Network Working Group	F. L. Templin, Ed.
Internet-Draft	Boeing Research & Technology
Intended status: Standards Track	November 18, 2011
Expires: May 21, 2012	

The Subnetwork Encapsulation and Adaptation Layer (SEAL)

draft-templin-intarea-seal-38.txt

Abstract

For the purpose of this document, a subnetwork is defined as a virtual topology configured over a connected IP network routing region and bounded by encapsulating border nodes. These virtual topologies are manifested by tunnels that may span multiple IP and/or sub-IP layer forwarding hops, and can introduce failure modes due to packet duplication and/or links with diverse Maximum Transmission Units (MTUs). This document specifies a Subnetwork Encapsulation and Adaptation Layer (SEAL) that accommodates such virtual topologies over diverse underlying link technologies.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

As Internet technology and communication has grown and matured, many techniques have developed that use virtual topologies (including tunnels of one form or another) over an actual network that supports the Internet Protocol (IP) [RFC0791][RFC2460]. Those virtual topologies have elements that appear as one hop in the virtual topology, but are actually multiple IP or sub-IP layer hops. These multiple hops often have quite diverse properties that are often not even visible to the endpoints of the virtual hop. This introduces failure modes that are not dealt with well in current approaches.

The use of IP encapsulation (also known as "tunneling") has long been considered as the means for creating such virtual topologies. However, the insertion of an outer IP header reduces the effective path MTU visible to the inner network layer. When IPv4 is used, this reduced MTU can be accommodated through the use of IPv4 fragmentation, but unmitigated in-the-network fragmentation has been found to be harmful through operational experience and studies conducted over the course of many years [FRAG][FOLK][RFC4963]. Additionally, classical path MTU discovery [RFC1191] has known operational issues that are exacerbated by in-the-network tunnels [RFC2923][RFC4459]. The following subsections present further details on the motivation and approach for addressing these issues.

1.1. Motivation

Before discussing the approach, it is necessary to first understand the problems. In both the Internet and private-use networks today, IPv4 is ubiquitously deployed as the Layer 3 protocol. The primary functions of IPv4 are to provide for routing, addressing, and a fragmentation and reassembly capability used to accommodate links with diverse MTUs.

While it is well known that the IPv4 address space is rapidly becoming depleted, there is a lesser-known but growing consensus that other IPv4 protocol limitations have already or may soon become problematic.

First, the IPv4 header Identification field is only 16 bits in length, meaning that at most 2^16 unique packets with the same (source, destination, protocol)-tuple may be active in the Internet at a given time [I-D.ietf-intarea-ipv4-id-update]. Due to the escalating deployment of high-speed links, however, this number has become too small by several orders of magnitude for high data rate packet sources such as tunnel endpoints [RFC4963]. Furthermore, there are many well-known limitations pertaining to IPv4 fragmentation and reassembly – even to the point that it has been deemed “harmful” in both classic and modern-day studies (see above). In particular, IPv4 fragmentation raises issues ranging from minor annoyances (e.g., in-the-network router fragmentation [RFC1981]) to the potential for major integrity issues (e.g., mis-association of the fragments of multiple IP packets during reassembly [RFC4963]).

As a result of these perceived limitations, a fragmentation-avoiding technique for discovering the MTU of the forward path from a source to a destination node was devised through the deliberations of the Path MTU Discovery Working Group (PMTUDWG) during the late 1980’s through early 1990’s (see Appendix D). In this method, the source node provides explicit instructions to routers in the path to discard the packet and return an ICMP error message if an MTU restriction is encountered.

However, this approach has several serious shortcomings that lead to an overall “brittleness” [RFC2923].
In particular, site border routers in the Internet have been known to discard ICMP error messages coming from the outside world. This is due in large part to the fact that malicious spoofing of error messages in the Internet is trivial since there is no way to authenticate the source of the messages [RFC5927]. Furthermore, when a source node that requires ICMP error message feedback when a packet is dropped due to an MTU restriction does not receive the messages, a path MTU-related black hole occurs. This means that the source will continue to send packets that are too large and never receive an indication from the network that they are being discarded. This behavior has been confirmed through documented studies showing clear evidence of path MTU discovery failures in the Internet today [TBIT][WAND][SIGCOMM].
The issues with both IPv4 fragmentation and this “classical” method of path MTU discovery are exacerbated further when IP tunneling is used [RFC4459]. For example, an ingress tunnel endpoint (ITE) may be required to forward encapsulated packets into the subnetwork on behalf of hundreds, thousands, or even more original sources within the end site that it serves. If the ITE allows IPv4 fragmentation on the encapsulated packets, persistent fragmentation could lead to undetected data corruption due to Identification field wrapping. If the ITE instead uses classical IPv4 path MTU discovery, it must rely on ICMP error messages coming from the subnetwork that may be suspect, subject to loss due to filtering middleboxes, or insufficiently provisioned for translation into error messages to be returned to the original sources.
Although recent works have led to the development of a robust end-to-end MTU determination scheme [RFC4821], they do not excuse tunnels from delivering path MTU discovery feedback when packets are lost due to size restrictions. Moreover, in current practice existing tunneling protocols mask the MTU issues by selecting a "lowest common denominator" MTU that may be much smaller than necessary for most paths and difficult to change at a later date. Therefore, a new approach to accommodate tunnels over links with diverse MTUs is necessary.

1.2. Approach

For the purpose of this document, a subnetwork is defined as a virtual topology configured over a connected network routing region and bounded by encapsulating border nodes. Example connected network routing regions include Mobile Ad hoc Networks (MANETs), enterprise networks and the global public Internet itself. Subnetwork border nodes forward unicast and multicast packets over the virtual topology across multiple IP and/or sub-IP layer forwarding hops that may introduce packet duplication and/or traverse links with diverse Maximum Transmission Units (MTUs).
This document introduces a Subnetwork Encapsulation and Adaptation Layer (SEAL) for tunneling network layer protocols (e.g., IPv4, IPv6, OSI, etc.) over IP subnetworks that connect Ingress and Egress Tunnel

Endpoints (ITEs/ETEs) of border nodes. It provides a modular specification designed to be tailored to specific associated tunneling protocols. A transport-mode of operation is also possible, and described in Appendix C.

SEAL provides a mid-layer encapsulation that accommodates links with diverse MTUs and allows routers in the subnetwork to perform efficient duplicate packet detection. The encapsulation further ensures data origin authentication, packet header integrity and anti-replay.

SEAL treats tunnels that traverse the subnetwork as ordinary links that must support network layer services. Moreover, SEAL provides dynamic mechanisms to ensure a maximal path MTU over the tunnel. This is in contrast to static approaches which avoid MTU issues by selecting a lowest common denominator MTU value that may be overly conservative for the vast majority of tunnel paths and difficult to change even when larger MTUs become available.

The following sections provide the SEAL normative specifications, while the appendices present non-normative additional considerations.

2. Terminology and Requirements

The following terms are defined within the scope of this document:

subnetwork
a virtual topology configured over a connected network routing region and bounded by encapsulating border nodes.

Ingress Tunnel Endpoint (ITE)
a virtual interface over which an encapsulating border node (host or router) sends encapsulated packets into the subnetwork.

Egress Tunnel Endpoint (ETE)
a virtual interface over which an encapsulating border node (host or router) receives encapsulated packets from the subnetwork.

ETE Link Path
a subnetwork path from an ITE to an ETE beginning with an underlying link of the ITE as the first hop.

inner packet
an unencapsulated network layer protocol packet (e.g., IPv6 [RFC2460], IPv4 [RFC0791], OSI/CLNP [RFC1070], etc.) before any outer encapsulations are added. Internet protocol numbers that identify inner packets are found in the IANA Internet Protocol

registry [RFC3232]. SEAL protocol packets that incur an additional layer of SEAL encapsulation are also considered inner packets.

outer IP packet
a packet resulting from adding an outer IP header (and possibly other outer headers) to a SEAL-encapsulated inner packet.

packet-in-error
the leading portion of an invoking data packet encapsulated in the body of an error control message (e.g., an ICMPv4 [RFC0792] error message, an ICMPv6 [RFC4443] error message, etc.).

Packet Too Big (PTB)
a control plane message indicating an MTU restriction (e.g., an ICMPv6 "Packet Too Big" message [RFC4443], an ICMPv4 "Fragmentation Needed" message [RFC0792], etc.).

IP
used to generically refer to either Internet Protocol (IP) version, i.e., IPv4 or IPv6.

The following abbreviations correspond to terms used within this document and/or elsewhere in common Internetworking nomenclature:

*DF - the IPv4 header "Don't Fragment" flag [RFC0791]

*ETE - Egress Tunnel Endpoint

*HLEN - the length of the SEAL header plus outer headers

*ICV - Integrity Check Vector

*ITE - Ingress Tunnel Endpoint

*MTU - Maximum Transmission Unit

*SCMP - the SEAL Control Message Protocol

*SDU - SCMP Destination Unreachable message

*SNA - SCMP Neighbor Advertisement message

*SNS - SCMP Neighbor Solicitation message

*SPP - SCMP Parameter Problem message

*SPTB - SCMP Packet Too Big message

*SEAL - Subnetwork Encapsulation and Adaptation Layer

*SEAL_PORT - a transport-layer service port number used for SEAL

*SEAL_PROTO - an IP protocol number used for SEAL

*TE - Tunnel Endpoint (i.e., either ingress or egress)

*VET - Virtual Enterprise Traversal

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]. When used in lower case (e.g., must, must not, etc.), these words MUST NOT be interpreted as described in [RFC2119], but are rather interpreted as they would be in common English.

3. Applicability Statement

SEAL was originally motivated by the specific case of subnetwork abstraction for Mobile Ad hoc Networks (MANETs), however the domain of applicability also extends to subnetwork abstractions over enterprise networks, ISP networks, SOHO networks, the global public Internet itself, and any other connected network routing region. SEAL, along with the Virtual Enterprise Traversal (VET) [I-D.templin-intarea-vet] tunnel virtual interface abstraction, are the functional building blocks for the Internet Routing Overlay Network (IRON) [I-D.templin-ironbis] and Routing and Addressing in Networks with Global Enterprise Recursion (RANGER) [RFC5720][RFC6139] architectures.

SEAL provides a network sublayer for encapsulation of an inner network layer packet within outer encapsulating headers. SEAL can also be used as a sublayer within a transport layer protocol data payload, where transport layer encapsulation is typically used for Network Address Translator (NAT) traversal as well as operation over subnetworks that give preferential treatment to certain "core" Internet protocols (e.g., TCP, UDP, etc.). The SEAL header is processed the same as for IPv6 extension headers, i.e., it is not part of the outer IP header but rather allows for the creation of an arbitrarily extensible chain of headers in the same way that IPv6 does.

To accommodate MTU diversity, the Egress Tunnel Endpoint (ETE) acts as a passive observer that simply informs the Ingress Tunnel Endpoint (ITE) of any packet size limitations. This allows the ITE to return appropriate path MTU discovery feedback even if the network path between the ITE and ETE filters ICMP messages.

SEAL further ensures data origin authentication, packet header integrity, and anti-replay. The SEAL framework is therefore similar to the IP Security (IPsec) Authentication Header (AH) [RFC4301][RFC4302], however it provides only minimal hop-by-hop authenticating services along a path while leaving full data integrity, authentication and confidentiality services as an end-to-end consideration. While SEAL performs data origin authentication, the origin site must also perform the necessary ingress filtering in order to provide full source address verification [I-D.ietf-savi-framework].

4. SEAL Specification

The following sections specify the operation of SEAL:

4.1. VET Interface Model

SEAL is an encapsulation sublayer used within VET non-broadcast, multiple access (NBMA) tunnel virtual interfaces. Each VET interface is configured over one or more underlying interfaces attached to subnetwork links. The VET interface connects an ITE to one or more ETE "neighbors" via tunneling across an underlying subnetwork, where tunnel neighbor relationship may be either unidirectional or bidirectional.
A unidirectional tunnel neighbor relationship allows the near end ITE to send data packets forward to the far end ETE, while the ETE only returns control messages when necessary. A bidirectional tunnel neighbor relationship is one over which both TEs can exchange both data and control messages.
Implications of the VET unidirectional and bidirectional models are discussed in [I-D.templin-intarea-vet].

4.2. SEAL Model of Operation

SEAL-enabled ITEs encapsulate each inner packet in a SEAL header, any outer header encapsulations, and in some instances a SEAL trailer as shown in Figure 1:

```
                             +--------------------+
                             ~   outer IP header  ~
                             +--------------------+
                             ~  other outer hdrs  ~
                             +--------------------+
                             ~    SEAL Header     ~
+--------------------+       +--------------------+
|                    |  -->  |                    |
~        Inner       ~  -->  ~        Inner       ~
~       Packet       ~  -->  ~       Packet       ~
|                    |  -->  |                    |
+--------------------+       +--------------------+
                             |    SEAL Trailer    |
                             +--------------------+
```

The ITE inserts the SEAL header according to the specific tunneling protocol. For simple encapsulation of an inner network layer packet within an outer IP header (e.g., [RFC1070][RFC2003][RFC2473][RFC4213], etc.), the ITE inserts the SEAL header between the inner packet and outer IP headers as: IP/SEAL/{inner packet}.

For encapsulations over transports such as UDP, the ITE inserts the SEAL header between the outer transport layer header and the inner packet, e.g., as IP/UDP/SEAL/{inner packet} (similar to [RFC4380]). In that case, the UDP header is seen as an "other outer header" as depicted in Figure 1.

When necessary, the ITE also appends a SEAL trailer at the end of the SEAL packet. In that case, the trailer is added after the final byte of the encapsulated packet.

SEAL supports both "nested" tunneling and "re-encapsulating" tunneling. Nested tunneling occurs when a first tunnel is encapsulated within a second tunnel, which may then further be encapsulated within additional tunnels. Nested tunneling can be useful, and stands in contrast to "recursive" tunneling which is an anomalous condition incurred due to misconfiguration or a routing loop. Considerations for nested tunneling are discussed in Section 4 of [RFC2473].

Re-encapsulating tunneling occurs when a packet arrives at a first ETE, which then acts as an ITE to re-encapsulate and forward the packet to a second ETE connected to the same subnetwork. In that case each ITE/ETE transition represents a segment of a bridged path between the ITE nearest the source and the ETE nearest the destination. Combinations of nested and re-encapsulating tunneling are also naturally supported by SEAL.

The SEAL ITE considers each {underlying interface, IP address} pair as the ingress attachment point to a subnetwork link path to the ETE. The ITE therefore maintains path MTU state on a per ETE link path basis, although it may instead maintain only the lowest-common-denominator values for all of the ETE's link paths in order to reduce state.

Finally, the SEAL ITE ensures that the inner network layer protocol will see a minimum MTU of 1280 bytes over each ETE link path regardless of the outer network layer protocol version, i.e., even if a small amount of fragmentation and reassembly are necessary.

4.3. SEAL Header and Trailer Format

The SEAL header is formatted as follows:

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |VER|C|P|R|T|U|Z|     NEXTHDR   |    PREFLEN    | LINK_ID |LEVEL|
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                             PKT_ID                            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                              ICV1                             |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   ~                      PREFIX (when present)                    ~
   . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

VER (2)

a 2-bit version field. This document specifies Version 0 of the SEAL protocol, i.e., the VER field encodes the value 0.

C (1)
the "Control/Data" bit. Set to 1 by the ITE in SEAL Control Message Protocol (SCMP) control messages, and set to 0 in ordinary data packets.

P (1)
the "Prefix Included" bit. Set to 1 if the header includes a Prefix Field. Used for SCMP messages that do not include a packet-in-error (see: [I-D.templin-intarea-vet]), and for NULL SEAL data packets used as probes (see: Section 4.4.6).

R (1)
the "Redirects Permitted" bit. For data packets, set to 1 by the ITE to inform the ETE that the source is accepting Redirects (see: [I-D.templin-intarea-vet]).

T (1)
the "Trailer Included" bit. Set to 1 if the ITE was obliged to include a trailer.

U (1)
the "Unfragmented Packet" bit. Set to 1 by the ITE in SEAL data packets for which it wishes to receive an explicit acknowledgement from the ETE if the packet arrives unfragmented.

Z (1)
the "Reserved" bit. Must be set to 0 for this version of the SEAL specification.

NEXTHDR (8) an 8-bit field that encodes the next header Internet Protocol number the same as for the IPv4 protocol and IPv6 next header fields.

PREFLEN (8) an 8-bit field that encodes the length of the prefix to be applied to the source address of the inner packets (when P==0) or the prefix included in the PREFIX field (when P==1).

LINK_ID (5)
a 5-bit link identification value, set to a unique value by the ITE for each underlying link as the first hop of a path over which it will send encapsulated packets to ETEs. Up to 32 ETE link paths are therefore supported for each ETE.

LEVEL (3)
a 3-bit nesting level; use to limit the number of tunnel nesting levels. Set to an integer value up to 7 in the innermost

SEAL encapsulation, and decremented by 1 for each successive additional SEAL encapsulation nesting level. Up to 8 levels of nesting are therefore supported.

PKT_ID (32)
a 32-bit per-packet identification field. Set to a monotonically-incrementing 32-bit value for each SEAL packet transmitted to this ETE, beginning with 0.

ICV1 (32)
a 32-bit header integrity check value that covers the leading 128 bytes of the packet beginning with the SEAL header. The value 128 is chosen so that at least the SEAL header as well as the inner packet network and transport layer headers are covered by the integrity check.

PREFIX (variable)
a variable-length string of bytes; present only when P==1. The field length is determined by calculating Len=(Ceiling(PREFLEN / 32) * 4). For example, if PREFLEN==63, the field is 8 bytes in length and encodes the leading 63 bits of the inner network layer prefix beginning with the most significant bit.

When T==1, SEAL encapsulation also includes a trailer formatted as follows:

```
     0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                              ICV2                             |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

ICV2 (32)
a 32-bit packet integrity check value. Present only when T==1, and covers the remaining length of the encapsulated packet beyond the leading 128 bytes (i.e., the remaining portion that was not covered by ICV1). Added as a trailing 32 bit field following the final byte of the encapsulated SEAL packet and used to detect reassembly misassociations. Need not be aligned on an even byte boundary.

4.4. ITE Specification

4.4.1. Tunnel Interface MTU

The tunnel interface must present a constant MTU value to the inner network layer as the size for admission of inner packets into the interface. Since VET NBMA tunnel virtual interfaces may support a large set of ETE link paths that accept widely varying maximum packet sizes, however, a number of factors should be taken into consideration when selecting a tunnel interface MTU.

Due to the ubiquitous deployment of standard Ethernet and similar networking gear, the nominal Internet cell size has become 1500 bytes; this is the de facto size that end systems have come to expect will either be delivered by the network without loss due to an MTU restriction on the path or a suitable ICMP Packet Too Big (PTB) message returned. When large packets sent by end systems incur additional encapsulation at an ITE, however, they may be dropped silently within the tunnel since the network may not always deliver the necessary PTBs [RFC2923].

The ITE should therefore set a tunnel interface MTU of at least 1500 bytes plus extra room to accommodate any additional encapsulations that may occur on the path from the original source. The ITE can also set smaller MTU values; however, care must be taken not to set so small a value that original sources would experience an MTU underflow. In particular, IPv6 sources must see a minimum path MTU of 1280 bytes, and IPv4 sources should see a minimum path MTU of 576 bytes.

The inner network layer protocol consults the tunnel interface MTU when admitting a packet into the interface. For non-SEAL inner IPv4 packets with the IPv4 Don't Fragment (DF) bit set to 0, if the packet is larger than the tunnel interface MTU the inner IPv4 layer uses IPv4 fragmentation to break the packet into fragments no larger than the tunnel interface MTU. The ITE then admits each fragment into the interface as an independent packet.

For all other inner packets, the inner network layer admits the packet if it is no larger than the tunnel interface MTU; otherwise, it drops the packet and sends a PTB error message to the source with the MTU value set to the tunnel interface MTU. The message contains as much of the invoking packet as possible without the entire message exceeding the network layer minimum MTU (e.g., 1280 bytes for IPv6, 576 bytes for IPv4, etc.).

The ITE can alternatively set an indefinite MTU on the tunnel interface such that all inner packets are admitted into the interface regardless of their size. For ITEs that host applications that use the tunnel interface directly, this option must be carefully coordinated with protocol stack upper layers since some upper layer protocols (e.g., TCP) derive their packet sizing parameters from the MTU of the outgoing interface and as such may select too large an initial size. This is not a problem for upper layers that use conservative initial maximum segment size estimates and/or when the tunnel interface can reduce the upper layer's maximum segment size, e.g., by reducing the size advertised in the MSS option of outgoing TCP messages (sometimes known as "MSS clamping").

In light of the above considerations, the ITE SHOULD configure an indefinite MTU on tunnel *router* interfaces so that subnetwork adaptation is handled from within the interface. The ITE MAY instead set a finite MTU on tunnel *host* interfaces.

4.4.2. Tunnel Neighbor Soft State

Within the tunnel virtual interface, the ITE maintains a per tunnel neighbor (i.e., a per-ETE) integrity check vector (ICV) calculation algorithm and (when data origin authentication is required) a symmetric secret key to calculate the ICV(s) in packets it will send to this ETE. The ITE also maintains a window of PKT_ID values for the packets it has recently sent to this ETE.

For each ETE link path, the ITE must account for the lengths of the headers to be used for encapsulation. The ITE therefore maintains the per ETE link path constant values "SHLEN" set to length of the SEAL header, "UHLEN" set to the length of the UDP encapsulating header (or 0 if UDP encapsulation is not used), "IHLEN" set to the length of the outer IP layer header, and "HLEN" set to (SHLEN+UHLEN+IHLEN). (The ITE must include the length of the uncompressed headers even if header compression is enabled when calculating these lengths.) In addition, the ETE maintains a constant value "MIN_MTU" set to 1280+HLEN as well as a variable "PATH_MTU" initialized to the MTU of the underlying link. For IPv4, the ITE also maintains the per ETE link path boolean variables "USE_DF" (initialized to "FALSE") and "USE_TRAILER" (initialized to "TRUE" if PATH_MTU is less than MIN_MTU; otherwise initialized to "FALSE") .

The ITE may instead maintain *HLEN, MIN_MTU, PATH_MTU, USE_DF, and USE_TRAILER as per ETE (rather than per ETE link path) values. In that case, the values reflect the lowest-common-denominator MTU across all of the ETE's link paths.

4.4.3. Pre-Encapsulation

For each inner packet admitted into the tunnel interface, if the packet is itself a SEAL packet (i.e., one with either SEAL_PROTO in the IP protocol/next-header field, or with SEAL_PORT in the transport layer destination port field) and the LEVEL field of the SEAL header contains the value 0, the ITE silently discards the packet.

Otherwise, for IPv4 inner packets with DF==0 in the IPv4 header, if the packet is larger than 512 bytes and is not the first fragment of a SEAL packet (i.e., not a packet that includes a SEAL header) the ITE fragments the packet into inner fragments no larger than 512 bytes. The ITE then submits each inner fragment for SEAL encapsulation as specified in Section 4.4.4.

For all other packets, if the packet is no larger than (MAX(PATH_MTU, MIN_MTU)-HLEN) for the corresponding ETE link path, the ITE submits it for SEAL encapsulation as specified in Section 4.4.4. Otherwise, the ITE sends a PTB error message toward the source address of the inner packet.

To send the PTB message, the ITE first checks its forwarding tables to discover the previous hop toward the source address of the inner packet. If the previous hop is reached via the same tunnel interface, the ITE sends an SCMP PTB (SPTB) message to the previous hop (see:

Section 4.6.1.1) with the MTU field set to (MAX(PATH_MTU, MIN_MTU)-HLEN). Otherwise, the ITE sends an ordinary PTB message appropriate to the inner protocol version with the MTU field set to (MAX(PATH_MTU, MIN_MTU)-HLEN).

After sending the (S)PTB message, the ITE discards the inner packet.

4.4.4. SEAL Encapsulation

The ITE next encapsulates the inner packet in a SEAL header formatted as specified in Section 4.3. The ITE sets NEXTHDR to the protocol number corresponding to the address family of the encapsulated inner packet. For example, the ITE sets NEXTHDR to the value '4' for encapsulated IPv4 packets [RFC2003], '41' for encapsulated IPv6 packets [RFC2473][RFC4213], '80' for encapsulated OSI/CLNP packets [RFC1070], etc.

The ITE then sets R=1 if redirects are permitted (see: [I-D.templin-intarea-vet]) and sets PREFLEN to the length of the prefix to be applied to the inner source address. The ITE's claimed PREFLEN is subject to verification by the ETE; hence, the ITE MUST set PREFLEN to the exact prefix length that it is authorized to use. (Note that if this process is entered via re-encapsulation (see: Section 4.5.4), PREFLEN and R are instead copied from the SEAL header of the re-encapsulated packet. This implies that the PREFLEN and R values are propagated across a re-encapsulating chain of ITE/ETEs that must all be authorized to represent the prefix.)

Next, the ITE sets (C=0; P=0; Z=0), then sets LINK_ID to the value assigned to the underlying ETE link path and sets PKT_ID to a monotonically-increasing integer value for this ETE, beginning with 0 in the first packet transmitted. The ITE also sets U=1 if it needs to determine whether the ETE will receive the packet without fragmentation, e.g., for ETE reachability determination (see: Section 4.4.6), to test whether a middlebox on the path is reassembling fragmented packets before they arrive at the ETE (see: Section 4.4.8), for stateful MTU determination (see Section 4.4.9), etc. Otherwise, the ITE sets U=0.

Next, if the inner packet is not itself a SEAL packet the ITE sets LEVEL to an integer value between 0 and 7 as a specification of the number of additional layers of nested SEAL encapsulations permitted. If the inner packet is a SEAL packet that is undergoing nested encapsulation, the ITE instead sets LEVEL to the value that appears in the inner packet's SEAL header minus 1. If the inner packet is undergoing SEAL re-encapsulation, the ITE instead copies the LEVEL value from the SEAL header of the packet to be re-encapsulated.

Next, if this is an IPv4 ETE link path with USE_TRAILER==TRUE, and the inner packet is larger than (128-SHLEN-UHLEN) bytes but no larger than 1280 bytes, the ITE sets T=1. Otherwise, the ITE sets T=0. The ITE then adds the outer encapsulating headers, calculates the ICV(s) and performs any necessary outer fragmentation as specified in Section 4.4.5.

4.4.5. Outer Encapsulation

Following SEAL encapsulation, the ITE next encapsulates the packet in the requisite outer headers according to the specific encapsulation format (e.g., [RFC1070], [RFC2003], [RFC2473], [RFC4213], etc.), except that it writes 'SEAL_PROTO' in the protocol field of the outer IP header (when simple IP encapsulation is used) or writes 'SEAL_PORT' in the outer destination transport service port field (e.g., when IP/UDP encapsulation is used).

When UDP encapsulation is used, the ITE sets the UDP header fields as specified in Section 5.5.4 of [I-D.templin-intarea-vet] (where the UDP header length field includes the length of the SEAL trailer, if present). The ITE then performs outer IP header encapsulation as specified in Section 5.5.5 of [I-D.templin-intarea-vet]. If this process is entered via re-encapsulation (see: Section 4.5.4), the ITE instead follows the outer IP/UDP re-encapsulation procedures specified in Section 5.5.6 of [I-D.templin-intarea-vet].

When IPv4 is used as the outer encapsulation layer, if USE_DF==FALSE the ITE sets DF=0 in the IPv4 header to allow the packet to be fragmented within the subnetwork if it encounters a restricting link. Otherwise, the ITE sets DF=1.

When IPv6 is used as the outer encapsulation layer, the "DF" flag is absent but implicitly set to 1. The packet therefore will not be fragmented within the subnetwork, since IPv6 deprecates in-the-network fragmentation.

The ITE next sets ICV1=0 in the SEAL header and calculates the packet ICVs. The ICVs are calculated using an algorithm agreed on by the ITE and ETE. When data origin authentication is required, the algorithm uses a symmetric secret key so that the ETE can verify that the ICVs were generated by the ITE.

The ITE first calculates the ICV over the leading 128 bytes of the packet (or up to the end of the packet if there are fewer than 128 bytes) beginning with the UDP header (if present) then places result in the ICV1 field in the header. If T==1, the ITE next calculates the ICV over the remainder of the packet and places the result in the ICV2 field in the SEAL trailer. The ITE then submits the packet for outer encapsulation.

Next, the ITE uses IP fragmentation if necessary to fragment the encapsulated packet into outer IP fragments that are no larger than PATH_MTU. By virtue of the pre-encapsulation packet size calculations specified in Section 4.4.3, fragmentation will therefore only occur for outer packets that are larger than PATH_MTU but no larger than MIN_MTU. (Note that, for IPv6, fragmentation must be performed by the ITE itself, while for IPv4 the fragmentation could instead be performed by a router in the ETE link path.)

The ITE then sends each outer packet/fragment via the underlying link corresponding to LINK_ID.

4.4.6. Path Probing and ETE Reachability Verification

All SEAL data packets sent by the ITE are considered implicit probes. SEAL data packets will elicit an SCMP message from the ETE if it needs to acknowledge a probe and/or report an error condition. SEAL data packets may also be dropped by either the ETE or a router on the path, which will return an ICMP message.
The ITE can also send an SCMP Router/Neighbor Solicitation message to elicit an SCMP Router/Neighbor Advertisement response (see: [I-D.templin-intarea-vet]) as verification that the ETE is still reachable via a specific link path.
The ITE processes ICMP messages as specified in Section 4.4.7.
The ITE processes SCMP messages as specified in Section 4.6.2.

4.4.7. Processing ICMP Messages

When the ITE sends SEAL packets, it may receive ICMP error messages[RFC0792][RFC4443] from another ITE on the path to the ETE (i.e., in case of nested encapsulations) or from an ordinary router within the subnetwork. Each ICMP message includes an outer IP header, followed by an ICMP header, followed by a portion of the SEAL data packet that generated the error (also known as the "packet-in-error") beginning with the outer IP header.
The ITE should process ICMPv4 Protocol Unreachable messages and ICMPv6 Parameter Problem messages with Code "Unrecognized Next Header type encountered" as a hint that the ETE does not implement the SEAL protocol. The ITE can also process other ICMP messages that do not include sufficient information in the packet-in-error as a hint that the ETE link path may be failing. Specific actions that the ITE may take in these cases are out of scope.
For other ICMP messages, the should use any outer header information available as a first-pass authentication filter (e.g., to determine if the source of the message is within the same administrative domain as the ITE) and discards the message if first pass filtering fails.
Next, the ITE examines the packet-in-error beginning with the SEAL header. If the value in the PKT_ID field is not within the window of packets the ITE has recently sent to this ETE, or if the value in the SEAL header ICV1 field is incorrect, the ITE discards the message.
Next, if the received ICMP message is a PTB the ITE sets the temporary variable "PMTU" for this ETE link path to the MTU value in the PTB message. If PMTU==0, the ITE consults a plateau table (e.g., as described in [RFC1191]) to determine PMTU based on the length field in the outer IP header of the packet-in-error. (For example, if the ITE receives a PTB message with MTU==0 and length 1500, it can set PMTU=1450. If the ITE subsequently receives a PTB message with MTU==0 and length 1450, it can set PMTU=1400, etc.) If the ITE is performing stateful MTU determination for this ETE link path (see Section 4.4.9), the ITE next sets PATH_MTU=PMTU. If PMTU is less than MIN_MTU, the ITE

sets PATH_MTU=PMTU (and for IPv4 also sets USE_TRAILER=TRUE), then discards the message.
If the ICMP message was not discarded, the ITE then transcribes it into a message to return to the previous hop. If the previous hop toward the inner source address within the packet-in-error is reached via the same tunnel interface the SEAL data packet was sent on, the ITE transcribes the ICMP message into an SCMP message. Otherwise, the ITE transcribes the ICMP message into a message appropriate for the inner protocol version.
To transcribe the message, the ITE extracts the inner packet from within the ICMP message packet-in-error field and uses it to generate a new message corresponding to the type of the received ICMP message. For SCMP messages, the ITE generates the message the same as described for ETE generation of SCMP messages in Section 4.6.1. For (S)PTB messages, the ITE writes (PMTU-HLEN) in the MTU field.
The ITE finally forwards the transcribed message to the previous hop toward the inner source address.

4.4.8. IPv4 Middlebox Reassembly Testing

For IPv4, the ITE can perform a qualification exchange over an ETE link path to ensure that the subnetwork correctly delivers fragments to the ETE. This procedure can be used, e.g., to determine whether there are middleboxes on the path that violate the [RFC1812], Section 5.2.6 requirement that: "A router MUST NOT reassemble any datagram before forwarding it".
When possible, the ITE should use knowledge of its topological arrangement as an aid in determining when middlebox reassembly testing is necessary. For example, if the ITE is aware that the ETE is located somewhere in the public Internet, middlebox reassembly testing is unnecessary. If the ITE is aware that the ETE is located behind a NAT or a firewall, however, then middlebox reassembly testing is recommended.
The ITE can perform a middlebox reassembly test by setting U=1 in the header of a SEAL data packet to be used as a probe. Next, the ITE encapsulates the packet in the appropriate outer headers, splits it into two outer IPv4 fragments, then sends both fragments over the same ETE link path.
While performing the test, the ITE should select only inner packets that are no larger than 1280 bytes for testing purposes in order to avoid reassembly buffer overruns. The ITE can also construct a NULL test packet instead of using ordinary SEAL data packets for testing.
To create the NULL packet, the ITE prepares a data packet with (C=0; P=1; R=0; T=0; U=1; Z=0) in the SEAL header, writes the length of the ITE's claimed prefix in the PREFLEN field, and writes the ITE's claimed prefix in the PREFIX field. The ITE then sets NEXTHDR according to the address family of the PREFIX, i.e., it sets NEXTHDR to the value '4' for an IPv4 prefix, '41' for an IPv6 prefix , '80' for an OSI/CLNP prefix, etc.

The ITE can further add padding following the PREFIX field to a length that would not cause the size of the NULL packet to exceed 1280 bytes before encapsulation. The ITE then sets LINK_ID, LEVEL and PKT_ID to the appropriate values for this ETE link path and calculates ICV1 the same as for an ordinary SEAL data packet.
The ITE should send a series of test packets (e.g., 3-5 tests with 1sec intervals between tests) instead of a single isolated test in case of packet loss, and will eventually receive an SPTB message from the ITE (see: Section 4.6.2.1). If the ETE returns an SCMP PTB message with MTU != 0, then the ETE link path correctly supports fragmentation.
If the ETE returns an SCMP PTB message with MTU==0, however, then a middlebox in the subnetwork is reassembling the fragments before forwarding them to the ETE. In that case, the ITE sets PATH_MTU=MIN_MTU and sets (USE_TRAILER=TRUE; USE_DF=FALSE). The ITE may instead enable stateful MTU determination for this ETE link path as specified in Section 4.4.9 to attempt to discover larger MTUs.
NB: Examples of middleboxes that may perform reassembly include stateful NATs and firewalls. Such devices could still allow for stateless MTU determination if they gather the fragments of a fragmented IPv4 SEAL data packet for packet analysis purposes but then forward the fragments on to the final destination rather than forwarding the reassembled packet.

4.4.9. Stateful MTU Determination

SEAL supports a stateless MTU determination capability, however the ITE may in some instances wish to impose a stateful MTU limit on a particular ETE link path. For example, when the ETE is situated behind a middlebox that performs IPv4 reassembly (see: Section 4.4.8) it is imperative that fragmentation of large packets be avoided on the path to the middlebox. In other instances (e.g., when the ETE link path includes performance-constrained links), the ITE may deem it necessary to cache a conservative static MTU in order to avoid sending large packets that would only be dropped due to an MTU restriction somewhere on the path.
To determine a static MTU value, the ITE can send a series of probe packets of various sizes to the ETE with U=1 in the SEAL header and DF=1 in the outer IP header. The ITE can then cache the size of the largest packet for which it receives a probe reply from the ETE as the PATH_MTU value this ETE link path.
For example, the ITE could send NULL probe packets of 1500 bytes, followed by 1450 bytes, followed by 1400 bytes, etc. then set PATH_MTU for this ETE link path to the size of the largest probe packet for which it receives an SPTB reply message. While probing with NULL probe packets, the ITE processes any ICMP PTB message it receives as a potential indication of probe failure then discards the message.
For IPv4, if the largest successful probe is larger than MIN_MTU the ITE then sets (USE_TRAILER=FALSE; USE_DF=TRUE) for this ETE link path; otherwise, the ITE sets (USE_TRAILER=TRUE; USE_DF=FALSE).

4.4.10. Detecting Path MTU Changes

For IPv6, the ITE can periodically reset PATH_MTU to the MTU of the underlying link to determine whether the ETE link path now supports larger packet sizes. If the path still has a too-small MTU, the ITE will receive a PTB message that reports a smaller size.
For IPv4, when USE_TRAILER==TRUE and PATH_MTU is larger than MIN_MTU the ITE can periodically reset USE_TRAILER=FALSE to determine whether the ETE link path still requires trailers. If the ITE receives an SPTB message for an inner packet that is no larger than 1280 bytes (see: Section 4.6.1.1), the ITE should again set USE_TRAILER=TRUE.
When stateful MTU determination is used, the ITE should periodically re-probe the path as described in Section 4.4.9 to determine whether routing changes have resulted in a reduced or increased PATH_MTU.

4.5. ETE Specification

4.5.1. Tunnel Neighbor Soft State

The ETE maintains a per-ITE ICV calculation algorithm and (when data origin authentication is required) a symmetric secret key to verify the ICV(s) in the SEAL header and trailer. The ETE also maintains a window of PKT_ID values for the packets it has recently received from this ITE.

4.5.2. IP-Layer Reassembly

The ETE must maintain a minimum IP-layer reassembly buffer size of 1500 bytes for both IPv4 [RFC0791] and IPv6 [RFC2460].
The ETE should maintain conservative reassembly cache high- and low-water marks. When the size of the reassembly cache exceeds this high-water mark, the ETE should actively discard stale incomplete reassemblies (e.g., using an Active Queue Management (AQM) strategy) until the size falls below the low-water mark. The ETE should also actively discard any pending reassemblies that clearly have no opportunity for completion, e.g., when a considerable number of new fragments have arrived before a fragment that completes a pending reassembly arrives.
The ETE processes non-SEAL IP packets as specified in the normative references, i.e., it performs any necessary IP reassembly then discards the packet if it is larger than the reassembly buffer size or delivers the (fully-reassembled) packet to the appropriate upper layer protocol module.
For SEAL packets, the ITE performs any necessary IP reassembly until it has received at least the first 1280 bytes beyond the SEAL header or up to the end of the packet. For IPv4, the ETE then submits the (fully- or partially-reassembled) packet for decapsulation as specified in Section 4.5.3. For IPv6, the ETE only submits the packet if it was fully-reassembled and no larger than the reassembly buffer size.

4.5.3. Decapsulation and Re-Encapsulation

For each SEAL packet submitted for decapsulation, the ETE first examines the PKT_ID and ICV1 fields. If the PKT_ID is not within the window of acceptable values for this ITE, or if the ICV1 field includes an incorrect value, the ETE silently discards the packet.

Next, if the SEAL header has T==1 and the inner packet is larger than 1280 bytes the ETE silently discards the packet. If the SEAL header has T==1 and the inner packet is no larger than 1280 bytes, the ETE instead verifies the ICV2 value and silently discards the packet if the value is incorrect.

Next, if the SEAL header has C==0 and there is an incorrect value in a SEAL header field (e.g., an incorrect "VER" field value), the ETE returns an SCMP "Parameter Problem" (SPP) message (see Section 4.6.1.2) and discards the packet.

Next, if the packet arrived as multiple IPv4 fragments and the inner packet is larger than 1280 bytes, the ETE sends an SPTB message back to the ITE with MTU set to the size of the largest fragment received minus HLEN (see: Section 4.6.1.1) then discards the packet. If the packet arrived as multiple IPv6 fragments and the inner packet is larger than 1280 bytes, the ETE instead silently discards the packet.

Next, if the packet arrived as multiple IPv4 fragments, the SEAL header has (C==0; T==0), and the inner packet is larger than (128-SHLEN-UHLEN) bytes, the ETE sends an SPTB message back to the ITE with MTU set to the size of the largest fragment received minus HLEN (see: Section 4.6.1.1) then continues to process the packet.

Next, if the SEAL header has C==1, the ETE processes the packet as an SCMP packet as specified in Section 4.6.2. Otherwise, the ETE continues to process the packet as a SEAL data packet.

Next, if the packet arrived unfragmented and the SEAL header has U==1, the ETE sends an SPTB message back to the ITE with MTU=0 (see: Section 4.6.1.1).

Next, if the SEAL header has P==1 the ETE discards the (NULL) packet.

Finally, the ETE discards the outer headers and processes the inner packet according to the header type indicated in the SEAL NEXTHDR field. If the next hop toward the inner destination address is via a different interface than the SEAL packet arrived on, the ETE discards the SEAL header and delivers the inner packet either to the local host or to the next hop interface if the packet is not destined to the local host.

If the next hop is on the same interface the SEAL packet arrived on, however, the ETE submits the packet for SEAL re-encapsulation beginning with the specification in Section 4.4.3 above. In this process, the packet remains within the tunnel interface (i.e., it does not exit and then re-enter the interface); hence, the packet is not discarded if the LEVEL field in the SEAL header contains the value 0.

4.6. The SEAL Control Message Protocol (SCMP)

SEAL provides a companion SEAL Control Message Protocol (SCMP) that uses the same message types and formats as for the Internet Control Message Protocol for IPv6 (ICMPv6) [RFC4443]. As for ICMPv6, each SCMP message includes a 4-byte header and a variable-length body. The TE encapsulates the SCMP message in a SEAL header and outer headers as shown in Figure 4:

```
                                    +--------------------+
                                    ~   outer IP header  ~
                                    +--------------------+
                                    ~  other outer hdrs  ~
                                    +--------------------+
                                    ~    SEAL Header     ~
       +--------------------+       +--------------------+
       | SCMP message header|  -->  | SCMP message header|
       +--------------------+       +--------------------+
       |                    |  -->  |                    |
       ~  SCMP message body ~  -->  ~  SCMP message body ~
       |                    |  -->  |                    |
       +--------------------+       +--------------------+

            SCMP Message                  SCMP Packet
        before encapsulation          after encapsulation
```

The following sections specify the generation, processing and relaying of SCMP messages.

4.6.1. Generating SCMP Error Messages

ETEs generate SCMP error messages in response to receiving certain SEAL data packets using the format shown in Figure 5:

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |     Type      |     Code      |          Checksum             |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                      Type-Specific Data                       |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |      As much of the inner packet within the invoking          |
   ~       SEAL data packet as possible without the SCMP           ~
   |              packet exceeding 576 bytes (*)                   |

   (*) also known as the "packet-in-error"
```

When the ETE processes a SEAL data packet for which the ICVs are correct but an error must be returned, it prepares an SCMP error message as shown in Figure 5. The ETE sets the Type and Code fields to

the same values that would appear in the corresponding ICMPv6 message and calculates the Checksum beginning with the SCMP message header and continuing to the end of the message. (When calculating the Checksum, the TE sets the Checksum field itself to 0.)
The ETE next encapsulates the SCMP message in the requisite SEAL header, outer headers and SEAL trailer as shown in Figure 4. During encapsulation, the ETE sets the outer destination address/port numbers of the SCMP packet to the outer source address/port numbers of the original SEAL data packet and sets the outer source address/port numbers to its own outer address/port numbers.
The ETE then sets (C=1; R=0; T=0; U=0; Z=0) in the SEAL header, then sets NEXTHDR, PREFLEN, LINK_ID, LEVEL, and PKT_ID to the same values that appeared in the SEAL header of the data packet. If the SEAL data packet header had P==1, the ETE also copies the PREFIX field from the data packet into the SEAL header and sets P=1; otherwise, it sets P=0. The ETE then calculates and sets the ICV1 field the same as specified for SEAL data packet encapsulation in Section 4.4.4. Next, the ETE encapsulates the SCMP message in the requisite outer encapsulations and sends the resulting SCMP packet to the ITE the same as specified for SEAL data packets in Section 4.4.5.
The following sections describe additional considerations for various SCMP error messages:

4.6.1.1. Generating SCMP Packet Too Big (SPTB) Messages

An ETE generates an SCMP "Packet Too Big" (SPTB) message when it receives a SEAL data packet that arrived as multiple outer IPv4 fragments and for which the reassembled inner packet would be larger than 1280 bytes. The ETE also generates an SPTB when it receives the fragments of a fragmented IPv4-encapsulated SEAL data packet with T==0 in the SEAL header but that following reassembly would be larger than (128-SHLEN-UHLEN) bytes but no larger than 1280 bytes. The ETE prepares the SPTB message the same as for the corresponding ICMPv6 PTB message, and writes the length of the largest outer IP fragment received minus HLEN in the MTU field of the message.
The ETE also generates an SPTB message when it accepts a SEAL protocol data packet which did not undergo IP fragmentation and with U==1 in the SEAL header. The ETE prepares the SPTB message the same as above, except that it writes the value 0 in the MTU field.

4.6.1.2. Generating Other SCMP Error Messages

An ETE generates an SCMP "Destination Unreachable" (SDU) message under the same circumstances that an IPv6 system would generate an ICMPv6 Destination Unreachable message.
An ETE generates an SCMP "Parameter Problem" (SPP) message when it receives a SEAL packet with an incorrect value in the SEAL header. IN THIS CASE ALONE, the ETE prepares the packet-in-error beginning with the SEAL header instead of beginning immediately after the SEAL header.

TEs generate other SCMP message types using methods and procedures specified in other documents. For example, SCMP message types used for tunnel neighbor coordinations are specified in VET [I-D.templin-intarea-vet].

4.6.2. Processing SCMP Error Messages

An ITE may receive SCMP messages after sending packets to an ETE. The ITE first verifies that the outer addresses of the SCMP packet are correct, and that the PKT_ID is within its window of values for this ETE. The ITE next verifies that the SEAL header fields are set correctly as specified in Section 4.6.1. The ITE then verifies the ICV1 value. If the outer addresses, SEAL header information and/or ICV1 value are incorrect, the ITE silently discards the message; otherwise, it processes the message as follows:

4.6.2.1. Processing SCMP PTB Messages

After an ITE sends a SEAL data packet to an ETE, it may receive an SPTB message with a packet-in-error containing the leading portion of the inner packet (see: Section 4.6.1.1). For IP SPTB messages with MTU==0, the ITE processes the message as confirmation that the ETE received an unfragmented SEAL data packet with U==1 in the SEAL header. The ITE then discards the message.
For IPv4 SPTB messages with MTU != 0, the ITE instead processes the message as an indication of a packet size limitation as follows. The ITE first determines the inner packet length by subtracting SHLEN from the length field in the UDP header within the packet-in-error (and also subtracting the length of the SEAL trailer when T=1). If the inner packet is no larger than 1280 bytes, the ITE sets USE_TRAILER=TRUE. If the inner packet is larger than 1280 bytes, the ITE instead examines the SPTB message MTU field. If the MTU value is not substantially less than (1500-HLEN), the value is likely to reflect the true MTU of the restricting link on the path to the ETE; otherwise, a router on the path may be generating runt fragments.
In that case, the ITE can consult a plateau table (e.g., as described in [RFC1191]) to rewrite the MTU value to a reduced size. For example, if the ITE receives an IPv4 SPTB message with MTU==256 and inner packet length 1500, it can rewrite the MTU to 1450. If the ITE subsequently receives an IPv4 SPTB message with MTU==256 and inner packet length 1450, it can rewrite the MTU to 1400, etc. If the ITE is performing stateful MTU determination for this ETE link path, it then writes the new MTU value in PATH_MTU.
The ITE then checks its forwarding tables to discover the previous hop toward the source address of the inner packet. If the previous hop is reached via the same tunnel interface the SPTB message arrived on, the ITE relays the message to the previous hop. In order to relay the message, the ITE rewrites the SEAL header fields with values corresponding to the previous hop and recalculates the ICV1 values

using the ICV calculation parameters associated with the previous hop. Next, the ITE replaces the SPTB's outer headers with headers of the appropriate protocol version and fills in the header fields as specified in Sections 5.5.4-5.5.6 of [I-D.templin-intarea-vet], where the destination address/port correspond to the previous hop and the source address/port correspond to the ITE. The ITE then sends the message to the previous hop the same as if it were issuing a new SPTB message.

If the previous hop is not reached via the same tunnel interface, the ITE instead transcribes the message into a format appropriate for the inner packet (i.e., the same as described for transcribing ICMP messages in Section 4.4.7) and sends the resulting transcribed message to the original source. The ITE then discards the SPTB message.

4.6.2.2. Processing Other SCMP Error Messages

An ITE may receive an SDU message with an appropriate code under the same circumstances that an IPv6 node would receive an ICMPv6 Destination Unreachable message. The ITE either transcribes or relays the message toward the source address of the inner packet within the packet-in-error the same as specified for SPTB messages in Section 4.6.2.1.

An ITE may receive an SPP message when the ETE receives a SEAL packet with an incorrect value in the SEAL header. The ITE should examine the SEAL header within the packet-in-error to determine whether a different setting should be used in subsequent packets, but does not relay the message further.

TEs process other SCMP message types using methods and procedures specified in other documents. For example, SCMP message types used for tunnel neighbor coordinations are specified in VET [I-D.templin-intarea-vet].

5. Link Requirements

Subnetwork designers are expected to follow the recommendations in Section 2 of [RFC3819] when configuring link MTUs.

6. End System Requirements

End systems are encouraged to implement end-to-end MTU assurance (e.g., using Packetization Layer Path MTU Discovery per [RFC4821]) even if the subnetwork is using SEAL.

7. Router Requirements

Routers within the subnetwork are expected to observe the router requirements found in the normative references, including the implementation of IP fragmentation and reassembly [RFC1812][RFC2460] as well as the generation of ICMP messages [RFC0792][RFC4443].

8. Nested Encapsulation Considerations

SEAL supports nested tunneling for up to 8 layers of encapsulation. In this model, the SEAL ITE has a tunnel neighbor relationship only with ETEs at its own nesting level, i.e., it does not have a tunnel neighbor relationship with any ITEs/ETEs at other nesting levels.
Therefore, when an ITE 'A' within an inner nesting level needs to return an error message to an ITE 'B' within an outer nesting level, it generates an ordinary ICMP error message the same as if it were an ordinary router within the subnetwork. 'B' can then perform message validation as specified in Section 4.4.7, but full message origin authentication is not possible.
Since ordinary ICMP messages are used for coordinations between ITEs at different nesting levels, nested SEAL encapsulations should only be used when the ITEs are within a common administrative domain and/or when there is no ICMP filtering middlebox such as a firewall or NAT between them. An example would be a recursive nesting of mobile networks, where the first network receives service from an ISP, the second network receives service from the first network, the third network receives service from the second network, etc.

9. IANA Considerations

The IANA is instructed to allocate an IP protocol number for 'SEAL_PROTO' in the 'protocol-numbers' registry.
The IANA is instructed to allocate a Well-Known Port number for 'SEAL_PORT' in the 'port-numbers' registry.
The IANA is instructed to establish a "SEAL Protocol" registry to record SEAL Version values. This registry should be initialized to include the initial SEAL Version number, i.e., Version 0.

10. Security Considerations

SEAL provides a segment-by-segment data origin authentication and anti-replay service across the (potentially) multiple segments of a re-encapsulating tunnel. It further provides a segment-by-segment integrity check of the headers of encapsulated packets, but does not verify the integrity of the rest of the packet beyond the headers unless fragmentation is unavoidable. SEAL therefore considers full message integrity checking, authentication and confidentiality as end-to-end considerations in a manner that is compatible with securing mechanisms such as TLS/SSL [RFC5246].
An amplification/reflection/buffer overflow attack is possible when an attacker sends IP fragments with spoofed source addresses to an ETE in an attempt to clog the ETE's reassembly buffer and/or cause the ETE to generate a stream of SCMP messages returned to a victim ITE. The SCMP message ICVs, PKT_ID, as well as the inner headers of the packet-in-error, provide mitigation for the ETE to detect and discard SEAL segments with spoofed source addresses.

The SEAL header is sent in-the-clear the same as for the outer IP and other outer headers. In this respect, the threat model is no different than for IPv6 extension headers. Unlike IPv6 extension headers, however, the SEAL header is protected by an integrity check that also covers the inner packet headers.
Security issues that apply to tunneling in general are discussed in [RFC6169].

11. Related Work

Section 3.1.7 of [RFC2764] provides a high-level sketch for supporting large tunnel MTUs via a tunnel-level segmentation and reassembly capability to avoid IP level fragmentation. This capability was implemented in the first edition of SEAL, but is now deprecated.
Section 3 of [RFC4459] describes inner and outer fragmentation at the tunnel endpoints as alternatives for accommodating the tunnel MTU.
Section 4 of [RFC2460] specifies a method for inserting and processing extension headers between the base IPv6 header and transport layer protocol data. The SEAL header is inserted and processed in exactly the same manner.
IPsec/AH is [RFC4301][RFC4301] is used for full message integrity verification between tunnel endpoints, whereas SEAL only ensures integrity for the inner packet headers. The AYIYA proposal [I-D.massar-v6ops-ayiya] uses similar means for providing full message authentication and integrity.
The concepts of path MTU determination through the report of fragmentation and extending the IPv4 Identification field were first proposed in deliberations of the TCP-IP mailing list and the Path MTU Discovery Working Group (MTUDWG) during the late 1980's and early 1990's. An historical analysis of the evolution of these concepts, as well as the development of the eventual path MTU discovery mechanism, appears in Appendix D of this document.

12. Acknowledgments

The following individuals are acknowledged for helpful comments and suggestions: Jari Arkko, Fred Baker, Iljitsch van Beijnum, Oliver Bonaventure, Teco Boot, Bob Braden, Brian Carpenter, Steve Casner, Ian Chakeres, Noel Chiappa, Remi Denis-Courmont, Remi Despres, Ralph Droms, Aurnaud Ebalard, Gorry Fairhurst, Washam Fan, Dino Farinacci, Joel Halpern, Sam Hartman, John Heffner, Thomas Henderson, Bob Hinden, Christian Huitema, Eliot Lear, Darrel Lewis, Joe Macker, Matt Mathis, Erik Nordmark, Dan Romascanu, Dave Thaler, Joe Touch, Mark Townsley, Ole Troan, Margaret Wasserman, Magnus Westerlund, Robin Whittle, James Woodyatt, and members of the Boeing Research & Technology NST DC&NT group.
Discussions with colleagues following the publication of RFC5320 have provided useful insights that have resulted in significant improvements to this, the Second Edition of SEAL.

Path MTU determination through the report of fragmentation was first proposed by Charles Lynn on the TCP-IP mailing list in 1987. Extending the IP identification field was first proposed by Steve Deering on the MTUDWG mailing list in 1989.

13. References

13.1. Normative References

[RFC0791]	Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.
[RFC0792]	Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, September 1981.
[RFC4443]	Conta, A., Deering, S. and M. Gupta, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", RFC 4443, March 2006.
[RFC3971]	Arkko, J., Kempf, J., Zill, B. and P. Nikander, "SEcure Neighbor Discovery (SEND)", RFC 3971, March 2005.
[RFC4861]	Narten, T., Nordmark, E., Simpson, W. and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861, September 2007.
[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
[RFC2460]	Deering, S.E. and R.M. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

13.2. Informative References

[RFC1063]	Mogul, J., Kent, C., Partridge, C. and K. McCloghrie, "IP MTU discovery options", RFC 1063, July 1988.
[RFC1191]	Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, November 1990.
[RFC1981]	McCann, J., Deering, S. and J. Mogul, "Path MTU Discovery for IP version 6", RFC 1981, August 1996.
[RFC2003]	Perkins, C., "IP Encapsulation within IP", RFC 2003, October 1996.
[RFC2473]	Conta, A. and S. Deering, "Generic Packet Tunneling in IPv6 Specification", RFC 2473, December 1998.
[RFC2923]	Lahey, K., "TCP Problems with Path MTU Discovery", RFC 2923, September 2000.
[RFC3366]	Fairhurst, G. and L. Wood, "Advice to link designers on link Automatic Repeat reQuest (ARQ)", BCP 62, RFC 3366, August 2002.
[RFC3819]	

	Karn, P., Bormann, C., Fairhurst, G., Grossman, D., Ludwig, R., Mahdavi, J., Montenegro, G., Touch, J. and L. Wood, "Advice for Internet Subnetwork Designers", BCP 89, RFC 3819, July 2004.
[RFC4213]	Nordmark, E. and R. Gilligan, "Basic Transition Mechanisms for IPv6 Hosts and Routers", RFC 4213, October 2005.
[RFC1812]	Baker, F., "Requirements for IP Version 4 Routers", RFC 1812, June 1995.
[RFC4380]	Huitema, C., "Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)", RFC 4380, February 2006.
[RFC4301]	Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.
[RFC4302]	Kent, S., "IP Authentication Header", RFC 4302, December 2005.
[RFC5246]	Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
[RFC4459]	Savola, P., "MTU and Fragmentation Issues with In-the-Network Tunneling", RFC 4459, April 2006.
[RFC4821]	Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, March 2007.
[RFC4963]	Heffner, J., Mathis, M. and B. Chandler, "IPv4 Reassembly Errors at High Data Rates", RFC 4963, July 2007.
[RFC2764]	Gleeson, B., Heinanen, J., Lin, A., Armitage, G. and A.G. Malis, "A Framework for IP Based Virtual Private Networks", RFC 2764, February 2000.
[RFC2675]	Borman, D., Deering, S. and R. Hinden, "IPv6 Jumbograms", RFC 2675, August 1999.
[RFC5445]	Watson, M., "Basic Forward Error Correction (FEC) Schemes", RFC 5445, March 2009.
[RFC1070]	Hagens, R., Hall, N. and M. Rose, "Use of the Internet as a subnetwork for experimentation with the OSI network layer", RFC 1070, February 1989.
[RFC3232]	Reynolds, J., "Assigned Numbers: RFC 1700 is Replaced by an On-line Database", RFC 3232, January 2002.
[RFC4191]	Draves, R. and D. Thaler, "Default Router Preferences and More-Specific Routes", RFC 4191, November 2005.
[RFC4987]	Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, August 2007.
[RFC5720]	Templin, F., "Routing and Addressing in Networks with Global Enterprise Recursion (RANGER)", RFC 5720, February 2010.

[I-D.templin-intarea-vet]	Templin, F, "Virtual Enterprise Traversal (VET)", Internet-Draft draft-templin-intarea-vet-31, November 2011.
[I-D.ietf-savi-framework]	Wu, J, Bi, J, Bagnulo, M, Baker, F and C Vogt, "Source Address Validation Improvement Framework", Internet-Draft draft-ietf-savi-framework-05, July 2011.
[I-D.templin-ironbis]	Templin, F, "The Internet Routing Overlay Network (IRON)", Internet-Draft draft-templin-ironbis-08, November 2011.
[RFC6139]	Russert, S., Fleischman, E. and F. Templin, "Routing and Addressing in Networks with Global Enterprise Recursion (RANGER) Scenarios", RFC 6139, February 2011.
[RFC5927]	Gont, F., "ICMP Attacks against TCP", RFC 5927, July 2010.
[RFC6169]	Krishnan, S., Thaler, D. and J. Hoagland, "Security Concerns with IP Tunneling", RFC 6169, April 2011.
[I-D.ietf-intarea-ipv4-id-update]	Touch, J, "Updated Specification of the IPv4 ID Field", Internet-Draft draft-ietf-intarea-ipv4-id-update-04, September 2011.
[I-D.templin-aero]	Templin, F, "Asymmetric Extended Route Optimization (AERO)", Internet-Draft draft-templin-aero-04, October 2011.
[I-D.massar-v6ops-ayiya]	Massar, J, "AYIYA: Anything In Anything", Internet-Draft draft-massar-v6ops-ayiya-02, July 2004.
[FRAG]	Kent, C and J Mogul, "Fragmentation Considered Harmful", October 1987.
[FOLK]	Shannon, C, Moore, D and k claffy, "Beyond Folklore: Observations on Fragmented Traffic", December 2002.
[MTUDWG]	, , "IETF MTU Discovery Working Group mailing list, gatekeeper.dec.com/pub/DEC/WRL/mogul/mtudwg-log, November 1989 - February 1995.", .
[TCP-IP]	, , "Archive/Hypermail of Early TCP-IP Mail List, http://www-mice.cs.ucl.ac.uk/multimedia/misc/tcp_ip/, May 1987 - May 1990.", .
[TBIT]	Medina, A, Allman, M and S Floyd, "Measuring Interactions Between Transport Protocols and Middleboxes", October 2004.
[WAND]	Luckie, M, Cho, K and B Owens, "Inferring and Debugging Path MTU Discovery Failures", October 2005.
[SIGCOMM]	Luckie, M and B Stasiewicz, "Measuring Path MTU Discovery Behavior", November 2010.

Appendix A. Reliability

Although a SEAL tunnel may span an arbitrarily-large subnetwork expanse, the IP layer sees the tunnel as a simple link that supports the IP service model. Links with high bit error rates (BERs) (e.g., IEEE 802.11) use Automatic Repeat-ReQuest (ARQ) mechanisms [RFC3366] to increase packet delivery ratios, while links with much lower BERs typically omit such mechanisms. Since SEAL tunnels may traverse arbitrarily-long paths over links of various types that are already either performing or omitting ARQ as appropriate, it would therefore often be inefficient to also require the tunnel endpoints to also perform ARQ.

Appendix B. Integrity

The SEAL header includes an ICV field that covers the SEAL header and at least the inner packet headers. This provides for header integrity verification on a segment-by-segment basis for a segmented re-encapsulating tunnel path. When IPv4 fragmentation is needed, the SEAL packet also contains a trailer with a secondary ICV that covers the remainder of the packet.
Fragmentation and reassembly schemes must consider packet-splicing errors, e.g., when two fragments from the same packet are concatenated incorrectly, when a fragment from packet X is reassembled with fragments from packet Y, etc. The primary sources of such errors include implementation bugs and wrapping IPv4 ID fields.
In terms of wrapping ID fields, the IPv4 16-bit ID field can wrap with only 64K packets with the same (src, dst, protocol)-tuple alive in the system at a given time [RFC4963] increasing the likelihood of reassembly mis-associations
When reassembly is unavoidable, SEAL provides an extended ICV to detect reassembly mis-associations for packets no larger than 1280 bytes and also discards any reassembled packets larger than 1280 bytes.

Appendix C. Transport Mode

SEAL can also be used in "transport-mode", e.g., when the inner layer comprises upper-layer protocol data rather than an encapsulated IP packet. For instance, TCP peers can negotiate the use of SEAL (e.g., by inserting a 'SEAL_OPTION' TCP option during connection establishment) for the carriage of protocol data encapsulated as IP/SEAL/TCP. In this sense, the "subnetwork" becomes the entire end-to-end path between the TCP peers and may potentially span the entire Internet.
If both TCPs agree on the use of SEAL, their protocol messages will be carried as IP/SEAL/TCP and the connection will be serviced by the SEAL protocol using TCP (instead of an encapsulating tunnel endpoint) as the transport layer protocol. The SEAL protocol for transport mode otherwise observes the same specifications as for Section 4.

Appendix D. Historic Evolution of PMTUD

The topic of Path MTU discovery (PMTUD) saw a flurry of discussion and numerous proposals in the late 1980's through early 1990. The initial problem was posed by Art Berggreen on May 22, 1987 in a message to the TCP-IP discussion group [TCP-IP]. The discussion that followed provided significant reference material for [FRAG]. An IETF Path MTU Discovery Working Group [MTUDWG] was formed in late 1989 with charter to produce an RFC. Several variations on a very few basic proposals were entertained, including:

1. Routers record the PMTUD estimate in ICMP-like path probe messages (proposed in [FRAG] and later [RFC1063])

2. The destination reports any fragmentation that occurs for packets received with the "RF" (Report Fragmentation) bit set (Steve Deering's 1989 adaptation of Charles Lynn's Nov. 1987 proposal)

3. A hybrid combination of 1) and Charles Lynn's Nov. 1987 (straw RFC draft by McCloughrie, Fox and Mogul on Jan 12, 1990)

4. Combination of the Lynn proposal with TCP (Fred Bohle, Jan 30, 1990)

5. Fragmentation avoidance by setting "IP_DF" flag on all packets and retransmitting if ICMPv4 "fragmentation needed" messages occur (Geof Cooper's 1987 proposal; later adapted into [RFC1191] by Mogul and Deering).

Option 1) seemed attractive to the group at the time, since it was believed that routers would migrate more quickly than hosts. Option 2) was a strong contender, but repeated attempts to secure an "RF" bit in the IPv4 header from the IESG failed and the proponents became discouraged. 3) was abandoned because it was perceived as too complicated, and 4) never received any apparent serious consideration. Proposal 5) was a late entry into the discussion from Steve Deering on Feb. 24th, 1990. The discussion group soon thereafter seemingly lost track of all other proposals and adopted 5), which eventually evolved into [RFC1191] and later [RFC1981].
In retrospect, the "RF" bit postulated in 2) is not needed if a "contract" is first established between the peers, as in proposal 4) and a message to the MTUDWG mailing list from jrd@PTT.LCS.MIT.EDU on Feb 19. 1990. These proposals saw little discussion or rebuttal, and were dismissed based on the following the assertions:

*routers upgrade their software faster than hosts

*PCs could not reassemble fragmented packets

- *Proteon and Wellfleet routers did not reproduce the "RF" bit properly in fragmented packets

- *Ethernet-FDDI bridges would need to perform fragmentation (i.e., "translucent" not "transparent" bridging)

- *the 16-bit IP_ID field could wrap around and disrupt reassembly at high packet arrival rates

The first four assertions, although perhaps valid at the time, have been overcome by historical events. The final assertion is addressed by the mechanisms specified in SEAL.

Author's Address

Fred L. Templin editor Templin Boeing Research & Technology P.O. Box 3707 Seattle, WA 98124 USA EMail: fltemplin@acm.org