                HIP BONE: Host Identity Protocol (HIP)
             Based Overlay Networking Environment (BONE)

Abstract

   This document specifies a framework to build HIP-based (Host Identity
   Protocol) overlay networks. This framework uses HIP to perform
   connection management. Other functions, such as data storage and
   retrieval or overlay maintenance, are implemented using protocols
   other than HIP. These protocols are loosely referred to as "peer
   protocols".

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

   The Host Identity Protocol (HIP) [RFC5201] defines a new name space
   between the network and transport layers. HIP provides upper layers
   with mobility, multihoming, NAT (Network Address Translation)
   traversal, and security functionality. HIP implements the so-called
   identifier/locator (ID/locator) split, which implies that IP
   addresses are only used as locators, not as host identifiers. This
   split makes HIP a suitable protocol to build overlay networks that
   implement identifier-based overlay routing over IP networks, which in
   turn implement locator-based routing.

   Using HIP-Based Overlay Networking Environment (HIP BONE), as opposed
   to a peer protocol, to perform connection management in an overlay
   has a set of advantages. HIP BONE can be used by any peer protocol.
   This keeps each peer protocol from defining primitives needed for
   connection management (e.g., primitives to establish connections and
   to tunnel messages through the overlay) and NAT traversal. Having
   this functionality at a lower layer allows multiple upper-layer
   protocols to take advantage of it.

   Additionally, having a solution that integrates mobility and
   multihoming is useful in many scenarios. Peer protocols do not
   typically specify mobility and multihoming solutions. Combining a
   peer protocol including NAT traversal with a separate mobility
   mechanism and a separate multihoming mechanism can easily lead to
   unexpected (and unpleasant) interactions.

   The remainder of this document is organized as follows. Section 3
   provides background information on HIP. Section 4 gives an overview
   of the HIP BONE (HIP-Based Overlay Networking Environment) framework
   and architecture and Section 5 describes the framework in more
   detail. Finally, Section 6 introduces new HIP parameters for overlay
   usage.

2. Terminology

   The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
   "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
   document are to be interpreted as described in RFC 2119 [RFC2119].

   The following terms are used in context of HIP BONEs:

   Overlay network: A network built on top of another network. In case
      of HIP BONEs, the underlying network is an IP network and the
      overlay can be, e.g., a peer-to-peer (P2P) network.

   Peer protocol: A protocol used by nodes in an overlay network for
      performing, e.g., data storage and retrieval or overlay
      maintenance.

   HIP BONE instance: A HIP-based overlay network that uses a
      particular peer protocol and is based on the framework presented
      in this document.

   Node ID: A value that uniquely identifies a node in an overlay
      network. The value is not usually human-friendly. As an example,
      it may be a hash of a public key.

   HIP association: An IP-layer communications context created using
      the Host Identity Protocol.

   Valid locator: A locator (as defined in [RFC5206]; usually an IP
      address or an address and a port number) at which a host is known
      to be reachable, for example, because there is an active HIP
      association with the host.

   Final recipient: A node is the final recipient of a HIP signaling
      packet if one of its Host Identity Tags (HITs) matches to the
      receiver's HIT in the HIP packet header.

3. Background on HIP

   This section provides background on HIP. Given the tutorial nature
   of this section, readers that are familiar with what HIP provides and
   how HIP works may want to skip it. All descriptions contain
   references to the relevant HIP specifications where readers can find
   detailed explanations on the different topics discussed in this
   section.

3.1. ID/Locator Split

   In an IP network, IP addresses typically serve two roles: they are
   used as host identifiers and as host locators. IP addresses are
   locators because a given host's IP address indicates where in the
   network that host is located. IP networks route based on these
   locators. Additionally, IP addresses are used to identify remote
   hosts. The simultaneous use of IP addresses as host identifiers and
   locators makes mobility and multihoming complicated. For example,
   when a host opens a TCP connection, the host identifies the remote
   end of the connection by the remote IP address (plus port). If the
   remote host changes its IP address, the TCP connection will not
   survive, since the transport layer identifier of the remote end of
   the connection has changed.

Mobility solutions such as Mobile IP keep the remote IP address from
changing so that it can still be used as an identifier. HIP, on the
other hand, uses IP addresses only as locators and defines a new
identifier space. This approach is referred to as the ID/locator
split and makes the implementation of mobility and multihoming more
natural. In the previous example, the TCP connection would be bound
to the remote host's identifier, which would not change when the
remote hosts moves to a new IP address (i.e., to a new locator). The
TCP connection is able to survive locator changes because the remote
host's identifier does not change.

3.1.1. Identifier Format

HIP uses 128-bit ORCHIDs (Overlay Routable Cryptographic Hash
Identifiers) [RFC4843] as identifiers. ORCHIDs look like IPv6
addresses but cannot collide with regular IPv6 addresses because
ORCHID spaces are registered with the IANA. That is, a portion of
the IPv6 address space is reserved for ORCHIDs. The ORCHID
specification allows the creation of multiple disjoint identifier
spaces. Each such space is identified by a separate Context
Identifier. The Context Identifier can be either drawn implicitly
from the context the ORCHID is used in or carried explicitly in a
protocol.

HIP defines a native socket API [HIP-NATIVE-API] that applications
can use to establish and manage connections. Additionally, HIP can
also be used through the traditional IPv4 and IPv6 TCP/IP socket
APIs. Section 3.4 describes how an application using these
traditional APIs can make use of HIP. Figure 1 shows all these APIs
between the application and the transport layers.

```
        +----------------------------------------+
        |              Application               |
        +---------------+------------------------+
        | HIP Native API | Traditional Socket API |
        +---------------+------------------------+
        |             Transport Layer            |
        +----------------------------------------+
```
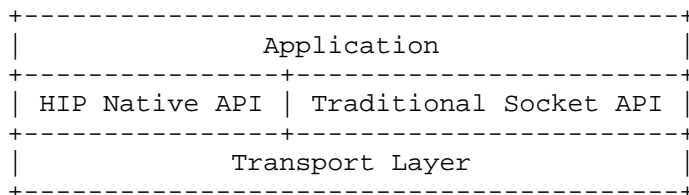
Figure 1: HIP API

3.1.2. HIP Base Exchange

Typically, before two HIP hosts exchange upper-layer traffic, they
perform a four-way handshake that is referred to as the HIP base
exchange. Figure 2 illustrates the HIP base exchange. The initiator

sends an I1 packet and receives an R1 packet from the responder.
After that, the initiator sends an I2 packet and receives an R2
packet from the responder.

```
            Initiator                     Responder

                   |             I1            |
                   |-------------------------->|
                   |             R1            |
                   |<--------------------------|
                   |             I2            |
                   |-------------------------->|
                   |             R2            |
                   |<--------------------------|
```
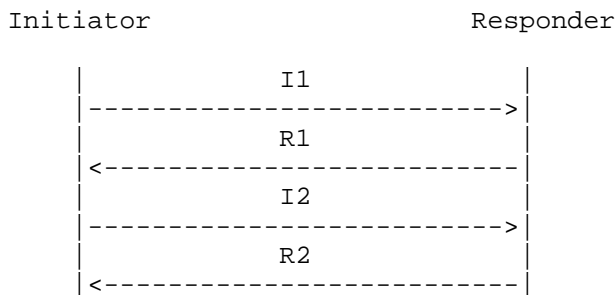
                   Figure 2: HIP Base Exchange

Of course, the initiator needs the responder's locator (or locators)
in order to send its I1 packet. The initiator can obtain locators
for the responder in multiple ways. For example, according to the
current HIP specifications the initiator can get the locators
directly from the DNS [RFC5205] or indirectly by sending packets
through a HIP rendezvous server [RFC5204]. However, HIP is an open-
ended architecture. The HIP architecture allows the locators to be
obtained by any means (e.g., from packets traversing an overlay
network or as part of the candidate address collection process in a
NAT traversal scenario).

3.1.3. Locator Management

Once a HIP connection between two hosts has been established with a
HIP base exchange, the hosts can start exchanging higher-layer
traffic. If any of the hosts changes its set of locators, it runs an
update exchange [RFC5206], which consists of three messages. If a
host is multihomed, it simply provides more than one locator in its
exchanges. However, if both of the endpoints move at the same time,
or through some other reason both lose track of the peers' currently
active locators, they need to resort to using a rendezvous server or
getting new peer locators by some other means.

3.2. NAT Traversal

HIP's NAT traversal mechanism [RFC5770] is based on ICE (Interactive
Connectivity Establishment) [RFC5245]. Hosts gather address
candidates and, as part of the HIP base exchange, hosts perform an
ICE offer/answer exchange where they exchange their respective

address candidates. Hosts perform end-to-end STUN-based [RFC5389]
connectivity checks in order to discover which address candidate
pairs yield connectivity.

Even though, architecturally, HIP lies below the transport layer
(i.e., HIP packets are carried directly in IP packets), in the
presence of NATs, HIP sometimes needs to be tunneled in a transport
protocol (i.e., HIP packets are carried by a transport protocol such
as UDP).

3.3. Security

   Security is an essential part of HIP. The following sections
   describe the security-related functionality provided by HIP.

3.3.1. DoS Protection

   HIP provides protection against DoS (denial-of-service) attacks by
   having initiators resolve a cryptographic puzzle before the responder
   stores any state. On receiving an I1 packet, a responder sends a
   pre-generated R1 packet that contains a cryptographic puzzle and
   deletes all the state associated with the processing of this I1
   packet. The initiator needs to resolve the puzzle in the R1 packet
   in order to generate an I2 packet. The difficulty of the puzzle can
   be adjusted so that, if a receiver is under a DoS attack, it can
   increase the difficulty of its puzzles.

   On receiving an I2 packet, a receiver checks that the solution in the
   packet corresponds to a puzzle generated by the receiver and that the
   solution is correct. If it is, the receiver processes the I2 packet.
   Otherwise, it silently discards it.

   In an overlay scenario, there are multiple ways in which this
   mechanism can be utilized within the overlay. One possibility is to
   cache the pre-generated R1 packets within the overlay and let the
   overlay directly respond with R1s to I1s. In that way, the responder
   is not bothered at all until the initiator sends an I2 packet, with
   the puzzle solution. Furthermore, a more sophisticated overlay could
   verify that an I2 packet has a correctly solved puzzle before
   forwarding the packet to the responder.

3.3.2. Identifier Assignment and Authentication

   As discussed earlier, HIP uses ORCHIDs [RFC4843] as the main
   representation for identifiers. Potentially, HIP can use different
   types of ORCHIDs as long as the probability of finding collisions
   (i.e., two nodes with the same ORCHID) is low enough. One way to
   completely avoid this type of collision is to have a central

authority generate and assign ORCHIDs to nodes. To secure the
binding between ORCHIDs and any higher-layer identifiers, every time
the central authority assigns an ORCHID to a node, it also generates
and signs a certificate stating who is the owner of the ORCHID. The
owner of the ORCHID then includes the corresponding certificate in
its R1 (when acting as responder) and I2 packets (when acting
initiator) to prove that it is actually allowed to use the ORCHID
and, implicitly, the associated public key.

Having a central authority works well to completely avoid collisions.
However, having a central authority is impractical in some scenarios.
As defined today, HIP systems generally use a self-certifying ORCHID
type called HIT (Host Identity Tag) that does not require a central
authority (but still allows one to be used).

A HIT is the hash of a node's public key. A node proves that it has
the right to use a HIT by showing its ability to sign data with its
associated private key. This scheme is secure due to the so-called
second-preimage resistance property of hash functions. That is,
given a fixed public key K1, finding a different public key K2 such
that hash(K1) = hash(K2) is computationally very hard. Optimally, a
preimage attack on the 100-bit hash function used in ORCHIDs will
take an order of 2^{100} operations to be successful, and can be
expected to take in the average 2^{99} operations. Given that each
operation requires the attacker to generate a new key pair, the
attack is fully impractical with current technology and techniques
(see [RFC4843]).

HIP nodes using HITs as ORCHIDs do not typically use certificates
during their base exchanges. Instead, they use a leap-of-faith
mechanism, similar to the Secure Shell (SSH) protocol [RFC4251],
whereby a node somehow authenticates remote nodes the first time they
connect to it and, then, remembers their public keys. While user-
assisted leap-of-faith mechanism (such as in SSH) can be used to
facilitate a human-operated offline path (such as a telephone call),
automated leap-of-faith mechanisms can be combined with a reputation
management system to create an incentive to behave. However, such
considerations go well beyond the current HIP architecture and even
beyond this proposal. For the purposes of the present document, we
merely want to point out that, architecturally, HIP supports both
self-generated opportunistic identifiers and administratively
assigned ones.

3.3.3. Connection Security

   Once two nodes complete a base exchange between them, the traffic
   they exchange is encrypted and integrity protected. The security
   mechanism used to protect the traffic is IPsec Encapsulating Security
   Payload (ESP) [RFC5202]. However, there is ongoing work to specify
   how to use other protection mechanisms.

3.4. HIP Deployability and Legacy Applications

   As discussed earlier, HIP defines a native socket API [HIP-NATIVE-
   API] that applications can use to establish and manage connections.
   New applications can implement this API to get full advantage of HIP.
   However, in most cases, legacy (i.e., non-HIP-aware) applications
   [RFC5338] can use HIP through the traditional IPv4 and IPv6 socket
   APIs.

   The idea is that when a legacy IPv6 application tries to obtain a
   remote host's IP address (e.g., by querying the DNS), the DNS
   resolver passes the remote host's ORCHID (which was also stored in
   the DNS) to the legacy application. At the same time, the DNS
   resolver stores the remote host's IP address internally at the HIP
   module. Since the ORCHID looks like an IPv6 address, the legacy
   application treats it as such. It opens a connection (e.g., TCP)
   using the traditional IPv6 socket API. The HIP module running in the
   same host as the legacy application intercepts this call somehow
   (e.g., using an interception library or setting up the host's routing
   tables so that the HIP module receives the traffic) and runs HIP (on
   behalf of the legacy application) towards the IP address
   corresponding to the ORCHID. This mechanism works well in almost all
   cases. However, applications involving referrals (i.e., passing of
   IPv6 addresses between applications) present issues, which are
   discussed in Section 5 below. Additionally, management applications
   that care about the exact IP address format may not work well with
   such a straightforward approach.

   In order to make HIP work through the traditional IPv4 socket API,
   the HIP module passes an LSI (Local Scope Identifier), instead of a
   regular IPv4 address, to the legacy IPv4 application. The LSI looks
   like an IPv4 address, but is locally bound to an ORCHID. That is,
   when the legacy application uses the LSI in a socket call, the HIP
   module intercepts it and replaces the LSI with its corresponding
   ORCHID. Therefore, LSIs always have local scope. They do not have
   any meaning outside the host running the application. The ORCHID is
   used on the wire; not the LSI. In the referral case, if it is not
   possible to rewrite the application level packets to use ORCHIDs

instead of LSIs, it may be hard to make IPv4 referrals work in
Internet-wide settings. IPv4 LSIs have been successfully used in
existing HIP deployments within a single corporate network.

4. Framework Overview

The HIP BONE framework combines HIP with different peer protocols to
provide robust and secure overlay network solutions.

Many overlays typically require three types of operations:

o   overlay maintenance,
o   data storage and retrieval, and
o   connection management.

Overlay maintenance operations deal with nodes joining and leaving
the overlay and with the maintenance of the overlay's routing tables.
Data storage and retrieval operations deal with nodes storing,
retrieving, and removing information in or from the overlay.
Connection management operations deal with the establishment of
connections and the exchange of lightweight messages among the nodes
of the overlay, potentially in the presence of NATs.

The HIP BONE framework uses HIP to perform connection management.
Data storage and retrieval and overlay maintenance are to be
implemented using protocols other than HIP. For lack of a better
name, these protocols are referred to as peer protocols.

One way to depict the relationship between the peer protocol and HIP
modules is shown in Figure 3. The peer protocol module implements
the overlay construction and maintenance features, and possibly
storage (if the particular protocol supports such a feature). The
HIP module consults the peer protocol's overlay topology part to make
routing decisions, and the peer protocol uses HIP for connection
management and sending peer protocol messages to other hosts. The
HIP BONE API that applications use is a combination of the HIP Native
API and traditional socket API (as shown in Figure 1) with any
additional API a particular instance implementation provides.

```
                        Application
           ----------------------------- HIP BONE API
           +---+   +-------------------+
           |   |   |   Peer Protocol   |
           |   |   +--------+ +--------+
           |   |<->|Topology| |(Storage)|
           |   |   +--------+---------+
           |   |            ^
           |   |            v
           |   +-----------------------+
           |                HIP        |
           +---------------------------+
```
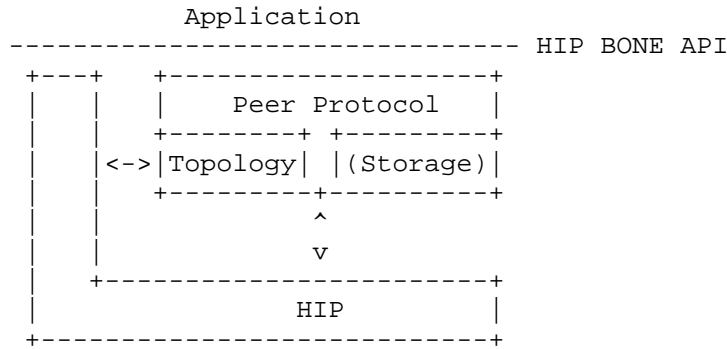
                 Figure 3: HIP with Peer Protocol

   Architecturally, HIP can be considered to create a new thin "waist"
   layer on top of the IPv4 and IPv6 networks; see Figure 4. The HIP
   layer itself consists of the HIP signaling protocol and one or more
   data transport protocols; see Figure 5. The HIP signaling packets
   and the data transport packets can take different routes. In the HIP
   BONE scenarios, the HIP signaling packets are typically first routed
   through the overlay and then directly (if possible), while the data
   transport packets are typically routed only directly between the
   endpoints.

```
        +------------------------------------+
        |   Transport (using HITs or LSIs)   |
        +------------------------------------+
        |                HIP                 |
        +-----------------+------------------+
        |     IPv4        |      IPv6        |
        +-----------------+------------------+
```
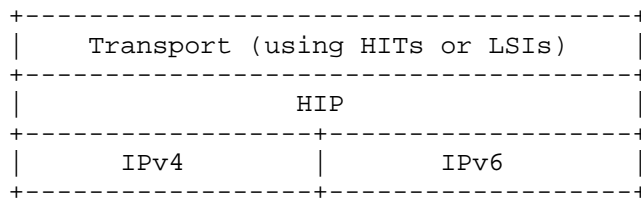
                  Figure 4: HIP as a Thin Waist

```
        +-----------------+------------------+
        |  HIP signaling  |  data transports |
        +-----------------+------------------+
```
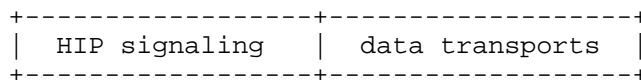
                 Figure 5: HIP Layer Structure

   In HIP BONE, the peer protocol creates a new signaling layer on top
   of HIP. It is used to set up forwarding paths for HIP signaling
   messages. This is a similar relationship that an IP routing
   protocol, such as OSPF, has to the IP protocol itself. In the HIP
   BONE case, the peer protocol plays a role similar to OSPF, and HIP
   plays a role similar to IP. The ORCHIDs (or, in general, Node IDs if
   the ORCHID prefix is not used) are used for forwarding HIP packets

according to the information in the routing tables. The peer
protocols are used to exchange routing information based on Node IDs
and to construct the routing tables.

Architecturally, routing tables are located between the peer protocol
and HIP, as shown in Figure 6. The peer protocol constructs the
routing table and keeps it updated. The HIP layer accesses the
routing table in order to make routing decisions. The bootstrap of a
HIP BONE overlay does not create circular dependencies between the
peer protocol (which needs to use HIP to establish connections with
other nodes) and HIP (which needs the peer protocol to know how to
route messages to other nodes) for the same reasons as the bootstrap
of an IP network does not create circular dependencies between OSPF
and IP. The first connections established by the peer protocol are
with nodes whose locators are known. HIP establishes those
connections as any connection between two HIP nodes where no overlays
are present. That is, there is no need for the overlay to provide a
rendezvous service for those connections.

```
+-------------------------------------+
|             Peer protocol           |
+-------------------------------------+
|             Routing table           |
+-------------------------------------+
|                  HIP                |
+-------------------------------------+
```
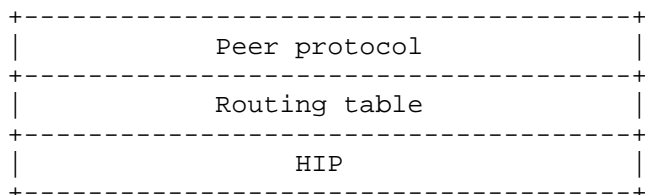
Figure 6: Routing Tables

It is possible that different overlays use different routing table
formats. For example, the structure of the routing tables of two
overlays based on different DHTs (Distributed Hash Tables) may be
very different. In order to make routing decisions, the HIP layer
needs to convert the routing table generated by the peer protocol
into a forwarding table that allows the HIP layer select a next hop
for any packet being routed.

In HIP BONE, the HIP usage of public keys and deriving ORCHIDs
through a hash function can be utilized at the peer protocol side to
better secure routing table maintenance and to protect against
chosen-peer-ID attacks.

HIP BONE provides quite a lot of flexibility with regards to how to
arrange the different protocols in detail. Figure 7 shows one
potential stack structure.

```
      +----------------------+--------------+
      | peer protocols       |    media     |
      +----------------+-----+--------------+
      | HIP signaling  | data transport     |
      |                |                    |
      +----------------+--------------------+
      | NAT    | non-NAT |                   |
      |                |                    |
      |      IPv4      |        IPv6        |
      +----------------+--------------------+
```

              Figure 7: Example HIP BONE Stack Structure

5. The HIP BONE Framework

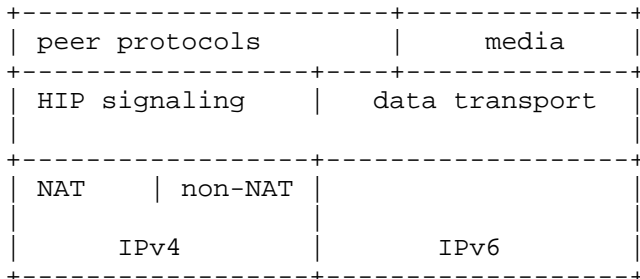   HIP BONE is a generic framework that allows the use of different peer
   protocols. A particular HIP BONE instance uses a particular peer
   protocol. The details on how to implement HIP BONE using a given
   peer protocol need to be specified in a, so-called, HIP BONE instance
   specification. Section 5.5 discusses what details need to be
   specified by HIP BONE instance specifications. For example, the HIP
   BONE instance specification for RELOAD [P2PSIP-BASE] is specified in
   [HIP-RELOAD-INSTANCE].

5.1. Node ID Assignment and Bootstrap

   Nodes in an overlay are primarily identified by their Node IDs.
   Overlays typically have an enrollment server that can generate Node
   IDs, or at least some part of the Node ID, and sign certificates. A
   certificate generated by an enrollment server authorizes a particular
   user to use a particular Node ID in a particular overlay. The way
   users are identified is defined by the peer protocol and HIP BONE
   instance specification.

   The enrollment server of an overlay using HITs derived from public
   keys as Node IDs could just authorize users to use the public keys
   and HITs associated to their nodes. Such a Node ID has the same
   self-certifying property as HITs and the Node ID can also be used in
   the HIP and legacy APIs as an ORCHID. This works well as long as the
   enrollment server is the one generating the public/private key pairs
   for all those nodes. If the enrollment server authorizes users to
   use HITs that are generated directly by the nodes themselves, the
   system is open to a type of chosen-peer-ID attack.

   If the overlay network or peer protocol has more specific
   requirements for the Node ID value that prevent using HITs derived
   from public keys, each host will need a certificate (e.g., in their
   HIP base exchanges) provided by the enrollment server to prove that

they are authorized to use a particular identifier in the overlay.
Depending on how the certificates are constructed, they typically
also need to contain the host's self-generated public key. Depending
on how the Node IDs and public keys are attributed, different
scenarios become possible. For example, the Node IDs may be
attributed to users, there may be user public key identifiers, and
there may be separate host public key identifiers. Authorization
certificates can be used to bind the different types of identifiers
together.

HITs, as defined in [RFC5201], always start with the ORCHID prefix.
Therefore, there are 100 bits left in the HIT for different Node ID
values. If an overlay network requires a larger address space, it is
also possible to use all the 128 bits of a HIT for addressing peer
layer identifiers. The benefit of using ORCHID prefix for Node IDs
is that it makes possible to use them with legacy socket APIs, but in
this context, most of the applications are assumed to be HIP aware
and able to use a more advanced API supporting full 128-bit
identifiers. Even larger address spaces could be supported with an
additional HIP parameter giving the source and destination Node IDs,
but defining such a parameter, if needed, is left for future
documents.

Bootstrap issues, such as how to locate an enrollment or a bootstrap
server, belong to the peer protocol.

5.2. Overlay Network Identification

It is possible for a HIP host to participate simultaneously in
multiple different overlay networks. It is also possible that some
HIP traffic is not intended to be forwarded over an overlay.
Therefore, a host needs to know to which overlay an incoming HIP
message belongs and the outgoing HIP messages need to be labeled as
belonging to a certain overlay. This document specifies a new
generic HIP parameter (in Section 6.1) for the purpose of directing
HIP messages to the right overlay.

In addition, an application using HIP BONE needs to define, either
implicitly or explicitly, the overlay to use for communication.
Explicit configuration can happen, e.g., by configuring certain local
HITs to be bound to certain overlays or by defining the overlay
identifier using advanced HIP socket options or other suitable APIs.
On the other hand, if no explicit configuration for a HIP association
is used, a host may have a configured default overlay where all HIP
messages without a valid locator are sent. The specification for how
to implement this coordination for locally originated messages is out
of scope for this document.

5.3. Connection Establishment

   Nodes in an overlay need to establish connections with other nodes in
   different cases. For example, a node typically has connections to
   the nodes in its forwarding table. Nodes also need to establish
   connections with other nodes in order to exchange application-layer
   messages.

   As discussed earlier, HIP uses the base exchange to establish
   connections. A HIP endpoint (the initiator) initiates a HIP base
   exchange with a remote endpoint by sending an I1 packet. The
   initiator sends the I1 packet to the remote endpoint's locator.
   Initiators that do not have any locator for the remote endpoint need
   to use a rendezvous service. Traditionally, a HIP rendezvous server
   [RFC5204] has provided such a rendezvous service. In HIP BONE, the
   overlay itself provides the rendezvous service.

   Therefore, in HIP BONE, a node uses an I1 packet (as usual) to
   establish a connection with another node in the overlay. Nodes in
   the overlay forward I1 packets in a hop-by-hop fashion according to
   the overlay's routing table towards its destination. This way, the
   overlay provides a rendezvous service between the nodes establishing
   the connection. If the overlay nodes have active connections with
   other nodes in their forwarding tables and if those connections are
   protected (typically with IPsec ESP), I1 packets may be sent over
   protected connections between nodes. Alternatively, if there is no
   such an active connection but the node forwarding the I1 packet has a
   valid locator for the next hop, the I1 packets may be forwarded
   directly, in a similar fashion to how I1 packets are today forwarded
   by a HIP rendezvous server.

   Since HIP supports NAT traversal, a HIP base exchange over the
   overlay will perform an ICE [RFC5245] offer/answer exchange between
   the nodes that are establishing the connection. In order to perform
   this exchange, the nodes need to first gather candidate addresses.
   Which nodes can be used to obtain reflexive address candidates and
   which ones can be used to obtain relayed candidates is defined by the
   peer protocol.

5.4. Lightweight Message Exchanges

   In some cases, nodes need to perform a lightweight query to another
   node (e.g., a request followed by a single response). In this
   situation, establishing a connection using the mechanisms in Section
   5.3 for a simple query would be an overkill. A better solution is to
   forward a HIP message through the overlay with the query and another
   one with the response to the query. The payload of such HIP packets
   is integrity protected [RFC6078].

Nodes in the overlay forward this HIP packet in a hop-by-hop fashion
according to the overlay's routing table towards its destination,
typically through the protected connections established between them.
Again, the overlay acts as a rendezvous server between the nodes
exchanging the messages.

5.5. HIP BONE Instantiation

As discussed in Section 5, HIP BONE is a generic framework that
allows using different peer protocols. A particular HIP BONE
instance uses a particular peer protocol. The details on how to
implement a HIP BONE using a given peer protocol need to be specified
in a, so-called, HIP BONE instance specification. A HIP BONE
instance specification needs to define, minimally:

o the peer protocol to be used,
o what kind of Node IDs are used and how they are derived,
o which peer protocol primitives trigger HIP messages, and
o how the overlay identifier is generated.

Additionally, a HIP BONE instance specification may need to specify
other details that are specific to the peer protocol used.

As an example, the HIP BONE instance specification for RELOAD
[P2PSIP-BASE] is specified in [HIP-RELOAD-INSTANCE].

The areas not covered by a particular HIP BONE instance specification
are specified by the peer protocol or elsewhere. These areas
include:

o the algorithm to create the overlay (e.g., a DHT),
o overlay maintenance functions,
o data storage and retrieval functions,
o the process for obtaining a Node ID,
o bootstrap function, and
o how to select STUN and TURN servers for the candidate address
   collection process in NAT traversal scenarios.

Note that the border between a HIP BONE instance specification and a
peer protocol specifications is fuzzy. Depending on how generic the
specification of a given peer protocol is, its associated HIP BONE
instance specification may need to specify more or less details.
Also, a HIP BONE instance specification may leave certain areas
unspecified in order to leave their configuration up to each
particular overlay.

6. Overlay HIP Parameters

   This section defines the generic format and protocol behavior for the
   Overlay Identifier and Overlay Time-to-Live (TTL) HIP parameters that
   can be used in HIP based overlay networks. HIP BONE instance
   specifications define the exact format and content of the Overlay
   Identifier parameter, the cases when the Overlay TTL parameter should
   be used, and any additional behavior for each packet.

6.1. Overlay Identifier

   To identify to which overlay network a HIP message belongs, all HIP
   messages that are sent via an overlay, or as a part of operations
   specific to a certain overlay, MUST contain an OVERLAY_ID parameter
   with the identifier of the corresponding overlay network. Instance
   specifications define how the identifier is generated for different
   types of overlay networks. The generation mechanism MUST be such
   that it is unlikely to generate the same identifier for two different
   overlay instances and any other means possible for preventing
   collisions SHOULD be used.

   The generic format of the OVERLAY_ID parameter is shown in Figure 8.
   Instance specifications define valid length for the parameter and how
   the identifier values are generated.

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |             Type              |             Length            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                          Identifier                          /
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   /                               |            Padding            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

   Type        4592
   Length      Length of the Identifier, in octets
   Identifier  The identifier value
   Padding     0-7 bytes of padding if needed
```
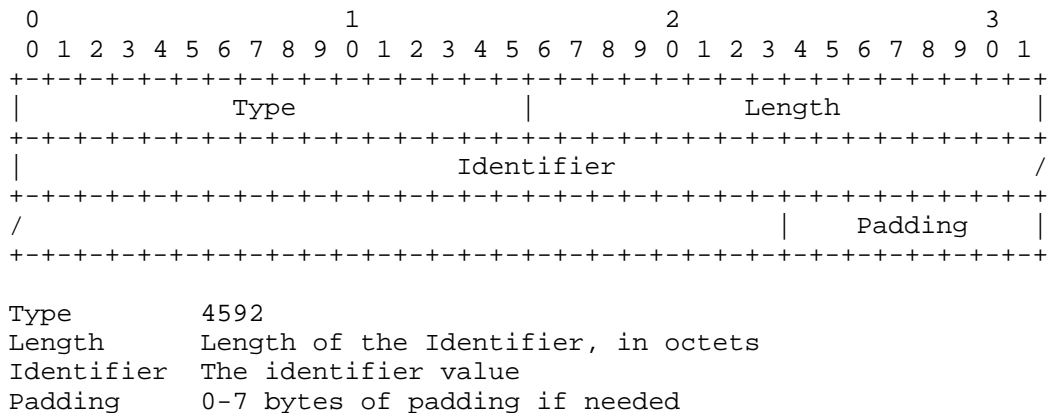
                  Figure 8: Format of the OVERLAY_ID Parameter
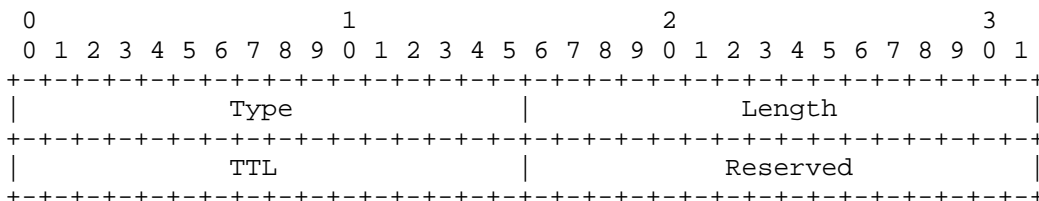
6.2. Overlay TTL

   HIP packets sent in an overlay network MAY contain an Overlay Time-
   to-live (OVERLAY_TTL) parameter whose TTL value is decremented on
   each overlay network hop. When a HIP host receives a HIP packet with

an OVERLAY_TTL parameter, and the host is not the final recipient of
the packet, it MUST decrement the TTL value in the parameter by one
before forwarding the packet.

If the TTL value in a received HIP packet is zero, and the receiving
host is not the final recipient, the packet MUST be dropped and the
host SHOULD send HIP Notify packet with NOTIFICATION error type
OVERLAY_TTL_EXCEEDED (value 70) to the sender of the original HIP
packet.

The Notification Data field for the OVERLAY_TTL_EXCEEDED
notifications SHOULD contain the HIP header and the TRANSACTION_ID
[RFC6078] parameter (if one exists) of the packet whose TTL was
exceeded.

Figure 9 shows the format of the OVERLAY_TTL parameter. The TTL
value is given as the number of overlay hops this packet has left and
it is encoded as an unsigned integer using network byte order.

```
     0                   1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |             Type              |             Length            |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |             TTL               |            Reserved           |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+


    Type          64011
    Length        4
    TTL           The Time-to-Live value
    Reserved      Reserved for future use
```

               Figure 9: Format of the OVERLAY_TTL Parameter

The type of the OVERLAY_TTL parameter is critical (as defined in
Section 5.2.1 of [RFC5201]) and therefore all the HIP nodes
forwarding a packet with this parameter MUST support it. If the
parameter is used in a scenario where the final recipient does not
support the parameter, the parameter SHOULD be removed before
forwarding the packet to the final recipient.

7. Security Considerations

   This document provides a high-level framework to build HIP-based
   overlays. The security properties of HIP and its extensions used in
   this framework are discussed in their respective specifications.
   Those security properties can be affected by the way HIP is used in a
   particular overlay. However, those properties are mostly affected by

the design decisions made to build a particular overlay; not so much
by the high-level framework specified in this document. Such design
decisions are typically documented in a HIP BONE instance
specification, which describes the security properties of the
resulting overlay.

8. Acknowledgements

   HIP BONE is based on ideas coming from conversations and discussions
   with a number of people in the HIP and P2PSIP communities. In
   particular, Philip Matthews, Eric Cooper, Joakim Koskela, Thomas
   Henderson, Bruce Lowekamp, and Miika Komu provided useful input on
   HIP BONE.

9. IANA Considerations

   This section is to be interpreted according to [RFC5226].

   This document updates the IANA Registry for HIP Parameter Types
   [RFC5201] by assigning HIP Parameter Type values for the new HIP
   Parameters OVERLAY_ID (defined in Section 6.1) and OVERLAY_TTL
   (defined in Section 6.2). This document also defines a new HIP
   Notify Message Type [RFC5201], OVERLAY_TTL_EXCEEDED in Section 6.2.
   This value is allocated in the error range.

10. References

10.1. Normative References

   [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
              Requirement Levels", BCP 14, RFC 2119, March 1997.

   [RFC4843] Nikander, P., Laganier, J., and F. Dupont, "An IPv6 Prefix
              for Overlay Routable Cryptographic Hash Identifiers
              (ORCHID)", RFC 4843, April 2007.

   [RFC5201] Moskowitz, R., Nikander, P., Jokela, P., Ed., and T.
              Henderson, "Host Identity Protocol", RFC 5201, April 2008.

   [RFC5202] Jokela, P., Moskowitz, R., and P. Nikander, "Using the
              Encapsulating Security Payload (ESP) Transport Format with
              the Host Identity Protocol (HIP)", RFC 5202, April 2008.

   [RFC5770] Komu, M., Henderson, T., Tschofenig, H., Melen, J., and A.
              Keranen, Ed., "Basic Host Identity Protocol (HIP)
              Extensions for Traversal of Network Address Translators",
              RFC 5770, April 2010.

   [RFC6078] Camarillo, G. and J. Melen, "Host Identity Protocol (HIP)
              Immediate Carriage and Conveyance of Upper-Layer Protocol
              Signaling (HICCUPS)", RFC 6078, January 2011.

10.2. Informative References

   [RFC4251] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH)
              Protocol Architecture", RFC 4251, January 2006.

   [RFC5204] Laganier, J. and L. Eggert, "Host Identity Protocol (HIP)
              Rendezvous Extension", RFC 5204, April 2008.

   [RFC5205] Nikander, P. and J. Laganier, "Host Identity Protocol
              (HIP) Domain Name System (DNS) Extensions", RFC 5205,
              April 2008.

   [RFC5206] Nikander, P., Henderson, T., Ed., Vogt, C., and J. Arkko,
              "End-Host Mobility and Multihoming with the Host Identity
              Protocol", RFC 5206, April 2008.

   [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an
              IANA Considerations Section in RFCs", BCP 26, RFC 5226,
              May 2008.

   [RFC5338] Henderson, T., Nikander, P., and M. Komu, "Using the Host
              Identity Protocol with Legacy Applications", RFC 5338,
              September 2008.

   [RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing,
              "Session Traversal Utilities for NAT (STUN)", RFC 5389,
              October 2008.

   [HIP-NATIVE-API]
              Komu, M. and T. Henderson, "Basic Socket Interface
              Extensions for Host Identity Protocol (HIP)", Work in
              Progress, January 2010.

   [RFC5245] Rosenberg, J., "Interactive Connectivity Establishment
              (ICE): A Protocol for Network Address Translator (NAT)
              Traversal for Offer/Answer Protocols", RFC 5245, April
              2010.

   [P2PSIP-BASE]
              Jennings, C., Lowekamp, B., Ed., Rescorla, E., Baset, S.,
              and H. Schulzrinne, "REsource LOcation And Discovery
              (RELOAD) Base Protocol", Work in Progress, November 2010.

   [HIP-RELOAD-INSTANCE]
              Keranen, A., Camarillo, G., and J. Maenpaa, "Host Identity
              Protocol-Based Overlay Networking Environment (HIP BONE)
              Instance Specification for REsource LOcation And Discovery
              (RELOAD)", Work in Progress, January 2011.

Authors' Addresses

   Gonzalo Camarillo
   Ericsson
   Hirsalantie 11
   Jorvas 02420
   Finland

   EMail: Gonzalo.Camarillo@ericsson.com

   Pekka Nikander
   Ericsson
   Hirsalantie 11
   Jorvas 02420
   Finland

   EMail: Pekka.Nikander@ericsson.com

   Jani Hautakorpi
   Ericsson
   Hirsalantie 11
   Jorvas 02420
   Finland

   EMail: Jani.Hautakorpi@ericsson.com

   Ari Keranen
   Ericsson
   Hirsalantie 11
   Jorvas 02420
   Finland

   EMail: Ari.Keranen@ericsson.com

   Alan Johnston
   Avaya
   St. Louis, MO 63124

   EMail: alan.b.johnston@gmail.com