SHIM6 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-ietf-shim6-multihome-shim-api-01

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 access to information about failure detection and path exploration.

This document is based on an assumption that a multhomed host is equipped with a conceptual sublayer (here after "shim") inside the IP layer that maintains mappings between identifiers and locators. Examples of the shim are SHIM6 and HIP.

Table of Contents

1. Introduction

HIP and SHIM6 have a commonality in their protocol design; separation of identifier and locator (hereafter identifier/locator separation). Both protocols aim to solve problems that are specific to multihoming environment in a host centric approach. In these protocols, a sub-layer within the IP layer maintains mappings of identifiers and locators.

The shim layer is useful in a sense that the IP layer can maintain the mapping of an identifier to corresponding locators. Under a multihomed environment, typically, a host has more than one IP address at a time. During a given transaction, a host may be required to switch the IP address used for the communication to another IP address to preserve the communication. A care is needed to not disrupt the upper layer protocols by the address update. The shim layer can make this locator update transparent to the upper layer protocols.

In a system which is based on identifier/locator separation, upper layer protocols are expected to deal with identifiers for establishing and handling the communications. If an application wants to have a multihoming support by the shim layer, the IP addresses specified as source and destination addresses must be identifiers. However, this does not necessarily mean that applications are prohibited to choose specific locators in its communication. It may be useful for applications, in some situation, to specify a preferred locator for the flow.

This document recommends that the identifier/locator adaptation is done only once inside the network stack of a host. That is, if multiple shim sublayers exist at the IP layer, any one of them should be applied exclusively for a given flow.

As this document specifies socket API, it is written so that the contents are in line with Posix standard [POSIX] as much as possible. The API specified in this document 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).

The target readers of this document are application programmers who develop application software which may benefit greatly from multihomed environment. In addition, this document should be of interest for the developers of a given shim protocol, as the shim layer should provide the interface to the application.

2. Terminology

This section provides terminology used in this document. Basically most of the terms used in this document are taken from the following documents:

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

In this document, the term "IP" refers to both IPv4 and IPv6, unless the protocol version is specifically mentioned. The followings are definitions of the terms that are frequently used in this document:

- Endpoint Identifier (EID) - An identifier used by the application to specify the endpoint of a given communication. Applications may handle EID in various ways such as long-lived connections, callbacks, and referrals[I-D.ietf-shim6-app-refer].
  - In the case of SHIM6, an identifier called an ULID serves as an EID. An ULID is chosen from locators available on the host.
  - In the case of HIP, an identifier which specifies communication endpoints is derived from the public key of the host, which is called a Host Identifier. For the sake of backward compatibility of the socket API, the Host Identifier is represented in a form of hash of public key.
- 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 on the wire.
  - 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 within a given context. As defined in [I-D.ietf-shim6-proto], the preferred locator of a host 'A' is denoted as Lp(A).
- 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 locators at a time when the host is multihomed. The term 'shim' does not refer to a specific protocol but refers to the conceptual sublayer inside the IP layer.
- identifier/locator adaptation - An adaptation performed at the shim layer between EIDs and locators within a given context. The adaptation may end up re-writing the source and destination addresses of the IP packet. In the outbound packet processing, the EID pair is converted to the associated locator pair, while

the locator pair is converted to the EID pair in the inbound packet processing.

- Context - State information shared by a given pair of peers, which stores a binding between the EIDs and associated locators. The context is maintained at the shim layer.
- Reachability Detection - A procedure to check 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 that prevents IP packets to flow from the source locator to the destination locator. When there is an outage, it means that there is no reachability between a given locator pair. The outage can be caused by various reasons, such as shortage of network resources, congestions, and human error (faulty operation).
- 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.
- Reachability Protocol (REAP) - A protocol for detecting failure and exploring reachability in a multihomed environment. REAP is defined in [I-D.ietf-shim6-failure-detection].

3. System Overview

Figure 1 illustrates the system overview. The shim layer and REAP component exist inside the IP layer. Applications can use the socket API defined in this document to interface the shim layer and transport layer for locator management and failure detection and path exploration.

It is also possible that the shim layer interacts with transport layers, but the interactions are outside the scope of this document.

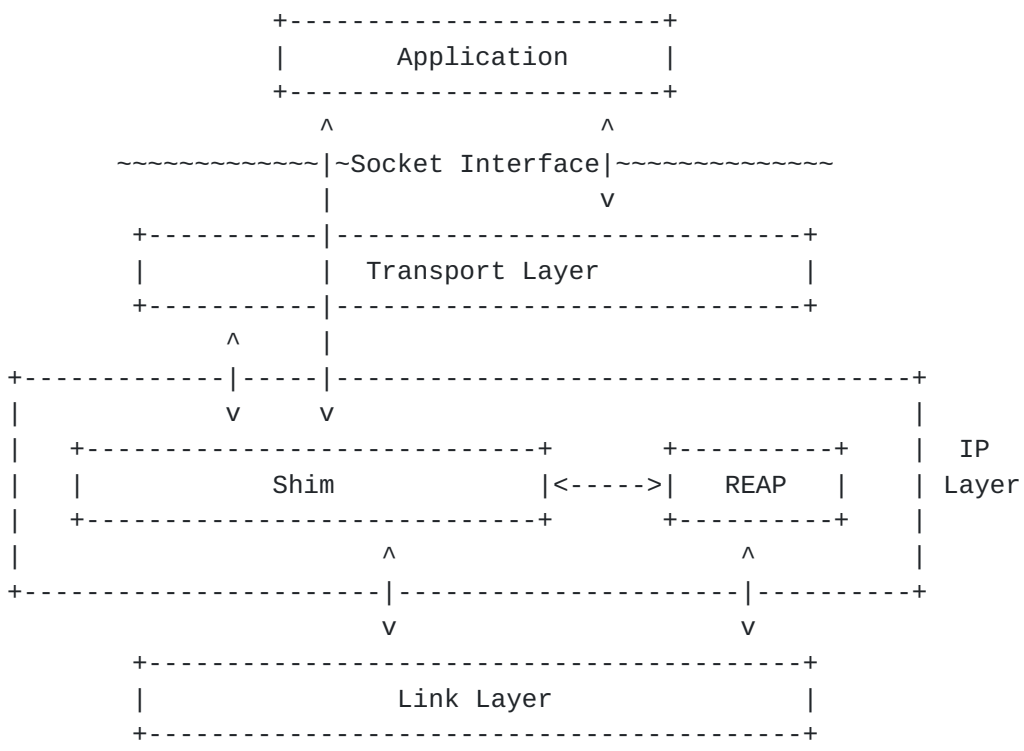

Figure 1: System overview

4. Requirements

The following is the list of requirements from the application perspective:

- 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 mentioned in [I-D.ietf-shim6-failure-detection]]:
  - 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 receive 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 of 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 an 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 whether the target locator is IPv4 or IPv6, the application should be able to use common interface to access the locator information.

5. Socket Options for Multihoming 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() or setsockopt() system call for an open 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 indicate whether 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 type of data structure specified with the socket option, which can store an argument for setsockopt() and result for getsockopt(). By default, the data structure type 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	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	o	Request for the destination locator of the received IP packet.	int
SHIM_LOC_PEER_RECV	o	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.	*2
SHIM_LOCLIST_PEER	o	o	Get or set a list of locators associated with the peer's EID.	*2
SHIM_APP_TIMEOUT	o	o	Inform the shim layer of a timeout value for detecting failure.	int
SHIM_PATHEXPLORE	o	o	Specify behavior of path exploration and failure detection.	*3

SHIM_CONTEXT_DEFERRED_SETUP	o	o	Specify if the context setup can be deferred or not.	int

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

*1: Pointer to a shim_locator which is defined in Section 7.

*2: Pointer to an array of shim_locator.

*3: Pointer to a shim_pathexplore which is defined in Section 7.

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) or any transport protocol (IPPROTO_TCP or IPPROTO_UDP).

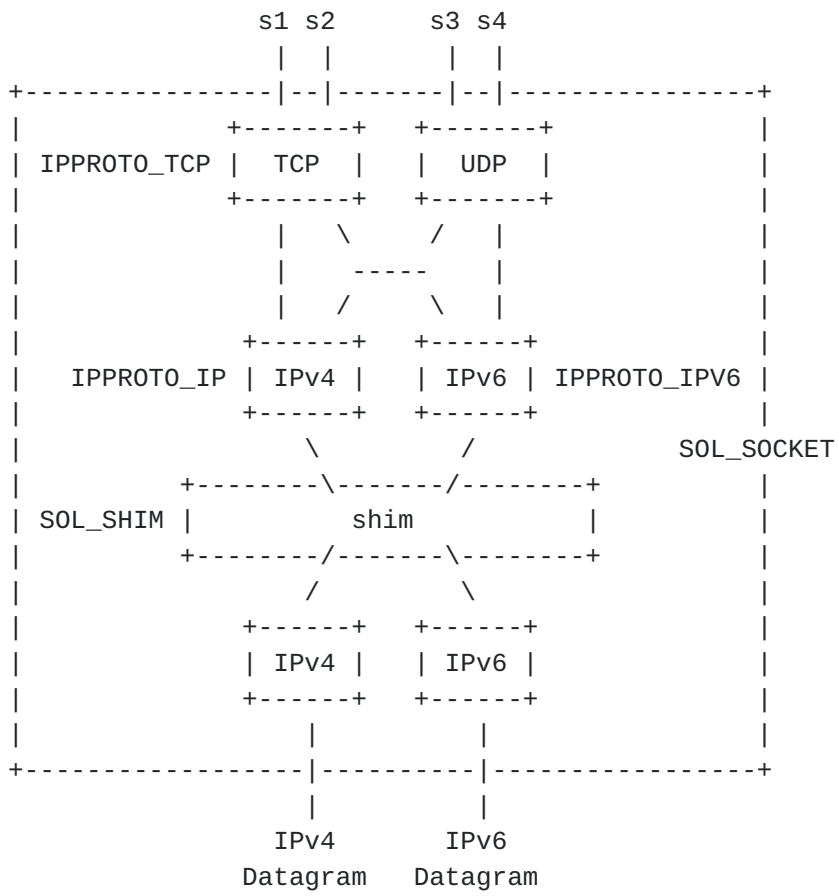

Figure 2: System model of socket API with shim layer

5.1. SHIM_ASSOCIATED

The SHIM_ASSOCIATED option can be used to check whether the socket is associated with any shim context or not.

This option is particularly meaningful in a case where the locator information of the received IP packet does not tell whether the identifier/locator adaptation is performed or not. Note that the EID pair and locator pair may be identical in some case.

This option can be specified by getsockopt(). Thus, the option is read-only and the result (0 or 1) is set in the option value (the fourth argument of getsockopt()).

Data type of the option value is integer. The option value indicates presence of shim context. A returned value 1 means that the socket is associated with a certain shim context at the shim layer, while a return value 0 indicates that there is no context associated with the socket.

For example, the option can be used by the application as follows:

```
int optval;
int optlen = sizeof(optval);

getsockopt(fd, SOL_SHIM, SHIM_ASSOCIATED, &optval, &optlen);
```

5.2. SHIM_DONTSHIM

The SHIM_DONTSHIM option can be used to request the shim layer to not apply the multihoming support for the communication established over the socket.

Data type of the option value is integer. The option value indicates whether the multihoming shim support is deprecated or not. The option value is binary (0 or 1). By default, the value is set to 0, meaning that the shim layer applies identifier/locator adaptation for the communication. In order to disable the socket option, the application should call setsockopt() with optval set as 0.

For example, the option can be disabled by the application as follows.

```
int optval;

optval = 0;

setsockopt(fd, SOL_SHIM, SHIM_DONTSHIM, &optval, sizeof(optval));
```

For example, the option value can be checked by the application as follows.

```
int optval;
int len;

len = sizeof(optval);

getsockopt(fd, SOL_SHIM, SHIM_DONTSHIM, &optval, &len);
```

5.3. SHIM_HOT_STANDBY

The SHIM_HOT_STANDBY option can be used to check if the shim layer uses hot-standby connection or not for the communication established over the socket. Hot-standby connection is another working locator pair than the current locator pair. Hence this option is effective only when there is a shim context associated with the socket.

Data type of the option value is integer.

The option value can be set by setsockopt().

The option value can be read by getsockopt().

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

For example, the option can be activated by the application as follows.

```
int optval;

optval = 1;

setsockopt(fd, SOL_SHIM, SHIM_HOT_STANDBY, &optval,
           sizeof(optval));
```

For example, the option value can be checked by the application as follows.

```
int optval;
int len;

len = sizeof(optval);

getsockopt(fd, SOL_SHIM, SHIM_HOT_STANDBY, &optval, &len);
```

5.4. SHIM_PATHEXPLORE

This option can be used to specify behavior of path exploration to be carried out. Path exploration is a procedure to find an alternative locator pair when the host finds any problem with current locator pair. A message used for finding an alternative locator pair is called a Probe message and it is sent per locator pair. Default value is defined for Initial Probe Timeout (0.5 seconds) and Initial Probe (4 times) in the REAP specification.

The option is effective only when there is a shim context associated with the socket.

Data type of the option value is a pointer to the buffer where a set of information for path exploration is stored. The data structure is defined in Section 7.

By default, the option value is set as NULL, meaning that the option is disabled.

An error ENOENT will be returned when there is no context associated

with the socket.

For example, the parameters for the path exploration can be set as follows.

```
    struct shim6_pathexplore pe;

    pe.pe_probenum = 4;        /* times */
    pe.pe_keepaliveto = 10;    /* seconds */
    pe.pe_initprobeto = 500;   /* milliseconds */
    pe.pe_reserved = 0;

    setsockopt(fd, SOL_SHIM, SHIM_PATHEXPLORE, &pe, sizeof(pe));
```

For example, the parameters for the path exploration can be read as follows.

```
    struct shim6_pathexplore pe;
    int len;

    len = sizeof(pe);

    getsockopt(fd, SOL_SHIM, SHIM_PATHEXPLORE, &pe, &len);
```

5.5. SHIM_LOC_LOCAL_PREF

The SHIM_LOC_LOCAL_PREF option can be used to read or set preferred locator on local side within a given context. Hence this option is effective only when there is a shim context associated with the socket.

Data type of the option value is a pointer to the a specific data structure which stores the locator information. The data structure is defined in Section 7.

By default, the option value is set as NULL, meaning that the option is disabled.

The preferred locator can be set by setsockopt(). Verification of the locator shall be done by the shim layer before updating the preferred locator.

The preferred locator can be read by getsockopt().

An error ENOENT 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.

For example, a preferred locator can be set as follows. It should be noted that some members of the shim_locator (lc_ifidx and lc_flags) are ignored in the write operation.

```
struct shim_locator lc;
struct in6_addr ip6;

/* ...set the locator (ip6)... */

bzero(&lc, sizeof(shim_locator));
lc.lc_family = AF_INET6;  /* IPv6 */
lc.lc_ifidx = 0;
lc.lc_flags = 0;
lc.lc_preference = 255;
memcpy(lc.lc_addr, &ip6, sizeof(in6_addr));

setsockopt(fd, SOL_SHIM, SHIM_LOC_LOCAL_PREF, &lc,
           sizeof(optval));
```

For example, the preferred locator of the context can be read by application as follows.

```
struct shim_locator lc;
int len;

len = sizeof(lc);

getsockopt(fd, SOL_SHIM, SHIM_LOC_LOCAL_PREF, &lc, &len);
```

5.6. SHIM_LOC_PEER_PREF

The SHIM_LOC_PEER_PREF option can be used to read or set preferred locator on peer side within a given context. Hence this option is effective only when there is a shim context associated with the socket.

Data type of the option value is a pointer to the a specific data structure which stores the locator information. The data structure is defined in Section 7.

By default, the option value is set as NULL, meaning that the option is disabled.

The preferred locator can be set by setsockopt(). Necessary verification of the locator shall be done by the shim layer before updating the preferred locator.

The preferred locator can be read by getsockopt().

An error ENOENT 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.

For example, a preferred locator can be set as follows. It should be noted that some members of the shim_locator (lc_ifidx and lc_flags) are ignored in the write operation.

The usage of the option is same as that of SHIM_LOC_LOCAL_PREF.

5.7. SHIM_LOC_LOCAL_RECV

The SHIM_LOC_LOCAL_RECV option can be used to request the shim layer to store the destination locator of the received IP packet in an ancillary data object which can be accessed by recvmsg(). Hence this option is effective only when there is a shim context associated with the socket.

Data type of the option value is integer. The option value should be binary (0 or 1). By default, the option value is set to 0, meaning that the option is disabled.

The option value can be set by setsockopt().

The option value can be read by getsockopt().

See section Section 6 for the procedure to access locator information stored in the ancillary data objects.

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

For example, the option can be activated by the application as follows:

```
int optval;

optval = 1;

setsockopt(fd, SOL_SHIM, SHIM_LOC_LOCAL_RECV, &optval,
           sizeof(optval));
```

For example, the option value can be checked by the application as follows:

```
int optval;
int len;

len = sizeof(optval);

getsockopt(fd, SOL_SHIM, SHIM_LOC_LOCAL_RECV, &optval, &len);
```

5.8. SHIM_LOC_PEER_RECV

The SHIM_LOC_PEER_RECV option can be used to request the shim layer to store the source locator of the received IP packet in an ancillary data object which can be accessed by recvmsg(). Hence this option is effective only when there is a shim context associated with the socket.

Data type of the option value is integer. The option value should be binary (0 or 1). By default, the option value is set to 0, meaning that the option is disabled.

The option value can be set by setsockopt().

The option value can be read by getsockopt().

See section Section 6 for the procedure to access locator information stored in the ancillary data objects.

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

The usage of the option is same as that of SHIM_LOC_LOCAL_RECV option.

5.9. SHIM_LOCLIST_LOCAL

The SHIM_LOCLIST_LOCAL option can be used to read or set the locator list associated with the local EID of the shim context associated with the socket. Hence this option is effective only when there is a shim context associated with the socket.

Data type of option value is pointer to the buffer where a locator list is stored. See section Section 7 for the data structure for storing the locator information. By default, the option value is set as NULL, meaning that the option is disabled.

The locator list can be read by getsockopt(). Note that the size of the buffer pointed by optval argument should be large enough to store an array of locator information. The number of the locator information is not known beforehand.

The locator list can be set by setsockopt(). The buffer pointed by optval argument should contain an array of locator list.

An error ENOENT 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.

Example is TBD.

5.10. SHIM_LOCLIST_PEER

The SHIM_LOCLIST_LOCAL option can be used to read or set the locator list associated with the peer EID of the shim context associated with the socket. Hence this option is effective only when there is a shim context associated with the socket.

Data type of option value is pointer to the buffer where a locator list is stored. See section Section 7 for the data structure for storing the locator information. By default, the option value is set as NULL, meaning that the option is disabled.

The locator list can be read by getsockopt(). Note that the size of the buffer pointed by optval argument should be large enough to store an array of locator information. The number of the locator information is not known beforehand.

The locator list can be set by setsockopt(). The buffer pointed by optval argument should contain an array of locator list.

An error ENOENT 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.

The usage of the option is same as that of SHIM_LOCLIST_LOCAL.

5.11. SHIM_APP_TIMEOUT

The SHIM_APP_TIMEOUT option indicates timeout value for application to detect failure. Hence this option is effective only when there is a shim context associated with the socket.

Data type of the option value is integer. The value indicates the period of timeout in seconds to send a REAP Keepalive message since the last outbound traffic. By default, the option value is set as 0,

meaning that the option is disabled. When the option is disabled, the REAP mechanism follows its default value of Send Timeout value as specified in [I-D.ietf-shim6-failure-detection]

If the timeout value specified is longer than the Send Timeout configured in the REAP component, the REAP Keepalive message should be suppressed.

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

For example, a specific timeout value can be configured by the application as follows:

```
int optval;

optval = 15; /* 15 seconds */

setsockopt(fd, SOL_SHIM, SHIM_APP_TIMEOUT, &optval,
           sizeof(optval));
```

For example, the option value namely the period of timeout can be checked by the application as follows:

```
int optval;
int len;

len = sizeof(optval);

getsockopt(fd, SOL_SHIM, SHIM_APP_TIMEOUT, &optval, &len);
```

5.12. SHIM_DEFERRED_CONTEXT_SETUP

The SHIM_DEFERRED_CONTEXT_SETUP option indicates how initiation of context setup is made in terms of timing (before or after) the initial communication flow. Deferred context means that the establishment of context does not put additional delay for an initial transaction.

Data type for the option value is integer. The option value should binary (0 or 1). By default, the value is set as 1, meaning that the context setup is deferred. In order to disable the option, the application should call setsockopt() with option value set as 0.

However, 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 should be established prior to the

communication.

For example, the option can be disabled by the application as follows:

```
int optval;

optval = 0;

setsockopt(fd, SOL_SHIM, SHIM_DEFERRED_CONTEXT_SETUP,
           &optval, sizeof(optval));
```

For example, the option value can be checked by the application as follows:

```
int optval;
int len;

len = sizeof(optval);

getsockopt(fd, SOL_SHIM, SHIM_DEFERRED_CONTEXT_SETUP,
           &optval, &len);
```

5.13. Error Handling

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

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

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 for security reasons prior to sending an IP packet to the peer's new locator; 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].

6. Ancillary Data for Multihoming Shim

In this section, definition and usage of the ancillary data which is specific to multihoming shim are provided.

As defined in Posix standard, sendmsg() and recvmsg() take msghdr structure as its argument and they can additionally handle control information along with data. Figure 18 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 can be stored in addition to other ancillary data objects.

```
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 18: msghdr structure

The buffer pointed from the msg_control member of the msghdr structure may contain a locator information which is 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 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_FEEDBACK_POSITIVE	o		TBD
SHIM_FEEDBACK_NEGATICE	o		TBD

Table 2: Shim specific ancillary data

*1: cmsg_data[] should include padding (if necessary) and a single

sockaddr_in{}/sockaddr_in6{}.

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.

6.1. 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.

6.2. 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 the interface identifier of the physical interface over which the application expects the packet to be transmitted.

Note that the effect is limited to the datagram transmitted by the sendmsg().

If the specified locator pair seems 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.

6.3. Notification from Application to Multihoming Shim

Application may provide feedback to the shim layer in accordance with its communication status. The notification can be made by specifying shim specific ancillary data in sendmsg() call. Note that this notification is dynamic rather than static.

6.3.1. SHIM_FEEDBACK_POSITIVE

The application can simply inform the shim layer that its communication is going well.

Data type is TBD.

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

6.3.2. SHIM_FEEDBACK_NEGATIVE

The application can inform the shim layer that its communication is not going well.

Data type is TBD.

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

7. Data Structures

In this section, data structures specifically defined for the multihoming shim layer are introduced. Those data structures are

7.1. Placeholder for Locator Information

As defined in Section 5, the SHIM_LOC_LOCAL_PREF, SHIM_LOC_PEER_PREF, SHIM_LOCLIST_LOCAL, and SHIM_LOCLIST_PEER socket options need to handle one or more locator information. Locator information includes not only the locator itself but also additional information about the locator which is useful for locator management. A new data structure is defined to serve as a placeholder for the locator information.

Figure 19 illustrates the data structure called shim_locator which stores a locator information.

```
struct shim_locator {
        uint8_t   lc_family;     /* address family */
        uint8_t   lc_ifidx;      /* interface index */
        uint8_t   lc_flags;      /* flags */
        uint8_t   lc_preference; /* preference value */
        uint8_t   lc_addr[16];   /* locator */
};
```

Figure 19: shim locator structure

lc_family
Address family of the locator (e.g. AF_INET, AF_INET6). It is required that the parameter contains non-zero value indicating the exact address family of the locator.

lc_ifidx
Interface index of the network interface to which the locator is assigned. This field should be valid only in read (getsockopt()) operation.

lc_flags
Each bit of the flags represents a specific characteristics of the locator. HBA is defined as 0x01. CGA is defined as 0x02. The other bits are TBD.

lc_preference
Indicates preference of the locator. The preference is represented by integer.

lc_addr
Contains the locator. For the cases where a locator whose size is smaller than 16 bytes, encoding rule should be provided for each locator of a given address family. For instance, in case of AF_INET (IPv4), the last 4 bytes of lc_addr should contain the IPv4 address.

7.2. Path Exploration Parameter

As defined in Section 5, SHIM_PATHEXPLORE allows application to set or read the parameters for path exploration and failure detection. A new data structure called shim_pathexplore is defined to store the necessary parameters. Figure 20 illustrates the data structure. The data structure can be used by getsockopt() or setsockopt() as an argument.

```
struct shim_pathexplore {
        uint8_t   pe_probenum;      /* # of initial probe */
        uint8_t   pe_keepaliveto;   /* Keepalive Timeout */
        uint16_t  pe_initprobeto;   /* Initial Probe Timeout */
        uint32_t  pe_reserved;      /* reserved */
};
```

Figure 20: path explore structure

pe_probenum
Indicates the number of initial probe messages to be sent. Default value of this parameter should follow what is specified in [I-D.ietf-shim6-failure-detection].

pe_keepaliveto
: Indicates timeout value for detecting a failure when the host does not receive any packets for a certain period of time while there is outbound traffic. When the timer expires, path exploration procedure will be carried out by sending a REAP Probe message. Default value of this parameter should follow what is specified in [I-D.ietf-shim6-failure-detection].

pe_initprobeto
: Indicates retransmission timer of REAP Probe message in milliseconds. Note that this timer is applied before exponential back-off is started. A REAP Probe message for the same locator pair may be retransmitted. Default value of this parameter should follow what is specified in [I-D.ietf-shim6-failure-detection].

pe_reserved
: A reserved field for future extension. By default, the field should be initialized with zero.

8. Implications for Existing Socket API Extensions

Some of the socket options defined in this document have some overlapping with existing socket API and care should be made for the usage not to confuse the features.

The socket options for requesting specific locators to be used for a given transaction (SHIM_LOC_LOCAL_PREF and SHIM_LOC_PEER_PREF) are semantically similar to the existing socket API (IPV6_PKTINFO). The socket options for obtaining the locator information from the received IP packet (SHIM_LOC_LOCAL_RECV and SHIM_LOC_PEER_RECV) are semantically similar to the existing socket API (IP_RECVDSTADDR and IPV6_PKTINFO).

In IPv4, application can obtain the destination IP address of the received IP packet (IP_RECVDSTADDR). If the shim layer performs identifier/locator adaptation for the received packet, the destination EID should be stored in the ancillary data (IP_RECVDSTADDR).

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. 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.

Baseline is that the above existing socket API (IP_RECVDSTADDR and IPV6_PKTINFO) is assumed to work above the multihoming shim layer.

In other words, the IP addresses those socket options deal with are EIDs rather than locators.

9. Discussion

In this section, open discussion issues are noted.

9.1. 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 Section 6, 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.

9.2. Issues with Shim Unaware Application

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

9.2.1. 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].

9.2.2. 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.

9.3. 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.

9.4. Issues of Header Conversion among Different Address Family

The shim layer performs identifier/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 ?

9.5. 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.

10. IANA Considerations

This document contains no IANA consideration.

11. 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.

12. 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 applications that may greatly benefit from multihomed environment. From the architectural perspective, the socket API enhances software development environment in a sense that it allows separate treatment of identifier and locator at the IP layer. The API is designed with a care not to break the semantics of existing socket API and minimize the impact to the legacy applications.

Multihoming shim socket options defined in this document can be used by getsockopt() and/or setcokopt() system calls, which allow applications to have control of locator management. Additionally, applications can specify locator information for outgoing packet and get locator information from incoming packet by using ancillary data

objects that are specific to the multihoming shim layer.

13. Acknowledgments

Authors would like to thank Jari Arkko who participated in the discussion that lead to the first version of this document, and Tatuya Jinmei who thoroughly reviewed the early version of this draft and provided detailed comments on socket API related issues.

14. References

14.1. 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-06 (work in progress), September 2006.

[I-D.ietf-shim6-proto]
Bagnulo, M. and E. Nordmark, "Level 3 multihoming shim protocol", draft-ietf-shim6-proto-05 (work in progress), May 2006.

[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.

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

14.2. 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-02 (work in progress), October 2006.

[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.