

SHIM6 Working Group	M. Komu
Internet-Draft	HIIT
Intended status: Informational	M. Bagnulo
Expires: October 05, 2011	UC3M
	K. Slavov
	S. Sugimoto, Ed.
	Ericsson
	April 03, 2011

Socket Application Program Interface (API) for Multihoming Shim
draft-ietf-shim6-multihome-shim-api-17

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

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on October 05, 2011.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

[Table of Contents](#)

- *1. [Introduction](#)
- *2. [Requirements Language](#)
- *3. [Terminology and Background](#)
- *4. [System Overview](#)
- *5. [Requirements](#)
- *6. [Socket Options for Multihoming Shim Sub-layer](#)
 - *6.1. [SHIM_ASSOCIATED](#)
 - *6.2. [SHIM_DONTSHIM](#)
 - *6.3. [SHIM_HOT_STANDBY](#)
 - *6.4. [SHIM_LOC_LOCAL_PREF](#)
 - *6.5. [SHIM_LOC_PEER_PREF](#)
 - *6.6. [SHIM_LOC_LOCAL_RECV](#)
 - *6.7. [SHIM_LOC_PEER_RECV](#)
 - *6.8. [SHIM_LOC_LOCAL_SEND](#)
 - *6.9. [SHIM_LOC_PEER_SEND](#)
 - *6.10. [SHIM_LOCLIST_LOCAL](#)
 - *6.11. [SHIM_LOCLIST_PEER](#)
 - *6.12. [SHIM_APP_TIMEOUT](#)
 - *6.13. [SHIM_PATHEXPLORE](#)

- *6.14. [SHIM_DEFERRED_CONTEXT_SETUP](#)
- *6.15. [Applicability](#)
- *6.16. [Error Handling](#)
- *7. [Ancillary Data for Multihoming Shim Sub-layer](#)
- *7.1. [Get Locator from Incoming Packet](#)
- *7.2. [Set Locator for Outgoing Packet](#)
- *7.3. [Notification from Application to Multihoming Shim Sub-layer](#)
- *7.4. [Applicability](#)
- *8. [Data Structures](#)
- *8.1. [Data Structure for Locator Information](#)
- *8.1.1. [Handling Locator behind NAT](#)
- *8.2. [Path Exploration Parameter](#)
- *8.3. [Feedback Information](#)
- *9. [System Requirements](#)
- *10. [Relation to Existing Sockets API Extensions](#)
- *11. [Operational Considerations](#)
- *11.1. [Conflict Resolution](#)
- *11.2. [Incompatibility between IPv4 and IPv6](#)
- *12. [IANA Considerations](#)
- *13. [Protocol Constants and Variables](#)
- *14. [Security Considerations](#)
- *14.1. [Treatment of Unknown Locator](#)
- *14.1.1. [Treatment of Unknown Source Locator](#)
- *14.1.2. [Treatment of Unknown Destination Locator](#)
- *15. [Changes](#)
- *15.1. [Changes from version 00 to version 01](#)

- *15.2. [Changes from version 01 to version 02](#)
- *15.3. [Changes from version 02 to version 03](#)
- *15.4. [Changes from version 03 to version 04](#)
- *15.5. [Changes from version 04 to version 05](#)
- *15.6. [Changes from version 05 to version 06](#)
- *15.7. [Changes from version 06 to version 07](#)
- *15.8. [Changes from version 07 to version 08](#)
- *15.9. [Changes from version 08 to version 09](#)
- *15.10. [Changes from version 09 to version 10](#)
- *15.11. [Changes from version 10 to version 11](#)
- *15.12. [Changes from version 11 to version 12](#)
- *15.13. [Changes from version 12 to version 13](#)
- *15.14. [Changes from version 13 to version 14](#)
- *15.15. [Changes from version 14 to version 15](#)
- *15.16. [Changes from version 15 to version 16](#)
- *15.17. [Changes from version 16 to version 17](#)
- *16. [Acknowledgments](#)
- *17. [References](#)
 - *17.1. [Normative References](#)
 - *17.2. [Informative References](#)
- *Appendix A. [Context Forking](#)
- *[Authors' Addresses](#)

1. Introduction

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 rehomeing event under the multihomed environment. There is, however, a need for 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 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. 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:

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:

*SHIM6 Protocol Specification [\[RFC5533\]](#)

*HIP Architecture [\[RFC4423\]](#)

*Reachability Protocol (REAP) [\[RFC5534\]](#)

*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](#)].

- 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 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. Locator 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, locator can be a private IPv4 address when the host is behind a NAT. Section [Section 8.1.1](#) gives detailed description about handling of locator behind 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 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 HBA set.

- Verified locator - A verified locator means that the locator is considered to be reachable according to the result of REAP

return routability check. Note that the verification applies only to peer's 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. 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 the case of SHIM6.

*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 [\[RFC5534\]](#).

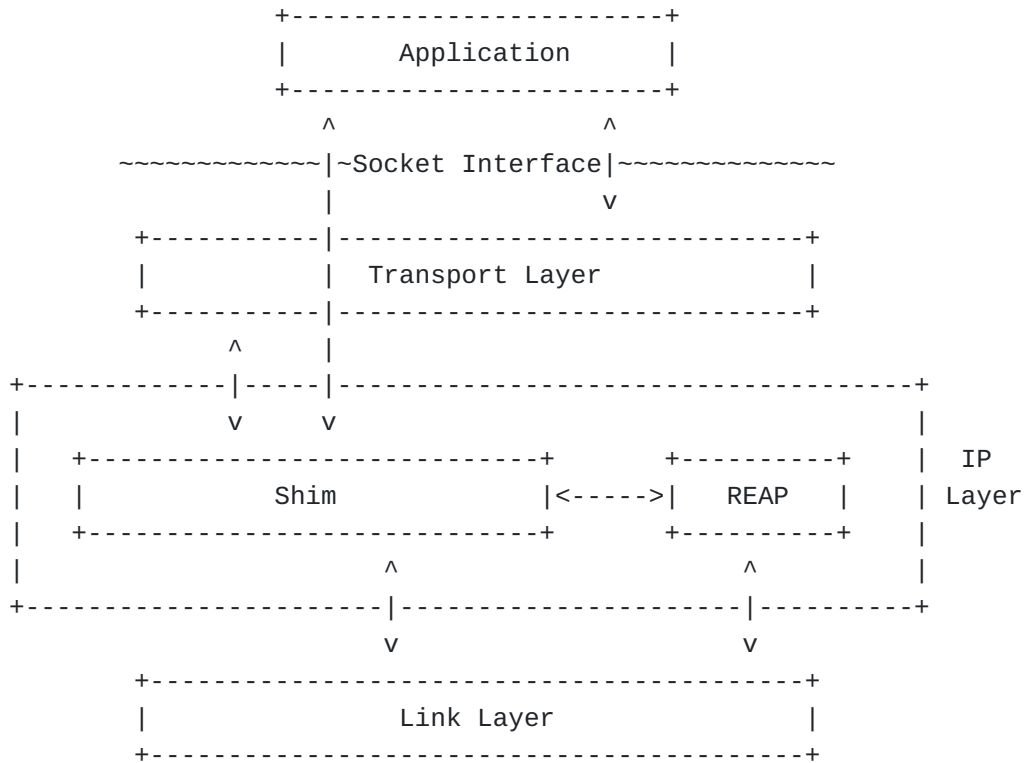
In this document, syntax and semantics of the API are given in the same way as in the Posix standard [\[POSIX\]](#). The API specifies how to use ancillary data (aka cmsg) to access the locator information with `recvmsg()` and/or `sendmsg()` I/O calls. The API is described in C language and data types 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 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.

[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.



5. 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.

- It should be possible to get 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).

*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 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 these negative feedback 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 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.

*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 packets.

-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 know if the shim-sublayer supports deferred context setup or not.

*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.

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 as 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 which indicates whether the socket is associated (1) with any shim context or not (0).	int
SHIM_DONTSHIM	o	o	Get or set the parameter which indicates whether to employ the multihoming	int

optname	get	set	description	dtype
			support by the shim sub-layer or not.	
SHIM_HOT_STANDBY	0	0	Get or set the parameter to request the shim sub-layer to prepare a hot-standby connection.	int
SHIM_LOC_LOCAL_PREF	0	0	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	0	0	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	0	0	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	0	0	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	0	0	Get or set the source locator of outgoing IP packets.	Note 1
SHIM_LOC_PEER_SEND	0	0	Get or set the destination locator of outgoing IP packets.	Note 1
SHIM_LOCLIST_LOCAL	0	0	Get or set the list of locators associated with the local EID.	Note 2
SHIM_LOCLIST_PEER	0	0	Get or set the list of locators associated with the peer's EID.	Note 2
SHIM_APP_TIMEOUT	0	0	Get or set the Send Timeout value of the REAP protocol.	int
SHIM_PATHEXPLORE	0	0	Get or set parameters for path exploration and failure detection.	Note 3
SHIM_CONTEXT_DEFERRED_SETUP	0		Get the parameter which indicates whether deferred	int

optname	get	set	description	dtype
			context setup is supported or not.	

Socket options for multihoming shim sub-layer

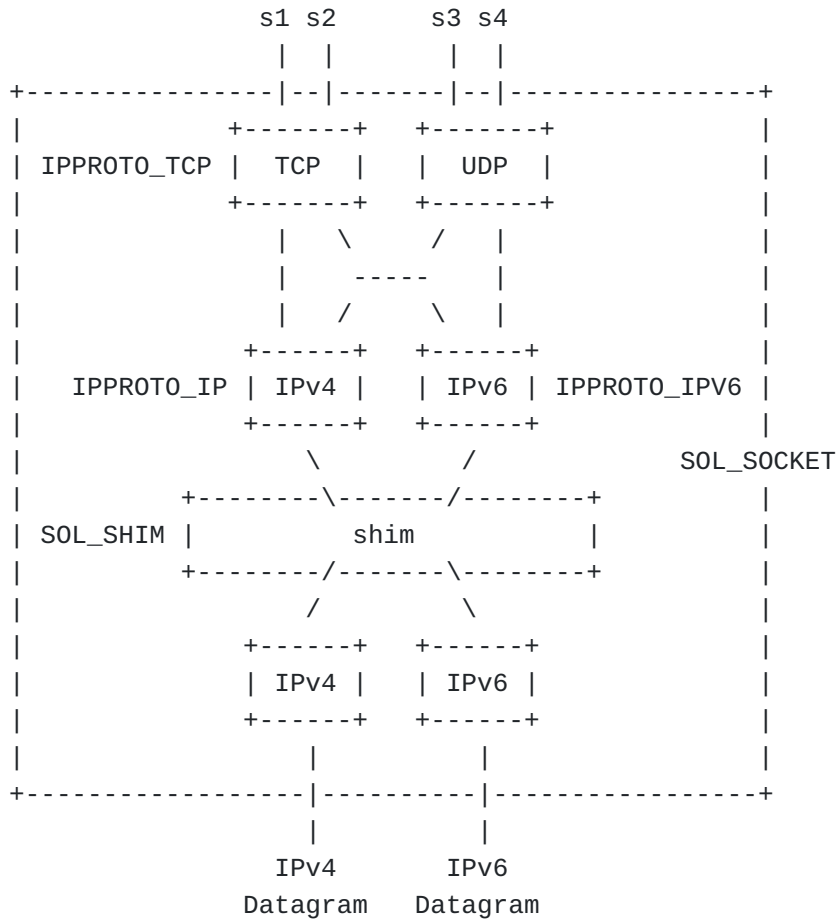
Note 1: Pointer to a shim_locator which is defined in [Section 8](#).

Note 2: Pointer to an array of shim_locator.

Note 3: Pointer to a shim_pathexplore which is defined in [Section 8](#).

[Figure 2](#) 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).



6.1. SHIM_ASSOCIATED

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.

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, an 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 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);
```

6.2. SHIM_DONTSHIM

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);
```

6.3. SHIM_HOT_STANDBY

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.

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

When there is no shim context associated with the socket, an 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 preference of locator in the same way.

The data type of the option value is a pointer to a locator information data structure which is defined in [Section 8](#).

When an application specifies the socket option to an unconnected socket, an error code EOPNOTSUPP is returned to the application. When there is no shim context associated with the socket, an error code ENOENT is returned to the application.

An error EINVALLOCATOR 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 have no effect in a set operation. Below is an example of 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 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 are more than one candidate destination locators, the shim sub-layer makes selection based on the priority and weight specified for each locator.

The data type of the option value is a pointer to the locator information data structure which is defined in [Section 8](#).

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

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

An error EINVALLOCATOR is returned when the validation of the requested locator fails.

An error EUNREACHABLELOCATOR is returned when the requested locator is determined to be not reachable according to a reachability check.

The usage of the option is 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 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()`. 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 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, an error code EOPNOTSUPP is returned to the application.

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

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);
```

6.7. SHIM_LOC_PEER_RECV

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()`. 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 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, an error code EOPNOTSUPP is returned to the application.

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

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

6.8. SHIM_LOC_LOCAL_SEND

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. 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 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()`.

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

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

An error EINVALLOCATOR 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 the shim sub-layer to 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 shim_locator data structure.

An application can set the remote locator by setsockopt() providing a 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(). The difference between the SHIM_LOC_PEER_SEND option and the SHIM_LOC_PEER_PREF option is that the former guarantee the use of requested locator when applicable whereas the latter does not.

When the application specifies the socket option to an unconnected socket, an error code EOPNOTSUPP is returned to the application. When there is no shim context associated with the socket, an error code ENOENT is returned to the application. An error EINVALLOCATOR is returned when the validation of the requested locator fails. An error EUNVERIFIEDLOCATOR is returned when reachability for the requested locator has not been verified yet. An error EUNREACHABLELOCATOR is returned when the requested locator is determined to be not reachable according to a reachability check. The usage of the option is the same as that of 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 MUST 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, an error code EOPNOTSUPP is returned to the application.

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

An error EINVALLOCATOR is returned when the validation of any of the specified locators failed.

An error 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 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 MUST 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.

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

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

An error EINVALLOCATOR is returned when the validation of any of the specified locators failed.

An error EUNVERIFIEDLOCATOR is returned when reachability for the requested locator has not been verified yet.

An error EUNREACHABLELOCATOR is returned when the requested locator is determined to be not reachable according to a reachability check.

An error 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 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 the REAP protocol[\[RFC5534\]](#). 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 value of Send Timeout value as specified in [\[RFC5534\]](#)

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

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

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

When there is no shim context associated with the socket, an 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 deferred context setup is possible 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. Note that the socket option is read-only and the option value can be ready by getsockopt().

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

When the application specifies the socket option to an unconnected socket, an 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 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 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 application.

When the application performs locator management by sendmsg() or recvmsg(), a member of the msg_hdr structure (given in [Figure 21](#)) 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 */
};

```

In the case of 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 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	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{}

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 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, an error code `ENOENT` is returned to the application.

An error code `EINVALIDLOCATOR` is returned when validation of the specified locator fails.

An error `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.

An error `EUNREACHABLELOCATOR` is returned when the requested locator is determined to be not reachable 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 are 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 `SHIM_FEEDBACK` option in `cmsg_type`.

When there is no shim context associated with the socket, an 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 how the shim sub-layer would react when a 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 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.

[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. 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.

```
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 22](#) illustrates the data structure called `shim_locator` which stores a locator information.

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. The value MUST be set to zero when there is no NAT involved. When the locator is behind NAT, the value MUST be set to `IPPROTO_UDP`.

lc_port

Port number which is used for handling locator behind NAT.

lc_prio

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

lc_weight

The 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 operation.

lc_addr

Contains the locator. In 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 characteristics of the locator. Hash Based Address (HBA) is defined as 0x01. Cryptographically Generated Address (CGA) is defined as 0x02. This field has no effect in set operation.

[8.1.1. Handling Locator behind NAT](#)

```
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));
```

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[\[RFC5770\]](#). 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.

8.2. Path Exploration Parameter

```
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 */
};
```

As defined in [Section 6](#), 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 24](#) illustrates the data structure. The data structure can be passed to getsockopt() or setsockopt() as an argument.

pe_probenum

Indicates the number of initial probe messages to be sent. The value MUST be set as per [\[RFC5534\]](#).

pe_keepaliveto

Indicates 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, path exploration procedure will be carried out by sending a REAP Probe message. The value MUST be set as per [\[RFC5534\]](#).

pe_keepaliveint

Indicates 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 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. 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

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

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 25](#).

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.

[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 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 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.

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 difference preference for the same shim context. For instance, suppose if application A and B 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.

It is possible that socket options defined in [Section 6](#) 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 the application specifies SHIM_LOC_*_SEND specifying a different source or destination locator which 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) which conflict with preference specified by the applications either by SHIM_LOC_PEER_SEND or SHIM_LOC_PEER_PREF, the shim sub-layer SHOULD supersede the preference made by the application over the preference 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 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 techniques [\[RFC2765\]\[I-D.ietf-behave-v6v4-xlate\]](#).

As studied in the previous works on protocol conversion [\[RFC2765\]\[I-D.ietf-behave-v6v4-xlate\]](#), some of the features (IPv6 routing headers, hop-by-hop extension headers, and 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 outside the scope of this document to describe how the shim sub-layer should behave in such erroneous cases.

12. IANA Considerations

There is 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 IETF or IANA but they are platform-specific.

13. Protocol Constants and Variables

This section defines protocol constants and variables.

SHIM_MAX_LOCATORS The maximum number of the 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 requests 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.

14.1.1. Treatment of Unknown Source Locator

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 requires treatment of unknown source locator:

*SHIM_LOC_LOCAL_SEND

*SHIM_LOC_LOCAL_PREF

*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 an UNASSOCIATED state. Details of locator handling in HIP is described in section 4.6 of [\[I-D.ietf-hip-native-api\]](#).

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

*SHIM_LOC_PEER_SEND

*SHIM_LOC_PEER_PREF

*SHIM_LOCLIST_PEER

15. Changes

15.1. Changes from version 00 to version 01

*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.

15.2. Changes from version 01 to version 02

*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).

15.3. Changes from version 02 to version 03

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

15.4. Changes from version 03 to version 04

*Updated reference.

*Correct typo and grammatical errors.

15.5. Changes from version 04 to version 05

*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.

15.6. Changes from version 05 to version 06

*Updated references.

15.7. Changes from version 06 to version 07

*Resolved editorial issues.

15.8. Changes from version 07 to version 08

No changes are made except for updates of the references.

15.9. Changes from version 08 to version 09

*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.

15.10. Changes from version 09 to version 10

*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).

15.11. Changes from version 10 to version 11

*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.

15.12. Changes from version 11 to version 12

*Reflected comments from Brian Carpenter.

*Reflected comments from Michael Scharf.

15.13. Changes from version 12 to version 13

*Reflected comments from Sebastien Barre.

*Removed the description about the notification from the shim sub-layer to applications.

*Narrowed down the scope of the applicability of the socket options and the ancillary data.

15.14. Changes from version 13 to version 14

*No change was made. The draft was re-submitted to avoid expiration.

15.15. Changes from version 14 to version 15

*Addressed the difference between SHIM_LOC_PEER_SEND and SHIM_LOC_PEER_PREF.

*Made clear distinction between validation of locator and verification of locator, and introduced two errors: EUNVERIFIEDLOCATOR and EUNREACHABLELOCATOR.

*Addressed exceptional case for HIP in handling of unknown destination locator.

15.16. Changes from version 15 to version 16

Updated the documents reflecting the comments received during the IETF Last Call.

*Added Keepalive Interval (pe_keepaliveint) as a member of the shim_pathexplore{} data structure.

*Addressed the unit of pe_keepaliveto.

*Rephrased the last sentence in Appendix A to make it clear that the addressed issue is for further study.

*Corrected a typo.

15.17. Changes from version 16 to version 17

Updated the documents reflecting the comments received during the IESG review.

*Applied the RFC 2119 terminology more strictly.

*Made it clear whether each socket option can be set and/or get.

*Made some adjustments to the semantics of SHIM_LOC_LOCAL_PREF.

*Made the usage of lc_proto clear.

*Removed a misleading sentence from the paragraph describing lc_ifidx.

16. Acknowledgments

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

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

17. References

17.1. Normative References

[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.
[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.

17.2. Informative References

[I-D.ietf-behave-v6v4-xlate]	Li, X, Bao, C and F Baker, " IP/ICMP Translation Algorithm ", Internet-Draft draft-ietf-behave-v6v4-xlate-23, September 2010.
[I-D.ietf-shim6-app-refer]	Nordmark, E, " Shim6 Application Referral Issues ", Internet-Draft draft-ietf-shim6-app-refer-00, July 2005.
[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, " Basic Host Identity Protocol (HIP) Extensions for Traversal of Network Address Translators ", RFC 5770, April 2010.

[I-D.ietf-hip-native-api]	Komu, M and T Henderson, " Basic Socket Interface Extensions for Host Identity Protocol (HIP) ", Internet-Draft draft-ietf-hip-native-api-12, January 2010.
---------------------------	---

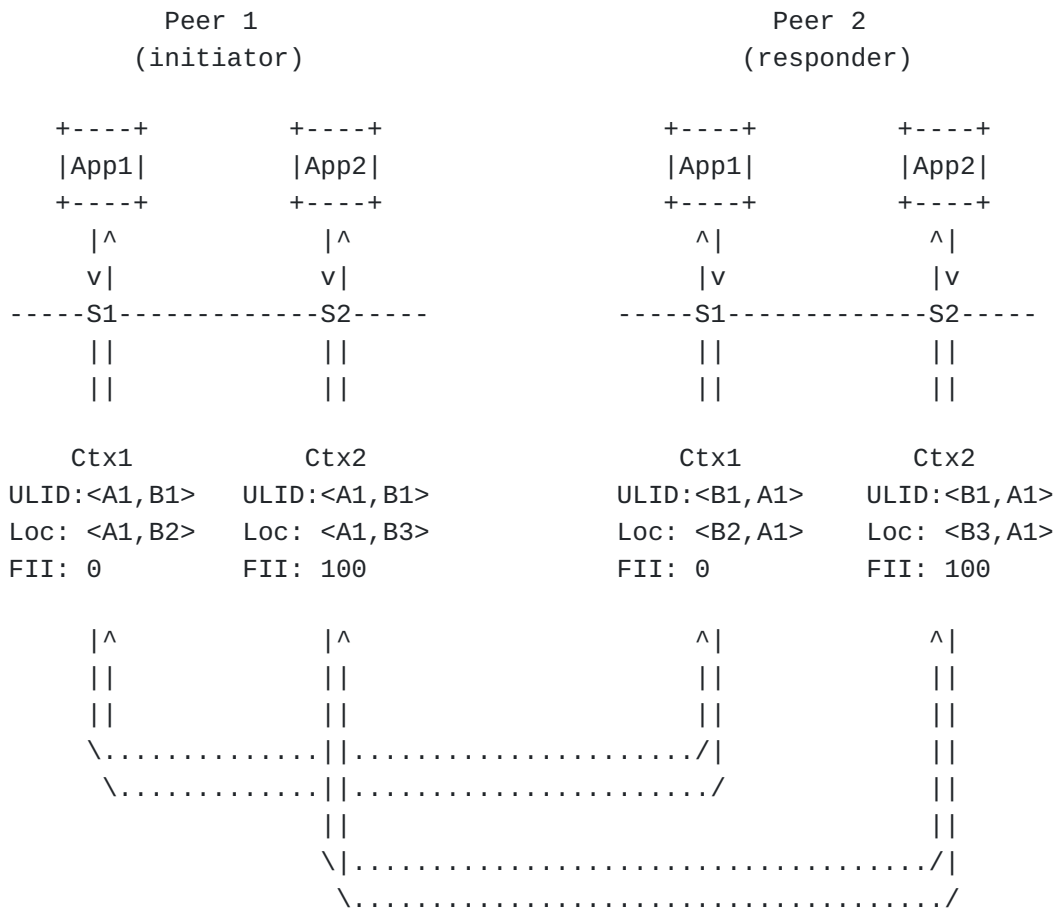
[Appendix A.](#) Context Forking

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 26](#) 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 26](#). 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. It is for further study how to solve the issue described above.

[Authors' Addresses](#)

Miika Komu Komu Helsinki Institute for Information Technology
 Tammasaarenkatu 3 Helsinki, Finland Phone: +358503841531 EMail:
miika@iki.fi URI: <http://www.hiit.fi/>

Marcelo Bagnulo 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 Slavov Ericsson Research Nomadiclab Hirsalantie 11
Jorvas, FI-02420 Finland Phone: +358 9 299 3286 EMail:
kristian.slavov@ericsson.com

Shinta Sugimoto editor Sugimoto Nippon Ericsson K.K. Koraku Mori
Building 1-4-14, Koraku, Bunkyo-ku Tokyo, 112-0004 Japan Phone: +81
3 3830 2241 EMail: shintas@wide.ad.jp