Erik Nordmark
Sun Microsystems

Multihoming without IP Identifiers

<draft-nordmark-multi6-noid-02.txt>

Status of this Memo

Abstract

This document outlines a potential solution to IPv6 multihoming in order to stimulate discussion.

This proposed solution relies on verification using the existing DNS to prevent redirection attacks, while allowing locator rewriting by (border) routers, with no per-packet overhead. The solution does not introduce a "stack name" type of identifier, instead it ensures that all upper layer protocols can operate unmodified in a multihomed setting while still seeing a stable IPv6 address.

Contents

1. INTRODUCTION

The goal of the IPv6 multihoming work is to allow a site to take advantage of multiple attachments to the global Internet without having a specific entry for the site visible in the global routing table. Specifically, a solution should allow users to use multiple attachments in parallel, or to switch between these attachment points dynamically in the case of failures, without an impact on the upper layer protocols.

This proposed solution uses existing DNS mechanisms to perform enough validation to prevent redirection attacks.

The goals for this proposed solution is to:

- Have no impact on upper layer protocols in general and on transport protocols in particular.
- Address the security threats in [M6THREATS].
- Allow routers rewriting the (source) locators as a means of quickly detecting which locator is likely to work for return traffic.
- No per-packet overhead.
- No extra roundtrip for setup.
- Take advantage of multiple locators/addresses for load spreading.

1.1. 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 locator 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. This proposal does not attempt to solve, perhaps related, problems such as host multihoming or host mobility.

This proposal also does not try to provide an IP identifier. Even though such a concept would be useful to ULPs and applications, especially if the management burden for such a name space was zero 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) when using the DNS to verify the locator relationships.

1.2. Assumptions

The main technical assumptions this proposal makes is that the DNS infrastructure can be used for verification of the relationship between locators on both the initiator of communication and the responding peer. In particular, it assumes that getting DNS reverse tree (ip6.arpa) populated for the hosts that wish to take advantage of multihoming, will not be a significant problem.

This version of the document further relaxes this constraint so that if two communication hosts are in separate multihomed sites, if only one of them has correct information in the forward plus reverse DNS, the communication between them can benefit from the multiple locators of that host. However, without loss of security it isn't possible to benefit from the multiple locators of a host with no DNS information (or incorrect/missing forward and reverse DNS information).

2. TERMINOLOGY

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.

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 identifier for an IP layer endpoint (stack name in [NSRG]). The transport endpoint is a function of the transport protocol and would typically include the IP identifier plus a port number. NOTE: This proposal does not contain any IP layer identifiers.

Application identifier (AID)
- an IP locator 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 AIDs in order to enable load spreading.

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.

FQDN - Fully Qualified Domain Name

2.1. Notational Conventions

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

FQDN(A) is the domain name for A.

Lsl(A) is the "local" locator set for A, that is, the set of locators that A itself knows it has. This set expresses which locators should be acceptable to receive packets sent by A.

Lsr(A) is the "remote" locator set for A, that is, the set of locators of A that its peer found in the DNS. Normally Lsl(A) and Lsr(A) are the same, but due to DNS updates, DNS load balancing, or misconfiguration they might differ.

Ls(A) is the locator set for A, which consists of L1(A), L2(A), ... Ln(A). For robustness Ls(A) is computed as the intersection of Lsl(A) and Lsr(A) by both A and its peer.

Lsv(A) is the verified locator set for A. This is the subset of Ls(A) which has been verified with both forward and reverse DNS lookups to be a set corresponding to a single FQDN. This set expresses which locators are safe to use as destinations when sending packets to A.

AID(A) is an application ID for A. In this proposal, AID(A) is always one member of A's locator set.

3. PROTOCOL OVERVIEW

In order to prevent redirection attacks this protocol relies on the DNS (for the hosts which want to take advantage of themselves having multiple locators) being maintained with consistent forward and reverse information. This allows any host, given one locator, to determine the corresponding FQDN and the set of locators for the host. Once those lookups have been performed, and the original locator is indeed part of the set, the host can happily allow any of those locators to be used without being subject to redirection attacks. Keeping the FQDN around allows the solution to handle graceful renumbering by being able to redo the DNS lookups (e.g., based on the TTL on the resource records).

DNS is also used to provide an indication of multihoming capability of a host. The details of this is TBD but a simple example would be to introduce a new M6 Resource Record type in the DNS which has no RDATA; thus the mere existence of such a record at a FQDN would imply that the host supports the M6 protocol. See Section 15.1 for an alternative approach that doesn't need a new RR type.

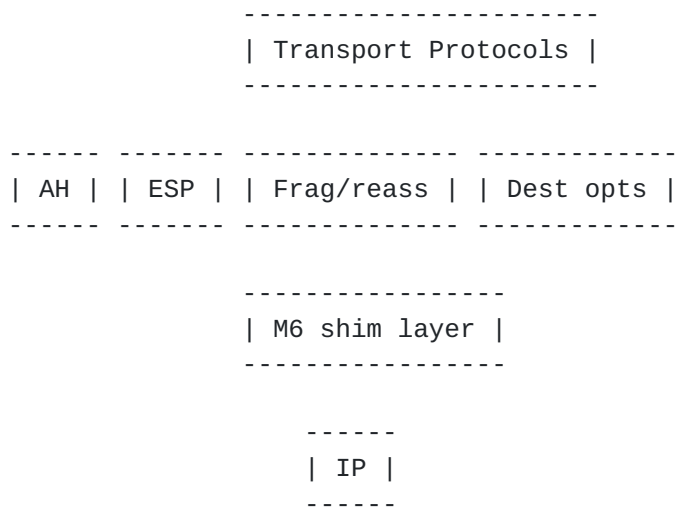


Figure 1: Protocol stack

The proposal uses an M6 shim layer between IP and the ULPs as shown in figure 1, in order to provide ULP independence. Conceptually the M6 shim layer behaves as if it is associated with an extension header, which would be ordered immediately after any hop-by-hop options in the packet. However, the amount of data that needs to be carried in an actual M6 extension header is close to zero. By using some encoding of the nexthdr value it is possible to carry the common protocols/extension headers without making the packets larger. The nexthdr encodings are discussed later in this document. We refer to

packets that use this encoding to indicate to the receiver that M6 processing should be applied as "M6 packets" (analogous to "ESP packets" or "TCP packets").

Layering AH and ESP above the M6 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. Layering the fragmentation header above the M6 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 M6 shim layer is placed between the IP endpoint sublayer, which handles fragmentation, reassembly, and IPsec, and the IP routing sublayer, which on a host selects which default router to use etc.

The proposal uses router rewriting of (source) locators as one way to determine which is the preferred (or only working) locator to use for return traffic. But not all packets can have their locators rewritten. In addition to existing IPv6 packets, the packets exchanged before M6 host-pair context state is established at the receiver can not have their locators rewritten. Thus a simple mechanism is needed to indicate to the routers on the path whether or not it is ok to rewrite the locators in the packet. Conceptually this is a single bit in the IPv6 header (we call it the "rewrite ok" bit) but there is no spare bit available. Later in the document we show how we solve this by allocating a range of next header values to denote this semantic bit.

Applications and upper layer protocols use AIDs which the M6 layer will map to/from different locators. The M6 layer maintains state, called host-pair context, in order to perform this mapping. The mapping is performed consistently at the sender and the receiver, thus from the perspective of the upper layer protocols, packets appear to be sent using AIDs from end to end, even though the packets travel through the network containing locators in the IP address fields, and even though those locators might be rewritten in flight.

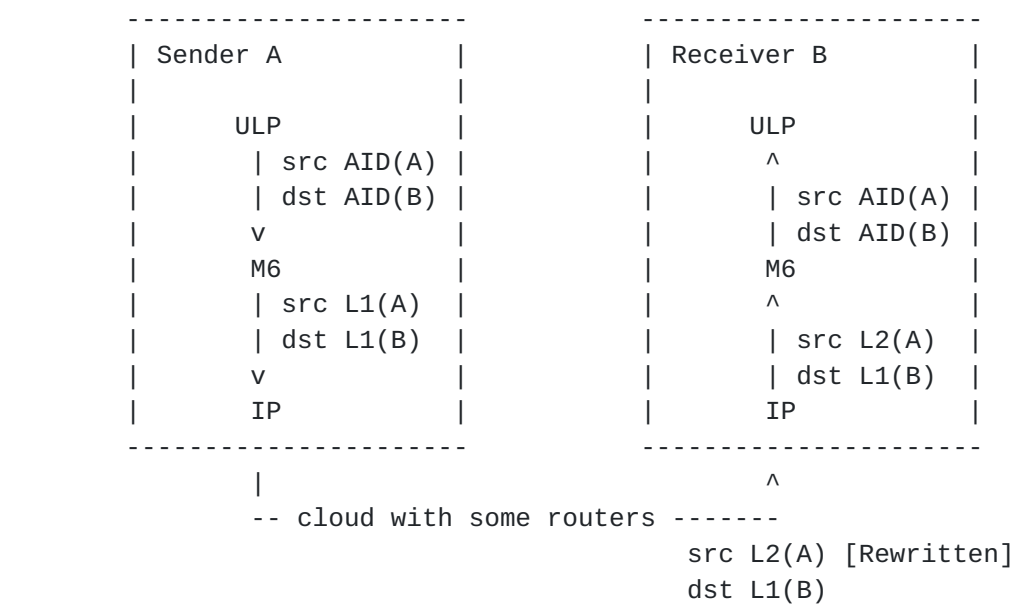


Figure 2: Mapping with router rewriting of locators.

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 AIDs and locators are being present in every packet, but with a header compression mechanism applied that removes the need for the AIDs once the state has been established. As we will see below the flow label will be used akin to a "compression tag" i.e., to indicate the correct context to use for decompression.

The need for some "compression tag" is because the desire to allow load spreading and handle site renumbering. Without those desires it could have been possible to e.g. designate one fixed locator as the AID for a host and storing that in the DNS. But instead different connections between two hosts are allowed to use different AIDs and on reception of a M6 packet the correct AIDs must be inserted into the IP address fields before passing the packet to the ULP. The flow label serves as a convenient "compression tag" without increasing the packet size, and this usage doesn't conflict with other flow label usage.

In addition to the zero overhead data messages, there are seven different M6 message types introduced (which are defined as new ICMPv6 messages). Four types are used to perform a 4-way handshake to create state at both endpoints without creating state on the first received packet (which would introduce a memory consumption DoS attack), and two types are used to update the set of locators used for the context, and finally a single message type to signal that

state has been lost. The seven message types are called:

- Context Initiator message (INIT); first message of the 4-way context establishment. Sent by the initiator when there is a desire to create a host-pair context for future locator agility. Normally sent when a data packet is passed down to IP from the ULP when there is no context state. An ULP packet can be piggybacked on this message.

- Context Check message (CC); second message of the 4-way context establishment. Sent in response to a INIT message. An ULP packet can be piggybacked on this message.

- Context Check Reply message (CCR); third message of the 4-way context establishment. Sent in response to a CC message. An ULP packet can be piggybacked on this message.

- Context Confirm message (CONF); Last message in the context establishment exchange; can be sent in response to a CCR or an INIT message. Carries the responders flow label as well as any initial reductions in the locator set to the initiator.

- Update Request message (UR); sent to update the local and remote locator sets. An ULP packet can be piggybacked on this message.

- Update Acknowledgement message (UA); sent to acknowledge an Update Request message. An ULP packet can be piggybacked on this message.

- Unknown Context message (UC); error which is sent when no state is found.

Similar to MAST [MAST] the above exchange can be performed asynchronously with data packets flowing between the two hosts; until context state has been established at both ends the packets would flow without allowing router rewriting of locators and without the ability for the hosts to switch locators.

Once the 4-way state creation exchange has completed there is host-pair context state at both hosts, and both ends know a set of locators for the peer that are acceptable as the source in received packets. At that point in time the responder (which didn't use DNS before the setup) can asynchronously start verifying additional locators using the DNS. Once a peer locator has been verified it will be a candidate destination locator including the ability to dynamically switch to using the last received source locator (that is already verified) as the destination locator for return traffic.

3.1. DNS Usage and Dependencies

For the protocol to be beneficial at least one of the communicating peers need to have a FQDN with consistent information in the forward and reverse trees. The protocol uses the DNS at the end of the communication which normally does the DNS lookup, to not only find all the locators (as AAAA records) but also verify that these locators have reverse tree entries which point back at the FQDN. The locators which do not point back to the FQDN will not be used as part of the locator set. At the end of the communication which isn't required to do a DNS lookup today (the responder or server), this protocol, at some point after the context has been created message is received, will use the AID (normally the source address of the CCR packet) to first find the peer's FQDN, and them perform a forward lookup plus a reverse lookup of all the IPv6 locators, in order to verify that these locators are bound to the same FQDN. As in the initiator case, the locators which do not verify using this method will be excluded from the locator set. If no locators verify, then only the AID will be viewed as part of the locator set i.e., the protocol falls back to not providing any address agility.

For the protocol to work well, both ends have to agree on each other's locator sets. There are several reasons why a host's notion of its locators (or IPv4 addresses today) might be different than the set of locators present in the DNS for the host's FQDN (misconfiguration, DNS load balancing, in-progress DNS update, etc.) In order to cope with this, the two communicating hosts exchange each other's notion of each other's locators and form the intersection about what they know (about themselves as well as the peer) and what the peer knows (about themselves as well as the peer). A property of this approach is that the locator set can never be larger than what is contained in the DNS, that is, a host can not use this to fool its peer to include additional locators as destinations, for instance for 3rd party DoS attacks [M6THREATS]. This intersection should never be empty; an empty intersection is an indication that only the AID should be used.

The hosts' locators might change over time due to renumbering. At some point in time such a change will need to be reflected in the DNS in order for this protocol to be able to take advantage of any added locators. The protocol uses the Update Request from the peer in combination with the DNS TTL as an indication when it makes sense to redo the DNS lookup of the peer's FQDN to detect changes in the locator set.

3.2. Host-Pair Context

The host-pair context is established on the initiator of communication based on information learned from the DNS (either by starting with a FQDN or with an IP address -> FQDN lookup). The responder will establish some initial state using the context creation 4-way handshake and later verify the peer's locators using the DNS.

The context state contains the following information:

- the peer locator which the ULP uses as ID; AID(peer)
- the local locator which the ULP uses as ID; AID(local)
- the set of peer locators determined from the DNS; Lsr(peer)
- the set of peer locators communicated from the peer; Lsl(peer)
- the set of peer locators; Ls(peer); intersection of the two above
- for each peer locator, a bit whether it has been verified with the DNS (by doing reverse + forward lookup). This can be viewed as forming a subset of Ls(peer) which we call Lsv(peer).
- the preferred peer locator - used as destination; Lp(peer)
- the set of local locators from local information; Lsl(local)
- the set of local locators communicated from the peer, that is, which the peer found in the DNS; Lsr(local)
- the set of local locators; Ls(local); intersection of the two above
- the preferred local locator - used as source; Lp(local)
- the flow label allocated by the peer that is used to transmit packets; F(peer)
- the flow label allocated by the host to expect in received packets; F(local)
- the fully qualified domain name for the peer; FQDN(peer)
- State about peer locators that are in the process of being verified in the DNS

- Peer's birthday counter (a counter which increments by one each time the host reboots)

This state is accessed differently in the transmit and receive paths. In the transmit path when the ULP passes down a packet the key to the context state is the tuple <AID(local), AID(peer)>; this key must identify at most one state record. In the receive path it is the F(local) flow label, which was allocated by the host itself, to uniquely identify a host-pair context.

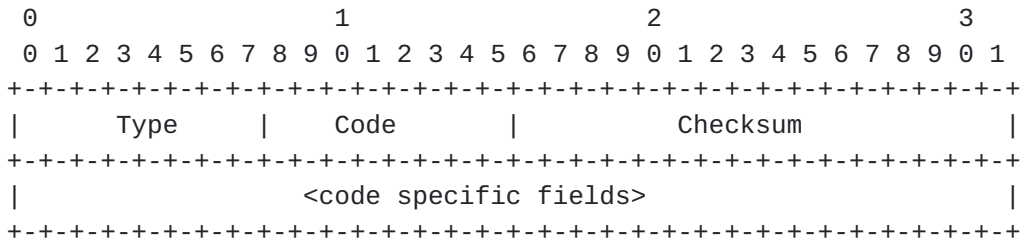
This limits a single host to maintaining about 1 Million host-pair contexts at a time; limited by the 20 bits of flow label. A host that needs to support more than this limit, can do so by acting as multiple hosts from the perspective of this protocol. This would entail having multiple FQDNs and each FQDN being associated with a different set of locators. For instance, these different locators might use different interface identifiers.

Note that the flow labels could be selected to be finer grain than above; for instance having a different flow label for each connection. Doing so requires some efficient data structure organization at the receiver to map multiple F(local) to the same context and it might make the host run out of flow labels more easily. However, the use of finer grain flow labels might be necessary when the flow labels are also used for QoS purposes in the routers. [RFC3697].

3.3. Message Formats

The set of messages and message sequences are similar to those in [HIP] and [WIMP] but the content is quite different, due to the different approach to security.

The base M6 header is an ICMPv6 header as follows:



ICMPv6 Fields:

Type

TBD [IANA]

Code
8-bit field. The type of M6 message. The M6 header carries seven different types of messages:

- o Context Initiator message (INIT)
- o Context Check message (CC)
- o Context Check Reply message (CCR)
- o Context Confirm message (CONF)
- o Update Request message (UR)
- o Update Acknowledgement message (UA)
- o Unknown Context message (UC)

Checksum The ICMPv6 checksum.

Future versions of this protocol may define new message codes. Receivers MUST silently ignore any message code they do not recognize.

This drafts doesn't contain actual message layout for the code specific part. However, the content of these messages is specified below.

3.3.1. Context Initiator (INIT) Message Format

The Context Initiator (INIT) message contains:

- Initiator Nonce (used to match the CC message with the INIT message)
- Initiator's AID
- Responder's AID
- Initiator's locally determined locators - Lsl(initiator)
- Responder's remotely determined locators - Lsr(responder)
- Initiator's flow label (20 bits)
- Initiator's birthday counter (a counter which increments by one

each time the host reboots)

3.3.2. Context Check (CC) Message Format

When the responder receives an INIT message it does not allocate any state to prevent a class of DoS attacks. Instead it sends a CC message, which effectively contains the state it would have allocated, and gets that information echoed back from the initiator in a CCR message.

This context state consists of:

- the two AIDs
- the initiator's flow label (The responder's flow label is allocated when the CCR message is received.)
- the initiator's locator set from the INIT message; Lsl(initiator)
- the responder's locator set from the INIT message; Lsr(responder)
- the responder's locator set as know to the responder itself; Lsl(responder). The encoding [TBD] for the responder's locator set can use two bits per locator to indicate whether a locator is in Lsr(responder), Lsl(responder), or both.
- The two birthday counters
- The "ULP packet discarded" bit indicating that the ULP packet piggybacked on an INIT message was not delivered to the ULP.

The Context Check (CC) message contains:

- Initiator's Nonce (copied from the INIT message)
- Context state as above
- A timestamp or nonce (for the benefit of the responder matching the CCR message with the CC, as well as finding the per-host key which was used with the hash below)
- A hash over the context state and timestamp/nonce (to prevent modification of the context state between sending it in the CC and receiving it back in the CCR message)

3.3.3. Context Check Reply (CCR) Message Format

The Context Check Reply (CCR) message contains:

- Initiator's nonce (to match the CONF message with the CCR)
- The context state, timestamp/nonce, and hash copied from the CC message.

3.3.4. Context Confirm (CONF) Message Format

In case the responder performs some verification of the peer's locator before it sends the CONF message, it can include the Lsr(initiator) in the CONF message. If this verification is deferred, then a Update Request message will need to be sent later if the DNS verification indicates that some of the peer's locators should not be used because they are not in the DNS.

The Context Confirm (CONF) message contains:

- Initiator's AID
- Responder's AID
- Initiator's Nonce (copied from the INIT or CCR message which triggered sending the CONF message)
- The responder's flow label
- Responder's birthday counter. (Needed when the CONF is in response to an INIT message).
- Optionally the initiator's remotely determined locators - Lsr(initiator)
- Optionally the responder's locally determined locators - Lsl(responder). (Needed when the CONF is in response to an INIT message).

3.3.5. Update Request (UR) Message Format

Either end of the communication can send an update request to inform the peer of a change in the locator sets. This change could be any combination of additions to the sender's local locators, deletions to

the sender's local locators, or changes to the peers locators that have been determined from the DNS. The Update request message contains:

- Request nonce/timestamp (used to match the UA with the UR)
- Sender's and receiver's AID
- Sender's and receiver's flow label for the context.
- Sender's locally determined locators - Lsl(sender)
- Receiver's remotely determined locators - Lsr(receiver)

3.3.6. Update Acknowledgement (UA) Message Format

The Update Acknowledgement message signals the receipt of the Update Request message and contains:

- Nonce/timestamp copied from the UR message
- Sender's and receiver's AID
- Sender's and receiver's flow label for the context.

If both ends need to update each other, each end has to send an Update Request message.

3.3.7. Unknown Context (UC) Message Format

The Unknown Context message contains:

- The 20-bit flow label from the triggering packet.
- The birthday counter (a counter which increments by one each time the host reboots)
- The triggering packet starting with the IPv6 header (as much of the packet as fits in the MTU)

4. PROTOCOL WALKTHROUGH

4.1. Initial Context Establishment

Here is the sequence of events when A starts talking to B:

1. A looks up FQDN(B) in the DNS which returns Lsr(B) plus "B is M6 capable". One locator is selected to be returned to the application: AID(B) = L1(B). The others are installed in the M6 layer on the host with AID(B) being the key to find that state.

   To make sure that the lookup from AID(B) returns a single state record it appears that one needs to do a reverse lookup of AID(B) to get the FQDN and check that the result is indeed FQDN(B). Whether this check can be deferred until two entities try to use the same AID(B) for a different Ls is for further study. Always doing the reverse lookup would be more predictable in any case. See section 16.2 for some more discussion.

2. The ULP creates "connection" state between AID(A)=L1(A) and AID(B) and sends the first packet down to the IP/M6 shim layer on A. L1(A) was picked using regular source address selection mechanisms.

3. The M6 layer matches on AID(B) and finds the proto-context state (setup in step #1) with Lsr(B). The existence of that state means that it is possible to establish a host-pair context. Host A can decide when to establish the context; it can defer this and use regular IPv6 with the locators being the AIDs for some time.

   In this example A decides to immediately establish the host-pair context. Thus it sends an INIT message with the ULP packet as part of the payload (if the MTU allows it to fit).

   Before sending the INIT message A selects a unique flow label F(local) for the new context and a nonce to match the CC with the INIT.

4. The packet (TCP SYN or whatever) is sent to the peer with locators L1(A) to L1(B) i.e., the same as the AIDs, piggybacked on the INIT message. It is possible to set the "rewrite ok" bit in the header for the INIT header, but if the locators are rewritten the piggybacked packet can not be passed to the ULP.

5. Host B receives the INIT message and passes the packet to the M6 shim layer. The M6 layer can't create state on the first packet, but if the packet was not rewritten in transit (that is, the AIDs are in the IP address fields) it can pass a piggybacked packet to the ULP. The ULP sees a packet identified by AID(A), AID(B). If the source address field is different than AID(A), then it is not safe to pass a piggybacked packet to the ULP since it could have been a case of spoofed AIDs. In this case the CC message would indicate that the ULP packet was dropped so that the initiator can retransmit the ULP packet once the context has been established.

   The CC message is sent to the source locator of the INIT, even if the peer AID is different.

   The same technique as in [MIPv6] is used to securely do the CC/CCR exchange without any local state; use a local per-host key which is never shared with anybody and pass the context state, a timestamp, and the keyed hash of the state+timestamp in the CC message. When the state, timestamp, and keyed hash value is returned in the CCR message, the hash is used to verify that the state hasn't been modified by the initiator.

   The 4-way exchange is done asynchronously with ULP packets, but it is possible (assuming the MTU allows) to piggyback ULP packets on the CC message.

   Should ULP packets be passed down to the M6 layer on B before the CCR message has been received, there will be no context state and no state installed as a result of a DNS lookup (unlike on A). This will indicate that the ULP message should be passed as-is (not as an M6 message) to the peer. Thus during the 4-way exchange packets can flow in both directions using the original locators=AIDs.

6. Host A receives the CC message. It verifies that the message is sent in response to the INIT by comparing the nonce.

   If a ULP packet was piggybacked A will pass that to the ULP. This can be done even if the locators where rewritten as long as the locators are part of the locator sets known to A.

   A forms the peer's locator set by taking the intersection of the Lsr it found in the DNS and the Lsl returned in the CC message. If the AID(peer) is not part of the Lsl(peer) it is an error and the AID is added to the locator set. [The fact that the peer forgot to include the AID could be an indication that it is seriously confused. The sender should ensure that the AID is

always included in the locator set when sending a locator set to a peer.]

Then A sends a CCR message which has the same information as the CC message. The message is sent to the source locator for the CC message.

7. Host B receives the CCR message. It verifies that the hash of the state is correct using its per-host key and verifies that the timestamp is recent. At this point in time it knows that A is at least not just blasting out INIT messages as a DoS - A is also responding to CC messages. Thus B goes ahead and allocates state at this point in time using the state that is in the CCR message, and allocates a unique F(local) for this context.

   At this point in time B has enough information to handle M6 packets from A, even though it hasn't yet verified the peer's locators in the DNS.

   If a ULP packet was piggybacked on the CCR message, B will pass that to the ULP. This is done even if the IP address fields do not contain the AIDs, since having created the context state when the CCR was received, any "response" packets from the ULP will only be sent out to the verified peer locators for the context. Hence there is no risk that this can be used for reflection attacks that bypass any deployed ingress filtering.

   In response to the CCR message, B sends a CONF message which includes the allocated F(B). Also, if B has performed some DNS verification of the initiator's locators prior to sending the CONF message, and not all locators in Lsl(A) verified, then B includes the Lsr(A) in the CONF message. If B later discovers that not all locators in Lsl(A) verified it will need to send an Update Request message.

   At this point in time B can start asynchronously and incrementally extracting and verifying Lsr(A) from the DNS. The first lookup consists of finding L1(A)=AID(A) in ip6.arpa to get the FQDN and record it, and lookup the AAAA RR set for that FQDN to get Lsr(A). Based on Lsr(A) and the Lsl(A) received in the CCR, B can form the intersection Ls(A). If this intersection does not contain AID(A) it is an error and the AID is added to the locator set.

   Once Ls(A) is known, B can verify (also incrementally) that each member of Ls(A) is indeed assigned to A by doing a reverse lookup of each one (except L1(A) which was already looked up above). Only when the reverse lookup of a given peer locator

has completed is that locator marked as verified. This reverse lookup of each locator prevents 3rd party DoS attacks as described in [M6THREATS].

At this point in time B knows that A has context state and it knows the flow label to use, thus it can start sending packets using the context.

8. Host A receives the CONF message, records F(peer) from the packet, and if present, extracts the Lsr(A) and forms the intersection Lsl(A) and Lsr(A) and uses this to update Ls(A). If the resulting Ls(A) does not contain AID(A) this is an error and the AID is added to the locator set.

   If a ULP packet was piggybacked on the CONF message, A will pass that to the ULP as long as the source locator of the CONF message was one of as part of the verified Ls(peer) locators.

   At this point in time A knows that B has context state and it knows the flow label to use, thus it can start sending packets using the context.

4.2. Locator Change

This is the sequence of events when B receives a packet with a previously unused source locator for A, for instance L2(A).

Host B receives M6 packet with source L2(A) and destination L1(B). Looks up context state using the flow label. If this lookup succeeds and the source address field contains a locator which is in Lsl(B), then the locator is acceptable for incoming packets (even though it might not have been verified for use as return traffic) and the packet is rewritten to contain the AIDs from the context state and passed to the ULP.

If L2(A) has not been verified then it would make sense for B to put that locator first in the list of asynchronous DNS verifications that are needed. If/once L2(A) has been verified B can make it the preferred peer locator to use when sending packets to AID(A).

The verification needs to complete before using the locator as a destination in order to prevent 3rd party DoS attacks [M6THREATS].

If a host receives a packet with a known flow label but where the locators (source and/or destination) are not part of the locator sets, the packet is silently dropped as specified in more detail in Section 7.11.

4.3. Concurrent Context Establishment

Should both A and B attempt to contact each other at about the same time using the same AIDs for each other, the message flow is different than above. However, if different AIDs are used this would result in two completely independent contexts between the two hosts following the basic content establishment above.

If B tries to contact A after B has received the CCR message, then B will discover the context state for AID(A) and not trigger creating a new context. But since B does not create any state when receiving the INIT, it is possible for B to send a INIT after it has received an INIT (and sent a CC), as well as the case of two crossing INIT messages.

4.3.1. Crossing INIT messages

Here is the sequence of events when A starts talking to B at the same time as B starts talking to A:

1. A sends an INIT message to B. As part of this A creates proto-state for the context and allocates its flow label. It has Lsr(B) from the DNS lookup.

2. B sends an INIT message to A. As part of this B creates proto-state for the context and allocates its flow label. It has Lsr(A) from the DNS lookup.

3. B receives the INIT message from A. It discovers the context it created in step 2 since it matches the AIDs. Thus it can immediately send a CONF message containing its flow label and Lsr(A).

4. A receives the INIT message from B. It discovers the context it created in step 1 since it matches the AIDs. Thus it can immediately send a CONF message containing its flow label and Lsr(B).

5. A receives the CONF message from B and can complete the context state creation.

6. B receives the CONF message from A and can complete the context state creation.

4.3.2. Sending INIT after sending CC

Here is the sequence of events when A starts talking to B, and after B has received the INIT but before B has received the CCR, B starts establishing a context with A:

1. A sends an INIT message to B. As part of this A creates proto-state for the context and allocates its flow label. It has Lsr(B) from the DNS lookup.

2. B receives the INIT message from A. It finds no matching context thus it proceeds in the normal context establishment and sends a CC message to A.

3. A ULP on B wants to send a packet to AID(A) which triggers B to send an INIT message to A. As part of this B creates proto-state for the context and allocates its flow label. B has Lsr(A) from the DNS lookup.

4. A receives the CC message from B and responds with a CCR message as in the normal context establishment.

5. A receives the INIT message from B. It discovers the context it created in step 1 since it matches the AIDs. Since A has sent the CCR it knows that the B will soon complete the context establishment and respond with a CONF message. Thus the INIT can be silently ignored.

6. B receives the CCR message from A. At this point in time it discovers that it is a bidirectional setup because it has context state indicating that it has sent an INIT message. B needs to inform A of the flow label it selected in step 3, which is does in the CONF message.

4.4. Handling Locator Failures

Should not all locators be working when the communication is initiated some extra complexity arises, because the ULP has already been told which AIDs to use. If the locators that where selected to be AIDs are not working it isn't possible to defer the context establishment, but instead it needs to be performed before ULP packets can be successfully exchanged. If the destination locator isn't reachable, the M6 layer needs to retry the INIT message with different destination locators until a working one is found. If the source locator isn't reachable for return traffic, routers which rewrite the source locator at site exit can be helpful to convey to

the peer which locator to use. When the initiator's AID is not working as a locator, it isn't possible to piggyback ULP packets on the INIT message, to prevent using packets with a source locator different than the source AID being used for reflection attacks that would bypass any deployed ingress filtering.

After context setup the sender can use retransmit hints from the ULP to get the M6 layer to try a different verified locator. This is the only possibility when only one end is (re)transmitting ULP packets and the destination locator is no longer working. Some heuristics are needed in the M6 layer to determine which alternative destination locator to try, and how often to try a different one when there are persistent failures.

If one outbound path from the site fails and the border routers rewrite source locators then the peer will see packets with the working source locators. Once that locator has been verified by the peer, the return path will switch to use the working locator. As long as both ends are transmitting packets this will relatively quickly switch to working locators except when both hosts experience a failing locator at the same time.

Without locator rewriting, it would be beneficial to add some notification e.g., by defining a new bit in the router advertisement prefixes to indicate that the prefix is problematic to use for due to failures at the site border (IMHO this is semantically different than the preferred vs. deprecated semantics), but we would also need some mechanism to carry this information from the border routers to the routers on each subnet. Perhaps the router renumbering protocol [RFC2894] could be extended to carry this information.

4.5. Locator Set Changes

Due to events like site renumbering, the set of locators assigned to a host might change at a slow rate. Since this proposal uses the locators in the DNS as the credible source for which locators are assigned, there is some coordination necessary to ensure that before a host, or the border routers for a site doing rewriting, start using a new source locator, the peer has been informed about the new locator. The Update Request message is sufficient to inform the peer that a new locator should be acceptable as the source of packets from the host, while DNS verification needs to be involved to make that locator usable as a destination.

Due to concerns about having packets with unknown, hence potentially bogus, source locators triggering DNS lookups this proposal instead uses the DNS TTL in combination with the locator sets in the Update

Request message, as an indication that the set of locators need to be refreshed.

Each host determines when its own notion of its locators change, and uses the Update Request message to inform the peer of these changes. If a locator is removed from the set this will make the peer immediately stop considering that locator as part of the locator set. However, if an additional local locator is communicated to the peer in an Update Request, it can not be added to the verified locator set at the peer until it has also been seen in the DNS. Thus, when a host observes that it has a new locator, the host might want to verify that this locator has been added to the DNS for itself, before announcing it to the peer in an Update Request message.

When a host receives an Update Request with an additional locator for the peer and the DNS TTL for the FQDN->Ls lookup has expired, the peer will redo this DNS lookup to find the, perhaps updated, set of locators in the DNS. (Presumably failures to redo the lookup shouldn't have a negative effect.)

TBD: Even if there is no Update Request, should the DNS verification be redone periodically based on the DNS TTL? If so, what should the minimum time be to avoid DNS storms for FQDNs which have a very low TTL?

When the DNS lookup for the peer's FQDN returns a different locator set than previously, the host will inform the peer of the new Lsr locator set in an Update Request. However, since this change to Lsv(peer) doesn't effect which source locators are acceptable in received packets, it is not time critical to send the Update Request message.

If a host wants to modify the set of locators from which the peer accepts packets for the context, the host can send an Update Request message with a different Lsl(sender).

When a host sees (based on router advertisements [DISCOVERY]) that one of its locators has become deprecated and it has additional locators that are still preferred, it is recommended that the host stop using the deprecated locator(s) as the source locator with the contexts that have already been established. This ensures that, should the deprecated locator become invalid, the peers have already verified other locator(s) for the host.

TBD: Is there is a need to explicitly signal to the peer "this locator is deprecated - please verify another locator and use that as Lp(peer)"

4.6. Preventing Premeditated Redirection Attacks

The threats document [M6THREATS] talks of premeditated redirection attacks, that is, where an attacker claims to be a host before the real host appears. The absence of an actual IP layer identifier in this proposal makes that a non-issue; the attacker could only claim to be host A if the attacker is reachable at one of A's locators. Thus by definition the attacker would have to be on the path between the communicating peers and such attackers can already perform redirection attacks in today's Internet.

5. HANDLING STATE LOSS

The protocol needs to handle two forms of state loss:

- a host loosing the M6 layer state followed by packets arriving from a peer which hasn't lost the state. This could be due to the host crashing and rebooting, or due to the M6 layer discarding the state too early.

- the M6 layer garbage collecting state too early due to not being aware of what all ULPs do, resulting in the ULP passing down packets when there is no context state any more.

Part of the first case is the already existing case of a host crashing and "rebooting" and as a result loosing transport and application state. In this case there are some added complications from the M6 layer since a peer will continue to send packets assuming the context still exists and due to the loss of state on the receiver it isn't even able to pass the correct packet up to the ULP (e.g., to be able to get TCP to generate a reset packet) since it doesn't know what AIDs to use when replacing the locators.

The second case is a bit more subtle and has two facets. Ideally an implementation shouldn't discard the context state when there is some ULP that still depends on this state. While this might be possible for some implementations with a fixed set of applications, it doesn't appear to be possible for implementations which provide the socket API; there can be things like user-level "connections" on top of UDP as well as application layer "session" above TCP which retain the identifiers from some previous communication and expect to use those identifiers at a later date. But the M6 layer has no ability to be aware of this.

Thus an implementation shouldn't discard context state when it knows it has ULP connection state (which can be checked in e.g., Unix for

TCP), or when there is active communication (UDP packets being sent to AID(A) recently), but when there is an infrequently communicating user-level "connection" over UDP or "session" over TCP the context state might be garbage collected even though it shouldn't.

Knowing whether the ULP depends on the M6 layer state is not sufficient; even if the ULP doesn't, the peer host might still keep the context state. Thus the peer might send a packet using the context at some future time. To the host this looks similar to the reboot state loss, in the sense that it receives packets from the peer and has no matching context. The host doesn't know whether this is due to a legitimate peer which has retained the context state, or whether it is a DoS attack using random flow labels. However, if the ULP on the host passes down a packet to transmit it would turn into the second case.

5.1. State Loss and Packets from the Peer

If B crashes and reboots and A retransmits a packet with flow label, L3(B), L2(A) then what is needed on B is a packet to L1(B) from L1(A) passed to the ULP so that the ULP can send an error (such as a TCP reset). But B has no matching state thus it needs to send an Unknown Context error back to try to help A discover the state loss.

Another case is when B decides that the context has not been used for a long time and as a result discards the context state, and then a packet (perhaps a TCP SYN for a new connection) arrives from the peer using the context i.e., an M6 packet with a flow label. In this case, B has no choice but send an Unknown Context error back to A. (But see Appendix A for a method to remove the need for this error ever arising.)

However, if A blindly trusts the Unknown Context message and uses it to restart a context establishment, that is, discarding its current context state and sending a INIT, then this could be used by attackers to force extra work, but also it would allow an attacker that arrives on the path after a context has been established to destroy the existing context and insert itself as a Man-in-The-Middle for the new context establishment.

Given that the locators might be rewritten by routers, the main thing A has to identify the context is the flow label in the packet that caused the Unknown Context error at B. While this uniquely identifies the context on host B, it does not do so on A. Thus if A maintains some number of contexts with different peers, it might not be able to uniquely tell to which context the Unknown Context error should be applied. The use of the source address field in the

Unknown Context message should help disambiguate things, but it isn't clear to what extent this helps. This ambiguity is an opportunity that an attacker can take advantage of. The introduction of the birthday counter (an idea from [HIP]), with a relatively small window (e.g., 900) of acceptable birthday counter values can substantially reduce the probability of off-path attackers being able to use the Unknown Context error to kill a context by sending bogus Unknown Context errors.

If we in addition introduce the close exchange, as suggested in Appendix A, we also remove most of the ability of an attacker to cause extra work by having a host respond to packets with random flow labels with Unknown Context errors. This combination of the birthday counter and the CLOSE/CLOSEACK messages, means that the Unknown Context error only needs to be when packets with an unknown flow label are received during the first X minutes after the reboot.

5.2. State Loss and Packets from the ULP

If host B only lost (for instance, by garbage collecting too early) the M6 context state, things are a bit more complicated for packets passed down from the ULP. Without any context state the M6 layer on B can not determine whether packets to AID(A) coming from the ULP are destinated to a standard IPv6 host, or to a host which supports multihoming. At a minimum the host needs to try to determine this, and if it somehow determines that the peer supports multihoming, then it should try to determine the peer's locators and reestablish a host-pair context.

B can determine whether A is M6 capable by doing a reverse lookup of AID(A)->FQDN(A) followed by a FQDN(A) lookup to see of there is an M6 record (and get the locator set of A as well). Or, if DNS reverse lookups are undesirable or do not work, perhaps a packet could be exchanged with A to ask it whether it supports multihoming. Simply sending an INIT message to the AID(A) will not only tell it whether A supports multihoming, but also let B know the flow label and Lsl(A) locators to use. But as this protocol is currently specified, using a new ICMP type, an INIT message sent to a host which doesn't support multihoming will be silently dropped. TBD: Would it make sense to instead use a new payload type for the M6 messages to at least receive an ICMP payload type unknow error from hosts which do not support multihoming?

If B is communicating with both standard IPv6 hosts and hosts which support multihoming, then for performance reasons it should avoid doing these DNS lookups or peer queries for every packet sent to a standard IPv6 host. Implementation tricks (such as "has this socket

ever used M6" flag at the socket layer, "negative caching" of peers that do not support M6) can be useful to avoid performance overhead, and caching of the peers that do support M6 can all be used to address this performance concern.

If, as part of this, B determines that A is M6 capable, it has the same information as the initiator during the initial context establishment thus it can follow that procedure starting by sending an INIT message. If A didn't garbage collect its end of the state this will result in receiving a CONF message from A which includes the flow label etc.

6. ENCODING BITS IN THE IPv6 HEADER?

The idea is to pick extra IP protocol values for common combinations, and have a designated protocol value to capture the uncommon IP protocols which might use M6. The uncommon IP protocol values would require an additional extension header when used over M6.

We pick two unused ranges of IP protocol values with 8 numbers each (assuming we will not need more than 7 common transport protocols). The ranges start at P1 and P2, respectively:

```
P1      TCP over M6 - rewrite ok
P1+1    UDP over M6 - rewrite ok
P1+2    SCTP over M6 - rewrite ok
P1+3    RDDP over M6 - rewrite ok
P1+4    ESP over M6 - rewrite ok
(...)
P1+7    escape - any protocol over M6 - rewrite ok
        In this case we spend another 8 bytes (minimum IPv6
        extension header size due to alignment rule) to carry the
        actual IP protocol.  This causes some mtu concerns for those
        protocols, but they aren't very likely to be used with M6?

P2      TCP over M6 - no rewrite
P2+1    UDP over M6 - no rewrite
P2+2    SCTP over M6 - no rewrite
P2+3    RDDP over M6 - no rewrite
P2+4    ESP over M6 - no rewrite
(...)
P2+7    escape - any protocol over M6 - no rewrite
        In this case we spend another 8 bytes (minimum IPv6
        extension header size due to alignment rule) to carry the
        actual IP protocol.  This causes some mtu concerns for those
        protocols, but they aren't very likely to be used with M6?
```

Thus a router would check if the protocol is in the P1 range and if so, it can rewrite the locator(s). A host would check a received packet against both P1 and P2 ranges and if so pass it to the M6 shim layer.

Some possible alternatives to the above encoding:

- use some combination of the universal/local and group bit in the interface id of the source address field to indicate "rewrite ok".

- always have a M6 shim header - adds 8 bytes overhead per packet.

7. PROTOCOL PROCESSING

A more detail description of the protocol is presented in this section.

7.1. State Machine

The protocol can be described using the following states:

o IDLE: no state for the context at all.

o INIT-SENT: an INIT message has been sent.

o CCR-SENT: a CCR message has been sent.

o ESTABLISHED: a CCR or CONF message has been received thus the context is fully established with data packets being carried using the flow labels and Section 6 encoding.

o UR-SENT: Established but awaiting an Update Acknowledgement.

7.2. Sending INIT messages

When a host needs a host-pair context it first checks if there already is a context which matches the pair of AIDs. If not, it will establish such a context by sending an INIT message. The sender of the INIT message is a called the "initiator" and the peer is called the "responder" even if this terminology is inexact when both ends send INIT messages at about the same time.

The initiator must verify that the AIDs are part of the locator sets;

if this is not the case it is most likely due to some local confusion on the initiator since the AIDs were selected (see Section 4.1) from the locator sets.

If the initiator knows that it doesn't have a FQDN or that a verification using forward and reverse lookups would fail, for instance due to the reverse tree not being populated, it might as well only include the AID in Lsl(initiator) since this saves the responder the effort to try to verify things in the DNS.

The initiator creates a proto-context state and allocates a unique flow label; F(local).

Then it picks a random [RANDOM] nonce to include in the INIT message, sends the INIT message, and sets the state to INIT-SENT.

The first INIT message is sent to the destination locator which is the AID. Should the INIT message be retransmitted, a different destination locator should be used. The INIT messages, as all other M6 control messages, can be sent with the "rewrite ok" bit set. If the routers do not rewrite the source locators it might be necessary for the initiator to try different source locators as well as destination locators as it is retransmitting the INIT message.

7.3. Receiving INIT messages

When a host receives an INIT message it first checks whether it has an existing context for the AID pair.

If such a context is found, the following additional checks are applied:

- The state is INIT-SENT or ESTABLISHED
- The IP source address field in the INIT contains a locator which is part of Lsl(peer)
- The IP destination address field in the INIT contains a locator which is part of Lsl(host)

If any of the above checks fail the INIT message is silently dropped. If all the checks succeed, the host can update the existing context from the INIT message (the peer's birthday counter, locator set, and flow label) and respond with a CONF message. The CONF message should include the nonce from the INIT message, the local flow label, the local birthday counter, Lsr(peer) and Lsl(responder) from the existing context. If the INIT message matching an existing context

has piggybacked a ULP packet, this packet is passed to the ULP.

In there is no state matching the AID pair, the host verifies that the AID(responder) is in fact one of its locators. Should this fail the INIT message is silently dropped.

Then the responder forms a CC message using a per-host key and a timestamp or serial number to produce a keyed hash to protect the state information carried in the CC message and returned in the CCR message.

If the responder knows that it doesn't have a FQDN or that a verification using forward and reverse lookups would fail, for instance due to the reverse tree not being populated, it might as well only include the AID in Lsl(responder) in the CC message since this might save the initiator the effort to try to verify things in the DNS.

If the INIT packet has a piggybacked ULP packet, that is, the nexthdr value is something different than NONXTHDR, this packet can potentially be passed to the ULP. It is safe to pass it to the ULP when the source and destination address fields in the INIT packet are the AIDs, and it is also safe to accept such packets when the destination address field contain a locator different than the AID. But if either the INIT packet was sent with a different source locator or the source locator was rewritten in transit, it is not safe to pass the piggybacked packet to the ULP. (This is because until the context state is created, any "response" packet from the ULP would be sent as a regular IPv6 packet to the peer AID. The need to prevent reflection attacks which can bypass any deployed ingress filtering means that we need to avoid this when the INIT packet was not sourced by the AID.) In this case the responder sets the "ULP packet discarded" bit in the CC message.

7.4. Receiving CC messages

The host looks for a matching context based on the AID pair. If no context is found, or if the context is not in state INIT-SENT, or if the nonce doesn't match what was sent in the INIT, the CC message is silently dropped.

Otherwise the host records the Lsl(peer) from the CC message, changes the state to CCR-SENT, and sends a CCR message. The CCR message can either reuse the nonce used with the INIT message, or a new random nonce can be selected.

The host forms Ls(peer) as the intersection between Lsr(peer) and

Lsl(peer). If this intersection does not contain AID(peer) it is an error and the AID is added to the set.

If the CC packet has a piggybacked ULP packet, this packet can potentially be passed to the ULP. It is clearly safe to pass it to the ULP when the source and destination address fields in the CC packet are the AIDs. But it seems to also be safe when the address field contents fall in the locators sets as known to the initiator. If this is not the case, then the piggybacked ULP packet is silently ignored.

7.5. Receiving CCR messages

The timestamp/serial is verified to be recent, and then the timestamp/serial is used to determine which per-host key was used for the keyed hash in the CC message. Using this key, the keyed hash is verified. If it does not verify, or if the timestamp/serial is too old, the CCR message is silently dropped.

Then the host looks for a matching context based on the AID pair. If a context is found this could be the second form of concurrent context establishment. The host performs the following checks:

- The state is INIT-SENT
- The IP source address field in the CCR contains a locator which is part of Lsl(peer)
- The IP destination address field in the CCR contains a locator which is part of Lsl(host)

If any of the above checks fail the CCR message is silently dropped.

If the checks all succeed, the host can update the existing context from the CCR message (the peer's birthday counter, locator set, and flow label) and respond with a CONF message. If this update results in Ls(local) or Ls(peer) not including the corresponding AID, this is an error and the AID is added to the set.

The CONF message should include the nonce from the CCR message, the local flow label, the local birthday counter, the Lsr(peer), and Lsl(responder) from the existing context.

If a context is not found, then one is created based on the information in the CCR message. The host allocates a flow label for the context. The host MAY perform a reverse plus forward DNS lookup starting with AID(peer), and if so it includes the resulting

Lsr(initiator) in the CONF message. Otherwise, the CONF message only contains the nonce, flow label and birthday counter.

If the CCR packet has a piggybacked ULP packet, it will be passed to the ULP independently of the IP address fields in the CCR message. This is possible because even if the CCR contains spoofed locators and/or AIDs, any "response" packet from the ULP will only be sent to the verified peer locators now that the context state has been created.

7.6. Receiving CONF messages

The host looks for a matching context based on the AID pair.

If no matching context is found the CONF message is silently discarded.

If the state is INIT-SENT and the nonce does not match the nonce sent in the INIT, the CONF message is silently discarded. If the state is CCR-SENT and the nonce does not match the nonce sent in the CCR, the CONF message is silently discarded.

In any other state the CONF message is silently discarded.

Then the CONF message is used to record the peer's flow label. If the CONF message contains the Lsr(initiator) this is used to update the context. As part of this Ls(initiator) is updated to be the intersection of Lsr(initiator) and Lsl(initiator). If this intersection does not contain AID(initiator) this is an error and the AID is added to the set.

If the CONF message contains the Lsl(responder), which is the case during bidirectional establishment i.e., the CONF was sent in response to an INIT message, then this is used to update the context and Ls(responder) is updated to be the intersection of Lsr(responder) and the received Lsl(responder). If this intersection does not contain AID(responder) this is an error and the AID is added to the set.

Finally the state is changed to be ESTABLISHED.

If the CONF packet has a piggybacked ULP packet, this packet can potentially be passed to the ULP. It is clearly safe to pass it to the ULP when the source and destination address fields in the CONF packet are the AIDs. But it seems to also be safe when the address field contents fall in the locators sets as known to the initiator. [Given that the CONF is know to be the result of a INIT/CCR packet,

it can be assumed that the locators fall in the sets?]

If this is not the case, then the piggybacked ULP packet is silently ignored.

7.7. Sending Update Request messages

When a hosts sees that either its own locator set has changed, or that the DNS verification (initial, or redone after DNS TTL expiry) of the peers locators show a reduction in the locator set, the host should send an Update Request.

The sender of the Update Request should verify that the AIDs are part of the locator sets as part of sending the Update Request.

The sender picks a random nonce to include in the message, sends the message to Lp(peer), and sets the state to UR-SENT.

7.8. Receiving Update Request messages

The host looks for a matching context based on the AID pair and the flow label pair in the Update Request message.

If no context is found, the Update Request is silently dropped.

If a context is found, it is verified that the locators that are in the IP address fields are part of the Lsl locator sets. If not, the update request is silently dropped. Then the locators in the Update Request are used to update the context state. As part of this the intersections of the local and remote locator sets are maintained. Should either of the intersections not include the corresponding AID this is an error and the AID in question is added to the set.

The receipt of a reduced Lsl(sender) should result in immediately no longer using the removed locators as destination locators for transmitted packets. The receipt of an increased Lsl(sender) should, if the remaining DNS TTL for the FQDN has reached zero, trigger a a DNS verification (forward and reverse lookup) of the new locators. Should this verification complete successfully, the locator(s) will be added to the verified set of peer locators hence be valid for transmitting packets.

The receipt of changes to Lsr(receiver) is less critical since the peer will accept as source locators any of the locators that are part of the Lsl(receiver) that was communicated to the peer during context establishment or in the most recent Update Request.

The host responds to the Update Request by sending an Update Acknowledgement back to the sender of the request copying the nonce/timestamp, AIDs and flow labels from the request.

7.9. Receiving Update Acknowledgement messages

The host looks for a matching context based on the AID pair and the flow label pair in the Update Acknowledgement message.

If no context is found, the Update Acknowledgement is silently dropped.

If a context is found, it is verified that the context state is in UR-SENT and that the nonce in the acknowledgement matches the nonce that was sent in the request. If either of those checks fail, the message is silently dropped.

Otherwise, the state for the context is changed back to ESTABLISHED.

7.10. Retransmission

The INIT messages are retransmitted by the initiator until a CC or a CONF message with a matching nonce is received.

The CC messages are not retransmitted; if they are lost the initiator will retransmit the INIT message.

The CCR messages are retransmitted by the initiator until a CONF message with a matching nonce is received.

The Update Request messages are retransmitted until an Update Acknowledgement with a matching nonce is received.

These retransmissions continue with binary exponential backoff until CONTEXT_IDLE_TIME (suggested 5 or 15 minutes) have passed. The initial retransmit timer is 4 seconds. It is suggested that some randomness be applied to the retransmit timer to avoid synchronization should lots of hosts in a site follow the same pattern of retransmissions.

7.11. Receiving Data Packets

Received M6 control packets (INIT) etc are dispatched based on the ICMP code and processed as indicated in the sections above.

Received M6 data packets (as indicated by the payload types in section 6) are processed as follows:

1. Lookup the context based on the flow label

   1a. If no context found, drop the packet and send a Unknown Context message to the sender.

2. Compare the destination address field with Lsl(local)

   2a. If destination is not in Lsl(local), silently drop the packet.

   2b. If destination is Lp(local), no action is needed.

   2c. If destination is not Lp(local), should we change Lp(local) so that the source locator for transmitted packets is changed? [TBD]

3. Compare the source address field with Lsl(peer), Lsv(peer)

   3a. If source is not in Lsl(peer), silently drop the packet.

   3b. If source is in Lsl(peer) but not in Lsv(peer), trigger verifying that locator (and accept the packet).

   3b. If source is Lp(peer), no action is needed.

   3c. If source is not Lp(peer) and is in Lsv(peer), change Lp(peer) so that the destination locator for transmitted packets is changed.

In the cases where the packet wasn't dropped above, the packet is rewritten to contain the AIDs from the context state and passed to the ULP.

7.12. Receiving Unknown Context messages

Look for matching contexts (there might be multiple) that have the same peer flow label as the one in the UC message, and where Lsl(peer) in the context includes the source address field of the UC message.

If no such context is found, silently discard the UC message.

If the birthday counter in the UC message is the same as the recorded birthday counter for the peer, the UC message was due to the context

state being garbage collected without the peer rebooting.

If the birthday counter in the UC message is greater than (using serial number arithmetic) the recorded birthday counter, but is not more than MAX_BIRTHDAY_INCR than this number, then the UC message is assumed to be due to a reboot of the peer.

In both of the above cases the context can be discarded and a new context be initiated by sending an INIT message. [TBD: discard vs. update the existing context based on the CC and CONF messages?]

If the birthday counter doesn't match either criteria, then UC message is silently dropped.

In both above cases of acceptable UC messages it is RECOMMENDED that the host also verify whether the included packet is consistent with a packet that the host might have sent recently; for instance that in the case of UDP, TCP, or SCTP, the port numbers match port numbers that are currently in use and that the TCP and SCTP sequence numbers are reasonable for the matched connections. [The utility of these checks is limited, because for some ULPs such as ICMP echo messages, the host might not have any way to check things hence it is required to accept the UC message.]

7.13. DNS Verification and Hosts without a FQDN

A host without a FQDN, or a host which knows that the forward and reverse DNS information for it is missing or inconsistent, can remove the need for the peer to detect this by passing a Lsl in INIT and CC messages which consists of only the AID for the context. When doing so, the "rewrite ok" bit should not be set for those messages.

This approach is possible even if the host has multiple locators; it implies that such a host can take advantage of any multihoming of the peer even though it can't take full advantage of itself being multihomed. Thus the peer's locators can change for the context, but the host itself can only use its AID for the context.

When a host verifies the locator relationship it treats the peer's AID as being implicitly verified by the context establishment handshake as long as the AID was used as the locator for this exchange. This is basically relying on the return routability property for the AID being a locator. Thus is Lsl(peer) contains the AID as a single member, and that AID is used as the locator during the context establishment, there is no need to perform any DNS verification.

Since the host might not itself know that it doesn't have a FQDN or

that the reverse plus forward lookups would fail to verify, the peer also needs to take this into account in the verification. When an initiator verifies things using the reverse lookup and it discovers that the lookup for one or more of the peer's locators fail with a permanent DNS error, it should exclude those locator(s) from the set. And if all of them fail, it should still include the peer's AID as the only member in the peer's locator set.

When a responder tries to verify the peer by performing DNS lookups (reverse and forward), if it fails to perform a reverse lookup on the peer AID due to a permanent DNS error, which is needed to find the FQDN, then it will assume that the peer has no FQDN. In this case the Lsr(peer) will contain only the AID(peer) i.e., the peer's locator can not change. However, it will still accept data packets with any of the Lsl(peer) locators in the source address fields; it will not send packets to any locator but the AID.

If the reverse lookup of the peer's AID fails with a transient DNS error, it is recommended that the DNS lookup is redone (using binary exponential backoff until it either succeeds or fails with a permanent error).

If the DNS lookups fail with a permanent error it is recommended that the locator set with the failed locators removed is signaled to be the peer as Lsr(peer) in an Update Request message.

This will make the peer stop using those locators as source locators even though the host sending the Update Request will continue to accept packets from any of the locators in Lsl(peer).

The initiator starts off with Lsr(responder) being based on what the forward DNS lookup returned. As successful reverse lookups, that is reverse lookups that point at the same FQDN, are completed on these locators, the locators are added to Lsv(responder) i.e. become candidate destination locators. The initial Lsv(responder) is the AID even without a reverse lookup. But should the INIT message be (re)transmitted to different destination locators, the AID will not be considered verified unless there has been a successful reverse lookup.

The responder starts off with Lsv(initiator) containing only the AID, if and only if the AID was used as a locator during the context establishment. If the AID was not used as a locator during the establishment, the Lsv(initiator) will initially be empty. This implies that data packets can not be sent until at least one locator has been verified using the DNS. [TBD: Is this too strict? Can we allow the locator that was used during establishment to be in Lsv even though it is not the AID?]

When the host receives messages (CC, CONF, Update Request) which shrinks Lsl(peer), it will remove the deleted locators from Ls(peer) and as a result from Lsv(peer) as well.

TBD: Need to know whether there is a strict definition of temporary vs. permanent DNS errors. Things like NXDOMAIN should be a permanent error.

7.14. Birthday Counter

Each host needs to maintain a birthday counter in stable storage to help with safe resynchronization when one of the ends have lost its state.

When a host is first initialized it allocates a random initial value for the birthday counter [RANDOM]. Each time the host looses all state, that is, crashes or powers off and reboots, it increments the birthday counter by one and saves the result in stable storage.

The protocol needs a constant, MAX_BIRTHDAY_INCR, which should be chosen so that during the lifetime of a context after the peer has stopped sending (Appendix A uses X minutes for this), no host should reboot and increment its birthday counter more than this number of times.

It might be reasonable to assume that no host can reboot more frequently than once a second. This implies that if the context is discarded 15 minutes after the last packet was received, the peer could have rebooted at most 900 times after it sent the last packet using the context. As a result the probability of an off-path attacker hitting this window of 900 in a 32-bit birthday counter is about 1 in a million. A 64-bit birthday counter might be overkill.

8. COMPATIBILITY WITH STANDARD IPv6

A host can easily implement M6 in a way that interoperates with current IPv6 as follows.

When the DNS lookup routines do not find an M6 record for the peer they will return the AAAA resource record set to the application; those would be the IPv6 addresses. When the ULP passes down these addresses, the M6 layer will not have any state generated by the DNS lookup code, thus no M6 processing will take place on the sender. (Note that this relates to the M6 layer state recovery in section 5.2)

The receive side handles both standard IPv6 and M6 since it demultiplexes on whether a packet is an M6 packet, that is, based on the payload types in Section 6.

9. APPLICATION USAGE OF IDENTIFIERS

The upper level protocols will operate on AIDs which are mere locators. Thus as long as a site hasn't renumbered, the AID can be used to either send packets to the host, or (e.g. if that locator isn't working) it is possible for an application to do a reverse lookup plus forward lookup of the AID to get the set of locators for the peer.

Once a site has been renumbered, the AIDs which contain the old prefix will no longer be useful. Hence applications must try to honor the DNS TTL somehow. But this is a renumbering issue and not an effect of the multihoming support.

Applications, which map the peer's IP address to a domain name, today perform a reverse lookup in the DNS (e.g., using the getnameinfo() API). This proposal doesn't add or subtract to the benefits of performing such reverse lookups.

Applications which today either retain a peer's IPv6 address for future use, such as connecting back to that peer ("callbacks"), or pass a peer's IPv6 address to a third party ("referrals") will not break with this multihoming support; they will end up retaining and/or passing the AID instead of an IPv6 address. Since the AID is a locator things will still work as long as that locator is reachable.

However, the AID doesn't contain information whether the host is M6 capable, thus the callbacks and referrals would use IPv6 without multihoming support unless something special is done. To be able to take advantage of the multihoming support the application or host would need to detect whether the AID is M6 capable, for instance, by doing a reverse lookup on the AID to get the FQDN and the a forward lookup on the FQDN to look for the "M6 capable" DNS RR. An alternative would be to use the suggestion in Section 5.2 to send an INIT message to the peer to discover if it is M6 capable.

Also, should the locator which is the AID not be reachable, the application/host will fail to communicate with the peer. A reverse plus forward lookup of the AID, or the INIT suggestion, can be performed to discover all the locators.

Whether these lookups can be hidden from the application, or whether the applications need to be modified to make the callbacks and referrals take full advantage of the multihoming is for further study.

10. CHECKSUM ISSUES

The IPv6 header does not have a checksum field; the IPv6 address fields are assumed to be protected by the ULP pseudo-header checksum. The general approach of an M6 shim which replaces locators with identifiers (where only the identifiers are covered by the ULP checksum) raises the potential issue of robustly handling bit errors in the address fields.

With the definition of the M6 shim there can be undetectable bit errors in the flow label field or the nexthdr field which might adversely affect the operation of the protocol. And since the AIDs are what's covered by the ULP's pseudo-header checksum the locators in the address fields are without checksum protection. An undetected bit error in the source locator would look like an unverified source locator to the receiver. In this proposal such packets are silently ignored.

Except for the obscure case when Ls(A) contains multiple locators, one or more of those are not working, and the bit error causes L1(A) to be replaced by L2(A). That would make the return traffic go to L2(A), but that might be a non-functioning locator. In this case the mistake will be corrected when a subsequent packet is received from A.

An undetected bit error in the destination address field is also harmless; it might cause misdelivery of the packet to a host which has no context but when the peer receives any resulting Unknown Context error message, it will not contain a source locator which is in the peer's locator set, hence it will be silently ignored.

An undetected bit error in the IPv6 next header field can potentially make a M6 packet appear as a non-M6 packet and vice versa. This isn't any different than undetected bit errors in IPv6 next header field without multihoming support.

An undetected bit error in the flow label in a data message could have two possible effects: not finding any context state, or finding the incorrect context state. In the first case any Unknown Context error message would be dropped by the peer since the flow label included in the error message doesn't match the flow label that was

originally sent. In the second case this will result in a packet with incorrect identifiers being delivered to the ULP which most like will drop it due to ULP checksums not matching.

11. IMPLICATIONS FOR PACKET FILTERING

Ingress filtering should be replaced by locator rewriting when the "rewrite ok" bit is set.

Locator rewriting (when the bit is set) can be applied at places where ingress filtering isn't currently performed (e.g., due to multihoming issues).

Firewall filtering potentially require modifications to be aware of M6. All the packets contain locators thus a stateful firewall would need to be aware of the context state to let the correct packets through as locators might change during some communication/connections. Such firewalls could optionally perform their own verification by issuing DNS lookups the same way as the endpoint. However, the firewalls probably has to be more careful not exposing themselves to DoS attacks by doing too much DNS lookups.

12. IPSEC INTERACTIONS

As specified, all of ESP, AH, and key management is layered above the M6 layer. Thus they benefit from the stable AIDs provided above the M6 layer. This means the IPsec security associations are unaffected by switching locators.

The alternative would be to layer M6 above IPsec, but that doesn't seem to provide any benefits. Since we want to allow routers performing locator rewriting it wouldn't be possible to take advantage of for instance AH to protect the integrity of the IP headers.

A result of layering M6 above IPsec is that the M6 protocol can potentially be used to redirect IPsec protected traffic as a selective DoS mechanism. If we somehow could require IPsec for the M6 protocol packets when the ULP packets between the same hosts use IPsec, then we could prevent such attacks.

However, due to the richness in IPsec policy, this would be a bit tricky. If only some protocols or port numbers/selectors are to be protected by IPsec per a host's IPsec policy, then how would one

determine whether M6 traffic needs to be protected? Should one take the conservative approach that if any packets between the hosts/AIDs need to be protected, then the M6 traffic should also be protected?

For this to be useful both communicating hosts would need to make the same policy decisions, so if we are to take this path there would need to some standardization in this area.

13. SECURITY CONSIDERATIONS

This analysis is far from complete. Early analysis indicates this addresses the issues in [M6THREATS].

Just as in today's Internet hosts on the path can inject bogus packets; in this proposal they need to extract the flow labels from the packets in order to do this which wouldn't be hard. Packet injection from off-path places becomes harder since it requires guessing the 20 bit flow label together with locators that are in the locator sets. If ingress filtering is deployed the attacker might not be able to freely choose the source locator, thus just as in today's Internet ingress filtering further limits attackers ability to inject packets with spoofed source addresses.

An attacker can inject bogus Update Request messages by picking random flow labels and locators, in an attempt to make the communication break or make it use less locators. Worst case this can cause the communication to drop down to only using the AIDs and no other locators. Such an attacker needs to know the AID pair and guess the flow label pair for the attack to be successful. Guessing a total of 40 bits of flow labels would be hard for an off-path attacker. One could make this even harder by, in addition to the flow labels, have the context establishment exchange some larger random numbers between the peers, and using those numbers to compute a HMAC on the Update Request. This would still be subject to Man-in-The-Middle attacks for on-path attackers, but it would make it a lot harder for off-path attackers to hit something with random packets.

An attacker can inject bogus Unknown Context messages. The protection against this is a locator and flow label match as well as hitting the relatively small window of acceptable Birthday Counters. In addition, the receiver will reject Unknown Context messages if the included payload packet was not a packet that it might have recently sent.

DNS verification implications TBD.

14. PRIVACY CONSIDERATIONS

The existence of mappings in the DNS from any locator to a FQDN and from the FQDN to all the locators of a host can potentially make it harder than in today's Internet for a host that is part of a multihomed site to be anonymous.

This is especially true if the host wants to take advantage of RFC 3401 temporary addresses, or if the host is moving between different subnets since the forward and reverse information would need to be updated.

Thus a host which is in a multihomed site which desires to have multiple pseudonyms and use the M6 protocol would need to, not only have multiple locators per prefix (as in RFC 3041), but also have multiple FQDNs each FQDN corresponding to a separate set of locators.

Note that hosts that communicate with peers that are multihomed are not required to have a FQDN to take advantage of the multiple locators of the peer, hence they can retain the same amount of pseudonyms as with RFC 3041.

15. DESIGN ALTERNATIVES

Several aspects of the protocol can be tweaked while following the original idea of using a set of locators as an equivalence set and using the DNS to verify the set membership.

A few to consider are listed in this section.

15.1. Avoid introduction of M6 DNS Resource Record type

It is possible to avoid introducing a new resource record type to indicate whether a peer is M6 capable.

Instead one of the M6 packets can be used. For instance, a host can send an INIT message to the peer and if the host receives a response it knows the peer is M6 capable.

This potentially has some performance implications, hence it would be desirable to structure the use of the protocol slightly differently. For instance, if the peer is not M6 capable it is useful if the INIT message would result in an ICMP error being returned. As currently specified, with the M6 messages being an informational ICMP type, no such error message will be returned. Thus it would be better if the M6 messages were instead defined to be a separate new payload type so

that hosts which do not implement the protocol respond with an ICMPv6 "payload type unknown" error.

If the initiator doesn't know whether the peer supports M6 or not, it would seem better to not piggyback a ULP packet on the INIT message but instead send any ULP packet separately. Since this might be suboptimal when the peer does support M6, it would be useful for the host to cache which AIDs/FQDNs support M6 so it can use piggybacking in that case.

And since sending the INIT message over and over to a host which doesn't support M6, it would also be useful for the host to cache which AIDs/FQDNs do not support M6 so that it can skip trying an INIT message when it has already tried once.

15.2. Extension header instead of using flow label

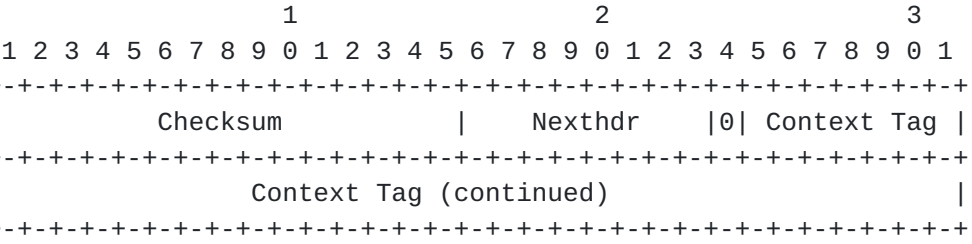
Use an actual extension header for M6 and use a context tag in that header instead of using the flow label. This would make the packets 8 bytes larger since the minimum extension header size is 8 bytes due to the alignment rules for extension headers in IPv6.

With an 8 byte extension header one could fit

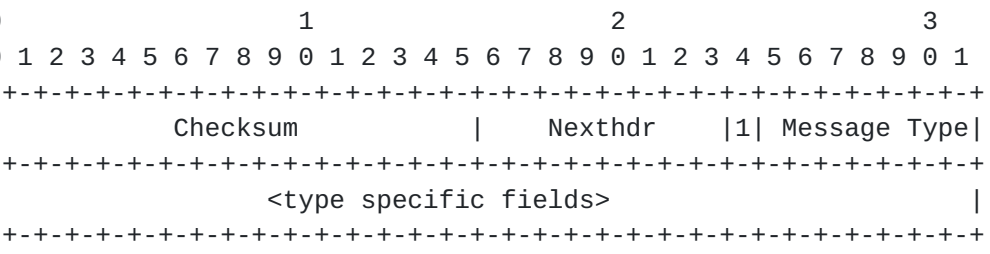
- 8 bits of next header value
- 16 bits of checksum (over the extension header only) [TBD: is this useful?]
- 1 bit to indicate that it is a M6 data packets (other M6 packets would have e.g. a 8 bit message type instead)
- 39 bits of context tag

As a result one would only need to allocate 2 protocol values instead of the 16 suggested in Section 6: One protocol value for "M6 - rewrite ok" and one value for "M6 - no rewrite".

The data messages could look like this:



M6 control messages could look like:



15.3. Explicit close exchange

Instead of relying on the Unknown Context error to try to synchronize when one has garbage collected the context state and the other end has not, we could define an explicit close exchange. An early attempt at this is specified in Appendix A. This approach would remove any DoS concerns related to responding to Unknown Context errors.

16. OPEN ISSUES

Either we need to specify an upper lifetime of a context after no packets have been received on the context, or we need to adopt a close handshake akin to the example in Appendix A. This need arises because the logic for the birthday counter needs to know how many times a peer can reboot while the host maintains an old context with that peer; this number determines the window of allowed birthday counter values for the Unknown Context error.

Which DNS errors should be treated as temporary vs. permanent?

Does it make sense to establish a larger context tag for each direction in addition to the flow label, so that Update Request messages and Close messages can be less subject to random packet injection?

Is it possible to facilitate transition to M6 using some "M6 proxy" at site boundaries until all important hosts in a site have been upgraded to support M6? Would would be the properties of such a proxy? Would it place any additional requirements on the protocol itself?

Would destination locator rewriting be a useful way for the routing system to pass some information to the host? Or is source locator rewriting sufficient?

Understanding the performance of DNS verification with and without DNSsec. With DNSsec how many public key signature verifications are likely to be needed for the reverse lookup of each locator?

How does the use of the last verified source locator as a destination locator for subsequent return traffic interact with ingress filtering? It would be nice if the protocol could operate with uncoordinated ingress filtering between the different exits/ISPs, but it is unclear whether this is feasible.

16.1. Renumbering Considerations

Need to write down any special coordination needed when a locator is added to a locator set or when one is removed; this can happen when a site is renumbered.

16.2. Initiator Confusion vs. "Virtual Hosting"

When A wants to communicate with host B and host X at the same time there can be some confusion since the DNS could return partially overlapping locator sets for the two remote hosts. For example,

The lookup of FQDN(B) returns Ls(B) which contains L1(B), L2(B), ... Ln(B).

The lookup of FQDN(X) returns L1(B), L1(X)

The result is that connections that could be intended to go to B and to X could both end up with an AID=L1(B), but the multihoming shim layer would have two separate locator sets associated with L1(B). Thus at a minimum when the second of the two communications starts there has to be some way to resolve this conflict.

In Section 4.1 this is resolved by the initiator performing a reverse lookup on the AID. Thus looking up L1(B) in the ip6.arpa tree in the above example. That works because it would return FQDN(B) thus X could be safely declared as being bogus. As a result communication with X would not be possible.

However, in many (IPv4) hosting setups today multiple domain names (www.foo.com, www.bar.com) are served by a single IP address. In this case the reverse lookup can't point back at both names unless the PTR resource record contains multiple records with different names. Per [RFC2181] section 10.2 this is allowed but it doesn't appear to be commonly used.

Can we depend on this little used feature of the PTR usage? If not it would seems to mean that each locator can only be used with one

FQDN which would be more restrictive than we have with IPv4 today.

17. ACKNOWLEDGEMENTS

The first version of this document was the result of discussions in a MULTI6 design team but is not the "product" of that design team. The scribe wishes to acknowledge the contributions of (in alphabetical order): Iljitsch van Beijnum, Brian Carpenter, Tony Li, Mike O'Dell, and Pekka Savola.

The idea to allow locator rewriting by routers was first presented by Mike O'Dell [ODELL96]. The techniques for avoiding state DoS attacks on the first packet are patterned after [MIPv6]. The idea to use a set of locators and not inventing a new identifier name space, as well as using the DNS for verification of the locators, was first brought up by Tony Li.

Later versions of the document have benefited from the comments of many participants in the Multi6 WG.

18. REFERENCES

18.1. Normative References

[M6THREATS] Nordmark, E., and T. Li, "Threats relating to IPv6 multihoming solutions", draft-ietf-multi6-multihoming-threats-00.txt, July 2004.

[ADDR-ARCH] S. Deering, R. Hinden, Editors, "IP Version 6 Addressing Architecture", RFC 3513, April 2003.

[IPv6] S. Deering, R. Hinden, Editors, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2461.

[RANDOM] Eastlake, D., Crocker, S. and J. Schiller, "Randomness Recommendations for Security", RFC 1750, December 1994.

18.2. Informative References

[NSRG] Lear, E., and R. Droms, "What's In A Name: Thoughts from the NSRG", draft-irtf-nsrg-report-09.txt (work in progress), March 2003.

[MIPv6] Johnson, D., C. Perkins, and J. Arkko, "Mobility Support in

IPv6", RFC 3775, June 2004.

[AURA02] Aura, T. and J. Arkko, "MIPv6 BU Attacks and Defenses", draft-aura-mipv6-bu-attacks-01 (work in progress), March 2002.

[NIKANDER03] Nikander, P., T. Aura, J. Arkko, G. Montenegro, and E. Nordmark, "Mobile IP version 6 Route Optimization Security Design Background", draft-nikander-mobileip-v6-ro-sec-02 (work in progress), June 2003.

[ODELL96] O'Dell M., "8+8 - An Alternate Addressing Architecture for IPv6", draft-odell-8+8-00.txt, October 1996,

[MAST] D. Crocker, "MULTIPLE ADDRESS SERVICE FOR TRANSPORT (MAST): AN EXTENDED PROPOSAL", draft-crocker-mast-protocol-01.txt, October, 2003.

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

[IPv6-SA] R. Atkinson. "Security Architecture for the Internet Protocol". RFC 2401, November 1998.

[IPv6-AUTH] R. Atkinson. "IP Authentication Header", RFC 2402, November 1998.

[IPv6-ESP] R. Atkinson. "IP Encapsulating Security Payload (ESP)", RFC 2406, November 1998.

[RFC2181] R. Elz, and R. Bush, "Clarifications to the DNS Specification", RFC 2181, July 1997.

[RFC3697] J. Rajahalme, A. Conta, B. Carpenter, S. Deering, "IPv6 Flow Label Specification", RFC 3697, March 2004.

[RFC2894] M. Crawford, "Router Renumbering for IPv6", RFC 2894, August 2000.

[RFC3041] T. Narten, R. Draves, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", RFC 3041, January 2001.

[HIP] Robert Moskowitz, "Host Identity Protocol", 11-Jun-04, <draft-ietf-hip-base-00.txt>

[WIMP] Jukka Ylitalo, "Weak Identifier Multihoming Protocol (WIMP)", July 2004, <draft-ylitalo-multi6-wimp-01.txt>

19. CHANGE LOG

Changes since version -01:

- Made the protocol allow for locator rewriting of all the M6 control messages.
- Allow for hosts without a FQDN or without a correct forward+reverse relationship benefit from the peer being multihomed.
- Handle inconsistencies between the DNS and host configuration, e.g. due to two-faced DNS or DNS-based load balancing, by having the host tell its peer what it received from the DNS and use the intersection of the DNS info and the info known by the host itself.
- Removed the probability of DNS inconsistencies resulting in packets being dropped due to having an unknown source locator; a host informs its peer which locators should be acceptable as source addresses while the destination locators continue to be verified in the DNS.
- Changed the context establishment to start with an INIT message from the Initiator instead of having the establishment be driven by a message with an unknown flow label arriving at a host. This allows the Initiator to decide when to establish the context; before the context is established "regular" packets can be exchanged as long as the locators used are the AIDs.
- Renamed the messages to have the same names as in WIMP; INIT/CC/CCR/CONF
- Simplified the uniqueness requirements for the flow label to be "unique per receiving host" in order to handle dynamic changes (driven from the DNS) of the set of locators assigned to a host. Without this change it was impossible to make the flow label plus locators uniquely identify a context at the receiver when the set of locators changes over time (e.g., due to renumbering). This limits each (virtual) host to about 1 Million contexts at any given time. Removing this limitation would imply carrying a larger context tag in an extension header, which is a possible tweak to this protocol.
- The simplified uniqueness requirement resulted in needing receiver instead of sender allocation of the flow label, which in turn resulted in the initiator sending the first message (the INIT) in the context establishment exchange.
- Made it possible to allow locator rewriting on the context establishment messages, while allowing piggybacking of ULP packets.

However, in some cases when locators are being rewritten, ULP packets can not be piggybacked until the context is established, to avoid security exposure.

- Added details of how the host-pair context is established when both ends initiate the setup at the same time.

- Worked out the details for how messages are retransmitted. As a result, the 4th message in the establishment exchange needs to be mandatory to ensure that the initiator always knows the correct flow label to use in transmitted packets. Thus the 3-way context establishment exchange becomes a 4-way exchange as in [HIP] and [WIMP], with the ability to piggyback ULP packets on all those messages.

- As a result of needing a 4-way exchange there is no longer any benefit in attempting a provisional allocation of a flow label when receiving the INIT. Hence the responders flow label allocation is now done when receiving the CCR message.

- Added details about how the host-pair context state could be removed in a coordinated fashion in Appendix A.

- Added details on state recovery when one end has lost the peer's host-pair context state, using a birthday counter as in [HIP]

- Added a suggestion on how one can avoid introducing a new M6 DNS Resource Record type.

- Renamed "flowid" to correctly be "flow label"

APPENDIX A: CONTEXT CLOSE PROTOCOL

This scheme is robust against arbitrary network partitioning and loss, whether in both directions or in one direction, through the use of timers plus new CLOSE and CLOSEACK messages. It introduces one additional state in the state machine:

- CLOSING: about to go away but the state is retained to be able to reliably inform the peer that the state is being removed. No data packets can be sent using the context when in closing state, but otherwise it is the same as ESTABLISHED state.

The underlying observation is that the network doesn't spontaneously generate packets, thus for a packet to be received it must have been

sent by the peer, and if the host does not send a packet no packet will be received by the peer. (But packets might be sent and never received.)

In the basic analysis we assume that the network delay (propagation and queuing) is zero, that is, a packet is received by the peer either immediately when sent, or never (in the case of loss).

The mechanism uses a time constant: X minutes.

The mechanism works as follows:

1. When no packet has been received on the context for X minutes the context transitions to CLOSING state. A CLOSE message is transmitted to the peer.

   When in CLOSING state the host must not send any data packets using the context. If there is a need to send a data packet a new context must be created by starting by sending an INIT.

2. When the host receives a CLOSE message it discards the context state and responds with a CLOSEACK message. (The authenticity of the CLOSE message can be verified the same way data messages are verified, that is, the flow label needs to match and the locators be in the locator sets.) This processing applies whether or not the state is CLOSING in order to handle CLOSE messages from both ends crossing in flight.

   Once the context state has been discarded any need to send data packets will trigger establishing a new context, starting with sending an INIT.

3. A CLOSE message which is received when there is no context state can not be verified but will result in a CLOSEACK response to speed up the peer discarding the state in the presence of packet loss.

4. The CLOSE message is retransmitted until either a CLOSEACK message is received, or it has been retransmitted for a total of X minutes. When either occurs the context state is discarded.

5. When a host receives a CLOSEACK message it verifies that it is in CLOSING state and that the CLOSEACK was in response to the CLOSE (using e.g., a nonce in the CLOSE message).

   It is possible to use stronger verification of the CLOSEACK based on secrets tied to the context state, but only for the first CLOSE message since the state is discarded on its

reception. Thus if the CLOSEACK response to the first CLOSE is lost, the host would need to wait for the full X minutes until discarding its state.

Due to packet loss the two sides can have different views of when the last packet might have been sent, but because no received packets in X minutes causes a state transition, this difference can not be larger than X. Since the CLOSE messages are retransmitted for X minutes (during which the peer can not possibly receive any data packets) the peer will transition to CLOSING and stop sending data packets before the host will discard its state. Example 2 shows a case when this happens.

Example 1: working communication in both directions

Time T: A sends a packet to B. While A doesn't know it yet, this is the last packet A will send using the context. B continues to send packets to A.

Time T+X: B has not received any packets from A for X seconds. B marks the context as CLOSING, that is, it will not use the state to transmit any more. B sends a CLOSE message to A.

Time T+X: A receives a CLOSE message from A. Discards the context state and responds with a CLOSEACK message.

Time T+X: B receives the CLOSEACK message and discards the context state.

Example 2: Unidirectional failure; A->B packets are all dropped.

Time T: A sends a packet to B. While A doesn't know it, this is the last packet B will receive from A.

Time T+1 etc: A sends a packet to B which is lost.

Time T+X: B has not received any packets from A for X seconds. B marks the context as CLOSING, that is, it will not use the state to transmit any more. B sends a CLOSE message to A.

Time T+X: A receives a CLOSE message from A. Discards the context state and responds with a CLOSEACK message. The CLOSEACK message is lost.

Time T+X+1 etc: B retransmits the CLOSE message.

Time T+X+1 etc: A receives the CLOSE message, has no context state,

and responds with a CLOSEACK message. The CLOSEACK message is lost.

Time T+2X: B stops retransmitting the CLOSE message and discards the session state.

Example 3: Bidirectional failure; all packets are dropped.

Time T1: A sends a packet to B. While A doesn't know it, this is the last packet B will receive from A.

Time T2: B sends a packet to A. While B doesn't know it, this is the last packet A will receive from B.

Time T1+1 etc: A sends a packet to B which is lost. Time T2+1 etc: B sends a packet to A which is lost.

Time T1+X: B has not received any packets from A for X seconds. B marks the association as CLOSING, that is, it will not use the state to transmit any more. B sends a CLOSE message to A. The CLOSE message is lost.

Time T2+X: A has not received any packets from B for X seconds. A marks the association as CLOSING, that is, it will not use the state to transmit any more. A sends a CLOSE message to B. The CLOSE message is lost.

Time T1+X+1 etc: B retransmits the CLOSE message, which is lost. Time T2+X+1 etc: A retransmits the CLOSE message, which is lost.

Time T1+2X: B stops retransmitting the CLOSE message and discards the session state.

Time T2+2X: A stops retransmitting the CLOSE message and discards the session state.

Since the difference between T1 and T2 can't be more than X, we know that the session state can not be discarded before the other end has transitioned to CLOSING.

The above examples generalize to arbitrary packet loss; in no case will a data packet be received when there is no association state. Hence data packet that are received and have no matching session state can be silently dropped; no need to send an Unknown Context error or an INIT message.

Intuitively it seems like network delays can be handled to make the period for the retransmission of the CLOSE message be X+MSL (Maximum

segment lifetime) instead of X, but this needs to be verified.

The introduction of the close mechanism adds some considerations to the state machine. For instance, if an INIT arrives when in CLOSING state, the INIT would need to create a separate, new context, which has different flow labels at both ends. That way the context in CLOSING state is allowed to expire based on timeout or receiving a Close Acknowledgement message, in parallel with the new context being created.

This means that there might be more than one context for the same AID pair. This has the following implications:

- When handling packets passed down from the ULP, the M6 layer should not match any contexts in CLOSING state, but instead behave as if there was no context even when there is a context in CLOSING state. [TBD: It might be useful to copy certain information from the context in CLOSING state to the new context.]

- The context establishment packets (INIT, CC, CCR, and CONF) should also ignore any context in CLOSING state.

- Data packets received from the wire identify which context to use with the flow label field.

- M6 control packets that are sent after the context is established needs to be able to indicate which of possibly multiple contexts are intended. As a result, the Update Request, Update Acknowledgement, Close and Close Acknowledgement messages carry both flow labels which are used to identify the state.

AUTHORS' ADDRESSES

Erik Nordmark
Sun Microsystems, Inc.
17 Network Circle
Mountain View, CA
USA

phone: +1 650 786 2921
fax: +1 650 786 5896
email: erik.nordmark@sun.com

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.