Host Identity Protocol	M. Komu
Internet-Draft	Helsinki Institute for Information
Intended status: Experimental	Technology
Expires: January 14, 2010	Henderson
	The Boeing Company
	July 13, 2009

Basic Socket Interface Extensions for Host Identity Protocol (HIP)
draft-ietf-hip-native-api-07

Status of this Memo

Copyright Notice

Abstract

This document defines extensions to the current sockets API for the Host Identity Protocol (HIP). The extensions focus on the use of public-key based identifiers discovered via DNS resolution, but define also interfaces for manual bindings between HITs and locators. With the extensions, the application can also support more relaxed security models where the communication can be non-HIP based, according to local policies. The extensions in document are experimental and provide basic tools for further experimentation with policies.

Table of Contents

---

1. Introduction

TOC

This document defines C-based sockets Application Programming Interface (API) extensions for handling HIP-based identifiers explicitly in HIP-aware applications. It is up to the applications, or high-level programming languages or libraries, to manage the identifiers. The extensions in this document are mainly related to the use case in which a DNS resolution step has occurred prior to the creation of a new socket, and assumes that the system has cached or is otherwise able to resolve identifiers to locators (IP addresses). The DNS extensions for HIP are described in [RFC5205] (Nikander, P. and J. Laganier, “Host

Identity Protocol (HIP) Domain Name System (DNS) Extensions," April 2008.). The extensions also cover the case in which an application may want to explicitly provide suggested locators with the identifiers, including supporting the opportunistic case in which the system does not know the peer host identity.

The Host Identity Protocol (HIP) [RFC4423] (Moskowitz, R. and P. Nikander, "Host Identity Protocol (HIP) Architecture," May 2006.) proposes a new cryptographic namespace by separating the roles of end-point identifiers and locators by introducing a new namespace to the TCP/IP stack. SHIM6 [I-D.ietf-shim6-proto] (Nordmark, E. and M. Bagnulo, "Shim6: Level 3 Multihoming Shim Protocol for IPv6," February 2009.) is another protocol based on identity-locator split. The APIs specified in this document are specific to HIP, but have been designed as much as possible so as not to preclude its use with other protocols. The use of these APIs with other protocols is, nevertheless, for further study.

The APIs in this document are based on IPv6 addresses with the ORCHID prefix [RFC4843] (Nikander, P., Laganier, J., and F. Dupont, "An IPv6 Prefix for Overlay Routable Cryptographic Hash Identifiers (ORCHID)," April 2007.). ORCHIDs are derived from Host Identifiers using a hash and fitting the result into an IPv6 address. Such addresses are called Host Identity Tags (HITs) and they can be distinguished from other IPv6 addresses with the ORCHID prefix.

Applications can observe the HIP layer and its identifiers in the networking stacks with varying degrees of visibility. [RFC5338] (Henderson, T., Nikander, P., and M. Komu, "Using the Host Identity Protocol with Legacy Applications," September 2008.) discusses the lowest levels of visibility in which applications are completely unaware of the underlying HIP layer. Such HIP-unaware applications in some circumstances use HIP-based identifiers, such as LSIs or HITs, instead of IPv4 or IPv6 addresses and cannot observe the identifier-locator bindings.

This document specifies extensions to [RFC3493] (Gilligan, R., Thomson, S., Bound, J., McCann, J., and W. Stevens, "Basic Socket Interface Extensions for IPv6," February 2003.) to define a new socket address family, AF_HIP. Similarly to other address families, AF_HIP can used as an alias for PF_HIP. The extensions also describe a new socket address structure for sockets using HITs explicitly and describe how the socket calls in [RFC3493] (Gilligan, R., Thomson, S., Bound, J., McCann, J., and W. Stevens, "Basic Socket Interface Extensions for IPv6," February 2003.) are adapted or extended as a result.

Some applications may accept incoming communications from any identifier. Other applications may initiate outgoing communications without the knowledge of the peer identifier in Opportunistic Mode (section 4.1.6 in [RFC5201] (Moskowitz, R., Nikander, P., Jokela, P., and T. Henderson, "Host Identity Protocol," April 2008.)) by just relying on a peer locator. This document describes how to address both situations using "wildcards" as described later in this document.

There are two related API documents. Multihoming and explicit locator-handling related APIs are defined in [I-D.ietf-shim6-multihome-shim-api] (Komu, M., Bagnulo, M., Slavov, K., and S. Sugimoto, “Socket Application Program Interface (API) for Multihoming Shim,” February 2010.). IPsec related policy attributes and channel bindings APIs are defined in [I-D.ietf-btns-c-api] (Richardson, M., Williams, N., Komu, M., and S. Tarkoma, “C-Bindings for IPsec Application Programming Interfaces,” March 2009.). Most of the extensions defined in this document can be used independently of the two mentioned API documents.

The identity-locator split introduced by HIP introduces some policy related challenges with datagram oriented sockets, opportunistic mode, and manual bindings between HITs and locators. The extensions in this document are of an experimental nature and provide basic tools for experimenting with policies. Policy related issues are left for further experimentation.

To recap, the extensions in this document have three goals. The first goal is to allow HIP-aware applications to open sockets to other hosts based on the HITs alone, presuming that the underlying system can resolve the HITs to addresses used for initial contact. The second goal is that applications can explicitly initiate communications with unknown peer identifiers. The third goal is to illustrate how HIP-aware applications can use the SHIM API [I-D.ietf-shim6-multihome-shim-api] (Komu, M., Bagnulo, M., Slavov, K., and S. Sugimoto, “Socket Application Program Interface (API) for Multihoming Shim,” February 2010.) to manually map locators to HITs.

---

2. Terminology

TOC

The terms used in this document are summarized in Table 1.

Term	**Explanation**
HIP	Host Identity Protocol
HIT	Host Identity Tag, a 100-bit hash of a public key with a 28 bit prefix
LSI	Local Scope Identifier, a local, 32-bit descriptor for a given public key.
Locator	Routable IPv4 or IPv6 address used at the lower layers

Table 1

---

3. API Overview

TOC

This section provides an overview of how the API can be used. First, the case in which a resolver is involved in name resolution is described, and then the case in which no resolver is involved is described.

3.1. Interaction with the Resolver

TOC

Before an application can establish network communications with the entity named by a given FQDN or relative host name, the application must translate the name into the corresponding identifier(s). DNS-based hostname-to-identifier translation is illustrated in Figure 1. The application calls the resolver in step (a) to resolve an FQDN to HIT(s). The resolver queries the DNS in step (b) to map the FQDN to a host identifier and locator (A and AAAA records). It should be noticed that the FQDN may map to multiple host identifiers and locators, and this step may involve multiple DNS transactions, including queries for A, AAAA, HI and possibly other resource records. The DNS server responds with a list of HIP resource records in step (c). Optionally in step (d), the resolver caches the HIT to locator mapping with the HIP module. The resolver converts the HIP records to HITs and returns the HITs to the application contained in HIP socket address structures in step (e). Depending on the parameters for the resolver call, the resolver may return also other socket address structures to the application. Finally, the application receives the socket address structure(s) from the resolver and uses them in socket calls such as connect() in step (f).

```
                                       +----------+
                                       |          |
                                       |   DNS    |
                                       |          |
                                       +----------+
                                           ^  |
                            b. QNAME=FQDN  |  | c. HIP and
                                           |  |    A/AAAA
                                           |  v    RR(s)
+-------------+ a. getaddrinfo(<FQDN>)  +----------+
|             |------------------------>|          |
| Application |                         | Resolver |
|             |<------------------------|          |
+-------------+        e. <HITs>        +----------+
        |                                    |
        |                                    | d. HIP and
        | f. connect(<HIT>)                  |    A/AAAA
        |    or any other socket call        |    RR(s)
        v                                    v
 +----------+                           +----------+
 |          |                           |          |
 |  TCP/IP  |                           |   HIP    |
 |  Stack   |                           |          |
 +----------+                           +----------+
```

Figure 1

In practice, the resolver functionality can be implemented in different ways. For example, it may be implemented in existing resolver libraries or as a HIP-aware interposing agent.

3.2. Interaction without a Resolver

TOC

The extensions in this document focus on the use of the resolver to map host names to HITs and locators in HIP-aware applications. The resolver may implicitly associate a HIT with the corresponding locator(s) by communicating the HIT-to-IP mapping to the HIP daemon. However, it is possible that an application operates directly on a peer HIT without interacting with the resolver. In such a case, the application may resort to the system to map the peer HIT to an IP address. Alternatively, the application can explicitly map the HIT to an IP address using socket options as specified in Section 4.6 (Explicit Handling of Locators). Full support for all of the extensions defined in this draft requires a number of shim socket options

[I-D.ietf-shim6-multihome-shim-api] (Komu, M., Bagnulo, M., Slavov, K., and S. Sugimoto, “Socket Application Program Interface (API) for Multihoming Shim,” February 2010.) to be implemented by the system.

4. API Syntax and Semantics

TOC

In this section, we describe the native HIP APIs using the syntax of the C programming language. We limit the description to the interfaces and data structures that are either modified or completely new, because the native HIP APIs are otherwise identical to the sockets API [POSIX] (Institute of Electrical and Electronics Engineers, “IEEE Std. 1003.1-2001 Standard for Information Technology - Portable Operating System Interface (POSIX),” Dec 2001.).

4.1. Socket Family and Address Structure Extensions

TOC

The sockets API extensions define a new protocol family, PF_HIP, and a new address family, AF_HIP. The AF_HIP and PF_HIP are aliases to each other. These definition shall be defined as a result of including <sys/socket.h>.
The use of the PF_HIP constant is mandatory with the socket() function when an application uses the native HIP APIs. The application gives the PF_HIP constant as the first argument (domain) to the socket() function. The system returns a positive integer representing a socket descriptor when the system supports HIP. Otherwise, the system returns -1 and sets errno to EAFNOSUPPORT.
Figure 2 shows socket address structure for HIP.

```
#include <netinet/in.h>

typedef struct in6_addr hip_hit_t;

struct sockaddr_hip {
          sa_family_t    ship_family;
          in_port_t      ship_port;
          uint32_t       ship_pad;
          uint64_t       ship_flags;
          hip_hit_t      ship_hit;
          uint8_t        ship_reserved[16];
};
```

Figure 2

Figure 2 is in in 4.3BSD format. The family of the socket, ship_family, is set to AF_HIP. The port number ship_port is two octets in network byte order and the ship_hit is 16 octets in network byte order. An implementation may have extra member(s) in this structure.

The application usually sets the ship_hit field using the resolver. However, the application can use three special constants to set a wildcard value manually into the ship_hit field. The constants are HIP_HIT_ANY, HIP_HIT_ANY_PUB, HIP_HIT_ANY_TMP and HIP_ENDPOINT_ANY. The first three equal to a HIT value associated with a wildcard HIT of any type, public type, or anonymous type. The fourth constant, HIP_ENDPOINT_ANY, denotes that the application accepts both HIT and any other type of addresses. The HIP_HIT_ANY denotes that the application accepts any type of HIT. The anonymous identifiers refer to the use of anonymous identifiers as specified in [RFC4423] (Moskowitz, R. and P. Nikander, “Host Identity Protocol (HIP) Architecture,” May 2006.). The system may designate anonymous identifiers as meta data associated with a HIT depending on whether it has been published or not. However, there is no difference in the classes of HITs from the protocol perspective.

The application can use the HIP_HIT_ANY_* and HIP_ENDPOINT_ANY constants to accept incoming communications to all of the HITs of the local host. Incoming communications refers here to functions such as bind(), recvfrom() and recvmsg(). The HIP_HIT_* constants are similar to the sockets API constants INADDR_ANY and IN6ADDR_ANY_INIT, but they are applicable to HITs only. After initial contact with the peer, the application can discover the local and peer HITs using getsockname() and getpeername() calls in the context of connection oriented sockets. The difference between the use of the HIP_HIT_* and HIP_ENDPOINT_ANY constants here is that the former allows only HIP-based communications but the latter also allows communications without HIP.

The application also uses the HIP_HIT_ANY constant in ship_hit field to establish outgoing communications in Opportunistic mode [RFC5201] (Moskowitz, R., Nikander, P., Jokela, P., and T. Henderson, “Host Identity Protocol,” April 2008.), i.e., when the application knows the remote peer locator but not the HIT. Outgoing communications refers here to the use of functions such as connect(), sendto() and sendmsg(). However, the application should first associate the socket with at least one IP address of the peer using SHIM_LOCLIST_PEER_PREF socket option. The use of the HIP_HIT_ANY constant guarantees that the communications will be based on HIP or none at all.

The use of HIP_ENDPOINT_ANY constant in the context of outgoing communications is left for further experimentation in the context of opportunistic mode. It can result in a data flow with or without HIP.

Some applications rely on system level access control, either implicit or explicit (such as accept_filter() function found on BSD-based systems), but such discussion is out of scope. Other applications implement access control themselves by using the HITs. In such a case, the application can compare two HITs contained in the ship_hit field using memcmp() or similar function. It should be noted that different connection attempts between the same two hosts can result in different HITs because a host is allowed to have multiple HITs.

4.2. Extensions to Resolver Data Structures

TOC

The HIP APIs introduce a new address family, AF_HIP, that HIP aware applications can use to control the address type returned from getaddrinfo() function [RFC3493] (Gilligan, R., Thomson, S., Bound, J., McCann, J., and W. Stevens, “Basic Socket Interface Extensions for IPv6,” February 2003.). The getaddrinfo() function uses a data structure called addrinfo in its "hints" and "res" argument which are described in more detail in the next section. The addrinfo data structure is illustrated in Figure 3.

```
#include <netdb.h>

struct addrinfo {
    int       ai_flags;          /* e.g. AI_EXTFLAGS */
    int       ai_family;         /* e.g. AF_HIP */
    int       ai_socktype;       /* e.g. SOCK_STREAM */
    int       ai_protocol;       /* 0 or IPPROTO_HIP */
    socklen_t ai_addrlen;        /* size of *ai_addr  */
    struct    sockaddr *ai_addr; /* sockaddr_hip */
    char     *ai_canonname;     /* canon. name of the host */
    struct    addrinfo *ai_next; /* next endpoint */
    int       ai_eflags;         /* RFC5014 extension */
};
```

Figure 3

An application resolving with the ai_family field set to AF_UNSPEC in the hints argument may receive any kind of socket address structures, including sockaddr_hip. When the application wants to receive only HITs contained in sockaddr_hip structures, it should set the ai_family field to AF_HIP. Otherwise, the resolver does not return any sockaddr_hip structures. The resolver returns EAI_FAMILY when AF_HIP is not supported.
The resolver ignores the AI_PASSIVE flag when the application sets the family in hints to AF_HIP.

The system may have a HIP-aware interposing DNS agent as described in section 3.2 in [RFC5338] (Henderson, T., Nikander, P., and M. Komu, “Using the Host Identity Protocol with Legacy Applications,” September 2008.). In such a case, the DNS agent may, according to local policy, return transparently LSIs or HITs in sockaddr_in and sockaddr_in6 structures when available. A HIP-aware application can override this local policy in two ways. First, the application can set the family to AF_HIP in the hints argument of getaddrinfo() when it requests only sockaddr_hip structures. Second, the application can set AI_EXTFLAGS and AI_NO_HIT flags to prevent the resolver from returning HITs in any kind of data structures.
When getaddrinfo() returns resolved outputs the results to res argument, it sets the family to AF_HIP when the related structure is sockaddr_hip.

4.2.1. Resolver Usage

TOC

A HIP-aware application creates the sockaddr_hip structures manually or obtains them from the resolver. The explicit configuration of locators is described in [I-D.ietf-shim6-multihome-shim-api] (Komu, M., Bagnulo, M., Slavov, K., and S. Sugimoto, “Socket Application Program Interface (API) for Multihoming Shim,” February 2010.). This document defines "automated" resolver extensions for getaddrinfo() resolver [RFC3493] (Gilligan, R., Thomson, S., Bound, J., McCann, J., and W. Stevens, “Basic Socket Interface Extensions for IPv6,” February 2003.).

```
#include <netdb.h>

int getaddrinfo(const char *nodename,
                const char *servname,
                const struct addrinfo *hints,
                struct addrinfo **res)
void free_addrinfo(struct addrinfo *res)
```

Figure 4

As described in [RFC3493] (Gilligan, R., Thomson, S., Bound, J., McCann, J., and W. Stevens, “Basic Socket Interface Extensions for IPv6,” February 2003.), the getaddrinfo function takes the nodename, servname, and hints as its input arguments. It places the result of the query into the res output argument. The return value is zero on success, or a non-zero error value on error. The nodename argument

specifies the host name to be resolved; a NULL argument denotes the HITs of the local host. The servname parameter declares the port number to be set in the socket addresses in the res output argument. Both the nodename and servname cannot be NULL at the same time.
The input argument "hints" acts like a filter that defines the attributes required from the resolved endpoints. A NULL hints argument indicates that any kind of endpoints are acceptable.
The output argument "res" is dynamically allocated by the resolver. The application frees the res argument with the free_addrinfo function. The res argument contains a linked list of the resolved endpoints. The linked list contains sockaddr_hip structures only when the input argument has the family set to AF_HIP. When the family is zero, the list contains sockaddr_hip structures before sockaddr_in and sockaddr_in6 structures.
Resolver can return a HIT which maps to multiple locators. The resolver may cache the locator mappings with the HIP module. The HIP module manages the multiple locators according to system policies of the host. The multihoming document [I-D.ietf-shim6-multihome-shim-api] (Komu, M., Bagnulo, M., Slavov, K., and S. Sugimoto, “Socket Application Program Interface (API) for Multihoming Shim,” February 2010.) describes how an application can override system default policies.
It should be noted that the application can configure the HIT explicitly without setting the locator or the resolver can fail to resolve any locator. In this scenario, the application relies on the system to map the HIT to an IP address. When the system fails to provide the mapping, it returns -1 in the called sockets API function to the application and sets errno to EADDRNOTAVAIL.

4.3. The Use of getsockname and getpeername Functions

TOC

The application usually discovers the local or peer HITs from the sockaddr_hip structures returned by getaddrinfo(). However, the sockaddr_hip structure does not contain a HIT when the application uses the HIP_HIT_ANY_* constants. In such a case, the application discovers the local and peer HITs using the getsockname() and getpeername() functions. The functions return sockaddr_hip structures when the family of the socket is AF_HIP.

4.4. Selection of Source HIT Type

TOC

The Socket API for Source Address Selection [RFC5014] (Nordmark, E., Chakrabarti, S., and J. Laganier, “IPv6 Socket API for Source Address Selection,” September 2007.) defines socket options to allow applications to influence source address selection mechanisms. In some cases, HIP-aware applications may want to influence source HIT selection; in particular, whether an outbound connection should use a published or anonymous HIT. Similar to IPV6_ADDR_PREFERENCES defined in RFC 5014, the following socket option HIT_PREFERENCES is defined for HIP-based sockets. This socket option can be used with setsockopt() and getsockopt() calls to set and get the HIT selection preferences affecting a HIP-enabled socket. The socket option value (optval) is a 32-bit unsigned integer argument. The argument consists of a number of flags where each flag indicates an address selection preference that modifies one of the rules in the default HIT selection; these flags are shown in Table 2.

Socket Option	Purpose
HIP_PREFER_SRC_HIT_TMP	Prefer an anonymous HIT
HIP_PREFER_SRC_HIT_PUBLIC	Prefer a public HIT

Table 2

If the system is unable to assign the type of HIT that is requested, at HIT selection time, the socket call (connect (), sendto(), or sendmsg()) will fail and errno will be set to EINVAL. If the application tries to set both of the above flags for the same socket, this also results in the error EINVAL.

4.5. Verification of Source HIT Type

TOC

An application that uses the HIP_ENDPOINT_ANY constant may want to check whether the actual communications was based on HIP or not. Also, the application may want to verify whether a local HIT is public or anonymous. The application accomplishes these using a new function called sockaddr_is_srcaddr() which is illustrated in Figure 5.

```
#include <netinet/in.h>

short sockaddr_is_srcaddr(struct sockaddr *srcaddr,
                          uint64_t flags);
```

Figure 5

The sockaddr_is_srcaddr() function operates in the same way as inet6_is_srcaddr() function [RFC5014] (Nordmark, E., Chakrabarti, S., and J. Laganier, “IPv6 Socket API for Source Address Selection,” September 2007.) which can be used to verify the type of an address belonging to the local host. The difference is that the sockaddr_is_srcaddr() function handles sockaddr_hip structures in addition to sockaddr_in6, and possibly some other socket structures in further extensions. The flags argument is also 64 bit instead of 32 bits because new function handles the same flags as defined in [RFC5014] (Nordmark, E., Chakrabarti, S., and J. Laganier, “IPv6 Socket API for Source Address Selection,” September 2007.) in addition to two HIP-specific flags, HIP_PREFER_SRC_HIT_TMP and HIP_PREFER_SRC_HIT_PUBLIC. With these two flags, the application can distinguish anonymous HITs from public HITs.
When given an AF_INET6 socket, sockaddr_is_srcaddr() behaves as inet6_is_srcaddr() function as described in [RFC5014] (Nordmark, E., Chakrabarti, S., and J. Laganier, “IPv6 Socket API for Source Address Selection,” September 2007.). With AF_HIP socket, the function returns 1 when the HIT contained in the socket address structure corresponds to a valid HIT of the local host and the HIT satisfies the given flags. The function returns -1 when the HIT does not belong to the local host or the flags are not valid. The function returns 0 when the preference flags are valid but the HIT does not match the given flags.

4.6. Explicit Handling of Locators

TOC

The system resolver, or the HIP module, maps HITs to locators implicitly. However, some applications may want to specify initial locator mappings explicitly. In such a case, the application first creates a socket with AF_HIP as the domain argument. Second, the application may get or set locator information with one of the following shim socket options as defined in the multihoming extensions in [I-D.ietf-shim6-multihome-shim-api] (Komu, M., Bagnulo, M., Slavov, K., and S. Sugimoto, “Socket Application Program Interface (API) for Multihoming Shim,” February 2010.). The related socket options are summarized briefly in Table 3.

optname	description
SHIM_LOC_LOCAL_PREF	Get or set the preferred locator on the local side for the context associated with the socket.
SHIM_LOC_PEER_PREF	Get or set the preferred locator on the remote side for the context associated with the socket.

SHIM_LOCLIST_LOCAL	Get or set a list of locators associated with the local EID.
SHIM_LOCLIST_PEER	Get or set a list of locators associated with the peer's EID.
SHIM_LOC_LOCAL_SEND	Set or get the default source locator of outgoing IP packets.
SHIM_LOC_PEER_SEND	Set or get the default destination locator of outgoing IP packets.

Table 3

As an example of locator mappings, a connection-oriented application creates a HIP-based socket and sets the SHIM_LOCLIST_PEER socket option to the socket. The HIP module uses the first address contained in the option if multiple are provided. If the application provides one or more addresses in the SHIM_LOCLIST_PEER setsockopt call, the system should not connect to the host via another destination address, in case the application intends to restrict the range of addresses permissible as a policy choice. The application can override the default peer locator by setting the SHIM_LOC_PEER_PREF socket option if necessary. Finally, the application provides a specific HIT in the ship_hit field of the sockaddr_hip in the connect() system call. If the system cannot reach the HIT at one of the addresses provided, the outbound socket API functions (connect, sendmsg, etc.) return -1 and set errno to EINVALIDLOCATOR.

Applications may also choose to associate local addresses with sockets. The procedures specified in [I-D.ietf-shim6-multihome-shim-api] (Komu, M., Bagnulo, M., Slavov, K., and S. Sugimoto, “Socket Application Program Interface (API) for Multihoming Shim,” February 2010.) are followed in this case.

Another use case is to use the opportunistic mode when the destination HIT is specified as a wildcard. The application sets one or more destination addresses using the SHIM_LOCLIST_PEER socket option as described above and then calls connect() with the wildcard HIT. The connect() call returns -1 and sets errno to EADDRNOTAVAIL when the application connects to a wildcard without specifying any destination address.

Applications using datagram-oriented sockets can use ancillary data to control the locators. This described in detail in [I-D.ietf-shim6-multihome-shim-api] (Komu, M., Bagnulo, M., Slavov, K., and S. Sugimoto, “Socket Application Program Interface (API) for Multihoming Shim,” February 2010.).

5. Summary of New Definitions

Table 4 summarizes the new constants and structures defined in this document.

Header	Definition
<sys/socket.h>	AF_HIP
<sys/socket.h>	PF_HIP
<netinet/in.h>	IPPROTO_HIP
<netinet/hip.h>	HIP_HIT_ANY
<netinet/hip.h>	HIP_HIT_ANY_PUB
<netinet/hip.h>	HIP_HIT_ANY_TMP
<netinet/hip.h>	HIP_ENDPOINT_ANY
<netinet/hip.h>	HIP_HIT_PREFERENCES
<netinet/hip.h>	hip_hit_t
<netdb.h>	AI_NO_HIT
<netinet/hip.h>	sockaddr_hip
<netinet/hip.h>	sockaddr_is_srcaddr

Table 4

6. IANA Considerations

TOC

No IANA considerations.

7. Security Considerations

TOC

No security considerations currently.

TOC

8. Contributors

Thanks for Jukka Ylitalo and Pekka Nikander for their original contribution, time and effort to the native HIP APIs. Thanks for Yoshifuji Hideaki for his contributions to this document.

---

9. Acknowledgements

TOC

Kristian Slavov, Julien Laganier, Jaakko Kangasharju, Mika Kousa, Jan Melen, Andrew McGregor, Sasu Tarkoma, Lars Eggert, Joe Touch, Antti Jarvinen, Anthony Joseph, Teemu Koponen, Jari Arkko, Ari Keranen, Juha-Matti Tapio, Shinta Sugimoto, Philip Matthews, Joakim Koskela, Jeff Ahrenholz and Gonzalo Camarillo have also provided valuable ideas or feedback. Thanks also for the APPS area folks, including Stephane Bortzmeyer, Chris Newman, Tony Finch, "der Mouse" and Keith Moore.

---

10. Normative References

TOC

[I-D.ietf-btns-c-api]	Richardson, M., Williams, N., Komu, M., and S. Tarkoma, “C-Bindings for IPsec Application Programming Interfaces,” draft-ietf-btns-c-api-04 (work in progress), March 2009 (TXT).
[I-D.ietf-shim6-multihome-shim-api]	Komu, M., Bagnulo, M., Slavov, K., and S. Sugimoto, “Socket Application Program Interface (API) for Multihoming Shim,” draft-ietf-shim6-multihome-shim-api-13 (work in progress), February 2010 (TXT).
[I-D.ietf-shim6-proto]	Nordmark, E. and M. Bagnulo, “Shim6: Level 3 Multihoming Shim Protocol for IPv6,” draft-ietf-shim6-proto-12 (work in progress), February 2009 (TXT).
[POSIX]	Institute of Electrical and Electronics Engineers, “IEEE Std. 1003.1-2001 Standard for Information Technology - Portable Operating System Interface (POSIX),” Dec 2001.
[RFC3493]	Gilligan, R., Thomson, S., Bound, J., McCann, J., and W. Stevens, “Basic Socket Interface Extensions for IPv6,” RFC 3493, February 2003 (TXT).
[RFC4423]	Moskowitz, R. and P. Nikander, “Host Identity Protocol (HIP) Architecture,” RFC 4423, May 2006 (TXT).
[RFC4843]	Nikander, P., Laganier, J., and F. Dupont, “An IPv6 Prefix for Overlay Routable Cryptographic Hash Identifiers (ORCHID),” RFC 4843, April 2007 (TXT).
[RFC5014]	

	Nordmark, E., Chakrabarti, S., and J. Laganier, "IPv6 Socket API for Source Address Selection," RFC 5014, September 2007 (TXT).
[RFC5201]	Moskowitz, R., Nikander, P., Jokela, P., and T. Henderson, "Host Identity Protocol," RFC 5201, April 2008 (TXT).
[RFC5205]	Nikander, P. and J. Laganier, "Host Identity Protocol (HIP) Domain Name System (DNS) Extensions," RFC 5205, April 2008 (TXT).
[RFC5338]	Henderson, T., Nikander, P., and M. Komu, "Using the Host Identity Protocol with Legacy Applications," RFC 5338, September 2008 (TXT).

Authors' Addresses

TOC

	Miika Komu
	Helsinki Institute for Information Technology
	Metsanneidonkuja 4
	Helsinki
	Finland
Phone:	+358503841531
Fax:	+35896949768
Email:	miika@iki.fi
URI:	http://www.iki.fi/miika/
	Thomas Henderson
	The Boeing Company
	P.O. Box 3707
	Seattle, WA
	USA
Email:	thomas.r.henderson@boeing.com