Shim6: Level 3 Multihoming Shim Protocol for IPv6

Status of This Memo

   This document specifies an Internet standards track protocol for the
   Internet community, and requests discussion and suggestions for
   improvements. Please refer to the current edition of the "Internet
   Official Protocol Standards" (STD 1) for the standardization state
   and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Abstract

   This document defines the Shim6 protocol, a layer 3 shim for
   providing locator agility below the transport protocols, so that
   multihoming can be provided for IPv6 with failover and load-sharing
   properties, without assuming that a multihomed site will have a
   provider-independent IPv6 address prefix announced in the global IPv6
   routing table. The hosts in a site that has multiple provider-
   allocated IPv6 address prefixes will use the Shim6 protocol specified
   in this document to set up state with peer hosts so that the state
   can later be used to failover to a different locator pair, should the
   original one stop working.

Table of Contents

1. Introduction

     This document describes a layer 3 shim approach and protocol for
     providing locator agility below the transport protocols, so that
     multihoming can be provided for IPv6 with failover and load-sharing
     properties [11], without assuming that a multihomed site will have a
     provider-independent IPv6 address announced in the global IPv6
     routing table. The hosts in a site that has multiple provider-
     allocated IPv6 address prefixes will use the Shim6 protocol specified
     in this document to set up state with peer hosts so that the state
     can later be used to failover to a different locator pair, should the
     original one stop working (the term locator is defined in Section 2).

     The Shim6 protocol is a site-multihoming solution in the sense that
     it allows existing communication to continue when a site that has
     multiple connections to the Internet experiences an outage on a
     subset of these connections or further upstream. However, Shim6
     processing is performed in individual hosts rather than through site-
     wide mechanisms.

     We assume that redirection attacks are prevented using Hash-Based
     Addresses (HBA) as defined in [3].

The reachability and failure-detection mechanisms, including how a
new working locator pair is discovered after a failure, are specified
in RFC 5534 [4]. This document allocates message types and option
types for that sub-protocol, and leaves the specification of the
message and option formats, as well as the protocol behavior, to RFC
5534.

1.1. Goals

The goals for this approach are to:

o Preserve established communications in the presence of certain
   classes of failures, for example, TCP connections and UDP streams.

o Have minimal impact on upper-layer protocols in general and on
   transport protocols and applications in particular.

o Address the security threats in [15] through a combination of the
   HBA/CGA approach specified in RFC 5535 [3] and the techniques
   described in this document.

o Not require an extra roundtrip up front to set up shim-specific
   state. Instead, allow the upper-layer traffic (e.g., TCP) to flow
   as normal and defer the set up of the shim state until some number
   of packets have been exchanged.

o Take advantage of multiple locators/addresses for load spreading
   so that different sets of communication to a host (e.g., different
   connections) might use different locators of the host. Note that
   this might cause load to be spread unevenly; thus, we use the term
   "load spreading" instead of "load balancing". This capability
   might enable some forms of traffic engineering, but the details
   for traffic engineering, including what requirements can be
   satisfied, are not specified in this document, and form part of
   potential extensions to this protocol.

1.2. Non-Goals

The problem we are trying to solve is site multihoming, with the
ability to have the set of site prefixes change over time due to site
renumbering. Further, we assume that such changes to the set of
locator prefixes can be relatively slow and managed: slow enough to
allow updates to the DNS to propagate (since the protocol defined in
this document depends on the DNS to find the appropriate locator
sets). However, note that it is an explicit non-goal to make
communication survive a renumbering event (which causes all the
locators of a host to change to a new set of locators). This
proposal does not attempt to solve the related problem of host

mobility. However, it might turn out that the Shim6 protocol can be
a useful component for future host mobility solutions, e.g., for
route optimization.

Finally, this proposal also does not try to provide a new network-
level or transport-level identifier name space distinct from the
current IP address name space. Even though such a concept would be
useful to upper-layer protocols (ULPs) and applications, especially
if the management burden for such a name space was negligible and
there was an efficient yet secure mechanism to map from identifiers
to locators, such a name space isn't necessary (and furthermore
doesn't seem to help) to solve the multihoming problem.

The Shim6 proposal doesn't fully separate the identifier and locator
functions that have traditionally been overloaded in the IP address.
However, throughout this document the term "identifier" or, more
specifically, upper-layer identifier (ULID), refers to the
identifying function of an IPv6 address. "Locator" refers to the
network-layer routing and forwarding properties of an IPv6 address.

1.3. Locators as Upper-Layer Identifiers (ULID)

The approach described in this document does not introduce a new
identifier name space but instead uses the locator that is selected
in the initial contact with the remote peer as the preserved upper-
layer identifier (ULID). While there may be subsequent changes in
the selected network-level locators over time (in response to
failures in using the original locator), the upper-level protocol
stack elements will continue to use this upper-level identifier
without change.

This implies that the ULID selection is performed as today's default
address selection as specified in RFC 3484 [7]. Some extensions are
needed to RFC 3484 to try different source addresses, whether or not
the Shim6 protocol is used, as outlined in [9]. Underneath, and
transparently, the multihoming shim selects working locator pairs
with the initial locator pair being the ULID pair. If communication
subsequently fails, the shim can test and select alternate locators.
A subsequent section discusses the issues that arise when the
selected ULID is not initially working, which creates the need to
switch locators up front.

Using one of the locators as the ULID has certain benefits for
applications that have long-lived session state or that perform
callbacks or referrals, because both the Fully Qualified Domain Name
(FQDN) and the 128-bit ULID work as handles for the applications.

   However, using a single 128-bit ULID doesn't provide seamless
   communication when that locator is unreachable. See [18] for further
   discussion of the application implications.

   There has been some discussion of using non-routable addresses, such
   as Unique-Local Addresses (ULAs) [14], as ULIDs in a multihoming
   solution. While this document doesn't specify all aspects of this,
   it is believed that the approach can be extended to handle the non-
   routable address case. For example, the protocol already needs to
   handle ULIDs that are not initially reachable. Thus, the same
   mechanism can handle ULIDs that are permanently unreachable from
   outside their site. The issue becomes how to make the protocol
   perform well when the ULID is known a priori to be unreachable (e.g.,
   the ULID is a ULA), for instance, avoiding any timeout and retries in
   this case. In addition, one would need to understand how the ULAs
   would be entered in the DNS to avoid a performance impact on
   existing, non-Shim6-aware IPv6 hosts potentially trying to
   communicate to the (unreachable) ULA.

1.4. IP Multicast

   IP multicast requires that the IP Source Address field contain a
   topologically correct locator for the interface that is used to send
   the packet, since IP multicast routing uses both the source address
   and the destination group to determine where to forward the packet.
   In particular, IP multicast routing needs to be able to do the
   Reverse Path Forwarding (RPF) check. (This isn't much different than
   the situation with widely implemented ingress filtering [6] for
   unicast.)

   While in theory it would be possible to apply the shim re-mapping of
   the IP address fields between ULIDs and locators, the fact that all
   the multicast receivers would need to know the mapping to perform
   makes such an approach difficult in practice. Thus, it makes sense
   to have multicast ULPs operate directly on locators and not use the
   shim. This is quite a natural fit for protocols that use RTP [10],
   since RTP already has an explicit identifier in the form of the
   synchronization source (SSRC) field in the RTP headers. Thus, the
   actual IP address fields are not important to the application.

   In summary, IP multicast will not need the shim to remap the IP
   addresses.

   This doesn't prevent the receiver of multicast to change its
   locators, since the receiver is not explicitly identified; the
   destination address is a multicast address and not the unicast
   locator of the receiver.

1.5. Renumbering Implications

   As stated above, this approach does not try to make communication
   survive renumbering in the general case.

   When a host is renumbered, the effect is that one or more locators
   become invalid, and zero or more locators are added to the host's
   network interface. This means that the set of locators that is used
   in the shim will change, which the shim can handle as long as not all
   the original locators become invalid at the same time; the shim's
   ability to handle this also depends on the time that is required to
   update the DNS and for those updates to propagate.

   But IP addresses are also used as ULIDs, and making the communication
   survive locators becoming invalid can potentially cause some
   confusion at the upper layers. The fact that a ULID might be used
   with a different locator over time opens up the possibility that
   communication between two ULIDs might continue to work after one or
   both of those ULIDs are no longer reachable as locators, for example,
   due to a renumbering event. This opens up the possibility that the
   ULID (or at least the prefix on which it is based) may be reassigned
   to another site while it is still being used (with another locator)
   for existing communication.

   In the worst case, we could end up with two separate hosts using the
   same ULID while both of them are communicating with the same host.

   This potential source for confusion is avoided by requiring that any
   communication using a ULID MUST be terminated when the ULID becomes
   invalid (due to the underlying prefix becoming invalid). This
   behavior can be accomplished by explicitly discarding the shim state
   when the ULID becomes invalid. The context-recovery mechanism will
   then make the peer aware that the context is gone and that the ULID
   is no longer present at the same locator(s).

1.6. Placement of the Shim

```
                   -----------------------
                   | Transport Protocols |
                   -----------------------


              -------------- -------------      IP endpoint
              | Frag/reass | | Dest opts |      sub-layer
              -------------- -------------


                 --------------------
                 | Shim6 shim layer  |
                 --------------------


                     ------                    IP routing
                     | IP |                    sub-layer
                     ------
```
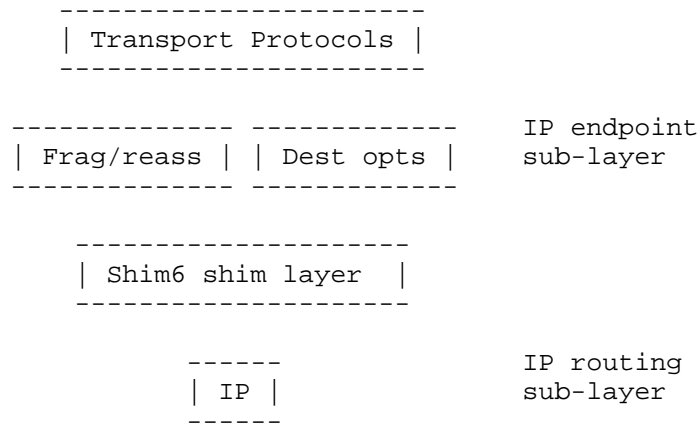
                   Figure 1: Protocol Stack

   The proposal uses a multihoming shim layer within the IP layer, i.e.,
   below the ULPs, as shown in Figure 1, in order to provide ULP
   independence. The multihoming shim layer behaves as if it is
   associated with an extension header, which would be placed after any
   routing-related headers in the packet (such as any hop-by-hop
   options). However, when the locator pair is the ULID pair, there is
   no data that needs to be carried in an extension header; thus, none
   is needed in that case.

   Layering the Fragmentation header above the multihoming shim makes
   reassembly robust in the case that there is broken multi-path routing
   that results in using different paths, hence potentially different
   source locators, for different fragments. Thus, the multihoming shim
   layer is placed between the IP endpoint sublayer (which handles
   fragmentation and reassembly) and the IP routing sublayer (which
   selects the next hop and interface to use for sending out packets).

   Applications and upper-layer protocols use ULIDs that the Shim6 layer
   maps to/from different locators. The Shim6 layer maintains state,
   called ULID-pair context, per ULID pair (that is, such state applies
   to all ULP connections between the ULID pair) in order to perform
   this mapping. The mapping is performed consistently at the sender
   and the receiver so that ULPs see packets that appear to be sent
   using ULIDs from end to end. This property is maintained even though
   the packets travel through the network containing locators in the IP
   address fields, and even though those locators may be changed by the
   transmitting Shim6 layer.

The context state is maintained per remote ULID, i.e., approximately
per peer host, and not at any finer granularity. In particular, the
context state is independent of the ULPs and any ULP connections.
However, the forking capability enables Shim6-aware ULPs to use more
than one locator pair at a time for a single ULID pair.

```
   ---------------------------        ---------------------------
  | Sender A                  |      | Receiver B                |
  |                           |      |                           |
  |    ULP                    |      |    ULP                    |
  |      | src ULID(A)=L1(A)  |      |      ^                    |
  |      | dst ULID(B)=L1(B)  |      |      | src ULID(A)=L1(A)  |
  |      v                    |      |      | dst ULID(B)=L1(B)  |
  |   multihoming shim        |      |   multihoming shim        |
  |      | src L2(A)          |      |      ^                    |
  |      | dst L3(B)          |      |      | src L2(A)          |
  |      v                    |      |      | dst L3(B)          |
  |      IP                   |      |      IP                   |
   ---------------------------        ---------------------------
          |                                  ^
           ------- cloud with some routers -------
```
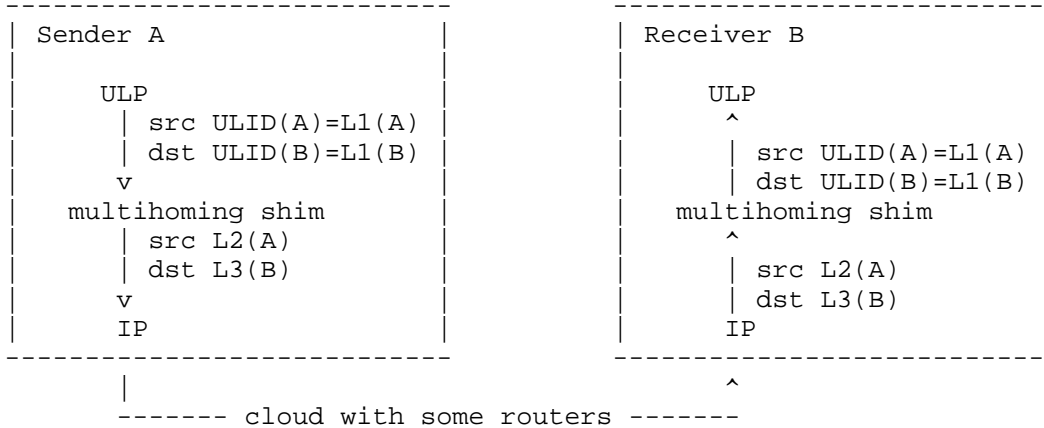
                Figure 2: Mapping with Changed Locators

The result of this consistent mapping is that there is no impact on
the ULPs. In particular, there is no impact on pseudo-header
checksums and connection identification.

Conceptually, one could view this approach as if both ULIDs and
locators are present in every packet, and as if a header-compression
mechanism is applied that removes the need for the ULIDs to be
carried in the packets once the compression state has been
established. In order for the receiver to re-create a packet with
the correct ULIDs, there is a need to include some "compression tag"
in the data packets. This serves to indicate the correct context to
use for decompression when the locator pair in the packet is
insufficient to uniquely identify the context.

There are different types of interactions between the Shim6 layer and
other protocols. Those interactions are influenced by the usage of
the addresses in these other protocols and the impact of the Shim6
mapping on these usages. A detailed analysis of the interactions of
different protocols, including the Stream Control Transmission
Protocol (SCTP), mobile IP (MIP), and Host Identity Protocol (HIP),
can be found in [19]. Moreover, some applications may need to have a
richer interaction with the Shim6 sublayer. In order to enable that,
an API [23] has been defined to enable greater control and
information exchange for those applications that need it.

1.7. Traffic Engineering

   At the time of this writing, it is not clear what requirements for
   traffic engineering make sense for the Shim6 protocol, since the
   requirements must both result in some useful behavior as well as be
   implementable using a host-to-host locator agility mechanism like
   Shim6.

   Inherent in a scalable multihoming mechanism that separates the
   locator function of the IP address from identifying function of the
   IP address is that each host ends up with multiple locators. This
   means that, at least for initial contact, it is the remote peer
   application (or layer working on its behalf) that needs to select an
   initial ULID, which automatically becomes the initial locator. In
   the case of Shim6, this is performed by applying RFC 3484 address
   selection.

   This is quite different than the common case of IPv4 multihoming
   where the site has a single IP address prefix, since in that case the
   peer performs no destination address selection.

   Thus, in "single prefix multihoming", the site (and in many cases its
   upstream ISPs) can use BGP to exert some control of the ingress path
   used to reach the site. This capability does not by itself exist in
   "multiple prefix multihoming" approaches such as Shim6. It is
   conceivable that extensions allowing site or provider guidance of
   host-based mechanisms could be developed. But it should be noted
   that traffic engineering via BGP, MPLS, or other similar techniques
   can still be applied for traffic on each individual prefix; Shim6
   does not remove the capability for this. It does provide some
   additional capabilities for hosts to choose between prefixes.

   These capabilities also carry some risk for non-optimal behaviour
   when more than one mechanism attempts to correct problems at the same
   time. However, it should be noted that this is not necessarily a
   situation brought about by Shim6. A more constrained form of this
   capability already exists in IPv6, itself, via its support of
   multiple prefixes and address-selection rules for starting new
   communications. Even IPv4 hosts with multiple interfaces may have
   limited capabilities to choose interfaces on which they communicate.
   Similarly, upper layers may choose different addresses.

   In general, it is expected that Shim6 is applicable in relatively
   small sites and individual hosts where BGP-style traffic engineering
   operations are unavailable, unlikely, or if run with provider-
   independent addressing, possibly even harmful, considering the growth
   rates in the global routing table.

The protocol provides a placeholder, in the form of the Locator
Preferences option, that can be used by hosts to express priority and
weight values for each locator. This option is merely a placeholder
when it comes to providing traffic engineering; in order to use this
in a large site, there would have to be a mechanism by which the host
can find out what preference values to use, either statically (e.g.,
some new DHCPv6 option) or dynamically.

Thus, traffic engineering is listed as a possible extension in
Appendix A.

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 [1].

2.1. Definitions

This document introduces the following terms:

upper-layer protocol (ULP)
                    A protocol layer immediately above IP. Examples
                    are transport protocols such as TCP and UDP;
                    control protocols such as ICMP; routing protocols
                    such as OSPF; and Internet or lower-layer
                    protocols being "tunneled" over (i.e.,
                    encapsulated in) IP, such as the Internet Packet
                    Exchange (IPX), AppleTalk, or IP itself.

    interface   A node's attachment to a link.

    address   An IP-layer name that both contains topological
                    significance and acts as a unique identifier for
                    an interface. 128 bits. This document only uses
                    the "address" term in the case where it isn't
                    specific whether it is a locator or an
                    identifier.

    locator   An IP-layer topological name for an interface or
                    a set of interfaces. 128 bits. The locators are
                    carried in the IP address fields as the packets
                    traverse the network.

    identifier   An IP-layer name for an IP-layer endpoint. The
                    transport endpoint name is a function of the
                    transport protocol and would typically include
                    the IP identifier plus a port number.

                    NOTE: This proposal does not specify any new form
                    of IP-layer identifier, but still separates the
                    identifying and locating properties of the IP
                    addresses.

   upper-layer identifier (ULID)
                    An IP address that has been selected for
                    communication with a peer to be used by the
                    upper-layer protocol. 128 bits. This is used for
                    pseudo-header checksum computation and connection
                    identification in the ULP. Different sets of
                    communication to a host (e.g., different
                    connections) might use different ULIDs in order
                    to enable load spreading.

                    Since the ULID is just one of the IP locators/
                    addresses of the node, there is no need for a
                    separate name space and allocation mechanisms.

   address field   The Source and Destination Address fields in the
                    IPv6 header. As IPv6 is currently specified,
                    these fields carry "addresses". If identifiers
                    and locators are separated, these fields will
                    contain locators for packets on the wire.

   FQDN   Fully Qualified Domain Name

   ULID-pair context The state that the multihoming shim maintains
                    between a pair of upper-layer identifiers. The
                    context is identified by a Context Tag for each
                    direction of the communication and also by a
                    ULID-pair and a Forked Instance Identifier (see
                    below).

   Context Tag   Each end of the context allocates a Context Tag
                    for the context. This is used to uniquely
                    associate both received control packets and Shim6
                    Payload Extension headers as belonging to the
                    context.

   current locator pair
                    Each end of the context has a current locator
                    pair that is used to send packets to the peer.
                    However, the two ends might use different current
                    locator pairs.

   default context   At the sending end, the shim uses the ULID pair
                       (passed down from the ULP) to find the context
                       for that pair. Thus, normally, a host can have
                       at most one context for a ULID pair. We call
                       this the "default context".

   context forking   A mechanism that allows ULPs that are aware of
                       multiple locators to use separate contexts for
                       the same ULID pair, in order to be able use
                       different locator pairs for different
                       communication to the same ULID. Context forking
                       causes more than just the default context to be
                       created for a ULID pair.

   Forked Instance Identifier (FII)
                       In order to handle context forking, a context is
                       identified by a ULID pair and a Forked Context
                       Identifier. The default context has an FII of
                       zero.

   initial contact   We use this term to refer to the pre-shim
                       communication when a ULP decides to start
                       communicating with a peer by sending and
                       receiving ULP packets. Typically, this would not
                       invoke any operations in the shim, since the shim
                       can defer the context establishment until some
                       arbitrary, later point in time.

   Hash-Based Addresses (HBA)
                       A form of IPv6 address where the interface ID is
                       derived from a cryptographic hash of all the
                       prefixes assigned to the host. See [3].

   Cryptographically Generated Addresses (CGA)
                       A form of IPv6 address where the interface ID is
                       derived from a cryptographic hash of the public
                       key. See [2].

   CGA Parameter Data Structure (PDS)
                       The information that CGA and HBA exchange in
                       order to inform the peer of how the interface ID
                       was computed. See [2] and [3].

2.2. Notational Conventions

   A, B, and C are hosts. X is a potentially malicious host.

   FQDN(A) is the Fully Qualified Domain Name for A.

   Ls(A) is the locator set for A, which consists of the locators L1(A),
   L2(A), ... Ln(A). The locator set is not ordered in any particular
   way other than maybe what is returned by the DNS. A host might form
   different locator sets containing different subnets of the host's IP
   addresses. This is necessary in some cases for security reasons.
   See Section 16.1.

   ULID(A) is an upper-layer identifier for A. In this proposal,
   ULID(A) is always one member of A's locator set.

   CT(A) is a Context Tag assigned by A.

   STATE (in uppercase) refers to the specific state of the state
   machine described in Section 6.2

2.3. Conceptual

   This document also makes use of internal conceptual variables to
   describe protocol behavior and external variables that an
   implementation must allow system administrators to change. The
   specific variable names, how their values change, and how their
   settings influence protocol behavior are provided to demonstrate
   protocol behavior. An implementation is not required to have them in
   the exact form described here, so long as its external behavior is
   consistent with that described in this document. See Section 6 for a
   description of the conceptual data structures.

3. Assumptions

   The design intent is to ensure that the Shim6 protocol is capable of
   handling path failures independently of the number of IP addresses
   (locators) available to the two communicating hosts, and
   independently of which host detects the failure condition.

   Consider, for example, the case in which both A and B have active
   Shim6 state and where A has only one locator while B has multiple
   locators. In this case, it might be that B is trying to send a
   packet to A, and has detected a failure condition with the current
   locator pair. Since B has multiple locators, it presumably has
   multiple ISPs, and (consequently) likely has alternate egress paths

toward A. B cannot vary the destination address (i.e., A's locator),
since A has only one locator. However, B may need to vary the source
address in order to ensure packet delivery.

In many cases, normal operation of IP routing may cause the packets
to follow a path towards the correct (currently operational) egress.
In some cases, it is possible that a path may be selected based on
the source address, implying that B will need to select a source
address corresponding to the currently operating egress. The details
of how routing can be accomplished is beyond the scope of this
document.

Also, when the site's ISPs perform ingress filtering based on packet
source addresses, Shim6 assumes that packets sent with different
source and destination combinations have a reasonable chance of
making it through the relevant ISP's ingress filters. This can be
accomplished in several ways (all outside the scope of this
document), such as having the ISPs relax their ingress filters or
selecting the egress such that it matches the IP source address
prefix. In the case that one egress path has failed but another is
operating correctly, it may be necessary for the packet's source
(node B in the previous paragraph) to select a source address that
corresponds to the operational egress, in order to pass the ISP's
ingress filters.

The Shim6 approach assumes that there are no IPv6-to-IPv6 NATs on the
paths, i.e., that the two ends can exchange their own notion of their
IPv6 addresses and that those addresses will also make sense to their
peer.

The security of the Shim6 protocol relies on the usage of Hash-Based
Addresses (HBA) [3] and/or Cryptographically Generated Addresses
(CGA) [2]. In the case that HBAs are used, all the addresses
assigned to the host that are included in the Shim6 protocol (either
as a locator or as a ULID) must be part of the same HBA set. In the
case that CGAs are used, the address used as ULID must be a CGA, but
the other addresses that are used as locators do not need to be
either CGAs or HBAs. It should be noted that it is perfectly
acceptable to run the Shim6 protocol between a host that has multiple
locators and another host that has a single IP address. In this
case, the address of the host with a single address does not need to
be an HBA or a CGA.

4. Protocol Overview

   The Shim6 protocol operates in several phases over time. The
   following sequence illustrates the concepts:

   o An application on host A decides to contact an application on host
      B using some upper-layer protocol. This results in the ULP on
      host A sending packets to host B. We call this the initial
      contact. Assuming the IP addresses selected by default address
      selection [7] and its extensions [9] work, then there is no action
      by the shim at this point in time. Any shim context establishment
      can be deferred until later.

   o Some heuristic on A or B (or both) determine that it is
      appropriate to pay the Shim6 overhead to make this host-to-host
      communication robust against locator failures. For instance, this
      heuristic might be that more than 50 packets have been sent or
      received, or that there was a timer expiration while active packet
      exchange was in place. This makes the shim initiate the 4-way,
      context-establishment exchange. The purpose of this heuristic is
      to avoid setting up a shim context when only a small number of
      packets is exchanged between two hosts.

      As a result of this exchange, both A and B will know a list of
      locators for each other.

      If the context-establishment exchange fails, the initiator will
      then know that the other end does not support Shim6, and will
      continue with standard (non-Shim6) behavior for the session.

   o Communication continues without any change for the ULP packets.
      In particular, there are no Shim6 Extension headers added to the
      ULP packets, since the ULID pair is the same as the locator pair.
      In addition, there might be some messages exchanged between the
      shim sublayers for (un)reachability detection.

   o At some point in time, something fails. Depending on the approach
      to reachability detection, there might be some advice from the
      ULP, or the shim (un)reachability detection might discover that
      there is a problem.

      At this point in time, one or both ends of the communication need
      to probe the different alternate locator pairs until a working
      pair is found, and then switch to using that locator pair.

   o Once a working alternative locator pair has been found, the shim
      will rewrite the packets on transmit and tag the packets with the
      Shim6 Payload Extension header, which contains the receiver's

      Context Tag. The receiver will use the Context Tag to find the
      context state, which will indicate which addresses to place in the
      IPv6 header before passing the packet up to the ULP. The result
      is that, from the perspective of the ULP, the packet passes
      unmodified end-to-end, even though the IP routing infrastructure
      sends the packet to a different locator.

   o The shim (un)reachability detection will monitor the new locator
      pair as it monitored the original locator pair, so that subsequent
      failures can be detected.

   o In addition to failures detected based on end-to-end observations,
      one endpoint might know for certain that one or more of its
      locators is not working. For instance, the network interface
      might have failed or gone down (at layer 2), or an IPv6 address
      might have become deprecated or invalid. In such cases, the host
      can signal its peer that trying this address is no longer
      recommended. This triggers something similar to a failure
      handling, and a new working locator pair must be found.

      The protocol also has the ability to express other forms of
      locator preferences. A change in any preference can be signaled
      to the peer, which will have made the peer record the new
      preferences. A change in the preferences might optionally make
      the peer want to use a different locator pair. In this case, the
      peer follows the same locator switching procedure as after a
      failure (by verifying that its peer is indeed present at the
      alternate locator, etc).

   o When the shim thinks that the context state is no longer used, it
      can garbage collect the state; there is no coordination necessary
      with the peer host before the state is removed. There is a
      recovery message defined to be able to signal when there is no
      context state, which can be used to detect and recover from both
      premature garbage collection as well as from complete state loss
      (crash and reboot) of a peer.

      The exact mechanism to determine when the context state is no
      longer used is implementation dependent. For example, an
      implementation might use the existence of ULP state (where known
      to the implementation) as an indication that the state is still
      used, combined with a timer (to handle ULP state that might not be
      known to the shim sublayer) to determine when the state is likely
      to no longer be used.

   NOTE 1: The ULP packets in Shim6 can be carried completely unmodified
   as long as the ULID pair is used as the locator pair. After a switch
   to a different locator pair, the packets are "tagged" with a Shim6

Extension header so that the receiver can always determine the
context to which they belong. This is accomplished by including an
8-octet Shim6 Payload Extension header before the (extension) headers
that are processed by the IP endpoint sublayer and ULPs. If,
subsequently, the original ULIDs are selected as the active locator
pair, then the tagging of packets with the Shim6 Extension header is
no longer necessary.

4.1. Context Tags

   A context between two hosts is actually a context between two ULIDs.
   The context is identified by a pair of Context Tags. Each end gets
   to allocate a Context Tag, and once the context is established, most
   Shim6 control messages contain the Context Tag that the receiver of
   the message allocated. Thus, at a minimum, the combination of <peer
   ULID, local ULID, local Context Tag> have to uniquely identify one
   context. But, since the Shim6 Payload Extension headers are
   demultiplexed without looking at the locators in the packet, the
   receiver will need to allocate Context Tags that are unique for all
   its contexts. The Context Tag is a 47-bit number (the largest that
   can fit in an 8-octet extension header), while preserving one bit to
   differentiate the Shim6 signaling messages from the Shim6 header
   included in data packets, allowing both to use the same protocol
   number.

   The mechanism for detecting a loss of context state at the peer
   assumes that the receiver can tell the packets that need locator
   rewriting, even after it has lost all state (e.g., due to a crash
   followed by a reboot). This is achieved because, after a rehoming
   event, the packets that need receive-side rewriting carry the Shim6
   Payload Extension header.

4.2. Context Forking

   It has been asserted that it will be important for future ULPs -- in
   particular, future transport protocols -- to be able to control which
   locator pairs are used for different communication. For instance,
   host A and host B might communicate using both Voice over IP (VoIP)
   traffic and ftp traffic, and those communications might benefit from
   using different locator pairs. However, the basic Shim6 mechanism
   uses a single current locator pair for each context; thus, a single
   context cannot accomplish this.

   For this reason, the Shim6 protocol supports the notion of context
   forking. This is a mechanism by which a ULP can specify (using some
   API not yet defined) that a context, e.g., the ULID pair <A1, B2>,

should be forked into two contexts. In this case, the forked-off
context will be assigned a non-zero Forked Instance Identifier, while
the default context has FII zero.

The Forked Instance Identifier (FII) is a 32-bit identifier that has
no semantics in the protocol other than being part of the tuple that
identifies the context. For example, a host might allocate FIIs as
sequential numbers for any given ULID pair.

No other special considerations are needed in the Shim6 protocol to
handle forked contexts.

Note that forking as specified does NOT allow A to be able to tell B
that certain traffic (a 5-tuple?) should be forked for the reverse
direction. The Shim6 forking mechanism as specified applies only to
the sending of ULP packets. If some ULP wants to fork for both
directions, it is up to the ULP to set this up and then instruct the
shim at each end to transmit using the forked context.

4.3. API Extensions

Several API extensions have been discussed for Shim6, but their
actual specification is out of scope for this document. The simplest
one would be to add a socket option to be able to have traffic bypass
the shim (not create any state and not use any state created by other
traffic). This could be an IPV6_DONTSHIM socket option. Such an
option would be useful for protocols, such as DNS, where the
application has its own failover mechanism (multiple NS records in
the case of DNS) and using the shim could potentially add extra
latency with no added benefits.

Some other API extensions are discussed in Appendix A. The actual
API extensions are defined in [23].

4.4. Securing Shim6

The mechanisms are secured using a combination of techniques:

o The HBA technique [3] for verifying the locators to prevent an
   attacker from redirecting the packet stream to somewhere else.

o Requiring a Reachability Probe+Reply (defined in [4]) before a new
   locator is used as the destination, in order to prevent 3rd party
   flooding attacks.

o The first message does not create any state on the responder.
   Essentially, a 3-way exchange is required before the responder
   creates any state. This means that a state-based DoS attack
   (trying to use up all memory on the responder) at least provides
   an IPv6 address that the attacker was using.

o The context-establishment messages use nonces to prevent replay
   attacks and to prevent off-path attackers from interfering with
   the establishment.

o Every control message of the Shim6 protocol, past the context
   establishment, carries the Context Tag assigned to the particular
   context. This implies that an attacker needs to discover that
   Context Tag before being able to spoof any Shim6 control message.
   Such discovery probably requires any potential attacker to be
   along the path in order to sniff the Context Tag value. The
   result is that through this technique, the Shim6 protocol is
   protected against off-path attackers.

4.5. Overview of Shim Control Messages

   The Shim6 context establishment is accomplished using four messages;
   I1, R1, I2, R2. Normally, they are sent in that order from initiator
   and responder, respectively. Should both ends attempt to set up
   context state at the same time (for the same ULID pair), then their
   I1 messages might cross in flight, and result in an immediate R2
   message. (The names of these messages are borrowed from HIP [20].)

   R1bis and I2bis messages are defined; they are used to recover a
   context after it has been lost. An R1bis message is sent when a
   Shim6 control or Shim6 Payload Extension header arrives and there is
   no matching context state at the receiver. When such a message is
   received, it will result in the re-creation of the Shim6 context
   using the I2bis and R2 messages.

   The peers' lists of locators are normally exchanged as part of the
   context-establishment exchange. But the set of locators might be
   dynamic. For this reason, there are Update Request and Update
   Acknowledgement messages as well as a Locator List option.

   Even when the list of locators is fixed, a host might determine that
   some preferences might have changed. For instance, it might
   determine that there is a locally visible failure that implies that
   some locator(s) are no longer usable. This uses a Locator
   Preferences option in the Update Request message.

   The mechanism for (un)reachability detection is called Forced
   Bidirectional Communication (FBD). FBD uses a Keepalive message
   which is sent when a host has received packets from its peer but has
   not yet sent any packets from its ULP to the peer. The message type
   is reserved in this document, but the message format and processing
   rules are specified in [4].

   In addition, when the context is established and there is a
   subsequent failure, there needs to be a way to probe the set of
   locator pairs to efficiently find a working pair. This document
   reserves a Probe message type, with the packet format and processing
   rules specified in [4].

   The above Probe and Keepalive messages assume we have an established
   ULID-pair context. However, communication might fail during the
   initial contact (that is, when the application or transport protocol
   is trying to set up some communication). This is handled using the
   mechanisms in the ULP to try different address pairs as specified in
   [7] and [9]. In future versions of the protocol, and with a richer
   API between the ULP and the shim, the shim might be able to help
   optimize discovering a working locator pair during initial contact.
   This is for further study.

4.6. Extension Header Order

   Since the shim is placed between the IP endpoint sublayer and the IP
   routing sublayer, the Shim header will be placed before any Endpoint
   Extension headers (Fragmentation headers, Destination Options header,
   AH, ESP) but after any routing-related headers (Hop-by-Hop Extensions
   header, Routing header, and a Destinations Options header, which
   precedes a Routing header). When tunneling is used, whether IP-in-IP
   tunneling or the special form of tunneling that Mobile IPv6 uses
   (with Home Address options and Routing header type 2), there is a
   choice whether the shim applies inside the tunnel or outside the
   tunnel, which affects the location of the Shim6 header.

   In most cases, IP-in-IP tunnels are used as a routing technique;
   thus, it makes sense to apply them on the locators, which means that
   the sender would insert the Shim6 header after any IP-in-IP
   encapsulation. This is what occurs naturally when routers apply IP-
   in-IP encapsulation. Thus, the packets would have:

   o Outer IP header

   o Inner IP header

o   Shim6 Extension header (if needed)

o   ULP

But the shim can also be used to create "shimmed tunnels", i.e.,
where an IP-in-IP tunnel uses the shim to be able to switch the
tunnel endpoint addresses between different locators. In such a
case, the packets would have:

o   Outer IP header

o   Shim6 Extension header (if needed)

o   Inner IP header

o   ULP

In any case, the receiver behavior is well-defined; a receiver
processes the Extension headers in order. However, the precise
interaction between Mobile IPv6 and Shim6 is for further study; it
might make sense to have Mobile IPv6 operate on locators as well,
meaning that the shim would be layered on top of the MIPv6 mechanism.

5. Message Formats

   The Shim6 messages are all carried using a new IP protocol number
   (140). The Shim6 messages have a common header (defined below) with
   some fixed fields, followed by type-specific fields.

   The Shim6 messages are structured as an IPv6 Extension header since
   the Shim6 Payload Extension header is used to carry the ULP packets
   after a locator switch. The Shim6 control messages use the same
   extension header formats so that a single "protocol number" needs to
   be allowed through firewalls in order for Shim6 to function across
   the firewall.

5.1. Common Shim6 Message Format

   The first 17 bits of the Shim6 header is common for the Shim6 Payload
   Extension header and for the control messages. It looks as follows:

```
  0                   1
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |  Next Header  |  Hdr Ext Len  |P|
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
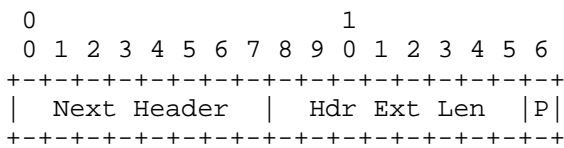
   Fields:

   Next Header: The payload that follows this header.

   Hdr Ext Len: 8-bit unsigned integer. Length of the Shim6 header in
                  8-octet units, not including the first 8 octets.

   P: A single bit to distinguish Shim6 Payload Extension
                  headers from control messages.

   Shim6 signaling packets may not be larger than 1280 bytes, including
   the IPv6 header and any intermediate headers between the IPv6 header
   and the Shim6 header. One way to meet this requirement is to omit
   part of the locator address information if, with this information
   included, the packet would become larger than 1280 bytes. Another
   option is to perform option engineering, dividing into different
   Shim6 messages the information to be transmitted. An implementation
   may impose administrative restrictions to avoid excessively large
   Shim6 packets, such as a limitation on the number of locators to be
   used.

5.2. Shim6 Payload Extension Header Format

   The Shim6 Payload Extension header is used to carry ULP packets where
   the receiver must replace the content of the Source and/or
   Destination fields in the IPv6 header before passing the packet to
   the ULP. Thus, this extension header is required when the locator
   pair that is used is not the same as the ULID pair.

```
    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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |  Next Header  |       0       |1|                             |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               |
   |                        Receiver Context Tag                   |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
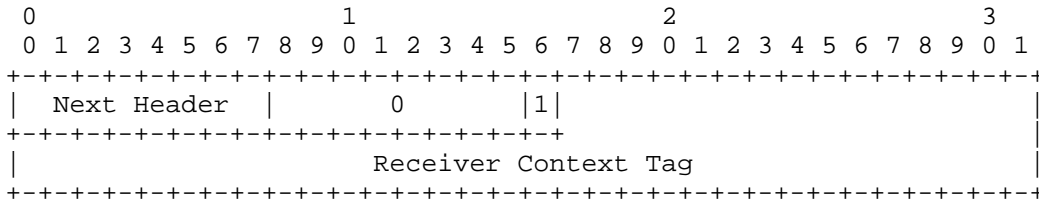
   Fields:

   Next Header: The payload that follows this header.

   Hdr Ext Len: 0 (since the header is 8 octets).

   P: Set to one. A single bit to distinguish this from the
                  Shim6 control messages.

     Receiver Context Tag:
                     47-bit unsigned integer. Allocated by the receiver to
                     identify the context.

5.3. Common Shim6 Control Header

   The common part of the header has a Next Header field and a Header
   Extension Length field that are consistent with the other IPv6
   Extension headers, even if the Next Header value is always "NO NEXT
   HEADER" for the control messages.

   The Shim6 headers must be a multiple of 8 octets; hence, the minimum
   size is 8 octets.

   The common Shim6 Control message header is as follows:

```
  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
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |  Next Header  |  Hdr Ext Len  |P|      Type     |Type-specific|S|
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |            Checksum           |                               |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               |
 |                         Type-specific format                  |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
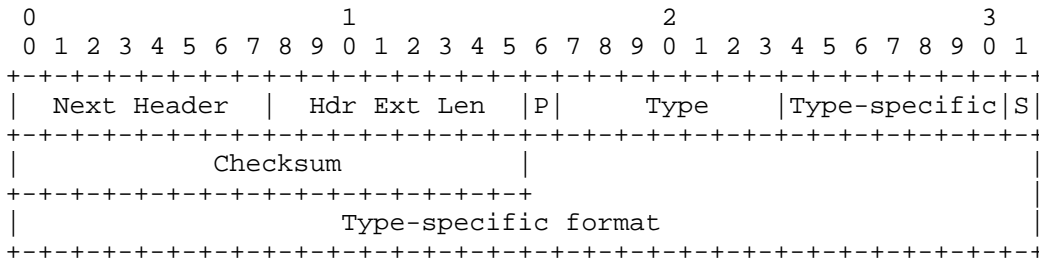
   Fields:

   Next Header: 8-bit selector. Normally set to NO_NXT_HDR (59).

   Hdr Ext Len: 8-bit unsigned integer. Length of the Shim6 header in
                  8-octet units, not including the first 8 octets.

   P: Set to zero. A single bit to distinguish this from
                  the Shim6 Payload Extension header.

   Type: 7-bit unsigned integer. Identifies the actual message
                  from the table below. Type codes 0-63 will not
                  trigger R1bis messages on a missing context, while
                  codes 64-127 will trigger R1bis.

   S: A single bit set to zero that allows Shim6 and HIP to
                  have a common header format yet still distinguishes
                  between Shim6 and HIP messages.

   Checksum: 16-bit unsigned integer. The checksum is the 16-bit
                  one's complement of the one's complement sum of the
                  entire Shim6 header message, starting with the Shim6

Next Header field and ending as indicated by the Hdr
Ext Len. Thus, when there is a payload following the
Shim6 header, the payload is NOT included in the Shim6
checksum. Note that, unlike protocols like ICMPv6,
there is no pseudo-header checksum part of the
checksum; this provides locator agility without having
to change the checksum.

Type-specific: Part of the message that is different for different
message types.

```
+------------+---------------------------------------------------+
| Type Value |                     Message                       |
+------------+---------------------------------------------------+
|     1      |  I1 (1st establishment message from the initiator) |
|     2      |  R1 (1st establishment message from the responder) |
|     3      |  I2 (2nd establishment message from the initiator) |
|     4      |  R2 (2nd establishment message from the responder) |
|     5      | R1bis (Reply to reference to non-existent context) |
|     6      |         I2bis (Reply to an R1bis message)          |
|    64      |                  Update Request                    |
|    65      |               Update Acknowledgement               |
|    66      |                    Keepalive                       |
|    67      |                  Probe Message                     |
|    68      |                  Error Message                     |
+------------+---------------------------------------------------+
```

Table 1

5.4. I1 Message Format

The I1 message is the first message in the context-establishment
exchange.
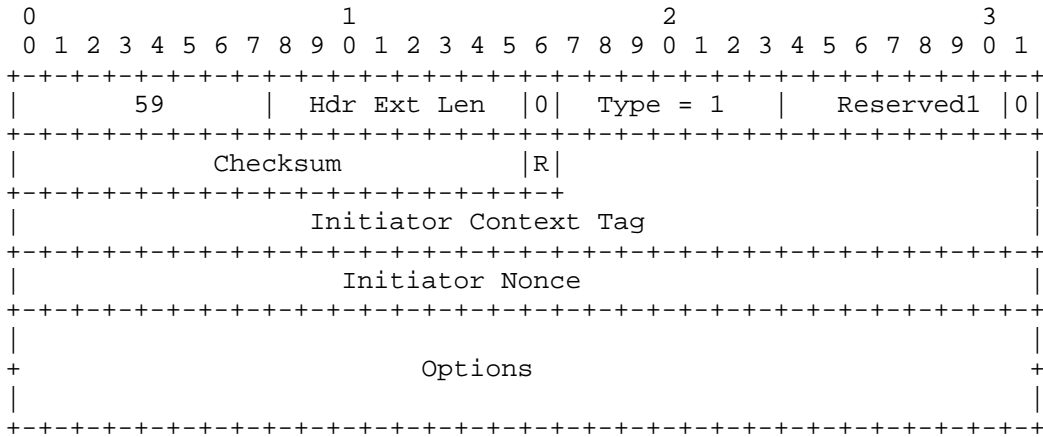
```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      59       |  Hdr Ext Len  |0|  Type = 1   |  Reserved1  |0|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Checksum           |R|                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               |
|                       Initiator Context Tag                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Initiator Nonce                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                            Options                            +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fields:

Next Header: NO_NXT_HDR (59).

Hdr Ext Len: At least 1, since the header is 16 octets when there
               are no options.

Type: 1

Reserved1:   7-bit field. Reserved for future use. Zero on
               transmit. MUST be ignored on receipt.

R: 1-bit field. Reserved for future use. Zero on
               transmit. MUST be ignored on receipt.

Initiator Context Tag:
               47-bit field. The Context Tag that the initiator has
               allocated for the context.

Initiator Nonce:
               32-bit unsigned integer. A random number picked by
               the initiator, which the responder will return in the
               R1 message.

The following options are defined for this message:

ULID pair: When the IPv6 source and destination addresses in the
               IPv6 header does not match the ULID pair, this option
               MUST be included. An example of this is when
               recovering from a lost context.

   Forked Instance Identifier:
                 When another instance of an existent context with the
                 same ULID pair is being created, a Forked Instance
                 Identifier option MUST be included to distinguish this
                 new instance from the existent one.

   Future protocol extensions might define additional options for this
   message. The C-bit in the option format defines how such a new
   option will be handled by an implementation. See Section 5.15.

5.5. R1 Message Format

   The R1 message is the second message in the context-establishment
   exchange. The responder sends this in response to an I1 message,
   without creating any state specific to the initiator.

```
     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
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |      59       |  Hdr Ext Len  |0| Type = 2    |  Reserved1  |0|
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |           Checksum            |          Reserved2           |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                        Initiator Nonce                       |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                        Responder Nonce                       |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                                                              |
    +                           Options                            +
    |                                                              |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   Fields:

   Next Header: NO_NXT_HDR (59).

   Hdr Ext Len: At least 1, since the header is 16 octets when there
                  are no options.

   Type: 2

   Reserved1:   7-bit field. Reserved for future use. Zero on
                  transmit. MUST be ignored on receipt.

   Reserved2:   16-bit field. Reserved for future use. Zero on
                  transmit. MUST be ignored on receipt.

Initiator Nonce:
                32-bit unsigned integer. Copied from the I1 message.

Responder Nonce:
                32-bit unsigned integer. A number picked by the
                responder, which the initiator will return in the I2
                message.

The following options are defined for this message:

Responder Validator:
                Variable length option. This option MUST be included
                in the R1 message. Typically, it contains a hash
                generated by the responder, which the responder uses
                together with the Responder Nonce value to verify that
                an I2 message is indeed sent in response to an R1
                message, and that the parameters in the I2 message are
                the same as those in the I1 message.

Future protocol extensions might define additional options for this
message. The C-bit in the option format defines how such a new
option will be handled by an implementation. See Section 5.15.

5.6. I2 Message Format

The I2 message is the third message in the context-establishment
exchange. The initiator sends this in response to an R1 message,
after checking the Initiator Nonce, etc.

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      59       | Hdr Ext Len |0| Type = 3    |   Reserved1 |0|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum          |R|                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                                 |
|                    Initiator Context Tag                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Initiator Nonce                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Responder Nonce                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Reserved2                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                         Options                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
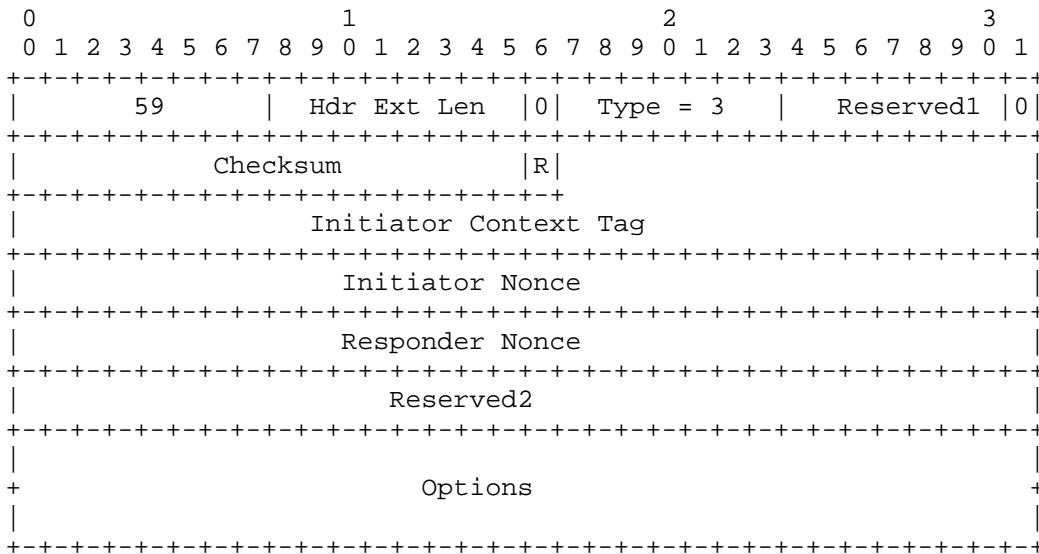
   Fields:

   Next Header: NO_NXT_HDR (59).

   Hdr Ext Len: At least 2, since the header is 24 octets when there
                   are no options.

   Type: 3

   Reserved1:   7-bit field. Reserved for future use. Zero on
                   transmit. MUST be ignored on receipt.

   R: 1-bit field. Reserved for future use. Zero on
                   transmit. MUST be ignored on receipt.

   Initiator Context Tag:
                   47-bit field. The Context Tag that the initiator has
                   allocated for the context.

   Initiator Nonce:
                   32-bit unsigned integer. A random number picked by
                   the initiator, which the responder will return in the
                   R2 message.

   Responder Nonce:
                   32-bit unsigned integer. Copied from the R1 message.

   Reserved2:   32-bit field. Reserved for future use. Zero on
                   transmit. MUST be ignored on receipt. (Needed to
                   make the options start on a multiple of 8 octet
                   boundary.)

   The following options are defined for this message:

   Responder Validator:
                   Variable length option. This option MUST be included
                   in the I2 message and MUST be generated by copying the
                   Responder Validator option received in the R1 message.

   ULID pair: When the IPv6 source and destination addresses in the
                   IPv6 header do not match the ULID pair, this option
                   MUST be included. An example of this is when
                   recovering from a lost context.
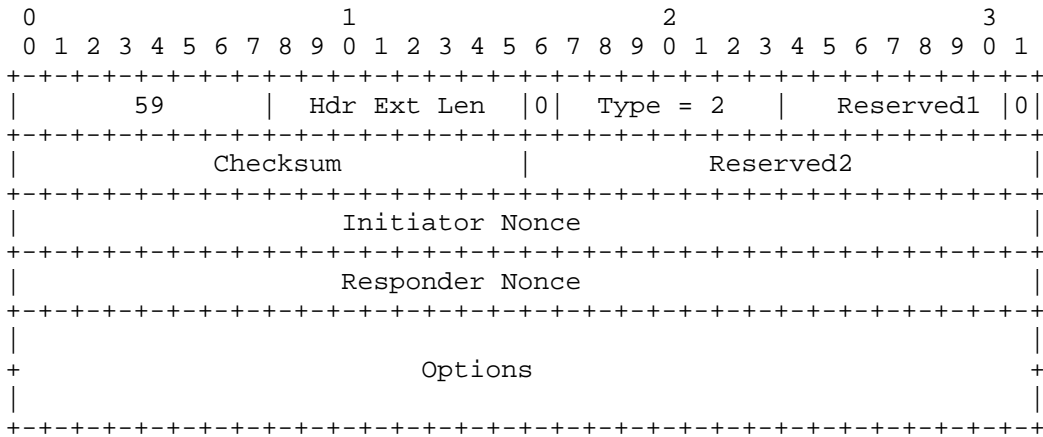
Forked Instance Identifier:
                When another instance of an existent context with the
                same ULID pair is being created, a Forked Instance
                Identifier option MUST be included to distinguish this
                new instance from the existent one.

Locator List: Optionally sent when the initiator immediately wants
                to tell the responder its list of locators. When it
                is sent, the necessary HBA/CGA information for
                verifying the locator list MUST also be included.

Locator Preferences:
                Optionally sent when the locators don't all have equal
                preference.

CGA Parameter Data Structure:
                This option MUST be included in the I2 message when
                the locator list is included so the receiver can
                verify the locator list.

CGA Signature: This option MUST be included in the I2 message when
                some of the locators in the list use CGA (and not HBA)
                for verification.

   Future protocol extensions might define additional options for this
   message. The C-bit in the option format defines how such a new
   option will be handled by an implementation. See Section 5.15.

5.7. R2 Message Format

   The R2 message is the fourth message in the context-establishment
   exchange. The responder sends this in response to an I2 message.
   The R2 message is also used when both hosts send I1 messages at the
   same time and the I1 messages cross in flight.
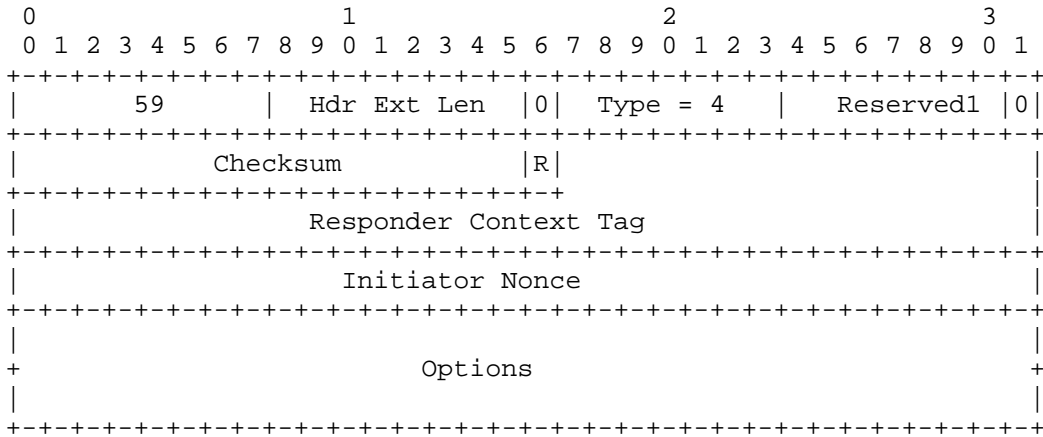
```
  0                   1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |       59      | Hdr Ext Len  |0| Type = 4  |  Reserved1  |0|
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |            Checksum          |R|                             |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                             |
 |                     Responder Context Tag                     |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                       Initiator Nonce                         |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                                                               |
 +                           Options                             +
 |                                                               |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fields:

Next Header: NO_NXT_HDR (59).

Hdr Ext Len: At least 1, since the header is 16 octets when there
               are no options.

Type: 4

Reserved1:   7-bit field. Reserved for future use. Zero on
               transmit. MUST be ignored on receipt.

R: 1-bit field. Reserved for future use. Zero on
               transmit. MUST be ignored on receipt.

Responder Context Tag:
               47-bit field. The Context Tag that the responder has
               allocated for the context.

Initiator Nonce:
               32-bit unsigned integer. Copied from the I2 message.

The following options are defined for this message:

Locator List: Optionally sent when the responder immediately wants
               to tell the initiator its list of locators. When it
               is sent, the necessary HBA/CGA information for
               verifying the locator list MUST also be included.

Locator Preferences:
               Optionally sent when the locators don't all have equal
               preference.

   CGA Parameter Data Structure:
                 Included when the locator list is included so the
                 receiver can verify the locator list.

   CGA Signature: Included when some of the locators in the list use CGA
                 (and not HBA) for verification.

   Future protocol extensions might define additional options for this
   message. The C-bit in the option format defines how such a new
   option will be handled by an implementation. See Section 5.15.

5.8. R1bis Message Format

   Should a host receive a packet with a Shim6 Payload Extension header
   or Shim6 control message with type code 64-127 (such as an Update or
   Probe message), and the host does not have any context state for the
   received Context Tag, then it will generate a R1bis message.

   This message allows the sender of the packet referring to the non-
   existent context to re-establish the context with a reduced context-
   establishment exchange. Upon the reception of the R1bis message, the
   receiver can proceed with re-establishing the lost context by
   directly sending an I2bis message.
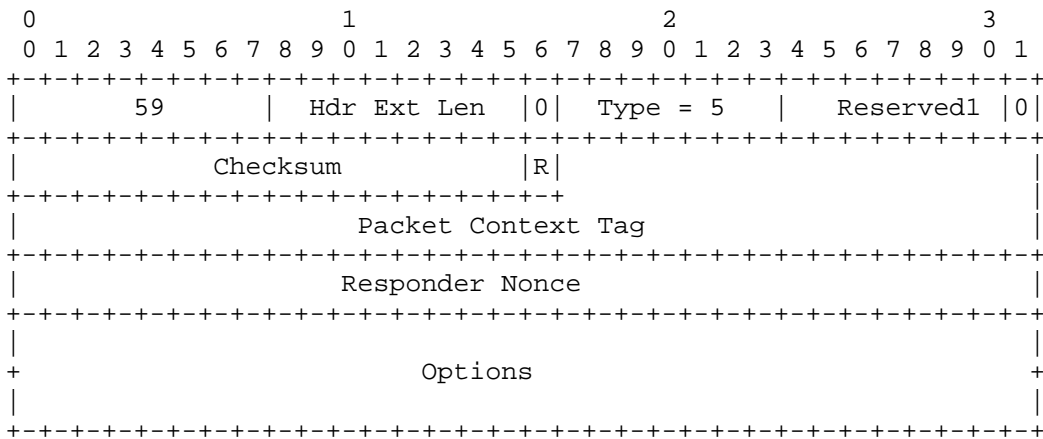
```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |      59       | Hdr Ext Len   |0| Type = 5    |   Reserved1 |0|
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |           Checksum            |R|                             |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                             |
   |                       Packet Context Tag                      |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                       Responder Nonce                         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   +                           Options                             +
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   Fields:

   Next Header: NO_NXT_HDR (59).

   Hdr Ext Len: At least 1, since the header is 16 octets when there
                  are no options.

   Type: 5

   Reserved1:   7-bit field. Reserved for future use. Zero on
                  transmit. MUST be ignored on receipt.

   R: 1-bit field. Reserved for future use. Zero on
                  transmit. MUST be ignored on receipt.

   Packet Context Tag:
                  47-bit unsigned integer. The Context Tag contained in
                  the received packet that triggered the generation of
                  the R1bis message.

   Responder Nonce:
                  32-bit unsigned integer. A number picked by the
                  responder which the initiator will return in the I2bis
                  message.

   The following options are defined for this message:

   Responder Validator:
                  Variable length option. Typically, a hash generated
                  by the responder, which the responder uses together
                  with the Responder Nonce value to verify that an I2bis
                  message is indeed sent in response to an R1bis
                  message.

   Future protocol extensions might define additional options for this
   message. The C-bit in the option format defines how such a new
   option will be handled by an implementation. See Section 5.15.

5.9. I2bis Message Format

   The I2bis message is the third message in the context-recovery
   exchange. This is sent in response to an R1bis message, after
   checking that the R1bis message refers to an existing context, etc.

```
      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
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |      59       |  Hdr Ext Len  |0| Type = 6  |   Reserved1 |0|
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |           Checksum            |R|                             |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                             |
     |                      Initiator Context Tag                    |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                        Initiator Nonce                        |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                        Responder Nonce                        |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                          Reserved2                            |
     |                               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                               |                               |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               |
     |                        Packet Context Tag                     |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                                                               |
     +                           Options                             +
     |                                                               |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
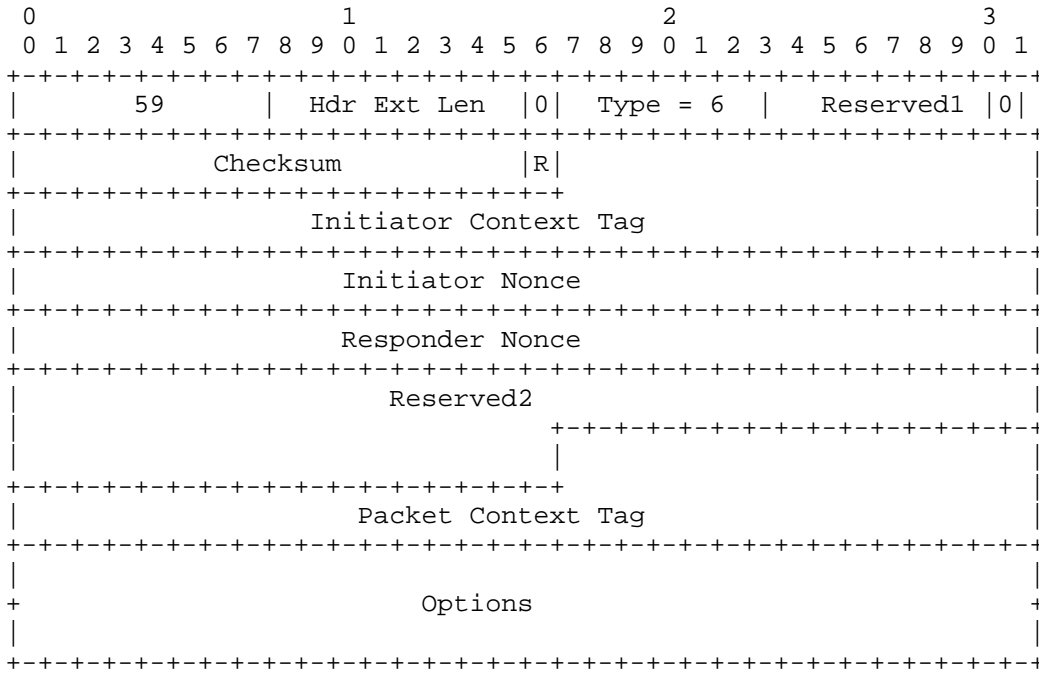
   Fields:

   Next Header: NO_NXT_HDR (59).

   Hdr Ext Len: At least 3, since the header is 32 octets when there
                   are no options.

   Type: 6

   Reserved1:   7-bit field. Reserved for future use. Zero on
                   transmit. MUST be ignored on receipt.

   R: 1-bit field. Reserved for future use. Zero on
                   transmit. MUST be ignored on receipt.

   Initiator Context Tag:
                   47-bit field. The Context Tag that the initiator has
                   allocated for the context.

   Initiator Nonce:
                   32-bit unsigned integer. A random number picked by
                   the initiator, which the responder will return in the
                   R2 message.

Responder Nonce:
            32-bit unsigned integer. Copied from the R1bis
            message.

Reserved2:   49-bit field. Reserved for future use. Zero on
                transmit. MUST be ignored on receipt. (Note that 17
                bits are not sufficient since the options need to
                start on a multiple-of-8-octet boundary.)

Packet Context Tag:
            47-bit unsigned integer. Copied from the Packet
            Context Tag field contained in the received R1bis.

The following options are defined for this message:

Responder Validator:
            Variable length option. Just a copy of the Responder
            Validator option in the R1bis message.

ULID pair: When the IPv6 source and destination addresses in the
                IPv6 header do not match the ULID pair, this option
                MUST be included.

Forked Instance Identifier:
            When another instance of an existent context with the
            same ULID pair is being created, a Forked Instance
            Identifier option is included to distinguish this new
            instance from the existent one.

Locator List: Optionally sent when the initiator immediately wants
                to tell the responder its list of locators. When it
                is sent, the necessary HBA/CGA information for
                verifying the locator list MUST also be included.

Locator Preferences:
            Optionally sent when the locators don't all have equal
            preference.

CGA Parameter Data Structure:
            Included when the locator list is included so the
            receiver can verify the locator list.
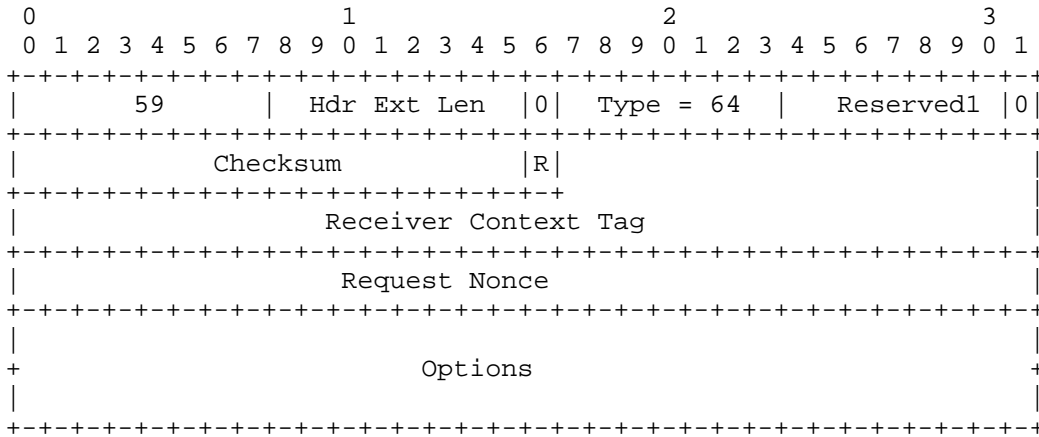
CGA Signature: Included when some of the locators in the list use CGA
                (and not HBA) for verification.

Future protocol extensions might define additional options for this
message. The C-bit in the option format defines how such a new
option will be handled by an implementation. See Section 5.15.

5.10. Update Request Message Format

   The Update Request message is used to update either the list of
   locators, the locator preferences, or both. When the list of
   locators is updated, the message also contains the option(s)
   necessary for HBA/CGA to secure this. The basic sanity check that
   prevents off-path attackers from generating bogus updates is the
   Context Tag in the message.

   The Update Request message contains options (the Locator List and the
   Locator Preferences) that, when included, completely replace the
   previous locator list and locator preferences, respectively. Thus,
   there is no mechanism to just send deltas to the locator list.

```
      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
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |       59      |  Hdr Ext Len  |0|  Type = 64  |  Reserved1  |0|
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |            Checksum           |R|                             |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                             |
     |                     Receiver Context Tag                      |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                        Request Nonce                          |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                                                               |
     +                           Options                             +
     |                                                               |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   Fields:

   Next Header: NO_NXT_HDR (59).

   Hdr Ext Len: At least 1, since the header is 16 octets when there
                   are no options.

   Type: 64

   Reserved1:   7-bit field. Reserved for future use. Zero on
                   transmit. MUST be ignored on receipt.

   R: 1-bit field. Reserved for future use. Zero on
                   transmit. MUST be ignored on receipt.

   Receiver Context Tag:
                   47-bit field. The Context Tag that the receiver has
                   allocated for the context.

Request Nonce:
               32-bit unsigned integer. A random number picked by
               the initiator, which the peer will return in the
               Update Acknowledgement message.

The following options are defined for this message:

Locator List: The list of the sender's (new) locators. The locators
               might be unchanged and only the preferences have
               changed.

Locator Preferences:
               Optionally sent when the locators don't all have equal
               preference.

CGA Parameter Data Structure (PDS):
               Included when the locator list is included and the PDS
               was not included in the I2/ I2bis/R2 messages, so the
               receiver can verify the locator list.

CGA Signature: Included when some of the locators in the list use CGA
               (and not HBA) for verification.

Future protocol extensions might define additional options for this
message. The C-bit in the option format defines how such a new
option will be handled by an implementation. See Section 5.15.

5.11. Update Acknowledgement Message Format

This message is sent in response to an Update Request message. It
implies that the Update Request has been received and that any new
locators in the Update Request can now be used as the source locators
of packets. But it does not imply that the (new) locators have been
verified to be used as a destination, since the host might defer the
verification of a locator until it sees a need to use a locator as
the destination.
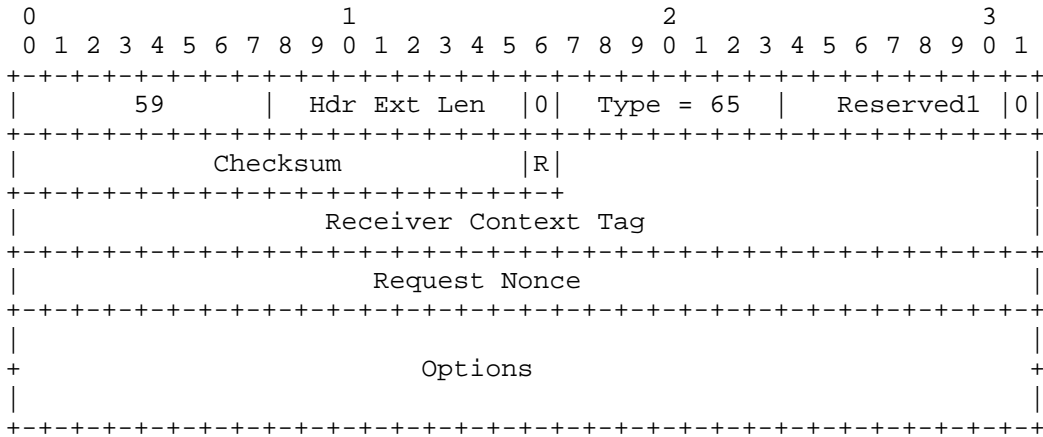
```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      59       | Hdr Ext Len  |0| Type = 65   |  Reserved1  |0|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |R|                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               |
|                     Receiver Context Tag                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Request Nonce                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                          Options                              +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fields:

Next Header: NO_NXT_HDR (59).

Hdr Ext Len: At least 1, since the header is 16 octets when there
                are no options.

Type: 65

Reserved1:   7-bit field. Reserved for future use. Zero on
                transmit. MUST be ignored on receipt.

R: 1-bit field. Reserved for future use. Zero on
                transmit. MUST be ignored on receipt.

Receiver Context Tag:
                47-bit field. The Context Tag the receiver has
                allocated for the context.

Request Nonce: 32-bit unsigned integer. Copied from the Update
                Request message.

No options are currently defined for this message.

Future protocol extensions might define additional options for this
message. The C-bit in the option format defines how such a new
option will be handled by an implementation. See Section 5.15.

5.12. Keepalive Message Format

   This message format is defined in [4].

   The message is used to ensure that when a peer is sending ULP packets
   on a context, it always receives some packets in the reverse
   direction. When the ULP is sending bidirectional traffic, no extra
   packets need to be inserted. But for a unidirectional ULP traffic
   pattern, the shim will send back some Keepalive messages when it is
   receiving ULP packets.

5.13. Probe Message Format

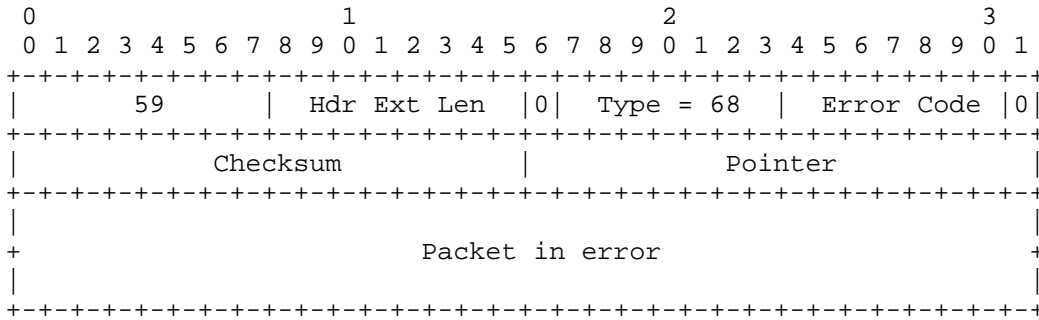   This message and its semantics are defined in [4].

   The goal of this mechanism is to test whether or not locator pairs
   work in the general case. In particular, this mechanism is to be
   able to handle the case when one locator pair works from A to B and
   another locator pair works from B to A, but there is no locator pair
   that works in both directions. The protocol mechanism is that, as A
   is sending Probe messages to B, B will observe which locator pairs it
   has received and report that back in Probe messages it sends to A.

5.14. Error Message Format

   The Error message is generated by a Shim6 receiver upon the reception
   of a Shim6 message containing critical information that cannot be
   processed properly.

   In the case that a Shim6 node receives a Shim6 packet that contains
   information that is critical for the Shim6 protocol and that is not
   supported by the receiver, it sends an Error Message back to the
   originator of the Shim6 message. The Error message is
   unacknowledged.

   In addition, Shim6 Error messages defined in this section can be used
   to identify problems with Shim6 implementations. In order to do so,
   a range of Error Code types is reserved for that purpose. In
   particular, implementations may generate Shim6 Error messages with
   Code types in that range, instead of silently discarding Shim6
   packets during the debugging process.

```
    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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |      59       | Hdr Ext Len  |0| Type = 68   | Error Code  |0|
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |           Checksum            |             Pointer           |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   +                        Packet in error                       +
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   Fields:

   Next Header: NO_NXT_HDR (59).

   Hdr Ext Len: At least 1, since the header is 16 octets. Depends on
                  the specific Error Data.

   Type: 68

   Error Code: 7-bit field describing the error that generated the
                  Error message. See Error Code list below.

   Pointer: 16-bit field. Identifies the octet offset within the
                  invoking packet where the error was detected.

   Packet in error:
                  As much of invoking packet as possible without the
                  Error message packet exceeding the minimum IPv6 MTU.

   The following Error Codes are defined:

```
   +---------+-----------------------------------------------------+
   |  Code   |                    Description                       |
   |  Value  |                                                     |
   +---------+-----------------------------------------------------+
   |    0    |             Unknown Shim6 message type              |
   |    1    |           Critical option not recognized            |
   |    2    |  Locator verification method failed (Pointer to the |
   |         |       inconsistent verification method octet)       |
   |    3    |       Locator List Generation number out of sync.   |
   |    4    | Error in the number of locators in a Locator Preference |
   |         |                        option                       |
   | 120-127 |            Reserved for debugging purposes           |
   +---------+-----------------------------------------------------+
```

                                Table 2

5.15. Option Formats

   The format of the options is a snapshot of the current HIP option
   format [20]. However, there is no intention to track any changes to
   the HIP option format, nor is there an intent to use the same name
   space for the option type values. But using the same format will
   hopefully make it easier to import HIP capabilities into Shim6 as
   extensions to Shim6, should this turn out to be useful.

   All of the TLV parameters have a length (including Type and Length
   fields) that is a multiple of 8 bytes. When needed, padding MUST be
   added to the end of the parameter so that the total length becomes a
   multiple of 8 bytes. This rule ensures proper alignment of data. If
   padding is added, the Length field MUST NOT include the padding. Any
   added padding bytes MUST be zeroed by the sender, and their values
   SHOULD NOT be checked by the receiver.

   Consequently, the Length field indicates the length of the Contents
   field (in bytes). The total length of the TLV parameter (including
   Type, Length, Contents, and Padding) is related to the Length field
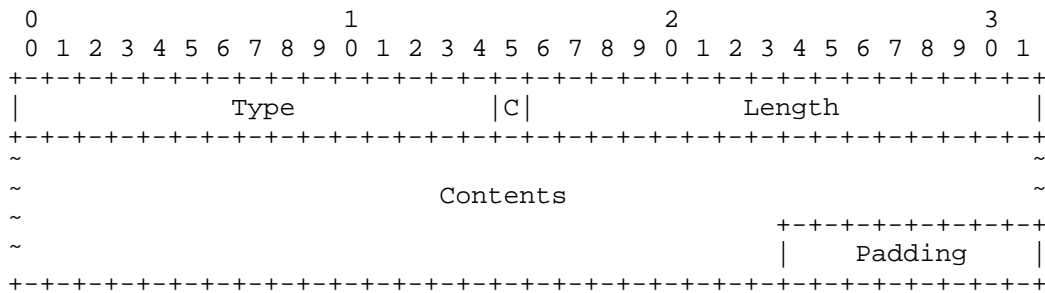   according to the following formula:

   Total Length = 11 + Length - (Length + 3) mod 8;

   The total length of the option is the smallest multiple of 8 bytes
   that allows for the 4 bytes of the Option header and option, itself.
   The amount of padding required can be calculated as follows:

   padding = 7 - ((Length + 3) mod 8)

   And:

   Total Length = 4 + Length + padding

```
 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              |C|          Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                                                               ~
~                           Contents                           ~
~                                       +-+-+-+-+-+-+-+-+
~                                       |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fields:

Type: 15-bit identifier of the type of option. The options
                defined in this document are below.

C: Critical. One, if this parameter is critical and MUST
                be recognized by the recipient; zero otherwise. An
                implementation might view the C-bit as part of the
                Type field by multiplying the type values in this
                specification by two.

Length: Length of the Contents, in bytes.

Contents: Parameter-specific, defined by Type.

Padding: Padding, 0-7 bytes, added if needed.

```
         +------+------------------------------+
         | Type |         Option Name          |
         +------+------------------------------+
         |   1  |      Responder Validator     |
         |   2  |         Locator List         |
         |   3  |      Locator Preferences     |
         |   4  | CGA Parameter Data Structure |
         |   5  |         CGA Signature        |
         |   6  |           ULID Pair          |
         |   7  |   Forked Instance Identifier |
         |  10  |    Keepalive Timeout Option  |
         +------+------------------------------+
```

                              Table 3

   Future protocol extensions might define additional options for the
   Shim6 messages. The C-bit in the option format defines how such a
   new option will be handled by an implementation.

   If a host receives an option that it does not understand (an option
   that was defined in some future extension to this protocol) or that
   is not listed as a valid option for the different message types
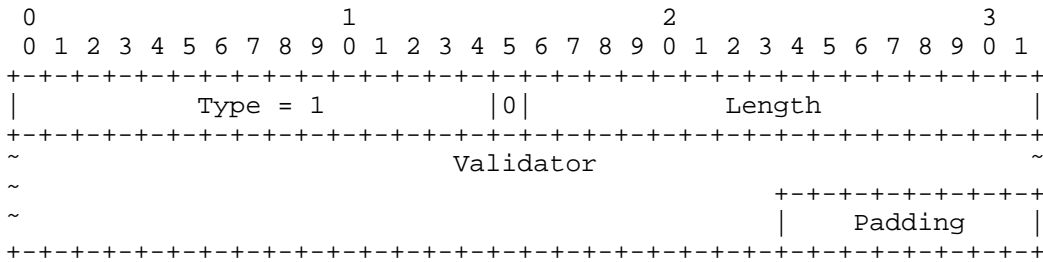   above, then the Critical bit in the option determines the outcome.

   o If C=0, then the option is silently ignored, and the rest of the
      message is processed.

   o If C=1, then the host SHOULD send back a Shim6 Error message with
      Error Code=1, with the Pointer field referencing the first octet
      in the Option Type field. When C=1, the rest of the message MUST
      NOT be processed.

5.15.1. Responder Validator Option Format

   The responder can choose exactly what input is used to compute the
   validator and what one-way function (such as MD5 or SHA1) it uses, as
   long as the responder can check that the validator it receives back
   in the I2 or I2bis message is indeed one that:

   1) computed,

   2) computed for the particular context, and

   3) isn't a replayed I2/I2bis message.

   Some suggestions on how to generate the validators are captured in
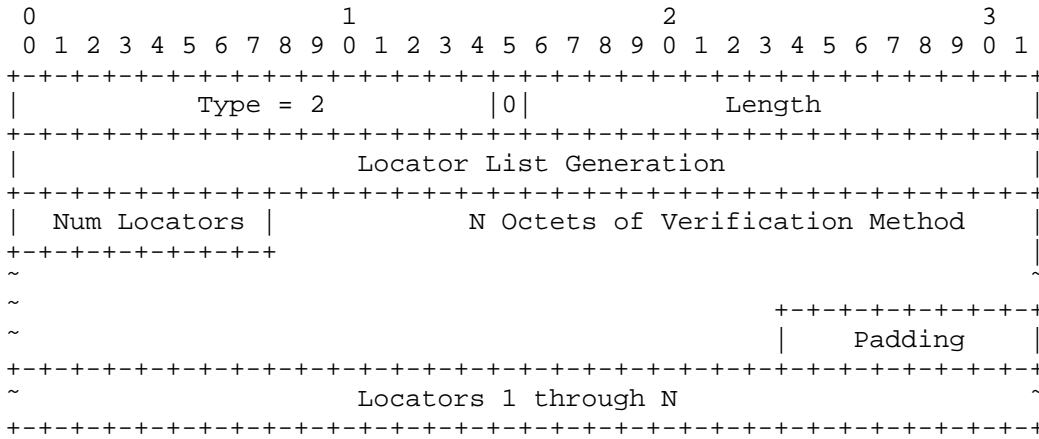   Sections 7.10.1 and 7.17.1.

```
     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 = 1           |0|              Length            |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    ~                           Validator                           ~
    ~                                       +-+-+-+-+-+-+-+-+-+
    ~                                       |      Padding     |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   Fields:

   Validator: Variable length content whose interpretation is local
                   to the responder.

   Padding: Padding, 0-7 bytes, added if needed. See
                   Section 5.15.

5.15.2. Locator List Option Format

   The Locator List option is used to carry all the locators of the
   sender. Note that the order of the locators is important, since the
   Locator Preferences option refers to the locators by using the index
   in the list.

   Note that we carry all the locators in this option even though some
   of them can be created automatically from the CGA Parameter Data
   Structure.

```
      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 = 2          |0|           Length               |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |                     Locator List Generation                   |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     |  Num Locators |          N Octets of Verification Method       |
     +-+-+-+-+-+-+-+-+                                                |
     ~                                                               ~
     ~                                      +-+-+-+-+-+-+-+-+
     ~                                      |    Padding    |
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
     ~                  Locators 1 through N                         ~
     +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fields:

Locator List Generation:
                32-bit unsigned integer. Indicates a generation
                number that is increased by one for each new locator
                list. This is used to ensure that the index in the
                Locator Preferences refers to the right version of the
                locator list.

Num Locators: 8-bit unsigned integer. The number of locators that
                are included in the option. We call this number "N"
                below.

Verification Method:
                N octets. The ith octet specifies the verification
                method for the ith locator.

Padding: Padding, 0-7 bytes, added if needed so that the
                Locators start on a multiple-of-8-octet boundary.
                Note that for this option, there is never a need to
                pad at the end since the Locators are a multiple-of-8-
                octets in length. This internal padding is included
                in the Length field.

Locators: N 128-bit locators.

The defined verification methods are:

```
+---------+-------------------------------+
| Value   |             Method            |
+---------+-------------------------------+
|    0    |           Reserved            |
|    1    |             HBA               |
|    2    |             CGA               |
|  3-200  | Allocated using Standards action |
| 201-254 |        Experimental use       |
|   255   |           Reserved            |
+---------+-------------------------------+
```

Table 4

5.15.3. Locator Preferences Option Format

   The Locator Preferences option can have some flags to indicate
   whether or not a locator is known to work. In addition, the sender
   can include a notion of preferences. It might make sense to define
   "preferences" as a combination of priority and weight, the same way
   that DNS SRV records have such information. The priority would
   provide a way to rank the locators, and, within a given priority, the
   weight would provide a way to do some load sharing. See [5] for how
   SRV defines the interaction of priority and weight.

   The minimum notion of preferences we need is to be able to indicate
   that a locator is "dead". We can handle this using a single octet
   flag for each locator.

   We can extend that by carrying a larger "element" for each locator.
   This document presently also defines 2-octet and 3-octet elements,
   and we can add more information by having even larger elements if
   need be.

   The locators are not included in the preference list. Instead, the
   first element refers to the locator that was in the first element in
   the Locator List option. The generation number carried in this
   option and the Locator List option is used to verify that they refer
   to the same version of the locator list.

```
 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 = 3           |0|            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Locator List Generation                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Element Len  |  Element[1]   |  Element[2]   |  Element[3]   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                            ...                                ~
~                                               +-+-+-+-+-+-+-+-+
~                                               |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
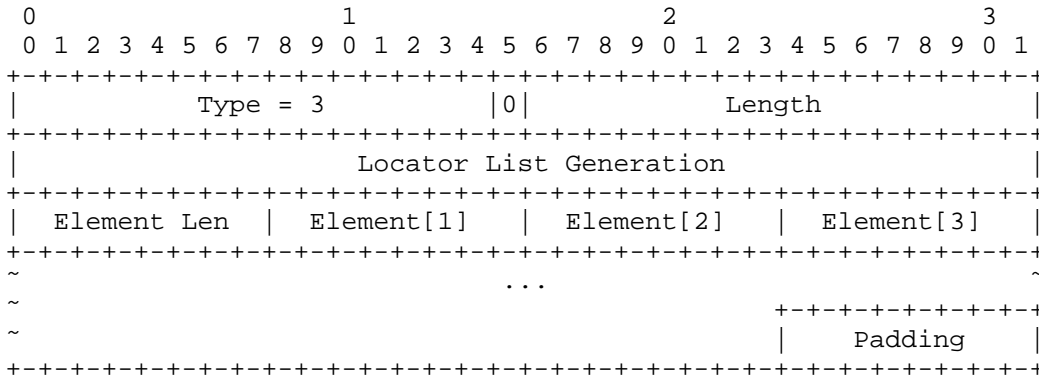
   Case of Element Len = 1 is depicted.

   Fields:

   Locator List Generation:
                  32-bit unsigned integer. Indicates a generation
                  number for the locator list to which the elements
                  should apply.

   Element Len: 8-bit unsigned integer. The length in octets of each
                  element. This specification defines the cases when
                  the length is 1, 2, or 3.

   Element[i]:   A field with a number of octets defined by the Element
                  Len field. Provides preferences for the ith locator
                  in the Locator List option that is in use.

   Padding: Padding, 0-7 bytes, added if needed. See
                  Section 5.15.

   When the Element length equals one, then the element consists of only
   a one-octet Flags field. The currently defined set of flags are:

      BROKEN: 0x01

      TRANSIENT: 0x02

   The intent of the BROKEN flag is to inform the peer that a given
   locator is known to be not working. The intent of TRANSIENT is to
   allow the distinction between more stable addresses and less stable
   addresses when Shim6 is combined with IP mobility, and when we might
   have more stable home locators and less stable care-of-locators.

When the Element length equals two, then the element consists of a
one-octet Flags field followed by a one-octet Priority field. This
Priority field has the same semantics as the Priority field in DNS
SRV records.

When the Element length equals three, then the element consists of a
one-octet Flags field followed by a one-octet Priority field and a
one-octet Weight field. This Weight field has the same semantics as
the Weight field in DNS SRV records.

This document doesn't specify the format when the Element length is
more than three, except that any such formats MUST be defined so that
the first three octets are the same as in the above case, that is, a
one-octet Flags field followed by a one-octet Priority field, and a
one-octet Weight field.

5.15.4. CGA Parameter Data Structure Option Format

This option contains the CGA Parameter Data Structure (PDS). When
HBA is used to verify the locators, the PDS contains the HBA
multiprefix extension in addition to the PDS mandatory fields and
other extensions unrelated to Shim6 that the PDS might have. When
CGA is used to verify the locators, in addition to the PDS option,
the host also needs to include the signature in the form of a CGA
Signature option.

```
 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 = 4         |0|              Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                 CGA Parameter Data Structure                  ~
~                                       +-+-+-+-+-+-+-+-+
~                                       |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
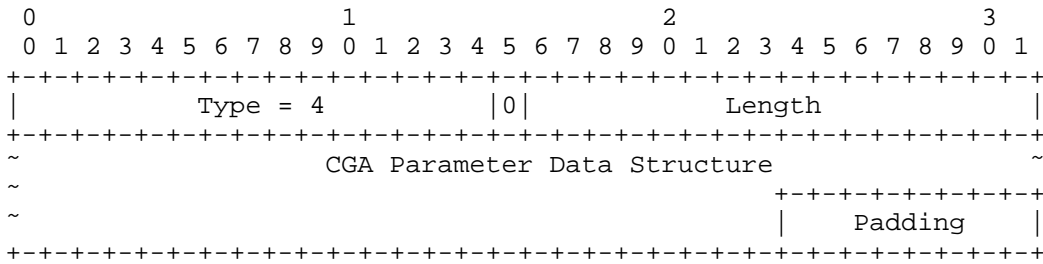
Fields:

CGA Parameter Data Structure:
                Variable length content. Content defined in [2] and
                [3].

Padding: Padding, 0-7 bytes, added if needed. See
                Section 5.15.

5.15.5. CGA Signature Option Format

   When CGA is used for verification of one or more of the locators in
   the Locator List option, then the message in question will need to
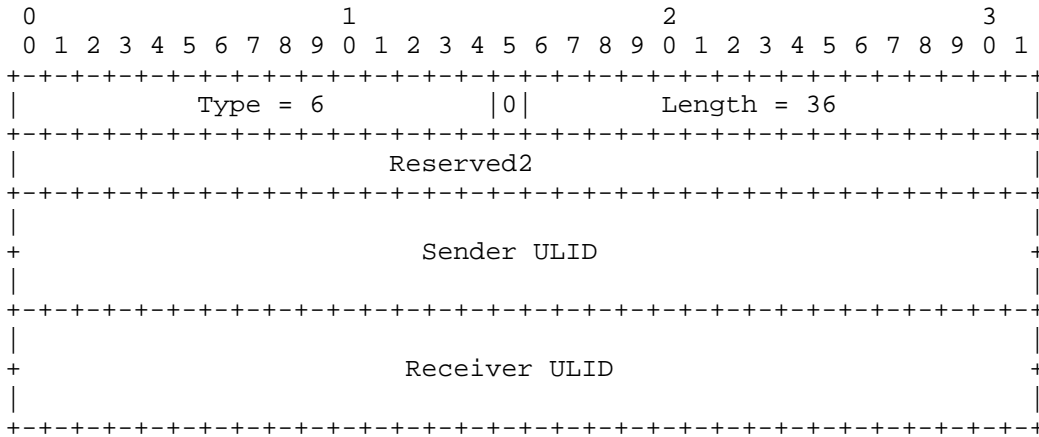   contain this option.

```
 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 = 5          |0|            Length            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                         CGA Signature                         ~
~                                               +-+-+-+-+-+-+-+-+
~                                               |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
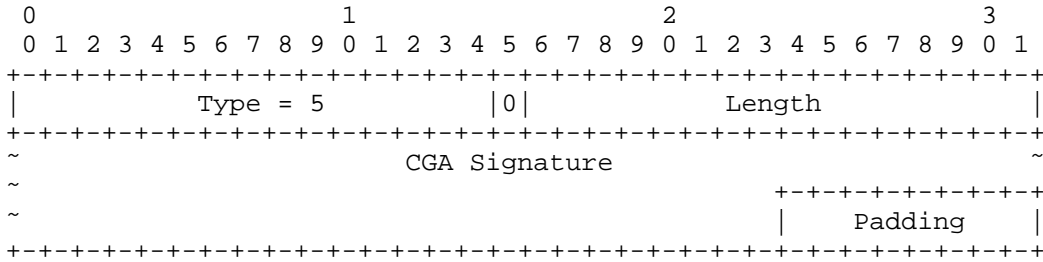
   Fields:

   CGA Signature: A variable-length field containing a PKCS#1 v1.5
                  signature, constructed by using the sender's private
                  key over the following sequence of octets:

                  1. The 128-bit CGA Message Type tag [CGA] value for
                      Shim6: 0x4A 30 5662 4858 574B 3655 416F 506A 6D48.
                      (The tag value has been generated randomly by the
                      editor of this specification.).

                  2. The Locator List Generation number of the
                      correspondent Locator List option.

                  3. The subset of locators included in the
                      correspondent Locator List option whose
                      verification method is set to CGA. The locators
                      MUST be included in the order in which they are
                      listed in the Locator List Option.

   Padding: Padding, 0-7 bytes, added if needed. See
                  Section 5.15.

5.15.6. ULID Pair Option Format

   I1, I2, and I2bis messages MUST contain the ULID pair; normally, this
   is in the IPv6 Source and Destination fields. In case the ULID for
   the context differs from the address pair included in the Source and
   Destination Address fields of the IPv6 packet used to carry the I1/
   I2/I2bis message, the ULID Pair option MUST be included in the I1/I2/
   I2bis message.

```
    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 = 6          |0|           Length = 36         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                            Reserved2                          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   +                          Sender ULID                          +
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   +                         Receiver ULID                         +
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   Fields:

   Reserved2:   32-bit field. Reserved for future use. Zero on
                  transmit. MUST be ignored on receipt. (Needed to
                  make the ULIDs start on a multiple-of-8-octet
                  boundary.)

   Sender ULID: A 128-bit IPv6 address.

   Receiver ULID: A 128-bit IPv6 address.

5.15.7. Forked Instance Identifier Option Format

```
    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 = 7           |0|           Length = 4          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                  Forked Instance Identifier                   |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
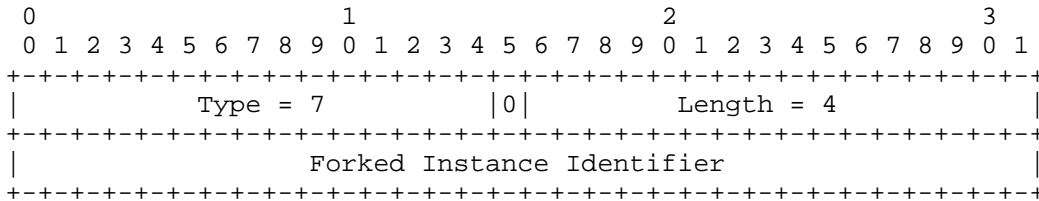
   Fields:

   Forked Instance Identifier:
                  32-bit field containing the identifier of the
                  particular forked instance.

5.15.8. Keepalive Timeout Option Format

   This option is defined in [4].

6. Conceptual Model of a Host

    This section describes a conceptual model of one possible data
    structure organization that hosts will maintain for the purposes of
    Shim6. The described organization is provided to facilitate the
    explanation of how the Shim6 protocol should behave. This document
    does not mandate that implementations adhere to this model as long as
    their external behavior is consistent with that described in this
    document.

6.1. Conceptual Data Structures

    The key conceptual data structure for the Shim6 protocol is the ULID-
    pair context. This is a data structure that contains the following
    information:

    o   The state of the context. See Section 6.2.

    o   The peer ULID: ULID(peer).

    o   The local ULID: ULID(local).

    o   The Forked Instance Identifier: FII. This is zero for the default
        context, i.e., when there is no forking.

    o   The list of peer locators with their preferences: Ls(peer).

    o   The generation number for the most recently received, verified
        peer locator list.

    o   For each peer locator, the verification method to use (from the
        Locator List option).

    o   For each peer locator, a flag specifying whether it has been
        verified using HBA or CGA, and a bit specifying whether the
        locator has been probed to verify that the ULID is present at that
        location.

    o   The current peer locator is the locator used as the destination
        address when sending packets: Lp(peer).

    o   The set of local locators and the preferences: Ls(local).

    o   The generation number for the most recently sent Locator List
        option.

    o   The current local locator is the locator used as the source
        address when sending packets: Lp(local).

o The Context Tag used to transmit control messages and Shim6
   Payload Extension headers; this is allocated by the peer:
   CT(peer).

o The context to expect in received control messages and Shim6
   Payload Extension headers; this is allocated by the local host:
   CT(local).

o Timers for retransmission of the messages during context-
   establishment and update messages.

o Depending how an implementation determines whether a context is
   still in use, there might be a need to track the last time a
   packet was sent/received using the context.

o Reachability state for the locator pairs as specified in [4].

o During pair exploration, information about the Probe messages that
   have been sent and received as specified in [4].

o During context-establishment phase, the Initiator Nonce, Responder
   Nonce, Responder Validator, and timers related to the different
   packets sent (I1,I2, R2), as described in Section 7.

6.2. Context STATES

   The STATES that are used to describe the Shim6 protocol are as
   follows:

```
+--------------------+------------------------------------------+
| STATE              | Explanation                              |
+--------------------+------------------------------------------+
| IDLE               | State machine start                      |
|                    |                                          |
| I1-SENT            | Initiating context-establishment exchange|
|                    |                                          |
| I2-SENT            | Waiting to complete context-establishment|
|                    | exchange                                 |
|                    |                                          |
| I2BIS-SENT         | Potential context loss detected          |
|                    |                                          |
| ESTABLISHED        | SHIM context established                 |
|                    |                                          |
| E-FAILED           | Context-establishment exchange failed    |
|                    |                                          |
| NO-SUPPORT         | ICMP Unrecognized Next Header type       |
|                    | (type 4, code 1) received, indicating    |
|                    | that Shim6 is not supported              |
+--------------------+------------------------------------------+
```

In addition, in each of the aforementioned STATES, the following
state information is stored:

```
+--------------------+--------------------------------------------+
| STATE              | Information                                |
+--------------------+--------------------------------------------+
| IDLE               | None                                       |
|                    |                                            |
| I1-SENT            | ULID(peer), ULID(local), [FII], CT(local), |
|                    | INIT Nonce, Lp(local), Lp(peer), Ls(local) |
|                    |                                            |
| I2-SENT            | ULID(peer), ULID(local), [FII], CT(local), |
|                    | INIT Nonce, RESP Nonce, Lp(local), Lp(peer),|
|                    | Ls(local), Responder Validator             |
|                    |                                            |
| ESTABLISHED        | ULID(peer), ULID(local), [FII], CT(local), |
|                    | CT(peer), Lp(local), Lp(peer), Ls(local),  |
|                    | Ls(peer), INIT Nonce?(to receive late R2)  |
|                    |                                            |
| I2BIS-SENT         | ULID(peer), ULID(local), [FII], CT(local), |
|                    | CT(peer), Lp(local), Lp(peer), Ls(local),  |
|                    | Ls(peer), CT(R1bis), RESP Nonce,           |
|                    | INIT Nonce, Responder Validator            |
|                    |                                            |
| E-FAILED           | ULID(peer), ULID(local)                    |
|                    |                                            |
| NO-SUPPORT         | ULID(peer), ULID(local)                    |
+--------------------+--------------------------------------------+
```

7. Establishing ULID-Pair Contexts

   ULID-pair contexts are established using a 4-way exchange, which
   allows the responder to avoid creating state on the first packet. As
   part of this exchange, each end allocates a Context Tag and shares
   this Context Tag and its set of locators with the peer.

   In some cases, the 4-way exchange is not necessary -- for instance,
   when both ends try to set up the context at the same time, or when
   recovering from a context that has been garbage collected or lost at
   one of the hosts.

7.1. Uniqueness of Context Tags

   As part of establishing a new context, each host has to assign a
   unique Context Tag. Since the Shim6 Payload Extension headers are
   demultiplexed based solely on the Context Tag value (without using
   the locators), the Context Tag MUST be unique for each context.

It is important that Context Tags are hard to guess for off-path
attackers. Therefore, if an implementation uses structure in the
Context Tag to facilitate efficient lookups, at least 30 bits of the
Context Tag MUST be unstructured and populated by random or pseudo-
random bits.

In addition, in order to minimize the reuse of Context Tags, the host
SHOULD randomly cycle through the unstructured tag name space that is
reserved for randomly assigned Context Tag values (e.g., following
the guidelines described in [13]).

7.2. Locator Verification

The peer's locators might need to be verified during context
establishment as well as when handling locator updates in Section 10.

There are two separate aspects of locator verification. One is to
verify that the locator is tied to the ULID, i.e., that the host that
"owns" the ULID is also the one that is claiming the locator
"ownership". The Shim6 protocol uses the HBA or CGA techniques for
doing this verification. The other aspect is to verify that the host
is indeed reachable at the claimed locator. Such verification is
needed not only to make sure communication can proceed but also to
prevent 3rd party flooding attacks [15]. These different aspects of
locator verification happen at different times since the first might
need to be performed before packets can be received by the peer with
the source locator in question, but the latter verification is only
needed before packets are sent to the locator.

Before a host can use a locator (different than the ULID) as the
source locator, it must know that the peer will accept packets with
that source locator as part of this context. Thus, the HBA/CGA
verification SHOULD be performed by the host before the host
acknowledges the new locator by sending either an Update
Acknowledgement message or an R2 message.

Before a host can use a locator (different than the ULID) as the
destination locator, it MUST perform the HBA/CGA verification if this
was not performed upon reception of the locator set. In addition, it
MUST verify that the ULID is indeed present at that locator. This
verification is performed by doing a return-routability test as part
of the Probe sub-protocol [4].

If the verification method in the Locator List option is not
supported by the host, or if the verification method is not
consistent with the CGA Parameter Data Structure (e.g., the Parameter
Data Structure doesn't contain the multiprefix extension and the
verification method says to use HBA), then the host MUST ignore the

Locator List and the message in which it is contained. The host
SHOULD generate a Shim6 Error message with Error Code=2 and with the
Pointer referencing the octet in the verification method that was
found inconsistent.

7.3. Normal Context Establishment

The normal context establishment consists of a 4-message exchange in
the order of I1, R1, I2, R2, as can be seen in Figure 3.

```
        Initiator                           Responder

         IDLE                                IDLE
             ------------- I1 -------------->
         I1-SENT
             <----------- R1 ---------------
                                             IDLE
             ------------- I2 -------------->
         I2-SENT
             <----------- R2 ---------------
         ESTABLISHED                         ESTABLISHED
```
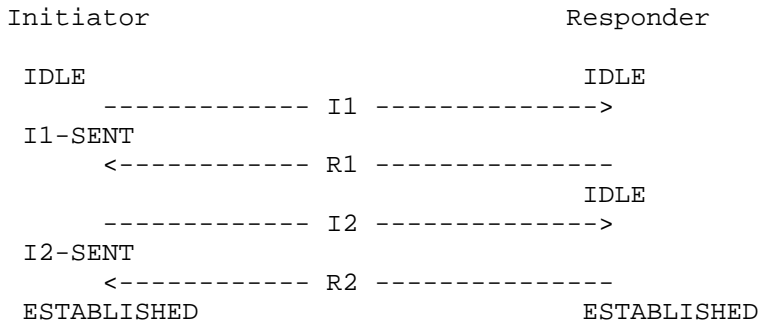
                Figure 3: Normal Context Establishment

7.4. Concurrent Context Establishment

When both ends try to initiate a context for the same ULID pair, then
we might end up with crossing I1 messages. Alternatively, since no
state is created when receiving the I1, a host might send an I1 after
having sent an R1 message.

Since a host remembers that it has sent an I1, it can respond to an
I1 from the peer (for the same ULID pair) with an R2, resulting in
the message exchange shown in Figure 4. Such behavior is needed for
reasons such as correctly responding to retransmitted I1 messages,
which occur when the R2 message has been lost.

```
       Host A                              Host B

        IDLE                                IDLE
          -\
        I1-SENT---\
               ---\             /---
                 --- I1 ---\   /---   I1-SENT
                          ---\
               /--- I1 ---/     ---\
            /---                    -->
          <---

          -\
        I1-SENT---\
               ---\             /---
                 --- R2 ---\   /---   I1-SENT
                          ---\
               /--- R2 ---/     ---\
            /---                    -->
          <---                      ESTABLISHED
        ESTABLISHED
```
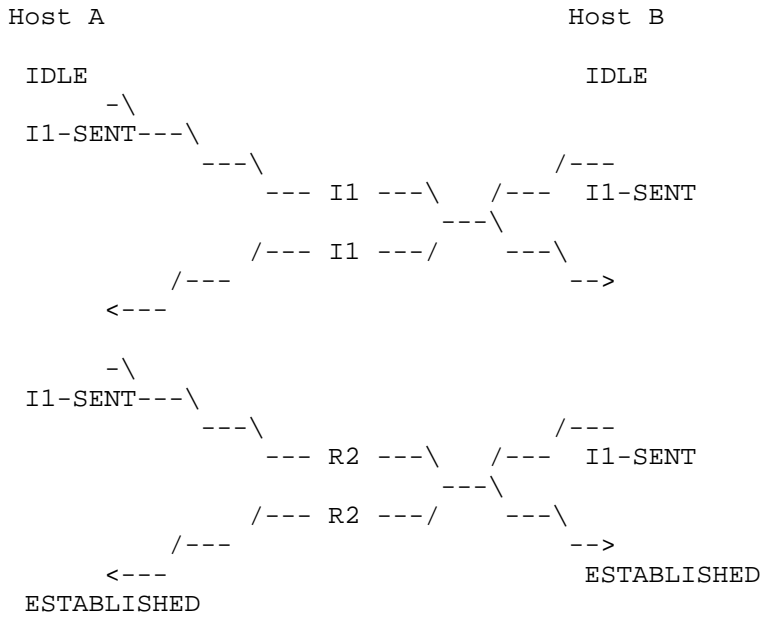
                  Figure 4: Crossing I1 Messages

   If a host has received an I1 and sent an R1, it has no state to
   remember this. Thus, if the ULP on the host sends down packets, this
   might trigger the host to send an I1 message itself. Thus, while one
   end is sending an I1, the other is sending an I2, as can be seen in
   Figure 5.

```
         Host A                              Host B

          IDLE                                IDLE
            -\
              ---\
          I1-SENT    ---\
                      --- I1 ---\
                                 ---\
                                     ---\
                                         -->

                                      /---
                                   /---  IDLE
                                ---
                     /--- R1--/
                 /---
               <---

               -\
          I2-SENT---\
                  ---\                  /---
                     --- I2---\    /---   I1-SENT
                               ---\
                   /--- I1 ---/      ---\
                /---                     -->
              <---                         ESTABLISHED

               -\
          I2-SENT---\
                  ---\                  /---
                     --- R2 ---\   /---
                                ---\
                   /--- R2 ---/      ---\
                /---                     -->
              <---                         ESTABLISHED
          ESTABLISHED
```
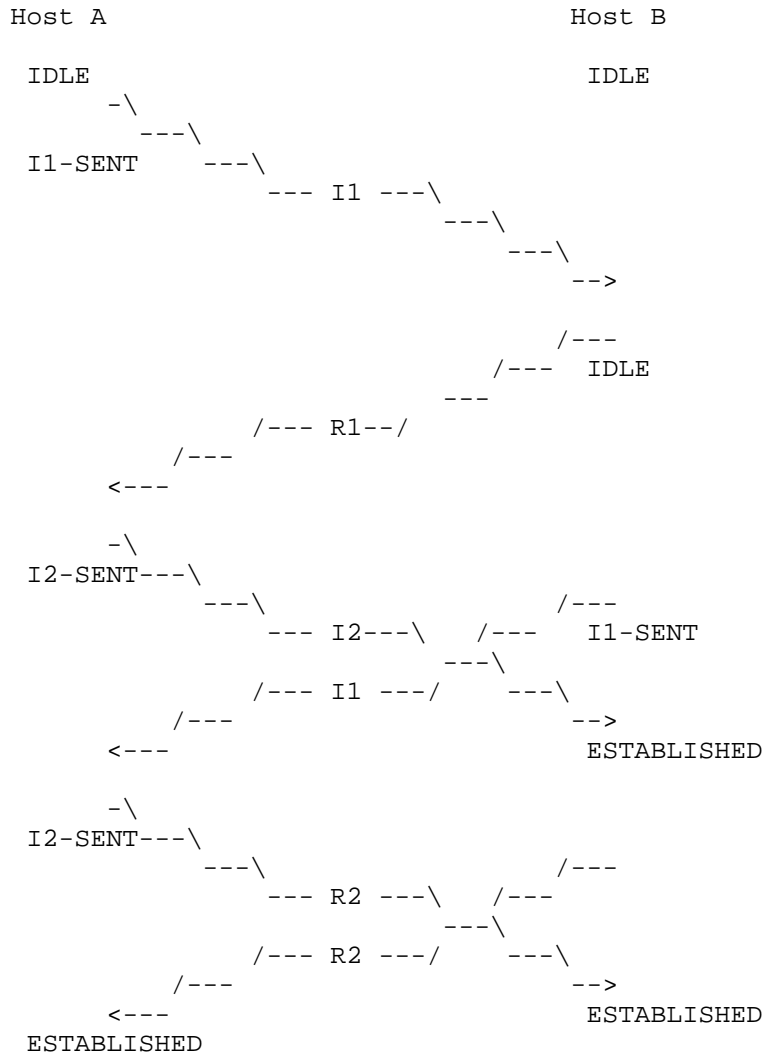
                      Figure 5: Crossing I2 and I1

7.5. Context Recovery

   Due to garbage collection, we can end up with one end having and
   using the context state, and the other end not having any state. We
   need to be able to recover this state at the end that has lost it
   before we can use it.

This need can arise in the following cases:

o The communication is working using the ULID pair as the locator
   pair but a problem arises, and the end that has retained the
   context state decides to probe alternate locator pairs.

o The communication is working using a locator pair that is not the
   ULID pair; hence, the ULP packets sent from a peer that has
   retained the context state use the Shim6 Payload Extension header.

o The host that retained the state sends a control message (e.g., an
   Update Request message).

In all cases, the result is that the peer without state receives a
shim message for which it has no context for the Context Tag.

We can recover the context by having the node that doesn't have a
context state send back an R1bis message, and then complete the
recovery with an I2bis and R2 message, as can be seen in Figure 6.

```
        Host A                              Host B

     Context for
     CT(peer)=X                      Discards context for
                                     CT(local)=X

      ESTABLISHED                         IDLE

         ---- payload, probe, etc. -----> No context state
                                             for CT(local)=X

         <------------ R1bis ------------
                                          IDLE

         ------------ I2bis ----------->
      I2BIS_SENT
         <------------ R2 --------------
      ESTABLISHED                      ESTABLISHED
```
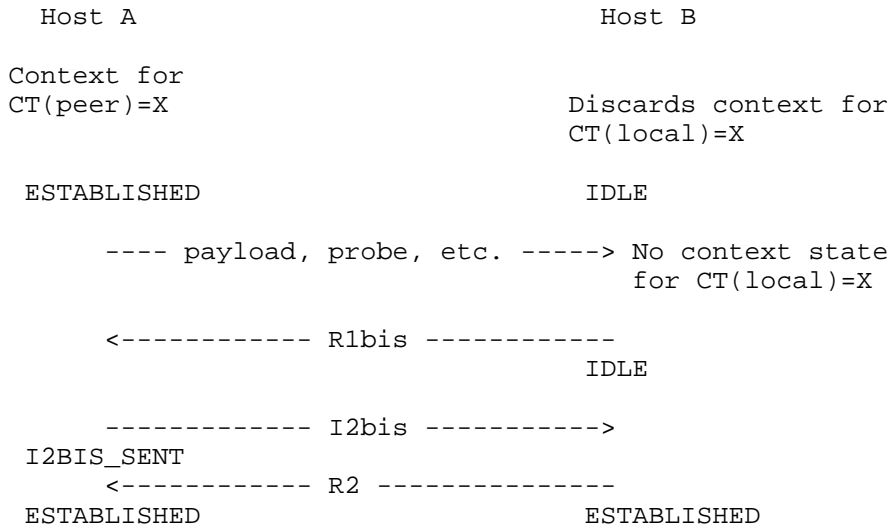
                    Figure 6: Context Loss at Receiver

If one end has garbage collected or lost the context state, it might
try to create a new context state (for the same ULID pair), by
sending an I1 message. In this case, the peer (that still has the
context state) will reply with an R1 message, and the full 4-way
exchange will be performed again, as can be seen in Figure 7.

```
      Host A                                  Host B

    Context for
    CT(peer)=X                          Discards context for
    ULIDs A1, B1                        CT(local)=X

      ESTABLISHED                           IDLE

    Finds  <------------ I1 -------------- Tries to set up
    existing                               for ULIDs A1, B1
    context,
    but CT(peer)                        I1-SENT
    doesn't match
            ------------ R1 -------------->
    Left old context
    in ESTABLISHED

          <------------ I2 --------------
    Re-create context
    with new CT(peer)                   I2-SENT
    and Ls(peer).

      ESTABLISHED
            ------------ R2 -------------->
    ESTABLISHED                             ESTABLISHED
```
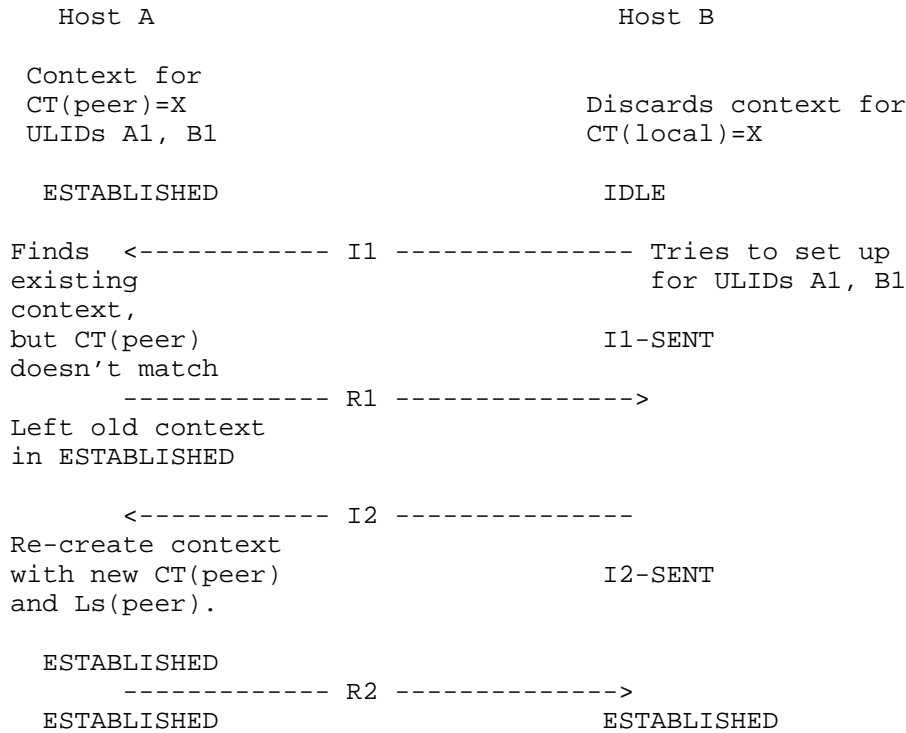
                    Figure 7: Context Loss at Sender

7.6. Context Confusion

   Since each end might garbage collect the context state, we can have
   the case where one end has retained the context state and tries to
   use it, while the other end has lost the state. We discussed this in
   the previous section on recovery. But, for the same reasons, when
   one host retains Context Tag X as CT(peer) for ULID pair <A1, B1>,
   the other end might end up allocating that Context Tag as CT(local)
   for another ULID pair (e.g., <A3, B1>) between the same hosts. In
   this case, we cannot use the recovery mechanisms since there needs to
   be separate Context Tags for the two ULID pairs.

   This type of "confusion" can be observed in two cases (assuming it is
   A that has retained the state and B that has dropped it):

   o B decides to create a context for ULID pair <A3, B1>, allocates X
      as its Context Tag for this, and sends an I1 to A.

   o A decides to create a context for ULID pair <A3, B1> and starts
      the exchange by sending I1 to B. When B receives the I2 message,
      it allocates X as the Context Tag for this context.

   In both cases, A can detect that B has allocated X for ULID pair <A3,
   B1> even though A still has X as CT(peer) for ULID pair <A1, B1>.
   Thus, A can detect that B must have lost the context for <A1, B1>.

   The confusion can be detected when I2/I2bis/R2 is received, since we
   require that those messages MUST include a sufficiently large set of
   locators in a Locator List option that the peer can determine whether
   or not two contexts have the same host as the peer by comparing if
   there is any common locators in Ls(peer).

   The old context that used the Context Tag MUST be removed; it can no
   longer be used to send packets. Thus, A would forcibly remove the
   context state for <A1, B1, X> so that it can accept the new context
   for <A3, B1, X>. An implementation MAY re-create a context to
   replace the one that was removed -- in this case, for <A1, B1>. The
   normal I1, R1, I2, R2 establishment exchange would then pick unique
   Context Tags for that replacement context. This re-creation is
   OPTIONAL, but might be useful when there is ULP communication that is
   using the ULID pair whose context was removed.

   Note that an I1 message with a duplicate Context Tag should not cause
   the removal of the old context state; this operation needs to be
   deferred until the reception of the I2 message.

7.7. Sending I1 Messages

   When the shim layer decides to set up a context for a ULID pair, it
   starts by allocating and initializing the context state for its end.
   As part of this, it assigns a random Context Tag to the context that
   is not being used as CT(local) by any other context . In the case
   that a new API is used and the ULP requests a forked context, the
   Forked Instance Identifier value will be set to a non-zero value.
   Otherwise, the FII value is zero. Then the initiator can send an I1
   message and set the context STATE to I1-SENT. The I1 message MUST
   include the ULID pair -- normally, in the IPv6 Source and Destination
   fields. But if the ULID pair for the context is not used as a
   locator pair for the I1 message, then a ULID option MUST be included
   in the I1 message. In addition, if a Forked Instance Identifier
   value is non-zero, the I1 message MUST include a Context Instance
   Identifier option containing the correspondent value.

7.8. Retransmitting I1 Messages

   If the host does not receive an R1 or R2 message in response to the
   I1 message after I1_TIMEOUT time, then it needs to retransmit the I1
   message. The retransmissions should use a retransmission timer with
   binary exponential backoff to avoid creating congestion issues for
   the network when lots of hosts perform I1 retransmissions. Also, the
   actual timeout value should be randomized between 0.5 and 1.5 of the
   nominal value to avoid self-synchronization.

   If, after I1_RETRIES_MAX retransmissions, there is no response, then
   most likely the peer does not implement the Shim6 protocol (or there
   could be a firewall that blocks the protocol). In this case, it
   makes sense for the host to remember not to try again to establish a
   context with that ULID. However, any such negative caching should be
   retained for at most NO_R1_HOLDDOWN_TIME, in order to be able to
   later set up a context should the problem have been that the host was
   not reachable at all when the shim tried to establish the context.

   If the host receives an ICMP error with "Unrecognized Next Header"
   type (type 4, code 1) and the included packet is the I1 message it
   just sent, then this is a more reliable indication that the peer ULID
   does not implement Shim6. Again, in this case, the host should
   remember not to try again to establish a context with that ULID.
   Such negative caching should be retained for at most
   ICMP_HOLDDOWN_TIME, which should be significantly longer than the
   previous case.

7.9. Receiving I1 Messages

   A host MUST silently discard any received I1 messages that do not
   satisfy all of the following validity checks in addition to those
   specified in Section 12.3:

   o The Hdr Ext Len field is at least 1, i.e., the length is at least
      16 octets.

   Upon the reception of an I1 message, the host extracts the ULID pair
   and the Forked Instance Identifier from the message. If there is no
   ULID-pair option, then the ULID pair is taken from the Source and
   Destination fields in the IPv6 header. If there is no FII option in
   the message, then the FII value is taken to be zero.

   Next, the host looks for an existing context that matches the ULID
   pair and the FII.

   If no state is found (i.e., the STATE is IDLE), then the host replies
   with an R1 message as specified below.

If such a context exists in ESTABLISHED STATE, the host verifies that
the locator of the initiator is included in Ls(peer). (This check is
unnecessary if there is no ULID-pair option in the I1 message.)

If the state exists in ESTABLISHED STATE and the locators do not fall
in the locator sets, then the host replies with an R1 message as
specified below. This completes the I1 processing, with the context
STATE being unchanged.

If the state exists in ESTABLISHED STATE and the locators do fall in
the sets, then the host compares CT(peer) for the context with the CT
contained in the I1 message.

o   If the Context Tags match, then this probably means that the R2
    message was lost and this I1 is a retransmission. In this case,
    the host replies with an R2 message containing the information
    available for the existent context.

o   If the Context Tags do not match, then it probably means that the
    initiator has lost the context information for this context and is
    trying to establish a new one for the same ULID pair. In this
    case, the host replies with an R1 message as specified below.
    This completes the I1 processing, with the context STATE being
    unchanged.

If the state exists in other STATE (I1-SENT, I2-SENT, I2BIS-SENT), we
are in the situation of concurrent context establishment, described
in Section 7.4. In this case, the host leaves CT(peer) unchanged and
replies with an R2 message. This completes the I1 processing, with
the context STATE being unchanged.

7.10. Sending R1 Messages

When the host needs to send an R1 message in response to the I1
message, it copies the Initiator Nonce from the I1 message to the R1
message, generates a Responder Nonce, and calculates a Responder
Validator option as suggested in the following section. No state is
created on the host in this case. (Note that the information used to
generate the R1 reply message is either contained in the received I1
message or is global information that is not associated with the
particular requested context (the S and the Responder Nonce values.))

When the host needs to send an R2 message in response to the I1
message, it copies the Initiator Nonce from the I1 message to the R2
message, and otherwise follows the normal rules for forming an R2
message (see Section 7.14).

7.10.1. Generating the R1 Validator

   As it is stated in Section 5.15.1, the validator-generation mechanism
   is a local choice since the validator is generated and verified by
   the same node, i.e., the responder. However, in order to provide the
   required protection, the validator needs to be generated by
   fulfilling the conditions described in Section 5.15.1. One way for
   the responder to properly generate validators is to maintain a single
   secret (S) and a running counter (C) for the Responder Nonce that is
   incremented in fixed periods of time (this allows the responder to
   verify the age of a Responder Nonce, independently of the context in
   which it is used).

   When the validator is generated to be included in an R1 message sent
   in response to a specific I1 message, the responder can perform the
   following procedure to generate the validator value:

   First, the responder uses the current counter C value as the
   Responder Nonce.

   Second, it uses the following information (concatenated) as input to
   the one-way function:

   o The secret S

   o That Responder Nonce

   o The Initiator Context Tag from the I1 message

   o The ULIDs from the I1 message

   o The locators from the I1 message (strictly only needed if they are
      different from the ULIDs)

   o The Forked Instance Identifier, if such option was included in the
      I1 message

   Third, it uses the output of the hash function as the validator value
   included in the R1 message.

7.11. Receiving R1 Messages and Sending I2 Messages

   A host MUST silently discard any received R1 messages that do not
   satisfy all of the following validity checks in addition to those
   specified in Section 12.3:

   o The Hdr Ext Len field is at least 1, i.e., the length is at least
      16 octets.

Upon the reception of an R1 message, the host extracts the Initiator
Nonce and the Locator Pair from the message (the latter from the
Source and Destination fields in the IPv6 header). Next, the host
looks for an existing context that matches the Initiator Nonce and
where the locators are contained in Ls(peer) and Ls(local),
respectively. If no such context is found, then the R1 message is
silently discarded.

If such a context is found, then the host looks at the STATE:

o If the STATE is I1-SENT, then it sends an I2 message as specified
   below.

o In any other STATE (I2-SENT, I2BIS-SENT, ESTABLISHED), then the
   host has already sent an I2 message and this is probably a reply
   to a retransmitted I1 message, so this R1 message MUST be silently
   discarded.

When the host sends an I2 message, it includes the Responder
Validator option that was in the R1 message. The I2 message MUST
include the ULID pair -- normally, in the IPv6 Source and Destination
fields. If a ULID-pair option was included in the I1 message, then
it MUST be included in the I2 message as well. In addition, if the
Forked Instance Identifier value for this context is non-zero, the I2
message MUST contain a Forked Instance Identifier option carrying the
Forked Instance Identifier value. Besides, the I2 message contains
an Initiator Nonce. This is not required to be the same as the one
included in the previous I1 message.

The I2 message may also include the initiator's locator list. If
this is the case, then it must also include the CGA Parameter Data
Structure. If CGA (and not HBA) is used to verify one or more of the
locators included in the locator list, then the initiator must also
include a CGA Signature option containing the signature.

When the I2 message has been sent, the STATE is set to I2-SENT.

7.12. Retransmitting I2 Messages

If the initiator does not receive an R2 message after I2_TIMEOUT time
after sending an I2 message, it MAY retransmit the I2 message, using
binary exponential backoff and randomized timers. The Responder
Validator option might have a limited lifetime -- that is, the peer
might reject Responder Validator options that are older than
VALIDATOR_MIN_LIFETIME to avoid replay attacks. In the case that the
initiator decides not to retransmit I2 messages, or in the case that

the initiator still does not receive an R2 message after
retransmitting I2 messages I2_RETRIES_MAX times, the initiator SHOULD
fall back to retransmitting the I1 message.

7.13. Receiving I2 Messages

A host MUST silently discard any received I2 messages that do not
satisfy all of the following validity checks in addition to those
specified in Section 12.3:

o The Hdr Ext Len field is at least 2, i.e., the length is at least
   24 octets.

Upon the reception of an I2 message, the host extracts the ULID pair
and the Forked Instance Identifier from the message. If there is no
ULID-pair option, then the ULID pair is taken from the Source and
Destination fields in the IPv6 header. If there is no FII option in
the message, then the FII value is taken to be zero.

Next, the host verifies that the Responder Nonce is a recent one
(nonces that are no older than VALIDATOR_MIN_LIFETIME SHOULD be
considered recent) and that the Responder Validator option matches
the validator the host would have computed for the ULID, locators,
Responder Nonce, Initiator Nonce, and FII.

If a CGA Parameter Data Structure (PDS) is included in the message,
then the host MUST verify if the actual PDS contained in the message
corresponds to the ULID(peer).

If any of the above verifications fail, then the host silently
discards the message; it has completed the I2 processing.

If all the above verifications are successful, then the host proceeds
to look for a context state for the initiator. The host looks for a
context with the extracted ULID pair and FII. If none exist, then
STATE of the (non-existing) context is viewed as being IDLE; thus,
the actions depend on the STATE as follows:

o If the STATE is IDLE (i.e., the context does not exist), the host
   allocates a Context Tag (CT(local)), creates the context state for
   the context, and sets its STATE to ESTABLISHED. It records
   CT(peer) and the peer's locator set as well as its own locator set
   in the context. It SHOULD perform the HBA/CGA verification of the
   peer's locator set at this point in time, as specified in
   Section 7.2. Then, the host sends an R2 message back as specified
   below.

o If the STATE is I1-SENT, then the host verifies if the source
   locator is included in Ls(peer) or in the Locator List contained
   in the I2 message and that the HBA/CGA verification for this
   specific locator is successful.

   * If this is not the case, then the message is silently discarded
      and the context STATE remains unchanged.

   * If this is the case, then the host updates the context
      information (CT(peer), Ls(peer)) with the data contained in the
      I2 message, and the host MUST send an R2 message back as
      specified below. Note that before updating Ls(peer)
      information, the host SHOULD perform the HBA/CGA validation of
      the peer's locator set at this point in time, as specified in
      Section 7.2. The host moves to ESTABLISHED STATE.

o If the STATE is ESTABLISHED, I2-SENT, or I2BIS-SENT, then the host
   verifies if the source locator is included in Ls(peer) or in the
   Locator List contained in the I2 message and that the HBA/CGA
   verification for this specific locator is successful.

   * If this is not the case, then the message is silently discarded
      and the context STATE remains unchanged.

   * If this is the case, then the host updates the context
      information (CT(peer), Ls(peer)) with the data contained in the
      I2 message, and the host MUST send an R2 message back as
      specified in Section 7.14. Note that before updating Ls(peer)
      information, the host SHOULD perform the HBA/CGA validation of
      the peer's locator set at this point in time, as specified in
      Section 7.2. The context STATE remains unchanged.

7.14. Sending R2 Messages

   Before the host sends the R2 message, it MUST look for a possible
   context confusion, i.e., where it would end up with multiple contexts
   using the same CT(peer) for the same peer host. See Section 7.15.

   When the host needs to send an R2 message, the host forms the message
   and its Context Tag, and copies the Initiator Nonce from the
   triggering message (I2, I2bis, or I1). In addition, it may include
   alternative locators and necessary options so that the peer can
   verify them. In particular, the R2 message may include the
   responder's locator list and the PDS option. If CGA (and not HBA) is
   used to verify the locator list, then the responder also signs the
   key parts of the message and includes a CGA Signature option
   containing the signature.

R2 messages are never retransmitted. If the R2 message is lost, then
the initiator will retransmit either the I2/I2bis or I1 message.
Either retransmission will cause the responder to find the context
state and respond with an R2 message.

7.15. Match for Context Confusion

When the host receives an I2, I2bis, or R2, it MUST look for a
possible context confusion, i.e., where it would end up with multiple
contexts using the same CT(peer) for the same peer host. This can
happen when the host has received the above messages, since they
create a new context with a new CT(peer). The same issue applies
when CT(peer) is updated for an existing context.

The host takes CT(peer) for the newly created or updated context, and
looks for other contexts which:

o Are in STATE ESTABLISHED or I2BIS-SENT

o Have the same CT(peer)

o Have an Ls(peer) that has at least one locator in common with the
   newly created or updated context

If such a context is found, then the host checks if the ULID pair or
the Forked Instance Identifier are different than the ones in the
newly created or updated context:

o If either or both are different, then the peer is reusing the
   Context Tag for the creation of a context with different ULID pair
   or FII, which is an indication that the peer has lost the original
   context. In this case, we are in a context confusion situation,
   and the host MUST NOT use the old context to send any packets. It
   MAY just discard the old context (after all, the peer has
   discarded it), or it MAY attempt to re-establish the old context
   by sending a new I1 message and moving its STATE to I1-SENT. In
   any case, once that this situation is detected, the host MUST NOT
   keep two contexts with overlapping Ls(peer) locator sets and the
   same Context Tag in ESTABLISHED STATE, since this would result in
   demultiplexing problems on the peer.

o If both are the same, then this context is actually the context
   that is created or updated; hence, there is no confusion.

7.16. Receiving R2 Messages

   A host MUST silently discard any received R2 messages that do not
   satisfy all of the following validity checks in addition to those
   specified in Section 12.3:

   o The Hdr Ext Len field is at least 1, i.e., the length is at least
      16 octets.

   Upon the reception of an R2 message, the host extracts the Initiator
   Nonce and the Locator Pair from the message (the latter from the
   Source and Destination fields in the IPv6 header). Next, the host
   looks for an existing context that matches the Initiator Nonce and
   where the locators are Lp(peer) and Lp(local), respectively. Based
   on the STATE:

   o If no such context is found, i.e., the STATE is IDLE, then the
      message is silently dropped.

   o If STATE is I1-SENT, I2-SENT, or I2BIS-SENT, then the host
      performs the following actions. If a CGA Parameter Data Structure
      (PDS) is included in the message, then the host MUST verify that
      the actual PDS contained in the message corresponds to the
      ULID(peer) as specified in Section 7.2. If the verification
      fails, then the message is silently dropped. If the verification
      succeeds, then the host records the information from the R2
      message in the context state; it records the peer's locator set
      and CT(peer). The host SHOULD perform the HBA/CGA verification of
      the peer's locator set at this point in time, as specified in
      Section 7.2. The host sets its STATE to ESTABLISHED.

   o If the STATE is ESTABLISHED, the R2 message is silently ignored,
      (since this is likely to be a reply to a retransmitted I2
      message).

   Before the host completes the R2 processing, it MUST look for a
   possible context confusion, i.e., where it would end up with multiple
   contexts using the same CT(peer) for the same peer host. See
   Section 7.15.

7.17. Sending R1bis Messages

   Upon the receipt of a Shim6 Payload Extension header where there is
   no current Shim6 context at the receiver, the receiver is to respond
   with an R1bis message in order to enable a fast re-establishment of
   the lost Shim6 context.

Also, a host is to respond with an R1bis upon receipt of any control
messages that have a message type in the range 64-127 (i.e.,
excluding the context-setup messages such as I1, R1, R1bis, I2,
I2bis, R2, and future extensions), where the control message refers
to a non-existent context.

We assume that all the incoming packets that trigger the generation
of an R1bis message contain a locator pair (in the address fields of
the IPv6 header) and a Context Tag.

Upon reception of any of the packets described above, the host will
reply with an R1bis including the following information:

o   The Responder Nonce is a number picked by the responder that the
    initiator will return in the I2bis message.

o   Packet Context Tag is the Context Tag contained in the received
    packet that triggered the generation of the R1bis message.

o   The Responder Validator option is included, with a validator that
    is computed as suggested in the next section.

7.17.1. Generating the R1bis Validator

One way for the responder to properly generate validators is to
maintain a single secret (S) and a running counter C for the
Responder Nonce that is incremented in fixed periods of time (this
allows the responder to verify the age of a Responder Nonce,
independently of the context in which it is used).

When the validator is generated to be included in an R1bis message --
that is, sent in response to a specific control packet or a packet
containing the Shim6 Payload Extension header message -- the
responder can perform the following procedure to generate the
validator value:

First, the responder uses the counter C value as the Responder Nonce.

Second, it uses the following information (concatenated) as input to
the one-way function:

o   The secret S

o   That Responder Nonce

o   The Receiver Context Tag included in the received packet

o   The locators from the received packet

Third, it uses the output of the hash function as the validator
string.

7.18. Receiving R1bis Messages and Sending I2bis Messages

   A host MUST silently discard any received R1bis messages that do not
   satisfy all of the following validity checks in addition to those
   specified in Section 12.3:

   o The Hdr Ext Len field is at least 1, i.e., the length is at least
      16 octets.

   Upon the reception of an R1bis message, the host extracts the Packet
   Context Tag and the Locator Pair from the message (the latter from
   the Source and Destination fields in the IPv6 header). Next, the
   host looks for an existing context where the Packet Context Tag
   matches CT(peer) and where the locators match Lp(peer) and Lp(local),
   respectively.

   o If no such context is found, i.e., the STATE is IDLE, then the
      R1bis message is silently discarded.

   o If the STATE is I1-SENT, I2-SENT, or I2BIS-SENT, then the R1bis
      message is silently discarded.

   o If the STATE is ESTABLISHED, then we are in the case where the
      peer has lost the context, and the goal is to try to re-establish
      it. For that, the host leaves CT(peer) unchanged in the context
      state, transitions to I2BIS-SENT STATE, and sends an I2bis
      message, including the computed Responder Validator option, the
      Packet Context Tag, and the Responder Nonce that were received in
      the R1bis message. This I2bis message is sent using the locator
      pair included in the R1bis message. In the case that this locator
      pair differs from the ULID pair defined for this context, then a
      ULID option MUST be included in the I2bis message. In addition,
      if the Forked Instance Identifier for this context is non-zero,
      then a Forked Instance Identifier option carrying the instance
      identifier value for this context MUST be included in the I2bis
      message. The I2bis message may also include a locator list. If
      this is the case, then it must also include the CGA Parameter Data
      Structure. If CGA (and not HBA) is used to verify one or more of
      the locators included in the locator list, then the initiator must
      also include a CGA Signature option containing the signature.

7.19. Retransmitting I2bis Messages

   If the initiator does not receive an R2 message after I2bis_TIMEOUT
   time after sending an I2bis message, it MAY retransmit the I2bis
   message, using binary exponential backoff and randomized timers. The
   Responder Validator option might have a limited lifetime -- that is,
   the peer might reject Responder Validator options that are older than
   VALIDATOR_MIN_LIFETIME to avoid replay attacks. In the case that the
   initiator decides not to retransmit I2bis messages, or in the case
   that the initiator still does not receive an R2 message after
   retransmitting I2bis messages I2bis_RETRIES_MAX times, the initiator
   SHOULD fall back to retransmitting the I1 message.

7.20. Receiving I2bis Messages and Sending R2 Messages

   A host MUST silently discard any received I2bis messages that do not
   satisfy all of the following validity checks in addition to those
   specified in Section 12.3:

   o The Hdr Ext Len field is at least 3, i.e., the length is at least
      32 octets.

   Upon the reception of an I2bis message, the host extracts the ULID
   pair and the Forked Instance Identifier from the message. If there
   is no ULID-pair option, then the ULID pair is taken from the Source
   and Destination fields in the IPv6 header. If there is no FII option
   in the message, then the FII value is taken to be zero.

   Next, the host verifies that the Responder Nonce is a recent one
   (nonces that are no older than VALIDATOR_MIN_LIFETIME SHOULD be
   considered recent) and that the Responder Validator option matches
   the validator the host would have computed for the locators,
   Responder Nonce, and Receiver Context Tag as part of sending an R1bis
   message.

   If a CGA Parameter Data Structure (PDS) is included in the message,
   then the host MUST verify if the actual PDS contained in the message
   corresponds to the ULID(peer).

   If any of the above verifications fail, then the host silently
   discards the message; it has completed the I2bis processing.

   If both verifications are successful, then the host proceeds to look
   for a context state for the initiator. The host looks for a context
   with the extracted ULID pair and FII. If none exist, then STATE of
   the (non-existing) context is viewed as being IDLE; thus, the actions
   depend on the STATE as follows:

   o If the STATE is IDLE (i.e., the context does not exist), the host
      allocates a Context Tag (CT(local)), creates the context state for
      the context, and sets its STATE to ESTABLISHED. The host SHOULD
      NOT use the Packet Context Tag in the I2bis message for CT(local);
      instead, it should pick a new random Context Tag just as when it
      processes an I2 message. It records CT(peer) and the peer's
      locator set as well as its own locator set in the context. It
      SHOULD perform the HBA/CGA verification of the peer's locator set
      at this point in time, as specified in Section 7.2. Then the host
      sends an R2 message back as specified in Section 7.14.

   o If the STATE is I1-SENT, then the host verifies if the source
      locator is included in Ls(peer) or in the Locator List contained
      in the I2bis message and if the HBA/CGA verification for this
      specific locator is successful.

      * If this is not the case, then the message is silently
         discarded. The context STATE remains unchanged.

      * If this is the case, then the host updates the context
         information (CT(peer), Ls(peer)) with the data contained in the
         I2bis message, and the host MUST send an R2 message back as
         specified below. Note that before updating Ls(peer)
         information, the host SHOULD perform the HBA/CGA validation of
         the peer's locator set at this point in time, as specified in
         Section 7.2. The host moves to ESTABLISHED STATE.

   o If the STATE is ESTABLISHED, I2-SENT, or I2BIS-SENT, then the host
      determines whether at least one of the two following conditions
      hold: i) if the source locator is included in Ls(peer) or, ii) if
      the source locator is included in the Locator List contained in
      the I2bis message and if the HBA/CGA verification for this
      specific locator is successful.

      * If none of the two aforementioned conditions hold, then the
         message is silently discarded. The context STATE remains
         unchanged.

      * If at least one of the two aforementioned conditions hold, then
         the host updates the context information (CT(peer), Ls(peer))
         with the data contained in the I2bis message, and the host MUST
         send an R2 message back, as specified in Section 7.14. Note
         that before updating Ls(peer) information, the host SHOULD
         perform the HBA/CGA validation of the peer's locator set at
         this point in time, as specified in Section 7.2. The context
         STATE remains unchanged.

8. Handling ICMP Error Messages

   The routers in the path as well as the destination might generate
   ICMP error messages. In some cases, the Shim6 can take action and
   solve the problem that resulted in the error. In other cases, the
   Shim6 layer cannot solve the problem, and it is critical that these
   packets make it back up to the ULPs so that they can take appropriate
   action.

   This is an implementation issue in the sense that the mechanism is
   completely local to the host itself. But the issue of how ICMP
   errors are correctly dispatched to the ULP on the host are important;
   hence, this section specifies the issue.

   All ICMP messages MUST be delivered to the ULP in all cases, except
   when Shim6 successfully acts on the message (e.g., selects a new
   path). There SHOULD be a configuration option to unconditionally
   deliver all ICMP messages (including ones acted on by shim6) to the
   ULP.

   According to that recommendation, the following ICMP error messages
   should be processed by the Shim6 layer and not passed to the ULP:

      ICMP error Destination Unreachable, with codes:
         0 (No route to destination)
         1 (Communication with destination administratively prohibited)
         2 (Beyond scope of source address)
         3 (Address unreachable)
         5 (Source address failed ingress/egress policy)
         6 (Reject route to destination)

      ICMP Time exceeded error.

      ICMP Parameter problem error, with the parameter that caused the
      error being a Shim6 parameter.

   The following ICMP error messages report problems that cannot be
   addressed by the Shim6 layer and that should be passed to the ULP (as
   described below):

      ICMP Packet too big error.

      ICMP Destination Unreachable with Code 4 (Port unreachable).

      ICMP Parameter problem (if the parameter that caused the problem
      is not a Shim6 parameter).

```
                 +--------------+
                 | IPv6 Header  |
                 |              |
                 +--------------+
                 |   ICMPv6     |
                 |   Header     |
      - -        +--------------+   - -
                 | IPv6 Header  |
                 | src, dst as  |   Can be dispatched
      IPv6       | sent by ULP  |   unmodified to ULP
                 | on host      |   ICMP error handler
      Packet     +--------------+
                 |     ULP      |
      in         |    Header    |
                 +--------------+
      Error      |              |
                 ~     Data     ~
                 |              |
      - -        +--------------+   - -
```
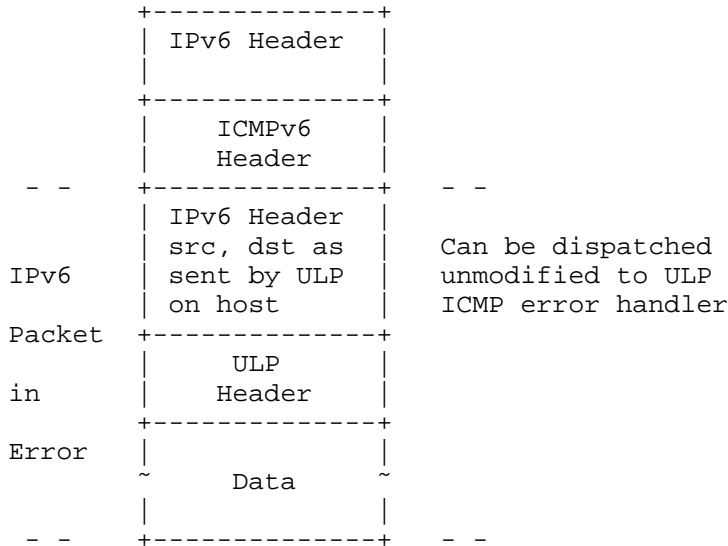
                Figure 8: ICMP Error Handling without the
                      Shim6 Payload Extension Header

   When the ULP packets are sent without the Shim6 Payload Extension
   header -- that is, while the initial locators=ULIDs are working --
   this introduces no new concerns; an implementation's existing
   mechanism for delivering these errors to the ULP will work. See
   Figure 8.

   But when the shim on the transmitting side inserts the Shim6 Payload
   Extension header and replaces the ULIDs in the IP address fields with
   some other locators, then an ICMP error coming back will have a
   "packet in error", which is not a packet that the ULP sent. Thus,
   the implementation will have to apply reverse mapping to the "packet
   in error" before passing the ICMP error up to the ULP, including the
   ICMP extensions defined in [25]. See Figure 9.

```
              +--------------+
              | IPv6 Header  |
              |              |
              +--------------+
              |    ICMPv6    |
              |    Header    |
      - -     +--------------+    - -
              | IPv6 Header  |
              | src, dst as  |    Needs to be
      IPv6    | modified by  |    transformed to
              | shim on host |    have ULIDs
              +--------------+    in src, dst fields,
     Packet   | Shim6 ext.   |    and Shim6 Ext.
              |    Header     |    header removed
       in     +--------------+    before it can be
              |  Transport   |    dispatched to the ULP
     Error    |    Header    |    ICMP error handler.
              +--------------+
              |              |
              ~     Data     ~
              |              |
      - -     +--------------+    - -
```
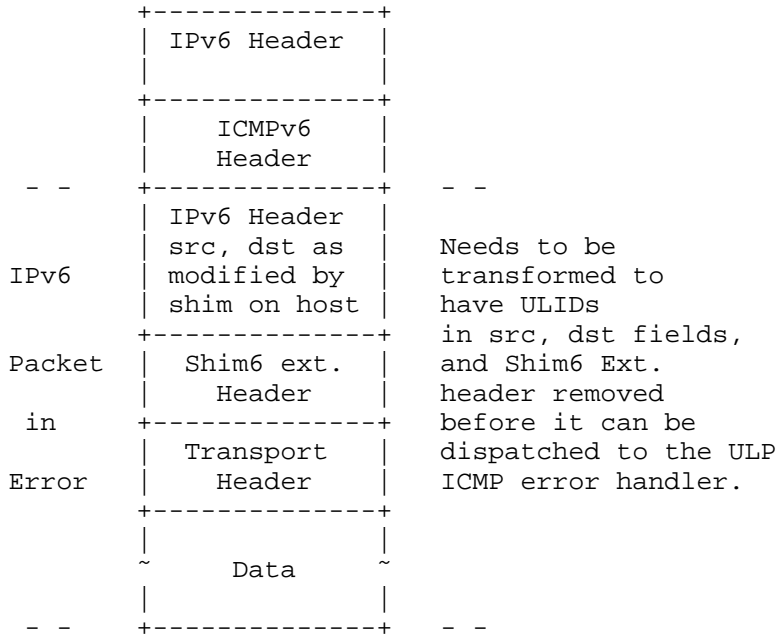
   Figure 9: ICMP Error Handling with the Shim6 Payload Extension Header

   Note that this mapping is different than when receiving packets from
   the peer with Shim6 Payload Extension headers because, in that case,
   the packets contain CT(local). But the ICMP errors have a "packet in
   error" with a Shim6 Payload Extension header containing CT(peer).
   This is because they were intended to be received by the peer. In
   any case, since the <Source Locator, Destination Locator, CT(peer)>
   has to be unique when received by the peer, the local host should
   also only be able to find one context that matches this tuple.

   If the ICMP error is a "packet too big", the reported MTU must be
   adjusted to be 8 octets less, since the shim will add 8 octets when
   sending packets.

   After the "packet in error" has had the original ULIDs inserted, then
   this Shim6 Payload Extension header can be removed. The result is a
   "packet in error" that is passed to the ULP which looks as if the
   shim did not exist.

9. Teardown of the ULID-Pair Context

   Each host can unilaterally decide when to tear down a ULID-pair
   context. It is RECOMMENDED that hosts do not tear down the context
   when they know that there is some upper-layer protocol that might use

the context. For example, an implementation might know this if there
is an open socket that is connected to the ULID(peer). However,
there might be cases when the knowledge is not readily available to
the shim layer, for instance, for UDP applications that do not
connect their sockets or for any application that retains some
higher-level state across (TCP) connections and UDP packets.

Thus, it is RECOMMENDED that implementations minimize premature
teardown by observing the amount of traffic that is sent and received
using the context, and tear down the state only after it appears
quiescent. A reasonable approach would be to not tear down a context
until at least 5 minutes have passed since the last message was sent
or received using the context. (Note that packets that use the ULID
pair as a locator pair and that do not require address rewriting by
the Shim6 layer are also considered as packets using the associated
Shim6 context.)

Since there is no explicit, coordinated removal of the context state,
there are potential issues around Context Tag reuse. One end might
remove the state and potentially reuse that Context Tag for some
other communication, and the peer might later try to use the old
context (which it didn't remove). The protocol has mechanisms to
recover from this, which work whether the state removal was total and
accidental (e.g., crash and reboot of the host) or just a garbage
collection of shim state that didn't seem to be used. However, the
host should try to minimize the reuse of Context Tags by trying to
randomly cycle through the 2^47 Context Tag values. (See Appendix C
for a summary of how the recovery works in the different cases.)

10. Updating the Peer

   The Update Request and Acknowledgement are used both to update the
   list of locators (only possible when CGA is used to verify the
   locator(s)) and to update the preferences associated with each
   locator.

10.1. Sending Update Request Messages

   When a host has a change in the locator set, it can communicate this
   to the peer by sending an Update Request. When a host has a change
   in the preferences for its locator set, it can also communicate this
   to the peer. The Update Request message can include just a Locator
   List option (to convey the new set of locators), just a Locator
   Preferences option, or both a new Locator List and new Locator
   Preferences.

Should the host send a new Locator List, the host picks a new random,
local generation number, records this in the context, and puts it in
the Locator List option. Any Locator Preference option, whether sent
in the same Update Request or in some future Update Request, will use
that generation number to make sure the preferences get applied to
the correct version of the locator list.

The host picks a random Request Nonce for each update and keeps the
same nonce for any retransmissions of the Update Request. The nonce
is used to match the acknowledgement with the request.

The Update Request message can also include a CGA Parameter Data
Structure (this is needed if the CGA PDS was not previously
exchanged). If CGA (and not HBA) is used to verify one or more of
the locators included in the locator list, then a CGA Signature
option containing the signature must also be included in the Update
Request message.

10.2. Retransmitting Update Request Messages

If the host does not receive an Update Acknowledgement R2 message in
response to the Update Request message after UPDATE_TIMEOUT time,
then it needs to retransmit the Update Request message. The
retransmissions should use a retransmission timer with binary
exponential backoff to avoid creating congestion issues for the
network when lots of hosts perform Update Request retransmissions.
Also, the actual timeout value should be randomized between 0.5 and
1.5 of the nominal value to avoid self-synchronization.

Should there be no response, the retransmissions continue forever.
The binary exponential backoff stops at MAX_UPDATE_TIMEOUT. But the
only way the retransmissions would stop when there is no
acknowledgement is when Shim6, through the REAP protocol or some
other mechanism, decides to discard the context state due to lack of
ULP usage in combination with no responses to the REAP protocol.

10.3. Newer Information while Retransmitting

There can be at most one outstanding Update Request message at any
time. Thus until, for example, an update with a new Locator List has
been acknowledged, any newer Locator List or new Locator Preferences
cannot just be sent. However, when there is newer information and
the older information has not yet been acknowledged, the host can,
instead of waiting for an acknowledgement, abandon the previous
update and construct a new Update Request (with a new Request Nonce)
that includes the new information as well as the information that
hasn't yet been acknowledged.

For example, if the original locator list was just (A1, A2), and if
an Update Request with the Locator List (A1, A3) is outstanding, and
the host determines that it should both add A4 to the locator list
and mark A1 as BROKEN, then it would need to:

o Pick a new random Request Nonce for the new Update Request.

o Pick a new random generation number for the new locator list.

o Form the new locator list: (A1, A3, A4).

o Form a Locator Preference option that uses the new generation
   number and has the BROKEN flag for the first locator.

o Send the Update Request and start a retransmission timer.

Any Update Acknowledgement that doesn't match the current Request
Nonce (for instance, an acknowledgement for the abandoned Update
Request) will be silently ignored.

10.4. Receiving Update Request Messages

A host MUST silently discard any received Update Request messages
that do not satisfy all of the following validity checks in addition
to those specified in Section 12.3:

o The Hdr Ext Len field is at least 1, i.e., the length is at least
   16 octets.

Upon the reception of an Update Request message, the host extracts
the Context Tag from the message. It then looks for a context that
has a CT(local) that matches the Context Tag. If no such context is
found, it sends an R1bis message as specified in Section 7.17.

Since Context Tags can be reused, the host MUST verify that the IPv6
Source Address field is part of Ls(peer) and that the IPv6
Destination Address field is part of Ls(local). If this is not the
case, the sender of the Update Request has a stale context that
happens to match the CT(local) for this context. In this case, the
host MUST send an R1bis message and otherwise ignore the Update
Request message.

If a CGA Parameter Data Structure (PDS) is included in the message,
then the host MUST verify if the actual PDS contained in the packet
corresponds to the ULID(peer). If this verification fails, the
message is silently discarded.

Then, depending on the STATE of the context:

o If ESTABLISHED, proceed to process message.

o If I1-SENT, discard the message and stay in I1-SENT.

o If I2-SENT, send I2 and proceed to process the message.

o If I2BIS-SENT, send I2bis and proceed to process the message.

The verification issues for the locators carried in the Update
Request message are specified in Section 7.2. If the locator list
cannot be verified, this procedure should send a Shim6 Error message
with Error Code=2. In any case, if it cannot be verified, there is
no further processing of the Update Request.

Once any Locator List option in the Update Request has been verified,
the peer generation number in the context is updated to be the one in
the Locator List option.

If the Update Request message contains a Locator Preference option,
then the generation number in the preference option is compared with
the peer generation number in the context. If they do not match,
then the host generates a Shim6 Error message with Error Code=3 and
with the Pointer field referring to the first octet in the Locator
List Generation number in the Locator Preference option. In
addition, if the number of elements in the Locator Preference option
does not match the number of locators in Ls(peer), then a Shim6 Error
message with Error Code=4 is sent with the Pointer field referring to
the first octet of the Length field in the Locator Preference option.
In both cases of failure, no further processing is performed for the
Update Request message.

If the generation numbers match, the locator preferences are recorded
in the context.

Once the Locator List option (if present) has been verified and any
new locator list or locator preferences have been recorded, the host
sends an Update Acknowledgement message, copying the nonce from the
request and using the CT(peer) as the Receiver Context Tag.

Any new locators (or, more likely, new locator preferences) might
result in the host wanting to select a different locator pair for the
context -- for instance, if the Locator Preferences option lists the
current Lp(peer) as BROKEN. The host uses the reachability
exploration procedure described in [4] to verify that the new locator
is reachable before changing Lp(peer).

10.5. Receiving Update Acknowledgement Messages

   A host MUST silently discard any received Update Acknowledgement
   messages that do not satisfy all of the following validity checks in
   addition to those specified in Section 12.3:

   o The Hdr Ext Len field is at least 1, i.e., the length is at least
      16 octets.

   Upon the reception of an Update Acknowledgement message, the host
   extracts the Context Tag and the Request Nonce from the message. It
   then looks for a context that has a CT(local) that matches the
   Context Tag. If no such context is found, it sends an R1bis message
   as specified in Section 7.17.

   Since Context Tags can be reused, the host MUST verify that the IPv6
   Source Address field is part of Ls(peer) and that the IPv6
   Destination Address field is part of Ls(local). If this is not the
   case, the sender of the Update Acknowledgement has a stale context
   that happens to match the CT(local) for this context. In this case,
   the host MUST send an R1bis message and otherwise ignore the Update
   Acknowledgement message.

   Then, depending on the STATE of the context:

   o If ESTABLISHED, proceed to process message.

   o If I1-SENT, discard the message and stay in I1-SENT.

   o If I2-SENT, send R2 and proceed to process the message.

   o If I2BIS-SENT, send R2 and proceed to process the message.

   If the Request Nonce doesn't match the nonce for the last sent Update
   Request for the context, then the Update Acknowledgement is silently
   ignored. If the nonce matches, then the update has been completed
   and the Update retransmit timer can be reset.

11. Sending ULP Payloads

   When there is no context state for the ULID pair on the sender, there
   is no effect on how ULP packets are sent. If the host is using some
   heuristic for determining when to perform a deferred context
   establishment, then the host might need to do some accounting (count
   the number of packets sent and received) even before there is a ULID-
   pair context.

If the context is not in ESTABLISHED or I2BIS-SENT STATE, then there
is also no effect on how the ULP packets are sent. Only in the
ESTABLISHED and I2BIS-SENT STATEs does the host have CT(peer) and
Ls(peer) set.

If there is a ULID-pair context for the ULID pair, then the sender
needs to verify whether the context uses the ULIDs as locators --
that is, whether Lp(peer) == ULID(peer) and Lp(local) == ULID(local).

If this is the case, then packets can be sent unmodified by the shim.
If it is not the case, then the logic in Section 11.1 will need to be
used.

There will also be some maintenance activity relating to
(un)reachability detection, whether or not packets are sent with the
original locators. The details of this are out of scope for this
document and are specified in [4].

11.1. Sending ULP Payload after a Switch

When sending packets, if there is a ULID-pair context for the ULID
pair, and if the ULID pair is no longer used as the locator pair,
then the sender needs to transform the packet. Apart from replacing
the IPv6 Source and Destination fields with a locator pair, an
8-octet header is added so that the receiver can find the context and
inverse the transformation.

If there has been a failure causing a switch, and later the context
switches back to sending things using the ULID pair as the locator
pair, then there is no longer a need to do any packet transformation
by the sender; hence, there is no need to include the 8-octet
Extension header.

First, the IP address fields are replaced. The IPv6 Source Address
field is set to Lp(local) and the Destination Address field is set to
Lp(peer). Note that this MUST NOT cause any recalculation of the ULP
checksums, since the ULP checksums are carried end-to-end and the ULP
pseudo-header contains the ULIDs that are preserved end-to-end.

The sender skips any "Routing Sublayer Extension headers" that the
ULP might have included; thus, it skips any Hop-by-Hop Extension
header, any Routing header, and any Destination Options header that
is followed by a Routing header. After any such headers, the Shim6
Extension header will be added. This might be before a Fragment
header, a Destination Options header, an ESP or AH header, or a ULP
header.

The inserted Shim6 Payload Extension header includes the peer's
Context Tag. It takes on the Next Header value from the preceding
Extension header, since that Extension header will have a Next Header
value of Shim6.

12. Receiving Packets

The receive side of the communication can receive packets associated
to a Shim6 context, with or without the Shim6 Extension header. In
case the ULID pair is being used as a locator pair, the packets
received will not have the Shim6 Extension header and will be
processed by the Shim6 layer as described below. If the received
packet does carry the Shim6 Extension header, as in normal IPv6
receive-side packet processing, the receiver parses the (extension)
headers in order. Should it find a Shim6 Extension header, it will
look at the "P" field in that header. If this bit is zero, then the
packet must be passed to the Shim6 payload handling for rewriting.
Otherwise, the packet is passed to the Shim6 control handling.

12.1. Receiving Payload without Extension Headers

The receiver extracts the IPv6 Source and Destination fields and uses
this to find a ULID-pair context, such that the IPv6 address fields
match the ULID(local) and ULID(peer). If such a context is found,
the context appears not to be quiescent; this should be remembered in
order to avoid tearing down the context and for reachability
detection purposes as described in [4]. The host continues with the
normal processing of the IP packet.

12.2. Receiving Shim6 Payload Extension Headers

The receiver extracts the Context Tag from the Shim6 Payload
Extension header and uses this to find a ULID-pair context. If no
context is found, the receiver SHOULD generate an R1bis message (see
Section 7.17).

Then, depending on the STATE of the context:

o If ESTABLISHED, proceed to process message.

o If I1-SENT, discard the message and stay in I1-SENT.

o If I2-SENT, send I2 and proceed to process the message.

o If I2BIS-SENT, send I2bis and proceed to process the message.

With the context in hand, the receiver can now replace the IP address
fields with the ULIDs kept in the context. Finally, the Shim6
Payload Extension header is removed from the packet (so that the ULP
doesn't get confused by it), and the Next Header value in the
preceding header is set to be the actual protocol number for the
payload. Then the packet can be passed to the protocol identified by
the Next Header value (which might be some function associated with
the IP endpoint sublayer or a ULP).

If the host is using some heuristic for determining when to perform a
deferred context establishment, then the host might need to do some
accounting (count the number of packets sent and received) for
packets that do not have a Shim6 Extension header and for which there
is no context. But the need for this depends on what heuristics the
implementation has chosen.

12.3. Receiving Shim Control Messages

A shim control message has the Checksum field verified. The Shim
Header Length field is also verified against the length of the IPv6
packet to make sure that the shim message doesn't claim to end past
the end of the IPv6 packet. Finally, it checks that neither the IPv6
Destination field nor the IPv6 Source field is a multicast address or
an unspecified address. If any of those checks fail, the packet is
silently dropped.

The message is then dispatched based on the shim message type. Each
message type is then processed as described elsewhere in this
document. If the packet contains a shim message type that is unknown
to the receiver, then a Shim6 Error message with Error Code=0 is
generated and sent back. The Pointer field is set to point at the
first octet of the shim message type.

All the control messages can contain any options with C=0. If there
is any option in the message with C=1 that isn't known to the host,
then the host MUST send a Shim6 Error message with Error Code=1 with
the Pointer field referencing the first octet of the Option Type.

12.4. Context Lookup

We assume that each shim context has its own STATE machine. We
assume that a dispatcher delivers incoming packets to the STATE
machine that it belongs to. Here, we describe the rules used for the
dispatcher to deliver packets to the correct shim context STATE
machine.

There is one STATE machine per identified context that is
conceptually identified by the ULID pair and Forked Instance
Identifier (which is zero by default) or identified by CT(local).
However, the detailed lookup rules are more complex, especially
during context establishment.

Clearly, if the required context is not established, it will be in
IDLE STATE.

During context establishment, the context is identified as follows:

o I1 packets: Deliver to the context associated with the ULID pair
   and the Forked Instance Identifier.

o I2 packets: Deliver to the context associated with the ULID pair
   and the Forked Instance Identifier.

o R1 packets: Deliver to the context with the locator pair included
   in the packet and the Initiator Nonce included in the packet (R1
   does not contain a ULID pair or the CT(local)). If no context
   exists with this locator pair and Initiator Nonce, then silently
   discard.

o R2 packets: Deliver to the context with the locator pair included
   in the packet and the Initiator Nonce included in the packet (R2
   does not contain a ULID pair or the CT(local)). If no context
   exists with this locator pair and Initiator Nonce, then silently
   discard.

o R1bis packets: Deliver to the context that has the locator pair
   and the CT(peer) equal to the Packet Context Tag included in the
   R1bis packet.

o I2bis packets: Deliver to the context associated with the ULID
   pair and the Forked Instance Identifier.

o Shim6 Payload Extension headers: Deliver to the context with
   CT(local) equal to the Receiver Context Tag included in the
   packet.

o Other control messages (Update, Keepalive, Probe): Deliver to the
   context with CT(local) equal to the Receiver Context Tag included
   in the packet. Verify that the IPv6 Source Address field is part
   of Ls(peer) and that the IPv6 Destination Address field is part of
   Ls(local). If not, send an R1bis message.

   o Shim6 Error messages and ICMP errors that contain a Shim6 Payload
      Extension header or other shim control packet in the "packet in
      error": Use the "packet in error" for dispatching as follows.
      Deliver to the context with CT(peer) equal to the Receiver Context
      Tag -- Lp(local) being the IPv6 source address and Lp(peer) being
      the IPv6 destination address.

   In addition, the shim on the sending side needs to be able to find
   the context state when a ULP packet is passed down from the ULP. In
   that case, the lookup key is the pair of ULIDs and FII=0. If we have
   a ULP API that allows the ULP to do context forking, then presumably
   the ULP would pass down the Forked Instance Identifier.

13. Initial Contact

   The initial contact is some non-shim communication between two ULIDs,
   as described in Section 2. At that point in time, there is no
   activity in the shim.

   Whether or not the shim ends up being used (e.g., the peer might not
   support Shim6), it is highly desirable that the initial contact can
   be established even if there is a failure for one or more IP
   addresses.

   The approach taken is to rely on the applications and the transport
   protocols to retry with different source and destination addresses,
   consistent with what is already specified in "Default Address
   Selection for IPv6" [7] as well as with some fixes to that
   specification [9], to make it try different source addresses and not
   only different destination addresses.

   The implementation of such an approach can potentially result in long
   timeouts. For instance, consider a naive implementation at the
   socket API that uses getaddrinfo() to retrieve all destination
   addresses and then tries to bind() and connect() to try all source
   and destination address combinations and waits for TCP to time out
   for each combination before trying the next one.

   However, if implementations encapsulate this in some new connect-by-
   name() API and use non-blocking connect calls, it is possible to
   cycle through the available combinations in a more rapid manner until
   a working source and destination pair is found. Thus, the issues in
   this domain are issues of implementations and the current socket API,
   and not issues of protocol specification. In all honesty, while
   providing an easy to use connect-by-name() API for TCP and other
   connection-oriented transports is easy, providing a similar

capability at the API for UDP is hard due to the protocol itself not
providing any "success" feedback. Yet, even the UDP issue is one of
APIs and implementation.

14. Protocol Constants

   The protocol uses the following constants:

   I1_RETRIES_MAX = 4

   I1_TIMEOUT = 4 seconds

   NO_R1_HOLDDOWN_TIME = 1 min

   ICMP_HOLDDOWN_TIME = 10 min

   I2_TIMEOUT = 4 seconds

   I2_RETRIES_MAX = 2

   I2bis_TIMEOUT = 4 seconds

   I2bis_RETRIES_MAX = 2

   VALIDATOR_MIN_LIFETIME = 30 seconds

   UPDATE_TIMEOUT = 4 seconds

   MAX_UPDATE_TIMEOUT = 120 seconds

   The retransmit timers (I1_TIMEOUT, I2_TIMEOUT, UPDATE_TIMEOUT) are
   subject to binary exponential backoff as well as to randomization
   across a range of 0.5 and 1.5 times the nominal (backed off) value.
   This removes any risk of synchronization between lots of hosts
   performing independent shim operations at the same time.

   The randomization is applied after the binary exponential backoff.
   Thus, the first retransmission would happen based on a uniformly
   distributed random number in the range of [0.5*4, 1.5*4] seconds; the
   second retransmission, [0.5*8, 1.5*8] seconds after the first one,
   etc.

15. Implications Elsewhere

15.1. Congestion Control Considerations

   When the locator pair currently used for exchanging packets in a
   Shim6 context becomes unreachable, the Shim6 layer will divert the
   communication through an alternative locator pair, which in most
   cases will result in redirecting the packet flow through an
   alternative network path. In this case, it is recommended that the
   Shim6 follows the recommendation defined in [21] and informs the
   upper layers about the path change, in order to allow the congestion
   control mechanisms of the upper layers to react accordingly.

15.2. Middle-Boxes Considerations

   Data packets belonging to a Shim6 context carrying the Shim6 Payload
   header contain alternative locators other than the ULIDs in the
   Source and Destination Address fields of the IPv6 header. On the
   other hand, the upper layers of the peers involved in the
   communication operate on the ULID pair presented to them by the Shim6
   layer, rather than on the locator pair contained in the IPv6 header
   of the actual packets. It should be noted that the Shim6 layer does
   not modify the data packets but, because a constant ULID pair is
   presented to upper layers irrespective of the locator pair changes,
   the relation between the upper-layer header (such as TCP, UDP, ICMP,
   ESP, etc) and the IPv6 header is modified. In particular, when the
   Shim6 Extension header is present in the packet, if those data
   packets are TCP, UDP, or ICMP packets, the pseudo-header used for the
   checksum calculation will contain the ULID pair, rather than the
   locator pair contained in the data packet.

   It is possible that some firewalls or other middle-boxes will try to
   verify the validity of upper-layer sanity checks of the packet on the
   fly. If they do that based on the actual source and destination
   addresses contained in the IPv6 header without considering the Shim6
   context information (in particular, without replacing the locator
   pair by the ULID pair used by the Shim6 context), such verifications
   may fail. Those middle-boxes need to be updated in order to be able
   to parse the Shim6 Payload header and find the next header. It is
   recommended that firewalls and other middle-boxes do not drop packets
   that carry the Shim6 Payload header with apparently incorrect upper-
   layer validity checks that involve the addresses in the IPv6 header
   for their computation, unless they are able to determine the ULID
   pair of the Shim6 context associated to the data packet and use the
   ULID pair for the verification of the validity check.

In the particular case of TCP, UDP, and ICMP checksums, it is
recommended that firewalls and other middle-boxes do not drop TCP,
UDP, and ICMP packets that carry the Shim6 Payload header with
apparently incorrect checksums when using the addresses in the IPv6
header for the pseudo-header computation, unless they are able to
determine the ULID pair of the Shim6 context associated to the data
packet and use the ULID pair to determine the checksum that must be
present in a packet with addresses rewritten by Shim6.

In addition, firewalls that today pass limited traffic, e.g.,
outbound TCP connections, would presumably block the Shim6 protocol.
This means that even when Shim6-capable hosts are communicating, the
I1 messages would be dropped; hence, the hosts would not discover
that their peer is Shim6-capable. This is, in fact, a benefit since,
if the hosts managed to establish a ULID-pair context, the firewall
would probably drop the "different" packets that are sent after a
failure (those using the Shim6 Payload Extension header with a TCP
packet inside it). Thus, stateful firewalls that are modified to
pass Shim6 messages should also be modified to pass the Shim6 Payload
Extension header so that the shim can use the alternate locators to
recover from failures. This presumably implies that the firewall
needs to track the set of locators in use by looking at the Shim6
control exchanges. Such firewalls might even want to verify the
locators using the HBA/CGA verification themselves, which they can do
without modifying any of the Shim6 packets through which they pass.

15.3. Operation and Management Considerations

This section considers some aspects related to the operations and
management of the Shim6 protocol.

Deployment of the Shim6 protocol: The Shim6 protocol is a host-based
solution. So, in order to be deployed, the stacks of the hosts using
the Shim6 protocol need to be updated to support it. This enables an
incremental deployment of the protocol since it does not require a
flag day for the deployment -- just single host updates. If the
Shim6 solution will be deployed in a site, the host can be gradually
updated to support the solution. Moreover, for supporting the Shim6
protocol, only end hosts need to be updated and no router changes are
required. However, it should be noted that, in order to benefit from
the Shim6 protocol, both ends of a communication should support the
protocol, meaning that both hosts must be updated to be able to use
the Shim6 protocol. Nevertheless, the Shim6 protocol uses a deferred
context-setup capability that allows end hosts to establish normal
IPv6 communications and, later on, if both endpoints are Shim6-
capable, establish the Shim6 context using the Shim6 protocol. This

has an important deployment benefit, since Shim6-enabled nodes can
talk perfectly to non-Shim6-capable nodes without introducing any
problem into the communication.

Configuration of Shim6-capable nodes: The Shim6 protocol itself does
not require any specific configuration to provide its basic features.
The Shim6 protocol is designed to provide a default service to upper
layers that should satisfy general applications. The Shim6 layer
would automatically attempt to protect long-lived communications by
triggering the establishment of the Shim6 context using some
predefined heuristics. Of course, if some special tunning is
required by some applications, this may require additional
configuration. Similar considerations apply to a site attempting to
perform some forms of traffic engineering by using different
preferences for different locators.

Address and prefix configuration: The Shim6 protocol assumes that, in
a multihomed site, multiple prefixes will be available. Such
configuration can increase the operation work in a network. However,
it should be noted that the capability of having multiple prefixes in
a site and multiple addresses assigned to an interface is an IPv6
capability that goes beyond the Shim6 case, and it is expected to be
widely used. So, even though this is the case for Shim6, we consider
that the implications of such a configuration is beyond the
particular case of Shim6 and must be addressed for the generic IPv6
case. Nevertheless, Shim6 also assumes the usage of CGA/HBA
addresses by Shim6 hosts. This implies that Shim6-capable hosts
should configure addresses using HBA/CGA generation mechanisms.
Additional consideration about this issue can be found at [19].

15.4. Other Considerations

The general Shim6 approach as well as the specifics of this proposed
solution have implications elsewhere, including:

o Applications that perform referrals or callbacks using IP
   addresses as the 'identifiers' can still function in limited ways,
   as described in [18]. But, in order for such applications to be
   able to take advantage of the multiple locators for redundancy,
   the applications need to be modified to either use Fully Qualified
   Domain Names as the 'identifiers' or they need to pass all the
   locators as the 'identifiers', i.e., the 'identifier' from the
   application's perspective becomes a set of IP addresses instead of
   a single IP address.

o Signaling protocols for QoS or for other things that involve
   having devices in the network path look at IP addresses and port
   numbers (or at IP addresses and Flow Labels) need to be invoked on

   the hosts when the locator pair changes due to a failure. At that
   point in time, those protocols need to inform the devices that a
   new pair of IP addresses will be used for the flow. Note that
   this is the case even though this protocol, unlike some earlier
   proposals, does not overload the Flow Label as a Context Tag; the
   in-path devices need to know about the use of the new locators
   even though the Flow Label stays the same.

   o MTU implications. By computing a minimum over the recently
      observed path MTUs, the path MTU mechanisms we use are robust
      against different packets taking different paths through the
      Internet. When Shim6 fails over from using one locator pair to
      another, this means that packets might travel over a different
      path through the Internet; hence, the path MTU might be quite
      different. In order to deal with this change in the MTU, the
      usage of Packetization Layer Path MTU Discovery as defined in [24]
      is recommended.

      The fact that the shim will add an 8-octet Shim6 Payload Extension
      header to the ULP packets after a locator switch can also affect
      the usable path MTU for the ULPs. In this case, the MTU change is
      local to the sending host; thus, conveying the change to the ULPs
      is an implementation matter. By conveying the information to the
      transport layer, it can adapt and reduce the Maximum Segment Size
      (MSS) accordingly.

16. Security Considerations

   This document satisfies the concerns specified in [15] as follows:

   o The HBA [3] and CGA [2] techniques for verifying the locators to
      prevent an attacker from redirecting the packet stream to
      somewhere else, prevent threats described in Sections 4.1.1,
      4.1.2, 4.1.3, and 4.2 of [15]. These two techniques provide a
      similar level of protection but also provide different
      functionality with different computational costs.

      The HBA mechanism relies on the capability of generating all the
      addresses of a multihomed host as an unalterable set of
      intrinsically bound IPv6 addresses, known as an HBA set. In this
      approach, addresses incorporate a cryptographic one-way hash of
      the prefix set available into the interface identifier part. The
      result is that the binding between all the available addresses is
      encoded within the addresses themselves, providing hijacking
      protection. Any peer using the shim protocol node can efficiently
      verify that the alternative addresses proposed for continuing the
      communication are bound to the initial address through a simple
      hash calculation.

   In a CGA-based approach, the address used as the ULID is a CGA
   that contains a hash of a public key in its interface identifier.
   The result is a secure binding between the ULID and the associated
   key pair. This allows each peer to use the corresponding private
   key to sign the shim messages that convey locator set information.
   The trust chain in this case is the following: the ULID used for
   the communication is securely bound to the key pair because it
   contains the hash of the public key, and the locator set is bound
   to the public key through the signature.

   Either of these two mechanisms, HBA and CGA, provides time-shifted
   attack protection (as described in Section 4.1.2 of [15]), since
   the ULID is securely bound to a locator set that can only be
   defined by the owner of the ULID. The minimum acceptable key
   length for RSA keys used in the generation of CGAs MUST be at
   least 1024 bits. Any implementation should follow prudent
   cryptographic practice in determining the appropriate key lengths.

   o 3rd party flooding attacks, described in Section 4.3 of [15], are
      prevented by requiring a Shim6 peer to perform a successful
      Reachability probe + reply exchange before accepting a new locator
      for use as a packet destination.

   o The first message does not create any state on the responder.
      Essentially, a 3-way exchange is required before the responder
      creates any state. This means that a state-based DoS attack
      (trying to use up all memory on the responder) at least requires
      the attacker to create state, consuming his own resources; it also
      provides an IPv6 address that the attacker was using.

   o The context-establishment messages use nonces to prevent replay
      attacks, which are described in Section 4.1.4 of [15], and to
      prevent off-path attackers from interfering with the
      establishment.

   o Every control message of the Shim6 protocol, past the context
      establishment, carry the Context Tag assigned to the particular
      context. This implies that an attacker needs to discover that
      Context Tag before being able to spoof any Shim6 control message
      as described in Section 4.4 of [15]. Such discovery probably
      requires an attacker to be along the path in order to sniff the
      Context Tag value. The result is that, through this technique,
      the Shim6 protocol is protected against off-path attackers.

16.1. Interaction with IPSec

   Shim6 has two modes of processing data packets. If the ULID pair is
   also the locator pair being used, then the data packet is not
   modified by Shim6. In this case, the interaction with IPSec is
   exactly the same as if the Shim6 layer was not present in the host.

   If the ULID pair differs from the current locator pair for that Shim6
   context, then Shim6 will take the data packet, replace the ULIDs
   contained in the IP Source and Destination Address fields with the
   current locator pair, and add the Shim6 extension with the
   corresponding Context Tag. In this case, as is mentioned in Section
   1.6, Shim6 conceptually works as a tunnel mechanism, where the inner
   header contains the ULID and the outer header contains the locators.
   The main difference is that the inner header is "compressed" and a
   compression tag, namely the Context Tag, is added to decompress the
   inner header at the receiving end.

   In this case, the interaction between IPSec and Shim6 is then similar
   to the interaction between IPSec and a tunnel mechanism. When the
   packet is generated by the upper-layer protocol, it is passed to the
   IP layer containing the ULIDs in the IP Source and Destination field.
   IPSec is then applied to this packet. Then the packet is passed to
   the Shim6 sublayer, which "encapsulates" the received packet and
   includes a new IP header containing the locator pair in the IP Source
   and Destination field. This new IP packet is in turn passed to IPSec
   for processing, just as in the case of a tunnel. This can be viewed
   as if IPSec is located both above and below the Shim6 sublayer and as
   if IPSec policies apply both to ULIDs and locators.

   When IPSec processed the packet after the Shim6 sublayer has
   processed it (i.e., the packet carrying the locators in the IP Source
   and Destination Address field), the Shim6 sublayer may have added the
   Shim6 Extension header. In that case, IPSec needs to skip the Shim6
   Extension header to find the selectors for the next layer's protocols
   (e.g., TCP, UDP, Stream Control Transmission Protocol (SCTP)).

   When a packet is received at the other end, it is processed based on
   the order of the extension headers. Thus, if an ESP or AH header
   precedes a Shim6 header, that determines the order. Shim6 introduces
   the need to do policy checks, analogous to how they are done for
   tunnels, when Shim6 receives a packet and the ULID pair for that
   packet is not identical to the locator pair in the packet.

16.2. Residual Threats

   Some of the residual threats in this proposal are:

   o An attacker that arrives late on the path (after the context has
      been established) can use the R1bis message to cause one peer to
      re-create the context and, at that point in time, can observe all
      of the exchange. But this doesn't seem to open any new doors for
      the attacker since such an attacker can observe the Context Tags
      that are being used and, once known, can use those to send bogus
      messages.

   o An attacker present on the path in order to find out the Context
      Tags can generate an R1bis message after it has moved off the
      path. For this packet to be effective, it needs to have a source
      locator that belongs to the context; thus, there cannot be "too
      much" ingress filtering between the attacker's new location and
      the communicating peers. But this doesn't seem to be that severe
      because, once the R1bis causes the context to be re-established, a
      new pair of Context Tags will be used, which will not be known to
      the attacker. If this is still a concern, we could require a
      2-way handshake, "did you really lose the state?", in response to
      the error message.

   o It might be possible for an attacker to try random 47-bit Context
      Tags and see if they can cause disruption for communication
      between two hosts. In particular, in the case of payload packets,
      the effects of such an attack would be similar to those of an
      attacker sending packets with a spoofed source address. In the
      case of control packets, it is not enough to find the correct
      Context Tag -- additional information is required (e.g., nonces,
      proper source addresses; see previous bullet for the case of
      R1bis). If a 47-bit tag, which is the largest that fits in an
      8-octet Extension header, isn't sufficient, one could use an even
      larger tag in the Shim6 control messages and use the low-order 47
      bits in the Shim6 Payload Extension header.

   o When the Shim6 Payload Extension header is used, an attacker that
      can guess the 47-bit random Context Tag can inject packets into
      the context with any source locator. Thus, if there is ingress
      filtering between the attacker and its target, this could
      potentially allow the attacker to bypass the ingress filtering.
      However, in addition to guessing the 47-bit Context Tag, the
      attacker also needs to find a context where, after the receiver's
      replacement of the locators with the ULIDs, the ULP checksum is
      correct. But even this wouldn't be sufficient with ULPs like TCP,
      since the TCP port numbers and sequence numbers must match an
      existing connection. Thus, even though the issues for off-path

      attackers injecting packets are different than today with ingress
      filtering, it is still very hard for an off-path attacker to
      guess. If IPsec is applied, then the issue goes away completely.

   o The validator included in the R1 and R1bis packets is generated as
      a hash of several input parameters. While most of the inputs are
      actually determined by the sender, and only the secret value S is
      unknown to the sender, the resulting protection is deemed to be
      enough since it would be easier for the attacker to just obtain a
      new validator by sending an I1 packet than to perform all the
      computations required to determine the secret S. Nevertheless, it
      is recommended that the host change the secret S periodically.

17. IANA Considerations

   IANA allocated a new IP Protocol Number value (140) for the Shim6
   Protocol.

   IANA recorded a CGA message type for the Shim6 protocol in the CGA
   Extension Type Tags registry with the value 0x4A30 5662 4858 574B
   3655 416F 506A 6D48.

   IANA established a Shim6 Parameter Registry with four components:
   Shim6 Type registrations, Shim6 Options registrations, Shim6 Error
   Code registrations, and Shim6 Verification Method registrations.

   The initial contents of the Shim6 Type registry are as follows:

   +------------+---+
   | Type Value | Message |
   +------------+---+
0	RESERVED
1	I1 (first establishment message from the initiator)
2	R1 (first establishment message from the responder)
3	I2 (2nd establishment message from the initiator)
4	R2 (2nd establishment message from the responder)
5	R1bis (Reply to reference to non-existent context)
6	I2bis (Reply to a R1bis message)
7-59	Allocated using Standards action
60-63	For Experimental use
64	Update Request
65	Update Acknowledgement
66	Keepalive
67	Probe Message
68	Error Message
69-123	Allocated using Standards action
124-127	For Experimental use
   +------------+---+

The initial contents of the Shim6 Options registry are as follows:

```
+-------------+-------------------------------+
|    Type     |          Option Name          |
+-------------+-------------------------------+
|      0      |            RESERVED            |
|      1      |      Responder Validator       |
|      2      |          Locator List          |
|      3      |      Locator Preferences        |
|      4      |  CGA Parameter Data Structure  |
|      5      |         CGA Signature          |
|      6      |            ULID Pair           |
|      7      |    Forked Instance Identifier   |
|     8-9     | Allocated using Standards action |
|     10      |    Keepalive Timeout Option     |
|  11-16383   | Allocated using Standards action |
| 16384-32767 |       For Experimental use      |
+-------------+-------------------------------+
```

The initial contents of the Shim6 Error Code registry are as follows:

```
+------------+---------------------------------------------+
| Code Value |                 Description                  |
+------------+---------------------------------------------+
|     0      |          Unknown Shim6 message type          |
|     1      |         Critical Option not recognized       |
|     2      |        Locator verification method failed     |
|     3      | Locator List Generation number out of sync   |
|     4      |        Error in the number of locators        |
|    5-19    |        Allocated using Standards action        |
|   120-127  |       Reserved for debugging purposes         |
+------------+---------------------------------------------+
```

The initial contents of the Shim6 Verification Method registry are as
follows:

```
+---------+---------------------------------+
|  Value  |       Verification Method        |
+---------+---------------------------------+
|    0    |             RESERVED             |
|    1    |               CGA                |
|    2    |               HBA                |
|  3-200  |  Allocated using Standards action |
| 201-254 |       For Experimental use        |
|   255   |             RESERVED             |
+---------+---------------------------------+
```

18. Acknowledgements

   Over the years, many people active in the multi6 and shim6 WGs have
   contributed ideas and suggestions that are reflected in this
   specification. Special thanks to the careful comments from Sam
   Hartman, Cullen Jennings, Magnus Nystrom, Stephen Kent, Geoff Huston,
   Shinta Sugimoto, Pekka Savola, Dave Meyer, Deguang Le, Jari Arkko,
   Iljitsch van Beijnum, Jim Bound, Brian Carpenter, Sebastien Barre,
   Matthijs Mekking, Dave Thaler, Bob Braden, Wesley Eddy, Pasi Eronen,
   and Tom Henderson on earlier versions of this document.

19. References

19.1. Normative References

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

   [2]   Aura, T., "Cryptographically Generated Addresses (CGA)",
          RFC 3972, March 2005.

   [3]   Bagnulo, M., "Hash-Based Addresses (HBA)", RFC 5535, June 2009.

   [4]   Arkko, J. and I. van Beijnum, "Failure Detection and Locator
          Pair Exploration Protocol for IPv6 Multihoming", RFC 5534,
          June 2009.

19.2. Informative References

   [5]   Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for
          specifying the location of services (DNS SRV)", RFC 2782,
          February 2000.

   [6]   Ferguson, P. and D. Senie, "Network Ingress Filtering:
          Defeating Denial of Service Attacks which employ IP Source
          Address Spoofing", BCP 38, RFC 2827, May 2000.

   [7]   Draves, R., "Default Address Selection for Internet Protocol
          version 6 (IPv6)", RFC 3484, February 2003.

   [8]   Nordmark, E., "Multihoming without IP Identifiers", Work
          in Progress, July 2004.

   [9]   Bagnulo, M., "Updating RFC 3484 for multihoming support", Work
          in Progress, November 2007.

   [10] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson,
         "RTP: A Transport Protocol for Real-Time Applications", STD 64,
         RFC 3550, July 2003.

   [11] Abley, J., Black, B., and V. Gill, "Goals for IPv6 Site-
         Multihoming Architectures", RFC 3582, August 2003.

   [12] Rajahalme, J., Conta, A., Carpenter, B., and S. Deering, "IPv6
         Flow Label Specification", RFC 3697, March 2004.

   [13] Eastlake, D., Schiller, J., and S. Crocker, "Randomness
         Requirements for Security", BCP 106, RFC 4086, June 2005.

   [14] Hinden, R. and B. Haberman, "Unique Local IPv6 Unicast
         Addresses", RFC 4193, October 2005.

   [15] Nordmark, E. and T. Li, "Threats Relating to IPv6 Multihoming
         Solutions", RFC 4218, October 2005.

   [16] Huitema, C., "Ingress filtering compatibility for IPv6
         multihomed sites", Work in Progress, September 2005.

   [17] Bagnulo, M. and E. Nordmark, "SHIM - MIPv6 Interaction", Work
         in Progress, July 2005.

   [18] Nordmark, E., "Shim6-Application Referral Issues", Work
         in Progress, July 2005.

   [19] Bagnulo, M. and J. Abley, "Applicability Statement for the
         Level 3 Multihoming Shim Protocol (Shim6)", Work in Progress,
         July 2007.

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

   [21] Schuetz, S., Koutsianas, N., Eggert, L., Eddy, W., Swami, Y.,
         and K. Le, "TCP Response to Lower-Layer Connectivity-Change
         Indications", Work in Progress, February 2008.

   [22] Williams, N. and M. Richardson, "Better-Than-Nothing Security:
         An Unauthenticated Mode of IPsec", RFC 5386, November 2008.

   [23] Komu, M., Bagnulo, M., Slavov, K., and S. Sugimoto, "Socket
         Application Program Interface (API) for Multihoming Shim", Work
         in Progress, November 2008.

   [24] Mathis, M. and J. Heffner, "Packetization Layer Path MTU
         Discovery", RFC 4821, March 2007.

   [25] Bonica, R., Gan, D., Tappan, D., and C. Pignataro, "Extended
         ICMP to Support Multi-Part Messages", RFC 4884, April 2007.

Appendix A. Possible Protocol Extensions

   During the development of this protocol, several issues have been
   brought up that are important to address but that do not need to be
   in the base protocol itself; instead, these can be done as extensions
   to the protocol. The key ones are:

   o As stated in the assumptions in Section 3, in order for the Shim6
      protocol to be able to recover from a wide range of failures (for
      instance, when one of the communicating hosts is single-homed) and
      to cope with a site's ISPs that do ingress filtering based on the
      source IPv6 address, there is a need for the host to be able to
      influence the egress selection from its site. Further discussion
      of this issue is captured in [16].

   o Is there need for keeping the list of locators private between the
      two communicating endpoints? We can potentially accomplish that
      when using CGA (not when using HBA), but only at the cost of doing
      some public key encryption and decryption operations as part of
      the context establishment. The suggestion is to leave this for a
      future extension to the protocol.

   o Defining some form of end-to-end "compression" mechanism that
      removes the need to include the Shim6 Payload Extension header
      when the locator pair is not the ULID pair.

   o Supporting the dynamic setting of locator preferences on a site-
      wide basis and using the Locator Preference option in the Shim6
      protocol to convey these preferences to remote communicating
      hosts. This could mirror the DNS SRV record's notion of priority
      and weight.

   o Specifying APIs in order for the ULPs to be aware of the locators
      that the shim is using and to be able to influence the choice of
      locators (controlling preferences as well as triggering a locator-
      pair switch). This includes providing APIs that the ULPs can use
      to fork a shim context.

   o Determining whether it is feasible to relax the suggestions for
      when context state is removed so that one can end up with an
      asymmetric distribution of the context state and still get (most
      of) the shim benefits. For example, the busy server would go
      through the context setup but would quickly remove the context
      state after this (in order to save memory); however, the not-so-
      busy client would retain the context state. The context-recovery
      mechanism presented in Section 7.5 would then re-create the state
      should the client send either a shim control message (e.g., Probe
      message because it sees a problem) or a ULP packet in a Shim6

Payload Extension header (because it had earlier failed over to an
alternative locator pair but had been silent for a while). This
seems to provide the benefits of the shim as long as the client
can detect the failure. If the client doesn't send anything and
it is the server that tries to send, then it will not be able to
recover because the shim on the server has no context state and
hence doesn't know any alternate locator pairs.

o Study what it would take to make the Shim6 control protocol not
   rely at all on a stable source locator in the packets. This can
   probably be accomplished by having all the shim control messages
   include the ULID-pair option.

o If each host might have lots of locators, then the current
   requirement to include essentially all of them in the I2 and R2
   messages might be constraining. If this is the case, we can look
   into using the CGA Parameter Data Structure for the comparison,
   instead of the prefix sets, to be able to detect context
   confusion. This would place some constraint on a (logical) only
   using, for example, one CGA public key; it would also require some
   carefully crafted rules on how two PDSs are compared for "being
   the same host". But if we don't expect more than a handful of
   locators per host, then we don't need this added complexity.

o ULP-specified timers for the reachability detection mechanism
   (which can be particularly useful when there are forked contexts).

o Pre-verify some "backup" locator pair, so that the failover time
   can be shorter.

o Study how Shim6 and Mobile IPv6 might interact [17].

Appendix B. Simplified STATE Machine

   The STATEs are defined in Section 6.2. The intent is for the
   stylized description below to be consistent with the textual
   description in the specification; however, should they conflict, the
   textual description is normative.

The following table describes the possible actions in STATE IDLE and
their respective triggers:

```
+--------------------+------------------------------------------+
| Trigger            | Action                                   |
+--------------------+------------------------------------------+
| Receive I1         | Send R1 and stay in IDLE                 |
|                    |                                          |
| Heuristics trigger | Send I1 and move to I1-SENT              |
| a new context      |                                          |
| establishment      |                                          |
|                    |                                          |
| Receive I2, verify | If successful, send R2 and move to       |
| validator and      | ESTABLISHED                              |
| RESP Nonce         |                                          |
|                    | If fail, stay in IDLE                    |
|                    |                                          |
| Receive I2bis,     | If successful, send R2 and move to       |
| verify validator   | ESTABLISHED                              |
| and RESP Nonce     |                                          |
|                    | If fail, stay in IDLE                    |
|                    |                                          |
| R1, R1bis, R2      | N/A (This context lacks the required info|
|                    | for the dispatcher to deliver them)      |
|                    |                                          |
| Receive Payload    | Send R1bis and stay in IDLE              |
| Extension header   |                                          |
| or other control   |                                          |
| packet             |                                          |
+--------------------+------------------------------------------+
```

The following table describes the possible actions in STATE I1-SENT
and their respective triggers:

```
+--------------------+------------------------------------------+
| Trigger            | Action                                   |
+--------------------+------------------------------------------+
| Receive R1, verify | If successful, send I2 and move to I2-SENT |
| INIT Nonce         |                                          |
|                    | If fail, discard and stay in I1-SENT     |
|                    |                                          |
| Receive I1         | Send R2 and stay in I1-SENT              |
|                    |                                          |
| Receive R2, verify | If successful, move to ESTABLISHED       |
| INIT Nonce         |                                          |
|                    | If fail, discard and stay in I1-SENT     |
|                    |                                          |
| Receive I2, verify | If successful, send R2 and move to       |
| validator and RESP | ESTABLISHED                              |
| Nonce              |                                          |
|                    | If fail, discard and stay in I1-SENT     |
|                    |                                          |
| Receive I2bis,     | If successful, send R2 and move to       |
| verify validator   | ESTABLISHED                              |
| and RESP Nonce     |                                          |
|                    | If fail, discard and stay in I1-SENT     |
|                    |                                          |
| Timeout, increment | If counter =< I1_RETRIES_MAX, send I1 and |
| timeout counter    | stay in I1-SENT                          |
|                    |                                          |
|                    | If counter > I1_RETRIES_MAX, go to E-FAILED |
|                    |                                          |
| Receive ICMP payload| Move to E-FAILED                        |
| unknown error      |                                          |
|                    |                                          |
| R1bis              | N/A (Dispatcher doesn't deliver since    |
|                    | CT(peer) is not set)                     |
|                    |                                          |
| Receive Payload    | Discard and stay in I1-SENT              |
| Extension header   |                                          |
| or other control   |                                          |
| packet             |                                          |
+--------------------+------------------------------------------+
```

The following table describes the possible actions in STATE I2-SENT
and their respective triggers:

```
+--------------------+------------------------------------------+
| Trigger            | Action                                   |
+--------------------+------------------------------------------+
| Receive R2, verify | If successful, move to ESTABLISHED        |
| INIT Nonce         |                                          |
|                    | If fail, stay in I2-SENT                 |
|                    |                                          |
| Receive I1         | Send R2 and stay in I2-SENT              |
|                    |                                          |
| Receive I2,        | Send R2 and stay in I2-SENT              |
| verify validator   |                                          |
| and RESP Nonce     |                                          |
|                    |                                          |
| Receive I2bis,     | Send R2 and stay in I2-SENT              |
| verify validator   |                                          |
| and RESP Nonce     |                                          |
|                    |                                          |
| Receive R1         | Discard and stay in I2-SENT              |
|                    |                                          |
| Timeout, increment | If counter =< I2_RETRIES_MAX, send I2 and |
| timeout counter    | stay in I2-SENT                          |
|                    |                                          |
|                    | If counter > I2_RETRIES_MAX, send I1 and go |
|                    | to I1-SENT                               |
|                    |                                          |
| R1bis              | N/A (Dispatcher doesn't deliver since    |
|                    | CT(peer) is not set)                     |
|                    |                                          |
| Receive Payload    | Accept and send I2 (probably R2 was sent |
| Extension header   | by peer and lost)                        |
| or other control   |                                          |
| packet             |                                          |
+--------------------+------------------------------------------+
```

The following table describes the possible actions in STATE I2BIS-
SENT and their respective triggers:

```
+--------------------+-------------------------------------------+
| Trigger            | Action                                    |
+--------------------+-------------------------------------------+
| Receive R2, verify | If successful, move to ESTABLISHED         |
| INIT Nonce         |                                           |
|                    | If fail, stay in I2BIS-SENT               |
|                    |                                           |
| Receive I1         | Send R2 and stay in I2BIS-SENT            |
|                    |                                           |
| Receive I2,        | Send R2 and stay in I2BIS-SENT            |
| verify validator   |                                           |
| and RESP Nonce     |                                           |
|                    |                                           |
| Receive I2bis,     | Send R2 and stay in I2BIS-SENT            |
| verify validator   |                                           |
| and RESP Nonce     |                                           |
|                    |                                           |
| Receive R1         | Discard and stay in I2BIS-SENT            |
|                    |                                           |
| Timeout, increment | If counter =< I2_RETRIES_MAX, send I2bis  |
| timeout counter    | and stay in I2BIS-SENT                     |
|                    |                                           |
|                    | If counter > I2_RETRIES_MAX, send I1 and  |
|                    | go to I1-SENT                             |
|                    |                                           |
| R1bis              | N/A (Dispatcher doesn't deliver since     |
|                    | CT(peer) is not set)                      |
|                    |                                           |
| Receive Payload    | Accept and send I2bis (probably R2 was    |
| Extension header   | sent by peer and lost)                    |
| or other control   |                                           |
| packet             |                                           |
+--------------------+-------------------------------------------+
```

The following table describes the possible actions in STATE
ESTABLISHED and their respective triggers:

```
+--------------------+--------------------------------------------+
| Trigger            | Action                                     |
+--------------------+--------------------------------------------+
| Receive I1, compare| If no match, send R1 and stay in ESTABLISHED|
| CT(peer) with      |                                            |
| received CT        | If match, send R2 and stay in ESTABLISHED  |
|                    |                                            |
|                    |                                            |
| Receive I2, verify | If successful, send R2 and stay in         |
| validator and RESP | ESTABLISHED                                |
| Nonce              |                                            |
|                    | Otherwise, discard and stay in ESTABLISHED |
|                    |                                            |
| Receive I2bis,     | If successful, send R2 and stay in         |
| verify validator   | ESTABLISHED                                |
| and RESP Nonce     |                                            |
|                    | Otherwise, discard and stay in ESTABLISHED |
|                    |                                            |
| Receive R2         | Discard and stay in ESTABLISHED            |
|                    |                                            |
| Receive R1         | Discard and stay in ESTABLISHED            |
|                    |                                            |
| Receive R1bis      | Send I2bis and move to I2BIS-SENT          |
|                    |                                            |
|                    |                                            |
| Receive Payload    | Process and stay in ESTABLISHED            |
| Extension header   |                                            |
| or other control   |                                            |
| packet             |                                            |
|                    |                                            |
| Implementation-    | Discard state and go to IDLE               |
| specific heuristic |                                            |
| (e.g., No open ULP |                                            |
| sockets and idle   |                                            |
| for some time )    |                                            |
+--------------------+--------------------------------------------+
```

The following table describes the possible actions in STATE E-FAILED
and their respective triggers:

```
+--------------------+------------------------------------------+
| Trigger            | Action                                   |
+--------------------+------------------------------------------+
| Wait for           | Go to IDLE                               |
| NO_R1_HOLDDOWN_TIME |                                         |
|                    |                                          |
| Any packet         | Process as in IDLE                       |
+--------------------+------------------------------------------+
```

The following table describes the possible actions in STATE NO-
SUPPORT and their respective triggers:

```
+--------------------+------------------------------------------+
| Trigger            | Action                                   |
+--------------------+------------------------------------------+
| Wait for           | Go to IDLE                               |
| ICMP_HOLDDOWN_TIME |                                          |
|                    |                                          |
| Any packet         | Process as in IDLE                       |
+--------------------+------------------------------------------+
```

B.1. Simplified STATE Machine Diagram

```
                                      Timeout/Null   +------------+
                      I1/R1        +-----------------| NO SUPPORT |
             Payload or Control/R1bis  |             +------------+
                    +---------+    |   |                    ^
                    |         |    |   |            ICMP Error/Null|
                    |         V    V   V                          |
                 +----------------+ Timeout/Null  +----------+    |
                 |                |<--------------| E-FAILED |    |
                 +-|     IDLE     |               +----------+    |
     I2 or I2bis/R2 | |           |                    ^          |
                 | +----------------+         (Tiemout#>MAX)/Null|    |
                 |    ^           |                    |          | |
                 |    |           | +------+           |          | |
     I2 or I2bis/R2   |   Heuristic/I1|         I1/R2  | |
     Payload/Null  |  |           |            Control/Null | |
     I1/R1 or R2   |  |           |            Payload/Null | |
     R1 or R2/Null | +--+ Heuristic/Null |     (Tiemout#<MAX)/I1 | |
        +---------+ |  |           |               +--------+ | |
        |         V V  |           |               |        V  | |
    +------------------+   R2/Null |          +---------------+
    |                  |   I2 or I2bis/R2  +------->|               |
    |   ESTABLISHED    |<--------------------------|    I1-SENT    |
    |                  |                           |               |
    +------------------+                           +---------------+
      |   ^          ^                                |   ^     ^
      |   |         |R2/Null               +-------------+  |     |
      |   |      +----------+              |R1/I2        |     |
      |   |      |          |              V             |     |
      |   |    +------------------+                      |     |
      |   |    |                  |    +-------------+   |     |
      |   |    |     I2-SENT      |    |(Timeout#>Max)/I1 |     |
      |   |    |                  |                      |     |
      |   |    +------------------+                      |     |
      |   |      |            ^                          |     |
      |   |    +--------------+                          |     |
      |   |      I1 or I2bis or I2/R2                    |     |
      |   |   (Timeout#<Max) or Payload/I2               |     |
      |   |      R1 or R1bis/Null                        |     |
      |   |                             (Timeout#>Max)/I1 |
      |  +-------+      +----------------------------------------+
      | R2/Null|       |
      |       V   |
      |    +------------------+
      |    |                  |<-+ (Timeout#<Max)/I2bis
    +-------->|   I2bis-SENT   |  | I1 or I2 or I2bis/R2
    R1bis/I2bis |              |--+ R1 or R1bis/Null
             +------------------+    Payload/I2bis
```

Appendix C. Context Tag Reuse

   The Shim6 protocol doesn't have a mechanism for coordinated state
   removal between the peers because such state removal doesn't seem to
   help, given that a host can crash and reboot at any time. A result
   of this is that the protocol needs to be robust against a Context Tag
   being reused for some other context. This section summarizes the
   different cases in which a Tag can be reused, and how the recovery
   works.

   The different cases are exemplified by the following case. Assume
   hosts A and B were communicating using a context with the ULID pair
   <A1, B2>, and that B had assigned Context Tag X to this context. We
   assume that B uses only the Context Tag to demultiplex the received
   Shim6 Payload Extension headers, since this is the more general case.
   Further, we assume that B removes this context state, while A retains
   it. B might then at a later time assign CT(local)=X to some other
   context, at which time, we have several possible cases:

   o The Context Tag is reassigned to a context for the same ULID pair
      <A1, B2>. We've called this "context recovery" in this document.

   o The Context Tag is reassigned to a context for a different ULID
      pair between the same two hosts, e.g., <A3, B3>. We've called
      this "context confusion" in this document.

   o The Context Tag is reassigned to a context between B and another
      host C, for instance, for the ULID pair <C3, B2>. That is a form
      of three-party context confusion.

C.1. Context Recovery

   This case is relatively simple and is discussed in Section 7.5. The
   observation is that since the ULID pair is the same, when either A or
   B tries to establish the new context, A can keep the old context
   while B re-creates the context with the same Context Tag CT(B) = X.

C.2. Context Confusion

   This case is a bit more complex and is discussed in Section 7.6.
   When the new context is created, whether A or B initiates it, host A
   can detect when it receives B's locator set (in the I2 or R2 message)
   in that it ends up with two contexts to the same peer host
   (overlapping Ls(peer) locator sets) that have the same Context Tag:
   CT(peer) = X. At this point in time, host A can clear up any
   possibility of causing confusion by not using the old context to send
   any more packets. It either just discards the old context (it might
   not be used by any ULP traffic, since B had discarded it) or it re-

creates a different context for the old ULID pair (<A1, B2>), for
which B will assign a unique CT(B) as part of the normal context-
establishment mechanism.

C.3. Three-Party Context Confusion

The third case does not have a place where the old state on A can be
verified since the new context is established between B and C. Thus,
when B receives Shim6 Payload Extension headers with X as the Context
Tag, it will find the context for <C3, B2> and, hence, will rewrite
the packets to have C3 in the Source Address field and B2 in the
Destination Address field before passing them up to the ULP. This
rewriting is correct when the packets are in fact sent by host C, but
if host A ever happens to send a packet using the old context, then
the ULP on A sends a packet with ULIDs <A1, B2> and the packet
arrives at the ULP on B with ULIDs <C3, B2>.

This is clearly an error, and the packet will most likely be rejected
by the ULP on B due to a bad pseudo-header checksum. Even if the
checksum is okay (probability 2^{-16}), the ULP isn't likely to have a
connection for those ULIDs and port numbers. And if the ULP is
connection-less, processing the packet is most likely harmless; such
a ULP must be able to copy with random packets being sent by random
peers in any case.

This broken state, where packets are sent from A to B using the old
context on host A, might persist for some time but will not remain
for very long. The unreachability detection on host A will kick in
because it does not see any return traffic (payload or Keepalive
messages) for the context. This will result in host A sending Probe
messages to host B to find a working locator pair. The effect of
this is that host B will notice that it does not have a context for
the ULID pair <A1, B2> and CT(B) = X, which will make host B send an
R1bis packet to re-establish that context. The re-established
context, just like in the previous section, will get a unique CT(B)
assigned by host B; thus, there will no longer be any confusion.

C.4. Summary

In summary, there are cases where a Context Tag might be reused while
some peer retains the state, but the protocol can recover from it.
The probability of these events is low, given the 47-bit Context Tag
size. However, it is important that these recovery mechanisms be
tested. Thus, during development and testing, it is recommended that
implementations not use the full 47-bit space but instead keep, for
example, the top 40 bits as zero, only leaving the host with 128
unique Context Tags. This will help test the recovery mechanisms.

Appendix D. Design Alternatives

   This document has picked a certain set of design choices in order to
   try to work out a bunch of the details and to stimulate discussion.
   But, as has been discussed on the mailing list, there are other
   choices that make sense. This appendix tries to enumerate some
   alternatives.

D.1. Context Granularity

   Over the years, various suggestions have been made whether the shim
   should, even if it operates at the IP layer, be aware of ULP
   connections and sessions and, as a result, be able to make separate
   shim contexts for separate ULP connections and sessions. A few
   different options have been discussed:

   o Each ULP connection maps to its own shim context.

   o The shim is unaware of the ULP notion of connections and just
      operates on a host-to-host (IP address) granularity.

   o Hybrids in which the shim is aware of some ULPs (such as TCP) and
      handles other ULPs on a host-to-host basis.

   Having shim state for every ULP connection potentially means higher
   overhead since the state-setup overhead might become significant;
   there is utility in being able to amortize this over multiple
   connections.

   But being completely unaware of the ULP connections might hamper ULPs
   that want different communication to use different locator pairs, for
   instance, for quality or cost reasons.

   The protocol has a shim that operates with host-level granularity
   (strictly speaking, with ULID-pair granularity) to be able to
   amortize the context establishment over multiple ULP connections.
   This is combined with the ability for Shim6-aware ULPs to request
   context forking so that different ULP traffic can use different
   locator pairs.

D.2. Demultiplexing of Data Packets in Shim6 Communications

   Once a ULID-pair context is established between two hosts, packets
   may carry locators that differ from the ULIDs presented to the ULPs
   using the established context. One of the main functions of the
   Shim6 layer is to perform the mapping between the locators used to
   forward packets through the network and the ULIDs presented to the
   ULP. In order to perform that translation for incoming packets, the

Shim6 layer needs to first identify which of the incoming packets
need to be translated and then perform the mapping between locators
and ULIDs using the associated context. Such operation is called
"demultiplexing". It should be noted that, because any address can
be used both as a locator and as a ULID, additional information,
other than the addresses carried in packets, needs to be taken into
account for this operation.

For example, if a host has addresses A1 and A2 and starts
communicating with a peer with addresses B1 and B2, then some
communication (connections) might use the pair <A1, B1> as ULID and
others might use, for example, <A2, B2>. Initially there are no
failures, so these address pairs are used as locators, i.e., in the
IP address fields in the packets on the wire. But when there is a
failure, the Shim6 layer on A might decide to send packets that used
<A1, B1> as ULIDs using <A2, B2> as the locators. In this case, B
needs to be able to rewrite the IP address field for some packets and
not others, but the packets all have the same locator pair.

In order to accomplish the demultiplexing operation successfully,
data packets carry a Context Tag that allows the receiver of the
packet to determine the shim context to be used to perform the
operation.

Two mechanisms for carrying the Context Tag information have been
considered in depth during the shim protocol design: those carrying
the Context Tag in the Flow Label field of the IPv6 header and those
using a new Extension header to carry the Context Tag. In this
appendix, we will describe the pros and cons of each mechanism and
justify the selected option.

D.2.1. Flow Label

A possible approach is to carry the Context Tag in the Flow Label
field of the IPv6 header. This means that when a Shim6 context is
established, a Flow Label value is associated with this context (and
perhaps a separate Flow Label for each direction).

The simplest way to do this is to have the triple <Flow Label, Source
Locator, Destination Locator> identify the context at the receiver.

The problem with this approach is that, because the locator sets are
dynamic, it is not possible at any given moment to be sure that two
contexts for which the same Context Tag is allocated will have
disjoint locator sets during the lifetime of the contexts.

Suppose that Node A has addresses IPA1, IPA2, IPA3, and IPA4 and that
Host B has addresses IPB1 and IPB2.

Suppose that two different contexts are established between Host A
and Host B.

Context #1 is using IPA1 and IPB1 as ULIDs. The locator set
associated to IPA1 is IPA1 and IPA2, while the locator set associated
to IPB1 is just IPB1.

Context #2 uses IPA3 and IPB2 as ULIDs. The locator set associated
to IPA3 is IPA3 and IPA4, while the locator set associated to IPB2 is
just IPB2.

Because the locator sets of Context #1 and Context #2 are disjoint,
hosts could think that the same Context Tag value can be assigned to
both of them. The problem arrives when, later on, IPA3 is added as a
valid locator for IPA1 in Context #2 and IPB2 is added as a valid
locator for IPB1 in Context #1. In this case, the triple <Flow
Label, Source Locator, Destination Locator> would not identify a
unique context anymore, and correct demultiplexing is no longer
possible.

A possible approach to overcome this limitation is to simply not
repeat the Flow Label values for any communication established in a
host. This basically means that each time a new communication that
is using different ULIDs is established, a new Flow Label value is
assigned to it. By these means, each communication that is using
different ULIDs can be differentiated because each has a different
Flow Label value.

The problem with such an approach is that it requires the receiver of
the communication to allocate the Flow Label value used for incoming
packets, in order to assign them uniquely. For this, a shim
negotiation of the Flow Label value to use in the communication is
needed before exchanging data packets. This poses problems with non-
Shim6-capable hosts, since they would not be able to negotiate an
acceptable value for the Flow Label. This limitation can be lifted
by marking the packets that belong to shim sessions from those that
do not. These markings would require at least a bit in the IPv6
header that is not currently available, so more creative options
would be required, for instance, using new Next Header values to
indicate that the packet belongs to a Shim6-enabled communication and
that the Flow Label carries context information as proposed in [8].
However, even if new Next Header values are used in this way, such an
approach is incompatible with the deferred-establishment capability
of the shim protocol, which is a preferred function since it
suppresses delay due to shim context establishment prior to the
initiation of communication. Such capability also allows nodes to

define at which stage of the communication they decide, based on
their own policies, that a given communication requires protection by
the shim.

In order to cope with the identified limitations, an alternative
approach that does not constrain the Flow Label values that are used
by communications using ULIDs equal to the locators (i.e., no shim
translation) is to only require that different Flow Label values are
assigned to different shim contexts. In such an approach,
communications start with unmodified Flow Label usage (could be zero
or as suggested in [12]). The packets sent after a failure when a
different locator pair is used would use a completely different Flow
Label, and this Flow Label could be allocated by the receiver as part
of the shim context establishment. Since it is allocated during the
context establishment, the receiver of the "failed over" packets can
pick a Flow Label of its choosing (that is unique in the sense that
no other context is using it as a Context Tag), without any
performance impact, respecting that, for each locator pair, the Flow
Label value used for a given locator pair doesn't change due to the
operation of the multihoming shim.

In this approach, the constraint is that Flow Label values being used
as context identifiers cannot be used by other communications that
use non-disjoint locator sets. This means that once a given Flow
Label value has been assigned to a shim context that has a certain
locator sets associated, the same value cannot be used for other
communications that use an address pair that is contained in the
locator sets of the context. This is a constraint in the potential
Flow Label allocation strategies.

A possible workaround to this constraint is to mark shim packets that
require translation, in order to differentiate them from regular IPv6
packets, using the artificial Next Header values described above. In
this case, the Flow Label values constrained are only those of the
packets that are being translated by the shim. This last approach
would be the preferred approach if the Context Tag is to be carried
in the Flow Label field. This is the case not only because it
imposes the minimum constraints to the Flow Label allocation
strategies, limiting the restrictions only to those packets that need
to be translated by the shim, but also because context-loss detection
mechanisms greatly benefit from the fact that shim data packets are
identified as such, allowing the receiving end to identify if a shim
context associated to a received packet is supposed to exist, as will
be discussed in the context-loss detection appendix below.

D.2.2. Extension Header

   Another approach, which is the one selected for this protocol, is to
   carry the Context Tag in a new Extension header. These Context Tags
   are allocated by the receiving end during the Shim6 protocol initial
   negotiation, implying that each context will have two Context Tags,
   one for each direction. Data packets will be demultiplexed using the
   Context Tag carried in the Extension header. This seems a clean
   approach since it does not overload existing fields. However, it
   introduces additional overhead in the packet due to the additional
   header. The additional overhead introduced is 8 octets. However, it
   should be noted that the Context Tag is only required when a locator
   other than the one used as ULID is contained in the packet. Packets
   where both the Source and Destination Address fields contain the
   ULIDs do not require a Context Tag, since no rewriting is necessary
   at the receiver. This approach would reduce the overhead because the
   additional header is only required after a failure. On the other
   hand, this approach would cause changes in the available MTU for some
   packets, since packets that include the Extension header will have an
   MTU that is 8 octets shorter. However, path changes through the
   network can result in a different MTU in any case; thus, having a
   locator change, which implies a path change, affect the MTU doesn't
   introduce any new issues.

D.3. Context-Loss Detection

   In this appendix, we will present different approaches considered to
   detect context loss and potential context-recovery strategies. The
   scenario being considered is the following: Node A and Node B are
   communicating using IPA1 and IPB1. Sometime later, a shim context is
   established between them, with IPA1 and IPB1 as ULIDs and with
   IPA1,...,IPAn and IPB1,...,IPBm as locator sets, respectively.

   It may happen that, later on, one of the hosts (e.g., Host A) loses
   the shim context. The reason for this can be that Host A has a more
   aggressive garbage collection policy than Host B or that an error
   occurred in the shim layer at Host A and resulted in the loss of the
   context state.

   The mechanisms considered in this appendix are aimed at dealing with
   this problem. There are essentially two tasks that need to be
   performed in order to cope with this problem: first, the context loss
   must be detected and, second, the context needs to be recovered/
   re-established.

   Mechanisms for detecting context loss.

These mechanisms basically consist in each end of the context that
periodically sends a packet containing context-specific information
to the other end. Upon reception of such packets, the receiver
verifies that the required context exists. In the case that the
context does not exist, it sends a packet notifying the sender of the
problem.

An obvious alternative for this would be to create a specific context
keepalive exchange, which consists in periodically sending packets
with this purpose. This option was considered and discarded because
it seemed an overkill to define a new packet exchange to deal with
this issue.

Another alternative is to piggyback the context-loss detection
function in other existent packet exchanges. In particular, both
shim control and data packets can be used for this.

Shim control packets can be trivially used for this because they
carry context-specific information. This way, when a node receives
one such packet, it will verify if the context exists. However, shim
control frequency may not be adequate for context-loss detection
since control packet exchanges can be very limited for a session in
certain scenarios.

Data packets, on the other hand, are expected to be exchanged with a
higher frequency but do not necessarily carry context-specific
information. In particular, packets flowing before a locator change
(i.e., a packet carrying the ULIDs in the address fields) do not need
context information since they do not need any shim processing.
Packets that carry locators that differ from the ULIDs carry context
information.

However, we need to make a distinction here between the different
approaches considered to carry the Context Tag -- in particular,
between those approaches where packets are explicitly marked as shim
packets and those approaches where packets are not marked as such.
For instance, in the case where the Context Tag is carried in the
Flow Label and packets are not marked as shim packets (i.e., no new
Next Header values are defined for shim), a receiver that has lost
the associated context is not able to detect that the packet is
associated with a missing context. The result is that the packet
will be passed unchanged to the upper-layer protocol, which in turn
will probably silently discard it due to a checksum error. The
resulting behavior is that the context loss is undetected. This is
one additional reason to discard an approach that carries the Context
Tag in the Flow Label field and does not explicitly mark the shim
packets as such. On the other hand, approaches that mark shim data
packets (like those that use the Extension header or the Flow Label

with new Next Header values) allow the receiver to detect if the
context associated to the received packet is missing. In this case,
data packets also perform the function of a context-loss detection
exchange. However, it must be noted that only those packets that
carry a locator that differs from the ULID are marked. This
basically means that context loss will be detected after an outage
has occurred, i.e., alternative locators are being used.

Summarizing, the proposed context-loss detection mechanisms use shim
control packets and Shim6 Payload Extension headers to detect context
loss. Shim control packets detect context loss during the whole
lifetime of the context, but the expected frequency in some cases is
very low. On the other hand, Shim6 Payload Extension headers have a
higher expected frequency in general, but they only detect context
loss after an outage. This behavior implies that it will be common
that context loss is detected after a failure, i.e., once it is
actually needed. Because of that, a mechanism for recovering from
context loss is required if this approach is used.

Overall, the mechanism for detecting lost context would work as
follows: the end that still has the context available sends a message
referring to the context. Upon the reception of such message, the
end that has lost the context identifies the situation and notifies
the other end of the context-loss event by sending a packet
containing the lost context information extracted from the received
packet.

One option is to simply send an error message containing the received
packets (or at least as much of the received packet that the MTU
allows to fit). One of the goals of this notification is to allow
the other end that still retains context state to re-establish the
lost context. The mechanism to re-establish the lost context
consists in performing the 4-way initial handshake. This is a time-
consuming exchange and, at this point, time may be critical since we
are re-establishing a context that is currently needed (because
context-loss detection may occur after a failure). So another
option, which is the one used in this protocol, is to replace the
error message with a modified R1 message so that the time required to
perform the context-establishment exchange can be reduced. Upon the
reception of this modified R1 message, the end that still has the
context state can finish the context-establishment exchange and
restore the lost context.

D.4. Securing Locator Sets

The adoption of a protocol like SHIM, which allows the binding of a
given ULID with a set of locators, opens the door for different types
of redirection attacks as described in [15]. The goal, in terms of

security, for the design of the shim protocol is to not introduce any
new vulnerability into the Internet architecture. It is a non-goal
to provide additional protection other than what is currently
available in the single-homed IPv6 Internet.

Multiple security mechanisms were considered to protect the shim
protocol. In this appendix we will present some of them.

The simplest option to protect the shim protocol is to use cookies,
i.e., a randomly generated bit string that is negotiated during the
context-establishment phase and then is included in subsequent
signaling messages. By these means, it would be possible to verify
that the party that was involved in the initial handshake is the same
party that is introducing new locators. Moreover, before using a new
locator, an exchange is performed through the new locator, verifying
that the party located at the new locator knows the cookie, i.e.,
that it is the same party that performed the initial handshake.

While this security mechanism does indeed provide a fair amount of
protection, it leaves the door open for so-called time-shifted
attacks. In these attacks, an attacker on the path discovers the
cookie by sniffing any signaling message. After that, the attacker
can leave the path and still perform a redirection attack since, as
he is in possession of the cookie, he can introduce a new locator
into the locator set and can also successfully perform the
reachability exchange if he is able to receive packets at the new
locator. The difference with the current single-homed IPv6 situation
is that in the current situation the attacker needs to be on-path
during the whole lifetime of the attack, while in this new situation
(where only cookie protection is provided), an attacker that was once
on the path can perform attacks after he has left the on-path
location.

Moreover, because the cookie is included in signaling messages, the
attacker can discover the cookie by sniffing any of them, making the
protocol vulnerable during the whole lifetime of the shim context. A
possible approach to increase security is to use a shared secret,
i.e., a bit string that is negotiated during the initial handshake
but that is used as a key to protect following messages. With this
technique, the attacker must be present on the path and sniffing
packets during the initial handshake, since this is the only moment
when the shared secret is exchanged. Though it imposes that the
attacker must be on path at a very specific moment (the establishment
phase), and though it improves security, this approach is still
vulnerable to time-shifted attacks. It should be noted that,
depending on protocol details, an attacker may be able to force the
re-creation of the initial handshake (for instance, by blocking

messages and making the parties think that the context has been
lost); thus, the resulting situation may not differ that much from
the cookie-based approach.

Another option that was discussed during the design of this protocol
was the possibility of using IPsec for protecting the shim protocol.
Now, the problem under consideration in this scenario is how to
securely bind an address that is being used as ULID with a locator
set that can be used to exchange packets. The mechanism provided by
IPsec to securely bind the address that is used with cryptographic
keys is the usage of digital certificates. This implies that an
IPsec-based solution would require a common and mutually trusted
third party to generate digital certificates that bind the key and
the ULID. Considering that the scope of application of the shim
protocol is global, this would imply a global public key
infrastructure (PKI). The major issues with this approach are the
deployment difficulties associated with a global PKI. The other
possibility would be to use some form of opportunistic IPSec, like
Better-Than-Nothing-Security (BTNS) [22]. However, this would still
present some issues. In particular, this approach requires a leap-
of-faith in order to bind a given address to the public key that is
being used, which would actually prevent the most critical security
feature that a Shim6 security solution needs to achieve from being
provided: proving identifier ownership. On top of that, using IPsec
would require to turn on per-packet AH/ESP just for multihoming to
occur.

In general, SHIM6 was expected to work between pairs of hosts that
have no prior arrangement, security association, or common, trusted
third party. It was also seen as undesirable to have to turn on per-
packet AH/ESP just for the multihoming to occur. However, Shim6
should work and have an additional level of security where two hosts
choose to use IPsec.

Another design alternative would have employed some form of
opportunistic or Better-Than-Nothing Security (BTNS) IPsec to perform
these tasks with IPsec instead. Essentially, HIP in opportunistic
mode is very similar to SHIM6, except that HIP uses IPsec, employs
per-packet ESP, and introduces another set of identifiers.

Finally, two different technologies were selected to protect the shim
protocol: HBA [3] and CGA [2]. These two techniques provide a
similar level of protection but also provide different functionality
with different computational costs.

The HBA mechanism relies on the capability of generating all the
addresses of a multihomed host as an unalterable set of intrinsically
bound IPv6 addresses, known as an HBA set. In this approach,

addresses incorporate a cryptographic one-way hash of the prefix set
available into the interface identifier part. The result is that the
binding between all the available addresses is encoded within the
addresses themselves, providing hijacking protection. Any peer using
the shim protocol node can efficiently verify that the alternative
addresses proposed for continuing the communication are bound to the
initial address through a simple hash calculation. A limitation of
the HBA technique is that, once generated, the address set is fixed
and cannot be changed without also changing all the addresses of the
HBA set. In other words, the HBA technique does not support dynamic
addition of address to a previously generated HBA set. An advantage
of this approach is that it requires only hash operations to verify a
locator set, imposing very low computational cost to the protocol.

In a CGA-based approach, the address used as ULID is a CGA that
contains a hash of a public key in its interface identifier. The
result is a secure binding between the ULID and the associated key
pair. This allows each peer to use the corresponding private key to
sign the shim messages that convey locator set information. The
trust chain in this case is the following: the ULID used for the
communication is securely bound to the key pair because it contains
the hash of the public key, and the locator set is bound to the
public key through the signature. The CGA approach then supports
dynamic addition of new locators in the locator set, since in order
to do that the node only needs to sign the new locator with the
private key associated with the CGA used as ULID. A limitation of
this approach is that it imposes systematic usage of public key
cryptography with its associate computational cost.

Either of these two mechanisms, HBA and CGA, provides time-shifted
attack protection, since the ULID is securely bound to a locator set
that can only be defined by the owner of the ULID.

So the design decision adopted was that both mechanisms, HBA and CGA,
are supported. This way, when only stable address sets are required,
the nodes can benefit from the low computational cost offered by HBA,
while when dynamic locator sets are required, this can be achieved
through CGAs with an additional cost. Moreover, because HBAs are
defined as a CGA extension, the addresses available in a node can
simultaneously be CGAs and HBAs, allowing the usage of the HBA and
CGA functionality when needed, without requiring a change in the
addresses used.

D.5. ULID-Pair Context-Establishment Exchange

Two options were considered for the ULID-pair context-establishment
exchange: a 2-way handshake and a 4-way handshake.

A key goal for the design of this exchange was protection against DoS
attacks. The attack under consideration was basically a situation
where an attacker launches a great amount of ULID-pair establishment-
request packets, exhausting the victim's resources similarly to TCP
SYN flooding attacks.

A 4-way handshake exchange protects against these attacks because the
receiver does not create any state associated to a given context
until the reception of the second packet, which contains prior-
contact proof in the form of a token. At this point, the receiver
can verify that at least the address used by the initiator is valid
to some extent, since the initiator is able to receive packets at
this address. In the worst case, the responder can track down the
attacker using this address. The drawback of this approach is that
it imposes a 4-packet exchange for any context establishment. This
would be a great deal if the shim context needed to be established up
front, before the communication can proceed. However, thanks to the
deferred context-establishment capability of the shim protocol, this
limitation has a reduced impact in the performance of the protocol.
(However, it may have a greater impact in the situation of context
recovery, as discussed earlier. However, in this case, it is
possible to perform optimizations to reduce the number of packets as
described above.)

The other option considered was a 2-way handshake with the
possibility to fall back to a 4-way handshake in case of attack. In
this approach, the ULID-pair establishment exchange normally consists
of a 2-packet exchange and does not verify that the initiator has
performed a prior contact before creating context state. In case a
DoS attack is detected, the responder falls back to a 4-way handshake
similar to the one described previously, in order to prevent the
detected attack from proceeding. The main difficulty with this
attack is how to detect that a responder is currently under attack.
It should be noted that, because this is a 2-way exchange, it is not
possible to use the number of half-open sessions (as in TCP) to
detect an ongoing attack; different heuristics need to be considered.

The design decision taken was that, considering the current impact of
DoS attacks and the low impact of the 4-way exchange in the shim
protocol (thanks to the deferred context-establishment capability), a
4-way exchange would be adopted for the base protocol.

D.6. Updating Locator Sets

There are two possible approaches to the addition and removal of
locators: atomic and differential approaches. The atomic approach
essentially sends the complete locator set each time a variation in
the locator set occurs. The differential approach sends the

differences between the existing locator set and the new one. The
atomic approach imposes additional overhead since all of the locator
set has to be exchanged each time, while the differential approach
requires re-synchronization of both ends through changes (i.e.,
requires that both ends have the same idea about what the current
locator set is).

Because of the difficulties imposed by the synchronization
requirement, the atomic approach was selected.

D.7. State Cleanup

There are essentially two approaches for discarding an existing state
about locators, keys, and identifiers of a correspondent node: a
coordinated approach and an unilateral approach.

In the unilateral approach, each node discards information about the
other node without coordination with the other node, based on some
local timers and heuristics. No packet exchange is required for
this. In this case, it would be possible that one of the nodes has
discarded the state while the other node still hasn't. In this case,
a No Context Error message may be required to inform the other node
about the situation; possibly a recovery mechanism is also needed.

A coordinated approach would use an explicit CLOSE mechanism, akin to
the one specified in HIP [20]. If an explicit CLOSE handshake and
associated timer is used, then there would no longer be a need for
the No Context Error message due to a peer having garbage collected
at its end of the context. However, there is still potentially a
need to have a No Context Error message in the case of a complete
state loss of the peer (also known as a crash followed by a reboot).
Only if we assume that the reboot takes at least the time of the
CLOSE timer, or that it is okay to not provide complete service until
CLOSE-timer minutes after the crash, can we completely do away with
the No Context Error message.

In addition, another aspect that is relevant for this design choice
is the context confusion issue. In particular, using a unilateral
approach to discard context state clearly opens up the possibility of
context confusion, where one of the ends unilaterally discards the
context state, while the other does not. In this case, the end that
has discarded the state can re-use the Context Tag value used for the
discarded state for another context, creating potential context
confusion. In order to illustrate the cases where problems would
arise, consider the following scenario:

o Hosts A and B establish context 1 using CTA and CTB as Context
   Tags.

o Later on, A discards context 1 and the Context Tag value CTA
   becomes available for reuse.

o However, B still keeps context 1.

This would create context confusion in the following two cases:

o A new context 2 is established between A and B with a different
   ULID pair (or Forked Instance Identifier), and A uses CTA as the
   Context Tag. If the locator sets used for both contexts are not
   disjoint, we have context confusion.

o A new context is established between A and C, and A uses CTA as
   the Context Tag value for this new context. Later on, B sends
   Payload Extension header and/or control messages containing CTA,
   which could be interpreted by A as belonging to context 2 (if no
   proper care is taken). Again we have context confusion.

One could think that using a coordinated approach would eliminate
such context confusion, making the protocol much simpler. However,
this is not the case, because even in the case of a coordinated
approach using a CLOSE/CLOSE ACK exchange, there is still the
possibility of a host rebooting without having the time to perform
the CLOSE exchange. So, it is true that the coordinated approach
eliminates the possibility of context confusion due to premature
garbage collection, but it does not prevent the same situations due
to a crash and reboot of one of the involved hosts. The result is
that, even if we went for a coordinated approach, we would still need
to deal with context confusion and provide the means to detect and
recover from these situations.

Authors' Addresses

   Erik Nordmark
   Sun Microsystems
   17 Network Circle
   Menlo Park, CA 94025
   USA

   Phone: +1 650 786 2921
   EMail: erik.nordmark@sun.com

   Marcelo Bagnulo
   Universidad Carlos III de Madrid
   Av. Universidad 30
   Leganes, Madrid 28911
   SPAIN

   Phone: +34 91 6248814
   EMail: marcelo@it.uc3m.es
   URI: http://www.it.uc3m.es