SHIM6 Working Group	M. Komu
Internet-Draft	HIIT
Intended status: Informational	M. Bagnulo
Expires: November 8, 2009	UC3M
	K. Slavov
	S. Sugimoto, Ed.
	Ericsson
	May 07, 2009

Socket Application Program Interface (API) for Multihoming Shim draft-ietf-shim6-multihome-shim-api-08

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.

Table of Contents

1. Introduction

TOC

HIP and SHIM6 have a commonality in their protocol design in the sense that the roles of an IP address as an identifier and a locator are clearly distinguished. Hereafter this design principle is called "identifier/locator separation" in this document. Both protocols aim to solve problems that are specific to multihoming environment in an endhost centric approach. In these protocols, a sub-layer within the IP layer maintains mappings of identifiers and locators.
The shim layer is useful in a sense that the IP layer can maintain the mapping of an identifier to the corresponding locators. Under a multihomed environment, typically, a host has more than one IP address at a time. During the transaction, the host may be required to switch the IP address in use to another IP address to preserve the communication. Such an address update should be kept hidden from the upper layer protocols to avoid communication disruption. The shim layer aims to make the address update transparent to the upper layer protocols.
In a system which is based on identifier/locator separation, upper layer protocols are expected to deal with identifiers for establishing and handling the communications. If an application wants to have multihoming support from the shim layer, the IP addresses specified as source and destination addresses must be identifiers. However, this does not necessarily mean that applications are prohibited to choose

specific locators for its communication. It may be useful for some applications to specify a preferred locator for a given flow.

This document recommends that the switching of identifier and locator is done only once inside the TCP/IP stack of an endhost. That is, if multiple shim sub-layers exist at the IP layer, any one of them should be applied exclusively for a given flow.

As this document specifies sockets API extensions, it is written so that the syntax and semantics are in line with 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.) as much as possible. The API specified in this document defines how to use ancillary data (aka cmsg) to access the locator information with recvmsg() and/or sendmsg() I/O calls. The definition of API is presented in C language and data types follow 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 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 multihoming 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[I-D.ietf-shim6-proto] (Bagnulo, M. and E. Nordmark, “Level 3 multihoming shim protocol,” February 2009.)
- *HIP Architecture[RFC4423] (Moskowitz, R. and P. Nikander, “Host Identity Protocol (HIP) Architecture,” May 2006.)
- *Reachability Protocol (REAP)[I-D.ietf-shim6-failure-detection] (Arkko, J. and I. Beijnum, “Failure Detection and Locator Pair Exploration Protocol for IPv6 Multihoming,” June 2008.)

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.

* 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 [I-D.ietf-shim6-proto] (Bagnulo, M. and E. Nordmark, “Level 3 multihoming shim protocol,” February 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 [I-D.ietf-shim6-proto] (Bagnulo, M. and E. Nordmark, “Level 3 multihoming shim protocol,” February 2009.), the preferred locator of a host 'A' is denoted as Lp(A).

* 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 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 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 [I-D.ietf-shim6-failure-detection] (Arkko, J. and I. Beijnum, “Failure Detection and Locator Pair Exploration Protocol for IPv6 Multihoming,” June 2008.).

---

3. System Overview

Figure 1 (System overview) illustrates the system overview. The shim layer and REAP component exist inside the IP layer. Applications use the sockets API defined in this document to interface with the shim layer and the transport layer for locator management, failure detection, and path exploration.
It may also be possible that the shim layer interacts with the transport layer, however, such an interaction is outside the scope of this document.

```
                 +------------------------+
                 |       Application      |
                 +------------------------+
                    ^                 ^
       ~~~~~~~~~~~~~|~Socket Interface|~~~~~~~~~~~~~~
                    |                 v
        +-----------|-------------------------------+
        |           |  Transport Layer              |
        +-----------|-------------------------------+
              ^     |
+-------------|-----|-------------------------------------+
|             v     v                                     |
|   +-----------------------------+       +----------+    |  IP
|   |            Shim             |<----->|   REAP   |    | Layer
|   +-----------------------------+       +----------+    |
|                       ^                      ^          |
+-----------------------|----------------------|----------+
                        v                      v
        +-------------------------------------------+
        |                Link Layer                 |
        +-------------------------------------------+
```

Figure 1: System overview

4. Requirements

TOC

The following is a list of requirements from applications:

- *Locator management. The shim layer selects a pair of locators for sending IP packets within a given context. The selection is made by taking miscellaneous conditions into account such as reachability of the path, application's preference, and characteristics of path. From applications' perspective:

  - -It should be possible to obtain the lists of locators of a given context: Ls(local) and Ls(remote).

  - -It should be possible to obtain the preferred locators of a given context: Lp(local) and Lp(remote).

*Notification from applications to the shim 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 layer. [NOTE: These feedbacks are mentioned in [I-D.ietf-shim6-failure-detection] (Arkko, J. and I. Beijnum, “Failure Detection and Locator Pair Exploration Protocol for IPv6 Multihoming,” June 2008.)]:

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

  -Applications and/or upper layer protocols (e.g., TCP) may provide negative feedbacks to the shim 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 layer. Applications should be able to inform the shim 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 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 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.

- 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 layer to apply multihoming support.

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

- An application should be able to know if the communication is now being served by the shim 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 Layer

TOC

In this section, socket options that are specific to multihomed shim are defined.
Table 1 (Socket options for multihoming shim) shows a list of the socket options that are specific to the multihoming shim layer. An application may specify 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) 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		Check if the socket is associated with any shim context or not.	int
SHIM_DONTSHIM	o	o	Request the shim layer not to apply any multihoming support for the communication.	int
SHIM_HOT_STANDBY	o	o	Request the shim layer to prepare a hot-standby connection (in addition to the current path).	int
SHIM_LOC_LOCAL_PREF	o	o	Get or set the preferred locator on the local side for the context associated with the socket.	*1
SHIM_LOC_PEER_PREF	o	o	Get or set the preferred locator on the remote side for the context associated with the socket.	*1
SHIM_LOC_LOCAL_RECV	o	o	Request for the destination locator of the received IP packet.	int
SHIM_LOC_PEER_RECV	o	o	Request for the source locator of the received IP packet.	int
SHIM_LOC_LOCAL_SEND	o	o	Request the use of specific locator as source locator of outgoing IP packets.	*2
SHIM_LOC_PEER_SEND	o	o	Request the use of specific locator as 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	Inform the shim layer of the timeout value for detecting failure.	int
SHIM_PATHEXPLORE	o	o	Specify behavior of path exploration and failure detection.	*4
SHIM_CONTEXT_DEFERRED_SETUP	o	o	Specify if the context setup can be deferred or not.	int

Table 1: Socket options for multihoming shim

---

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

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

2. With newly defined socket options for the shim 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).

```
                s1 s2      s3 s4
                 |  |       |  |
+----------------|--|-------|--|----------------+
|             +-------+   +-------+             |
| IPPROTO_TCP |  TCP  |   |  UDP  |             |
|             +-------+   +-------+             |
|                |   \     /   |                |
|                |    -----    |                |
|                |   /     \   |                |
|              +------+   +------+               |
|   IPPROTO_IP | IPv4 |   | IPv6 | IPPROTO_IPV6 |
|              +------+   +------+               |
|                  \         /             SOL_SOCKET
|          +--------\-------/--------+          |
| SOL_SHIM |          shim           |          |
|          +--------/-------\--------+          |
|                  /         \                  |
|              +------+   +------+               |
|              | IPv4 |   | IPv6 |               |
|              +------+   +------+               |
|                  |          |                  |
+------------------|----------|-----------------+
                   |          |
                 IPv4       IPv6
               Datagram   Datagram
```

Figure 2: System model of sockets API with shim layer

5.1. SHIM_ASSOCIATED

TOC

The SHIM_ASSOCIATED option can be used to check whether the socket is associated with any shim context or not.
This option is particularly meaningful in the case where the locator information of the received IP packet does not tell whether the identifier/locator adaptation is performed or not. Note that the EID pair and locator pair may be identical in some case.
This option can be specified by getsockopt(). Thus, the option is read-only and the result (0 or 1) is set in the option value (the fourth argument of getsockopt()).
The data type of the option value is an integer. The option value indicates the presence of shim context. A returned value 1 means that

the socket is associated with a shim context at the shim layer, while a return value 0 indicates that there is no shim context associated with the socket.
For example, the option can be used by the application as follows:

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

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

5.2. SHIM_DONTSHIM

The SHIM_DONTSHIM option can be used to request the shim layer to not apply the multihoming support for the communication established over the socket.
The data type of the option value is an integer. The option value indicates whether the multihoming shim support is deprecated or not. The option value is binary (0 or 1). By default, the value is set to 0, which means that the shim layer applies identifier/locator adaptation for the flow. In order to disable the socket option, the application should call setsockopt() with optval set to 0.
For example, the application may disable the socket option as follows:

```
         int optval;

         optval = 0;

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

For example, the application may 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

The SHIM_HOT_STANDBY option can be used to check if the shim layer uses hot-standby connection or not for the communication established over the socket. A hot-standby connection is based on 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, the option can be activated by the application as follows.

```
         int optval;

         optval = 1;

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

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

```
         int optval;
         int len;

         len = sizeof(optval);

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

5.4. SHIM_PATHEXPLORE

TOC

The application may specify this socket option to specify specify behavior of path exploration. Path exploration is a procedure to find an alternative locator pair when the host finds any problem with the current locator pair. The message used for finding an alternative locator pair is called the Probe message and it is sent per locator pair. The REAP specification defines the default values for Initial Probe Timeout and Initial Probe.
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 will be returned when there is no context associated with the socket.
For example, the parameters for the path exploration can be set as follows.

```
         struct shim6_pathexplore pe;

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

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

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

```
         struct shim6_pathexplore pe;
         int len;

         len = sizeof(pe);

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

5.5. SHIM_LOC_LOCAL_PREF

TOC

The SHIM_LOC_LOCAL_PREF option can be used to read or set preferred locator on local side within a given context. Hence this option is effective only when there is a shim context associated with the socket.
The data type of the option value is a pointer to the specific data structure which stores the locator information. 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.
The preferred locator can be set by setsockopt(). Verification of the locator shall be done by the shim layer before updating the preferred locator.
The preferred locator can be read by getsockopt().
An error ENOENT will be returned when there is no context associated with the socket.
An error EINVALIDLOCATOR will be returned when the validation of the specified locator failed.
For example, a preferred locator can be set as follows. It should be noted that some members of the shim_locator (lc_ifidx and lc_flags) are ignored in the write operation.

```
        struct shim_locator lc;
        struct in6_addr ip6;

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

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

```
        struct shim_locator lc;
        int len;

        len = sizeof(lc);

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

5.6. SHIM_LOC_PEER_PREF

TOC

The SHIM_LOC_PEER_PREF option can be used to read or set preferred locator on peer side within a given context. Hence this option is effective only when there is a shim context associated with the socket. The data type of the option value is a pointer to the specific data structure which stores the locator information. 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.
The preferred locator can be set by setsockopt(). The shim layer shall perform verification of the locator before updating the preferred locator.
The preferred locator can be read by getsockopt().
An error ENOENT will be returned when there is no context associated with the socket.
An error EINVALIDLOCATOR will be returned when the validation of the specified locator failed.

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

5.7. SHIM_LOC_LOCAL_RECV

TOC

The SHIM_LOC_LOCAL_RECV option can be used to request the shim layer to store the destination locator of the received IP packet in an ancillary data object which can be accessed by recvmsg(). Hence this option is effective only when there is a shim context associated with the socket.
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) for the procedure to access locator information stored in the ancillary data objects.
An error ENOENT will be returned when there is no context associated with the socket.
For example, the option can be activated by the application as follows:

```
         int optval;

         optval = 1;

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

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

```
         int optval;
         int len;

         len = sizeof(optval);

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

5.8. SHIM_LOC_PEER_RECV

TOC

The SHIM_LOC_PEER_RECV option can be used to request the shim layer to store the source locator of the received IP packet in an ancillary data

object which can be accessed by recvmsg(). Hence this option is effective only when there is a shim context associated with the socket.
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) for the procedure to access locator information stored in the ancillary data objects.
An error ENOENT will be returned when there is no context associated with the socket.
The usage of the option is same as that of SHIM_LOC_LOCAL_RECV option.

5.9. SHIM_LOC_LOCAL_SEND

The SHIM_LOC_LOCAL_SEND option can be used to request the shim layer to use specific locator for the source 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 option value is pointer to shim_locator data structure.
The local locator can be specified 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.
The local locator specified can be obtained by getsockopt(). The locator can be obtained from the option value.
An error ENOENT will be returned when there is no context associated with the socket.
An error EINVALIDLOCATOR when invalid locator is specified.
For example, a preferred local locator can be specified 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, a preferred local locator can be read 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 can be used to request the shim layer to use 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.
The remote locator can be specified 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.
The remote locator specified can be obtained by getsockopt(). The locator can be obtained from the option value.
An error ENOENT will be 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

The SHIM_LOCLIST_LOCAL option can be used to read or set the locator list associated with the local EID of the shim context associated with the socket. Hence this option is effective only when there is a shim context associated with the socket.
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.
The locator list can be read by getsockopt(). Note that the size of the buffer pointed by optval argument should be large enough to store an array of locator information. The number of the locator information is not known beforehand.
The locator list can be set by setsockopt(). The buffer pointed by optval argument should contain an array of locator list.
An error ENOENT will be returned when there is no context associated with the socket.
An error EINVALIDLOCATOR will be returned when the validation of the specified locator failed.
For example, a list of locators to be associated with the local EID can be specified 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, a list of locators that are associated with the local EID can be obtained 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 can be used to read or set the locator list associated with the peer EID of the shim context associated with the socket. Hence this option is effective only when there is a shim context associated with the socket.
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.
The locator list can be read by getsockopt(). Note that the size of the buffer pointed by optval argument should be large enough to store an array of locator information. The number of the locator information is not known beforehand.
The locator list can be set by setsockopt(). The buffer pointed by optval argument should contain an array of locator list.
An error ENOENT will be returned when there is no context associated with the socket.
An error EINVALIDLOCATOR will be returned when the validation of the specified locator failed.
The usage of the option is same as that of SHIM_LOCLIST_LOCAL.

5.13. SHIM_APP_TIMEOUT

TOC

The SHIM_APP_TIMEOUT option indicates timeout value for application to detect failure. Hence this option is effective only when there is a shim context associated with the socket.
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 [I-D.ietf-shim6-failure-detection] (Arkko, J. and I. Beijnum, “Failure Detection and Locator Pair Exploration Protocol for IPv6 Multihoming,” June 2008.)
If the timeout value specified is longer than the Send Timeout configured in the REAP component, the REAP Keepalive message should be suppressed.
An error ENOENT will be returned when there is no context associated with the socket.
For example, a specific timeout value can be configured by the application as follows:

```
         int optval;

         optval = 15; /* 15 seconds */

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

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

```
         int optval;
         int len;

         len = sizeof(optval);

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

5.14. SHIM_DEFERRED_CONTEXT_SETUP

TOC

The SHIM_DEFERRED_CONTEXT_SETUP option indicates how initiation of context setup is made in terms of timing (before or after) the initial communication flow. Deferred context means that the establishment of context does not put additional delay for an initial transaction. The data type for the option value is an integer. The option value should 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. However, it should be noted that deferred context setup may not be possible in some cases. For instance, an EID may be non-routable address (e.g., Host Identifier in HIP) and there is no way to transmit any IP packet unless there is a context providing the locators. In such a case, a context should be established prior to the communication. For example, the option can be disabled by the application as follows:

```
         int optval;

         optval = 0;

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

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

```
         int optval;
         int len;

         len = sizeof(optval);

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

5.15. Error Handling

TOC

If successful, getsockopt() and setsockopt() return 0; otherwise, the functions return -1 and set errno to indicate 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 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.), [I-D.ietf-shim6-hba] (Bagnulo, M., "Hash Based Addresses (HBA)," December 2007.).

6. Ancillary Data for Multihoming Shim

TOC

In this section, the definition and the usage of the ancillary data specific to multihoming shim are provided.
As defined in the Posix standard, sendmsg() and recvmsg() input a msghdr structure as their arguments. These system calls can handle control information along with data. Figure 3 (msghdr structure) shows the msghdr structure which is defined in <sys/socket.h>. The member msg_control holds a pointer to the buffer where the shim specific ancillary data objects can be stored in addition to other ancillary data objects.

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

Figure 3: msghdr structure

The buffer pointed by the member msg_control of the msghdr structure may contain locator information which is a single locator and it should be possible to process them with the existing macros defined in Posix and [RFC3542] (Stevens, W., Thomas, M., Nordmark, E., and T. Jinmei, “Advanced Sockets Application Program Interface (API) for IPv6,” May 2003.). Each cmsghdr{} should be followed by data which stores a single locator.
In case of non-connected socket, msg_name member stores the socket address of the peer which should be considered as an identifier rather than a locator. The locator of the peer node should be retrieved by SHIM_LOC_PEER_RECV as specified below.
Table 2 (Shim specific ancillary data) is a list of the shim specific ancillary data which can be used for recvmsg() or sendmsg(). In any case, SOL_SHIM must be set as cmsg_level.

cmsg_type	sendmsg()	recvmsg()	cmsg_data[]
SHIM_LOC_LOCAL_RECV		o	*1
SHIM_LOC_PEER_RECV		o	*1
SHIM_LOC_LOCAL_SEND	o		*1
SHIM_LOC_PEER_SEND	o		*1
SHIM_FEEDBACK	o		shim_feedback{}

Table 2: Shim specific ancillary data

*1: cmsg_data[] should include padding (if necessary) and a single sockaddr_in{}/sockaddr_in6{}.
It should be noted that the above ancillary data can only be handled by a UDP or a raw socket and not by a TCP socket. This is because there is

no one-to-one mapping between a single send/receive operation and a TCP segment being transmitted/received.

6.1. Get Locator Information 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.

6.2. Specify Locator Information 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.
In addition, the application can specify the outgoing interface by SHIM_IF_SEND ancillary data. The ancillary data should contain the interface identifier of the physical interface over which the application expects the packet to be transmitted.
Note that the effect is limited to the datagram transmitted by the sendmsg().
If the specified locator pair is verified, the shim layer overrides the locators of the IP packet.
An error EINVALIDLOCATOR will be returned when validation of the specified locator failed.

6.3. Notification from Application to Multihoming Shim

An application may provide feedbacks to the shim layer about the communication status. Such feedbacks are particularly useful for the shim 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 will be 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. Note that this specification does not specify the exact behavior of the shim layer when a feedback is given by an application.

7. Data Structures

TOC

In this section, data structures specifically defined for the multihoming shim layer are introduced. These data structures are either used as a parameter for setsockopt()/getsockopt() (as mentioned in Section 5 (Socket Options for Multihoming Shim Layer)) or as a parameter for ancillary data to be processed by sendmsg()/recvmsg() (as mentioned in Section 6 (Ancillary Data for Multihoming Shim)).

7.1. Placeholder for Locator Information

TOC

As defined in Section 5 (Socket Options for Multihoming Shim 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_ifidx;      /* interface index */
        uint8_t   lc_flags;      /* flags */
        uint8_t   lc_preference; /* preference value */
        uint8_t   lc_addr[16];   /* address data */
};
```

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_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_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_preference Indicates a preference of the locator. The preference is represented by an integer.

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

7.2. Path Exploration Parameter

TOC

As defined in Section 5 (Socket Options for Multihoming Shim 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 5 (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 5: path explore structure

pe_probenum Indicates the number of initial probe messages to be sent. Default value of this parameter should follow what is specified in [I-D.ietf-shim6-failure-detection] (Arkko, J. and I. Beijnum, “Failure Detection and Locator Pair Exploration Protocol for IPv6 Multihoming,” June 2008.).

pe_keepaliveto Indicates timeout value for detecting a failure when the host does not receive any packets for a certain period of time while there is outbound traffic. When the timer expires, path exploration procedure will be carried out by sending a REAP Probe message. Default value of this parameter should follow what is specified in [I-D.ietf-shim6-failure-detection] (Arkko, J. and I. Beijnum, “Failure Detection and Locator Pair Exploration Protocol for IPv6 Multihoming,” June 2008.).

pe_initprobeto Indicates retransmission timer of REAP Probe message in milliseconds. Note that this timer is applied before exponential back-off is started. A REAP Probe message for the same locator pair may be retransmitted. Default value of this parameter should follow what is specified in [I-D.ietf-shim6-failure-detection] (Arkko, J. and I. Beijnum, “Failure Detection and Locator Pair Exploration Protocol for IPv6 Multihoming,” June 2008.).

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

7.3. Feedback Information

As mentioned in Section 6.3 (Notification from Application to Multihoming Shim), applications can inform the shim 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 6 (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 6: 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 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.

8. Implications for Existing Socket API Extensions

TOC

Some of the socket options defined in this document are overlapping with existing sockets API and care should be taken for the usage not to confuse with the overlapping features.
The socket options for requesting specific locators to be used for a given transaction (SHIM_LOC_LOCAL_PREF and SHIM_LOC_PEER_PREF) are semantically similar to the existing sockets API (IPV6_PKTINFO). The socket options for obtaining the locator information from the received IP packet (SHIM_LOC_LOCAL_RECV and SHIM_LOC_PEER_RECV) are semantically similar to the existing sockets API (IP_RECVDSTADDR and IPV6_PKTINFO).
In IPv4, application can obtain the destination IP address of the received IP packet (IP_RECVDSTADDR). If the shim layer performs identifier/locator adaptation for the received packet, the destination EID should be stored in the ancillary data (IP_RECVDSTADDR).
In IPv6, [RFC3542] (Stevens, W., Thomas, M., Nordmark, E., and T. Jinmei, “Advanced Sockets Application Program Interface (API) for IPv6,” May 2003.) defines that IPV6_PKTINFO can be used to specify source IPv6 address and the outgoing interface for outgoing packets, and retrieve destination IPv6 address and receiving interface for incoming packets. This information is stored in ancillary data being IPV6_PKTINFO specified as cmsg_type. Existing sockets API should

continue to work above the shim layer, that is, the IP addresses handled in IPV6_PKTINFO should be EIDs, not the locators.
Baseline is that the above existing sockets API (IP_RECVDSTADDR and IPV6_PKTINFO) is assumed to work above the multihoming shim layer. In other words, the IP addresses those socket options deal with are EIDs rather than locators.

---

9. Resolving Conflicts with Preference Values

TOC

Since the multihoming shim API allows application to specify preference value for the context which is associated with the socket instance, there may be a conflict with preference values specified by different applications. For instance, application A and B may establish communication over the same EID pair while both applications have different preference in their choice of local locator.
SHIM6 supports a notion of context forking in which a context is split when there is a conflict with preference values specified by multiple applications. Thus, context forking can simply resolve the conflicting situation which may be caused by the use of socket options for multihoming shim layer.

9.1. Implicit Forking

TOC

Socket options defined in Section 5 (Socket Options for Multihoming Shim Layer) may cause conflicting situation when the target context is shared by multiple applications. In such case, socket handler and the multihoming shim layer should react as follows; socket handler should inform the shim 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 multihoming shim layer during the lifetime of associated socket instance. When the socket is closed, the multihoming shim layer SHOULD delete associated context. In this way, garbage collection can be carried out to cleanup unused forked contexts. Upon garbage collection, every forked context SHOULD be checked if there is no socket (process) associated with the context. If there is none, the forked context should be deleted. When a forked context is torn down, SHIM6 should notify the peer about the deletion of forked context.
As opposed to socket options, context forking MUST NOT be triggered by any use of ancillary data that is specific to multihoming shim as defined in Section 6 (Ancillary Data for Multihoming Shim).

10. Discussion

TOC

In this section, open issues are introduced.

10.1. Naming at Socket Layer

TOC

The getsockname() and getpeername() system calls are used to obtain the 'name' of an endpoint which is actually a pair of IP address and port number assigned to a given socket. getsockname() is used when an application wants to obtain the local IP address and port number assigned for a given socket instance. getpeername() is used when an application obtains the remote IP address and port number.
The above is based on a traditional system model of the sockets API where an IP address is expected to play both the role of identifier and the role of locator.
In a system model where a shim layer exists inside the IP layer, both getsockname() and getpeername() deal with identifiers, namely EIDs. In this sense, the shim layer serves to (1) hide locators and (2) provide access to the identifier for the application over the legacy socket APIs.

10.2. Additional Requirements from Applications

TOC

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

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

10.3. Issues of Header Conversion among Different Address Family

TOC

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

conversion such as SIIT[RFC2765] (Nordmark, E., “Stateless IP/ICMP Translation Algorithm (SIIT),” February 2000.).
As addressed in SIIT specification, some of the features (IPv6 routing headers, hop-by-hop extension headers, or destination headers) from IPv6 are not convertible to IPv4. In addition, notion of source routing is not exactly the same in IPv4 and IPv6. Hence, there is certain limitation in protocol conversion between IPv4 and IPv6.
The question is how should the shim layer behave when it is face with limitation problem of protocol conversion. Should we introduce new error something like ENOSUITABLELOCATOR ?

10.4. Handling of Unknown Locator Provided by Application

TOC

There might be a case where application provides the shim layer new locator with the SHIM_LOC_*_PREF socket options or SHIM_LOC_*_SEND ancillary data. Then there is a question how should the shim layer treat the new locator informed by the application.
In principle, locator information are exchanged by the shim protocol. However, there might be a case where application acquires information about the locator and prefers to use it for its communication.

---

11. Changes

TOC

11.1. Changes from version 00 to version 01

TOC

The followings are 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.

11.2. Changes from version 01 to version 02

TOC

The followings are 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).

11.3. Changes from version 02 to version 03

TOC

The followings are changes from version 02 to version 03:

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

11.4. Changes from version 03 to version 04

TOC

The followings are changes from version 03 to version 04:

- *Updated reference.
- *Correct typo and grammatical errors.

11.5. Changes from version 04 to version 05

TOC

The followings are 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.

11.6. Changes from version 05 to version 06

TOC

The followings are changes from version 04 to version 05:

- *Updated references.

11.7. Changes from version 06 to version 07

TOC

The followings are changes from version 06 to version 07:

- *Resolved editorial issues.

11.8. Changes from version 07 to version 08

TOC

No changes are made except for updates of the references.

12. IANA Considerations

TOC

This document contains no IANA consideration.

13. Security Considerations

TOC

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

TOC

14. Conclusion

In this document, the Application Program Interface (API) for multihoming shim layer is specified. The sockets API allows applications to have additional control of the locator management and interface to the REAP mechanism inside the multihoming shim layer. Socket options for multihoming shim layer can be used by getsockopt() and/or setsockopt() system calls. Besides, applications can use some ancillary data that are specific to multihoming shim layer to get locator from received packet or to set locator for outgoing packet. From an architectural point of view, the sockets API provides extends the existing sockets API framework in the face of ID/Locator separation. With regard to API that relate to IP address management, it is assured that existing sockets API continue to work above the shim layer dealing with identifiers, while multihoming shim API deals with locators.

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.

16. References

TOC

16.1. Normative References

TOC

[I-D.ietf-shim6-failure-detection]	Arkko, J. and I. Beijnum, “Failure Detection and Locator Pair Exploration Protocol for IPv6 Multihoming,” draft-ietf-shim6-failure-detection-13 (work in progress), June 2008 (TXT, PDF).
[I-D.ietf-shim6-proto]	Bagnulo, M. and E. Nordmark, “Level 3 multihoming shim protocol,” draft-ietf-shim6-proto-12 (work in progress), February 2009 (TXT).
[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).

16.2. Informative References

TOC

[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-hba]	Bagnulo, M., “Hash Based Addresses (HBA),” draft-ietf-shim6-hba-05 (work in progress), December 2007 (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).

Appendix A. Context Forking

TOC

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 initiator.
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 implementation specific issue. However, on the responder side, there is a question on how the outbound packet can be multiplexed by the shim layer. Since 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 some other information and associate a given packet flow with specific context.

Figure 7 (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 the same ULID pair. The first context pair (Ctx1) is established for the transaction of App1. Given the requests from App2, the shim 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 7 (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.

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

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

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

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

Figure 7: context forking

To overcome the problem mentioned above, there are some solutions.

One viable approach is to let the system implicitly maintain an association between the socket and the associated context by keeping the record of inbound packet processing. That is, the system stores the information about the context on which the inbound packet flow was demultiplexed. The information comprises the ULID pair and FII of the context and is stored in the socket instance. Later, the system can use the information to identify the associated context in outbound packet processing. This approach should be feasible as far as there is bi-directional user traffic.

Another viable approach is to extend SHIM6 protocol by adding capability of exchanging additional information to identify the packet flow from others which needs to be handled by a newly forked context. The information exchange can be done during the context establishment. The initiator appends 5 tuple of the packet flow to be handled by the newly forked context. Note that the additional information provided by the 5 tuple are source and destination port numbers and upper layer protocol. The information is later used by the shim layer to multiplex the outbound packet flow on the responder side.

The socket options for multihoming shim can be used by the application to trigger the context forking in implicit manner. The peer becomes an initiator in the establishment of the forked context. Once the forked context is established between the peers, application on each end can influence the preference on context by utilizing the multihoming shim API.

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.sugimoto@ericsson.com