Network Working Group
Internet-Draft
Expires: August 5, 2006

M. Komu
HIIT
M. Bagnulo
UC3M
K. Slavov
S. Sugimoto, Ed.
Ericsson
February 2006

Socket Application Program Interface (API) for Multihoming Shim
draft-sugimoto-multihome-shim-api-00

Status of this Memo

Copyright Notice

Abstract

This document specifies a socket API for the multihoming shim layer. The API aims to enable interactions between the applications and the multihoming shim layer for advanced locator management and also for accessing to information about failure detection and path

exploration.

This document is based on an assumption that a multhomed host is equipped with a 'shim' layer which essentially maintains mappings between identifiers and locators at the IP layer. SHIM6 and HIP are examples of this shim layer. Hence, the API can be commonly used by SHIM6 and HIP.

Table of Contents

1. Introduction

This document specifies a socket API for the multihoming shim layer. The API aims to enable interactions between application and the multihoming shim layer for advanced locator management and for accessing to information about failure detection and path exploration.

This document is based on an assumption that a multhomed host is equipped with a 'shim' layer which essentially maintains mapping between identifiers and locators at the IP layer. SHIM6 and HIP are examples of the shim. Hence, the API can be commonly used by SHIM6 and HIP.

We suggest that the ID/Locator adaptation is done only once inside the network stack. In other words, if there exist multiple shim sublayers at the IP layer, any one of them should be exclusively applied for a given flow.

We try to make this document be in line with Posix standard [POSIX] as much as possible. And the API defines how to use ancillary data (aka cmsg) to access locator information with recvmsg() and/or sendmsg() I/O calls. Definition of API is presented in C language and data types follow Posix format: intN_t means a singed integer of exactly N bits (e.g. int16_t) and uintN_t means an unsigned integer of exactly N bits (e.g. uint32_t).

2. Target

The primary target reader of this document is application programmers who develop application software which may run on top of a multihomed environment. In particular, the API should be beneficial for application development of the software which takes advantage of multihomed environment aiming to achieve better failover.

Secondly, this document should be of interest for the developers of a given protocol stack for the shim layer (e.g. SHIM6 and HIP). This is because this document specifies what kinds of information exchange should be possible between the applications and the shim layer.

3. Terminology

This document does not intend to give new definitions for technical terms that are relevant to multihomed environments but tries to inherit definitions provided in the existing documents as listed below:

- SHIM6 Protocol Specification[I-D.ietf-shim6-proto]
- HIP Architecture[I-D.ietf-hip-arch]
- Reachability Protocol (REAP)[I-D.ietf-shim6-failure-detection]

For clarification, we provide definition for the terms that are frequently used in this document:

- Endpoint Identifier (EID) - An identifier used by the application to specify an endpoint of the communication. As addressed in [I-D.ietf-shim6-app-refer], application may handle and EID in various ways in different types of communication models such as long-lived connections, callbacks, and referrals.
  - In case of SHIM6, the EID is called ULID. The ULID is chosen from one of the locators available on the host.
  - In case of HIP, the EID is essentially a public key of the host. In order to preserve backward compatibility with legacy applications, a hash of public key called Host Identity Tag (HIT) is used by the applications as a handle for the EID.
- Locator - An IP address actually used to deliver IP packets. Locators should be present in the source and destination fields of the IP header of a packet that appears on wire. Normally, a locator is assigned to the network interface of the host. And the IP packet destined to a given locator is delivered to the correspondent network interface by the routing system.
- Shim - A conceptual (sub-)layer inside the IP Layer which maintains mappings of EIDs and locators. An EID can be associated with more than one locator at a time when the host is multihomed. It should be noted that the term 'shim' does not refer to a specific protocol but refers to the generic concept of a layer that enables the mapping between identifiers and locators. SHIM6 and HIP are examples of the shim.
- Context - A state information shared by the peers, which essentially stores a binding between the EIDs and associated locators. The context is maintained at the shim layer of the host.
- List of Locators - A list of locators associated with an EID. There are two lists of locators stored in a given context, one is associated with the local EID and the other is associated with the remote EID. As defined in [I-D.ietf-shim6-proto], the list of locators associated with an EID 'A' can be denoted as Ls(A).
- Preferred Locator - The (source/destination) locator currently used to send packets. As defined in [I-D.ietf-shim6-proto], the preferred locator of a host which EID is 'A' can be denoted as Lp(A).
- Reachability Detection - A procedure to detect reachability between a given locator pair.

- Path - A sequence of routers that an IP packet goes through to reach the destination.
- Path Exploration - A procedure to explore available paths for a given set of locator pairs.
- Outage - An incident meaning that the reachability among a given locator pair is lost. The outage could be caused by any kind of problems inside the routing infrastructure and/or problems of the network interfaces of the end hosts.
- Working Address Pair - An address pair is said to be working if the packet containing the first address from the pair as source address and the second address from the pair as destination address can safely travel from the source to the destination. If the reachability is confirmed in both directions, the address pairs is said to be bi-directional. Otherwise, it's unidirectional.
- REAP - A protocol for detecting failure and exploring reachability in a multihomed environment. REAP is defined in[I-D.ietf-shim6-failure-detection].
- Endpoint Descriptor (ED) - The representation of endpoints is hidden from the applications. ED is a "handle" or "pointer" to the actual EID.

4. System Overview

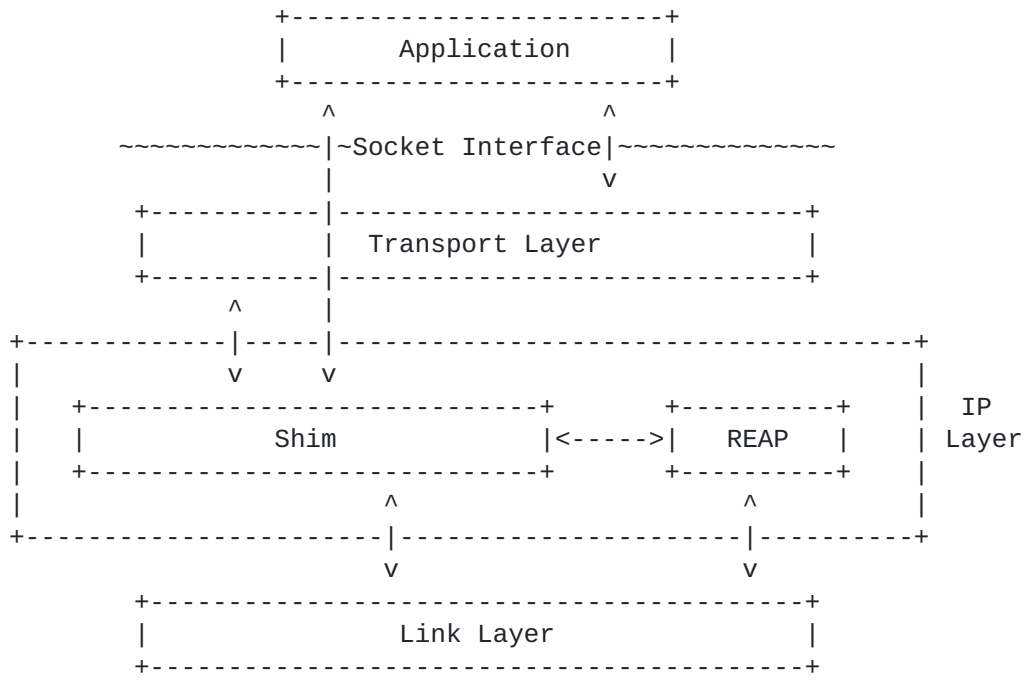

Figure 1: System overview

Figure 1 illustrates the system overview. The application can use the socket API to interact with the shim layer and the transport layer for better control of locator management, failure detection and path exploration.

Inside the IP layer, there is the shim which closely interacts with REAP component. There could be interactions between the shim and the transport layer, however they are outside of scope of this document. The scope of this document is an interface from the application to the shim layer, which is enabled via the socket interface.

5. Requirements

The list of requirements from the application perspective is the following. These requirements are mainly identified during the discussions on SHIM6 WG mailing list. Some requirements are derived from Reachability Protocol document[I-D.ietf-shim6-failure-detection].

- Locator management. The shim layer selects a pair of locators for sending IP packets within a given context. The selection is made by taking miscellaneous conditions into account such as reachability of the path, application's preference, and characteristics of path. From the application's perspective:
  - It should be possible to obtain the lists of locators of a given context: Ls(local) and Ls(remote).
  - It should be possible to obtain the preferred locators of a given context: Lp(local) and Lp(remote).
- Notification from the application to the shim layer about the status of the communication. Note that the notification is made in an event based manner. There are mainly two aspects of the feedback that application or upper layer protocol may provide for the shim layer, positive and negative feedbacks [NOTE: These feedbacks are addressed in section 4.3 and section 5.2 of REAP specification]:
  - Positive feedback could be given by the application or upper layer protocol (e.g. TCP) to the shim layer informing that the communication is going well.
  - Negative feedback could be given by the application or upper layer protocol (e.g. TCP) to the shim layer informing that the communication status is not satisfactory. TCP could detect a problem when it does not receives expected ACK from the peer. ICMP error messages delivered to the upper layer protocol could be a clue for application to detect potential problems. REAP module may be triggered by these negative feedbacks and invoke

procedure of path exploration.

- Feedback from application to shim layer. The application should be able to inform the shim layer about the timeout values for detecting failures, for sending keepalives, for starting the exploration procedure. In particular, the application should be able to suppress the keepalives.
- Hot-standby. The application may request the shim layer for hot-standby capabilities. In this case, alternative paths are known to be working before a failure is detected. Hence it is possible for the host to immediately replace the current locator pair with the alternative locator pair. Hot-standby may allow applications to achieve better failover.
- Eagerness of locator exploration. The application should be able to inform the shim layer how aggressive it wants REAP mechanism to perform path exploration (e.g. specifying the number of concurrent attempts of discovering working locator pair) when an outage occurs on the path between the currently selected locator pair.
- Providing locator information to application. The application should be able to obtain information about the locator pair which was actually used to send or receive the packet.
  * For inbound traffic, the application may be interested in the locator pair which was actually used to receive the packet.
  * For outbound traffic, the application may be interested in the locator pair which was actually used to transmit the packet.

  In this way, the application may have additional control on the locator management. For example, the application can verify if its preference of locator is actually applied to the flow or not.
- The application should be able to specify if it wants to defer the context setup or if it wants context establishment to be started immediately in case there is no available context. With deferred context setup, there should be no additional delay imposed by context establishment in initiation of communication.
- Turn on/off shim. The application should be able to request to turn on/off the multihoming support by the shim layer:
  * Apply shim. The application should be able to explicitly request the shim layer to apply multihoming support.
  * Don't apply shim. The application should be able to request the shim layer not to apply the multihoming support but to apply normal IP processing at the IP layer.
- The application should be able to know if the communication is now served by the shim layer or not.
- The application should be able to access locator information regardless of its address family. In other words, no matter the target locator is IPv4 or IPv6, the application should be able to use common interface to access the locator information.

6. Socket Options for Multihomed Shim Layer

In this section, the socket options for the interface between the application and the multihomed shim layer are defined. These options can be used either by getsockopt() and/or setsockopt() system calls for an opened socket. Table 1 provides a list of the socket options. Note that all socket options are defined at level SOL_SHIM.

The first column of the table gives the name of the option. The second and third columns indicates whether if the option is for getsockopt() and/or setsockopt(), respectively. The fourth column provides a brief description of the socket option. The fifth column shows the data structure specified with the socket option, which can store an argument for setsockopt() and result for getsockopt(). By default, the data structure is an integer.

optname	get	set	description	dtype
SHIM_ASSOCIATED	o		Check if the socket is associated with any shim context or not.	int
SHIM_DONTSHIM	o	o	Request the shim layer not to apply any multihoming support for the communication.	int
SHIM_HOT_STANDBY		o	Request the shim layer to prepare a hot-standby connection (in addition to the current path).	int
SHIM_LOC_LOCAL_PREF	o	o	Get or set the preferred locator on the local side for the context associated with the socket.	*1

SHIM_LOC_PEER_PREF	o	o	Get or set the preferred locator on the remote side for the context associated with the socket.	*1
SHIM_LOC_LOCAL_RECV		o	Request for the destination locator of the received IP packet.	int
SHIM_LOC_PEER_RECV		o	Request for the source locator of the received IP packet.	int
SHIM_LOCLIST_LOCAL	o	o	Get or set a list of locators associated with the local EID.	*1
SHIM_LOCLIST_PEER	o	o	Get or set a list of locators associated with the peer's EID.	*1
SHIM_APP_TIMEOUT		o	Inform the shim layer about a timeout value for detecting failure.	int
SHIM_FEEDBACK_POSITIVE		o	Provide a positive feedback to the shim layer.	int
SHIM_FEEDBACK_NEGATIVE		o	Provide a negative feedback to the shim layer.	*2
SHIM_PATHEXPLORE		o	Specify how path exploration should be performed in case of failure.	*3

SHIM_CONTEXT_DEFERRED_SETUP		o	Specify if the context setup can be deferred or not.	int	
SHIM_IF_RECV		o	Request for a receiving interface.	int	
SHIM_IF_SEND		o	Request for an outgoing interface.	int	

Table 1: Shim specific socket options for getsockopt() and setsockopt()

*1: Pointer to the buffer which stores arrays of locator information. The buffer is actually the chained list of addrinfo structure.

*2: TBD.

*3: TBD.

Figure 2 illustrates how the shim specific socket options fit into the system model of socket API. In the figure, it can be seen that the shim layer and the additional protocol components (IPv4 and IPv6) below the shim layer are new to the system model. As previously mentioned, all the shim specific socket options are defined at SOL_SHIM level. This design choice brings the following advantages:

1. It is assured that the existing socket API continue to work at the layer above the shim layer. That is, those legacy API deal with 'identifier' aspect of the IP addresses.
2. With newly defined socket options for the shim layer, the application obtains additional control on locator management.
3. The shim specific socket options are not specific to any address family (IPPROTO_IP or IPPROTO_IPV6) nor any transport protocol (SOCK_STREAM or SOCK_DGRAM or SOCK_RAW).

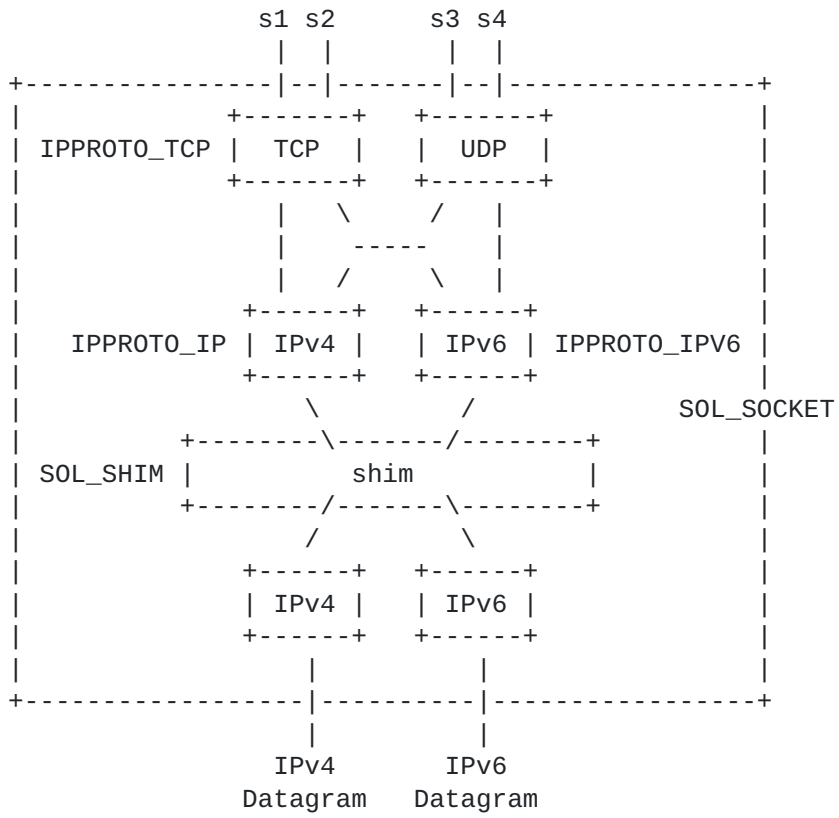

Figure 2: System model of socket API with shim layer

6.1. SHIM_ASSOCIATED

This option can be specified by getsockopt() to check if the socket is associated with a shim context or not. Thus, the option is read-only and the result (0 or 1) is set in optval. A returned value 1 means that the socket is associated with a given shim context at the shim layer, while a return value 0 indicates that there is no context associated with the socket.

This option is particularly meaningful in a case where locator information of the received IP packet is not enough for identifying if the ID/Locator adaptation is performed or not. Note that the EID pair and locator pair maybe identical in some case.

ISSUE: Should we limit this option only for 'connected' socket ?

6.2. SHIM_DONTSHIM

This option can be specified either by getsockopt() or setsockopt().

The application can specify the option by setsockopt() taking the argument optval with value 1 to request the shim layer not to apply any multihoming support for the communication. The application can also obtain the current setting by specifying the the socket option in getsockopt(). The result should be binary (0 or 1).

By default, the value is set to 0, meaning that the shim layer will try to apply ID/Locator adaptation for the communication over a given socket.

Once the socket option is specified by setsockopt(), it remains effective until it is deactivated (sticky option).

6.3. SHIM_HOT_STANDBY

This option can be specified by setsockopt().

By setting 1 in the optval for the setsockopt(), the application can request the shim layer to use a hot-standby connection. The hot-standby connection is another working locator pair than the current locator pair.

By default, the value is set to 0, meaning that hot-standby connection is disabled.

Once the socket option is specified by setsockopt(), it remains effective until it is deactivated (sticky option).

6.4. SHIM_PATHEXPLORE

This option can be specified either by setsockopt() or getsockopt(). The value specified by the option indicates how aggressive the application wants path exploration to be performed in case of failure. Therefore, this option is effective only when there is associated shim context for the socket.

The information to be provided by this socket option should contain: suggested number of attempts for path exploration, frequency of the path exploration, and so on. Need further discussions.

The data type for the argument optval is TBD.

Once the socket option is specified by setsockopt(), it remains effective until it is deactivated (sticky option).

6.5. SHIM_LOC_LOCAL_PREF

This option can be specified either by setsockopt() or getsockopt().

When specified by setsockopt(), the preferred locator on local side is explicitly given to the shim layer. The shim layer shall accordingly update the preferred locator of the context associated with the socket.

When specified by getsockopt(), the preferred locator on local side is returned by the shim layer.

An error ENOSHIMCONTEXT will be returned when there is no context associated with the socket.

Once the socket option is specified by setsockopt(), it remains effective until it is deactivated (sticky option).

6.6. SHIM_LOC_PEER_PREF

This option can be specified either by setsockopt() or getsockopt().

When specified by setsockopt(), the preferred locator on remote side is explicitly given to the shim layer. The shim layer shall accordingly update the preferred locator of the context associated with the socket.

When specified by getsockopt(), the preferred locator on remote side is returned by the shim layer.

An error ENOSHIMCONTEXT will be returned when there is no context associated with the socket.

An error EINVALIDLOCATOR will be returned when the validation of the specified locator failed.

Once the socket option is specified by setsockopt(), it remains effective until it is deactivated (sticky option).

6.7. SHIM_LOC_LOCAL_RECV

This option can be specified by setsockopt().

When specified by setsockopt(), the shim layer stores the destination locator of the received IP packet in an ancillary data object which can be accessed by recvmsg(). The argument optval value should be set to 1.

An error ENOSHIMCONTEXT will be returned when there is no context associated with the socket.

Once the socket option is specified by setsockopt(), it remains

effective until it is deactivated (sticky option).

6.8. SHIM_LOC_PEER_RECV

This option can be specified by setsockopt().

When specified by setsockopt(), the shim layer stores the source locator of the received IP packet in an ancillary data object which can be accessed by recvmsg(). The argument optval value should be set to 1.

An error ENOSHIMCONTEXT will be returned when there is no context associated with the socket.

Once the socket option is specified by setsockopt(), it remains effective until it is deactivated (sticky option).

6.9. SHIM_LOCLIST_LOCAL

This option can be specified either by getsockopt() or setsockopt().

When specified by setsockopt(), the application provides a list of locators which is associated with the local EID to the shim layer. Accordingly, the shim layer shall update the list of locators Ls(local). The argument optval should contain a pointer to the buffer in which a list of locators are stored. See Section 8 for detail.

When specified by getsockopt(), the application obtains a list of locators which is associated with the local EID.

An error ENOSHIMCONTEXT will be returned when there is no context associated with the socket.

Once the socket option is specified by setsockopt(), it remains effective until it is deactivated (sticky option).

6.10. SHIM_LOCLIST_PEER

This option can be specified either by getsockopt() or setsockopt().

When specified by setsockopt(), the application provides a list of locators which is associated with the remote EID to the shim layer. Accordingly, the shim layer shall update the list of locators Ls(remote). The argument optval should contain a pointer to the buffer in which a list of locators are stored. See Section Section 8.1for detail.

When specified by getsockopt(), the application obtains a list of locators which is associated with the remote EID.

An error ENOSHIMCONTEXT will be returned when there is no context associated with the socket.

Once the socket option is specified by setsockopt(), it remains effective until it is deactivated (sticky option).

6.11. SHIM_APP_TIMEOUT

This option can be specified by setsockopt().

The application can inform the shim layer about the timeout value for detecting failure. The argument optval should contain the timeout value in seconds. Accordingly, the shim layer shall update the strategy for reachability test. Especially, this is efficient in a case where the informed timeout value is shorter than the interval of keepalive. In such case, keepalives to be performed by REAP may be suppressed.

An error ENOSHIMCONTEXT will be returned when there is no context associated with the socket.

Once the socket option is specified by setsockopt(), it remains effective until it is deactivated (sticky option).

6.12. SHIM_FEEDBACK_POSITIVE

This option can be specified by setsockopt().

The application can simply inform the shim layer that its communication is going well. The argument optval should be set to 1.

An error ENOSHIMCONTEXT will be returned when there is no context associated with the socket.

6.13. SHIM_FEEDBACK_NEGATIVE

This option can be specified by setsockopt().

The application can inform the shim layer that its communication is not going well. The argument optval should be TBD.

An error ENOSHIMCONTEXT will be returned when there is no context associated with the socket.

6.14. SHIM_DEFERRED_CONTEXT_SETUP

This option can be specified by setsockopt().

When specified by the setsockopt(), optval should be set to 1 if the context setup can be deferred. Otherwise, the context setup is invoked immediately when there is no shim context setup for the flow. By default, the value is set to 1.

It should be noted that in some case, deferred context setup is not possible; given EID is non-routable address and there is no way to transmit any IP packet unless there is a context providing the locators. In such case, context establishment should be made prior to communication.

6.15. SHIM_IF_RECV

This option can be specified by setsockopt().

The application can request the shim layer to provide information about interface from which the packet was received. Once the socket option is successfully set, the interface information can be obtained by recvmsg() from the ancillary data. The argument optval should be set to 1.

An error ENOSHIMCONTEXT will be returned when there is no context associated with the socket.

Once the socket option is specified by setsockopt(), it remains effective until it is deactivated (sticky option).

6.16. SHIM_IF_SEND

This option can be specified by setsockopt().

The application can specify outgoing interface of the outbound traffic over the socket. Application should specify the requested interface identifier in the argument optval. Alternatively, this option can also be specified in ancillary data in along with sendmsg() call.

Once the socket option is specified by setsockopt(), it remains effective until it is deactivated (sticky option).

6.17. Error Handling

If successful, getsockopt() and setsockopt() return 0; otherwise, the functions return -1 and set errno to indicate error.

Following are errno codes newly defined for some shim specific socket options indicating that the getsockopt() or setsockopt() finished incompletely:

ENOSHIIMCONTEXT
There is no shim context associated with the socket.
EINVALIDLOCATOR
This indicates that at least one of the necessary validations inside the shim layer for the specified locator has failed. In case of SHIM6, there are two kinds of verifications required prior sending an IP packet to the peer's new address; one is return routability (check if the peer is actually willing to receive data with the specified locator) and the other is verifications based on given crypto identifier mechanisms[RFC3972], [I-D.ietf-shim6-hba].

7. Access to Locator Information

In this section, the way how to access locator information with some I/O calls is presented. As defined in Posix, sendmsg() and recvmsg() take msghdr structure as its argument and they can additionally handle control information in along with data. Figure 3 shows the msghdr structure which is defined in <sys/socket.h>. msg_control member holds a pointer to the buffer where the shim specific ancillary data objects are stored.

```
struct msghdr {
        caddr_t msg_name;       /* optional address */
        u_int   msg_namelen;    /* size of address */
        struct  iovec *msg_iov; /* scatter/gather array */
        u_int   msg_iovlen;     /* # elements in msg_iov */
        caddr_t msg_control;    /* ancillary data, see below */
        u_int   msg_controllen; /* ancillary data buffer len */
        int     msg_flags;      /* flags on received message */
};
```

Figure 3: msghdr structure

ISSUE: Should we introduce a new flag for msg_flags (e.g. MSG_SHIMMED) ? Following the practice, it seems reasonable to do so, but not sure how much it is useful.

The buffer pointed from the msg_control member of the msghdr structure should contain a single locator and it should be possible to process them with the existing macros defined in Posix and [RFC3542]. Each cmsghdr{} should be followed by a data which stores a single locator.

In case of non-connected socket, msg_name member stores the socket address of the peer which should be considered as an identifier rather than a locator. The locator of the peer node should be retrieved by SHIM_LOC_PEER_RECV as specified below.

Table 2 is a list of the shim specific ancillary data which can be used for recvmsg() or sendmsg(). In any case, SOL_SHIM must be set as cmsg_level.

cmsg_type	sendmsg()	recvmsg()	cmsg_data[]
SHIM_LOC_LOCAL_RECV		o	*1
SHIM_LOC_PEER_RECV		o	*1
SHIM_LOC_LOCAL_SEND	o		*1
SHIM_LOC_PEER_SEND	o		*1
SHIM_IF_RECV		o	int
SHIM_IF_SEND	o		int

Table 2: Shim specific ancillary data

*1: cmsg_data[] should include padding (if necessary) and a single sockaddr_storage{} a protocol independent placeholder for socket addresses.

ISSUE: Is the design choice (to use sockaddr_storage{}) reasonable ?

It should be noted that the above ancillary data can only be handled in UDP and raw sockets, not in TCP sockets. As explained in [RFC3542], there is no one-to-one mapping of send/receive operations and the TCP segments being transmitted/received. In case of TCP, application may use setsockopt() or getsockopt() to access or specify some of locator information provided by the shim layer.

7.1. Get Locator Information from Incoming Packet

Application can get locator information from the received IP packet by specifying the shim specific socket options for the socket. When SHIM_LOC_LOCAL_RECV and/or SHIM_LOC_PEER_RECV socket options are set, the application can retrieve local and/or remote locator from the ancillary data.

In addition, the application can get the receiving interface from the ancillary data marked with SHIM_IF_RECV. The ancillary data should contain an interface identifier of the physical interface which was actually used to receive the packet.

7.2. Specify Locator Information for Outgoing Packet

Application can specify the locators to be used for transmitting an IP packet by sendmsg(). When ancillary data of cmsg_type SHIM_LOC_LOCAL_SEND and/or SHIM_LOC_PEER_SEND are specified, the application can explicitly specify source and/or destination locators to be used for the communication over the socket.

In addition, the application can specify the outgoing interface by SHIM_IF_SEND ancillary data. The ancillary data should contain an interface identifier of the physical interface over which the application expect the packet to be transmitted.

Note that the effect is limited to the datagram transmitted by the sendmsg(). If the specified locator pair seem to be valid, the shim layer overrides the locator of the IP packet as requested.

An error EINVALIDLOCATOR will be returned when validation of the specified locator failed.

ISSUE: Is there any other type of error that we should specifically handle ?

8. Data Structures

Some of the socket options defined in this document requires specific data structures for exchanging information. Those data structures are illustrated in this section.

8.1. Placeholder for Locator Information

Some of the socket options defined in this document handle locator information. Locator information could be a single locator or an array of locators. An important requirement is that the locator information should be handled in a protocol independent manner. In other words, an interface to the locator information should not be dependent to any address family.

8.1.1. addrinfo structure

addrinfo structure in along with getaddrinfo() function are defined in Posix, which is useful for programming applications in protocol independent manner. As mentioned earlier, protocol independency is required for the locator management at the shim layer, thus we propose to use addrinfo structure as a placeholder for locators.

A chain of addrinfo structures can be used to represent a list of

locators. Note that addrinfo structure itself does not contain the locator data but it holds a pointer to sockaddr structure where the actual data of a given locator is stored. Figure 4 illustrates the addrinfo structure defined in <netdb.h>.

```
struct addrinfo {
        int ai_flags;           /* input flags */
        int ai_family;          /* protocol family for socket */
        int ai_socktype;        /* socket type */
        int ai_protocol;        /* protocol for socket */
        socklen_t ai_addrlen;   /* length of socket-address */
        struct sockaddr *ai_addr; /* socket-address for socket */
        char *ai_canonname;     /* canonical name for
                                   service location */
        struct addrinfo *ai_next; /* pointer to next in list */
};
```

Figure 4: addrinfo structure

8.1.2. sockaddr_storage structure

[RFC3493] specifies a protocol independent placeholder for socket address, called sockaddr_storage structure as shown in Figure 5. By definition, the structure can store socket address of any protocol (IPv4 or IPv6) and is simply suitable for a placeholder for the locator information. In this document, we suggest to use sockaddr_storage structure to store the locator information to be specified in the ancillary data. In those cases, the locator information is a single locator.

```
/*
 * Desired design of maximum size and alignment
 */
#define _SS_MAXSIZE    128  /* Implementation specific max size */
#define _SS_ALIGNSIZE  (sizeof (int64_t))
                  /* Implementation specific desired alignment */
/*
 * Definitions used for sockaddr_storage structure paddings design.
 */
#define _SS_PAD1SIZE   (_SS_ALIGNSIZE - sizeof (sa_family_t))
#define _SS_PAD2SIZE   (_SS_MAXSIZE - (sizeof (sa_family_t) +
                              _SS_PAD1SIZE + _SS_ALIGNSIZE))

    struct sockaddr_storage {
        sa_family_t  ss_family;    /* address family */
        /* Following fields are implementation specific */
        char      __ss_pad1[_SS_PAD1SIZE];
        int64_t   __ss_align;
        char      __ss_pad2[_SS_PAD2SIZE];
    };
```

Figure 5: sockaddr_storage structure

9. Implications for Existing Socket API Extensions

As the socket options proposed in this document allow the application to specify the locators for transmitting IP packet, there may be conflict with some of the existing socket API. As stated in Section 6, a basic assumption is that the legacy API should continue to work above the shim layer.

In IPv4, application can obtain the destination IP address of the received IP packet (IP_RECVDSTADDR) as well as the receiving interface (IP_RECVIF). If the shim layer performs ID/Locator adaptation for the received packet, the destination EID should be stored in the ancillary data (IP_RECVDSTADDR). Accordingly, the receiving interface should be aligned with the destination EID of the packet. That is, the shim layer should set appropriate interface to which the destination EID is assigned in the ancillary data object. However, from the application perspective, knowing the receiving interface which is associated with the destination EID may not be useful, especially in the case where application is particularly interested in the characteristics of the receiving interface. Hence, we suggest application programmer to use SHIM_IF_RECV instead of IP_RECVIF in such case.

In IPv6, [RFC3542] defines that IPV6_PKTINFO can be used to specify

source IPv6 address and the outgoing interface for outgoing packets, and retrieve destination IPv6 address and receiving interface for incoming packets. This information is stored in ancillary data being IPV6_PKTINFO specified as cmsg_type. Now, similar to the case of IPv4, the shim layer may affect the behavior of socket API which deals with IPV6_PKFINFO. We again would like note that existing socket API should continue to work above the shim layer, that is, the IP addresses handled in IPV6_PKTINFO should be EIDs, not the locators. Hence we recommend application programmers to use shim specific socket options (SHIM_IF_RECV or SHIM_IF_SEND) if the interest in the communicating interface comes from lower level (e.g. characteristics of physical interface). For the same reason, in order to handle locator information, we suggest to use shim specific socket options defined in Section 7.

In summary, a care should be taken in potential conflict with existing socket API which treats the IP address as a locator rather than identifier. Basic assumption is that the existing socket API works above the shim layer.

10. Discussion

In this section, open discussion issues are noted.

10.1. Issues with a Context Shared by Applications

A context is by definition, system-wide. This means that a context could be shared by applications whose communications are using the same EID pair.

When a context is shared by applications, there may be some problems when the shim layer needs to handle feedbacks from the multiple applications. As mentioned in Section X, an application may provide the shim layer feedback about timeout values from its own settings. This implies that there is potentially a race condition at the shim layer.

First of all, the socket options must be used with a proper privilege. Feedback from the application which is run under a root privilege must always override the feedback provided by application which is run under normal user privilege.

For other cases, one could rely on a kind of heuristics of the configuration. For instance, prioritizing feedback with higher demand (e.g. timeout value 300 seconds are more demanding then timeout value 600 seconds) may make sense in some cases. However, it is still an open issue what kind of timer value could be handled in

this way.

Further discussions are needed how the shim layer can accommodate feedbacks from multiple applications within a same context.

10.2. Issues with Shim Unaware Application

In multihomed environment where either or both of the peers have multiple locators, there are some issues with shim unaware application which uses legacy socket API.

10.2.1. Initial Contact with Multiple Locator Pairs

In a connection oriented communication, the connect() system call is used to make the initial contact to the peer, which typically requires IP address and port number to specify the endpoint. Hence, name-to-address resolution should be performed prior to connect(). The application needs to resolve the FQDN of the peer to an IP address by any available name-to-address conversion method.

In typical case, the application receives information from the resolver. If the application ends up with receiving multiple IP addresses to reach the peer, it should iterate through each destination address one-by-one. It should be noted that the host may also have multiple source addresses.

The different resulting address pairs may have different reachability status so, in order to find a working address pair, it may be required to explore all the available address pairs (as opposed to explore all available destination addresses).

In normal case, the application issues a connect() by specifying the resolved IP address of the peer. If the connect() fails, it iterates through the available IP addresses one by one sequentially until working pair is found. Another approach is to initiate concurrent connect() with every locator of the peer. connect() can also be called in a sequence which would probably require more time to find the working pair.

There is a case where involvement of the shim layer is expected for handling initial contact. In such case, behavior of the shim layer will depend on presence of the required context. This case occurs when there exists a context for the EID specified in connect(), the initial contact can be made in accordance with the context information. Otherwise, the shim layer should invoke context establishment with the peer EID specified in the argument for connect().

Additional efforts would be required in a case where the peer cannot be reachable through the EID (for example, EID is non-routable or non-IP reachable) but it can be reached through alternative locator. In particular, the shim layer should somehow discover the alternate locator for the EID to establish context. [I-D.nordmark-shim6-esd] addresses the possible approach to perform reverse DNS lookup from EID to FQDN, then perform forward lookup again to find the full-set of locators and EID.

In HIP, resolving HITs to IP addresses using DNS is not feasible because HITs do not contain any hierarchical information. To mitigate this problem, there are a few alternatives. Firstly, resolver library on end-host can be modified to provide HIT-to-IP mappings for HIP software module. Secondly, a distributed hash table (DHT) service can be used for storing and looking up the mappings because it supports non-hierarchical identifiers, such as HITs [I-D.ietf-hip-arch]. Thirdly, it is possible to use IP addresses in legacy applications as described in [I-D.henderson-hip-applications].

10.2.2. Naming at Socket Layer

getsockname() and getpeername() system calls are used to obtain the 'name' of endpoint which is actually a pair of IP address and port number assigned to a given socket. getsockname() is used when an application wants to obtain the local IP address and port number assigned for a given socket instance. getpeername() is used when an application wants to obtain the remote IP address and port number.

The above is based on a traditional system model of the socket API where an IP address is expected to play both the role of identifier and the role of locator.

In a system model where a shim layer exists inside the IP layer, both getsockname() and getpeername() deal with identifiers, namely EIDs. In this sense, the shim layer serves to (1) hide locators and (2) provide access to the identifier for the application over the legacy socket APIs.

10.3. Additional Requirements from Application

At the moment, it is not certain if following requirements are common in all the multihomed environments (SHIM6 and HIP). These are mainly identified during discussions made on SHIM6 WG mailing list.

- The application should be able to set preferences for the locators, local and remote one and also to the preferences of the local locators that will be passed to the peer.

10.4. Issues of Header Conversion among Different Address Family

The shim layer performs ID/Locator adaptation. Therefore, in some case, the whole IP header can be replaced with new IP header of a different address family (e.g. conversion from IPv4 to IPv6 or vice versa). Hence, there is an issue how to make the conversion with minimum impact. Note that this issue is common in other protocol conversion such as SIIT[RFC2765].

As addressed in SIIT specification, some of the features (IPv6 routing headers, hop-by-hop extension headers, or destination headers) from IPv6 are not convertible to IPv4. In addition, notion of source routing is not exactly the same in IPv4 and IPv6. Hence, there is certain limitation in protocol conversion between IPv4 and IPv6.

The question is how should the shim layer behave when it is face with limitation problem of protocol conversion. Should we introduce new error something like ENOSUITABLELOCATOR ?

10.5. Handling of Unknown Locator Provided by Application

There might be a case where application provides the shim layer new locator with the SHIM_LOC_*_PREF socket options or SHIM_LOC_*_SEND ancillary data. Then there is a question how should the shim layer treat the new locator informed by the application.

In principle, locator information are exchanged by the shim protocol. However, there might be a case where application acquires information about the locator and prefers to use it for its communication.

11. IANA Considerations

This document contains no IANA consideration.

12. Security Considerations

This document does not specify any security mechanism for the shim layer. Fundamentally, the shim layer has a potential to impose security threats, as it changes the source and/or destination IP addresses of the IP packet being sent or received. Therefore, the basic assumption is that the security mechanism defined in each protocol of the shim layer is strictly applied.

13. Conclusion

In this document, the Application Program Interface (API) for multihomed shim layer is specified. The socket API allows applications to have additional control on the locator management and interface to the REAP mechanism inside the shim layer. The socket API is expected to be useful for the application that takes full advantage of multihomed environment. From architectural perspective, the socket API aims to enhance software development environment in terms of support for separation of identifier and locator. That is, with new API, application can handle identifier and locator separately still being allowed to use legacy socket API.

Shim specific socket options can be used by getsockopt() and/or setcokopt() system calls, which allows applications to get information about locator management. Additionally, applications can specify locator information for outgoing packet and get locator information from incoming packet by using ancillary data.

14. Acknowledgments

Jari Arkko participated in the discussion that lead to the first version of this document.

15. References

15.1. Normative References

[I-D.henderson-hip-applications]
Henderson, T. and P. Nikander, "Using HIP with Legacy Applications", draft-henderson-hip-applications-03 (work in progress), May 2006.

[I-D.ietf-hip-arch]
Moskowitz, R. and P. Nikander, "Host Identity Protocol Architecture", draft-ietf-hip-arch-03 (work in progress), August 2005.

[I-D.ietf-shim6-failure-detection]
Arkko, J. and I. Beijnum, "Failure Detection and Locator Pair Exploration Protocol for IPv6 Multihoming", draft-ietf-shim6-failure-detection-03 (work in progress), December 2005.

[I-D.ietf-shim6-proto]
Bagnulo, M. and E. Nordmark, "Level 3 multihoming shim

protocol", draft-ietf-shim6-proto-03 (work in progress), December 2005.

[POSIX] "IEEE Std. 1003.1-2001 Standard for Information Technology -- Portable Operating System Interface (POSIX). Open group Technical Standard: Base Specifications, Issue 6, http://www.opengroup.org/austin", December 2001.

[RFC3493] Gilligan, R., Thomson, S., Bound, J., McCann, J., and W. Stevens, "Basic Socket Interface Extensions for IPv6", RFC 3493, February 2003.

[RFC3542] Stevens, W., Thomas, M., Nordmark, E., and T. Jinmei, "Advanced Sockets Application Program Interface (API) for IPv6", RFC 3542, May 2003.

15.2. Informative References

[I-D.ietf-shim6-app-refer] Nordmark, E., "Shim6 Application Referral Issues", draft-ietf-shim6-app-refer-00 (work in progress), July 2005.

[I-D.ietf-shim6-hba] Bagnulo, M., "Hash Based Addresses (HBA)", draft-ietf-shim6-hba-01 (work in progress), October 2005.

[I-D.nordmark-shim6-esd] Nordmark, E., "Extended Shim6 Design for ID/loc split and Traffic Engineering", draft-nordmark-shim6-esd-00 (work in progress), February 2006.

[RFC2765] Nordmark, E., "Stateless IP/ICMP Translation Algorithm (SIIT)", RFC 2765, February 2000.

[RFC3972] Aura, T., "Cryptographically Generated Addresses (CGA)", RFC 3972, March 2005.

Authors' Addresses

Miika Komu
Helsinki Institue for Information Technology
Tammasaarenkatu 3
Helsinki
Finland

Phone: +358503841531
Fax: +35896949768
Email: miika@iki.fi
URI: http://www.hiit.fi/

Marcelo Bagnulo
Universidad Carlos III de Madrid
Av. Universidad 30
Leganes 28911
SPAIN

Phone: +34 91 6248837
Email: marcelo@it.uc3m.es
URI: http://it.uc3m.es/marcelo

Kristian Slavov
Ericsson Research Nomadiclab
Hirsalantie 11
Jorvas FI-02420
Finland

Phone: +358 9 299 3286
Email: kristian.slavov@ericsson.com

Shinta Sugimoto (editor)
Nippon Ericsson K.K.
Koraku Mori Building
1-4-14, Koraku, Bunkyo-ku
Tokyo 112-0004
Japan

Phone: +81 3 3830 2241
Email: shinta.sugimoto@ericsson.com

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.