SHIM6 WG
Internet-Draft
Expires: November 2, 2007

E. Nordmark
Sun Microsystems
M. Bagnulo
UC3M

May 2007

Shim6: Level 3 Multihoming Shim Protocol for IPv6
draft-ietf-shim6-proto-08.txt

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on November 2, 2007.

Copyright Notice

Copyright (C) The IETF Trust (2007).

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 which is announced in the global IPv6 routing table. The hosts in a site which has multiple provider allocated IPv6 address prefixes, will use the Shim6 protocol specified in this document to setup 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 [16], without assuming that a multihomed site will have a provider independent IPv6 address which is announced in the global IPv6 routing table. The hosts in a site which has multiple provider allocated IPv6 address prefixes, will use the Shim6 protocol specified in this document to setup 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 [8].

The reachability and failure detection mechanisms, including how a new working locator pair is discovered after a failure, are specified in a separate document [9]. 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 that document.

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 [20] through the combination of the HBA/CGA approach specified in a separate document [8] and techniques described in this document.

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

packets have been exchanged.

- 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 a potential extensions to this protocol.

1.2. Non-Goals

The assumption is that 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). Note, however 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, and "locator" 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 [13]. Some extensions are needed to RFC 3484 to try different source addresses, whether or not the Shim6 protocol is used, as outlined in [14]. 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 when the selected ULID is not initially working hence there is a need to switch locators up front.

Using one of the locators as the ULID has certain benefits for applications which have long-lived session state or performs callbacks or referrals, because both the 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 [23] for further discussion of the application implications.

There has been some discussion of using non-routable addresses, such as Unique-Local Addresses (ULAs) [19], 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 not reachable (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 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, it need to be able to do the RPF check. (This isn't much

different than the situation with widely implemented ingress filtering [11] 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 which use RTP [15], since RTP already has an explicit identifier in the form of the 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 and depending 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 open 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) is 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 requiring that any communication using a ULID MUST be terminated when the ULID becomes invalid (due to the underlying prefix becoming invalid). This behaviour 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
             | AH | | ESP | | 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, or routing header). 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 AH and ESP above the multihoming shim means that IPsec can be made to be unaware of locator changes the same way that transport protocols can be unaware. Thus the IPsec security associations remain stable even though the locators are changing. This means that the IP addresses specified in the selectors should be the ULIDs.

Layering the fragmentation header above the multihoming shim makes reassembly robust in the case that there is broken multi-path routing which results in using different paths, hence potentially different source locators, for different fragments. Thus, effectively the multihoming shim layer is placed between the IP endpoint sublayer,

which handles fragmentation, reassembly, and IPsec, and the IP routing sublayer, which selects which next hop and interface to use for sending out packets.

Applications and upper layer protocols use ULIDs which the Shim6 layer map to/from different locators. The Shim6 layer maintains state, called ULID-pair context, per ULID pair (that is, 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, it is independent of the ULPs and any ULP connections. However, the forking capability enables shim-aware ULPs to use more than one locator pair at a time for an 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 being present in every packet, and with a header compression mechanism 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 recreate 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.

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) 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 can't easily be recreated in "multiple prefix multihoming" such as Shim6.

The protocol provides a placeholder, in the form of the Locator Preferences option, which can be used by hosts to express priority and weight values for each locator. This is intentionally made identical to the DNS SRV [10] specification of priority and weight, so that DNS SRV records can be used for initial contact and the shim for failover, and they can use the same way to describe the preferences. But the Locator Preference option is merely a place holder 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

This document uses the terms MUST, SHOULD, RECOMMENDED, MAY, SHOULD NOT and MUST NOT defined 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 IPX, AppleTalk, or IP itself.
interface	A node's attachment to a link.
address	An IP layer name that contains both 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 which 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 this 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 identified by the pair of ULID 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 payload extension headers as belonging to the context.
Current locator pair	Each end of the context has a current locator pair which is used to send packets to the peer. The two ends might use different current locator pairs though.

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 which 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 a FII of zero.
Initial contact	We use this term to refer to the pre-shim communication when some 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 [8].
Cryptographically Generated Addresses (CGA)	A form of IPv6 address where the interface ID is derived from a cryptographic hash of the public key. See [6].
CGA Parameter Data Structure (PDS)	The information that CGA and HBA exchanges in order to inform the peer of how the interface ID was computed. See [6]., [8].

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 in not ordered in any particular way other than maybe what is returned by the DNS.

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

CT(X) is a context tag assigned by X.

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. However, B cannot vary the destination address (i.e., A's locator), since A has only one locator.

The above scenario leads to the assumption that a host should be able to cause different egress paths from its site to be used. The most reasonable approach to accomplish this is to have the host use different source addresses and have the source address affect the selection of the site egress. The details of how this can be accomplished is beyond the scope of this document, but without this capability the ability of the shim to try different "paths" by trying different locator pairs will have limited utility.

The above assumption applies whether or not the ISPs perform ingress filtering.

In addition, 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.

Further discussion of this issue is captured in [21].

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.

4. Protocol Overview

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

- 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 [13] and its extensions [14] work, then there is no action by the shim at this point in time. Any shim context establishment can be deferred until later.

- 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 a timer expiration while active packet exchange is 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.

- Communication continues without any change for the ULP packets. In particular, there are no shim 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 sub-layers for (un)reachability detection.

- 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 switch to using that locator pair.

- Once a working alternative locator pair has been found, the shim will rewrite the packets on transmit, and tag the packets with 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.

- 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.

- 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 this address is no longer recommended to try. 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 preferences 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).

- 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 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 sub-layer) to determine when the state is likely to no longer be used.

NOTE: 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 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 which can fit in an 8-octet extension header), while preserving one bit to differentiate the Shim6 signalling 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 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 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 for 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 which has no semantics in the protocol other then being part of the tuple which 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

4.4. Securing Shim6

The mechanisms are secured using a combination of techniques:

o The HBA technique [8] for verifying the locators to prevent an attacker from redirecting the packet stream to somewhere else.

o Requiring a Reachability Probe+Reply /defined in [9]) 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 of 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, 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. Such discovery probably requires any potential attacker to be along the path in order to be 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 [26].]

R1bis and I2bis messages are defined, which are used to recover a context after it has been lost. A R1bis message is sent when a Shim6 control or 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, and 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 [9].

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 [9].

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 setup some communication). This is handled using the mechanisms in the ULP to try different address pairs as specified in [13] [14]. In the future versions of the protocol, and with a richer API between the ULP and the shim, the shim might be 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 sub-layer and the IP routing sub-layer, 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, 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, but 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 [to be assigned by IANA]. 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 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 Payload extension header and the control messages and 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 which 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 Payload extension headers from control messages.

Shim6 signalling 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. Payload Extension Header Format

The payload extension headers 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 locators 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 which 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 for use to identify the context.

5.3. Common Shim6 Control header

The common part of the header has a next header and header extension length field which is 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 shim 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  |0|     Type    |Type-specific|0|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            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 64-127 will trigger R1bis.
0:	A single bit (set to zero) which allows Shim6 and HIP to have a common header format yet telling Shim6 and HIP messages apart.
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 protocol like ICMPv6, there is no pseudo-header checksum part of the checksum, in order to provide locator agility without having to change the checksum.
Type-specific:	Part of message that is different for different message types.

Type Value	Message
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)
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 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 is 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. Typically 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 a 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 a 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 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.  Just a copy of the
                  Responder Validator option in the R1 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 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 the 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 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 the 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 shim 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 reestablishing 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 a 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 a 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 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 start on a multiple of 8 octet boundary.)

Packet Context Tag: 47-bit unsigned integer. Copied from the Packet Context Tag 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 does 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 the 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, and 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 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 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 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 the 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 a 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 [9].

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 [9].

The goal of this mechanism is to test whether locator pairs work or not in the general case. In particular, this mechanism is to be able to handle the case when one locator pair works in from A to B, and another locator pair works from B to A, but there is no locator pair which 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 from and report that back in probe messages it is sending 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 which contains information that is critical for the Shim6 protocol 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 that, a range of Error Code Types is reserved for that purpose. In particular, implementations may generate Shim6 Error messages with Code Type 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 Value	Description
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 pruposes

Table 2

5.15. Option Formats

The format of the options is a snapshot of the current HIP option format [26]. 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) which 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 option header and the 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 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 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, 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)- it computed,

2)- it computed for the particular context, and

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

Some suggestions on how to generate the validators are captured in Section 7.10.1 and Section 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 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 which is increased by one for each new locator list. This is used to ensure that the index in the Locator Preferences refer 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 i'th octet specifies the verification method for the i'th 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
3-255	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 has 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 [10] 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 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 i'th 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, 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 1 octet flags field followed by a 1 octet priority field. The priority

has the same semantics as the priority in DNS SRV records.

When the Element length equals three, then the element consists of a 1 octet flags field followed by a 1 octet priority field, and a 1 octet weight field. The weight has the same semantics as the weight 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 of a 1 octet flags field followed by a 1 octet priority field, and a 1 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. 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 [6] and [8].

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 value of the correspondent Locator List Option.

3. The subset of locators included in the correspondent Locator List Option which verification method is set to CGA. The locators MUST be included in the order 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 that the ULID for the context differ 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 [9].

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 which contains the following information:

- The state of the context. See Section 6.2.
- The peer ULID; ULID(peer)
- The local ULID; ULID(local)
- The Forked Instance Identifier; FII. This is zero for the default context i.e., when there is no forking.
- The list of peer locators, with their preferences; Ls(peer)
- The generation number for the most recently received, verified peer locator list.
- For each peer locator, the verification method to use (from the Locator List option).
- For each peer locator, a flag whether it has been verified using HBA or CGA, and a bit whether the locator has been probed to verify that the ULID is present at that location.
- The preferred peer locator - used as destination; Lp(peer)
- The set of local locators and the preferences; Ls(local)
- The generation number for the most recently sent Locator List option.
- The preferred local locator - used as source; Lp(local)
- The context tag used to transmit control messages and payload extension headers - allocated by the peer; CT(peer)

o The context to expect in received control messages and payload extension headers - 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 [9].

o During pair exploration, information about the probe messages that have been sent and received as specified in [9].

o During context establishment phase, Init 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 it 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 setup 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 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 unstrucutred tag name space reserved for randomly assigned context tag values,(e.g. following the guidelines described in [18]).

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 which "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 is to verify that the host is indeed reachable at the claimed locator. Such verification is needed both to make sure communication can proceed, but also to prevent 3rd party flooding attacks [20]. These different verifications 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 being part of this context. Thus the HBA/CGA verification SHOULD be performed by the host before the host acknowledges the new locator, by sending 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 before upon the 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 [9].

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, and the host SHOULD generate a Shim6 Error Message with Error Code=2, 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 25.

```
     Initiator                          Responder

      IDLE                               IDLE
           ------------- I1 -------------->
      I1-SENT
           <------------ R1 ---------------
                                         IDLE
           ------------- I2 -------------->
      I2-SENT
           <------------ R2 ---------------
      ESTABLISHED                        ESTABLISHED
```

Figure 25: 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 a I1 after

having sent a 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 a R2, resulting in the message exchange shown in Figure 26. Such behavior is needed for other reasons such as to correctly respond 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 26: 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 27.

```
        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 27: 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 the cases the result is that the peer without state receives a shim message for which it has no context for the context tag.

In all of those cases we can recover the context by having the node which doesn't have a context state, send back an R1bis message, and have then complete the recovery with a I2bis and R2 message as can be seen in Figure 28.

```
       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 28: 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. The peer (that still has the context state) will reply with an R1 message and the full 4-way exchange will be performed again in this case as can be seen in Figure 29.

```
          Host A                             Host B

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

         ESTABLISHED                         IDLE

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

              <------------ I2 ---------------
       Recreate context

       with new CT(peer)                     I2-SENT
       and Ls(peer).

         ESTABLISHED
              ------------- R2 -------------->
         ESTABLISHED                         ESTABLISHED
```

Figure 29: Context loss at sender

7.6. Context confusion

Since each end might garbage collect the context state we can have the case when 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 can not use the recovery mechanisms since there need 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 has dropped it):

o B decides to create a context for ULID pair <A3, B1>, and 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 that A still 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 requirement is that the old context which 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 which 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 setup 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 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 I2 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 to not try again to establish a context with that ULID. However, any such negative caching should retained for at most NO_R1_HOLDDOWN_TIME, to be able to later setup 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 to not try again to establish a context with that ULID. Such negative caching should 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 which matches the ULID pair and the FII.

If no state is found (i.e., the state is IDLE), then the host replies with a 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 a 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 a 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 it is trying to establish a new one for the same ULID-pair. In this case, the host replies with a 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 a R2 message. This completes the I1 processing, with the context state being unchanged.

7.10. Sending R1 messages

When the host needs to send a 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 it 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 a 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

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).

In the case the validator is generated to be included in a R1 message, for each I1 message. The responder use the current counter C value as the Responder Nonce, and use 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

and then the output of the hash function is used as the validator octet string.

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 which 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 then 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, then 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 this value. Besides, the I2 message contains an Initiator Nonce. This is not required to be the same than the one included in the previous I1 message.

The I2 message also includes the Initiator's locator list and the CGA parameter data structure. If CGA (and not HBA) is used to verify the locator list, then Initiator also signs the key parts of the message and includes 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 recieve 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 fails, then the host silently discards the message and 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, it is included in the Locator List contained in the I2 message and 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 a 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, it is included in the Locator List contained in the I2 message and 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 a 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 using its locators and its context tag, copies the Initiator Nonce from the triggering message (I2, I2bis, or I1), and includes the necessary options so that the peer can verify the locators. In particular, the R2 message includes 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 it has received the above messages since they create a new context with a new CT(peer). 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 Where Ls(peer) 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 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 the 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 which 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 a R1bis upon receipt of any control messages that has 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 which 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).

In the case the validator is generated to be included in a R1bis message, for each received payload extension header or control message, the responder use the counter C value as the Responder Nonce, and use 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

and then the output of the hash function is used as the validator octet 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 not 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 a I2bis message, including the computed Responder Validator option, the Packet Context Tag, and the Responder Nonce 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 an 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.

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 recieve an R2 message after retransmitting I2bis messages I2bis_RETRIES_MAX times, the initiator SHOULD fallback 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 fails, then the host silently discard the message and 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, it is included in the Locator List contained in the I2 message and the HBA/CGA verification for this specific locator is successful

  * If this is not the case, then the message is silently discarded. The 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 a 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, it is included in the Locator List contained in the I2 message and the HBA/CGA verification for this specific locator is successful

   * If this is not the case, then the message is silently discarded. The 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 a 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 various ICMP error messages, such as host unreachable, packet too big, and Unrecognized Next Header type. 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.

```
                  +--------------+
                  | 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 30: ICMP error handling without payload extension header

When the ULP packets are sent without the 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 30.

But when the shim on the transmitting side inserts the 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 the reverse mapping to the "packet in error" before passing the ICMP error up to the ULP. See Figure 31.

```
                +--------------+
                | 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 31: ICMP error handling with payload extension header

Note that this mapping is different than when receiving packets from the peer with a payload extension headers, because in that case the packets contain CT(local). But the ICMP errors have a "packet in error" with an 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 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 which 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 which do not connect their sockets, or any application which 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 only after it appears quiescent, tear down the state. A reasonable approach would be not to 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 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 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)), as well as updating the preferences associated with each locator.

10.1. Sending Update Request messages

When a host has a change in the locator set, then 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 (which requires a CGA signature option as well), 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 send 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.

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 the shim, through the Probe protocol or some other mechanism, decides to discard the context state due to lack of ULP usage in combination with no responses to the Probes.

10.3. Newer Information While Retransmitting

There can be at most one outstanding Update Request message at any time. Thus until e.g. an update with a new Locator List has been acknowledged, any even newer Locator List or new Locator Preferences can not 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) which includes the new information as well as the information that hadn'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:

- Pick a new random Request Nonce for the new Update Request.
- Pick a new random Generation number for the new locator list.
- Form the new locator list - (A1, A3, A4)
- Form a Locator Preference option which uses the new generation number and has the BROKEN flag for the first locator.
- Send the Update Request and start a retransmission timer.

Any Update Acknowledgement which 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:

- 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 which has a CT(local) that matches the context tag. If no such context is found, it sends a 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 which happens to match the CT(local) for this context. In this case the host MUST send a 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:

- If ESTABLISHED: Proceed to process message.

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

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

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

The verification issues for the locators carried in the Locator Update message are specified in Section 7.2. If the locator list can not be verified, this procedure should send a Shim6 Error message with Error Code=2. In any case, if it can not 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 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 with the Pointer field referring to the first octet in the 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 referring to the first octet of the Length field in the Locator Preference option. In both cases of failures, no further processing is performed for the Locator Update message.

If the generation number matches, 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) in 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 lists the current Lp(peer) as BROKEN. The host uses the Probe message in [9] 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 which has a CT(local) that matches the context tag. If no such context is found, it sends a 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 which happens to match the CT(local) for this context. In this case the host MUST send a 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, then send R2 and proceed to process the message.

o If I2BIS-SENT, then 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 it 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 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 packets are sent with the original locators or not. The details of this is out of scope for this document and is specified in [9].

11.1. Sending ULP Payload after a Switch

When sending packets, if there is a ULID-pair context for the ULID pair, and 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 which are preserved end-to-end.

The sender skips any "routing sub-layer 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 extenson header. In case that the ULID pair is being used as 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 and this should be remembered in order to avoid tearing down the context and for reachability detection porpuses as described in [9]. The host continues with the normal processing of the IP packet.

12.2. Receiving Payload Extension Headers

The receiver extracts the context tag from the payload extension header, and uses this to find a ULID-pair context. If no context is found, the receiver SHOULD generate a 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, then send I2 and proceed to process the message.
- o If I2BIS-SENT, then 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 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 does 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 the neither the IPv6 destination field nor the IPv6 source field is a multicast 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 which 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 context identified that is conceptually identified by 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 ULID pair nor the CT(local)). If no context exist 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 ULID pair nor the CT(local)). If no context exists with this locator pair and INIT nonce, then silently discard.

- o R1bis packet: 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 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 a R1bis message.

- o Shim6 Error Messages and ICMP errors which 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 the shim ends up being used or not (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 [13], and some fixes to that specification [14] 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, a naive implementation at the socket API which uses getaddrinfo() to retrieve all destination addresses and then tries to bind() and connect() to try all source and destination address combinations waiting 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. But 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

The retransmit timers (I1_TIMEOUT, I2_TIMEOUT, UPDATE_TIMEOUT) are subject to binary exponential backoff, as well as 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 [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 reccomended that the Shim6 follows the reccomendation defined in [28] and it informs the upper layers about the path change, in order to allow the congestion control mechanisms of the upper layers can 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 by the Shim6 layer to them, rather 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 presudoheader 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 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 header after that. 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 pseudoheader computation, unless they implement 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 feature, since if the hosts managed to establish a ULID-pair context, then 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 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 they pass through.

15.3. Other considerations

The general Shim6 approach, as well as the specifics of this proposed solution, has 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 [23]. 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 applications perspective becomes a set of IP addresses instead of a single IP address.

o Signaling protocols for QoS or other things that involve having devices in the network path look at IP addresses and port numbers, or 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.

- MTU implications. The path MTU mechanisms we use are robust against different packets taking different paths through the Internet, by computing a minimum over the recently observed path MTUs. When Shim6 fails over from using one locator pair to another pair, this means that packets might travel over a different path through the Internet, hence the path MTU might be quite different. Perhaps such a path change would be a good hint to the path MTU mechanism to try a larger MTU?

  The fact that the shim will add an 8 octet 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.

16. Security Considerations

This document satisfies the concerns specified in [20] as follows:

- o The HBA [6] and CGA technique [8] for verifying the locators to prevent an attacker from redirecting the packet stream to somewhere else. The minimum acceptable key length for public keys used in the generation of CGAs SHOULD be 1024 bits. Any implementation should follow prudent cryptographic practice in determining the appropriate key lengths.

- o Requiring a Reachability Probe+Reply 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 of memory on the responder) at least requires the attacker to create state, consuming his own resources and also it 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, 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. Such discovery probably requires to be along the path in order to be sniff the context tag value. The result is that through this technique, the Shim6 protocol is protected against off-path attackers.

Interaction with IPSec

The Shim6 sub-layer is implemented below the IPSec layer within the IP layer. This deserves some additional considerations for a couple of specific cases: First, it should be noted that the Shim6 approach does not preclude using IPSEC tunnels on Shim6 packets within the network transit path. Second, in case that IPSec is implemented as Bump-In-The-Wire (BITW) [7], either the shim MUST be disabled, or the shim MUST also be implemented as Bump-In-The-Wire, in order to satisfy the requirement that IPsec is layered above the shim.

Some of the residual threats in this proposal are:

- An attacker which arrives late on the path (after the context has been established) can use the R1bis message to cause one peer to recreate the context, and at that point in time the attacker 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 it can use those to send bogus messages.

- An attacker which is present on the path so that it can find out the context tags, can generate a R1bis message after it has moved off the path. For this packet to be effective it needs to have a source locator which belongs to the context, thus there can not be "too much" ingress filtering between the attackers 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.

- 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 attack would be similar of those of an attacker sending packets with spoofed source address. In the case of control packets, it is not enough to find the correct context tag, but 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 payload extension header.

- When the 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, this could potentially allow 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 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.

- The validator included in the R1 and R1bis packets are 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 sending a I1 packet than performing all the computations required to determine the secret S. Nevertheless, it is recommended that the host changes the secret S periodically.

17. IANA Considerations

IANA is directed to allocate a new IP Protocol Number value for the Shim6 Protocol.

IANA is directed to record a CGA message type for the Shim6 Protocol in the [CGA] namespace registry with the value 0x4A30 5662 4858 574B 3655 416F 506A 6D48.

IANA is directed to establish a Shim6 Parameter Registry with three components: Shim6 Type registrations, Shim6 Options registrations Shim6 Error Code 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	Can be allocated using Standards Action
60-63	For Experimental use
64	Update Request
65	Update Acknowledgement
66	Keepalive
67	Probe Message
68-123	Can be 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
120-127	Reserved for debugging pruposes

18. Acknowledgements

Over the years many people active in the multi6 and shim6 WGs have contributed ideas a suggestions that are reflected in this specification. Special thanks to the careful comments from 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 and Tom Henderson on earlier versions of this document.

Appendix A. Possible Protocol Extensions

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

- As stated in the assumptions in Section 3, the 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 singly-homed, and 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 [21].

- Is there need for keeping the list of locators private between the two communicating endpoints? We can potentially accomplish that when using CGA but not with HBA, but it comes 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.

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

- Supporting the dynamic setting of locator preferences on a site-wide basis, and use 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.

- Potentially recommend that more application protocols use DNS SRV records to allow a site some influence on load spreading for the initial contact (before the Shim6 context establishment) as well as for traffic which does not use the shim.

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

- 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) but the not-so-busy client would retain the context state. The context recovery mechanism presented in Section 7.5 would then be recreate 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 an 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, 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 currently 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 e.g., one CGA public key, and would 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 locators per host, then we don't need this added complexity.

o ULP specified timers for the reachability detection mechanism (which can be useful particularly 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. There existing an initial draft on this topic [22].

Appendix B. Simplified State Machine

The states are defined in Section 6.2. The intent is that the stylized description below be consistent with the textual description in the specification, but 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 a new context establishment	Send I1 and move to I1-SENT
Receive I2, verify validator and RESP nonce	If successful, send R2 and move to ESTABLISHED If fail, stay in IDLE
Receive I2bis, verify validator and RESP nonce	If successful, send R2 and move to ESTABLISHED If fail, stay in IDLE
R1, R1bis, R2	N/A (This context lacks the required info for the dispatcher to deliver them)
Receive payload extension header or other control packet	Send R1bis and stay in IDLE

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

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

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

Trigger	Action
Receive R2, verify INIT nonce	If successful move to ESTABLISHED If fail, stay in I2-SENT
Receive I1	Send R2 and stay in I2-SENT
Receive I2 verify validator and RESP nonce	Send R2 and stay in I2-SENT
Receive I2bis verify validator and RESP nonce	Send R2 and stay in I2-SENT
Receive R1	Discard and stay in I2-SENT
Timeout, increment timeout counter	If counter =< I2_RETRIES_MAX, send I2 and 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 or extension header other control packet	Accept and send I2 (probably R2 was sent by peer and lost)

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

Trigger	Action
Receive R2, verify INIT nonce	If successful move to ESTABLISHED If fail, stay in I2BIS-SENT
Receive I1	Send R2 and stay in I2BIS-SENT
Receive I2 verify validator and RESP nonce	Send R2 and stay in I2BIS-SENT
Receive I2bis verify validator and RESP nonce	Send R2 and stay in I2BIS-SENT
Receive R1	Discard and stay in I2BIS-SENT
Timeout, increment timeout counter	If counter =< I2_RETRIES_MAX, send I2bis 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 or extension header other control packet	Accept and send I2bis (probably R2 was sent by peer and lost)

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

Trigger	Action
Receive I1, compare CT(peer) with received CT	If no match, send R1 and stay in ESTABLISHED If match, send R2 and stay in ESTABLISHED
Receive I2, verify validator and RESP nonce	If successful, then send R2 and stay in ESTABLISHED Otherwise, discard and stay in ESTABLISHED
Receive I2bis, verify validator and RESP nonce	If successful, then send R2 and stay in ESTABLISHED 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 or extension header other control packet	Process and stay in ESTABLISHED
Implementation specific heuristic (E.g., No open ULP sockets and idle for some time)	Discard state and go to IDLE

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

Trigger	Action
Wait for NO_R1_HOLDDOWN_TIME	Go to IDLE
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 ICMP_HOLDDOWN_TIME	Go to IDLE
Any packet	Process as in IDLE

Appendix B.1. Simplified State Machine diagram

```
                                          Timeout/Null    +------------+
                            I1/R1      +------------------| NO SUPPORT |
            Payload or Control/R1bis   |                  +------------+
                        +---------+    |                              ^
                        |         |    |               ICMP Error/Null|
                        |         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 host 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 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, and we have several 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 to 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 other host C, for instance for the ULID pair <C3, B2>. That is a form of three party context confusion.

Appendix 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.

Appendix C.2. Context Confusion

This cases 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), 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

recreates 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.

Appendix 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 payload extension headers with X as the context tag, it will find the context for <C3, B2>, hence 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 ok (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 sent from A to B using the old context on host A might persist for some time, but it 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.

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 e.g. 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 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.

Appendix 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 where 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 which 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 shim-aware ULPs to request context forking so that different ULP traffic can use different locator pairs.

Appendix 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 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, need to be taken into account for this operation.

For example, if a host has address 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 e.g., <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 the usage of a new extension header to carry the context tag. In this appendix we will describe the pros and cons of each approach and justify the selected option.

Appendix 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 approach that does 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 HostA and HostB.

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 and the locator set associated to IPB2 is just IPB2.

Because the locator sets of the 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 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 simply not to 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 this mean, each communication that is using different ULIDs can be differentiated because it has a different Flow Label value.

The problem with such approach is that it requires that the receiver of the communication allocates 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-shim 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 marking 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 the now expired NOID draft. However, even if this is done, this approach is incompatible with the deferred establishment capability of the shim protocol, which is a preferred function, since it suppresses the delay due to the shim context establishment prior to initiation of the communication and it also allows nodes to define at which stage

of the communication they decide, based on their own policies, that a given communication requires to be protected by the shim.

In order to cope with the identified limitations, an alternative approach that does not constraints the flow label values used by communications that are 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 approach communications start with unmodified flow label usage (could be zero, or as suggested in [17]). 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, and 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 that 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 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 suppose to exist, as it will be discussed in the Context Loss detection appendix below.

Appendix 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 8 octets shorter. However, path changes through the network can result in different MTU in any case, thus having a locator change, which implies a path change, affect the MTU doesn't introduce any new issues.

Appendix 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 IPA1,...,IPAn and IPB1,...,IPBm as locator set 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 HostB or that an error occurred in the shim layer at host A resulting in the loss of the context state.

The mechanisms considered in this appendix are aimed to deal 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/ reestablished.

Mechanisms for detecting context loss.

These mechanisms basically consist in that each end of the context 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 case that the context does not exist, it sends a packet notifying the problem to the

sender.

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.

An 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, so that when a node receives one of such packets, 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 they do not necessarily carry context specific information. In particular, packets flowing before a locator change (i.e. 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 the Extension Header or the Flow Label with new Next Header values approaches) 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 form 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 uses shim control packets and 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, 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 that 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 context loss event to the other end 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 in). One of the goals of this notification is to allow the other end that still retains context state, to reestablish the lost context. The mechanism to reestablish the loss 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 reestablishing 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 by 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.

Appendix D.4. Securing locator sets

The adoption of a protocol like SHIM that allows the binding of a given ULID with a set of locators opens the doors for different types of redirection attacks as described in [20]. The goal in terms of security for the design of the shim protocol is not to introduce any new vulnerability in the Internet architecture. It is a non-goal to provide additional protection than the 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 was to use cookies i.e. a randomly generated bit string that is negotiated during the context establishment phase and then it is included in following signaling messages. By this mean, 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 mechanisms does indeed provide a fair amount of protection, it does leave the door open for the so-called time shifted attacks. In these attacks, an attacker that once was on the path, it 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 in the locator set and he 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 if provided, an attacker that once was 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 the security was 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 sniffing packets during the initial handshake, since it is the only moment where the shared secret is exchanged. While this improves the security, it is still vulnerable to time shifted attacks, even though it imposes that the attacker must be on path at a very specific moment (the establishment phase) to actually be able to launch the attack. While this seems to substantially improve the situation, it should be noted that, depending on protocol details, an attacker may be able to force the recreation of the initial handshake (for instance by blocking messages and making the parties think that the context has been lost), so the resulting situation may not differ that much from the cookie based approach.

Another option that was discussed during the design of the 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 used with the cryptographic keys is the usage of digital certificates. This implies that an IPsec based solution would require that the generation of digital certificates that bind the key and the ULID by a common third trusted party for both parties involved in the communication. Considering that the scope of application of the shim protocol is global, this would imply a global public key infrastructure. The major issues with this approach are the deployment difficulties associated with a global PKI.

Finally two different technologies were selected to protect the shim protocol: HBA [8] and CGA [6]. These two approaches provide a similar level of protection but they provide different functionality with a different computational cost.

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.

Any of these two mechanisms HBA and CGA provide 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, so that 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.

Appendix 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 that 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 victim's resources, similar to TCP SYN flooding attacks.

A 4 way-handshake exchange protects against these attacks because the receiver does not creates any state associate to a given context until the reception of the second packet which contains a 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 at some extent valid, since the initiator is able to receive packets at this address. In the worse 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 deferred context establishment capability of the shim protocol, this limitation has a reduced impact in the performance of the protocol. (It may however have a greater impact in the situation of context recover as discussed earlier, but 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 in a 2-packet exchange and it does not verify that the initiator has performed a prior contact before creating context state. In case that 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 to proceed. 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 2-way exchange, it is not possible to use the number of half open sessions (as in TCP) to detect an ongoing attack and 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.

Appendix 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 send the complete locators set each time that a variation in the locator set occurs. The differential approach send the differences between the existing locator set and the new one. The atomic approach imposes additional overhead, since all the locator set has to be exchanged each time while the differential approach requires re-synchronization of both ends through changes i.e. 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.

Appendix 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 the 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 about the situation and possibly a recovery mechanism is also needed.

A coordinated approach would use an explicit CLOSE mechanism, akin to the one specified in HIP [26]. 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 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 CLOSE timer, or that it is ok 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, other aspect that is relevant for this design choice is the context confusion issue. In particular, using an unilateral approach to discard context state clearly opens the possibility of context confusion, where one of the ends unilaterally discards the context state, while the peer 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 a another context, creating a potential context confusion situation. In order to illustrate the cases where problems would arise consider the following scenario:

- Hosts A and B establish context 1 using CTA and CTB as context tags.
- Later on, A discards context 1 and the context tag value CTA becomes available for reuse.
- However, B still keeps context 1.

This would become a context confusion situation in the following two cases:

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

One could think that using a coordinated approach would eliminate these context confusion situations, 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 a context confusion situation because 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 this situations.

Appendix E. Change Log

[RFC Editor: please remove this section]

The following changes have been made since draft-ietf-shim6-proto-07:

- New Error Message format added in the Format section
- Added new registry for Error codes in the IANA considerations section
- Changed the Format section so a Shim6 error message is sent back when a crtical option is not recognized (instead of an ICMP error message)
- Changed the ULID estbalishment section so that a Shim6 error message is sent back when the locator verification is not recgnized or not consistent with the current CGA PDS
- Changed the Locator Update section so that Shim6 error messages are sent instead of ICMP error messages
- Changed the receiving packet section so that Shim6 error messages are generated instead of ICMP error messages
- added new section about middle box consideration in the implication elsewhere section
- added text for allowing strcuture in context tag name space, while still randomly cycling though part of the tag name space
- changed the name of TEMPORARY flag for the TRANSIENT flag
- clarified option length calculation
- Editorial commnets from Iljitsch review
- added new sub-section in the introduction about congestion notification to upper layer and include a reference to I-D.schuetz-tcpm-tcp-rlci
- added reccomendation to keep the shim6 message length below 1280 bytes
- added the init nonce in the description of the verification of the validator when receiving I2 messages

- removed FII and ULID in the verification of the validator when receiving I2BIS meesages, and added receiver context tag.

- Clarified section about retransmision of I2 and I2bis messages, in case that the initiator decides not to retransmit I2/I2bis messages and retransmits I1 message

- Clarified the effect of packets associated with a context but without the shim6 header when considering tearing down a context

- Added new section in section 12 about how to process packets associated with a context that do not carry the shim6 ext header

- Added respon der validator as information stored in I2-SENT and Responder validator, init nonce and RESP nonce as information available in I2BISSENT

- Added Init Nonce, Responder Nonce, and Responder validator as information available for a shim6 context in the conceptual model during establishment phase.

- Clarified how the Responder Validator is calculated based on a running counter that is independent of any received message

- Editorial corrections resulting from Dave Thaler and Bob Braden reviews.

The following changes have been made since draft-ietf-shim6-proto-06:

- Changed wording in the renumberin considerations section, so that a shim6 context using a ULID that has been renumbered, MUST be discarded

- Included text in the security considerations about IPSec BITW and IPSec tunnels.

- Added text about the minimum key length of CGA in the security considerations section

- fixed Payload/update message processing

- synchonized with READ draft

The following changes have been made since draft-ietf-shim6-proto-05:

- Removed the possibility to keep on using the ULID after a renumbering event

o Editorial corrections resulting from Dave Meyer's and Jim Bound's reviews.

The following changes have been made since draft-ietf-shim6-proto-04:

o Defined I1_RETRIES_MAX as 4.

o Added text in section 7.9 clarifying the no per context state is stored at the receiver in order to reply an I1 message.

o Added text in section 5 and in section 5.14 in particular, on defining additional options (including critical and non critical options).

o Added text in the security considerations about threats related to secret S for generating the validators and recommendation to change S periodically.

o Added text in the security considerations about the effects of attacks based on guessing the context tag being similar to spoofing source addresses in the case of payload packets.

o Added clarification on what a recent nonce is in I2 and I2bis.

o Removed (empty) open issues section.

o Editorial corrections.

The following changes have been made since draft-ietf-shim6-proto-03:

o Editorial clarifications based on comments from Geoff, Shinta, Jari.

o Added "no IPv6 NATs as an explicit assumption.

o Moving some things out of the Introduction and Overview sections to remove all SHOULDs and MUSTs from there.

o Added requirement that any Locator Preference options which use an element length greater than 3 octets have the already defined first 3 octets of flags, priority and weight.

o Fixed security hole where a single message (I1) could cause CT(peer) to be updated. Now a three-way handshake is required before CT(peer) is updated for an existing context.

The following changes have been made since draft-ietf-shim6-proto-02:

- Replaced the Context Error message with the R1bis message.

- Removed the Packet In Error option, since it was only used in the Context Error message.

- Introduced a I2bis message which is sent in response to an I1bis message, since the responders processing is quite in this case than in the regular R1 case.

- Moved the packet formats for the Keepalive and Probe message types and Event option to [9]. Only the message type values and option type value are specified for those in this document.

- Removed the unused message types.

- Added a state machine description as an appendix.

- Filled in all the TBDs - except the IANA assignment of the protocol number.

- Specified how context recovery and forked contexts work together. This required the introduction of a Forked Instance option to be able to tell which of possibly forked instances is being recovered.

- Renamed the "host-pair context" to be "ULID-pair context".

- Picked some initial retransmit timers for I1 and I2; 4 seconds.

- Added timer values as protocol constants. The retransmit timers use binary exponential backoff and randomization (between .5 and 1.5 of the nominal value).

- Require that the R1/R1bis verifiers be usable for some minimum time so that the initiator knows for how long time it can safely retransmit I2 before it needs to go back to sending I1 again. Picked 30 seconds.

- Split the message type codes into 0-63, which will not generate R1bis messages, and 64-127 which will generate R1bis messages. This allows extensibility of the protocol with new message types while being able to control when R1bis is generated.

- Expanded the context tag from 32 to 47 bits.

- Specified that enough locators need to be included in I2 and R2 messages. Specified that the HBA/CGA verification must be performed when the locator set is received.

- Specified that ICMP parameter problem errors are sent in certain error cases, for instance when the verification method is unknown to the receiver, or there is an unknown message type or option type.

- Renamed "payload message" to be "payload extension header".

- Many editorial clarifications suggested by Geoff Huston.

- Modified the dispatching of payload extension header to only compare CT(local) i.e., not compare the source and destination IPv6 address fields.

The following changes have been made since draft-ietf-shim6-proto-00:

- Removed the use of the flow label and the overloading of the IP protocol numbers. Instead, when the locator pair is not the ULID pair, the ULP payloads will be carried with an 8 octet extension header. The belief is that it is possible to remove these extra bytes by defining future Shim6 extensions that exchange more information between the hosts, without having to overload the flow label or the IP protocol numbers.

- Grew the context tag from 20 bits to 32 bits, with the possibility to grow it to 47 bits. This implies changes to the message formats.

- Almost by accident, the new Shim6 message format is very close to the HIP message format.

- Adopted the HIP format for the options, since this makes it easier to describe variable length options. The original, ND-style, option format requires internal padding in the options to make them 8 octet length in total, while the HIP format handles that using the option length field.

- Removed some of the control messages, and renamed the other ones.

- Added a "generation" number to the Locator List option, so that the peers can ensure that the preferences refer to the right "version" of the Locator List.

- In order for FBD and exploration to work when there the use of the context is forked, that is different ULP messages are sent over different locator pairs, things are a lot easier if there is only one current locator pair used for each context. Thus the forking of the context is now causing a new context to be established for the same ULID; the new context having a new context tag. The

original context is referred to as the "default" context for the ULID pair.

- Added more background material and textual descriptions.

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] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[3] Narten, T., Nordmark, E., and W. Simpson, "Neighbor Discovery for IP Version 6 (IPv6)", RFC 2461, December 1998.

[4] Thomson, S. and T. Narten, "IPv6 Stateless Address Autoconfiguration", RFC 2462, December 1998.

[5] Conta, A. and S. Deering, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", RFC 2463, December 1998.

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

[7] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.

[8] Bagnulo, M., "Hash Based Addresses (HBA)", draft-ietf-shim6-hba-02 (work in progress), October 2006.

[9] Arkko, J. and I. Beijnum, "Failure Detection and Locator Pair Exploration Protocol for IPv6 Multihoming", draft-ietf-shim6-failure-detection-07 (work in progress), December 2006.

19.2. Informative References

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

[11] 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.

[12] Narten, T. and R. Draves, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", RFC 3041, January 2001.

[13] Draves, R., "Default Address Selection for Internet Protocol

version 6 (IPv6)", RFC 3484, February 2003.

[14] Bagnulo, M., "Updating RFC 3484 for multihoming support", draft-bagnulo-ipv6-rfc3484-update-00 (work in progress), December 2005.

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

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

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

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

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

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

[21] Huitema, C., "Ingress filtering compatibility for IPv6 multihomed sites", draft-huitema-shim6-ingress-filtering-00 (work in progress), September 2005.

[22] Bagnulo, M. and E. Nordmark, "SHIM - MIPv6 Interaction", draft-bagnulo-shim6-mip-00 (work in progress), July 2005.

[23] Nordmark, E., "Shim6 Application Referral Issues", draft-ietf-shim6-app-refer-00 (work in progress), July 2005.

[24] Bagnulo, M. and J. Abley, "Applicability Statement for the Level 3 Multihoming Shim Protocol (Shim6)", draft-ietf-shim6-applicability-02 (work in progress), October 2006.

[25] Huston, G., "Architectural Commentary on Site Multi-homing using a Level 3 Shim", draft-ietf-shim6-arch-00 (work in progress), July 2005.

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

[27] Eronen, P., "IKEv2 Mobility and Multihoming Protocol (MOBIKE)",

draft-ietf-mobike-protocol-08 (work in progress), February 2006.

[28] Schuetz, S., "TCP Response to Lower-Layer Connectivity-Change Indications", draft-schuetz-tcpm-tcp-rlci-01 (work in progress), March 2007.

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

Full Copyright Statement

Copyright (C) The IETF Trust (2007).

This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY, THE IETF TRUST AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).