2 Routing Headers, and Mobility headers at the kernel level. However, the kernel keeps a copy of the received extension headers and passes them up to the API, which is used by the user-level applications purely for monitoring and debugging Mobile IPv6 packets.

On an IPv6 host that does not implement Mobile IPv6, the IPv6 specification [3] requires that packets with the Home Address option or Type 2 Routing Header (where segments left is non-zero) be dropped on receipt. This means that it is not possible to implement Mobile IPv6 as an application on such a system. Thus, on such a system, the applicability of this API is limited to the first case above, enabling debugging and monitoring applications (such as tcpdump) to parse and interpret Mobile IPv6 packets.

3. Overview

This document can be divided into the following parts:

1. Definitions of constants and structures for C programs that capture the Mobile IPv6 packet formats on the wire. A common definition of these is useful at least for packet snooping applications. This is captured in Section 4. In addition, Section 4 also defines data structures for Home Address destination option, Type 2 Routing Header, and new ICMPv6 messages related to Mobile IPv6.

2. Notes on how to use the IPv6 Advanced API to access Home Address options and Type 2 Routing Headers. This is captured in Section 5.

3. Notes on how user-level applications can observe MH (Mobility Header) packets using raw sockets (in Section 6). The IPv6 RAW socket interface described in this document allows applications to receive MH packets whether or not the system's MH processing takes place in the "kernel" or at the "user space".

4. A name is suggested for IPv6 Mobility Header protocol in /etc/protocols (in Section 7).

All examples in this document omit error checking in favor of brevity, as it is following the same style as the Advanced Socket API [1].

Note that many of the functions and socket options defined in this document may have error returns that are not defined in this document.

Data types in this document follow the Posix.1g format: intN_t means a signed integer of exactly N bits (e.g., int16_t), and uintN_t means an unsigned integer of exactly N bits (e.g., uint32_t).

Once the API specification becomes mature and is deployed, it may be formally standardized by a more appropriate body, as has been done with the Basic API [6]. However, since this specification largely builds upon the Advanced Socket API [1], such standardization would make sense only if the Advanced Socket API [1] were also standardized.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

4. Common Structures and Definitions

In this section, the structures are specified in a way so that they maximize the probability that the compiler-layout of data structures are identical to the packet formats on the wire. However, ANSI-C provides few guarantees about the size and alignment of data structures.

The assumption is that the Advanced Socket API [1] will pass up the actual packet content (the wire format) in the buffer and in the ancillary data objects. Thus, if an implementor has to handle a system where the ANSI-C compiler does not and can not lay out these structures to match the wire formats in RFC 3775 [2], the structures defined by this API can not be supported on such a system.

The constants and structures shown below are in network byte order, so an application needs to perform the appropriate byte order conversion (ntohs(), etc) when necessary.

The structures and constants below will be included when the (new) header file is included : <netinet/ip6mh.h>

4.1. The Mobility Header Data Structures

4.1.1. The ip6_mh Structure

The following structure is defined as a result of including <netinet/ip6mh.h>. This is the fixed part of the Mobility Header. Different Mobility message types are defined in Mobile IPv6 [2]. For portability and alignment reasons, each mobility message type includes the mobility header fields instead of including the ip6_mh structure, followed by the message-specific fields.

```
struct  ip6_mh {
    uint8_t    ip6mh_proto;   /* NO_NXTHDR by default */
    uint8_t    ip6mh_hdrlen;  /* Header Len in unit of 8 Octets
                                 excluding the first 8 Octets */
    uint8_t    ip6mh_type;    /* Type of Mobility Header */
    uint8_t    ip6mh_reserved;   /* Reserved */
    uint16_t   ip6mh_cksum;   /* Mobility Header Checksum */
    /* Followed by type specific messages */
};
```

4.1.2. Binding Refresh Request Mobility Message

```
struct  ip6_mh_binding_request {
    uint8_t    ip6mhbr_proto;
    uint8_t    ip6mhbr_hdrlen;
    uint8_t    ip6mhbr_type;
    uint8_t    ip6mhbr_reserved;
    uint16_t   ip6mhbr_cksum;
    uint16_t   ip6mhbr_reserved2;
    /* Followed by optional Mobility Options */
};
```

4.1.3. Home Address Test Init (HoTI) Message

```
struct   ip6_mh_home_test_init {
   uint8_t    ip6mhhti_proto;
   uint8_t    ip6mhhti_hdrlen;
   uint8_t    ip6mhhti_type;
   uint8_t    ip6mhhti_reserved;
   uint16_t   ip6mhhti_cksum;
   uint16_t   ip6mhhti_reserved2;
   uint32_t   ip6mhhti_cookie[2]; /* 64 bit Cookie by MN */
   /* Followed by optional Mobility Options */
};
```

4.1.4. Care-of Address Test Init (CoTI) Message

```
struct   ip6_mh_careof_test_init {
   uint8_t    ip6mhcti_proto;
   uint8_t    ip6mhcti_hdrlen;
   uint8_t    ip6mhcti_type;
   uint8_t    ip6mhcti_reserved;
   uint16_t   ip6mhcti_cksum;
   uint16_t   ip6mhcti_reserved2;
   uint32_t   ip6mhcti_cookie[2]; /* 64 bit Cookie by MN */
   /* Followed by optional Mobility Options */
};
```

4.1.5. Home Address Test (HOT) Message

```
 struct  ip6_mh_home_test {
    uint8_t    ip6mhht_proto;
    uint8_t    ip6mhht_hdrlen;
    uint8_t    ip6mhht_type;
    uint8_t    ip6mhht_reserved;
    uint16_t   ip6mhht_cksum;
    uint16_t   ip6mhht_nonce_index;
    uint32_t   ip6mhht_cookie[2];    /* Cookie from HOTI msg */
    uint32_t   ip6mhht_keygen[2];  /* 64 Bit Key by CN */
    /* Followed by optional Mobility Options */
};
```

4.1.6. Care Of Address Test (COT) Message

```
struct  ip6_mh_careof_test {
   uint8_t    ip6mhct_proto;
   uint8_t    ip6mhct_hdrlen;
   uint8_t    ip6mhct_type;
   uint8_t    ip6mhct_reserved;
   uint16_t   ip6mhct_cksum;
   uint16_t   ip6mhct_nonce_index;
   uint32_t   ip6mhct_cookie[2]; /* Cookie from COTI message */
   uint32_t   ip6mhct_keygen[2];  /* 64bit key by CN */
   /* Followed by optional Mobility Options */
};
```

4.1.7. Binding Update Mobility Message

```
struct ip6_mh_binding_update {
    uint8_t     ip6mhbu_proto;
    uint8_t     ip6mhbu_hdrlen;
    uint8_t     ip6mhbu_type;
    uint8_t     ip6mhbu_reserved;
    uint16_t    ip6mhbu_cksum;
    uint16_t    ip6mhbu_seqno;      /* Sequence Number */
    uint16_t    ip6mhbu_flags;
    uint16_t    ip6mhbu_lifetime; /* Time in unit of 4 sec */
    /* Followed by optional Mobility Options */
};

 /* Binding Update Flags, in network byte-order */
 #define IP6_MH_BU_ACK    0x8000  /* Request a binding ack */
 #define IP6_MH_BU_HOME   0x4000  /* Home Registration */
 #define IP6_MH_BU_LLOCAL 0x2000  /* Link-local compatibility */
 #define IP6_MH_BU_KEYM   0x1000  /* Key management mobility  */
```

4.1.8. Binding Acknowledgement Mobility Message

```
struct  ip6_mh_binding_ack {
   uint8_t   ip6mhba_proto;
   uint8_t   ip6mhba_hdrlen;
   uint8_t   ip6mhba_type;
   uint8_t   ip6mhba_reserved;
   uint16_t  ip6mhba_cksum;
   uint8_t   ip6mhba_status;    /* Status code */
   uint8_t   ip6mhba_flags;
   uint16_t  ip6mhba_seqno;
   uint16_t  ip6mhba_lifetime;
   /* Followed by optional Mobility Options */
};

 /* Binding Acknowledgement Flags */
 #define IP6_MH_BA_KEYM       0x80  /* Key management mobility */
```

4.1.9. Binding Error Mobility Message

```
 struct   ip6_mh_binding_error {
    uint8_t   ip6mhbe_proto;
    uint8_t   ip6mhbe_hdrlen;
    uint8_t   ip6mhbe_type;
    uint8_t   ip6mhbe_reserved;
    uint16_t  ip6mhbe_cksum;
    uint8_t   ip6mhbe_status;  /* Error Status */
    uint8_t   ip6mhbe_reserved2;
    struct in6_addr ip6mhbe_homeaddr;
    /* Followed by optional Mobility Options */
  };
```

4.1.10. Mobility Option TLV data structure

```
struct   ip6_mh_opt {
   uint8_t    ip6mhopt_type;   /* Option Type */
   uint8_t    ip6mhopt_len;    /* Option Length */
   /* Followed by variable length Option Data in bytes */
};
```

4.1.11. Mobility Option Data Structures

4.1.11.1. Binding Refresh Advice

```
struct ip6_mh_opt_refresh_advice {
    uint8_t  ip6mora_type;
    uint8_t  ip6mora_len;
    uint16_t ip6mora_interval; /* Refresh interval in 4 sec */
};
```

4.1.11.2. Alternate Care-of Address

```
struct ip6_mh_opt_altcoa {
    uint8_t ip6moa_type;
    uint8_t ip6moa_len;
    struct in6_addr ip6moa_addr; /* Alternate CoA */
};
```

4.1.11.3. Nonce Indices

```
struct ip6_mh_opt_nonce_index {
    uint8_t ip6moni_type;
    uint8_t ip6moni_len;
    uint16_t ip6moni_home_nonce;
    uint16_t ip6moni_coa_nonce;
};
```

4.1.11.4. Binding Authorization Data

```
struct ip6_mh_opt_auth_data {
    uint8_t ip6moad_type;
    uint8_t ip6moad_len;
    uint8_t ip6moad_data[12];
};
```

4.2. Mobility Header Constants

IPv6 Next Header Value for Mobility:

```
<netinet/in.h>


#define IPPROTO_MH       135 /* IPv6 Mobility Header: IANA */
```

Mobility Header Message Types:

<netinet/ip6mh.h>

```
#define IP6_MH_TYPE_BRR       0   /* Binding Refresh Request */
#define IP6_MH_TYPE_HOTI      1   /* HOTI Message   */
#define IP6_MH_TYPE_COTI      2   /* COTI Message  */
#define IP6_MH_TYPE_HOT       3   /* HOT Message   */
#define IP6_MH_TYPE_COT       4   /* COT Message  */
#define IP6_MH_TYPE_BU        5   /* Binding Update */
#define IP6_MH_TYPE_BACK      6   /* Binding ACK */
#define IP6_MH_TYPE_BERROR    7   /* Binding Error */
```

Mobility Header Message Option Types:

<netinet/ip6mh.h>

```
#define  IP6_MHOPT_PAD1       0x00  /* PAD1 */
#define  IP6_MHOPT_PADN       0x01  /* PADN */
#define  IP6_MHOPT_BREFRESH   0x02  /* Binding Refresh */
#define  IP6_MHOPT_ALTCOA     0x03  /* Alternate COA */
#define  IP6_MHOPT_NONCEID    0x04  /* Nonce Index */
#define  IP6_MHOPT_BAUTH      0x05  /* Binding Auth Data */
```

Status values accompanied with Mobility Binding Acknowledgement:

<netinet/ip6mh.h>

```
#define IP6_MH_BAS_ACCEPTED          0   /* BU accepted */
#define IP6_MH_BAS_PRFX_DISCOV       1   /* Accepted, but prefix
                                            discovery Required */
#define IP6_MH_BAS_UNSPECIFIED       128 /* Reason unspecified */
#define IP6_MH_BAS_PROHIBIT          129 /* Administratively
                                            prohibited */
#define IP6_MH_BAS_INSUFFICIENT      130 /* Insufficient
                                            resources */
#define IP6_MH_BAS_HA_NOT_SUPPORTED  131 /* HA registration not
                                            supported */
#define IP6_MH_BAS_NOT_HOME_SUBNET   132  /* Not Home subnet */
#define IP6_MH_BAS_NOT_HA            133  /* Not HA for this
                                             mobile node */
#define IP6_MH_BAS_DAD_FAILED        134  /* DAD failed */
#define IP6_MH_BAS_SEQNO_BAD         135  /* Sequence number out
                                             of range */
```

```
#define IP6_MH_BAS_HOME_NI_EXPIRED   136  /* Expired Home nonce
                                             index */
#define IP6_MH_BAS_COA_NI_EXPIRED    137  /* Expired Care-of
                                             nonce index */
#define IP6_MH_BAS_NI_EXPIRED        138  /* Expired Nonce
                                             Indices */
#define IP6_MH_BAS_REG_NOT_ALLOWED   139  /* Registration type
                                             change disallowed */
```

Status values for the Binding Error mobility messages:

<netinet/ip6mh.h>

```
#define IP6_MH_BES_UNKNOWN_HAO    1 /* Unknown binding for HOA */
#define IP6_MH_BES_UNKNOWN_MH     2 /* Unknown MH Type */
```

4.3. IPv6 Home Address Destination Option

Due to alignment issues in the compiler, and the alignment requirements for this option, the included IPv6 address must be specified as an array of 16 octets.

<netinet/ip6.h>

```
/* Home Address Destination Option */
struct ip6_opt_home_address {
     uint8_t           ip6oha_type;
     uint8_t           ip6oha_len;
     uint8_t           ip6oha_addr[16];   /* Home Address */
};
```

Option Type Definition:

```
#define IP6OPT_HOME_ADDRESS        0xc9    /* 11 0 01001 */
```

4.4. Type 2 Routing Header

<netinet/ip6.h>

```
/* Type 2 Routing header for Mobile IPv6 */
struct ip6_rthdr2 {
     uint8_t  ip6r2_nxt;       /* next header */
     uint8_t  ip6r2_len;       /* length : always 2 */
     uint8_t  ip6r2_type;      /* always 2 */
     uint8_t  ip6r2_segleft;   /* segments left: always 1 */
     uint32_t ip6r2_reserved;  /* reserved field */
     struct in6_addr ip6r2_homeaddr;  /* Home Address */
};
```

4.5. New ICMP Messages for Mobile IPv6

ICMP message types and definitions for Mobile IPv6 are defined in <netinet/icmp6.h>.

```
#define MIP6_HA_DISCOVERY_REQUEST    144
#define MIP6_HA_DISCOVERY_REPLY      145
#define MIP6_PREFIX_SOLICIT          146
#define MIP6_PREFIX_ADVERT           147
```

The following data structures can be used for the ICMP message types discussed in Sections 6.5 through 6.8 in the base Mobile IPv6 [2] specification.

```
struct mip6_dhaad_req {    /* Dynamic HA Address Discovery */
       struct  icmp6_hdr   mip6_dhreq_hdr;
};

#define  mip6_dhreq_type      mip6_dhreq_hdr.icmp6_type
#define  mip6_dhreq_code      mip6_dhreq_hdr.icmp6_code
#define  mip6_dhreq_cksum     mip6_dhreq_hdr.icmp6_cksum
#define  mip6_dhreq_id        mip6_dhreq_hdr.icmp6_data16[0]
#define  mip6_dhreq_reserved  mip6_dhreq_hdr.icmp6_data16[1]


struct mip6_dhaad_rep {    /* HA Address Discovery Reply */
       struct icmp6_hdr   mip6_dhrep_hdr;
       /* Followed by Home Agent IPv6 addresses */
};

#define  mip6_dhrep_type      mip6_dhrep_hdr.icmp6_type
#define  mip6_dhrep_code      mip6_dhrep_hdr.icmp6_code
#define  mip6_dhrep_cksum     mip6_dhrep_hdr.icmp6_cksum
#define  mip6_dhrep_id        mip6_dhrep_hdr.icmp6_data16[0]
#define  mip6_dhrep_reserved  mip6_dhrep_hdr.icmp6_data16[1]


struct mip6_prefix_solicit {   /* Mobile Prefix Solicitation */
       struct icmp6_hdr     mip6_ps_hdr;
};

#define  mip6_ps_type          mip6_ps_hdr.icmp6_type
#define  mip6_ps_code          mip6_ps_hdr.icmp6_code
#define  mip6_ps_cksum         mip6_ps_hdr.icmp6_cksum
#define  mip6_ps_id            mip6_ps_hdr.icmp6_data16[0]
#define  mip6_ps_reserved      mip6_ps_hdr.icmp6_data16[1]
```

```
struct mip6_prefix_advert {  /* Mobile Prefix Advertisements */
       struct  icmp6_hdr   mip6_pa_hdr;
        /* Followed by one or more PI options */
};

#define  mip6_pa_type            mip6_pa_hdr.icmp6_type
#define  mip6_pa_code            mip6_pa_hdr.icmp6_code
#define  mip6_pa_cksum           mip6_pa_hdr.icmp6_cksum
#define  mip6_pa_id              mip6_pa_hdr.icmp6_data16[0]
#define  mip6_pa_flags_reserved  mip6_pa_hdr.icmp6_data16[1]


/* Mobile Prefix Advertisement Flags in network-byte order */
 #define  MIP6_PA_FLAG_MANAGED    0x8000
 #define  MIP6_PA_FLAG_OTHER      0x4000
```

Prefix options are defined in IPv6 Advanced Socket API [1]. The Mobile IPv6 Base specification [2] describes the modified behavior in the 'Modifications to IPv6 Neighbor Discovery' section. Prefix Options for Mobile IP are defined in the following section.

4.6. IPv6 Neighbor Discovery Changes

IPv6 Neighbor Discovery changes are also defined in <netinet/icmp6.h>.

New 'Home Agent' flag in router advertisement: #define ND_RA_FLAG_HOMEAGENT 0x20 /* Home Agent flag in RA */

New Router flag with prefix information of the home agent: #define ND_OPT_PI_FLAG_ROUTER 0x20 /* Router flag in PI */

As per the Mobile IPv6 specification [2], Section 7.2, a Home Agent MUST include at least one prefix option with the Router Address (R) bit set. Advanced Socket API [1] defines data structure for prefix option as follows:

```
struct nd_opt_prefix_info {    /* prefix information */
     uint8_t   nd_opt_pi_type;
     uint8_t   nd_opt_pi_len;
     uint8_t   nd_opt_pi_prefix_len;
     uint8_t   nd_opt_pi_flags_reserved;
     uint32_t  nd_opt_pi_valid_time;
     uint32_t  nd_opt_pi_preferred_time;
     uint32_t  nd_opt_pi_reserved2;
     struct in6_addr  nd_opt_pi_prefix;
};
```

New advertisement interval option and home agent information options are defined in Mobile IPv6 [2] base specification.

```
struct nd_opt_adv_interval { /* Advertisement interval option */
     uint8_t        nd_opt_ai_type;
     uint8_t        nd_opt_ai_len;
     uint16_t       nd_opt_ai_reserved;
     uint32_t       nd_opt_ai_interval;
};
```

The option types for the new Mobile IPv6 specific options:

```
#define  ND_OPT_ADV_INTERVAL    7     /* Adv Interval Option  */
#define  ND_OPT_HA_INFORMATION  8     /* HA Information option */

struct nd_opt_homeagent_info {  /* Home Agent information */
   uint8_t        nd_opt_hai_type;
   uint8_t        nd_opt_hai_len;
   uint16_t       nd_opt_hai_reserved;
   uint16_t       nd_opt_hai_preference;
   uint16_t       nd_opt_hai_lifetime;
};
```

5. Access to Home Address Destination Option and Routing Headers

Applications that need to be able to access Home Address destination option and Type 2 Routing Header information can do so by setting the appropriate setsockopt option and using ancillary data objects. The order of extension headers is defined in Mobile IPv6 [2] when an IPv6 packet with a Home Address Destination Option is sent with other possible extension headers. Section 5.3 elaborates on the extension header order when all possible cases are present.

This document does not recommend that the user-level program set the Home Address destination option or Type 2 Routing Header option; however, for clarity it defines the order of extension headers. See Section 2 of this document for appropriate usage of sending and receiving of Home Address destination options and Type 2 Routing Header extension headers.

This document defines a new socket option, IPV6_MIPDSTOPTS for sending Home Address destination options. In order to receive a Home Address destination option or Type 2 Route Header, applications must call setsockopt() to turn on the corresponding flag as described in IPv6 Advanced Socket API [1] (for brevity, error checking is not performed in the examples):

```
int  on = 1;

setsockopt(fd, IPPROTO_IPV6, IPV6_RECVRTHDR,    &on, sizeof(on));
setsockopt(fd, IPPROTO_IPV6, IPV6_RECVDSTOPTS,
             &on, sizeof(on));
```

When any of these options are enabled, the corresponding data is returned as control information by recvmsg(), as one or more ancillary data objects. Receiving the above information for TCP applications is not defined in this document (see Section 4.1 of Advanced Sockets API for IPv6 [1]).

Note that if the IP implementation on the host does not implement the handling of Type 2 Routing Headers or Home Address options, per RFC 2460 [3] the IP stack is required to drop the packet. Thus, receiving Home Address destination option and Type 2 Routing Header at the application layer requires implementation of respective extension headers at the IP layer in the kernel, as defined in RFC3775 [2].

For receiving the Home Address destination option header, the Mobile IPv6 implementation SHOULD follow the initial processing rules of the Home Address destination option (Section 9.3.1 of Mobile IPv6 [2]) before passing the information to the API level. This includes initial processing of IPSec authentication data in a packet when it exists. Each Destination options header is returned as one ancillary data object described by a cmsghdr structure with cmsg_level set to IPPROTO_IPV6 and cmsg_type set to IPV6_DSTOPTS.

For sending the Home Address destination option, ancillary data can be used to specify the option content for a single datagram. This applies only to datagram and raw sockets, not to TCP sockets. The Advanced API [1] document restricts one IPV6_xxx ancillary data object for a particular extension header in the control buffer. Thus, there would be a single ancillary data object for the Home address destination option in an ancillary data buffer. If multiple destination options are present, then the header order should be in compliance with Section 6.3 and 9.3.2 of the Mobile IPv6 [2] base specification.

For TCP data packets with the Home Address destination option, the "sticky" option may be used for all transmitted packets. The application can remove the sticky Home Destination option header by calling setsockopt() for IPV6_MIPDSTOPTS with a zero option length.

Note that Section 2 of this document does not encourage setting the Home Address destination option at the user level. A Mobile IPv6 implementation should set and process the Home Address destination

option and Routing Header Type 2 at the kernel level. The setting of Routing Header Type 2 and the Home Address destination option are described in this document for completeness and flexibility to use them in the future, if there is a need.

The following socket option parameters and cmsghdr fields may be used for sending (although not a recommended usage):

opt level/ cmsg_level	optname/ cmsg_type	optval/ cmsg_data[]
IPPROTO_IPV6	IPV6_MIPDSTOPTS	ip6_dest structure
IPPROTO_IPV6	IPV6_RTHDR	ip6_rthdr structure

Some IPv6 implementations may support "sticky" options [1] for the IPv6 destination option for datagram and RAW sockets.

Behavior of Legacy IPv6 Socket Applications:

Legacy IPv6 applications/implementations using the Advanced Socket API [1] mechanisms, upon receiving Home Address destination options or Routing headers(Type 2), will discard the packet as per Sections 4.2 and 4.4 of IPV6 Protocol [3] specification, respectively; otherwise, they should properly handle the Home Address destination option and the Routing Header Type 2 specified in this document.

5.1. Routing Header Access Functions

IPV6 Protocol [3] defines a Routing header extension header for Type 0. Thus, in order to access the IPv6 Routing header Type 2 extension header, one MUST use type = 2 and segment = 1. The following existing functions defined in Advanced API for IPv6 Sockets [1] are supported for Mobile IPv6 applications for sending and receiving Routing Header Type 2 headers:

For Sending:

```
size_t inet6_rth_space(int type, int segments);
void *inet6_rth_init(void *bp, int bp_len, int type, int segments);
int inet6_rth_add(void *bp, const struct in6_addr *addr);
```

For Receiving:

```
int inet6_rth_segments(const void *bp);
struct in6_addr *inet6_rth_getaddr(const void *bp, int index);
```

NOTE: Reversing operation is not possible using the Route Header Type 2 extension header. Thus, inet6_rth_reverse() is not used.

Detailed descriptions and examples of accessing an IPv6 Routing Header are discussed in the Advanced Sockets API for IPv6 [1]. However, Section 7 of Advanced API for IPv6 Sockets [1] indicates that multiple types of routing headers can be received as multiple ancillary data objects to the application (with cmsg_type set to IPV6_RTHDR). Currently, there are no API functions defined to return the routing header type. However, this document does not define a helper function, since it is easy to access the Routing Header Type field just as easily as the ip6r_segleft field. An excerpt of a code sample is provided for extracting the type of the received routing header:

```
if (msg.msg_controllen != 0 &&
    cmsgptr->cmsg_level == IPPROTO_IPV6 &&
    cmsgptr->cmsg_type == IPV6_RTHDR) {
        struct in6_addr *in6;
        char asciiname[INET6_ADDRSTRLEN];
        struct ip6_rthdr *rthdr;
        int    segments, route_type;

        rthdr = (struct ip6_rthdr *)extptr;
        segments = inet6_rth_segments(extptr);
        printf("route (%d segments, %d left): ",
            segments, rthdr->ip6r_segleft);
        route_type = rthdr->ip6r_type;
        if (route_type == 2) {
                printf ("Routing header Type 2 present\n");
        }
}
```

5.2. Content of Type 2 Routing Header

It is recommended that no portable applications send Type 2 Routing Header ancillary data from the application layer, since many implementations take care of that at the kernel layer and may not support the API for sending Type 2 Routing Header.

Mobile IPv6 [2] defines the Type 2 Routing Header to allow the packet to be routed directly from a correspondent to the mobile node's care-of address. The mobile node's care-of address is inserted into the IPv6 Destination Address field. Once the packet arrives at the care-of address, the mobile node retrieves its home address from the routing header, and this is used as the final destination address for the received IPv6 packet.

For user-level applications that receive Type 2 Routing Header, inet6_rth_getaddr() returns the care-of address or on-the-wire destination address of the received packet. This complies with the existing Routing header Type=0 processing for IPv6 [1].

Thus, on the receive side, the socket application will always receive data packets at its original home address. The implementations are responsible for processing the Type 2 Routing Header packet as per Mobile IPv6 RFC [2] before passing the Type 2 Routing Header information to the Socket API.

If a pure IPv6 [3] system receives the Routing Header Type 2 packets, it will follow the process described in Section 4.4 of the IPv6 [3] base specification.

5.3. Order of Extension Headers for Home Address Destination Options

Section 6.3 of Mobile IPV6 [2] defines the extension header order for the Home address destination option.

```
Routing Header
Home Address Destination Option
Fragment Header
AH/ESP Header
```

IPv6 [3] specifies that the destination header can be either before the Routing header or after the AH/ESP header if they are all present.

Thus, when the Home Address destination option is present along with other extension headers, the order will be:

```
Hop-by-Hop Options header
Destination Options header
Routing header
Destination Options [Home Address Option]
Fragment header
Authentication header
Encapsulating Security Payload header
Destination Options header
upper-layer header
```

Any user-level implementation or application that sends the Home address destination option through ancillary data objects should follow the order extension header defined in this document when using IPV6_MIPDSTOPTS socket options.

5.4. Home Address Destination Option Access Functions

The application must enable the IPV6_RECVDSTOPTS socket option in order to receive the Home Address destination option (error checking is not performed in the example for brevity):

```
int  on = 1;

setsockopt(fd, IPPROTO_IPV6, IPV6_RECVDSTOPTS, &on, sizeof(on));
```

Each Destination option header is returned as one ancillary data object described by a cmsghdr structure, with cmsg_level set to IPPROTO_IPV6 and cmsg_type set to IPV6_DSTOPTS.

The received side Home Address destination option is further processed by calling the inet6_opt_next(), inet6_opt_find(), and inet6_opt_get_value() functions as defined in Advanced API for IPv6 sockets [1].

This document assumes that portable Mobile IPv6 applications will not send a Home Address Destination Option from the application level, as the Mobile IPv6 implementation underneath takes care of sending the Home Address option and the routing header type 2 at the kernel. However, some embedded software implementations may implement the IPv6 packet processing/sending at the user-level; those implementations may choose to provide the API support for sending a home-address option at the application layer. In this case, the Home Address destination options are normally constructed by using the inet6_opt_init(), inet6_opt_append(), inet6_opt_finish(), and inet6_opt_set_val() functions, described in Section 10 of the Advanced sockets API for IPv6 [1].

5.5. Content of Home Address Destination Option

The received ancillary data object for the Home Address destination option SHOULD contain the care-of address of the mobile node. It is assumed that the initial processing of the Home Address destination option will verify the validity of the home address, as described in Sections 6.3 and 9.5 of the Mobile IPv6 Specification [2], and swap the source address of the packet (COA) with the contents of Home Address destination option.

Note that whether or not these new APIs are used, the sender’s home address is contained in the source address (which is passed to the application using the socket-level functions recvfrom(), recvmsg(), accept(), and getpeername()). This is necessary for:

maintaining consistency between simple user-level applications running between mobile nodes and the diagnostic applications on the home agent or correspondent node that use this API;

obtaining the COA address of the mobile node when the Home Address destination option is used; and

maintaining consistency of existing IPv6 Socket APIs and processing of the Home Address destination option.

If an implementation supports send-side Home Address destination API, then it must follow the same rule for data content as specified in Mobile IPv6 RFC [2] for sending a home-address option. Thus, the home-address option will contain the home address, and the implementation will use the care-of address as the source address of the outgoing packet. If the implementation uses IPSec, then it should use the content of Home Address destination option as the source address of the packet for security association. Note that regular user applications must not set the home address destination option.

6. Mobility Protocol Headers

Mobile IPv6 [2] defines a new IPv6 protocol header to carry mobility messages between Mobile Nodes, Home Agents and Correspondent Nodes. These protocol headers carry Mobile IPv6 Binding messages as well as Return Routability [2] messages. Currently the specification [2] does not allow transport packets (piggybacking) along with the mobility messages. Thus the mobility protocol header can be accessed through an IPv6 RAW socket. An IPv6 RAW socket that is opened for protocol IPPROTO_MH should always be able to see all the MH (Mobility Header) packets. It is possible that future applications may implement part of Mobile IPv6 signal processing at the application level. Having a RAW socket interface may also enable an application to execute the Return Routability protocol or other future authentication protocol involving the mobility header at the user-level.

6.1. Receiving and Sending Mobility Header Messages

This specification recommends that the IPv6 RAW sockets mechanism send and receive Mobility Header (MH) packets. The behavior is similar to ICMPV6 processing, where the kernel passes a copy of the mobility header packet to the receiving socket. Depending on the implementation, the kernel may process the mobility header in addition to passing the mobility header to the application. In order to comply with the restriction in the Advanced Sockets API for IPv6 [1], applications should set the IPV6_CHECKSUM socket option with

IPPROTO_MH protocol RAW Sockets. A Mobile IPv6 implementation that supports the Mobile IPv6 API must implement Mobility Header API checksum calculations by default at the kernel for both incoming and outbound paths. A Mobile IPv6 implementation must not return error on the IPV6_CHECKSUM socket option setting, even if the socket option is a NO-OP function for that implementation because it verifies the checksum at the kernel level. The Mobility Header checksum procedure is described in the Mobile IPv6 Protocol [2] specification. Again, for application portability it is recommended that the applications set the IPV6_CHECKSUM socket option along with the RAW sockets for IPPROTO_MH protocol.

As an example, a program that wants to send or receive a mobility header protocol(MH) could open a socket as follows (for brevity, the error checking is not performed in the example below):

```
fd = socket(AF_INET6, SOCK_RAW, IPPROTO_MH);

int offset = 4;
setsockopt(fd, IPPROTO_IPV6, IPV6_CHECKSUM, &offset,
     sizeof(offset));
```

For example, if an implementation likes to handle HOTI/HOT and COTI/COT message processing, it can do so by using IPv6 RAW Sockets for IPPROTO_MH at the application layer. The same application may also set the IPV6_RECVDSTOPTS socket option for receiving Home Address destination option in a binding update [2] from the mobile node.

IPv6 RAW sockets are described in Section 3 of the IPv6 Advanced Socket API [1] specification. All data sent and received via raw sockets must be in network byte order. The data structures that are defined in this document are in network byte order, and they are believed to be supported by most compilers to hold packet formats directly for transmission on the wire.

The usual send/recv functions for datagram should be used for the Mobile IPv6 RAW sockets in order to send and receive data, respectively.

7. Protocols File

Many hosts provide the file /etc/protocols, which contains the names of the various IP protocols and their protocol numbers. The protocol numbers are obtained through function getprotoXXX() functions.

The following addition should be made to the /etc/protocols file, in addition to what is defined in Section 2.4 of the Advanced Sockets API for IPv6 [1].

The protocol number for Mobility Header:
(http://www.iana.org/assignments/protocol-numbers)

```
ipv6-mh          135      # Mobility Protocol Header
```

8. IPv4-Mapped IPv6 Addresses

The various socket options and ancillary data specifications defined in this document apply only to true IPv6 sockets. It is possible to create an IPv6 socket that actually sends and receives IPv4 packets, using IPv4-mapped IPv6 addresses, but the mapping of the options defined in this document to an IPv4 datagram is beyond the scope of this document. The above statement is in compliance with Section 13 of the IPv6 Socket API [1].

9. Security Considerations

The setting of the Home Address Destination option and Route Header Type 2 IPV6_RTHDR socket option may not be allowed at the application level in order to prevent denial-of-service attacks or man-in-the-middle attacks by hackers. Sending and receiving of mobility header messages are possible by IPv6 RAW sockets. Thus, it is assumed that this operation is only possible by privileged users. However, this API does not prevent the existing security threat from a hacker sending a bogus mobility header or other IPv6 packets using the Home Address option and Type 2 Routing Header extensions.

10. IANA Considerations

This document does not define a new protocol. However, it uses the Mobility Header Protocol for IPv6 to define an API for the /etc/protocols file. (ref: http://www.iana.org/assignments/protocol-numbers)

11. Acknowledgements

Thanks to Brian Haley for the thorough review of this document and many helpful comments. Keiichi Shima, Alexandru Petrescu, Ryuji Wakikawa, Vijay Devarapalli, Jim Bound, Suvidh Mathur, Karen Nielsen, Mark Borst, Vladislav Yasevich, and other mobile-ip working group members provided valuable input. Antti Tuominen suggested the routing header type function for this API document. During IESG review, Bill Fenner suggested accessing the routing header type directly for being consistent with RFC3542. A new socket option for Home Address Destination Option is added per Bill Fenner’s suggestion for clarity of extension header orders. Thanks to Thomas Narten and Jari Arkko for the review of this document.

12. References

12.1. Normative References

[1] Stevens, W., Thomas, M., Nordmark, E., and T. Jinmei, "Advanced Sockets Application Program Interface (API) for IPv6", RFC 3542, May 2003.

[2] Johnson, D., Perkins, C., and J. Arkko, "Mobility Support in IPv6", RFC 3775, June 2004.

12.2. Informative References

[3] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[4] Devarapalli, V., Wakikawa, R., Petrescu, A., and P. Thubert, "Network Mobility (NEMO) Basic Support Protocol", RFC 3963, January 2005.

[5] Nordmark, E., "IPv6 Socket API for source address selection", Work in Progress, July 2005.

[6] Gilligan, R., Thomson, S., Bound, J., McCann, J., and W. Stevens, "Basic Socket Interface Extensions for IPv6", RFC 3493, February 2003.

Authors' Addresses

Samita Chakrabarti

EMail: samitac2@gmail.com

Erik Nordmark
Sun Microsystems
17 Network Circle
Menlo Park, CA 94025
USA

Phone: +1 650 786 2921
EMail: erik.nordmark@sun.com

Full Copyright Statement

Copyright (C) The Internet Society (2006).

This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Acknowledgement

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).