SHIM6 Working Group	M. Komu	
Internet-Draft	HIIT	
Intended status: Informational	M. Bagnulo	
Expires: July 13, 2010	UC3M	
	K. Slavov	
	S. Sugimoto, Ed.	
	Ericsson	
	January 09, 2010	

TOC

Socket Application Program Interface (API) for Multihoming Shim
draft-ietf-shim6-multihome-shim-api-12

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 "shim") inside the IP layer that maintains mappings between identifiers and locators. Examples of the shim are SHIM6 and HIP.

Status of this Memo

Table of Contents

---

1. Introduction

TOC

This document defines socket 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 it applies to SHIM6 and 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 which may be caused when the endhost changes its attachment point to the Internet, for instance, in the case of rehoming event under the multihomed environment. The key design of the shim sub-layer is to treat identifier and locator separately. Identifiers are presented to upper layer protocols and used as communication endpoints. Locators

represent toplogical location of endhosts and are used to route packet from the source to the destiantion. The shim sub-layer maintains mapping of identifiers and locators.
Note that the shim sub-layer may conflict with other multihoming mechanisms such as SCTP and multipath TCP[I-D.ietf-shim6-applicability] (Abley, J., Bagnulo, M., and A. Garcia-Martinez, “Applicability Statement for the Level 3 Multihoming Shim Protocol (Shim6),” November 2009.). To avoid any conflict, only one of SHIM6 and HIP should be in use.
In this document, syntax and semantics of the API are given in the same way as the Posix standard [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.). 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 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 distinction between "connected" sockets and "unconnected" sockets is important when discussing the applicability of the socket 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.
The target readers of this document are application programmers who develop application software which may benefit greatly from multihomed environments. In addition, this document aims to provide necessary information for developers of shim protocols to implement API for enabling advanced locator management.

---

2. Terminology

TOC

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[RFC5533] (Bagnulo, M. and E. Nordmark, “Level 3 multihoming shim protocol,” June 2009.)
- HIP Architecture[RFC4423] (Moskowitz, R. and P. Nikander, “Host Identity Protocol (HIP) Architecture,” May 2006.)

* Reachability Protocol (REAP)[RFC5534] (Arkko, J. and I. Beijnum, "Failure Detection and Locator Pair Exploration Protocol for IPv6 Multihoming," June 2009.)

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:

* 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[I-D.ietf-shim6-app-refer] (Nordmark, E., "Shim6 Application Referral Issues," July 2005.).

  - In the case of SHIM6, an identifier called a ULID 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 a form of hash of public key.

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

* 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.

  - 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] (Bagnulo, M. and E. Nordmark, "Level 3 multihoming shim protocol," June 2009.), 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. As defined in [RFC5533] (Bagnulo, M. and E. Nordmark, "Level 3 multihoming shim protocol," June 2009.), the preferred locator of a host 'A' is denoted as Lp(A).

  - 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 unknown locator.

*Shim - The conceptual sub-layer inside the IP layer which 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.

*Identifier/locator adaptation - The adaptation performed at the shim sub-layer which 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.

*Context - The state information shared by a given pair of peers, which stores a binding between the EID and associated locators. Contexts are maintained by the shim sub-layer.

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

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

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

*Outage - The 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 may be caused by various reasons, such as shortage of network resources, congestion, and human error (faulty operation).

*Working address pair - The address pair is 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.

*Reachability protocol (REAP) - The protocol for detecting failure and exploring reachability in a multihomed environment. REAP is defined in [RFC5534] (Arkko, J. and I. Beijnum, “Failure Detection and Locator Pair Exploration Protocol for IPv6 Multihoming,” June 2009.).

3. System Overview

TOC

Figure 1 (System overview) 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 may also be possible that the shim sub-layer interacts with the transport layer, however, such an interaction is outside the scope of this document.

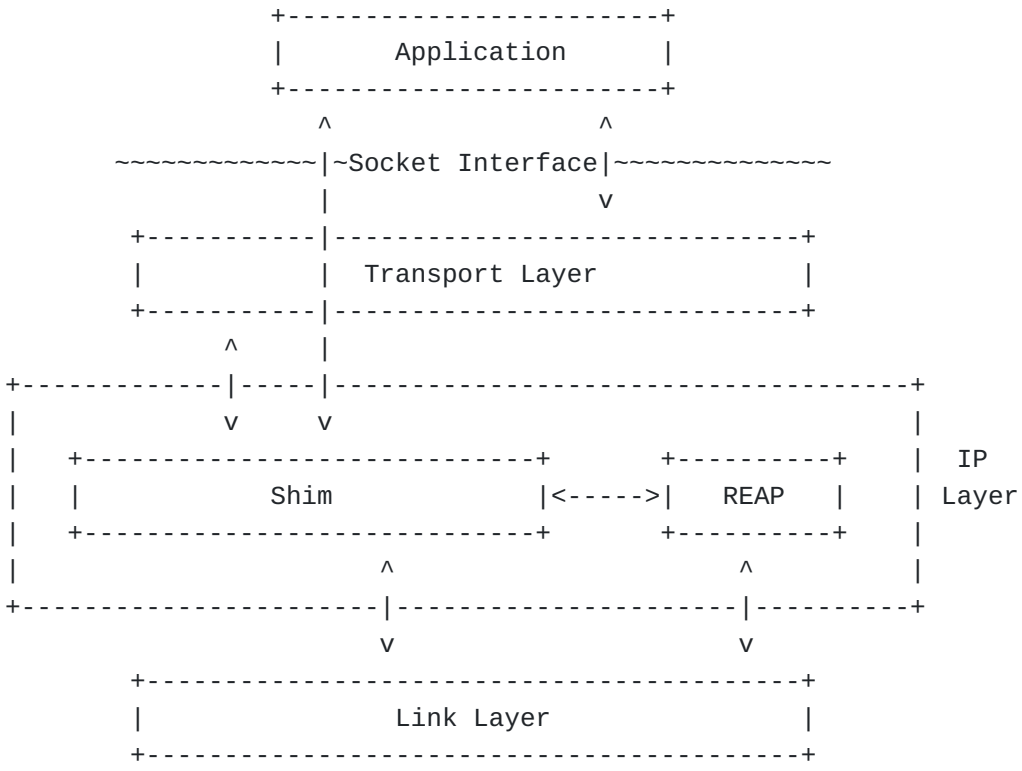

Figure 1: System overview

TOC

4. Requirements

The following is a list of requirements from applications:

*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 the shim sub-layer to apply multihoming support.

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

  -Note that this function is also required by other types of multihoming mechanisms such as SCTP and multipath TCP to avoid potential conflict with the shim sub-layer.

*Locator management.

  -It should be possible to set preferred source and/or destination locator within a given context: Lp(local) and/or Lp(remote).

  -It should be possible to get preferred source and/or destination locator within a given context: Lp(local) and/or Lp(remote).

  -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).

*Notification from applications 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 feedbacks or negative feedbacks to the shim sub-layer. Note that these feedbacks are mentioned in [RFC5534] (Arkko, J. and I. Beijnum, “Failure Detection and Locator Pair Exploration Protocol for IPv6 Multihoming,” June 2009.):

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

  -Applications and/or upper layer protocols (e.g., TCP) may provide negative feedbacks 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. Besides, a receipt of an ICMP error message could be a clue for the application to detect problems. The REAP module may be triggered by these negative feedbacks and invoke the path exploration procedure.

*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.

*Hot-standby. Applications may request the shim sub-layer for the 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 host to immediately replace the current locator pair with an alternative locator pair.

*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.

*Providing locator information to applications. An 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, applications may have additional control on the locator management. For example, an application becomes able to verify if its preference for locator is actually applied to the flow or not.

*Applications should be able to specify if they want to defer the context setup, or if they want context establishment to be started immediately in the case where there is no available context. A deferred context setup means that the initiation of communication should not be blocked to wait for completion of the context establishment.

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

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

---

5. Socket Options for Multihoming Shim Sub-layer

TOC

In this section, socket options that are specific to the shim sub-layer are defined.
Table 1 (Socket options for multihoming shim sub-layer) shows a list of the socket options that are specific to the shim sub-layer. An application may use these socket options for a given socket either by the getsockopt() system call or by the setsockopt() system call. All of these socket options are defined at level SOL_SHIM.
The first column of Table 1 (Socket options for multihoming shim sub-layer) gives the name of the option. The second and third columns indicate whether the option can be handled by the getsockopt() system call and/or by the setsockopt() system call. 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 which indicates whether if the socket is associated with any shim context or not.	int
SHIM_DONTSHIM	o	o	Get or set the parameter which indicates whether to employ the multihoming support by the shim sub-layer or not.	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	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	Get or set the parameter which is used to request the shim sub-layer to store the destination locator of the received IP packet.	int
SHIM_LOC_PEER_RECV	o	o	Get or set the parameter which is used to request the shim sub-layer to store the source locator of the received IP packet.	int
SHIM_LOC_LOCAL_SEND	o	o	Get or set the source locator of outgoing IP packets.	*2
SHIM_LOC_PEER_SEND	o	o	Get or set the destination locator of outgoing IP packets.	*2
SHIM_LOCLIST_LOCAL	o	o	Get or set the list of locators associated with the local EID.	*3
SHIM_LOCLIST_PEER	o	o	Get or set the list of locators associated with the peer's EID.	*3
SHIM_APP_TIMEOUT	o	o	Get or set the timeout value for detecting failure.	int
SHIM_PATHEXPLORE	o	o	Get or set parameters for path exploration and failure detection.	*4
SHIM_CONTEXT_DEFERRED_SETUP	o	o	Get or set the parameter which indicates whether deferred context setup is supported or not.	int

Table 1: Socket options for multihoming shim sub-layer

---

*1: Pointer to a shim_locator which is defined in Section 7 (Data Structures).
*2: Pointer to shim_locator data structure.
*3: Pointer to an array of shim_locator.
*4: Pointer to a shim_pathexplore which is defined in Section 7 (Data Structures).

Figure 2 (System model of sockets API with shim sub-layer) illustrates how the shim specific socket options fit into the system model of socket 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 API continue to work at the layer above the shim sub-layer. That is, those legacy API 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).

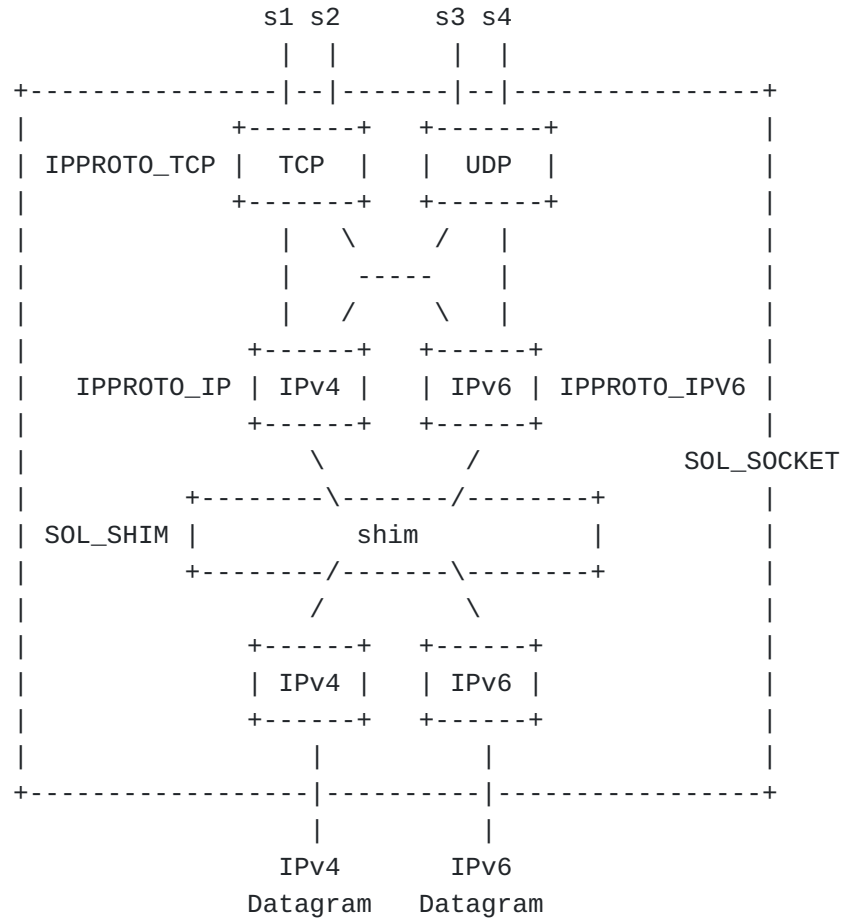

Figure 2: System model of sockets API with shim sub-layer

5.1. SHIM_ASSOCIATED

TOC

The SHIM_ASSOCIATED option is used to check whether the socket is associated with any shim context or not.
This option is meaningful when 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 the locator pair may be identical in some cases.
This option can be specified by getsockopt(). Thus, the option is read-only and the result (0/1/2) is set in the option value (the fourth argument of getsockopt()).
The data type of the option value is an integer. The option value indicates the presence of shim context. A return value 1 means that the socket is associated with a shim context at the shim sub-layer. A return value 0 indicates that there is no shim context associated with the socket. A return value 2 means that it is not known whether the socket is associated with a shim context or not, 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);
```

5.2. SHIM_DONTSHIM

TOC

The SHIM_DONTSHIM option is used to request the shim layer not to 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 0 means that the shim sub-layer is employed if available. An option value 1 means that the application does not want the shim sub-layer to provide the multihoming support for the communication established over the socket.
Default value is set as 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, an error code EOPNOTSUPP MUST be returned.

For example, an application can request the system not to 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);
```

5.3. SHIM_HOT_STANDBY

TOC

The SHIM_HOT_STANDBY option is used to control the shim sub-layer whether to employ a hot-standby connection for the socket or not. 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.
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);
```

5.4. SHIM_PATHEXPLORE

TOC

The application may use this socket option to specify 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.
The 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 7 (Data Structures).
By default, the option value is set to NULL, meaning that the option is disabled.
An error ENOENT is returned when there is no context associated with the socket.
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);
```

5.5. SHIM_LOC_LOCAL_PREF

TOC

The SHIM_LOC_LOCAL_PREF option is used to get 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.
The data type of the option value is a pointer to a locator information data structure which is defined in Section 7 (Data Structures).
By default, the option value is set to NULL, meaning that the option is disabled.
The preferred locator can be set by setsockopt(). The shim sub-layer shall verify requested locator before it updating the preferred locator.
An application can get the preferred locator by getsockopt().
An error ENOENT is returned when there is no context associated with the socket.
An error EINVALIDLOCATOR is returned when the validation of the specified locator failed.
For example, an application can set the preferred locator by using the socket option as follows. Note that some members of the shim_locator (lc_ifidx and lc_flags) are ignored in the 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_preference = 255;
memcpy(lc.lc_addr, &ip6, sizeof(in6_addr));

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

For example, an application can get the preferred locator by using the socket option 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

TOC

The SHIM_LOC_PEER_PREF option is used to get 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.
The data type of the option value is a pointer to the locator information data structure which is defined in Section 7 (Data Structures).
By default, the option value is set to NULL, meaning that the option is disabled.
The preferred locator can be set by setsockopt(). The shim sub-layer shall verify requested locator before it updating the preferred locator.
An application can get the preferred locator by getsockopt().
An error ENOENT is returned when there is no context associated with the socket.
An error EINVALIDLOCATOR is returned when the validation of the requested locator fails.
The usage of the option is same as that of SHIM_LOC_LOCAL_PREF. Note that some members of the shim_locator (lc_ifidx and lc_flags) are ignored in the set operation.

5.7. SHIM_LOC_LOCAL_RECV

TOC

The SHIM_LOC_LOCAL_RECV option can be used to request the shim sub-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.
The 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.
An application can set the option value by setsockopt().
An application can get the option value by getsockopt().
See Section 6 (Ancillary Data for Multihoming Shim Sub-layer) for the procedure to access locator information stored in the ancillary data objects.
An error ENOENT is returned when there is no context associated with the socket.
For example, an application can request the shim sub-layer to store 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);
```

5.8. SHIM_LOC_PEER_RECV

TOC

The SHIM_LOC_PEER_RECV option is used to request the shim sub-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.
The 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 6 (Ancillary Data for Multihoming Shim Sub-layer) for the procedure to access locator information stored in the ancillary data objects.
An error ENOENT is 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_LOC_LOCAL_SEND

TOC

The SHIM_LOC_LOCAL_SEND option is used to request the shim sub-layer to use a specific locator as the source locator for the IP packets to be sent from the socket. Hence this option is effective only when there is a shim context associated with the socket.
The data type of option value is pointer to shim_locator data structure.

An application can set the local locator by setsockopt() providing a valid locator which is stored in a shim_locator data structure. When a zero-filled locator is specified, pre-existing setting of local locator is inactivated.
An application can get the local locator by getsockopt().
An error ENOENT is returned when there is no context associated with the socket.
An error EINVALIDLOCATOR is returned when 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 = 1;
locator.lc_flags = 0;
locator.lc_preference = 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 preferred 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 */
```

5.10. SHIM_LOC_PEER_SEND

TOC

The SHIM_LOC_PEER_SEND option is used to request the shim sub-layer to use a specific locator for the destination locator of IP packets to be

sent from the socket. Hence 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 shim_locator data structure.
An application can set the remote locator by setsockopt() providing a valid locator which is stored in a shim_locator data structure. When a zero-filled locator is specified, pre-existing setting of remote locator is inactivated.
An application can get the specified remote locator by getsockopt().
An error ENOENT is returned when there is no context associated with the socket.
An error EINVALIDLOCATOR when invalid locator is specified.
The usage of the option is as the same as that of SHIM_LOC_LOCAL_SEND option.

5.11. SHIM_LOCLIST_LOCAL

TOC

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. Hence 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 7 (Data Structures) 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 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 local locator list can be set by setsockopt(). The buffer pointed by optval argument should contain an array of locator list.
An error ENOENT is returned when there is no context associated with the socket.
An error EINVALIDLOCATOR is returned when the validation of the specified locator failed.
An error ETOOMANYLOCATORS is returned when the number of locators specified exceeds the limit (SHIM_MAX_LOCATORS).
For example, an application can set a list of locators to be associated with the local EID by using the socket option as follows:

```
        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_preference = 1;
        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_preference = 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 */
        ...
```

5.12. SHIM_LOCLIST_PEER

TOC

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. Hence 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 7 (Data Structures) 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 by 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 by optval argument should contain an array of locator list.
An error ENOENT is returned when there is no context associated with the socket.
An error EINVALIDLOCATOR is returned when the validation of the specified locator failed.
An error ETOOMANYLOCATORS is returned when the number of locators specified exceeds the limit (SHIM_MAX_LOCATORS).
The usage of the option is same as that of SHIM_LOCLIST_LOCAL.

5.13. SHIM_APP_TIMEOUT

TOC

The SHIM_APP_TIMEOUT option is used to get or set the timeout value for application to detect failure. Hence 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 value of Send Timeout value as specified in [RFC5534] (Arkko, J. and I. Beijnum, “Failure Detection and Locator Pair Exploration Protocol for IPv6 Multihoming,” June 2009.)
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 is returned when there is no context associated with the socket.
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);
```

5.14. SHIM_DEFERRED_CONTEXT_SETUP

TOC

The SHIM_DEFERRED_CONTEXT_SETUP option is used to specify whether to enable deferred context setup or not. 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.
The data type for the option value is an integer. The option value should be binary (0 or 1). By default, the value is set to 1, meaning that the context setup is deferred. In order to disable the option, the application should call setsockopt() with option value set to 0.
Note that HIP does not support deferred context setup, by default. When the application requests to enable deferred context setup in case of HIP, it may mean that the application allows the system to start TCP handshake even when there is no IPsec security association with the peer. Such a usage of the SHIM_DEFERRED_CONTEXT_SETUP option should be considered as experimental and left for further study.
For example, an application can disable the deferred context setup by using the socket option as follows:

```
int optval;

optval = 0;

setsockopt(fd, SOL_SHIM, SHIM_DEFERRED_CONTEXT_SETUP,
           &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_DEFERRED_CONTEXT_SETUP,
           &optval, &len);
```

5.15. Applicability

TOC

All the socket options for the shim sub-layer except for SHIM_HOT_STANDBY are applicable to both connected and unconnected sockets. SHIM_HOT_STANDBY socket option is applicable only to connected sockets. This is because the shim sub-layer cannot initiate context establishment to create a hot standby connection when the peer's IP address is not known until the application writes data to the unconnected socket.

5.16. Error Handling

TOC

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 at least one of the necessary validations inside the shim sub-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 the return routability (check if the peer is actually willing to receive data with the specified locator) and the other one is the verification based on crypto identifier mechanisms [RFC3972] (Aura, T., “Cryptographically Generated Addresses (CGA),” March 2005.), [RFC5535] (Bagnulo, M., “Hash Based Addresses (HBA),” June 2009.).

TOC

6. 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 application.
When the application performs locator management by sendmsg() or recvmsg(), a member of the msghdr structure (given in Figure 3 (msghdr structure)) called msg_control holds a pointer to the buffer in which one ore 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] (Stevens, W., Thomas, M., Nordmark, E., and T. Jinmei, “Advanced Sockets Application Program Interface (API) for IPv6,” May 2003.).

```
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 unconnected socket, msg_name stores the socket address of the peer which should be considered to be an identifier rather than a locator. SHIM_LOC_PEER_RECV should be used to get the locator of the peer node.
Table 2 (Shim specific ancillary data) is a list of the shim specific ancillary data which can be used for locator management by recvmsg() or sendmsg(). In any case, the value of cmsg_level must be set as SOL_SHIM.

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	o		shim_feedback{}

Table 2: Shim specific ancillary data

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

6.1. Get Locator from Incoming Packet

TOC

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 local and/or remote locator from the ancillary data.

6.2. Set Locator for Outgoing Packet

TOC

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.
Note that the effect is limited to the datagram transmitted by the sendmsg().
If the specified locator pair is verified, the shim sub-layer overrides the locators of the IP packet.
An error EINVALIDLOCATOR is returned when validation of the specified locator failed.

6.3. Notification from Application to Multihoming Shim Sub-layer

TOC

An application may provide feedbacks to the shim sub-layer about the communication status. Such feedbacks are particularly useful for the shim sub-layer in the absence of REAP mechanism 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 SHIM_FEEDBACK option in cmsg_type.
An error ENOENT is returned when there is no context associated with the socket.
See Section 7.3 (Feedback Information) for details of the data structure to be used.
It is outside the scope of this document how the shim sub-layer would react when a feedback is provided by an application.

6.4. Notification from Multihoming Shim Sub-layer to Application

TOC

The shim sub-layer MAY provide notification about a locator change within a multihome shim context to applications that have concern with the context. Such a notification may be useful, for example, for an application which is sensitive to the characteristics of the current path. A locator change is caused when either of local or peer's locator (or both) is changed. Note that locators discussed here are the ones that appear in the IP packet header, and not the ones that are included in the locator list. A locator change may take place asynchronously. The notification is handled as an out-of-band data by the application.

1. Application calls the select() system call by setting non-NULL value for the fourth argument.

2. When there is a notification, the application reads out-of-band data from the socket by calling recvmsg().

3. The application checks the flag in the msghdr (msg_flags) to see if there is any notification about locator change delivered. If the MSG_SHIM_LOCATOR_CHANGE flag is set, application parses the chain of control message to read a pair of ancillary data objects which contains the source locator and the destination locator. Note that the direction of locator change is distinguished by the value of cmsg_type; SHIM_LOC_*_RECV is used for inbound locator change, and SHIM_LOC_*_SEND is used for outbound locator change.

There is no restriction in terms of applicability of the notification about locator change. The notification can be delivered to any type of socket (connected or unconnected, stream-oriented or datagram-oriented).

TOC

6.5. Applicability

All the ancillary data for the shim sub-layer is applicable to both connected and unconnected sockets.
A 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 gurantee that the locator(s) set in the SHIM_LOC_*_RECV ancillary data is identical to the locator(s) that appear 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.

---

7. Data Structures

TOC

This section data structures for the shim sub-layer. These data structures are either used as a parameter for setsockopt() or getsockopt() (as mentioned in Section 5 (Socket Options for Multihoming Shim Sub-layer)) or as a parameter for ancillary data to be processed by sendmsg() or recvmsg() (as mentioned in Section 6 (Ancillary Data for Multihoming Shim Sub-layer)).

7.1. Placeholder for Locator Information

TOC

As defined in Section 5 (Socket Options for Multihoming Shim Sub-layer), 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 4 (shim locator structure) illustrates the data structure called shim_locator which stores a 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_flags;        /* flags */
        uint16_t   lc_pref;         /* preference value */
        uint32_t   lc_ifidx;        /* interface index */
        struct in6_addr lc_addr;    /* address */
};
```

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 non-zero value indicating the exact address family of the locator.

lc_proto Internet Protocol number for the protocol which is used to handle locator behind NAT. Typically, this value is set as UDP (17) when the locator is a UDP encapsulation interface.

lc_port Port number which is used for handling locator behind NAT.

lc_flags Each bit of the flags represents a specific characteristics of the locator. Hash Based Address (HBA) is defined as 0x01. Cryptographically Generated Address (CGA) is defined as 0x02.

lc_pref Preference of the locator. The preference is represented by an integer.

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

lc_addr Contains the locator. In the case where a locator whose size is smaller than 16 bytes, an encoding rule should be provided for each locator of a given address family. For instance, in case of AF_INET (IPv4), the locator should be in the format of an IPv4-mapped IPv6 address as defined in [RFC4291] (Hinden, R. and S. Deering, “IP Version 6 Addressing Architecture,” February 2006.).

TOC

7.1.1. 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 HIP host to tunnel control and data traffic over UDP[I-D.ietf-hip-nat-traversal] (Komu, M., Henderson, T., Tschofenig, H., Melen, J., and A. Keranen, "Basic HIP Extensions for Traversal of Network Address Translators," October 2009.). Note also that the locator behind NAT is not necessarily an IPv4 address but it 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_flags = 0;
locator.lc_pref = 0;
locator.lc_ifidx = 3;

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

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

Figure 5: Handling locator behind NAT

7.2. Path Exploration Parameter

TOC

As defined in Section 5 (Socket Options for Multihoming Shim Sub-layer), 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 6 (path explore structure) illustrates the data structure. The data structure can be passed to 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 6: 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 [RFC5534] (Arkko, J. and I. Beijnum, "Failure Detection and Locator Pair Exploration Protocol for IPv6 Multihoming," June 2009.).

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

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

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

TOC

7.3. Feedback Information

As mentioned in Section 6.3 (Notification from Application to Multihoming Shim Sub-layer), 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 (feedback information structure).

```
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

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

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.

reserved Reserved field. Must be ignored by the receiver.

TOC

8. System Requirements

This section gives system requirements posed by the sockets API defined in this document. There exist requirements for the system (kernel) to maintain the association between sockets and shim contexts.
As addressed in Section 5 (Socket Options for Multihoming Shim Sub-layer), all the socket options and ancillary data defined in this document except for the SHIM_HOT_STANDBY socket option are applicable to both connected and unconnected sockets.
There are less system requirements to enable support for applications that use connected sockets. This is because the kernel can maintain the association between a connected socket and a shim context in a static manner because a connected socket is bound to a source and destination IP addresses (identifiers). The kernel should be able to identify shim context associated with a connected socket by searching shim context keyed by the pair of source and destination identifiers. However, this is not the case for unnconnected sockets. The kernel needs to dynamically resolve association between an unconnected socket and a shim context, if any, upon packet processing. As to outbound packet processing, the kernel needs to check if there is any shim context whose EID pair matches with the source and destination IP addresses of the user data originating from an unconnected socket. If a matching context is found, the shim sub-layer performs packet processing taking the application's preference into account. Note that the shim sub-layer should be able to backtrack the socket from which the user data was originated. As to inbound packet processing, the shim sub-layer needs to check not only the IP header but also the transport layer protocol header to resolve the destination socket. If the destination socket is resolved and it contains any values concerning the shim sub-layer socket options, the shim sub-layer processes the IP packet as requested (e.g., set locator information of received packet in the ancillary data).

---

9. Relation to Existing Sockets API Extensions

TOC

This section explains relation between the sockets API defined in this document and the existing sockets API extensions.
As mentioned in Section 5 (Socket Options for Multihoming Shim Sub-layer), 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_PREF and SHIM_LOC_PEER_PREF socket options are semantically similar to the IPV6_PKTINFO socket API in the sense that both provide a means for 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 socket APIs in the sense that both provides a means for application to get the source and/or destination IP address of inbound IP packets. getsockname() and getpeername() enable application to get 'name' of the communication endpoints which is represented by a pair of IP address and port number assigned to the socket. getsockname() gives IP address and port number assigned to the socket on the local side, and getpeername() gives IP address and port number of the peer side.

---

10. Operational Considerations

TOC

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

10.1. Conflict Resolution

TOC

There may be a conflicting situation when different applications specify difference preference for the same shim context. For instance, application A and B may establish communication with the same EID pair while both applications have different preference in their choice of local locator. The notion of context forking in SHIM6 can resolve the conflicting situation.
Socket options defined in Section 5 (Socket Options for Multihoming Shim Sub-layer) may cause 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, an unique identifier called Forked Instance Identifier (FII) is assigned to the newly forked context. The forked context is then exclusively associated with the socket through which non-default preference value was specified. The forked context is maintained by the shim sub-layer during the lifetime of associated socket instance. When the socket is closed, the shim sub-layer SHOULD delete associated context. When a forked context is torn down, the SHIM6 implementation should notify the peer about the deletion of forked context.

10.2. Incompatiblility between IPv4 and IPv6

TOC

The shim sub-layer performs identifier/locator adaptation. Therefore, in some cases, 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] (Nordmark, E., “Stateless IP/ICMP Translation Algorithm (SIIT),” February 2000.).
As addressed in the 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. This means that an error may occur during the conversion of identifier and locator. It is ffs exactly how the shim sub-layer should behave in such erroneous cases.

---

11. IANA Considerations

TOC

This document contains no IANA consideration.

---

12. Protocol Constants and Variables

TOC

This section defines protocol constants and variables.

> **SHIM_MAX_LOCATORS** The maximum number of the locators to be included in a locator list. 32.

---

13. Security Considerations

TOC

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

13.1. Treatment of Unknown Locator

TOC

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

13.1.1. Treatment of Unknown Source Locator

TOC

The shim sub-layer checks if the requested locator is available on any of the 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 may require treatment of unknown source locator:

*SHIM_LOC_LOCAL_SEND

*SHIM_LOC_LOCAL_PREF

*SHIM_LOC_LIST_LOCAL

13.1.2. Treatment of Unknown Destination Locator

TOC

If the shim sub-layer turns out to be SHIM6, the SHIM6 implementation MUST reject the request for using unknown destination locator.
If the shim sub-layer turns out to be HIP, the HIP implementation MAY accept the request for using unknown destination locator.
Use of the following socket options and ancillary data may require treatment of unknown destination locator:

*SHIM_LOC_PEER_SEND

*SHIM_LOC_PEER_PREF

*SHIM_LOC_LIST_PEER

---

14. Changes

TOC

14.1. Changes from version 00 to version 01

TOC

*Define shim_locator{} data type which is a placeholder for locator.

*Define shim_pathexplore{} data type in which a set of REAP parameters are stored.

*Remove descriptions about "stickiness" of socket options.

*Deprecate SHIM_IF_RECV and SHIM_IF_SEND socket options.

*Give default value and how to disable given socket option.

14.2. Changes from version 01 to version 02

TOC

*Add section describing context forking.

*Rephrase conclusion section.

*Separate normative references from informative references.

*Remove texts from discussion section that are not relevant to the contents of the document.

*Add section describing change history (this section).

14.3. Changes from version 02 to version 03

TOC

*Add an Appendix section describing the issue of context forking.

14.4. Changes from version 03 to version 04

TOC

*Updated reference.

*Correct typo and grammatical errors.

14.5. Changes from version 04 to version 05

TOC

*Added definition of SHIM_FEEDBACK ancillary data.

*Added an example of code using the SHIM_LOCLIST_LOCAL

*Added SHIM_LOC_LOCAL_SEND and SHIM_LOC_PEER_SEND socket options.

14.6. Changes from version 05 to version 06

TOC

*Updated references.

14.7. Changes from version 06 to version 07

TOC

*Resolved editorial issues.

14.8. Changes from version 07 to version 08

TOC

No changes are made except for updates of the references.

14.9. Changes from version 08 to version 09

TOC

*Updated texts for Section 1 and Section 5 according to the comments provided by Samu Varjonen.

*Made it clear that downgrading the multihoming shim support (i.e., specifying value 1 with the SHIM_DONTSHIM socket option) is only allowed before the socket is connected.

*Updated locator information (shim_locator{}) so that it can contain a locator behind NAT.

14.10. Changes from version 09 to version 10

TOC

*Addressed applicability of socket options and ancillary data for the shim sub-layer.

*Addressed system requirements.

*Removed unnecessary description about deprecated socket option (SHIM_IF_RECV).

14.11. Changes from version 10 to version 11

TOC

*Added short descriptions about connected sockets and unconnected sockets.

*Relaxed applicability of the socket options.

*Relaxed applicability of the ancillary data.

*Added notification about locator change.

14.12. Changes from version 11 to version 12

TOC

*Reflected comments from Brian Karpenter

*Reflected comments from Michael Scharf

---

15. Acknowledgments

TOC

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 sockets API related issues. Thomas Henderson provided valuable comments especially from HIP perspectives. Authors sincerely thank to the following people for their help to improve this document: Samu Varjonen and Dmitriy Kuptsov.

---

16. References

TOC

16.1. Normative References

TOC

[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 (TXT).
[RFC4423]	Moskowitz, R. and P. Nikander, "Host Identity Protocol (HIP) Architecture," RFC 4423, May 2006 (TXT).
[RFC5533]	Bagnulo, M. and E. Nordmark, "Level 3 multihoming shim protocol," RFC 5533, June 2009 (TXT).
[RFC5534]	Arkko, J. and I. Beijnum, "Failure Detection and Locator Pair Exploration Protocol for IPv6 Multihoming," RFC 5534, June 2009 (TXT).

16.2. Informative References

TOC

[I-D.ietf-hip-nat-traversal]	Komu, M., Henderson, T., Tschofenig, H., Melen, J., and A. Keranen, "Basic HIP Extensions for Traversal of Network Address Translators," Internet Draft draft-ietf-hip-nat-traversal-09, October 2009.
[I-D.ietf-shim6-app-refer]	Nordmark, E., "Shim6 Application Referral Issues," draft-ietf-shim6-app-refer-00 (work in progress), July 2005 (TXT).
[I-D.ietf-shim6-applicability]	Abley, J., Bagnulo, M., and A. Garcia-Martinez, "Applicability Statement for the Level 3 Multihoming Shim Protocol (Shim6)," draft-ietf-shim6-applicability-04 (work in progress), November 2009 (TXT).
[RFC2765]	Nordmark, E., "Stateless IP/ICMP Translation Algorithm (SIIT)," RFC 2765, February 2000 (TXT).
[RFC3972]	Aura, T., "Cryptographically Generated Addresses (CGA)," RFC 3972, March 2005 (TXT).
[RFC4291]	Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture," RFC 4291, February 2006 (TXT).
[RFC5535]	Bagnulo, M., "Hash Based Addresses (HBA)," RFC 5535, June 2009 (TXT).

TOC

Appendix A. Context Forking

In this section, an issue concerning context forking and its relation to the multihoming shim API are discussed.
SHIM6 supports a notion of context forking. A peer may decide to fork a context for certain reason (e.g. upper layer protocol prefers to use different locator pair than the one defined in available context). The procedure of forking context is done similar 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 how the outbound packet can be multiplexed by the shim sub-layer because there are more than one SHIM6 contexts that match 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 (context forking) 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 (context forking). 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 occur also on the responder side.

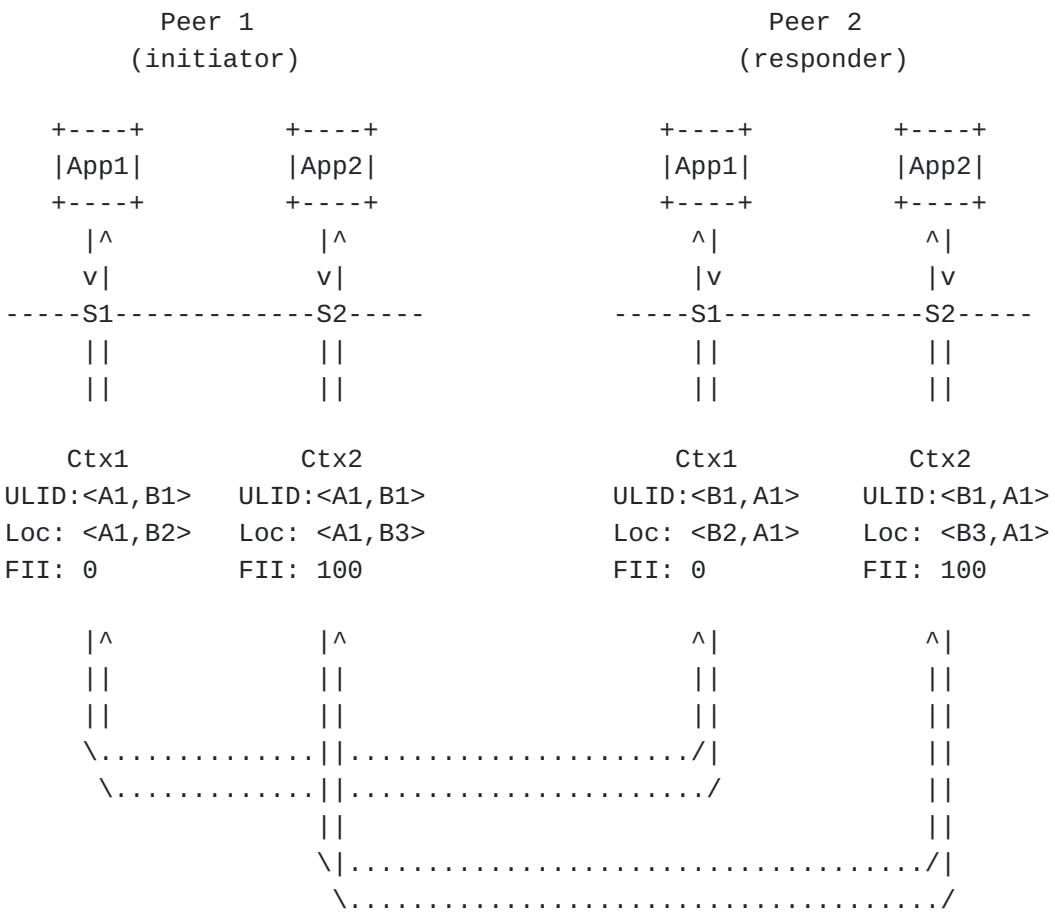

Figure 8: context forking

Any solution is needed to overcome the problem mentioned above.

Authors' Addresses

TOC

	Miika Komu
	Helsinki Institute 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@sfc.wide.ad.jp