       Sockets Application Program Interface (API) for Multihoming Shim

Abstract

   This document specifies sockets API extensions for the multihoming
   shim layer. The API aims to enable interactions between 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 multihomed host is
   equipped with a conceptual sub-layer (hereafter called "shim sub-
   layer") inside the IP layer that maintains mappings between
   identifiers and locators. Examples of the shim are Shim6 and the
   Host Identity Protocol (HIP).

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

   This document defines sockets API extensions by which upper-layer
   protocols may be informed about and control the way in which a
   multihoming shim sub-layer in the IP layer manages the dynamic choice
   of locators. Initially, the multihoming shim sub-layer refers to
   Shim6 and/or HIP, but it is defined generically.

   The role of the multihoming shim sub-layer (hereafter called "shim
   sub-layer" in this document) is to avoid impacts to upper-layer
   protocols that may be caused when the endhost changes its attachment
   point to the Internet -- for instance, in the case of a rehoming
   event under the multihomed environment. There is, however, a need
   for an API in the cases where 1) the upper-layer protocol is
   particularly sensitive to impacts, or 2) the upper-layer protocol
   wants to benefit from better knowledge of what is going on
   underneath.

There are various kinds of technologies that aim to solve the same
issue (the multihoming issue). Note that there will be conflict when
more than one shim sub-layer is active at the same time. The
assumption made in this document is that there is only a single shim
sub-layer (HIP or Shim6) activated on the system.

The target readers of this document are application programmers who
develop application software that may benefit greatly from multihomed
environments. In addition, this document aims to provide necessary
information for developers of shim protocols to implement APIs for
enabling advanced locator management.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
"SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
document are to be interpreted as described in [RFC2119].

3. Terminology and Background

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

o Shim6 Protocol Specification [RFC5533]

o HIP Architecture [RFC4423]

o Reachability Protocol (REAP) [RFC5534]

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

o Endpoint Identifier (EID) -- The identifier used by the
   application to specify the endpoint of a given communication.
   Applications may handle EIDs in various ways, such as long-lived
   connections, callbacks, and referrals [SHIM6-APP-REFER].

   * In the case of Shim6, an identifier called a ULID (Upper-Layer
      Identifier) serves as an EID. A ULID is chosen from locators
      available on the host.

   * In the case of HIP, an identifier called a Host Identifier
      serves as an EID. A Host Identifier is derived from the public
      key of a given host. For the sake of backward compatibility
      with the sockets API, the Host Identifier is represented in the
      form of a hash of a public key.

        * Note that the EID appears in the standard sockets API as an
           address, and does not appear in the extensions defined in this
           document, which only concern locators.

   o Locator - The IP address actually used to deliver IP packets.
      Locators are present in the source and destination fields of the
      IP header of a packet on the wire. A locator as discussed in this
      document could be either an IPv4 address or an IPv6 address. Note
      that HIP can handle both IPv4 and IPv6 locators, whereas Shim6 can
      handle only IPv6 locators. For the HIP case, a locator can be a
      private IPv4 address when the host is behind a NAT. Section 8.1.1
      gives a detailed description about the handling of a locator
      behind a NAT.

        * 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 [RFC5533], the list of
           locators associated with an EID 'A' is denoted as Ls(A).

        * Preferred locator - The (source/destination) locator currently
           used to send packets within a given context.

        * Unknown locator - Any locator that does not appear in the
           locator list of the shim context associated with the socket.
           When there is no shim context associated with the socket, any
           source and/or destination locator requested by the application
           is considered to be an unknown locator.

        * Valid locator - A valid locator means that the locator is
           considered to be valid in the security sense. More
           specifically, the validity indicates whether the locator is
           part of a Hash-Based Address (HBA) set [RFC5535].

        * Verified locator - A verified locator means that the locator is
           considered to be reachable according to the result of a REAP
           return routability check. Note that the verification applies
           only to the peer's locator.

   o Shim - The conceptual sub-layer inside the IP layer. This sub-
      layer maintains mappings between EIDs and locators. An EID can be
      associated with more than one locator at a time when the host is
      multihomed. The term "shim" does not refer to a specific protocol
      but refers to the conceptual sub-layer inside the IP layer.

   o Identifier/locator adaptation - The adaptation performed at the
      shim sub-layer. This adaptation may end up re-writing the source
      and/or destination addresses of an IP packet. In the outbound
      packet processing, the EID pair is converted to the associated
      locator pair. In the inbound packet processing, the locator pair
      is converted to the EID pair.

   o Context - The state information shared by a given pair of peers.
      Context stores a binding between the EID and associated locators.
      Contexts are maintained by the shim sub-layer. Deferred context
      setup is a scenario where a context is established after the
      communication starts. Deferred context setup is possible if the
      ULID is routable, such as in the case of Shim6.

   o Reachability detection - The procedure to check reachability
      between a given locator pair.

   o Path - The sequence of routers that an IP packet goes through to
      reach the destination.

   o Path exploration - The procedure to explore available paths for a
      given set of locator pairs.

   o Outage - The incident that prevents IP packets flowing 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 may be caused by various reasons, such
      as a shortage of network resources, congestion, and human error
      (faulty operation).

   o Working address pair - Considered to be "working" if the packet
      can safely travel from the source to the destination, where the
      packet contains the first address from the pair as the source
      address and the second address from the pair as the destination
      address. If reachability is confirmed in both directions, the
      address pair is considered to be working bi-directionally.

   o Reachability Protocol (REAP) - The protocol for detecting failure
      and exploring reachability in a multihomed environment. REAP is
      defined in [RFC5534].

In this document, syntax and semantics of the API are given in the
same way as in the Portable Operating System Interface (POSIX)
standard [POSIX]. The API specifies how to use ancillary data (aka
cmsg) to access the locator information with recvmsg() and/or
sendmsg() I/O calls. The API is described in C language, and data
types are defined in the POSIX format; intN_t means a signed integer
of exactly N bits (e.g., int16_t), and uintN_t means an unsigned
integer of exactly N bits (e.g., uint32_t).

The distinction between "connected" sockets and "unconnected" sockets
is important when discussing the applicability of the sockets API
defined in this document. A connected socket is bound to a given
peer, whereas an unconnected socket is not bound to any specific
peers. A TCP socket becomes a connected socket when the TCP
connection establishment is completed. UDP sockets are unconnected,
unless the application uses the connect() system call.

4. System Overview

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

It is also possible that the shim sub-layer interacts with the
transport layer; however, such an interaction is outside the scope of
this document.

```
                +------------------------+
                |      Application        |
                +------------------------+
                     ^              ^
        ~~~~~~~~~~~~|~Socket Interface|~~~~~~~~~~~~~~
                     |              v
        +-----------|------------------------------+
        |           |  Transport Layer             |
        +-----------|------------------------------+
            ^       |
    +-----------|-----|------------------------------------+
    |           v     v                                    |
    |   +----------------------------+    +----------+   |  IP
    |   |          Shim              |<----->|  REAP    |   |  Layer
    |   +----------------------------+    +----------+   |
    |                   ^                       ^         |
    +-----------------------|----------------------|----------+
                    v                       v
        +------------------------------------------+
        |              Link Layer                  |
        +------------------------------------------+
```
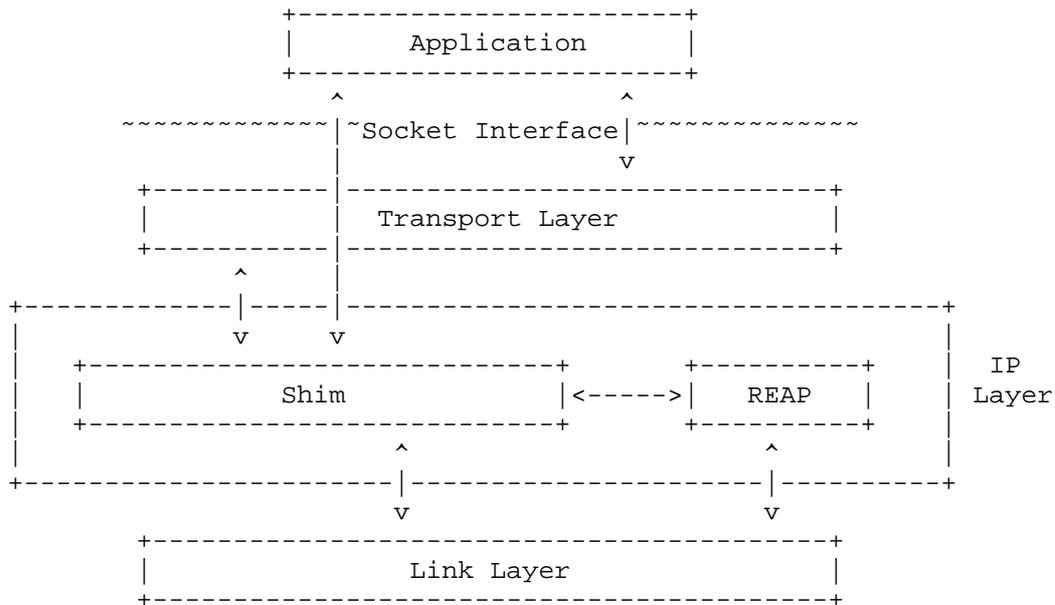
               Figure 1: System Overview

5. Requirements

   The following is a list of requirements from applications:

   o Turn on/off shim. An application should be able to request to
      turn on or turn off the multihoming support by the shim layer:

      * Apply shim. The application should be able to explicitly
         request that the shim sub-layer apply multihoming support.

      * Don't apply shim. The application should be able to request
         that the shim sub-layer not apply the multihoming support but
         apply normal IP processing at the IP layer.

      * Note that this function is also required by other types of
         multihoming mechanisms, such as the Stream Control Transmission
         Protocol (SCTP) and multipath TCP, to avoid potential conflict
         with the shim sub-layer.

o Locator management.

   * It should be possible to set a preferred source and/or
      destination locator within a given context.

   * It should be possible to get a preferred source and/or
      destination locator within a given context.

   * It should be possible to set a list of source and/or
      destination locators within a given context: Ls(local) and
      Ls(remote).

   * It should be possible to get a list of source and/or
      destination locators within a given context: Ls(local) and
      Ls(remote).

o Notification from applications and upper-layer protocols to the
   shim sub-layer about the status of the communication. The
   notification occurs in an event-based manner. Applications and/or
   upper-layer protocols may provide positive feedback or negative
   feedback to the shim sub-layer. Note that these types of feedback
   are mentioned in [RFC5534]:

   * Applications and/or upper-layer protocols (e.g., TCP) may
      provide positive feedback to the shim sub-layer informing that
      the communication is going well.

   * Applications and/or upper-layer protocols (e.g., TCP) may
      provide negative feedback to the shim sub-layer informing that
      the communication status is not satisfactory. TCP may detect a
      problem when it does not receive any expected ACK message from
      the peer. The REAP module may be triggered by the negative
      feedback and invoke the path exploration procedure.

o Feedback from applications to the shim sub-layer. Applications
   should be able to inform the shim sub-layer of the timeout values
   for detecting failures, sending keepalives, and starting the
   exploration procedure. In particular, applications should be able
   to suppress keepalives.

o Hot-standby. Applications may request the shim sub-layer for a
   hot-standby capability. This means that alternative paths are
   known to be working in advance of a failure detection. In such a
   case, it is possible for the shim sub-layer to immediately replace
   the current locator pair with an alternative locator pair.

   o Eagerness for locator exploration. An application should be able
      to inform the shim sub-layer of how aggressively it wants the REAP
      mechanism to perform a path exploration (e.g., by specifying the
      number of concurrent attempts of discovery of working locator
      pairs) when an outage occurs on the path between the locator pair
      in use.

   o Providing locator information to applications. An application
      should be able to obtain information about the locator pair that
      was actually used to send or receive packets.

      * For inbound traffic, the application may be interested in the
         locator pair that was actually used to receive the packet.

      * For outbound traffic, the application may be interested in the
         locator pair that was actually used to transmit the packet.

      In this way, applications may have additional control of the
      locator management. For example, an application becomes capable
      of verifying if its preference for a locator is actually applied
      to the flow or not.

   o Applications should be able to know if the shim sub-layer supports
      deferred context setup or not.

   o An application should be able to know if the communication is now
      being served by the shim sub-layer or not.

   o An application should be able to use a common interface to access
      an IPv4 locator and an IPv6 locator.

6. Socket Options for Multihoming Shim Sub-Layer

   In this section, socket options that are specific to the shim
   sub-layer are defined.

   Table 1 shows a list of the socket options that are specific to the
   shim sub-layer. All of these socket options are defined at the level
   SOL_SHIM. When an application uses one of the socket options by
   getsockopt() or setsockopt(), the second argument MUST be set to
   SOL_SHIM.

The first column of Table 1 gives the name of the option. The second
column indicates whether the value for the socket option can be read
by getsockopt(), and the third column indicates whether the value for
the socket option can be written by setsockopt(). The fourth column
provides a brief description of the socket option. The fifth column
shows the type of data structure specified along with the socket
option. By default, the data structure type is an integer.

optname	get	set	description	dtype
SHIM_ASSOCIATED	o		Get the parameter that indicates whether the socket is associated (1) with any shim context or not (0).	int
SHIM_DONTSHIM	o	o	Get or set the parameter that indicates whether or not to employ multihoming support by the shim sub-layer.	int
SHIM_HOT_STANDBY	o	o	Get or set the parameter to request the shim sub-layer to prepare a hot-standby connection.	int
SHIM_LOC_LOCAL_PREF	o	o	Set the preference value for a source locator for outbound traffic. Get the preferred locator for the source locator for outbound traffic.	Note 1

SHIM_LOC_PEER_PREF	o	o	Set the preference value for a destination locator for outbound traffic. Get the preferred locator for the destination locator for outbound traffic.	Note 1
SHIM_LOC_LOCAL_RECV	o	o	Request the shim sub-layer to store the destination locator of the received IP packet in an ancillary data object.	int
SHIM_LOC_PEER_RECV	o	o	Request the shim sub-layer to store the source locator of the received IP packet in an ancillary data object.	int
SHIM_LOC_LOCAL_SEND	o	o	Get or set the source locator of outgoing IP packets.	Note 1
SHIM_LOC_PEER_SEND	o	o	Get or set the destination locator of outgoing IP packets.	Note 1
SHIM_LOCLIST_LOCAL	o	o	Get or set the list of locators associated with the local EID.	Note 2

```
 |  SHIM_LOCLIST_PEER          | o   | o   | Get or set the | Note  |
 |                             |     |     | list of        | 2     |
 |                             |     |     | locators       |       |
 |                             |     |     | associated with|       |
 |                             |     |     | the peer's EID.|       |
 |  SHIM_APP_TIMEOUT           | o   | o   | Get or set the | int   |
 |                             |     |     | Send Timeout   |       |
 |                             |     |     | value of REAP. |       |
 |  SHIM_PATHEXPLORE           | o   | o   | Get or set     | Note  |
 |                             |     |     | parameters for | 3     |
 |                             |     |     | path           |       |
 |                             |     |     | exploration and|       |
 |                             |     |     | failure        |       |
 |                             |     |     | detection.     |       |
 |  SHIM_CONTEXT_DEFERRED_SETUP| o   |     | Get the        | int   |
 |                             |     |     | parameter that |       |
 |                             |     |     | indicates      |       |
 |                             |     |     | whether        |       |
 |                             |     |     | deferred       |       |
 |                             |     |     | context setup  |       |
 |                             |     |     | is supported or|       |
 |                             |     |     | not.           |       |
 +-----------------------------+-----+-----+----------------+-------+
```

           Table 1: Socket Options for Multihoming Shim Sub-Layer

   Note 1: Pointer to a shim_locator as defined in Section 8.

   Note 2: Pointer to an array of shim_locator data.

   Note 3: Pointer to a shim_pathexplore as defined in Section 8.

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

   1. The existing sockets APIs continue to work at the layer above the
       shim sub-layer. That is, those legacy APIs handle IP addresses
       as identifiers.

   2. With newly defined socket options for the shim sub-layer, the
       application obtains additional control of locator management.

   3. The shim-specific socket options can be kept independent from
       address family (IPPROTO_IP or IPPROTO_IPV6) and transport
       protocol (IPPROTO_TCP or IPPROTO_UDP) settings.

```
                        s1 s2       s3 s4
                         |  |        |  |
          +--------------|--|-------|--|---------------+
          |              +------+   +------+           |
          | IPPROTO_TCP  | TCP  |   | UDP  |           |
          |              +------+   +------+           |
          |                |   \     /   |             |
          |                |    -----    |             |
          |                |   /     \   |             |
          |              +------+   +------+           |
          | IPPROTO_IP   | IPv4 |   | IPv6 | IPPROTO_IPV6 |
          |              +------+   +------+           |
          |                  \         /      SOL_SOCKET |
          |           +--------\-------/--------+       |
          | SOL_SHIM  |          shim           |       |
          |           +--------/-------\--------+       |
          |                  /           \             |
          |              +------+   +------+           |
          |              | IPv4 |   | IPv6 |           |
          |              +------+   +------+           |
          |                  |           |             |
          +------------------|-----------|-------------+
                             |           |
                           IPv4        IPv6
                         Datagram    Datagram
```
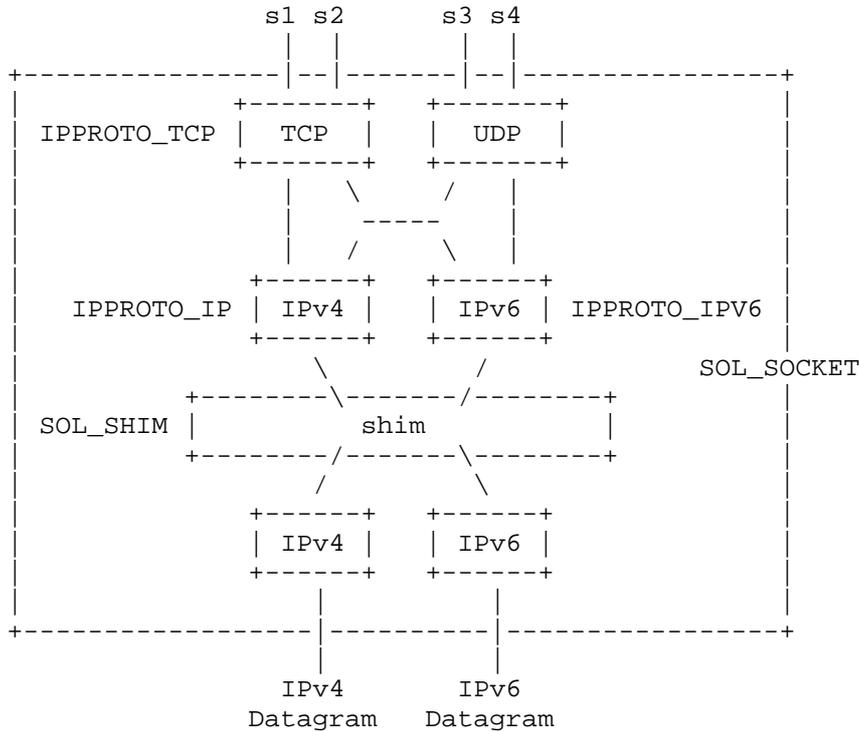
          Figure 2: System Model of Sockets API with Shim Sub-Layer

6.1. SHIM_ASSOCIATED

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

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

   Note that the socket option is read-only, and the option value can be
   read by getsockopt(). The result (0/1/2) is set in the option value
   (the fourth argument of getsockopt()).

   When the application specifies the socket option to an unconnected
   socket, error code EOPNOTSUPP is returned to the application.

The data type of the option value is an integer. The option value
indicates the presence of shim context. A return value of 1 means
that the socket is associated with a shim context at the shim
sub-layer. A return value of 0 indicates that there is no shim
context associated with the socket. A return value of 2 means that
it is not known whether or not the socket is associated with a shim
context, and this MUST be returned only when the socket is
unconnected. In other words, the returned value MUST be 0 or 1 when
the socket is connected.

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);
```

6.2. SHIM_DONTSHIM

The SHIM_DONTSHIM option is used to request that the shim layer not
provide the multihoming support for the communication established
over the socket.

The data type of the option value is an integer, and it takes 0 or 1.
An option value of 0 means that the shim sub-layer is employed if
available. An option value of 1 means that the application does not
want the shim sub-layer to provide the multihoming support for the
communication established over the socket.

The default value is set to 0, which means that the shim sub-layer
performs identifier/locator adaptation if available.

Any attempt to disable the multihoming shim support MUST be made by
the application before the socket is connected. If an application
makes such an attempt for a connected socket, error code EOPNOTSUPP
MUST be returned.

For example, an application can request that the system not apply the
multihoming support as follows:

```
    int optval;

    optval = 1;

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

For example, the application can check the option value as follows:

```
int optval;
int len;

len = sizeof(optval);

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

6.3. SHIM_HOT_STANDBY

   The SHIM_HOT_STANDBY option is used to control whether or not the
   shim sub-layer employs a hot-standby connection for the socket. A
   hot-standby connection is an alternative working locator pair to the
   current locator pair. This option is effective only when there is a
   shim context associated with the socket.

   The data type of the option value is an 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.

   When the application specifies the socket option to an unconnected
   socket, error code EOPNOTSUPP is returned to the application.

   When there is no shim context associated with the socket, error code
   ENOENT is returned to the application.

   For example, an application can request establishment of a hot-
   standby connection by using the socket option as follows:

```
int optval;

optval = 1;

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

For example, an application can get the option value by using the
socket option as follows:

```
int optval;
int len;

len = sizeof(optval);

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

6.4. SHIM_LOC_LOCAL_PREF

   The SHIM_LOC_LOCAL_PREF option is used to set the preference value
   for a source locator for outbound traffic, or to get the preference
   value of the source locator for outbound traffic that has the highest
   preference value.

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

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

   The preference of a locator is defined by a combination of priority
   and weight as per DNS SRV [RFC2782]. Note that the Shim6 base
   protocol defines the preference of a locator in the same way.

   The data type of the option value is a pointer to the shim_locator
   information data structure as defined in Section 8.1.

   When an application specifies the socket option to an unconnected
   socket, error code EOPNOTSUPP is returned to the application.

   When there is no shim context associated with the socket, error code
   ENOENT is returned to the application.

   Error code EINVALIDLOCATOR is returned when the validation of the
   specified locator fails.

   An application can set the preference value for a source locator for
   outbound traffic by setsockopt() with the socket option. Note that
   lc_ifidx and lc_flags (as defined in Section 8.1) have no effect in a
   set operation. Below is an example of such a set operation.

```
    struct shim_locator lc;
    struct in6_addr ip6;

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

    memset(&lc, 0, sizeof(shim_locator));
    lc.lc_family = AF_INET6;  /* IPv6 */
    lc.lc_ifidx = 0;
    lc.lc_flags = 0;
    lc.lc_prio = 1;
    lc.lc_weight = 10;
    memcpy(&lc.lc_addr, &ip6, sizeof(in6_addr));

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

   An application can get the source locator for outbound traffic that
   has the highest preference value by using the socket option. Below
   is an example of such a get operation.

```
    struct shim_locator lc;
    int len;

    len = sizeof(lc);

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

6.5. SHIM_LOC_PEER_PREF

   The SHIM_LOC_PEER_PREF option is used to set the preference value for
   a destination locator for outbound traffic, or to get the preference
   value of the destination locator for outbound traffic that has the
   highest preference value.

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

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

   As defined earlier, the preference of a locator is defined by a
   combination of priority and weight as per DNS SRV [RFC2782]. When
   there is more than one candidate destination locator, the shim
   sub-layer makes a selection based on the priority and weight
   specified for each locator.

   The data type of the option value is a pointer to the shim_locator
   information data structure as defined in Section 8.1.

When the application specifies the socket option to an unconnected
socket, error code EOPNOTSUPP is returned to the application.

When there is no shim context associated with the socket, error code
ENOENT is returned to the application.

Error code EINVALIDLOCATOR is returned when the validation of the
requested locator fails.

Error code EUNREACHABLELOCATOR is returned when the requested locator
is determined to be unreachable according to a reachability check.

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

6.6. SHIM_LOC_LOCAL_RECV

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

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

   An application can set the option value by setsockopt().

   An application can get the option value by getsockopt().

   See Section 7 for the procedure to access locator information stored
   in the ancillary data objects.

   When the application specifies the socket option to an unconnected
   socket, error code EOPNOTSUPP is returned to the application.

   When there is no shim context associated with the socket, error code
   ENOENT is returned to the application.

   For example, an application can request the shim sub-layer to store a
   destination locator by using the socket option as follows:

        int optval;

        optval = 1;

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

For example, an application can get the option value as follows:

```
int optval;
int len;

len = sizeof(optval);

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

6.7. SHIM_LOC_PEER_RECV

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

   The data type of the option value is an integer. The option value
   MUST 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 7 for the procedure to access locator information stored
   in the ancillary data objects.

   When the application specifies the socket option to an unconnected
   socket, error code EOPNOTSUPP is returned to the application.

   When there is no shim context associated with the socket, error code
   ENOENT is returned to the application.

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

6.8. SHIM_LOC_LOCAL_SEND

   The SHIM_LOC_LOCAL_SEND option is used to request that the shim
   sub-layer use a specific locator as the source locator for the IP
   packets to be sent from the socket. This option is effective only
   when there is a shim context associated with the socket.

   The data type of the option value is a pointer to the shim_locator
   data structure.

   An application can set the local locator by setsockopt(), providing a
   locator that is stored in a shim_locator data structure. When a
   zero-filled locator is specified, the pre-existing setting of the
   local locator is inactivated.

   An application can get the local locator by getsockopt().

   When the application specifies the socket option to an unconnected
   socket, error code EOPNOTSUPP is returned to the application.

   When there is no shim context associated with the socket, error code
   ENOENT is returned to the application.

   Error code EINVALIDLOCATOR is returned when an invalid locator is
   specified.

   For example, an application can request the shim sub-layer to use a
   specific local locator by using the socket option as follows:

```
        struct shim_locator locator;
        struct in6_addr ia6;

        /* an IPv6 address preferred for the source locator is copied
           to the parameter ia6 */

        memset(&locator, 0, sizeof(locator));

        /* fill shim_locator data structure */
        locator.lc_family = AF_INET6;
        locator.lc_ifidx = 0;
        locator.lc_flags = 0;
        locator.lc_prio = 0;
        locator.lc_weight = 0;
        memcpy(&locator.lc_addr, &ia6, sizeof(ia6));

        setsockopt(fd, SOL_SHIM, SHIM_LOC_LOCAL_SEND, &locator,
                   sizeof(locator));
```

   For example, an application can get the designated local locator by
   using the socket option as follows:

```
       struct shim_locator locator;

       memset(&locator, 0, sizeof(locator));

       getsockopt(fd, SOL_SHIM, SHIM_LOC_LOCAL_SEND, &locator,
                 sizeof(locator));

       /* check locator */
```

6.9. SHIM_LOC_PEER_SEND

   The SHIM_LOC_PEER_SEND option is used to request that the shim
   sub-layer use a specific locator for the destination locator of IP
   packets to be sent from the socket. This option is effective only
   when there is a shim context associated with the socket.

   The data type of the option value is a pointer to the shim_locator
   data structure.

   An application can set the remote locator by setsockopt(), providing
   a locator that is stored in a shim_locator data structure. When a
   zero-filled locator is specified, the pre-existing setting of the
   remote locator is inactivated.

   An application can get the specified remote locator by getsockopt().

   The difference between the SHIM_LOC_PEER_SEND option and the
   SHIM_LOC_PEER_PREF option is that the former guarantees the use of a
   requested locator when applicable, whereas the latter does not.

   When the application specifies the socket option to an unconnected
   socket, error code EOPNOTSUPP is returned to the application.

   When there is no shim context associated with the socket, error code
   ENOENT is returned to the application.

   Error code EINVALIDLOCATOR is returned when the validation of the
   requested locator fails.

   Error code EUNVERIFIEDLOCATOR is returned when reachability for the
   requested locator has not been verified yet.

   Error code EUNREACHABLELOCATOR is returned when the requested locator
   is determined to be unreachable according to a reachability check.

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

6.10. SHIM_LOCLIST_LOCAL

   The SHIM_LOCLIST_LOCAL option is used to get or set the locator list
   associated with the local EID of the shim context associated with the
   socket. This option is effective only when there is a shim context
   associated with the socket.

   The data type of the option value is a pointer to the buffer in which
   a locator list is stored. See Section 8 for the data structure for
   storing the locator information. By default, the option value is set
   to NULL, meaning that the option is disabled.

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

   The local locator list can be set by setsockopt(). The buffer
   pointed to by the optval argument MUST contain an array of locator
   structures.

   When the application specifies the socket option to an unconnected
   socket, error code EOPNOTSUPP is returned to the application.

   When there is no shim context associated with the socket, error code
   ENOENT is returned to the application.

   Error code EINVALIDLOCATOR is returned when the validation of any of
   the specified locators failed.

   Error code ETOOMANYLOCATORS is returned when the number of locators
   specified exceeds the limit (SHIM_MAX_LOCATORS), or when the size of
   the buffer provided by the application is not large enough to store
   the locator list provided by the shim sub-layer.

   For example, an application can set a list of locators to be
   associated with the local EID by using the socket option as follows.
   Note that an IPv4 locator can be handled by HIP and not by Shim6.

```
     struct shim_locator locators[SHIM_MAX_LOCATORS];
     struct sockaddr_in *sin;
     struct sockaddr_in6 *sin6;

     memset(locators, 0, sizeof(locators));

     ...

     /* obtain local IP addresses from local interfaces */

     ...

     /* first locator (an IPv6 address) */
     locators[0].lc_family = AF_INET6;
     locators[0].lc_ifidx = 0;
     locators[0].lc_flags = 0;
     locators[0].lc_prio = 1;
     locators[0].lc_weight = 0;
     memcpy(&locators[0].lc_addr, &sa6->sin6_addr,
            sizeof(sa6->sin6_addr));

     ...

     /* second locator (an IPv4 address) */
     locators[1].lc_family = AF_INET;
     locators[1].lc_ifidx = 0;
     locators[1].lc_flags = 0;
     locators[1].lc_prio = 0;
     locators[1].lc_weight = 0;
     memcpy(&locators[1].lc_addr, &sa->sin_addr,
            sizeof(sa->sin_addr));

     setsockopt(fd, SOL_SHIM, SHIM_LOCLIST_LOCAL, locators,
                sizeof(locators));
```

   For example, an application can get a list of locators that are
   associated with the local EID by using the socket option as follows:

```
     struct shim_locator locators[SHIM_MAX_LOCATORS];

     memset(locators, 0, sizeof(locators));

     getsockopt(fd, SOL_SHIM, SHIM_LOC_LOCAL_RECV, locators,
                sizeof(locators));

     /* parse locators */
     ...
```

6.11. SHIM_LOCLIST_PEER

   The SHIM_LOCLIST_PEER option is used to get or set the locator list
   associated with the peer EID of the shim context associated with the
   socket. This option is effective only when there is a shim context
   associated with the socket.

   The data type of the option value is a pointer to the buffer where a
   locator list is stored. See Section 8 for the data structure for
   storing the locator information. By default, the option value is set
   to NULL, meaning that the option is disabled.

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

   An application can set the locator list by setsockopt(). The buffer
   pointed to by the optval argument MUST contain an array of locator
   list items.

   When the application specifies the socket option to an unconnected
   socket, error code EOPNOTSUPP is returned to the application.

   When there is no shim context associated with the socket, error code
   ENOENT is returned to the application.

   Error code EINVALIDLOCATOR is returned when the validation of any of
   the specified locators failed.

   Error code EUNVERIFIEDLOCATOR is returned when reachability for the
   requested locator has not been verified yet.

   Error code EUNREACHABLELOCATOR is returned when the requested locator
   is determined to be unreachable according to a reachability check.

   Error code ETOOMANYLOCATORS is returned when the number of locators
   specified exceeds the limit (SHIM_MAX_LOCATORS), or when the size of
   the buffer provided by the application is not large enough to store
   the locator list provided by the shim sub-layer.

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

6.12. SHIM_APP_TIMEOUT

   The SHIM_APP_TIMEOUT option is used to get or set the Send Timeout
   value of REAP [RFC5534]. This option is effective only when there is
   a shim context associated with the socket.

   The data type of the option value is an 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
   to 0, meaning that the option is disabled. When the option is
   disabled, the REAP mechanism follows its default Send Timeout value
   as specified in [RFC5534].

   When the application specifies the socket option to an unconnected
   socket, error code EOPNOTSUPP is returned to the application.

   When there is no shim context associated with the socket, error code
   ENOENT is returned to the application.

   When there is no REAP instance on the system, error code EOPNOTSUPP
   is returned to the application.

   For example, an application can set the timeout value by using the
   socket option as follows:

        int optval;

        optval = 15; /* 15 seconds */

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

   For example, an application can get the timeout value by using the
   socket option as follows:

        int optval;
        int len;

        len = sizeof(optval);

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

6.13. SHIM_PATHEXPLORE

   The application MAY use this socket option to get or set parameters
   concerning path exploration. Path exploration is a procedure to find
   an alternative locator pair to the current locator pair. As the REAP
   specification defines, a peer may send Probe messages to find an
   alternative locator pair.

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

   The 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 8.

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

   When the application specifies the socket option to an unconnected
   socket, error code EOPNOTSUPP is returned to the application.

   When there is no shim context associated with the socket, error code
   ENOENT is returned to the application.

   For example, an application can set parameters for path exploration
   by using the socket option 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, an application can get parameters for path exploration
   by using the socket option as follows:

       struct shim6_pathexplore pe;
       int len;

       len = sizeof(pe);

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

6.14. SHIM_DEFERRED_CONTEXT_SETUP

   The SHIM_DEFERRED_CONTEXT_SETUP option is used to check whether or
   not deferred context setup is possible. Deferred context setup means
   that the context is established in parallel with the data
   communication. Note that Shim6 supports deferred context setup and
   HIP does not, because EIDs in HIP (i.e., Host Identifiers) are non-
   routable.

   Note that the socket option is read-only, and the option value can be
   read by getsockopt().

   The data type for the option value is an integer. The option value
   MUST be binary (0 or 1). The option value of 1 means that the shim
   sub-layer supports deferred context setup.

   When the application specifies the socket option to an unconnected
   socket, error code EOPNOTSUPP is returned to the application.

   For example, an application can check whether deferred context setup
   is possible or not as follows:

```
       int optval;
       int len;

       len = sizeof(optval);

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

6.15. Applicability

   All the socket options defined in this section except for the
   SHIM_DONTSHIM option are applicable to applications that use
   connected sockets.

   All the socket options defined in this section except for the
   SHIM_ASSOCIATED, SHIM_DONTSHIM, and SHIM_CONTEXT_DEFERRED_SETUP
   options are effective only when there is a shim context associated
   with the socket.

6.16. Error Handling

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

   The following are new error values defined for some shim-specific
   socket options indicating that the getsockopt() or setsockopt()
   finished incompletely:

   EINVALIDLOCATOR
      This indicates that the locator is not part of the HBA set
      [RFC5535] within the shim context associated with the socket.

   EUNVERIFIEDLOCATOR
      This indicates that the reachability of the locator has not been
      confirmed. This error is applicable to only the peer's locator.

   EUNREACHABLELOCATOR
      This indicates that the locator is not reachable according to the
      result of the reachability check. This error is applicable to
      only the peer's locator.

7. Ancillary Data for Multihoming Shim Sub-Layer

   This section provides definitions of ancillary data to be used for
   locator management and notification from/to the shim sub-layer to/
   from the application.

   When the application performs locator management by sendmsg() or
   recvmsg(), a member of the msghdr structure (given in Figure 3)
   called msg_control holds a pointer to the buffer in which one or more
   shim-specific ancillary data objects may be stored. An ancillary
   data object can store a single locator. It should be possible to
   process the shim-specific ancillary data object by the existing
   macros defined in the POSIX standard and [RFC3542].

```
     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

In the case of an unconnected socket, msg_name stores the socket
address of the peer. Note that the address is not a locator of the
peer but the identifier of the peer. SHIM_LOC_PEER_RECV can be used
to get the locator of the peer node.

Table 2 is a list of the shim-specific ancillary data that can be
used for locator management by recvmsg() or sendmsg(). In any case,
the value of cmsg_level MUST be set to SOL_SHIM.

```
+---------------------+----------+----------+----------------+
| cmsg_type           | sendmsg()| recvmsg()|   cmsg_data[]  |
+---------------------+----------+----------+----------------+
| SHIM_LOC_LOCAL_RECV |          |    o     |    Note 1      |
| SHIM_LOC_PEER_RECV  |          |    o     |    Note 1      |
| SHIM_LOC_LOCAL_SEND |    o     |          |    Note 1      |
| SHIM_LOC_PEER_SEND  |    o     |          |    Note 1      |
| SHIM_FEEDBACK       |    o     |          | shim_feedback{}|
+---------------------+----------+----------+----------------+
```

                 Table 2: Shim-Specific Ancillary Data

Note 1: cmsg_data[] within msg_control includes a single
sockaddr_in{} or sockaddr_in6{} and padding if necessary

7.1. Get Locator from Incoming Packet

An 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 a local and/or remote locator from
the ancillary data.

When there is no shim context associated with the socket, the shim
sub-layer MUST return zero-filled locator information to the
application.

7.2. Set Locator for Outgoing Packet

An application can specify the locators to be used for transmitting
an IP packet by sendmsg(). When the ancillary data of cmsg_type
SHIM_LOC_LOCAL_SEND and/or SHIM_LOC_PEER_SEND are specified, the
application can explicitly specify the source and/or the destination
locators to be used for the communication over the socket. If the
specified locator pair is verified, the shim sub-layer overrides the
locator(s) of the outgoing IP packet. Note that the effect is
limited to the datagram transmitted by the sendmsg().

When there is no shim context associated with the socket, error code
ENOENT is returned to the application.

Error code EINVALIDLOCATOR is returned when validation of the
specified locator fails.

Error code EUNVERIFIEDLOCATOR is returned when reachability for the
requested locator has not been verified yet. The application is
recommended to use another destination locator until the reachability
check for the requested locator is done.

Error code EUNREACHABLELOCATOR is returned when the requested locator
is determined to be unreachable according to a reachability check.
The application is recommended to use another destination locator
when receiving the error.

7.3. Notification from Application to Multihoming Shim Sub-Layer

An application MAY provide feedback to the shim sub-layer about the
communication status. Such feedback is useful for the shim sub-layer
to monitor the reachability status of the currently used locator pair
in a given shim context.

The notification can be made by sendmsg() specifying a new ancillary
data called SHIM_FEEDBACK. The ancillary data can be handled by
specifying the SHIM_FEEDBACK option in cmsg_type.

When there is no shim context associated with the socket, error code
ENOENT is returned to the application.

See Section 8.3 for details of the data structure to be used.

It is outside the scope of this document to describe how the shim
sub-layer would react when feedback is provided by an application.

7.4. Applicability

All the ancillary data for the shim sub-layer is applicable to
connected sockets.

Care is needed when the SHIM_LOC_*_RECV socket option is used for
stream-oriented sockets (e.g., TCP sockets) because there is no one-
to-one mapping between a single send or receive operation and the
data (e.g., a TCP segment) being received. In other words, there is

no guarantee that the locator(s) set in the SHIM_LOC_*_RECV ancillary
data is identical to the locator(s) that appears in the IP packets
received. The shim sub-layer SHOULD provide the latest locator
information to the application in response to the SHIM_LOC_*_RECV
socket option.

8. Data Structures

   This section gives data structures for the shim sub-layer. These
   data structures are either used as a parameter for setsockopt() or
   getsockopt() (as mentioned in Section 6), or as a parameter for
   ancillary data to be processed by sendmsg() or recvmsg() (as
   mentioned in Section 7).

8.1. Data Structure for Locator Information

   As defined in Section 6, the SHIM_LOC_*_PREF, SHIM_LOC_*_SEND, and
   SHIM_LOCLIST_* socket options need to handle one or more locator
   information points. Locator information includes not only the
   locator itself but also additional information about the locator that
   is useful for locator management. A new data structure is defined to
   serve as a placeholder for the locator information.

   Figure 4 illustrates the data structure called shim_locator, which
   stores locator information.

```
     struct shim_locator {
             uint8_t    lc_family;        /* address family */
             uint8_t    lc_proto;         /* protocol */
             uint16_t   lc_port;          /* port number */
             uint16_t   lc_prio;          /* preference value */
             uint16_t   lc_weight;        /* weight */
             uint32_t   lc_ifidx;         /* interface index */
             struct in6_addr lc_addr;     /* address */
             uint16_t   lc_flags;         /* flags */
     };
```

                    Figure 4: Shim Locator Structure

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

lc_proto
      Internet Protocol number for the protocol that is used to handle a
      locator behind a NAT. The value MUST be set to zero when there is
      no NAT involved. When the locator is behind a NAT, the value MUST
      be set to IPPROTO_UDP.

lc_port
      Port number that is used for handling a locator behind a NAT.

lc_prio
      Priority of the locator. The range is 0-65535. The lowest
      priority value means the highest priority.

lc_weight
      Weight value indicates a relative weight for locators with the
      same priority value. The range is 0-65535. A locator with higher
      weight value is prioritized over the other locators with lower
      weight values.

lc_ifidx
      Interface index of the network interface to which the locator is
      assigned. This field is applicable only to local locators, and
      has no effect in set operations.

lc_addr
      Contains the locator. In the case of IPv4, the locator MUST be
      formatted in the IPv4-mapped IPv6 address as defined in [RFC4291].
      The locator MUST be stored in network byte order.

lc_flags
      Each bit of the flags represents a specific characteristic of the
      locator. The Hash-Based Address (HBA) is defined as 0x01. The
      Cryptographically Generated Address (CGA) is defined as 0x02.
      This field has no effect in set operations.

8.1.1. Handling Locator behind NAT

   Note that the locator information MAY contain a locator behind a
   Network Address Translator (NAT). Such a situation may arise when
   the host is behind the NAT and uses a local address as a source
   locator to communicate with the peer. Note that a NAT traversal
   mechanism for HIP is defined, which allows a HIP host to tunnel
   control and data traffic over UDP [RFC5770]. Note also that the
   locator behind a NAT is not necessarily an IPv4 address and can be an
   IPv6 address. Below is an example where the application sets a UDP
   encapsulation interface as a source locator when sending IP packets.

```
        struct shim_locator locator;
        struct in6_addr ia6;

        /* copy the private IPv4 address to the ia6 as an IPv4-mapped
           IPv6 address */

        memset(&locator, 0, sizeof(locator));

        /* fill shim_locator data structure */
        locator.lc_family = AF_INET;
        locator.lc_proto = IPPROTO_UDP;
        locator.lc_port = 50500;
        locator.lc_ifidx = 0;
        locator.lc_flags = 0;
        locator.lc_prio = 0;
        locator.lc_weight = 0;

        memcpy(&locator.lc_addr, &ia6, sizeof(ia6));

        setsockopt(fd, SOL_SHIM, SHIM_LOC_LOCAL_SEND, &locator,
                   sizeof(locator));
```

                   Figure 5: Handling Locator behind NAT

8.2. Path Exploration Parameter

   As defined in Section 6, SHIM_PATHEXPLORE allows an 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 6 illustrates the data
   structure. The data structure can be passed to getsockopt() or
   setsockopt() as an argument.

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

                      Figure 6: Path Explore Structure

   pe_probenum
      Indicates the number of initial Probe messages to be sent. The
      value MUST be set as per [RFC5534].

   pe_keepaliveto
      Indicates the timeout value in seconds 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, the
      path exploration procedure will be carried out by sending a REAP
      Probe message. The value MUST be set as per [RFC5534].

   pe_keepaliveint
      Indicates the interval of REAP Keepalive messages in seconds to be
      sent by the host when there is no outbound traffic to the peer
      host. The value MUST be set as per [RFC5534].

   pe_initprobeto
      Indicates the retransmission timer of the 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. The value MUST be set as per
      [RFC5534].

   pe_reserved
      A reserved field for future extension. By default, the field MUST
      be initialized to zero.

8.3. Feedback Information

   As mentioned in Section 7.3, applications can inform the shim
   sub-layer about the status of unicast reachability of the locator
   pair currently in use. The feedback information can be handled by
   using ancillary data called SHIM_FEEDBACK. A new data structure
   named shim_feedback is illustrated in Figure 7.

```
        struct shim_feedback {
                uint8_t   fb_direction;    /* direction of traffic */
                uint8_t   fb_indicator;    /* indicator (1-3) */
                uint16_t  fb_reserved;     /* reserved */
        };
```

                 Figure 7: Feedback Information Structure

   fb_direction
      Indicates the direction of reachability between the locator pair
      in question. A value of 0 indicates outbound direction, and a
      value of 1 indicates inbound direction.

   fb_indicator
      A value indicating the degree of satisfaction of a unidirectional
      reachability for a given locator pair.

      * 0: Default value. Whenever this value is specified, the
         feedback information MUST NOT be processed by the shim
         sub-layer.

      * 1: Unable to connect. There is no unidirectional reachability
         between the locator pair in question.

      * 2: Unsatisfactory. The application is not satisfied with the
         unidirectional reachability between the locator pair in
         question.

      * 3: Satisfactory. There is satisfactory unidirectional
         reachability between the locator pair in question.

   fb_reserved
      Reserved field. MUST be ignored by the receiver.

9. System Requirements

   As addressed in Section 6, most of the socket options and ancillary
   data defined in this document are applicable to connected sockets.
   It is assumed that the kernel is capable of maintaining the
   association between a connected socket and a shim context. This
   requirement is considered to be reasonable because a pair of source
   and destination IP addresses is bound to a connected socket.

10. Relation to Existing Sockets API Extensions

   This section explains the relation between the sockets API defined in
   this document and the existing sockets API extensions.

   As mentioned in Section 6, the basic assumption is that the existing
   sockets API continues to work above the shim sub-layer. This means
   that the existing sockets API deals with identifiers, and the sockets
   API defined in this document deals with locators.

SHIM_LOC_LOCAL_SEND and SHIM_LOC_PEER_SEND socket options are
semantically similar to the IPV6_PKTINFO sockets API in the sense
that both provide a means for an application to set the source IP
address of outbound IP packets.

SHIM_LOC_LOCAL_RECV and SHIM_LOC_PEER_RECV socket options are
semantically similar to the IP_RECVDSTADDR and IPV6_PKTINFO sockets
APIs in the sense that both provide a means for an application to get
the source and/or destination IP address of inbound IP packets.

getsockname() and getpeername() enable an application to get the
"name" of the communication endpoints, which is represented by a pair
of IP addresses and port numbers assigned to the socket.
getsockname() gives the IP address and port number assigned to the
socket on the local side, and getpeername() gives the IP address and
port number of the peer side.

11. Operational Considerations

This section gives operational considerations of the sockets API
defined in this document.

11.1. Conflict Resolution

There can be a conflicting situation when different applications
specify different preferences for the same shim context. For
instance, suppose that applications A and B establish communication
with the same EID pair while both applications have different
preferences in their choice of local locator. The notion of context
forking in Shim6 can resolve the conflicting situation.

It is possible that socket options defined in Section 6 cause a
conflicting situation when the target context is shared by multiple
applications. In such a case, the socket handler should inform the
shim sub-layer that context forking is required. In Shim6, when a
context is forked, a unique identifier called the Forked Instance
Identifier (FII) is assigned to the newly forked context. The forked
context is then exclusively associated with the socket through which
a non-default preference value was specified. The forked context is
maintained by the shim sub-layer during the lifetime of the
associated socket instance. When the socket is closed, the shim
sub-layer SHOULD delete the associated context.

When the application specifies SHIM_LOC_*_SEND specifying a different
source or destination locator that does not have the highest priority
and weight specified by the SHIM_LOC_*_PREF, the shim sub-layer
SHOULD supersede the request made by SHIM_LOC_*_SEND over the
preference specified by SHIM_LOC_*_PREF.

When the peer provides preferences of the locators (e.g., a Shim6
peer sends a locator with a Locator Preferences Option) that conflict
with preferences specified by the applications either by
SHIM_LOC_PEER_SEND or SHIM_LOC_PEER_PREF, the shim sub-layer SHOULD
supersede the preferences made by the applications over the
preferences specified by the peer.

11.2. Incompatibility between IPv4 and IPv6

The shim sub-layer performs identifier/locator adaptation.
Therefore, in some cases, the whole IP header can be replaced with a
new IP header of a different address family (e.g., conversion from
IPv4 to IPv6 or vice versa). Hence, there is an issue regarding how
to make the conversion with minimum impact. Note that this issue is
common in other protocol conversion techniques [RFC2765] [RFC6145].

As studied in the previous works on protocol conversion [RFC2765],
[RFC6145] some of the features (IPv6 routing headers, hop-by-hop
extension headers, and destination headers) from IPv6 are not
convertible to IPv4. In addition, the notion of source routing is
not exactly the same in IPv4 and IPv6. This means that an error may
occur during the conversion of the identifier and locator. It is
outside the scope of this document to describe how the shim sub-layer
should behave in such erroneous cases.

12. IANA Considerations

There are no IANA considerations for the socket options (SHIM_*), the
ancillary data, and the socket level (SOL_SHIM) that are defined in
this document. All the numbers concerned are not under the control
of the IETF or IANA, but they are platform-specific.

13. Protocol Constant

This section defines a protocol constant.

SHIM_MAX_LOCATORS The maximum number of locators to be included in a
   locator list. The value is set to 32.

14. Security Considerations

This section gives security considerations of the API defined in this
document.

14.1. Treatment of Unknown Locator

   When sending IP packets, there is a possibility that an application
   will request the use of an unknown locator for the source and/or
   destination locators. Note that the treatment of an unknown locator
   can be a subject of security considerations, because the use of an
   invalid source and/or destination locator may cause a redirection
   attack.

14.1.1. Treatment of Unknown Source Locator

   The shim sub-layer checks to determine if the requested locator is
   available on any local interface. If not, the shim sub-layer MUST
   reject the request and return an error message with the
   EINVALIDLOCATOR code to the application. If the locator is confirmed
   to be available, the shim sub-layer SHOULD initiate the procedure to
   update the locator list.

   Use of the following socket options and ancillary data requires
   treatment of an unknown source locator:

   o SHIM_LOC_LOCAL_SEND

   o SHIM_LOC_LOCAL_PREF

   o SHIM_LOCLIST_LOCAL

14.1.2. Treatment of Unknown Destination Locator

   If the shim sub-layer turns out to be Shim6, the Shim6 layer MUST
   reject the request for using an unknown destination locator.

   If the shim sub-layer turns out to be HIP, the HIP layer MUST reject
   the request for using an unknown destination locator. There is,
   however, an exceptional case where the HIP layer SHOULD accept the
   request, provided that the HIP association is in the UNASSOCIATED
   state. Details of locator handling in HIP are described in
   Section 4.6 of [RFC6317].

   Use of the following socket options and ancillary data requires
   treatment of an unknown destination locator:

   o SHIM_LOC_PEER_SEND

   o SHIM_LOC_PEER_PREF

   o SHIM_LOCLIST_PEER

15. Acknowledgments

   The authors would like to thank Jari Arkko, who participated in the
   discussion that led to the first version of this document, and Tatuya
   Jinmei, who thoroughly reviewed the early draft version of this
   document and provided detailed comments on sockets API-related
   issues. Thomas Henderson provided valuable comments, especially from
   the HIP perspective.

   The authors sincerely thank the following people for their helpful
   comments regarding the document: Samu Varjonen, Dmitriy Kuptsov,
   Brian Carpenter, Michael Scharf, Sebastien Barre, and Roni Even.

16. References

16.1. Normative References

   [POSIX]   "IEEE Std. 1003.1-2008 Standard for Information
                Technology -- Portable Operating System Interface
                (POSIX). Open group Technical Standard: Base
                Specifications, Issue 7", September 2008,
                <http://www.opengroup.org/austin>.

   [RFC2119]   Bradner, S., "Key words for use in RFCs to Indicate
                Requirement Levels", BCP 14, RFC 2119, March 1997.

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

   [RFC4423]   Moskowitz, R. and P. Nikander, "Host Identity Protocol
                (HIP) Architecture", RFC 4423, May 2006.

   [RFC5533]   Nordmark, E. and M. Bagnulo, "Shim6: Level 3 Multihoming
                Shim Protocol for IPv6", RFC 5533, June 2009.

   [RFC5534]   Arkko, J. and I. van Beijnum, "Failure Detection and
                Locator Pair Exploration Protocol for IPv6 Multihoming",
                RFC 5534, June 2009.

16.2. Informative References

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

   [RFC2782]   Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for
               specifying the location of services (DNS SRV)", RFC 2782,
               February 2000.

   [RFC4291]   Hinden, R. and S. Deering, "IP Version 6 Addressing
               Architecture", RFC 4291, February 2006.

   [RFC5535]   Bagnulo, M., "Hash-Based Addresses (HBA)", RFC 5535,
               June 2009.

   [RFC5770]   Komu, M., Henderson, T., Tschofenig, H., Melen, J., and
               A. Keranen, Ed., "Basic Host Identity Protocol (HIP)
               Extensions for Traversal of Network Address Translators",
               RFC 5770, April 2010.

   [RFC6145]   Li, X., Bao, C., and F. Baker, "IP/ICMP Translation
               Algorithm", RFC 6145, April 2011.

   [RFC6317]   Komu, M. and T. Henderson, "Basic Socket Interface
               Extensions for the Host Identity Protocol (HIP)",
               RFC 6317, July 2011.

   [SHIM6-APP-REFER]
               Nordmark, E., "Shim6 Application Referral Issues", Work
               in Progress, July 2005.

Appendix A. Context Forking

   In this section, an issue concerning context forking and its relation
   to the multihoming shim API are discussed.

   Shim6 supports the notion of context forking. A peer may decide to
   fork a context for a certain reason (e.g., an upper-layer protocol
   prefers to use a different locator pair than the one defined in an
   available context). The procedure of context forking is done
   similarly to the normal context establishment, performing the 4-way
   message exchange. A peer who has decided to fork a context initiates
   the context establishment. Hereafter, we call this peer the
   "initiator". The peer of the initiator is called the "responder".

   Once the forked context is established between the peers, on the
   initiator side, it is possible to apply forked context to the packet
   flow, since the system maintains an association between the forked
   context and the socket owned by the application that has requested
   the context forking. How this association is maintained is an
   implementation-specific issue. However, on the responder side, there
   is a question of how the outbound packet can be multiplexed by the
   shim sub-layer, because there is more than one Shim6 context that
   matches with the ULID pair of the packet flow. There is a need to
   differentiate packet flows not only by the ULID pairs but by some
   other information and associate a given packet flow with a specific
   context.

   Figure 8 gives an example of a scenario where two communicating peers
   fork a context. Initially, there has been a single transaction
   between the peers, by the application 1 (App1). Accordingly, another
   transaction is started, by application 2 (App2). Both of the
   transactions are made based on the same ULID pair. The first context
   pair (Ctx1) is established for the transaction of App1. Given the
   requests from App2, the shim sub-layer on Peer 1 decides to fork a
   context. Accordingly, a forked context (Ctx2) is established between
   the peers, which should be exclusively applied to the transaction of
   App2. Ideally, multiplexing and demultiplexing of packet flows that
   relate to App1 and App2 should be done as illustrated in Figure 8.
   However, as mentioned earlier, the responder needs to multiplex
   outbound flows of App1 and App2 somehow. Note that if a context
   forking occurs on the initiator side, a context forking needs to also
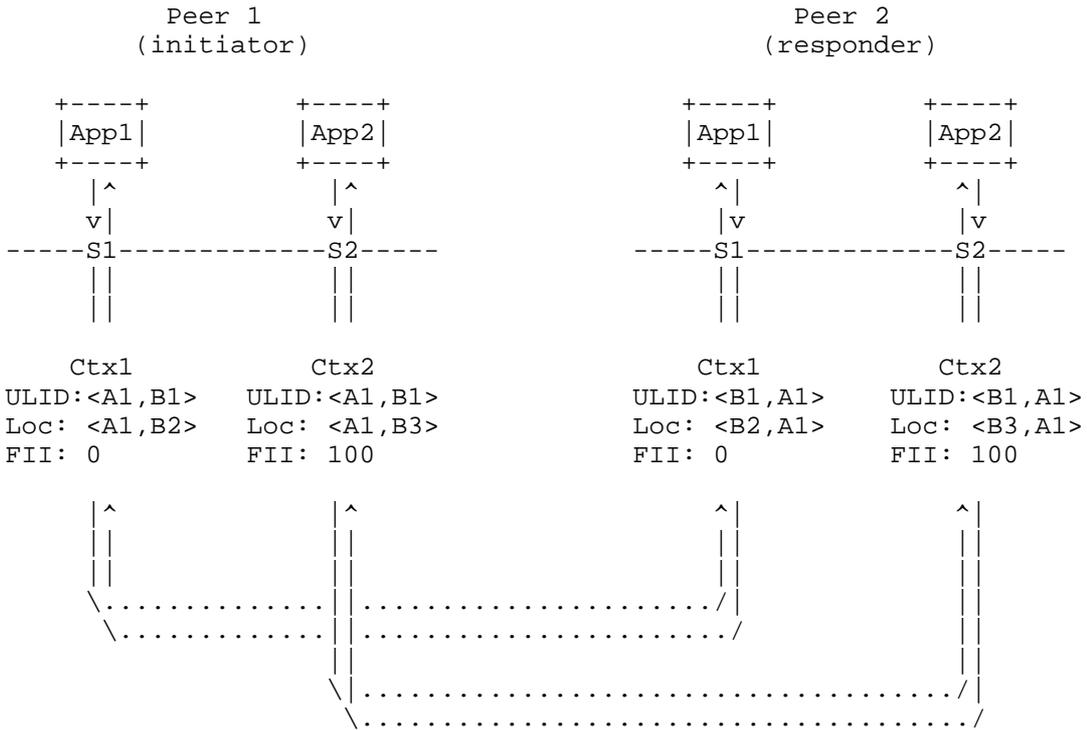   occur on the responder side.

```
        Peer 1                                  Peer 2
      (initiator)                             (responder)


   +----+          +----+              +----+          +----+
   |App1|          |App2|              |App1|          |App2|
   +----+          +----+              +----+          +----+
    |^               |^                 ^|               ^|
    v|               v|                 |v               |v
  -----S1------------S2-----          -----S1------------S2-----
     ||               ||                 ||               ||
     ||               ||                 ||               ||

      Ctx1             Ctx2               Ctx1             Ctx2
  ULID:<A1,B1>     ULID:<A1,B1>       ULID:<B1,A1>     ULID:<B1,A1>
  Loc: <A1,B2>     Loc: <A1,B3>       Loc: <B2,A1>     Loc: <B3,A1>
  FII: 0           FII: 100           FII: 0           FII: 100

     |^               |^                 ^|               ^|
     ||               ||                 ||               ||
     ||               ||                 ||               ||
     \..............||...................../|             ||
      \.............||......................./             ||
                    ||                       ||            ||
                    \|...................................../|
                     \....................................../
```

              Figure 8: Context Forking

   How to solve the issue described above is a topic for further study.
```

Authors' Addresses

   Miika Komu
   Aalto University
   Espoo
   Finland

   Phone: +358505734395
   Fax:   +358947025014
   EMail: miika@iki.fi
   URI:   http://cse.aalto.fi/research/groups/datacommunications/people/


   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