INTERNET-DRAFT
Oct 27, 2003

Erik Nordmark
Sun Microsystems

Multihoming without IP Identifiers

<draft-nordmark-multi6-noid-01.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.

DISCLAIMER: This work has been discussed extensively in a design team. The design team is still exploring multiple approaches and

this is an attempt to capture one such approach on paper. Because of this and due to lack of time to review the document one can not say that this is a product of the DT; errors and confusions should be attributed to the scribe and not to the DT.

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 [M6SEC].
- 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 maps (ip6.arpa) populated for the hosts that wish to take advantage of multihoming will not be a significant problem.

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.

Ls(A) is the locator set for A, which consists of L1(A), L2(A), ... Ln(A).

AID(A) is an application ID for A. In this proposal, AID(A) is always one member of Ls(A).

3. PROTOCOL OVERVIEW

In order to prevent redirection attacks this protocol relies on the DNS (for the hosts which support this protocol) being maintained with consistent forward and reverse maps. 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 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 RR 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.

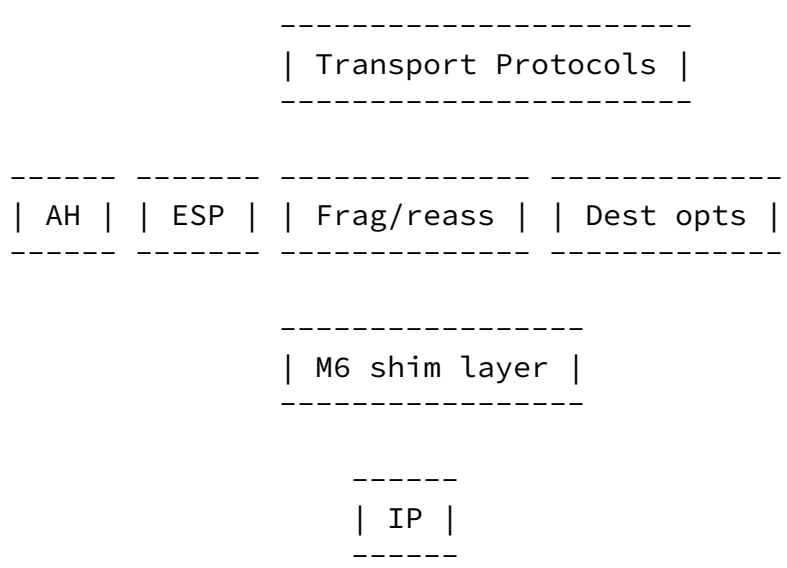


Figure 1: Protocol stack

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 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.

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.

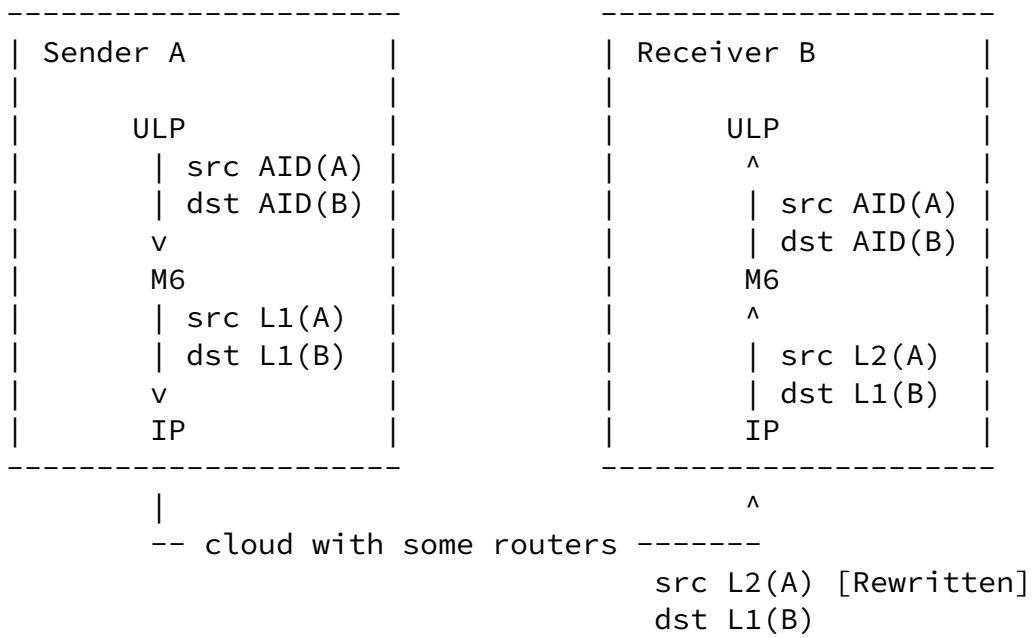


Figure 2: Mapping with router rewriting of locators.

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 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 flowid 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 flowid serves as a convenient "compression tag" without increasing the packet size, and this usage doesn't conflict with other flowid usage.

In addition to the zero overhead data messages, there are four different M6 message types introduced (which could be defined as new ICMPv6 messages). Three types are used to perform a 3-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 finally a single message type to signal that state has been lost. The four message types are called:

- Context request message; first message of the 3-way context establishment. Sent by the responder when a data packet arrives with no context state. An ULP packet can be piggybacked on this message.

- Context response message; second message of the 3-way context establishment. Sent in response to a context request. An ULP packet can be piggybacked on this message.

- Context confirm message; third message of the 3-way context establishment. Sent in response to a context response. An ULP packet can be piggybacked on this message.

- Unknown context message; 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 3-way state creation exchange has completed there is host-

pair context state at both hosts. At that point in time the responder (which didn't use DNS before the setup) can asynchronously start retrieving and 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. 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 3-way handshake and later discover and 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; Ls(peer)
- for each peer locator, a bit whether it has been verified with the DNS (by doing reverse + forward lookup)
- the preferred peer locator - used as destination; Lp(peer)
- the set of local locators; Ls(local)
- the preferred local locator - used as source; Lp(local)
- the flowid used to transmit packets; F(local)
- the flowid to expect in receive packets; F(peer)

- 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

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(peer) plus one of the locators in each of Ls(local) and Ls(peer) that are used to identify at most one state record. Thus the sender allocated flowid is part of the key for looking up the context state at the receiver.

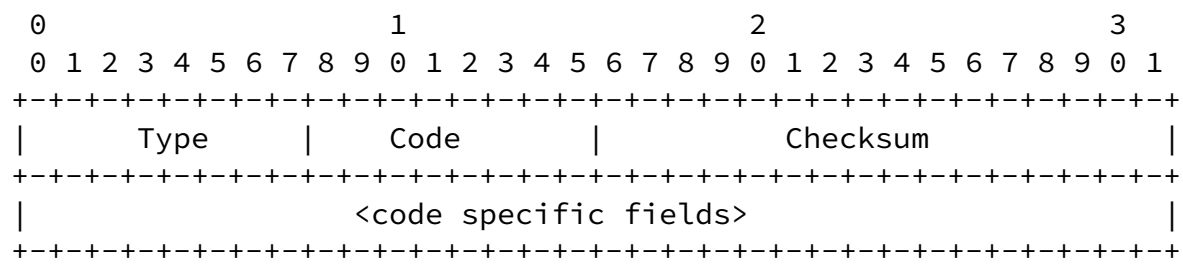
These uniqueness requirements imposed by those lookup keys uniquely identifying one state record means that one can not create multiple records (e.g. with different FQDN or locator sets) that have the same AID pair, and the peer must pick a flowid so that host-pair contexts which have at least one common members in Ls(local) and in the Ls(peer) sets, but with different AID pair, gets a different F(local). The context state at both ends must be consistent for this to be completely robust. One way of ensuring this is to have each host perform a periodic DNS lookup of its own FQDN in order to have a current Ls(local) that is the same as the Ls(peer) that the peer would find in the DNS.

Note that the flowids could be selected to be finer grain than above; for instance having a different flowid for each connection. Doing so requires some efficient data structure organization at the receiver to map multiple F(peer) to the same context.

3.2. Message Formats

These message formats are largely the same as in [CB128] but the context request, response, and confirm are sent in the opposite direction.

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 4 different types of messages:

- Context request message; first message of the 3-way context establishment. An ULP packet can be piggybacked on this message.
- Context response message; second message of the 3-way context establishment. An ULP packet can be piggybacked on this message.
- Context confirm message; third message of the 3-way context establishment. An ULP packet can be piggybacked on this message.
- Unknown context message; error which is sent when no context state found.

Checksum The ICMPv6 checksum.

Future versions of this protocol may define message codes. Receivers MUST silently ignore? Reject? [TBD] any message code they do not recognize.

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

specified below.

The Context request message contains:

- Sender Nonce
- Sender AID
- Receiver AID
- Sender flowid (20 bits)

The Context response message contains:

- Receiver Nonce (copied from Sender Nonce in request)
- Context state consisting of: the two AIDs, the two flowids, and the initial locators
- A timestamp or nonce (for sender's benefit)
- A hash over the context state and timestamp (to prevent modification)

The Context confirm message contains:

- The context state, timestamp/nonce, and hash copied from the context response.

The Unknown context message contains:

- The 20-bit flowid from the triggering packet.

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 Ls(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 AID(B)->FQDN and check that the result is 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 14.4 for some more discussion.

2. The ULP creates "connection" state between AID(A)=L1(A) and AID(B) and sends the first packet. 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 Ls(B). The existence of that state will make the M6 layer send a M6 packet. The M6 layer selects a flowid F(local) consistent with the uniqueness requirements in section 3.1 (which ensure that the receiver will map to the correct AID pair).

4. The packet (TCP SYN or whatever) is sent to peer with locators L1(A) to L1(B) i.e., the same as the AIDs. Since B doesn't have any context state yet, A will not set the "rewrite ok" bit in the header.

5. Host B receives packet and sees it is a "M6 packet". Passes the packet to the M6 shim layer. The M6 layer can't create state on the first packet, but since the rewrite bit is not set in the packet it can pass the packet unmodified to the ULP. The ULP sees a packet identified by AID(A), AID(B).

The M6 layer initiates a state creation 3-way exchange by forming a context request message. The same technique as in [MIPv6] can be used to securely do this exchange without any local state; use a local key which is never shared with anybody and pass the context state, a timestamp, and the keyed hash of the state+timestamp in the context request packet. When the state, timestamp, and keyed hash value is returned in the context response message, the hash is used to verify that the state hasn't been modified.

The 3-way exchange is done asynchronously with ULP packets, but it is possible (assuming the MTU allows) to piggyback ULP packets on this exchange.

Should ULP packets be passed down to the M6 layer on B before the context response 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 3-way exchange packets can flow in both directions using the original locators=AIDs. (However, this has some interactions with the suggestions in section 5.)

6. Host A receives the context request message. It verifies that the message is related to something it sent by looking at the locators (should match the AIDs) and the flowid it sent (which is in the state in the context request message).

   If a ULP packet was piggybacked A will pass that to the ULP.

   Then A sends a content response which has the same information as the context request plus a nonce/timestamp that A selected.

7. Host B receives the context response 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 packets as a DoS - A is also responding to context request messages. Thus B goes ahead and allocates state at this

point in time using the state that is in the context response message.

The M6 layer selects a flowid F(B) consistent with the uniqueness requirements in section 3.1 (which ensure that the receiver will map to the correct AID pair). At this point in time B has enough information to handle M6 packets from A, even though it hasn't yet determined and verified any additional peer locators from the DNS. It has also the state (F(B) mainly) necessary send data packets to A with "rewrite ok" set. Thus B sends a context confirm message to A which contains A's nonce/timestamp from the context response and F(B).

If a ULP packet was piggybacked on the context response B will pass that to the ULP.

At this point in time B can start asynchronously and incrementally extracting and verifying Ls(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 Ls(A). Then 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). 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 [M6SEC].

8. Host A receives the context confirm message, verifies the nonce/timestamp, and records F(peer) from the packet.

   If a ULP packet was piggybacked on the context confirm A will pass that to the ULP.

   At this point in time A knows that B has context state, thus it can start sending packets with "rewrite ok" set.

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 flowid and the locators. If this lookup succeeds 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 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 for 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 [M6SEC].

If a host receives a packet with a known flowid but where the locators (source and destination) are not part of the locator sets it drops the packet and sends an Unknown context error as specified in section 5.

4.3. 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 send a zero-overhead initial packet from A to B. Instead both the AIDs and the working locators need to be conveyed. This could be done by either reusing IP-in-IP encapsulating or defining another M6 message type which carries both. Details TBD.

After context setup the sender can use retransmit hints from the ULP to get the M6 layer to try a different verified locator.

If one outbound path from the site fails and the border routers rewrite source locators then the peer in another site will see packets with the working source locators. Once that locator has been verified, 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 one would need to add some notification e.g., by defining a new bit in the router advertisement prefixes (IMHO this is semantically different than the preferred vs.

deprecated stuff), but we also need some mechanism to carry this

info from the border routers to the routers on each subnet.

4.4. 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, that locator has propagated through the DNS so that the peer could have discovered it.

Due to concerns about having packets with unknown, hence potentially bogus, source locators triggering DNS lookups this proposal instead uses the DNS TTL as an indication that the set of locators need to be refreshed. One could also envision a combination of receiving a packet *and* the DNS TTL having expired as the trigger to redo the DNS lookups.

When DNS TTL expires on either host it performs a new FQDN->Ls lookup to get the new set of locators. (Presumably failures to redo the lookup shouldn't have a negative effect.)

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 start using the preferred locator(s) 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.

4.5. Preventing Premeditated Redirection Attacks

The threats document [M6SEC] 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 perform redirection attacks in today's Internet.

5. HANDLING STATE LOSS

The protocol needs to handle two forms of state loss:

- a peer loosing all state,
- the M6 layer garbage collecting state too early due to not being aware of what all ULPs do.

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. 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.

For instance, if B crashes and rebooted and A retransmits a packet with flowid, L3(B), L2(A) then what is needed 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. (Should the packet not have "rewrite ok" set host B can pass it to the ULP since it knows that such packets contain locators that are AIDs. But once the context has been established the peer is likely to send all packets with "rewrite ok" set.)

If host B instead only lost (garbage collected too early) the M6

context state things are a bit more complicated for packets passed down from the ULP. Without 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 a host which supports multihoming. Either B can determine this 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.

If B is communicating with both standard IPv6 hosts and hosts which support multihoming then it has to 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, and "negative caching" of peers that do not support M6) can be useful to avoid performance overhead.

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. If A didn't garbage collect its end of the state this will require some extra work to come up with a single host-pair context for a pair of AIDs at both ends with consistent flowids in the two hosts (i.e., F(local) needs to match F(peer) at the other host). Specifying this is for further study.

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 is to:

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

- steal the ECN bits from the traffic class before ECN becomes a proposed standard? Don't think this will be popular!

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

7. 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.)

The receive side handles both standard IPv6 and M6 since it demultiplexing on whether a packet is an M6 packet.

8. 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.

Applications which use to 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.

9. 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 flowid 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. Thus the packet would (after replacing locators with identifiers based on the context) be passed to the ULP and a challenge response exchange be triggered. In the case of a bit error in the locator this challenge isn't likely to receive a response; and if there is a response by someone it wouldn't be from the actual peer thus the verification would fail. Thus such an undetected bit error is harmless.

Except for the obscure case when Ls(A) contains multiple verified 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 the reception of the resulting Unknown context error message will show that it arrives from the incorrect locator thus it will be 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 flowid in a data message could have two possible effects: not finding any context state, or finding the incorrect context state. In the first case the Unknown context error message would be dropped by the peer since the flowid included in the error message doesn't match the flowid 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.

10. IMPLICATIONS FOR PACKET FILTERING

Ingress filtering should be replaced by locator rewrite 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 locator thus a firewall would need to be aware of the context state to let the correct packets through. 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.

11. IPSEC INTERACTIONS

As specified all of ESP, AH, and key management is layered above the M6 layer. Thus they benefit from the stable identifiers 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.

12. SECURITY CONSIDERATIONS

This analysis is far from complete. Early analysis indicates this addresses the issues in [M6SEC].

Just as in today's Internet hosts on the path can inject bogus packets; in this proposal they need to extract the flowids 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 flowid together with locators that are in the locator sets.

DNS verification implications TBD

13. DESIGN ALTERNATIVES

Use an actual extension header for M6 and use a context tag in that header instead of using the flowid. 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.

14. OPEN ISSUES

DNS lookup fails or times out on the receiver; what should one do? Send error?

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?

One of the issues with FQDNs mapping to AAAA records is that in some cases multiple AAAA records mean a multihomed host and in other cases it means multiple hosts providing the same service. If we need to introduce a new RR type for M6, would it be useful to try to make this host/service distinction more clear at the same time? An example solution would be that the M6 record would

by its existence indicate the M6 capability, and its RDATA would contain a list of host names which would be used to resolve the AAAA records for each host implementing the service.

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?

14.1. Handling Hosts without a FQDN

As specified in this document each host (including the initiating one) whether or not multihomed needs to have a FQDN.

However, it isn't hard to allow hosts without a FQDN to communicate with multihomed hosts that have a FQDN; as a result the hosts without a FQDN would not benefit from "rehoming".

This requires that 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 then it will assume that the peer has no FQDN. In this case the Ls(peer) will contain only the AID(peer) i.e., the peer locator can not change.

Whether the reverse lookup on the AID should be repeated (in order to handle transient failures) is TBD.

14.2. Locator Set Inconsistencies

Due to transient failures of the DNS lookups, misconfigured DNS

(returning different information "locally" than in remote lookups), or changes to the resource record sets during a renumbering event, the two ends of a context host-pair might have conflicting views on each others locator sets.

This can result in black holes if the sender uses a source locator which the receiver has not discovered using DNS lookups. It is unclear whether the error messages sent back could be used to detect and recover from this type of inconsistency.

But it is possible to add an additional protocol mechanism to make the two ends converge on the set of locators which is the intersection of what the two ends know. This could be done any

time after the context has been established.

For example, A could send some new message type to B containing what it thinks is Ls(A) and Ls(B). When B receives this message it calculates the intersection between the received sets and its knowledge of the locator sets. The result is used both in B's context state and sent back to A. When A receives the response it can verify that the result is in fact a subset of its existing locator sets (simply by forming the intersection between its state and the received sets) and use that. As a sanity check the AIDs should not be removed from the locator sets as part of this exchange.

Verifying the flowids in this exchange guards against off-path attackers artificially reducing the locator sets.

14.3. 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.

14.4. 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.

15. ACKNOWLEDGEMENTS

This document is 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].

16. REFERENCES

16.1. Normative References

[M6SEC] Nordmark, E., and T. Li, "Threats relating to IPv6 multihoming solutions", draft-nordmark-multi6-threats-00.txt, October 2003.

[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.

16.2. 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", draft-ietf-mobileip-ipv6-24.txt (work in progress), June 2003.

[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-01 (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.

[CB128] E. Nordmark, "Strong Identity Multihoming using 128 bit Identifiers (SIM/CBID128)", draft-nordmark-multi6-sim-00.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.

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

Full Copyright Statement

Acknowledgement

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