          Native Application Programming Interfaces for SHIM Layer Prococols
                      draft-ietf-hip-native-api-01

Status of this Memo

Copyright Notice

Abstract

   This document proposes extensions to the current networking APIs for
   protocols based on identifier/locator split. Currently, the document
   focuses on HIP, but the extensions can be used also by other
   protocols implementing identifier locator split. Using the API
   extensions, new SHIM aware applications can gain a better control of

the SHIM layer and endpoint identifiers. For example, the
applications can query and set SHIM related attributes, or specify
their own endpoint identifiers for a host. In addition, a new
indirection element called endpoint descriptor is defined for SHIM
aware applications that can be used for implementing opportunistic
mode in a clean way.

Table of Contents

1. Introduction

The extensions defined in this draft can be used also by other
protocols based on the identifier/locator split. However, this
documented focuses mainly to HIP.

Host Identity Protocol proposes a new cryptographic namespace and a
new layer to the TCP/IP architecture. Applications can see these new
changes in the networking stacks with varying degrees of visibility.
[I-D.henderson-hip-applications] discusses the lowest levels of
visibility in which applications are either completely or partially
unaware of HIP. In this document, we discuss about the highest level
of visibility. The applications are completely HIP aware and can
control the HIP layer and Host Identifiers. The applications aquery
and set security related attributes and even create their own Host
Identifiers.

Existing applications can be used with HIP as described in
[I-D.henderson-hip-applications]. The reason why HIP can be used in
a backwards compatible way lies in the identifiers. A HIP enabled
system can support the use of LSIs, HITs and even IP addresses as
upper layer identifiers to accomodate varying application
requirements. However, these types of identifiers are not forwards
compatible. The length of HIT may turn out insecure in the future.
There may be a need to change the HITs on the fly to an already
connected socket for dynamic session mobility. Or, the socket is
going to be associated to multiple HITs for HIP based multicast.

To support forwards compatibility, we introduce a new, generalized
identifier called the endpoint descriptor (ED). The ED acts as a
handle to the actual identifier that separates application layer
indentifiers from the lower layer identifiers.

The ED can already now be used for implementing HIP opportunistic
mode in a clean way. The problem with implementing HIP opportunistic
mode is that e.g. sockets API connect() call should be bound to a HIT

in order to use HIP, but the HIT is unknown until the reception the
R1 packet. At this point it is too late to change the binding e.g.
from a IP to HIT. However, the ED has to property of late binding
and therefore provides a cleaner way to implement the opportunistic
mode.

The ED socket address structure does not reveal the transport
layerport number to the application even though it is possible to
request it explicitly. This makes it possible to change the port
number dynamically without affecting the application. Also, it seems
that the port number is irrelevant, or even misleading, in todays
NATted networks to the application.

The document also introduces a new address family, PF_SHIM, for
sockets that use EDs. The new family is a direct consequence of
introducing a new address type (ED) to the sockets API. It can also
be used for quick detection of SHIM support in the localhost. This
is especially useful discover when SHIM aware applications are tried
on a host that does not support SHIM.

The ED concept is similar to Local Scope Identifier
[I-D.henderson-hip-applications] in the sense that it is also valid
only within a host. However, it has some differences. A minor
difference is that two LSIs are the same when they refer to the same
endpoint, but ED does not have this constraint. LSIs have a prefix
to separate them from IP addresses, but ED do not. However, the main
reason why ED is not denoted as LSI in this document is that the LSIs
are bound to AF_INET sockets whereas EDs are bound to PF_SHIM
sockets.

2. Design Architecture

In this section, the native SHIM API design is described from an
architectural point of view. We introduce the ED concept, which is a
central idea in the API. We describe the layering and namespace
models along with the socket bindings. We conclude the discussion
with a description of the endpoint identifier resolution mechanism.

2.1. Endpoint Descriptor

The representation of endpoints is hidden from the applications. The

ED is a ``handle'' to a HI. A given ED serves as a pointer to the
corresponding HI entry in the HI database of the host. It should be
noticed that the ED cannot be used as a referral that is passed from
one host to another because it has only local significance.

2.2. Layering Model

The application layer accesses the transport layer via the socket
interface. The application layer uses the traditional TCP/IP IPv4 or
IPv6 interface, or the new native SHIM API interface provided by the
socket layer. The layering model is illustrated in Figure 1. For
simplicity, the IPsec layer has been excluded from the figure.

```
                        +--------------------------------+
     Application Layer  |            Application          |
                        +----------+----------+----------+
         Socket Layer   | IPv4 API | IPv6 API | SHIM API |
                        +----------+----+-----+----------+
      Transport Layer   |    TCP        |        UDP      |
                        +---------------+----------------+
           SHIM Layer   |      HIP  and other SHIMs       |
                        +---------------+----------------+
        Network Layer   |     IPv4      |      IPv6       |
                        +---------------+----------------+
           Link Layer   |   Ethernet    |      Etc        |
                        +---------------+----------------+
```
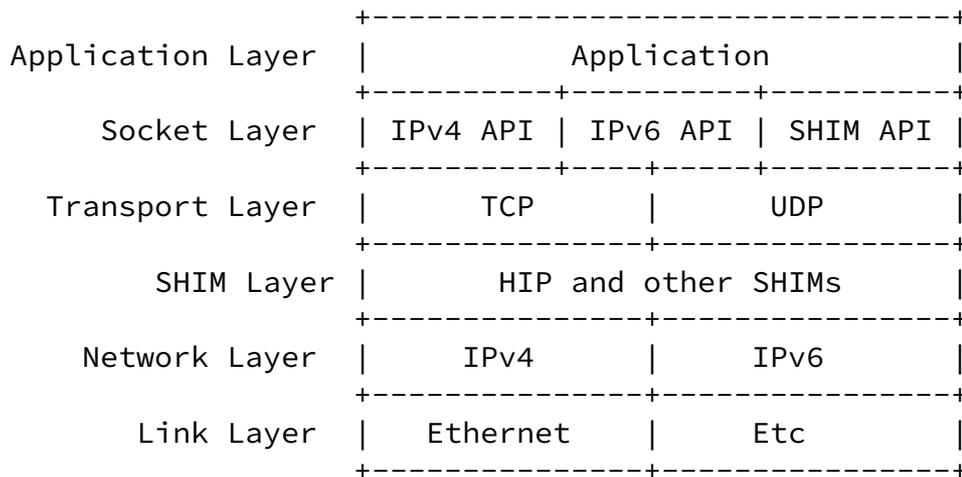
                                Figure 1

The SHIM layer is as a shim/wedge layer between the transport and
network layers. The datagrams delivered between the transport and
network layers are intercepted in the SHIM layer to see if the
datagrams are SHIM related and require SHIM intervention.

2.3. Namespace Model

The namespace model is shown in from HIP view point. The namespace
identifiers are described in this section.

```
        +------------------+----------------------+
        | Layer            | Identifier           |
        +------------------+----------------------+
        | User Interface   | FQDN                 |
        | Application Layer | ED, port and protocol |
        | Transport Layer  | HI, port             |
        | SHIM Layer       | HI                   |
        | Network Layer    | IP address           |
        +------------------+----------------------+
```

                            Table 1

   People prefer human-readable names when referring to network
   entities. The most commonly used identifier in the User Interface is
   the FQDN, but there are also other ways to name network entities.
   The FQDN format is still the preferred UI level identifier in the
   context of the native SHIM API.

   In the current API, connection associations in the application layer
   are uniquely distinguished by the source IP address, destination IP
   address, source port, destination port, and protocol. HIP changes
   this model by using HIT in the place of IP addresses. The HIP model

   is further expanded in the native HIP API model by using ED instead
   of HITs. Now, the application layer uses source ED, destination ED,
   source port, destination port, and transport protocol type, to
   distinguish between the different connection associations.

   Basically, the difference between the application and transport layer
   identifiers is that the transport layer uses HIs instead of EDs. The
   TLI is named with source HI, destination HI, source port, and
   destination port at the transport layer.

   Correspondingly, the HIP layer uses HIs as identifiers. The HIP
   security associations are based on source HI and destination HI
   pairs.

   The network layer uses IP addresses, i.e., locators, for routing

purposes. The network layer interacts with the HIP layer to exchange
information about changes in the local interfaces addresses and peer
addresses.

2.4. Socket Bindings

A HIP based SHIM socket is associated with one source and one
destination ED, along with their port numbers and protocol type. The
relationship between a socket and ED is a many-to-one one. Multiple
EDs can be associated with a single HI. Further, the source HI is
associated with a set of network interfaces at the local host. The
destination HI, in turn, is associated with a set of destination
addresses of the peer. The socket bindings are visualized in
Figure 2.

```
                       1 +---------+ *  * +--------+ *  * +-----------+
                       +---+ Src EID +------+ Src HI +------+ Src Iface |
       +--------+ *     |   +---------+      +--------+      +-----------+
       |  HIP   +------+
       |        |
       | Socket +------+
       +--------+ *     |   +---------+      +--------+      +-----------+
                       +---+ Dst EID +------+ Dst HI +------+  Dst IP   |
                       1 +---------+ *  * +--------+ *  * +-----------+
```
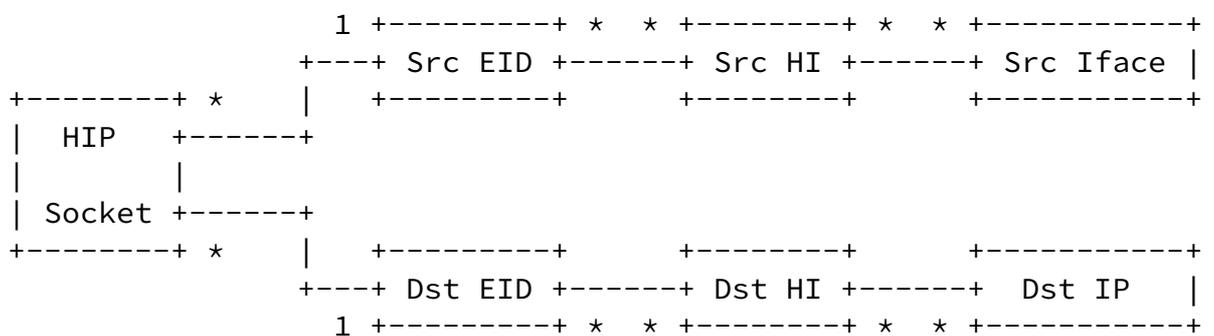
                              Figure 2

The relationship between a source ED and a source HI is usually many-
to-one one, but it can be also many-to-many on certain cases. There
are two refinements to the relationship. First, a listening socket
is allowed to accept connections from all local HIs of the host.
Second, the opportunistic mode allows the base exchange to be
initiated to an unknown destination HI. In a way, the relationship

between the local ED and local HI is a many-to-undefined relationship
momentarily in both of the cases, but once the connection is
established, the ED will be permanently associated with a certain HI.

The DNS based endpoint discovery mechanism is illustrated in
Figure 3. The application calls the resolver (step a.) to resolve an
FQDN (step b.). The DNS server responds with a ED and a set of
locators (step c.). The resolver does not directly pass the ED and

the locators to the application, but sends them to the SHIM module
(step d.). Finally, the resolver receives an ED from the SHIM module
(step e.) and passes the ED to the application (step f.).

```
                          +----------+
                          |          |
                          |   DNS    |
                          |          |
                          +----------+
                             ^   |
                 b. <FQDN> |   | c. <EID, locator>
                             |   v
    +-------------+ a. <FQDN>  +----------+
    |             |---------->|          |
    | Application |           | Resolver |
    |             |<----------|          |
    +-------------+  f. <ED>   +----------+
                             ^   |
                 e. <ED>   |   | d. <EID, locator>
                             |   v
                          +----------+
                          |          |
                          |   SHIM   |
                          |          |
                          +----------+
```
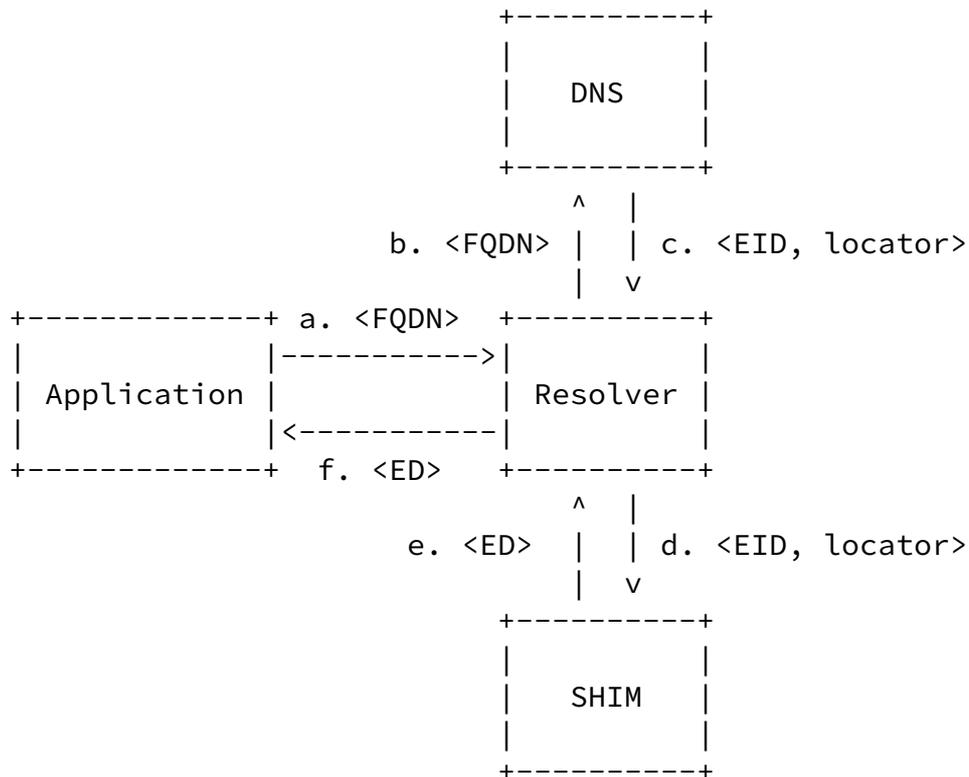
                             Figure 3

   The application can also receive multiple EDs from the resolver when
   the FQDN is associated with multiple EIDs. The endpoint discovery
   mechanism is still almost the same. The difference is that the DNS
   returns a set of EIDs (along with the associated locators) to the
   resolver. The resolver sends all of them to the SHIM module and
   receives a set of EDs in return, each ED corresponding to a single
   HI. Finally, the EDs are sent to the application.

3. Interface Syntax and Description

   In this section, we describe the native SHIM API using the syntax of

   the C programming language and present only the ``external''

interfaces and data structures that are visible to the applications.
We limit the description to those interfaces and data structures that
are either modified or completely new, because the native SHIM API is
otherwise identical to the sockets API [POSIX].

3.1. Data Structures

We introduce a new protocol family, PF_SHIM, for the sockets API.
The AF_SHIM constant is an alias for it. The use of the PF_SHIM
constant is mandatory with the socket function if the native SHIM API
is to be used in the application. The PF_SHIM constant is given as
the first argument (domain) to the socket function.

The ED abstraction is realized in the sockaddr_ed structure, which is
shown in Figure 4. The family of the socket, ed_family, is set to
PF_SHIM. The port number ed_port is two octets and the ED value
ed_val is four octets. The ED value is just an opaque number to the
application. The application should not try to associate it directly
to a EID or even compare it to other ED values, because there are
separate functions for those purposes. The ED family is stored in
host byte order. The ED value is stored in network byte order. It
should be noticed that the port number is not present in the socket
address structure, but it can be queried with the functions in
section Section 3.2.2.

```
struct sockaddr_ed {
        unsigned short int ed_family;
        sa_ed_t ed_val;
}
```

Figure 4

The ed_val field is usually set by special native SHIM API functions,
which are described in the following section. However, three special
macros can be used to directly set a value into the ed_val field.
The macros are SHIM_ED_ANY, SHIM_ED_ANY_PUB and SHIM_ED_ANY_ANON.
They denote an ED value associated with a wildcard HI of any, public,
or anonymous type. They are useful to a ``server'' application that
is willing to accept connections to all of the HIs of the host. The
macros correspond to the sockets API macros INADDR_ANY and
IN6ADDR_ANY_INIT, but they are applicable on the SHIM layer. It
should be noted that only one process at a time can bind with the
SHIM_ED_*ANY macro on a certain port to avoid ambiguous bindings.

The native SHIM API has a new resolver function which is used for
querying both endpoint identifiers and locators. The resolver
introduces a new data structure, which is used both as the input and

output argument for the resolver. We reuse the existing resolver
datastructure shown in Figure 5.

```
struct addrinfo {
    int    ai_flags;         /* e.g. AI_ED */
    int    ai_family;        /* e.g. PF_SHIM */
    int    ai_socktype;      /* e.g. SOCK_STREAM */
    int    ai_protocol;      /* usually just zero */
    size_t ai_addrlen;       /* length of the endpoint */
    struct sockaddr *ai_addr; /* endpoint socket address */
    char   *ai_canonname;    /* canon. name of the host */
    struct addrinfo *ai_next; /* next endpoint */
};
```

Figure 5

In addrinfo structures, the family field is set to PF_SHIM when the
socket address structure contains an ED that refers to a SHIM
identifier, such as HI.

The flags in the addrinfo structure control the behavior of the
resolver and describe the attributes of the endpoints and locators:

o The flag AI_ED must be set, or otherwise the resolver does not
   return EDs to guarantee that legacy applications won't break.
   When AI_ED is set, the resolver returns a linked list which
   contains first the sockaddr_ed structures for SHIM identifiers if
   any was found. After that, any other type of socket addresses are
   returned except that HITs in sockaddr_in6 format are excluded
   because they were already included in the returned sockaddr_ed
   structures.

o When querying local identifiers, the AI_ED_ANON flag forces the
   resolver to query only local anonymous identifiers. The default
   action is first to resolve the public endpoints and then the
   anonymous endpoints.

o Some applications may prefer configuring the locators manually and
   can set the AI_ED_NOLOCATORS flag to prohibit the resolver from
   resolving any locators.

o The AI_ED_ANY, AI_ED_ANY_PUB and AI_ED_ANY_ANON flags cause the
   resolver to output only a single socket address containing an ED
   that would be received using the corresponding SHIM_ED_*ANY macro.
   When these flags are used for resolving local addresses, they
   allow wildcard late binding to certain types of local idenfiers.

When the flags are used for peer resolving, they allow to contact
the peer using opportunistic mode.

   o The getaddrinfo resolver does not return IP addresses belonging to
      a SHIM rendezvous server unless AI_ED is defined. AI_ED_RVS, can
      appear both in the input and output arguments of the resolver. In
      the input, it can be used for resolving only rendezvous server
      addresses. On the output, it denotes that the address is a
      rendezvous rather than end-point address.

   Application specified endpoint identifiers are essentially private
   keys. To support application specified identifiers in the API, we
   introduce new data structures for storing the private keys. The
   private keys need an uniform format so that they can be easily used
   in the API calls. The keys are stored in the endpoint structures as
   shown in Figure 6.

```
        struct endpoint {
                se_length_t   length;
                se_family_t   family;
        };
        struct endpoint_hip {
                se_length_t length;
                se_family_t family; /* EF_HI in the case of HIP */
                se_hip_flags_t flags;
                union {
                        struct hip_host_id host_id;
                        hit_t hit;
                } id;
        };
```

                            Figure 6

   The endpoint structure represents a generic endpoint and the
   endpoint_hip structure represents a HIP specific endpoint. The
   family field distinguishes whether the identifier is HIP or other
   protocol related. The HIP endpoint is public by default unless
   SHIM_ENDPOINT_FLAG_ANON flag is set in the structure to anonymize the
   endpoint. The id union contains the HI in the host_id member in the
   format specified in [I-D.ietf-hip-base]. If the key is private, the
   material is appended to the host_id with the length adjusted
   accordingly. The flag SHIM_ENDPOINT_FLAG_PRIVATE is also set. The

hit member of the union is used only when the SHIM_ENDPOINT_FLAG_HIT
flag is set.

3.2. Functions

In this section, some existing sockets API functions are reintroduced
along with their additions. Also, some new auxiliary functions are
defined.

3.2.1. Resolver Interface

The native SHIM API does not introduce changes to the interface
syntax of the primitive sockets API functions bind, connect, send,
sendto, sendmsg, recv, recvfrom, and recvmsg. However, the
application usually calls the functions with sockaddr_ed structures
instead of sockaddr_in or sockaddr_in6 structures. The source of the
sockaddr_ed structures in the native SHIM API is the resolver
function getaddrinfo [RFC3493] which is shown in Figure 7.

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

                                Figure 7

The getaddrinfo function takes the nodename, servname, and hints as
its input arguments. It places the result of the query into the res
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 local host. The servname
parameter sets the port number to be set in the socket addresses in
the res output argument. Both the nodename and servname cannot be
NULL.

The output argument res is dynamically allocated by the resolver.
The application must free res argument with the free_addrinfo
function. The res argument contains a linked list of the resolved
endpoints. The input argument hints acts like a filter that defines
the attributes required from the resolved endpoints. For example,

setting the flag SHIM_ENDPOINT_FLAG_ANON in the hints forces the
resolver to return only anonymous endpoints in the output argument
res. A NULL hints argument indicates that any kind of endpoints are
acceptable.

3.2.2. Application Specified Identities

   Application specified local and peer endpoints can be retrieved from
   files using the function shown in Figure 8. The function
   shim_endpoint_load_pem is used for retrieving a private or public key
   from a given file filename. The file must be in PEM encoded format.
   The result is allocated dynamically and stored into the endpoint
   argument. The return value of the function is zero on success, or a
   non-zero error value on failure. The result is deallocated with the
   free system call.

```
        int shim_endpoint_pem_load(const char *filename,
                                   struct endpoint **endpoint)
```

                               Figure 8

   Alternatively, the application can load the image directly from
   memory as shown in Figure 9

```
        int shim_endpoint_pem_load_str(const char *pem_str,
                                       struct endpoint **endpoint);
```

                               Figure 9

   The endpoint structure cannot be used directly in the sockets API
   function calls. The application must convert the endpoint into an ED
   first. Local endpoints are converted with the getlocaled function
   and peer endpoints with getpeered function. The functions are
   illustrated in Figure 10.

```
      struct sockaddr_ed *getlocaled(const struct endpoint *endpoint,
                                     const char *servname,
                                     const struct addrinfo *addrs,
                                     const struct if_nameindex *ifaces,
```

```
                                          int flags)
        struct sockaddr_ed *getpeered(const struct endpoint *endpoint,
                                      const char *servname,
                                      const struct addrinfo *addrs,
                                      int flags)
```

                                Figure 10

   The result of the conversion, an ED socket address, is returned by
   both of the functions. A failure in the conversion causes a NULL
   return value to be returned and the errno to be set accordingly. The
   caller of the functions is responsible of freeing the returned socket
   address structure.

   The application can retrieve the endpoint argument e.g. with the
   shim_endpoint_load_pem function. If the endpoint is NULL, the system
   selects an arbitrary EID and associates it with the ED value of the
   return value.

   The servname argument is the service string. The function converts
   it to a numeric port number and fills the port number into the
   returned ED socket structure for the convenience of the application.

   The addrs argument defines the initial IP addresses of the local host
   or peer host. The argument is a pointer to a linked list of addrinfo
   structures containing the initial addresses of the peer. The list
   pointer can be obtained with a getaddrinfo [RFC3493] function call.
   A NULL pointer indicates that the application trusts the host to
   already know the locators of the peer. We recommend that a NULL
   pointer is not given to the getpeered function to ensure reachability
   with the peer.

   The getlocaled function accepts also a list of network interface
   indexes in the ifaces argument. The list can be obtained with the
   if_nameindex [RFC3493] function call. A NULL list pointer indicates
   all the interfaces of the local host. Both the IP addresses and
   interfaces can be combined to select a specific address from a
   specific interface.

   The last argument is the flags. The following flags are valid only
   for the getlocaled function:

o Flags SHIM_ED_REUSE_UID, SHIM_ED_REUSE_GID and SHIM_ED_REUSE_ANY
      allow the EID (e.g. a large private key) to be reused for
      processes with the same UID, GID or any UID as the calling
      process.

   o Flags SHIM_ED_IPV4 and SHIM_ED_IPV6 can be used for limiting the
      address family scope of the local interface.

   It should noticed that the SHIM_ED_ANY, SHIM_ED_ANY_PUB and
   SHIM_ED_ANY_ANON macros can be implemented as calls to the getlocaled
   call with a NULL endpoint, NULL interface, NULL address argument and
   the flag corresponding to the macro name set.

3.2.3. Querying Endpoint Related Information

   The getlocaled and getpeered functions have also their reverse
   counterparts. Given an ED, the getlocaledinfo and getpeeredinfo
   functions search for the EID (e.g. a HI) and the current set of
   locators associated with the ED. The first argument is the ED to be
   searched for. The functions write the results of the search, the
   transport layer port number of the occupied by the correponding HIT,
   the HIs and locators, to the rest of the function arguments. The
   function interfaces are depicted in Figure 11. The caller of the
   functions is responsible for freeing the memory reserved for the
   search results.

```
        int getlocaledinfo(const struct sockaddr_ed *my_ed,
                           struct in_port_t **port
                           struct endpoint **endpoint,
                           struct addrinfo **addrs,
                           struct if_nameindex **ifaces)
        int getpeeredinfo(const struct sockaddr_ed *peer_ed,
                           struct in_port_t **port
                           struct endpoint **endpoint,
                           struct addrinfo **addrs)
```

                              Figure 11

The getlocaledinfo and getpeeredinfo functions are especially useful
for an advanced application that receives multiple EDs from the
resolver. The advanced application can query the properties of the
EDs using getlocaledinfo and getpeeredinfo functions and select the
ED that matches the desired properties.

3.2.4. HIP Related Policy Attributes

Multihoming related attributes are defined in
[I-D.ietf-shim6-multihome-shim-api]. It also specifies an event
driven API for application, which can be used for listening for
changes in locators.

HIP related policy attributes are accessed using the definitions in
[I-D.komu-btns-api]

Some of the policy attributes must be set before the hosts have
established connection. The implementation may refuse to accept the
option when there is already an existing connection and dynamic
renegotiation of the option is not possible. In addition, the SHIM
may return an error value if the corresponding SHIM protocol does not
support the given option.

Table 2shows HIP related policy attributes that are accessed with the
APIs defined in [I-D.komu-btns-api].

```
  +--------------------+----------------------------------------------+
  | Attribute          | Purpose                                      |
  +--------------------+----------------------------------------------+
  | IPSEC_ESP_TRANSFORM | Preferred ESP transform                     |
```

```
       | IPSEC_SA_LIFETIME  | Preferred IPsec SA lifetime in seconds    |
       | SHIM_PROTOCOL      | Get or set current SHIM protocol. Currently |
       |                    | only PF_HIP is defined.                   |
       | SHIM_CHALLENGE_SIZE | Puzzle challenge size                    |
       | SHIM_SHIM_TRANSFORM | Preferred SHIM transform                 |
       | SHIM_DH_GROUP_IDS  | The preferred Diffie-Hellman Group        |
       | SHIM_AF_FAMILY     | The preferred locator family. The default |
       |                    | family is AF_ANY.                         |
       | SHIM_FAST_FALLBACK | If set to one, use the extensions in      |
       |                    | [I-D.lindqvist-hip-opportunistic]         |
       | SHIM_FAST_HANDSHAKE | If set to one, use the extensions in     |
       |                    | [I-D.lindqvist-hip-tcp-piggybacking]      |
       +--------------------+-------------------------------------------+
```

                             Table 2

4. IANA Considerations

   No IANA considerations.

5. Security Considerations

   To be done.

6. Acknowledgements

   Jukka Ylitalo and Pekka Nikander have contributed many ideas, time
   and effort to the native HIP API. Thomas Henderson, Kristian Slavov,
   Julien Laganier, Jaakko Kangasharju, Mika Kousa, Jan Melen, Andrew
   McGregor, Sasu Tarkoma, Lars Eggert, Joe Touch, Antti Jaervinen,
   Anthony Joseph, Teemu Koponen and Juha-Matti Tapio have also provided
   valuable ideas and feedback.

7. References

7.1. Normative References

   [I-D.henderson-hip-applications]
              Henderson, T. and P. Nikander, "Using HIP with Legacy
              Applications", draft-henderson-hip-applications-03 (work

                in progress), May 2006.

   [I-D.ietf-hip-base]
                Moskowitz, R., "Host Identity Protocol",
                draft-ietf-hip-base-07 (work in progress), February 2007.

   [I-D.ietf-shim6-multihome-shim-api]
                Komu, M., "Socket Application Program Interface (API) for
                Multihoming Shim", draft-ietf-shim6-multihome-shim-api-01
                (work in progress), October 2006.

   [I-D.komu-btns-api]
                Komu, M., "IPsec Application Programming Interfaces",
                draft-komu-btns-api-00 (work in progress), October 2006.

   [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.

7.2. Informative References

   [I-D.lindqvist-hip-opportunistic]
                Lindqvist, J., "Establishing Host Identity Protocol
                Opportunistic Mode with TCP Option",
                draft-lindqvist-hip-opportunistic-01 (work in progress),
                March 2006.

   [I-D.lindqvist-hip-tcp-piggybacking]
                Lindqvist, J., "Piggybacking TCP to Host Identity
                Protocol", draft-lindqvist-hip-tcp-piggybacking-00 (work
                in progress), July 2006.

Author's Address

   Miika Komu
   Helsinki Institute for Information Technology
   Tammasaarenkatu 3
   Helsinki
   Finland

   Phone: +358503841531
   Fax: +35896949768
   Email: miika@iki.fi
   URI: http://www.iki.fi/miika/

Full Copyright Statement

Intellectual Property

http://www.ietf.org/ipr.

Acknowledgment