RFC 3706

A Traffic-Based Method of Detecting Dead Internet Key Exchange (IKE) Peers

(Czerny Andeas)

1. Introduction

When two peers communicate with IKE [2] and IPSec [3], the situation may arise in which connectivity between the two goes down unexpectedly. This situation can arise because of routing problems, one host rebooting, etc., and in such cases, there is often no way for IKE and IPSec to identify the loss of peer connectivity.

As such, the SAs can remain until their lifetimes naturally expire, resulting in a "black hole" situation where packets are tunneled to oblivion. It is often desirable to recognize black holes as soon as possible so that an entity can failover to a different peer quickly. Likewise, it is sometimes necessary to detect black holes to recover lost resources.

This problem of detecting a dead IKE peer has been addressed by proposals that require sending periodic HELLO/ACK messages to prove liveliness.
These schemes tend to be unidirectional (a HELLO only) called „heartbeat" or bidirectional (a HELLO/ACK pair) called „keepalive".

The problem with current heartbeat and keepalive proposals is their reliance upon their messages to be sent at regular intervals. In the implementation, this translates into managing some timer to service these message intervals.

Similarly, because rapid detection of the dead peer is often desired, these messages must be sent with some frequency, again translating into considerable overhead for message processing. In implementations and installations where managing large numbers of simultaneous IKE sessions is of concern, these regular heartbeats/keepalives prove to be infeasible.

To this end, a number of vendors have implemented their own approach to detect peer liveliness without needing to send messages at regular intervals. This informational document describes the curre practice of those implementations. This scheme, called Dead Peer Detection (DPD), relies on IKE Notify messages to query the liveliness of an IKE peer.

2.Keepalives and Heartbeats

2.1 Keepalives:

Consider a keepalives scheme in which peer A and peer B require regular acknowledgements of each other's liveliness. The messages are exchanged by means of an authenticated notify payload.

The two peers must agree upon the interval at which keepalives are sent, meaning that some negotiation is required during Phase 1. For any prompt failover to be possible, the keepalives must also be sent at rather frequent intervals -- around 10 seconds or so. In this hypothetical keepalives scenario, peers A and B agree to exchange keepalives every 10 seconds.

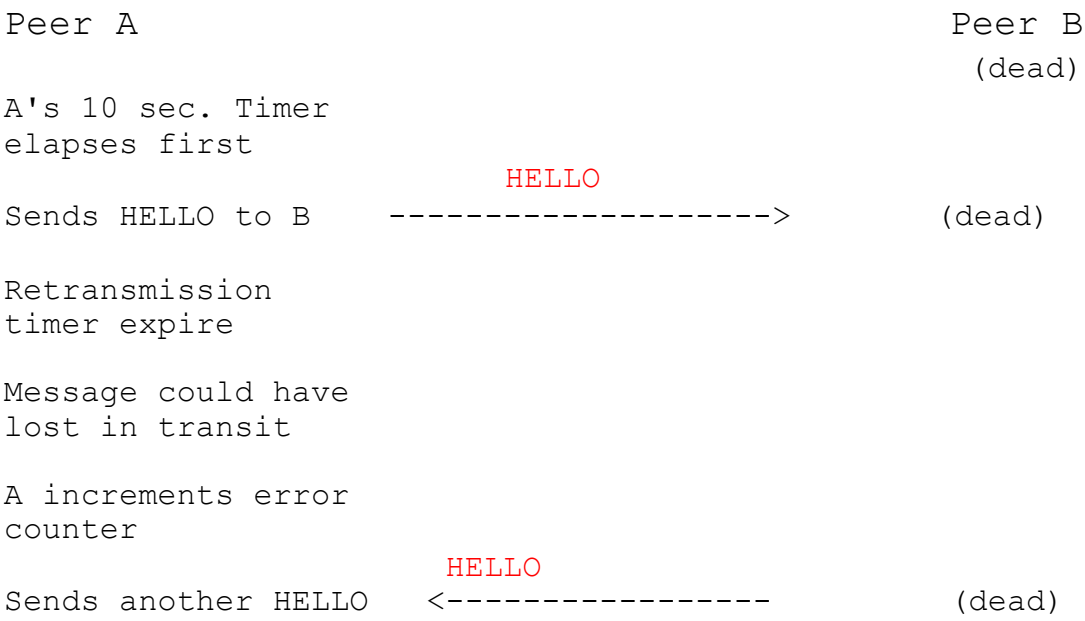
Essentially, every 10 seconds, one peer must send a HELLO to the other. This HELLO serves as proof of liveliness for the sending entity. In turn, the other peer must acknowledge each keepalive HELLO. If the 10 seconds elapse, and one side has not received a HELLO, it will send the HELLO message itself, using the peer's ACK as proof of liveliness.

Receipt of either a HELLO or ACK causes an entity's keepalive timer to reset. Failure to receive an ACK in a certain period of time signals an error. A clarification is presented below:

Scenario 1

Peer A		Peer B
A's 10 sec. Timer elapses first		
	HELLO	
Sends HELLO to B	---------->	Receives HELLO
		Acknowledges A's liveiness
		Resets keepalive timer
	ACK	
Receives ACK as proof of B's liveliness	<----------	Send ACK
Reset keepalive timer		

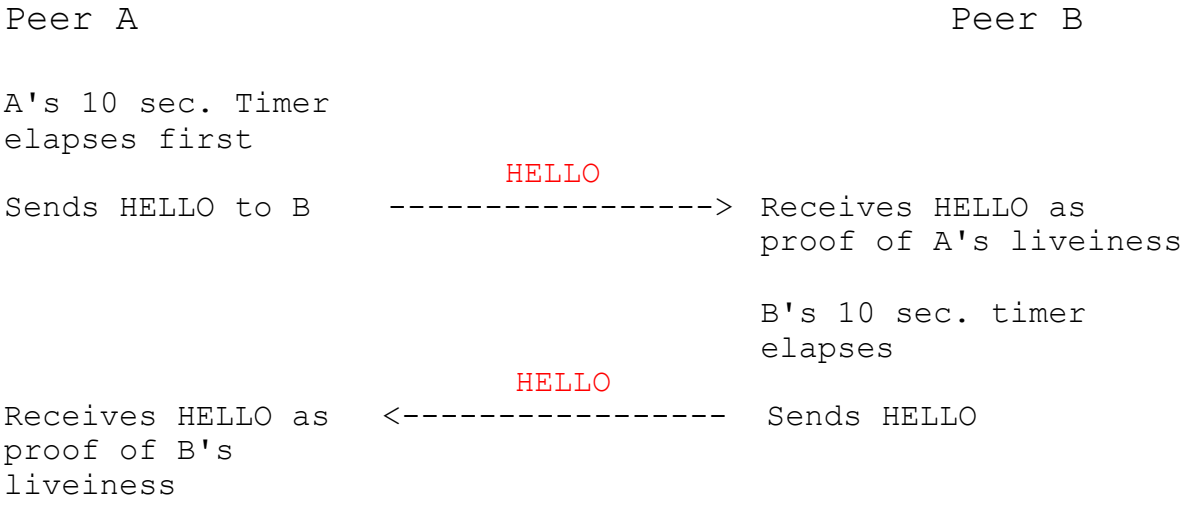
Scenario 2



2.2 Heartbeats

By contrast, consider a proof-of-liveliness scheme involving unidirectional (unacknowledged) messages. An entity interested in its peer's liveliness would rely on the peer itself to send periodic messages demonstrating liveliness. In such a scheme, the message exchange might look like this.

Scenario 3



Scenario 4

```
Peer A                                                  Peer B
                                                        (dead)

A's 10 sec. Timer
elapses first
                              HELLO
Sends HELLO to B     ------------------->     (dead)

...

Assumes B is dead
```

3.DPD Protocol

In a DPD Protocol each peer is free to request proof of liveliness when it needs it, and the asynchronous property allows fewer messages to be sent. Another good idea is, to use IPSec traffic as the proof of liveliness. So as long as both peer has outbound traffic, no other methode is necerssary. Furthermore, knowledge of the peer's liveliness is only interesting if there is any traffic to be sent.

The decission about when to initiate a DPD exchange is implementation specific. So each peer can define its own „worry metric", the time how long a peer is waiting until it sends a HELLO message to the other peer. And the peers DPD state is largely independent of the other's.

3.1 **Message exchanges**

Bothe peers of an IKE session must send the DPD vendor ID before DPD exchange can begin

```
 0 1  2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
!                               !M!M!
!      HASHED_VENDOR_ID         !J!N!
!                               !R!R!
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

MJR and MNR correspond to the current major and minor version of this protocol.

```
Peer A                                              Peer B

NOTIFY(R-U-THERE)      -------------->

                       <---------------    NOTIFY(R-U-THERE-ACK)
```

The DPD exchange is a bidirectional message and bothe are simply ISAKMP Notify payloads.
(Internet Security Association and Key Management Protocol)

Notify	Message Value
R-U-THERE	36136
R-U-THERE-ACK	36137

A peer must keep track of the state of a given DPD exchange and retranskit R-U-THERE queries when it fails to receive an a R-U-THERE-ACK. If after a number of messages no ACK is returning, the peer deletes the IPSec and IKE Sas to the other peer.

3.2 **Message format**

The R-U-There message must have the following form.

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
! Next Payload  !   RESERVED    !         Payload Length        !
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
!              Domain of Interpretation  (DOI)                 !
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
!  Protocol-ID  !    SPI Size   !      Notify Message Type      !
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
!                                                             !
~                Security Parameter Index (SPI)               ~
!                                                             !
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
!                   Notification Data                         !
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

As this message is an ISAKMP NOTIFY, the Next Payload, RESERVED, and Payload Length fields should be set accordingly.

- Domain of Interpretation (4 octets) - SHOULD be set to IPSEC-DOI.

- Protocol ID (1 octet) - MUST be set to the protocol ID for ISAKMP.

- SPI Size (1 octet) - SHOULD be set to sixteen (16), the length of two octet-sized ISAKMP cookies.

- Notify Message Type (2 octets) - MUST be set to R-U-THERE.

- Security Parameter Index (16 octets) - SHOULD be set to the cookies of the Initiator and Responder of the IKE SA (in that order).

- Notification Data (4 octets) - MUST be set to the sequence number corresponding to this message.

The format of the R-U-THERE-ACK message is the same, with the exception that the Notify Message Type MUST be set to R-U-THERE-ACK. Again, the Notification Data MUST be sent to the sequence number corresponding to the received R-U-THERE message.

3.3 **Implementation suggestion**

The liveliness of a peer is only questionable when no traffi is exchange, so a viable implementation might begin by monitoring idleness. Also the peer's liveliness is only important when there is any outbound traffic to be sent. A peer should only initiate a DPD exchange if outbound IPSec traffic was sent, but no inbound IPSec packets was received. So a complete DPD exchange will serve as proof of liveliness untill the nect idle period.

3.4 **DPD vs keepalive/heartbeats**

PDP has got a performance benefit, because it is not necssesary to sent regular messages to the other peer. So the number of IKE messages to be sent and processed is reduced. Another benefit is that DPD needs only 1 timer, dring keepalive/heartbeats needs 1 timer for the periodic sent od the HELLO message, and 1 elapse timer.

4. Resistance to replay attack and false proof of liveliness

4.1 **Sequence number in DPD messages**

Every peer has his own sequence number, that increments by 1 after sending a R-U-THERE message. A responder to an R-U-THERE message must send an R-U-THERE-ACK with the same sequence number. The initial sender reject the R-U-THERE-ACK if the sequence number fails to match the one sent with the R-U-THERE message. Additional both should check the validity of the initiator and responder cookies in the SPI field of the payload.

4.2 **Selection an maintenances of Sequence Numbers**

Both DPD peers can initiate a DPD exchange, but each peer must maintain its own sequence number. The first R-U-THERE message sent in a session, must be a randomly chosen number. To prevent an overflow, the high-bit of the sequence number initially should be set to zero. Its also beneficial if the sequence numbers reset at the expiry of the IKE SA.

4.3 **Benefit of sequence numbers**

Sequence numbers help to detecting replayed messages. So if someone starts a „man in the middle“ attack, its only necessary to decrypt the message and proof the sequence number. But for replayed messages the peer don't have to build, encrypt and send an ACK.

The sequence number is also an extra assurance of the peer's liveliness. As long as the sequence number increases, the peer must be alive.

5. List of literature

- RFC 3706 (http://www.faqs.org/rfcs/)