Network Working Group
Internet-Draft
Intended status: Experimental
Expires: October 18, 2008

T. Brunner
University of Applied Sciences,
Rapperswil
April 16, 2008

IKEv2 Mediation Extension
draft-brunner-ikev2-mediation-00

Status of This Memo

Abstract

This document describes the IKEv2 Mediation Extension (IKE-ME), a connectivity extension to the Internet Key Exchange IKEv2. IKE-ME allows two peers, each behind one or more Network Address Translators (NATs) or firewalls to establish a direct and secure connection without the need to configure any of the intermediate network devices. To establish this direct connection, a process similar to Interactive Connectivity Establishment (ICE) is used.

Table of Contents

1. Introduction

IKEv2 [RFC4306] inherently supports the traversal of Network Address Translators (NATs) by doing automatic NAT discovery during the IPsec connection setup. If a NAT situation is detected, IKE floats to UDP source and destination ports 4500 and after a CHILD_SA has been successfully established, ESP packets encapsulated in UDP datagrams [RFC3948] will share the same floated ports. While both IPsec and IKEv2 are peer-to-peer protocols by their nature, NATs and firewalls often restrict these protocols to a unidirectional mode where only the peer on the inside is able to actively set up a connection. If both peers are hidden by NATs or firewalls, the IKEv2 protocol usually fails to establish IPsec connectivity.

In the area of multimedia communications the Interactive Connectivity Establishment protocol [I-D.ietf-mmusic-ice] has been developed to solve the NAT and firewall problems mentioned above. Unfortunately the proposed solution is rather closely bound to the Session Initiation Protocol (SIP) and Session Description Protocol (SDP), and generally tends to solve problems specific to voice and/or video media streams.

The IKEv2 Mediation Extension (IKE-ME) adapts the connectivity establishment methods known from ICE to the IPsec domain, allowing secure IP connections to be established in environments with multiple NATs or firewalls.

The IKEv2 Mediation Extension protocol uses a mediation server to locate other peers and allows them to exchange their communication

endpoints. It implements an ICE-like mechanism with a minimum impact on the standard IKEv2 protocol. IKEv2 exchanges are used for communication between peers and the mediation server to simplify implementation in existing IKEv2 products.

1.1. Terminology and Notation

The following terms are used throughout this document:

Peer

A peer is an IKEv2 host that supports the protocol defined in this document and wants to establish a direct connection with another peer.

Mediation Server

A server is an IKEv2 host that helps peers to establish a direct connection between them. The server has to be reachable

by all peers involved in the mediation scheme.

Transport Address

A transport address is the combination of an IP address, a transport protocol (limited to UDP in this specification), and a port number.

Endpoint

An endpoint is a transport address that is obtained in order to be used in a direct connection. In addition to a plain transport address it has a type, a priority, and a base. The term endpoint may also be used to simply indicate the end of a connection. The actual meaning should be clear from the context.

Host Endpoint

An endpoint directly obtained from a local interface.

Server Reflexive Endpoint

Server reflexive endpoints are endpoints allocated on a NAT and are learned by a method such as Session Traversal Utilities for NAT (STUN).

Relayed Endpoint

Relayed endpoints are like remote host endpoints. Traversal Using Relays around NAT (TURN) is a possible source for relayed endpoints.

Peer Reflexive Endpoint

Peer reflexive endpoints are learned during connectivity checks. See Section 5 for how this is done.

Base

The base of an endpoint is the transport address from which messages are actually sent. For instance, a peer cannot send messages directly from a server reflexive endpoint which it got allocated on a NAT, but only from the host endpoint from which it obtained the server reflexive endpoint. See Section 3.3 for details.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",

"SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Protocol Overview

2.1. Basic Operation

In order to establish a direct connection between them, two peers need to connect to a mediation server first. The mediation server is required to forward the endpoints on which a peer is potentially reachable by another peer. Figure 1 provides a general overview of the most common situation. Peer 1 and Peer 2 want to establish a secure direct connection between them. Since both are behind a NAT they cannot reach one another directly - they most likely don't even know where to try. This is where the mediation server comes into

play. It helps to locate other peers and to exchange endpoints over which a peer may be reachable.

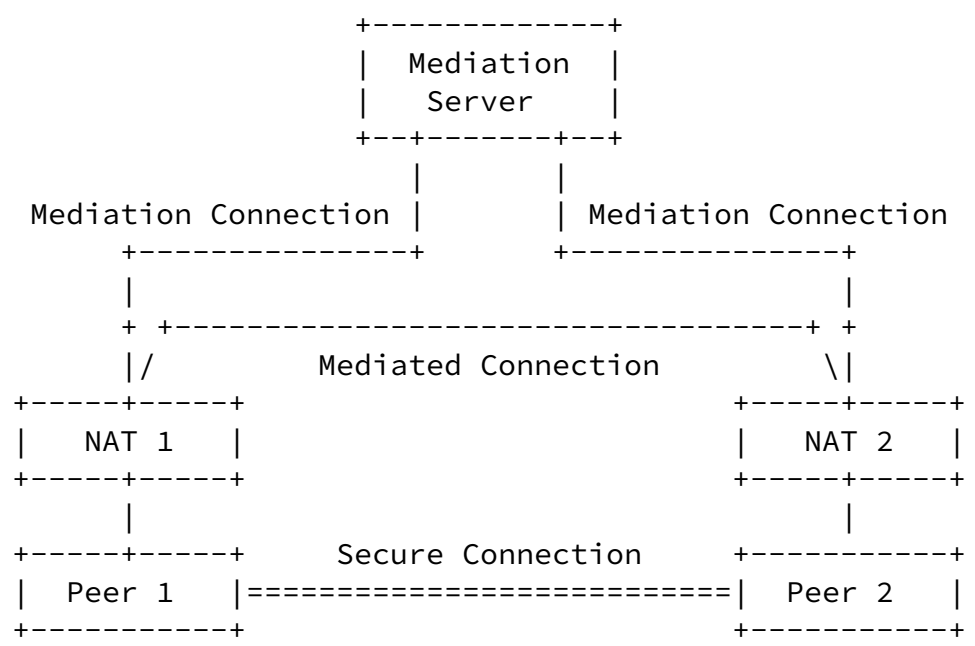


Figure 1: Overview

Each peer registers itself with the mediation server in order to announce its online presence. It does so by setting up an IKE_SA including special mediation payloads. No CHILD_SA is established between a peer and the mediation server because there is no need to exchange any encrypted IP payloads.

Before a peer can connect to other peers it has to collect a number of endpoints on which it is potentially reachable by other hosts. To

obtain endpoints an arbitrary method can be used. For instance, STUN might be used to learn server reflexive endpoints and TURN could be used to obtain a relayed endpoint. A client may also request a server reflexive endpoint from the mediation server. By connecting to the mediation server, the peer automatically gets transport addresses allocated on the intermediate NATs. The transport address on the NAT nearest to the mediation server is the source from which the mediation server receives the messages from the peer. This

transport address can be requested from the mediation server and provides a server reflexive endpoint.

If a peer requests a connection to another peer that is already registered, the mediation server acts as a relay to allow the peers to exchange their endpoints.

Each peer then performs connectivity checks on all available endpoint pairs constructed by combining its own with the received endpoints.

After all path combinations have been probed and the best suited endpoint pair has been elected, the initiating peer then goes on to set up an IKE_SA using the standard IKEv2 protocol and including at least one request for a CHILD_SA.

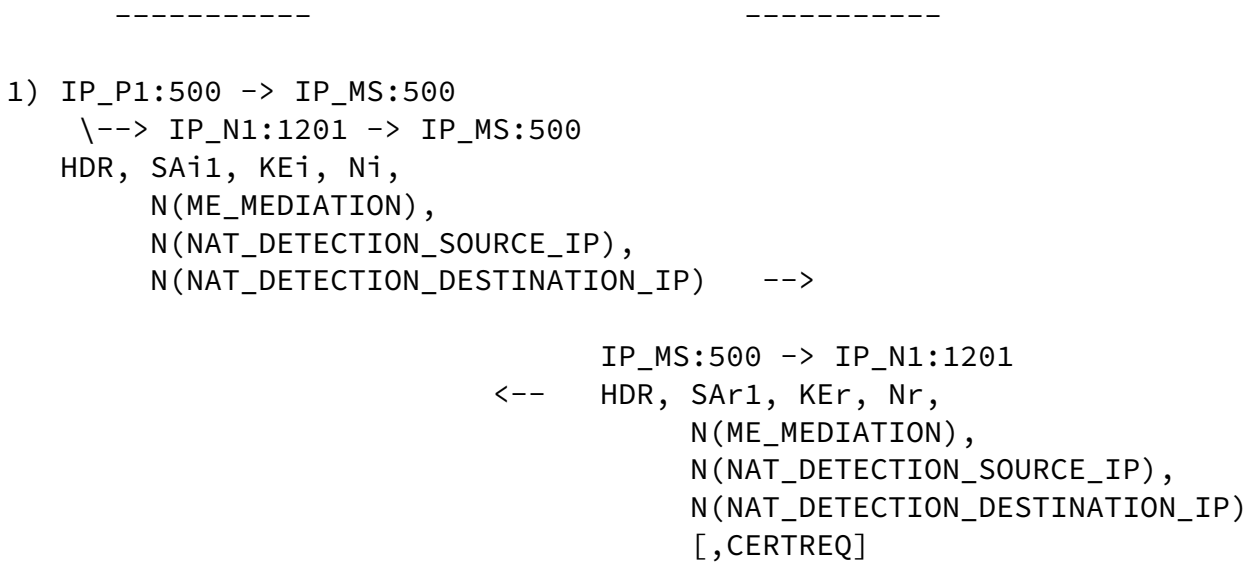
The protocol is designed to establish connectivity between peers in any network topology. As local endpoints are included in the checks, peers in the same (private) network can establish a connection directly. Depending on the NAT implementation, the used hole punching mechanism may not work. If both NAT are too restrictive, a relayed endpoint may be used to establish an IKE_SA between the peers.

2.2. Example Protocol Exchanges

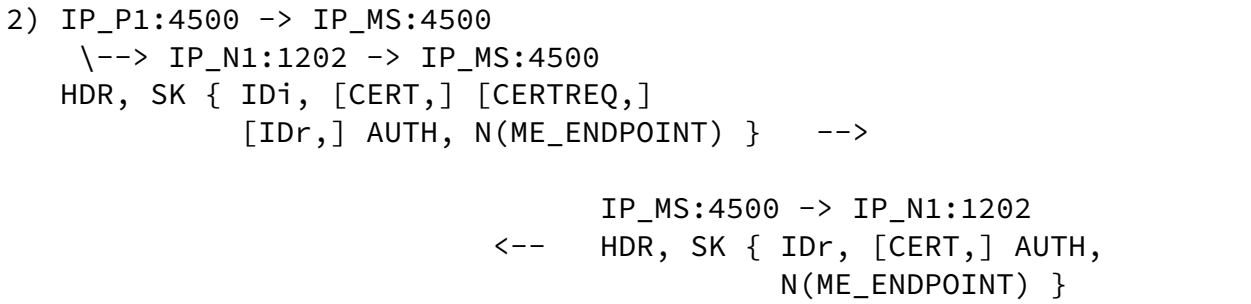
This section illustrates some example protocol exchanges. The notation is based on [RFC4306], Section 1.2. In addition, the source/destination IP addresses and ports are shown for each packet: here IP_P1, IP_P2, and IP_MS represent IP addresses used by the two peers and the mediation server, respectively. Referring to Figure 1, the two peers are each located behind a NAT. Thus, the modifications on outgoing packets, as performed by the NATs, are also shown. At this, IP_N1 and IP_N2 denote the public addresses of the NATs.

In a first step, Peer 1 connects to the mediation server starting with an IKE_SA_INIT exchange.

Initiator Responder



The IKEv2 NAT_DETECTION_SOURCE_IP and NAT_DETECTION_DESTINATION_IP Notify payloads are used to detect if there is any NAT between the peer and the mediation server. The new ME_MEDIATION Notify payload announces the request for a mediation connection. As mentioned above, we assume that both peers are behind a NAT. Therefore Peer 1 floats to UDP port 4500 before continuing with a modified IKE_AUTH exchange that does not contain a CHILD_SA proposal.



The peer uses the new ME_ENDPOINT Notify payload to request a server reflexive endpoint from the mediation server. After this exchange Peer 1 is connected to the mediation server and thus available for mediation with any other peer, as well as eligible to request a mediated connection itself. Peer 2 connects to the mediation server using the same procedure.

```
3) IP_P2:500 -> IP_MS:500
    \--> IP_N2:1024 -> IP_MS:500
   HDR, SAi1, KEi, Ni,
        N(ME_MEDIATION),
        N(NAT_DETECTION_SOURCE_IP),
        N(NAT_DETECTION_DESTINATION_IP)   -->

                                IP_MS:500 -> IP_N2:1024
                           <--  HDR, SAr1, KEr, Nr,
                                     N(ME_MEDIATION),
                                     N(NAT_DETECTION_SOURCE_IP),
                                     N(NAT_DETECTION_DESTINATION_IP)
                                     [,CERTREQ]

4) IP_P2:4500 -> IP_MS:4500
    \--> IP_N2:1025 -> IP_MS:4500
   HDR, SK { IDi, [CERT,] [CERTREQ,]
             [IDr,] AUTH, N(ME_ENDPOINT) }   -->

                                IP_MS:4500 -> IP_N2:1025
                           <--  HDR, SK { IDr, [CERT,] AUTH,
                                          N(ME_ENDPOINT) }
```

A direct connection is initiated by Peer 1 with the transmission of a ME_CONNECT request to the mediation server. Peers are identified by the ID with which they authenticate against the mediation server. So, this request includes the ID of the other peer, denoted IDp2, and several endpoints on which Peer 1 is potentially reachable by the other peer. Also included are a randomly generated ID and a randomly generated key that are mainly used for the ensuing connectivity checks.

```
5) IP_P1:4500 -> IP_MS:4500
    \--> IP_N1:1202 -> IP_MS:4500
   HDR, SK { IDp2, N(ME_CONNECTID), N(ME_CONNECTKEY),
             N(ME_ENDPOINT), N(ME_ENDPOINT) }   -->

                                IP_MS:4500 -> IP_N1:1202
                            <-- HDR, SK {}
```

The mediation server relays this ME_CONNECT request to the other peer, but replaces the IDp payload with the ID of Peer 1.

```
   IP_MS:4500 -> IP_N2:1025
   HDR, SK { IDp1, N(ME_CONNECTID), N(ME_CONNECTKEY),
             N(ME_ENDPOINT), N(ME_ENDPOINT) }   -->

                                  IP_P2:4500 -> IP_MS:4500
                                   \--> IP_N2:1025 -> IP_MS:4500
                            <--   HDR, SK {}
```

Peer 2 answers with a ME_CONNECT exchange of its own, including the initiating peer's ID, the connect ID, as well as its own randomly generated key and obtained endpoints. To mark the exchange as a response a ME_RESPONSE Notify payload is included. The mediation server extracts this information and forwards it back to Peer 1, again, exchanging the IDp accordingly.

```
6) IP_P2:4500 -> IP_MS:4500
    \--> IP_N2:1025 -> IP_MS:4500
   HDR, SK { IDp1, N(ME_RESPONSE), N(ME_CONNECTID),
             N(ME_CONNECTKEY), N(ME_ENDPOINT),
             N(ME_ENDPOINT) }   -->

                                  IP_MS:4500 -> IP_N2:1025
                            <--   HDR, SK {}

   IP_MS:4500 -> IP_N1:1202
   HDR, SK { IDp2, N(ME_RESPONSE), N(ME_CONNECTID),
             N(ME_CONNECTKEY), N(ME_ENDPOINT),
             N(ME_ENDPOINT) }   -->

                                  IP_P1:4500 -> IP_MS:4500
                                   \--> IP_N1:1202 -> IP_MS:4500
                            <--   HDR, SK {}
```

Both peers now pair their own endpoints with those received from the other end and proceed with connectivity checks. Connectivity checks are done using unprotected INFORMATIONAL exchanges that include the

connect ID, an ME_ENDPOINT payload, and a ME_CONNECTAUTH Notify payload, which contains a MAC to authenticate the sender of the check. In this example we assume that both NATs perform endpoint independent mapping and filtering.


```
7) IP_P1:4500 -> IP_P2:4500
   HDR, N(ME_CONNECTID), N(ME_ENDPOINT),
        N(ME_CONNECTAUTH) -->      !!  NOT REACHABLE

   IP_P1:4500 -> IP_N2:1025
   \--> IP_N1:1202 -> IP_N2:1025
   HDR, N(ME_CONNECTID), N(ME_ENDPOINT),
        N(ME_CONNECTAUTH) -->

                                    IP_P2:4500 -> IP_N1:1202
                                    \--> IP_N2:1025 -> IP_N1:1202
                               <--  HDR
```

Peer 2 does the same in the opposite direction. If at least one connectivity check is successful, the initiating peer proceeds with a normal IKE_SA_INIT request using the endpoints from the successful check.

3. Mediation Connection

This section describes the protocol between peers and the mediation server.

3.1. Initial IKE Exchanges

To establish a mediation connection with a mediation server an implementation MUST include a ME_MEDIATION notification in the IKE_SA_INIT exchange. The initiator MUST stop the initiation if the responder does not include a ME_MEDIATION notification in its response.

The format of the ME_MEDIATION notification is described in Section 7.

If the transport address used to communicate with the mediation server is also to be used as Host endpoint (see Section 3.3.2.2), the peer MUST now float to port 4500 even if no NAT is detected between the peer and the mediation server. Because connectivity checks are sent with non-ESP marker in front of the IKE header it would be confusing for implementations to receive such packets on port 500.

As no CHILD_SAs are established on mediation connections, the IKE_AUTH exchange differs from [RFC4306]. The payloads SAi2 and TSi, and SAr2 and TSr MUST be omitted from request and response, respectively. If any of these payloads are found included in the request, an implementation MUST respond with a NO_ADDITIONAL_SAS notification without any other payloads, and then delete the IKE_SA.

All other payloads of the IKE_AUTH exchange remain as defined in [RFC4306].

A peer MUST NOT have more than one connection to a specific mediation server at the same time. Thus, a mediation server MUST delete an existing IKE_SA with a peer upon receipt of a valid IKE_AUTH request of the same peer.

An implementation that supports MOBIKE [RFC4555] SHALL include the MOBIKE_SUPPORTED notification in the IKE_AUTH exchange.

Optionally, a peer MAY obtain a server reflexive endpoint from the mediation server, as described in Section 3.3.2.2.

3.2. CREATE_CHILD_SA Exchange

The absence of CHILD_SAs on mediation connections also affects the allowed usages of the CREATE_CHILD_SA exchange. Exchanges of this type SHALL only be used to rekey the IKE_SA. An implementation MUST respond to CREATE_CHILD_SA requests that demand the creation of a CHILD_SA with a NO_ADDITIONAL_SAS notification, without any other payloads.

3.3. Obtaining Endpoints

A peer obtains endpoints before requesting a mediated connection or before responding to such a request. There are four types of endpoints defined in this document - host, peer reflexive, server reflexive, and relayed endpoints. Since every peer decides on its own which endpoints it wants to share with other peers, the methods to obtain these endpoints can vary widely.

3.3.1. Host Endpoints

Host endpoints are obtained by binding ports to an IP address on a peer's host. A peer could use the same endpoint it uses to communicate with the mediation server, but it could also use a different port. If a peer is multihomed, it SHOULD obtain endpoints for every available IP address.

3.3.2. Server Reflexive and Relayed Endpoints

Server reflexive and relayed endpoints can be obtained from various sources. One possibility is to use STUN ([I-D.ietf-behave-rfc3489bis]) and its Binding Discovery and Relay Usages ([I-D.ietf-behave-turn]). This specification does not restrict implementations on the methods used to obtain such endpoints. But a peer SHOULD obtain server reflexive and MAY obtain

relayed endpoints for each host endpoint, to increase the probability of a successful connection.

Use of relays is expensive, and when using this protocol, relays will only be utilized when both peers are behind NATs that perform address and port dependent mapping. Consequently, some deployments might consider this use case marginal and decide not to use relays.

3.3.2.1. Considerations Concerning TURN

An implementation that opts for STUN's Relay Usage ([I-D.ietf-behave-turn]) as source for relayed endpoints has to consider several implications that result from that decision. For instance, as long as no active destination is set for such an endpoint, any IKE or ESP traffic that will be transferred through that endpoint will be encapsulated in Data Indication messages. Aside from the overhead of this additional layer of encapsulation, this also means that the implementation has to be able to process

such traffic. This may be significantly easier for IKE traffic, since IKE traffic is often processed in user space, whereas ESP traffic is usually handled in kernel space, where the introduction of an additional layer of encapsulation might be more difficult to implement. Therefore, it is RECOMMENDED that an owner of such a relayed endpoint sets an active destination as soon as it becomes apparent that the endpoint is being used to establish the mediated connection. Thus, it depends on the selected pair and the associated endpoints. If the initiator owns the relayed endpoint of the selected endpoint pair, it sets the active destination to the remote endpoint of that pair, just before sending the IKE_SA_INIT request to initiate the mediated connection. Because the responder does not know which pair finally gets selected by the initiator, it waits until it gets the IKE_SA_INIT request and just before sending the IKE_SA_INIT response sets the active destination to the endpoint provided in the REMOTE-ADDRESS attribute of the Data Indication message. In the extremely rare case of the selected pair consisting of two relayed endpoints, the procedure is the same, with both peers taking appropriate measures. This could happen, for instance, if both peers are behind a NAT and neither did provide server reflexive endpoints.

3.3.2.2. Obtaining Server Reflexive Endpoints from Mediation Servers

A peer MAY obtain a server reflexive endpoint from the mediation server. To do so, it includes a ME_ENDPOINT Notify payload either in the IKE_AUTH request or at a later stage in a separate INFORMATIONAL exchange.

The priority, family, and port fields of this payload are set to

zero, the address field is zero length, and the type field is set to SERVER_REFLEXIVE. Upon receiving such a payload, the mediation server includes in its answer a ME_ENDPOINT notification of the same type filling in the family, address and port of the endpoint it received the request from.

The mediation server MUST ignore the ME_ENDPOINT Notify payload if the type is not SERVER_REFLEXIVE [anchor11].

If MOBIKE [RFC4555] is in use on the mediation connection, detection of changes in NAT mappings SHOULD be activated (as specified in

[RFC4555], Section 3.8). A peer that previously obtained a server reflexive endpoint from the mediation server SHOULD refresh that endpoint, whenever MOBIKE indicates that the NAT mapping has changed.

3.3.3. Peer Reflexive Endpoints

Peer reflexive endpoints are different from the previous endpoint types. Endpoints of this type are never obtained before a connection attempt, but dynamically learned during the connectivity checks. The process of how and when these endpoints MAY be learned is explained in Section 5.

3.3.4. The Base of Local Endpoints

All local endpoints have a Base. This is the transport address used to send the actual messages for an endpoint. Since it is not possible to send messages directly from a server reflexive endpoint, the base of such an endpoint is the host endpoint from which the server reflexive endpoint was obtained. If the peer is not behind a NAT, the base of a server reflexive endpoint will equal that endpoint, which is then redundant and will be eliminated. The base of host endpoints is the endpoint itself. The same is true for relayed endpoints, since these are like remote host endpoints. Peer reflexive endpoints also have a base; it is the base of the local endpoint of the pair from whose connectivity check the peer reflexive endpoint was learned.

3.3.5. Prioritizing Endpoints

Each obtained endpoint is assigned a unique priority that MUST be a positive integer between 0 and 2**32 - 1. A peer SHOULD compute this priority using the formula in Section 3.3.5.1 and choose its parameters using the guidelines in Section 3.3.6. Using a different formula will most likely break the coordination in the connectivity checks, causing the protocol to take longer to converge.

3.3.5.1. Recommended Formula

The priority is based on a preference for each type of endpoint (host, peer reflexive, server reflexive and relayed) and a preference

for each of a peer's local IP addresses, in case it is multihomed. These two preferences are combined to compute the priority for an endpoint using the following formula (which is derived from the formula defined in [I-D.ietf-mmusic-ice], Section 4.1.2):

```
priority = (2**16)*(type preference) +
           IP address preference
```

The type preference MUST be an integer from 0 to 255 inclusive and represents the preference for the type of the endpoint. 255 is the highest preference, and 0 is the lowest. Setting the value to 0 means that endpoints of this type will only be used as a last resort. The type preference MUST be identical for all endpoints of the same type and MUST be different for endpoints of different types. The type preference for peer reflexive endpoints MUST be higher than that of server reflexive endpoints. This is because it is easier for an attacker to foist a bad server reflexive endpoint on a peer, than it is to do the same with peer reflexive endpoints.

The IP address preference MUST be an integer from 0 to 65535 inclusive. It represents a preference for the particular IP address from which the endpoint was obtained in case a peer is multihomed. 65535 represents the highest preference and 0 the lowest. When there is only a single IP address, this value SHOULD be set to 65535. If a peer is dual-stacked the IP address preference SHOULD be equal to the precedence value for IP addresses as described in [RFC3484].

3.3.6. Guidelines for Choosing Type and IP Address Preferences

The RECOMMENDED values for the type preference are 255 for host endpoints, 128 for peer reflexive endpoints, 64 for server reflexive endpoints, and 0 for relayed endpoints.

One criteria for the selection of the IP address preference values is IP address family. This protocol works with both IPv4 and IPv6. It also allows dual-stack hosts to prefer connections over IPv6, but to fall back to IPv4. Other criteria MAY be established as a matter of local optimization.

3.3.7. Eliminating Redundant Endpoints

After obtaining the endpoints, the peer eliminates redundant ones. An endpoint is redundant if its transport address equals that of another endpoint and its base equals the base of that other endpoint. Two endpoints that share the same transport address but have different bases are not considered redundant. The peer SHOULD eliminate the redundant candidate with the lower priority.

3.4. Initiating a Connection

To initiate a direct connection with another peer and to exchange endpoints, a new exchange type (ME_CONNECT) is defined. The communication between initiating peer and responding peer passes through the mediation server and therefore consists of multiple exchanges. Request and response between the peers are each composed of two distinct exchanges between the mediation server and the peers. This results in the following message flow:

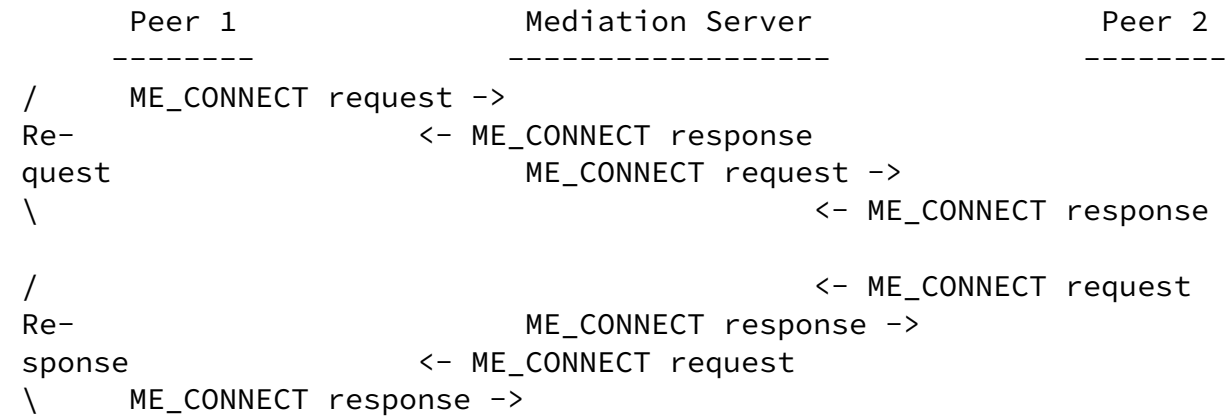


Figure 2: ME_CONNECT Exchanges

3.4.1. ME_CONNECT Exchange

The first payload included in a ME_CONNECT request is an IDp payload containing the ID of the other peer. All other payloads are notifications.

The first two notifications are ME_CONNECTID and ME_CONNECTKEY. ME_CONNECTID contains a randomly chosen value that is used to identify the current connection setup. This identifier is provided by the initiator and is sent back by the other peer in the reply in order to be able to distinguish concurrent ME_CONNECT exchanges initiated by both sides. Each peer also provides a randomly chosen

authenticate the connectivity checks.

If the requested peer is currently not online, that is, not connected to the mediation server, the mediation server MUST include a ME_CONNECT_FAILED error notification in its response. To prevent an initiator from constantly having to poll the other peer's online status, it MAY include a ME_CALLBACK notification in its request. This instructs the mediation server to notify the initiator as soon as the requested peer gets online.

To transmit the previously obtained endpoints, notification payloads of type ME_ENDPOINT are used. A ME_CONNECT request MUST include at least one such payload. Mediation servers MUST reply with a ME_CONNECT_FAILED if a request contains no endpoints.

Figure 3 shows a schematic overview of the ME_CONNECT exchange that is used to initiate a connection. Figure 4 shows the exchange used to indicate a failure.

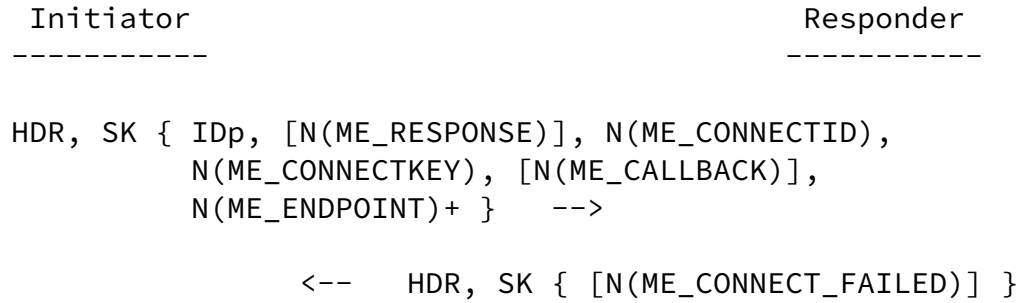


Figure 3: ME_CONNECT Exchange: Initiation

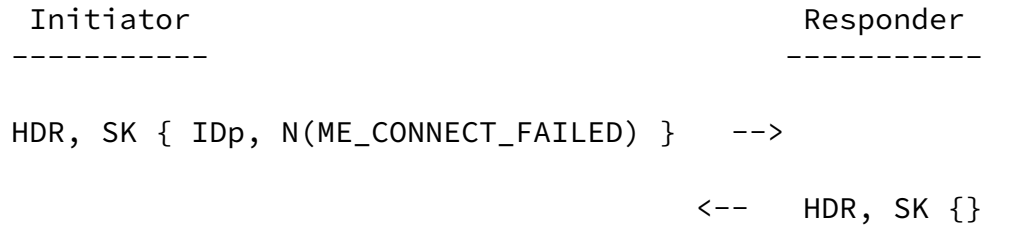


Figure 4: ME_CONNECT Exchange: Failure

On every ME_CONNECT request the mediation server checks whether the requested peer is connected to it. If this is the case, the mediation server forwards the data included in the request to the requested peer by initiating another ME_CONNECT exchange, thereby

replacing the IDp payload with the ID of the initiator. If the requested peer is not available the mediation server responds immediately with a ME_CONNECT_FAILED notification. If the initiator included a ME_CALLBACK notification in its request, the mediation server registers the requested ID. Once the requested peer connects, the mediation server notifies all waiting peers by initiating a ME_CONNECT exchange containing the peer ID of the requested peer and a ME_CALLBACK Notify payload, as shown in Figure 5. Afterwards, the mediation server removes the ID from the list of requested peers.

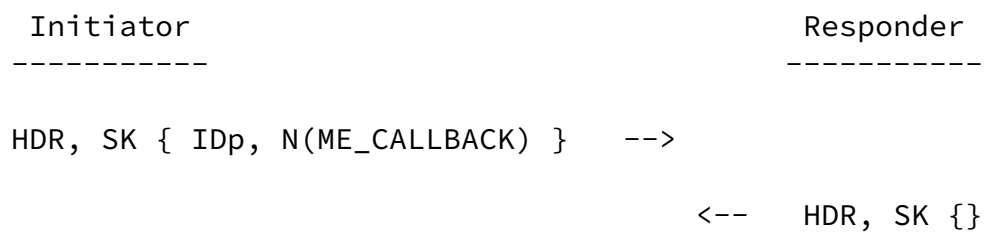


Figure 5: ME_CONNECT Exchange: Callback

3.4.2. Receiving a ME_CONNECT Request

Upon receipt of a ME_CONNECT request from the mediation server, a peer has to obtain endpoints itself. Actually the peer could have done that earlier, even before connecting to the mediation server, keeping the endpoints alive while waiting for incoming requests. The peer then assembles a ME_CONNECT request which contains its own endpoints, the ID of the other peer, and a randomly generated value for the ME_CONNECTKEY payload. It also includes the ME_CONNECTID payload from the request and a ME_RESPONSE Notify payload to mark this exchange as a response. This message is then sent to the mediation server which should confirm it with an empty response. If

the response contains a ME_CONNECT_FAILED notification, the other peer is not connected to the mediation server anymore. In this case the peer stops handling the request, otherwise, it proceeds with connectivity checks, as described beginning with Section 4.

In case a peer is unable to handle the request for a mediated connection - this could be due to missing configuration, local policy or other failures - it immediately responds with a ME_CONNECT_FAILED in the response to the ME_CONNECT request it received from the mediation server. If it later faces a condition that prevents it from responding to the request, it SHOULD initiate a ME_CONNECT exchange containing only an IDp and a ME_CONNECT_FAILED Notify payload. This notification is then forwarded to the initiating peer to inform it of this situation.

3.4.3. Receiving a ME_CONNECT Response

The initiator eventually gets a ME_CONNECT request from the mediation server containing the response from the other peer. It correlates the response with the previously sent request using the ID contained in the ME_CONNECTID payload. It extracts the endpoints and key provided by the responder and proceeds with connectivity checks, as described beginning with Section 4.

3.4.4. Timeout for the Overall Transaction

Since the whole transaction is split in four separate exchanges (see Figure 2) a timeout for the overall transaction is required. This timeout allows the initiator to act appropriately in case any of the three exchanges, in which it is not actively involved, fails. The nature of appropriate means is not defined by this specification, a peer might just restart the process, cancel it and log a message, or might take more sophisticated measures (like contacting an alternative mediation server). The timer controlling this timeout SHOULD be started right after the initial ME_CONNECT exchange finished successfully.

Since [RFC4306] does not exactly specify how retransmissions for IKEv2 messages have to be effected and does not define the time frame within which dead peers have to be detected, it becomes impossible to specify an exact timeout value. Therefore this document only

specifies that an overall timeout value MUST be configurable to allow it to be adapted to specific conditions. As a recommendation the timeout value SHOULD approximately amount to at least three times the maximum time it takes the initiating peer to conclude that the retransmission of an IKEv2 message has finally failed.

4. Building Endpoint Pairs

After receiving endpoints with a ME_CONNECT exchange, a peer builds a list of endpoint pairs. This is done by pairing each local endpoint with each remote endpoint (endpoints get only paired if they share the same IP address family). Then for each pair a priority is computed. The resulting list is then sorted in decreasing order of priorities. The formula used to compute this priority is as follows (it is basically the same formula as defined in [I-D.ietf-mmusic-ice], Section 5.7.2):

```
priority = (2**32) * MIN(pI, pR) +
                  2 * MAX(pI, pR) + (pI > pR ? 1 : 0)
```


where pI and pR denote the priorities of the initiator and the responder, respectively. MIN and MAX are functions that result in either the minimum or the maximum value of their parameters, respectively. The last term of the formula evaluates to 1 if pI is greater than pR or to 0 otherwise.

A peer cannot send messages directly from a reflexive endpoint, but only from its base. Since a peer generated pairs with both host endpoints and server reflexive endpoints as local endpoints, it's likely that there are duplicate entries in the list of pairs. Therefore, the peer MUST prune the list. This is done by removing a pair if the base of its local endpoint and the remote endpoint are identical to those of a pair higher up on the list.

After sorting and pruning the list, the pairs are numbered serially. This number serves as a message ID in connectivity checks. The result is a sequentially numbered, ordered list of endpoint pairs, called the checklist.

Each pair in the checklist has a specific state assigned to it that changes during the connectivity checks. Initially all pairs are in state Waiting. The possible states are as follows:

Waiting: No check has been performed yet for this pair. As soon as it becomes the highest priority Waiting pair on the checklist, a check can be performed.

In-Progress: A check has been sent for this pair, but the transaction is still in progress.

Succeeded: A check for this pair produced a successful result.

Failed: A check for this pair was done and it failed.

An implementation SHOULD limit the number of endpoints it accepts in a ME_CONNECT exchange as well as the number of pairs in a single checklist. This specification does not define what the limits are but the limits MUST be configurable, so that users can adjust the limits if a specific situation demands it. If more endpoints are received than the configured upper limit, the implementation SHOULD discard them according to their priority. The same procedure is RECOMMENDED for supernumerary pairs.

5. Connectivity Checks

Connectivity checks are done using unprotected INFORMATIONAL exchanges. The peers process the checklist sequentially and send a request from the local endpoint to the remote endpoint of each pair.

In addition to the checklist each peer maintains a FIFO queue, called the triggered check queue, which contains pairs for which checks are to be sent at the next available opportunity. A periodically firing timer T controls the generation of the checks. Whenever timer T fires, a peer first checks whether there are any elements in the triggered check queue. If so, it removes the first pair from it and initiates a connectivity check for that pair. Otherwise the peer sends a check for the topmost pair in the checklist which is in state Waiting. If no such pair exists the peer does nothing in this time slot. This process is illustrated in Figure 6. Once a check has been sent the state of the pair is set to In-Progress.

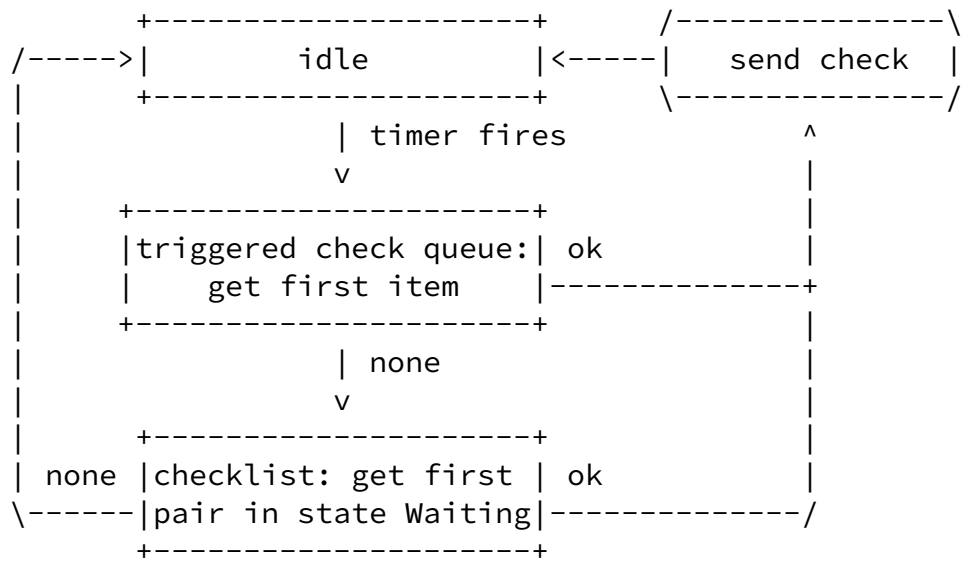


Figure 6: Sending Connectivity Checks

There is no RECOMMENDED setting for timer T specified in this document. But timer T MUST be configurable so that a user may change the setting to adjust to specific environments. There is a second timer called retransmission timer R which is started for each connectivity check request after it has been initially sent. Whenever timer R fires the request is retransmitted. This not done indefinitely, though. After a set number of retransmissions the connectivity check times out and the state of the pair is set to Failed. As with timer T, this specification does not restrict implementors on how to design these retransmissions. However, it is RECOMMENDED that a user may be able to configure how often and how long retransmissions are sent in order to improve the connectivity in specific situations.

5.1. Forming Connectivity Checks

Specially crafted unprotected INFORMATIONAL exchanges act as connectivity checks. The INFORMATIONAL request is formed as follows. The SPI fields in the IKE header are set to zero. The message ID is

set to the ID of the corresponding entry in the checklist. Three payloads follow the header. The first one is a ME_CONNECTID notification containing the value provided by the initiator that allows the recipient to locate the correct checklist. The next payload is a ME_ENDPOINT Notify payload that has all fields but the priority and the type set to zero. The priority field is set equal to the priority that would be assigned based on the formula in Section 3.3.5.1 to a peer reflexive endpoint. Hence, the type field is set to PEER_REFLEXIVE. To authenticate the message a ME_CONNECTAUTH notification is built and added, containing an SHA-1 hash of several parts of the message and the value of the appropriate ME_CONNECTKEY (see Section 5.1.1 for details). Request and response of a connectivity check are always authenticated with the same key, that of the responder. Thus a connectivity check from peer L to peer R (and its response) is authenticated with the key provided by R. Likewise, a connectivity check from R to L (and its response) is authenticated with the key provided by L.

To simplify things, the IKE messages used to do connectivity checks are always sent with a non-ESP marker in front of the IKE header, as defined in [RFC3948], even if the port numbers used are not 4500.

Figure 7 provides a schematic diagram of a connectivity check.

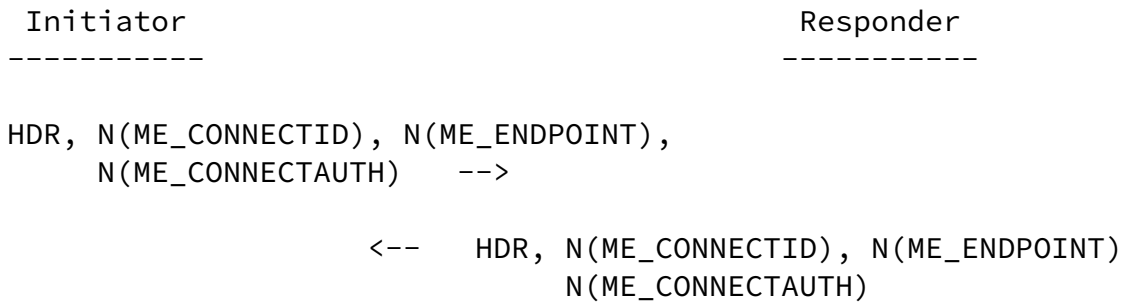


Figure 7: Connectivity Checks

5.1.1. ME_CONNECTAUTH

The formula used to compute the value of the ME_CONNECTAUTH Notify payload is:

```
auth = Hash(MID | ME_CONNECTID | ME_ENDPOINT | ME_CONNECTKEY)
```

where MID denotes the message ID in the IKE Header in network byte order and | indicates concatenation. Of each included Notify payload only the notification data is considered. The hash function used is SHA-1.

5.2. Responding to Connectivity Checks

After receiving a connectivity check request, a peer uses the value of the ME_CONNECTID payload to locate the correct checklist and the appropriate key. It verifies that the message is genuine, by computing the hash as the sender did and comparing the result with the content of the ME_CONNECTAUTH Notify payload. If either the checklist is not found or the verification fails, the peer MUST ignore the connectivity check request. Otherwise, it proceeds as follows. Refer to Figure 8 for an illustration of this process.

1. It checks whether the source address and port of the message are already included in the list of remote endpoints. If this is not the case, this represents a new peer reflexive endpoint. The priority of this endpoint is set to the priority noted in the ME_ENDPOINT payload of the request and it is then added to the list of remote endpoints.

2. A new pair is constructed setting the local endpoint to the one on which the request was received, and the remote endpoint to the one where the request came from (this may be the peer reflexive endpoint just learned). The priority of this pair is computed as usual.

3. If this pair is already in the checklist, further processing depends on the state of that pair.

    * If the pair is in waiting state, a check for it is enqueued into the triggered check queue.

    * If the state is In-Progress, retransmissions for the pending request will be cancelled, but the peer will wait the duration of the retransmission timeout for a response. If there is no answer the peer MUST schedule a new connectivity check for that pair, by enqueuing a check in the triggered check queue. The state of the pair is then changed to Waiting.

    * If the state of the pair is Failed, it is changed to Waiting and the peer MUST enqueue a new connectivity check for that pair in the triggered check queue.

* If the state is already Succeeded, nothing is done.

If the pair had not yet been included in the checklist, it is now inserted based on its priority. The ID is set to the number of pairs in the checklist plus one. The state is set to Waiting and a connectivity check is enqueued in the triggered check queue.

4. A response is then sent back. It includes the same ME_CONNECTID as the request, the ME_ENDPOINT is filled with the source endpoint from which the request was received - for relayed endpoints that are obtained using STUN, the source address is included in the REMOTE-ADDRESS attribute, if it was encapsulated in a Data Indication message, or it is the current active destination for the STUN relay session, otherwise - and the ME_CONNECTAUTH is built as in the request, using the appropriate key.

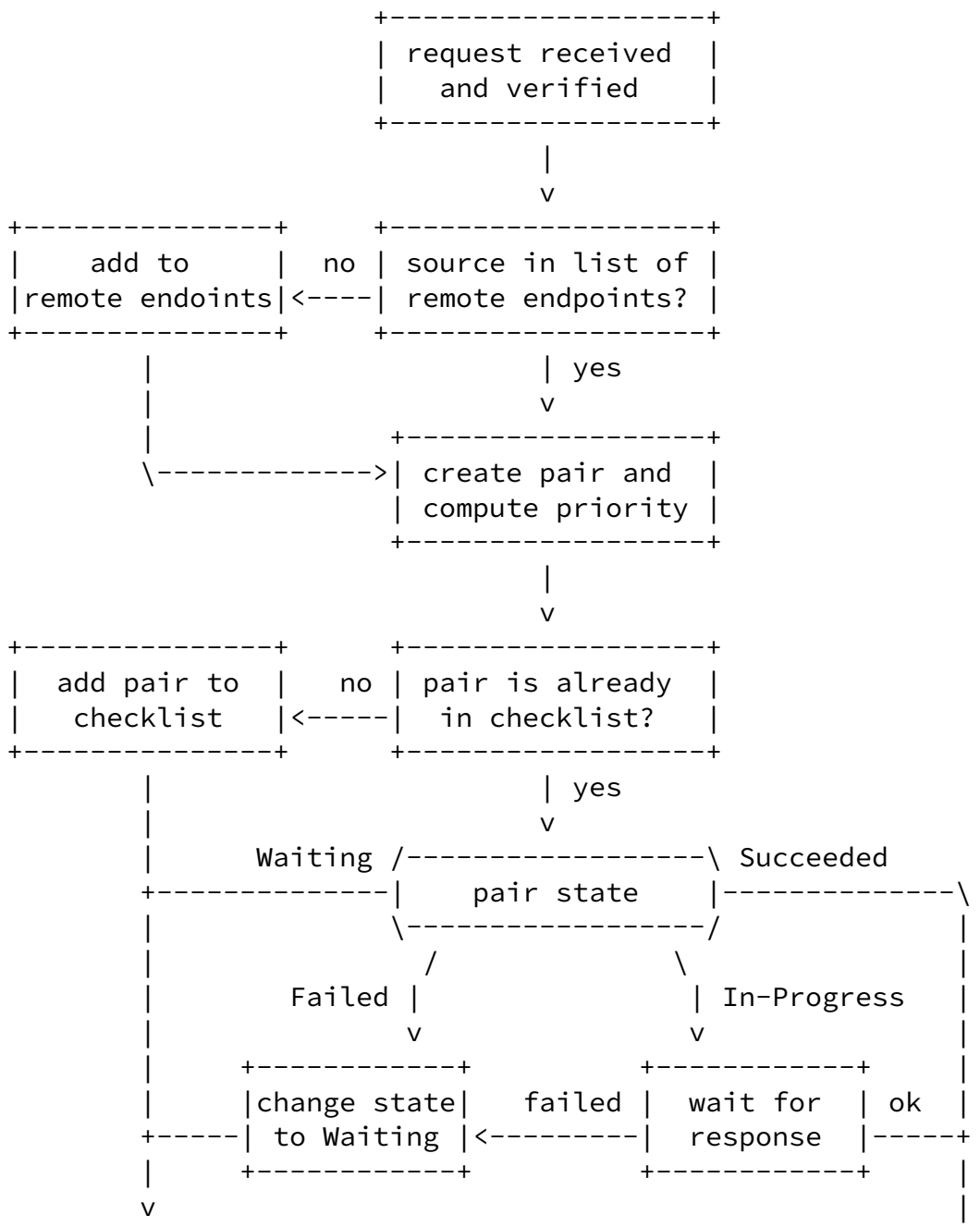

request received and verified
add to remote endoints
no
source in list of remote endpoints?
yes
create pair and compute priority
add pair to checklist
no
pair is already in checklist?
yes
Waiting
pair state
Succeeded
Failed
In-Progress
change state to Waiting
failed
wait for response
ok

Figure 8: Responding to Connectivity Checks

5.3. Processing Connectivity Checks

This section describes how responses to connectivity checks are processed. On receipt of a connectivity check response a peer correlates it to the corresponding pair using, first, the ME_CONNECTID to find the correct checklist and then the message ID to identify the pair. It MUST verify the authenticity of the check using the key provided by the other peer.

5.3.1. Failure Cases

If the peer either cannot find the checklist or cannot find the corresponding pair or if the verification of the check fails, it MUST ignore the check response.

Implementations MAY support receipt of ICMP errors for connectivity checks. If a connectivity check generates an ICMP error, a peer sets the state of the corresponding pair to Failed.

If a connectivity check times out, the peer also sets the state of the corresponding pair to Failed.

The peer MUST check that the source address and port of the response equals the remote endpoint of the pair, and the destination address and port of the response equals the base of the local endpoint of the pair. If either of these comparisons fails the state of the pair is set to Failed.

5.3.2. Success Cases

A connectivity check is considered a success, if the following are true:

- The source address and port of the response equal the remote endpoint of the pair.
- The destination address and port of the response match the base of the local endpoint of the pair.

After verifying that the check is successful, the peer checks the mapped endpoint that is returned in the ME_ENDPOINT Notify payload. If the endpoint does not match any of the local endpoints that the peer knows about, the mapped endpoint represents a new peer reflexive endpoint. The base of this endpoint is set equal to the base of the local endpoint of the pair the check was sent for. The priority is set equal to the value noted in the payload. This endpoint is then added to the list of local endpoints and a new pair is built as follows.

A new pair is constructed whose local endpoint equals the endpoint from the ME_ENDPOINT Notify payload as described in the preceding paragraph, and whose remote endpoint equals the destination address to which the request was sent. This pair is inserted into a second list called valid list, since it has been validated by a connectivity check. The valid pair may equal the pair that generated the check, may equal a different pair in the checklist, or may be a pair not currently in the checklist.

If the pair is not on the checklist, the priority is computed as usual. If the local endpoint is peer reflexive, its priority is equal to the priority field of the ME_ENDPOINT payload. The priority of the remote endpoint is looked up in the list of remote endpoints.

The state of the pair that generated the check is then set to Succeeded.

5.3.3. Stopping the Checks and Selecting the Endpoints

Once one or more connectivity checks have completed successfully, valid pairs are generated and added to the valid list. The

initiating peer lets the checks continue until some stopping criteria is met and then selects one pair from the valid list based on an evaluation criteria. The criteria for stopping the checks and for evaluating the valid pairs is entirely a matter of local optimization.

The responding peer does not stop the checks for a checklist until it receives an IKE_SA_INIT request that includes a ME_CONNECTID Notify payload containing the respective connect ID.

6. Mediated Connection

6.1. Initiating the Mediated Connection

After the initiating peer has selected a valid pair, it uses these endpoints to initiate an IKE_SA_INIT exchange with the other peer. Like the connectivity checks, the IKE traffic on mediated connections is sent with the non-ESP Marker prepended to the IKE header, as defined in [RFC3948]. Whether UDP encapsulation of ESP traffic is enabled on the mediated connection, is decided as usual, using NAT detection as defined in [RFC4306], Section 2.23.

In addition to all the default payloads in the IKE_SA_INIT exchange the initiating peer also includes a ME_CONNECTID Notify payload containing the appropriate connect ID.

7. Payload Formats

7.1. Identification Payload - Peer Identity

This payload, denoted IDp in this document, is used to exchange the identities of mediated peers. It is identical to the Identification Payloads defined in [RFC4306], Section 3.5.

The payload type for this Identification Payload (IDp) is TBD_BY_IANA.

7.2. Notify Messages - Error Types

7.2.1. ME_CONNECT_FAILED Notify Payload

This notification is used to signal that the attempt to mediate a connection with a peer has failed. It is used in the ME_CONNECT exchange request or response.

The Notify Message Type for ME_CONNECT_FAILED is TBD-BY-IANA. The Protocol ID and SPI Size fields are set to zero. There is no data associated with this Notify type.

7.3. Notify Messages - Status Types

7.3.1. ME_MEDIATION Notify Payload

This notification is included in the IKE_SA_INIT exchange of a mediation connection to indicate that both parties support this specification and want to establish a mediation connection.

The Notify Message Type for ME_MEDIATION is TBD-BY-IANA. The Protocol ID and SPI Size fields are set to zero. The notification data field MUST be left empty (zero-length) when sending, and its contents (if any) MUST be ignored when this notification is received. This allows the field to be used by future versions of this protocol.

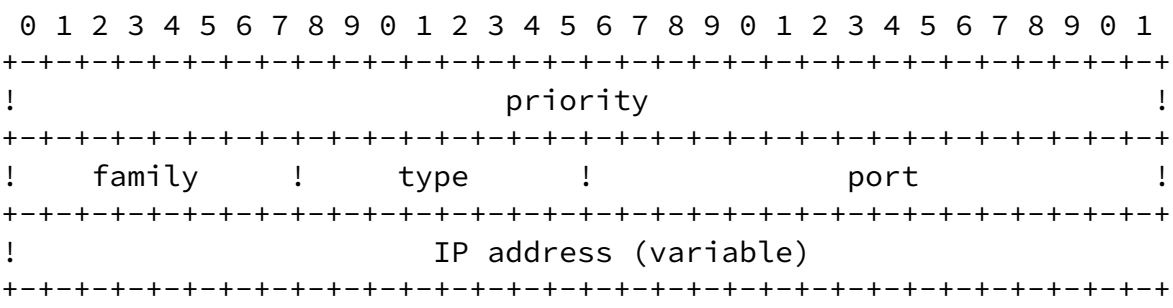
7.3.2. ME_ENDPOINT Notify Payloads

This notification is used to exchange endpoints.

The Notify Message Type for ME_ENDPOINT is TBD-BY-IANA. The Protocol ID and SPI Size fields are set to zero. The data associated with these Notify types starts with a four-octet long number denoting the endpoint's priority, followed by the eight bit long address family, and an eight bit long number indicating the type of the endpoint. The data further consists of a two-octet port number, which is finally followed by either a four-octet IPv4 address or a 16-octet

IPv6 address.

The following figure illustrates the format of the notification data of the ME_ENDPOINT payload:



The address family can take on the following values:

Name	Value
RESERVED	0
IPv4	1
IPv6	2

The following endpoint types are defined:

Name	Value
RESERVED	0
HOST	1
PEER_REFLEXIVE	2
SERVER_REFLEXIVE	3
RELAYED	4

This payload is also used to request endpoints from a mediation server and in connectivity checks. Refer to Section 3.3 and Section 5 for details.

7.3.3. ME_CALLBACK Notify Payload

This notification allows a peer to instruct the mediation server to send out a notification once a specific peer connects to it, if it was not available when the peer initially sent the ME_CONNECT. The

mediation server also includes a Notify payload of this type in the requested callback.

The Notify Message Type for ME_CONNECTID is TBD-BY-IANA. The Protocol ID and SPI Size fields are set to zero. There is no data associated with this Notify type.

7.3.4. ME_CONNECTID Notify Payload

This notification is used to exchange an identification number that uniquely identifies a direct connection attempt. The initiator provides this identifier in the ME_CONNECT exchange. It is then later used in the connectivity checks as well as in the IKE_SA_INIT request of the mediated connection. The randomly generated identifier MUST have a length of 4 to 16 octets.

The Notify Message Type for ME_CONNECTID is TBD-BY-IANA. The Protocol ID and SPI Size fields are set to zero. The data associated with this Notify type consists of random data of variable length.

7.3.5. ME_CONNECTKEY Notify Payload

This notification contains a symmetric key used in the MAC that authenticates the connectivity checks. The randomly generated key MUST be at least 16 octets long, but MAY have a length of up to 32 octets.

The Notify Message Type for ME_CONNECTKEY is TBD-BY-IANA. The Protocol ID and SPI Size fields are set to zero. The data associated with this Notify type consists of random data of variable length.

7.3.6. ME_CONNECTAUTH Notify Payload

This notification contains the message authentication code (MAC) in a connectivity check.

The Notify Message Type for ME_CONNECTAUTH is TBD-BY-IANA. The Protocol ID and SPI Size fields are set to zero. The data associated with this Notify type consists of a 20-octet SHA-1 digest.

7.3.7. ME_RESPONSE Notify Payload

This notification is used in ME_CONNECT exchanges to mark an exchange as a response. Since ME_CONNECT exchanges usually contain the same payloads, this Notify payload is required to distinguish between exchanges that serve as requests and exchanges that serve as responses. This is particularly important in the case of two peers trying to initiate a connection to each other at the same time.

The Notify Message Type for ME_RESPONSE is TBD-BY-IANA. The Protocol ID and SPI Size fields are set to zero. There is no data associated with this Notify type.

8. Security Considerations

8.1. Trusting the Mediation Servers

The peers must at least partially trust the mediation servers they use. Because the information that is passed to other peers is not encrypted in an end-to-end fashion the mediation server can observe all the exchanged endpoints. This could lead to the unwanted disclosure of private IP addresses and address ranges. Of course each peer can decide which endpoints it wants to share with other peers and hence with the mediation server.

9. IANA Considerations

This document does not create any new namespaces to be maintained by IANA, but it requires new values in namespaces that have been defined in the IKEv2 base specification [RFC4306].

This document defines a new IKEv2 exchange whose value is to be allocated from the "IKEv2 Exchange Types" namespace:

Exchange Type	Value
ME_CONNECT	TBD-BY-IANA (38..239)

These exchange is described in Section 3.4.1.

This document defines one new IKEv2 payload whose value is to be allocated from the "IKEv2 Payload Types" namespace:

Payload Type	Notation	Value
Identification - Peer	IDp	TBD-BY-IANA (49..127)

This payload is described in Section 7.

This document defines several new IKEv2 notifications whose values are to be allocated from the "IKEv2 Notify Message Types" namespace:

Notify Messages - Error Types	Value
ME_CONNECT_FAILED	TBD-BY-IANA (42..8191)

Notify Messages - Status Types	Value
ME_MEDIATION	TBD-BY-IANA (16406..40959)
ME_ENDPOINT	TBD-BY-IANA (16406..40959)
ME_CALLBACK	TBD-BY-IANA (16406..40959)
ME_CONNECTID	TBD-BY-IANA (16406..40959)
ME_CONNECTKEY	TBD-BY-IANA (16406..40959)
ME_CONNECTAUTH	TBD-BY-IANA (16406..40959)
ME_RESPONSE	TBD-BY-IANA (16406..40959)

These notification payloads are described in Section 7.

10. IAB Considerations

TODO?

11. Acknowledgements

The author would like to thank Martin Willi for his work on the introductory sections, and both him and Andreas Steffen for their comments and suggestions. A special thanks goes to Daniel Roethlisberger who worked with the author on an early revision of this specification.

12. References

12.1. Normative References

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

[RFC3948] Huttunen, A., Swander, B., Volpe, V., DiBurro, L., and M. Stenberg, "UDP Encapsulation of IPsec ESP Packets", RFC 3948, January 2005.

[RFC4306] Kaufman, C., "Internet Key Exchange (IKEv2) Protocol", RFC 4306, December 2005.

12.2. Informative References

[I-D.ietf-behave-rfc3489bis] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for (NAT) (STUN)", draft-ietf-behave-rfc3489bis-15 (work in progress), February 2008.

[I-D.ietf-behave-turn] Rosenberg, J., Mahy, R., and P. Matthews, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", draft-ietf-behave-turn-07 (work in progress), February 2008.

[I-D.ietf-mmusic-ice] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/ Answer Protocols", draft-ietf-mmusic-ice-19 (work in progress), October 2007.

[RFC3484] Draves, R., "Default Address Selection for Internet Protocol version 6 (IPv6)", RFC 3484, February 2003.

[RFC4555] Eronen, P., "IKEv2 Mobility and Multihoming Protocol (MOBIKE)",

RFC 4555, June 2006.

Editorial Comments

[anchor11] this allows later revisions of this specification to define a relay usage

Appendix A. Open Issues

A.1. Is the second ME_CONNECTKEY required?

The key provided by the responding peer is not really required. It's just that the MACs of the connectivity check requests from both peers would look the same, if only one key was used. On the other hand using just one key, would allow us to drop the ME_RESPONSE Notify payload from the ME_CONNECT response. Because the response from the other peer would not contain a ME_CONNECTKEY Notify it were clearly distinguishable from a ME_CONNECT request (see Appendix B.2).

A.2. Different NAT, Same Subnet

One problem arises when two hosts behind different NATs are attached to the same subnet. Is this just a configuration problem that we need or need not to document, or is it a main issue that we should provide a solution for (Virtual IP?).

A.3. Relaying Provided by the Mediation Server

As we provide the possibility to request a server reflexive endpoint from the mediation server, should we also provide relayed endpoints?

A.4. Compatibility/Synergy with MOBIKE

What happens in the following situations: Moving behind a NAT. Moving out of a NAT. External IP changes (NAT/no NAT). Multihomed host (active link goes down). If both peers still are connected to the mediation server, is there a possibility to update the endpoints? If a peer notices an address change with MOBIKE, should it update the endpoints? Should it send updated endpoints to the other peer? If the mediation server notices that our endpoint changed, does it send us a notice (other than through MOBIKE)?

Appendix B. Design Decisions

B.1. Two exchanges between mediation server and second peer

This document proposes the initiation of a connection to be composed of four exchanges: from one peer to the mediation server, from the mediation server to the other peer, and vice-versa. The two exchanges between the other peer and the mediation server could theoretically be combined in one exchange. This would be problematic in situations where the second peer first has to obtain endpoints before being able to answer the request. As this will take some time, the mediation server would most likely have retransmitted the request due to a timeout. And, if the peer wants to acquire a server reflexive endpoint from the mediation server a window size higher than one is required.

B.2. Why the ME_RESPONSE Notify payload is needed

It might seem that the ME_RESPONSE is rather superfluous. The ME_CONNECTID alone seems to be enough to distinguish requests from responses. A peer just has to maintain a list of issued ids and then simply compares the ME_CONNECTID of received ME_CONNECT messages with the items in this list. If an item matches, the received message is a response, otherwise, it is a request. Since the ME_CONNECTID is randomly chosen by the initiator, the ids contained in requests from

two different peers should never match. So, this should even work if two peers concurrently initiate a mediation with each other.

But there is a problem: What happens if a peer looses its state (e.g. due to a crash/restart) right after initiating a mediation, but immediately reconnects to the mediation server? Now, the ME_CONNECTID included in the answer from the other peer to the previously sent request is not included in the list of issued ids anymore. The answer thus looks exactly like a request for a new mediation. To avoid such a misunderstanding, peers have to be able to clearly distinguish requests from responses. Therefore, a ME_RESPONSE Notify payload is included in mediation responses.

Appendix C. Changelog

C.1. Changes from -.3 to -00

- Lots of clarifications and refinements. Major work on the introductory sections.

C.2. Changes from -.2 to -.3

- Refined some details after implementing the protocol.

C.3. Changes from -.1 to -.2

- Complete redesign to an ICE-like solution.

C.4. Changes from -.0 to -.1

- P2P_CONNECT for both sides
- "Endpoint..." terms expanded.

Author's Address

Tobias Brunner
University of Applied Sciences, Rapperswil
Oberseestrasse 10
Rapperswil, SG 8640
Switzerland

EMail: tobias.brunner@hsr.ch
URI: http://ita.hsr.ch

Full Copyright Statement

Intellectual Property