Network Working Group
Internet-Draft
Intended status: Informational
Expires: May 17, 2008

F. Templin, Ed.
Boeing Phantom Works

November 14, 2007

Simple Protocol for Robust IP/*/IP Tunnel Endpoint MTU Determination (sprite-mtu)
draft-templin-inetmtu-06.txt

Status of this Memo

This Internet-Draft will expire on May 17, 2008.

Copyright Notice

Abstract

The nominal Maximum Transmission Unit (MTU) of today's Internet has become 1500 bytes, but IP/*/IP tunneling mechanisms impose an encapsulation overhead that can reduce the effective path MTU to smaller values. Additionally, existing tunneling mechanisms are limited in their ability to support larger MTUs. This document specifies a simple protocol for robust IP/*/IP tunnel endpoint MTU determination (sprite-mtu).

Table of Contents

1. Introduction

The nominal Maximum Transmission Unit (MTU) of today's Internet has become 1500 bytes due to the preponderance of networking gear that configures an MTU of that size. Since not all links in the Internet configure a 1500 byte MTU, however, packets can be dropped due to an MTU restriction on the path.

Upper layers see IP/*/IP tunnels as ordinary links, but even for small packets these links are susceptible to silent loss (e.g., due to path MTU restrictions, lost error messages, layered encapsulations, reassembly buffer limitations, etc.) resulting in poor performance and/or communications failures [RFC2923][RFC4459][RFC4821][RFC4963].

This document specifies a simple protocol for robust IP/*/IP tunnel endpoint MTU determination (sprite-mtu), and updates the functional specifications for Tunnel Endpoints (TEs) found in existing tunneling mechanisms (see: Section 6).

This document seeks to achieve an appropriate balance between function in the network and function in the end systems [RFC1958], and further observes the tunnel management specifications in [RFC2003][RFC2473][RFC4213].

2. Terminology and Requirements

The following abbreviations and terms are used in this document:

ICMP - ICMPv4 [RFC0793] or ICMPv6 [RFC4443].

IP - IPv4 [RFC0791] or IPv6 [RFC2460].

IP/*/IP - an inner IP packet encapsulated in outer */IP headers (e.g. for "*" = NULL, UDP, TCP, AH, ESP, etc.)

inner packet/fragment/header - an IP packet/fragment/header before */IP encapsulation.

outer packet/fragment/header - a */IP packet/fragment/header after encapsulation.

AQM - Active Queue Management

DF - the IPv4 header "Don't Fragment" flag

ENCAPS - the size of the encapsulating */IP headers plus trailers

EMTU_R - Effective Maximum Transmission Unit to Receive [RFC1122] for the TFE

MTU - Maximum Transmission Unit

linkMTU - MTU assigned to a link over which the tunnel is configured

pathMTU - the minimum path MTU for the tunnel

PTB - an ICMPv4 "Destination Unreachable - fragmentation needed" [RFC1191] or an ICMPv6 "Packet Too Big" [RFC1981] message.

sprite-mtu - the sprite-mtu protocol, specified in this document

sprite-udp - the sprite-udp UDP messaging service, also specified in this document

sprite - a message of the "sprite-udp" service

TE - Tunnel Endpoint

TFE - Tunnel Far End

TNE - Tunnel Near End

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in [RFC2119].

3. Concept of Operation

TEs use the tunnel management specifications in [RFC2003][RFC2473][RFC4213] and also participate in the "sprite-mtu" protocol to confirm the participation of the TFE, to determine per-tunnel MTU values, to detect path MTU-related black-holes, and to detect congestion. The protocol is supported through the exchange of messages between TEs using the "sprite-udp" UDP service. The mechanisms provide robust MTU determination and congestion control when both TEs support the protocol, and support the legacy behavior otherwise.

4. "sprite-udp" UDP Service

The "sprite-udp" service is a simple UDP service based on "sprite" messages. Sprite requests set the UDP destination port to the sprite-udp service port (see: Section 7), and set the source port to either the sprite-udp service port or a dynamic port number chosen by the source. Sprite replys set the UDP source port to the sprite-udp service port and set the UDP destination port to the value included in the UDP source port in the soliciting sprite request.

All sprite requests and replys are formatted as shown in Figure 1 (format for other sprite messages may be specified in future documents):

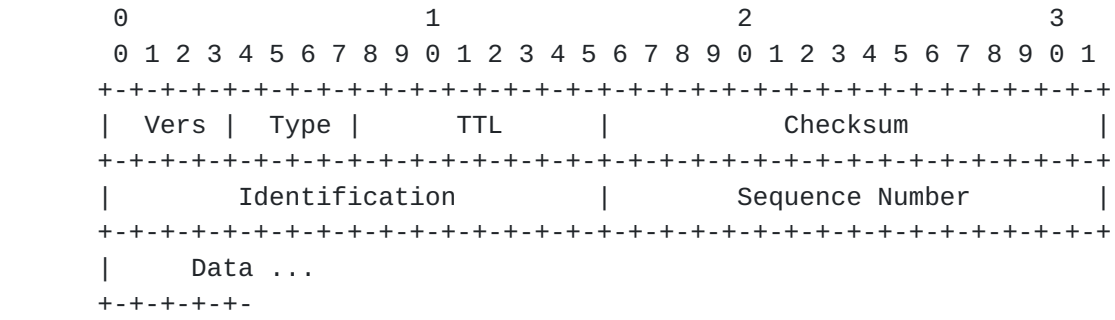

Figure 1: Sprite Message Format

where the fields of the message body are defined as follows:

Vers
: the Version field indicates the sprite-udp protocol version. This document describes version 1.

Type
: the message type. Currently defined values are:

  0 - request

  1 - reply

  2 - 15 reserved for future use

TTL
: in sprite requests, set to 0; in sprite replies, set to the TTL/ Hop Limit in the IP header of the request to which this packet is a reply.

Checksum
the 16-bit one's complement of the one's complement sum of the message body, starting with the version field and ending at the end of the data field. For computing the checksum, the checksum field is first set to zero. An all zero transmitted checksum value means that the transmitter generated no checksum.

Identification, Sequence Number
Two 16-bit fields, used exactly as specified for the corresponding fields in ICMP echo request and reply messages.

Data
Zero or more octets of arbitrary data, included in the sprite request and echoed in the reply.

5. "sprite-mtu" Protocol Specification

TEs that implement the sprite-udp service MUST also participate in the "sprite-mtu" protocol to: 1) determine whether the TFE implements the scheme, 2) detect path MTU-related black holes, 3) provide timely aging of stale path MTU information, 4) determine the length of the forward path through the tunnel, 5) determine accurate round trip times, and 6) detect and report congestion. The following sections specify the protocol details:

5.1. Interactions with End-to-End MTU Determination

The sprite-mtu protocol operates independently of any end-to-end MTU determination, however it offers improved convergence time and efficiency when end-to-end mechanisms such as [RFC4821] are also used.

5.2. Tunnel Virtual Interface MTU and linkMTU

TEs SHOULD configure an MTU on the tunnel virtual interface (i.e., the MTU that is seen by upper layers) that is at least as large as the largest linkMTU for all underlying interfaces over which the tunnel virtual interface is configured. For IPv6/*/IP tunnels, the tunnel virtual interface MUST configure an MTU no smaller than 1280 bytes, and for IPv4/*/IP tunnels it SHOULD configure an MTU no smaller than 576 bytes.

Additionally, operators SHOULD observe the recommendations in [RFC3819], Section 2, i.e., they should avoid setting a too-small linkMTU on any of the underlying interfaces over which the tunnel virtual interface is configured.

5.3. Sprite Addresses

TEs must configure a time-varying link-local address (e.g., they regenerate a new link-local address every 30 seconds) for the tunnel interface known as the "sprite address". The sprite address is associated with the tunnel interface (i.e., not assigned to the interface) hence it need not be checked for uniqueness and will only be used for the purpose of sprite exchanges for soft state management (see below).

For IPv6/*/IPv4 tunnels, link-local IPv6 privacy addresses [RFC4941] are used. For IPv4/*/IP tunnels, random assignments from the IPv4 link local address range [RFC3927] are used, however the number of bits for randomness is significantly smaller than for IPv6.

This addressing scheme is not available for unidirectional tunnels, since link local addresses would not be routable on the (non-tunneled) return path from the TFE to the TNE.

5.4. Per-Tunnel Tunnel Soft State

TEs maintain per-tunnel soft state information (e.g., in a conceptual neighbor cache) that is initialized when there is evidence that a continuous flow of data will traverse the tunnel and is scheduled for deletion based on idle timers, resource limitations, etc. thereafter. The TNE maintains state regarding the forward path to the TFE. When necessary (e.g., when the tunnel is fragmenting), the TFE also maintains state regarding the number of packets received, packets in error, etc. The minimum state kept by the TNE and TFE is given in the following sections:

5.4.1. TNE Soft State

The TNE keeps the following minimum per-tunnel soft state for active tunnels, and MAY keep additional soft state (e.g., packets/bytes sent, collisions, etc.):

isQualified
: boolean indicating whether the TFE implements the protocol.

  Initial value: FALSE

TFEAddr
: the inner IP address of the TFE.

pathMTU
the current minimum path MTU across the tunnel to the TFE, determined through sprite probing to determine the largest size outer packet that can traverse the tunnel.

Initial value: 0.

TTL
the current path length across the tunnel to the TFE, determined through sprite probing.

Initial value: 0.

RTT
the round trip time for the TFE.

Initial value: none

spriteList
a list of sprite requests that have been sent into the tunnel but not yet acknowledged by the TFE.

Initial list: NULL

5.4.2. TFE Soft State

When requested by the TNE, the TFE keeps the following minimum per-tunnel soft state information, and MAY keep additional soft state (e.g., congestion, error rate, etc.):

rxTime
the system time at which the soft state for this TNE was initialized.

rxPackets
number of packets received.

rxBytes
number of bytes received.

rxDropped
number of packets dropped due to incorrect checksums, congestion, etc.

5.5. Soft State Maintenance

5.5.1. TNE Soft State Maintenance

When a TNE has data to send to a TFE, it obeys the specification in the normative IP/*/IP reference. When there is evidence that a persistent flow of data will traverse the tunnel, the TNE also creates soft state per Section 5.3.1 (unless the soft state already exists) and sends sprite requests into the tunnel. The TNE includes in the inner IP header of each sprite request its current sprite address as the source address and a destination address that is either the current sprite address of the TFE or a different link local address. The TNE then sets 'Vers' to '1', sets 'Type' to '0', sets 'TTL' to '0', sets 'Identification' and 'Sequence Number' to identifying values, and includes a randomly-chosen nonce value (8 bytes recommended) in the 'Data' field along with any other data to be echoed. The TNE finally calculates the checksum and writes its value in the 'Checksum' field (or, writes the value '0'), then sends the sprite request into the tunnel.

If the TNE receives a sprite reply message that is apparently from the TFE and also includes an 'Identification', 'Sequence Number', and 'Data' that matches a request in its 'spriteList', it sets 'isQualified' to TRUE; otherwise, it discards the reply. The TNE also records the round-trip time (RTT) relative to the time at which it sent the soliciting sprite request, as well as the difference between the TTL of the soliciting request and the TTL encoded in the reply. The TNE finally records the source address in the inner IP header of the sprite reply in 'TFEAddr'.

When the TNE requires the TFE to maintain state (e.g., when the tunnel is fragmenting), it sends continuous sprite requests at a rate of no more than 1 request per second and sets the inner destination address of each request to 'TFEAddr'. When the flow of data ceases, the TNE stops sending sprite requests and schedules the soft state entry for deletion. When the flow of data resumes, the TNE resumes sending sprite requests.

The TNE removes sprite requests from its 'spriteList' when either a matching reply is received, or after a timeout period during which no matching reply is received. (Timeouts on the order of IPv6 neighbor discovery [RFC4861] are recommended.)

5.5.2. TFE Soft State Maintenance

When a TFE receives a sprite request, it prepares a reply that includes the inner IP source address of the request in the inner IP destination address and its current sprite address in the inner IP

source address. The TFE then sets 'Vers' to '1', sets 'Type to '1', copies the TTL/Hop Limit from the sprite request into the 'TTL' field, and copies 'Identification', 'Sequence Number' and 'Data' from the request message body into the corresponding fields of the reply. The TFE finally calculates the checksum and writes its value in the 'Checksum' field (or, writes the value '0'), then sends the sprite reply back to the TNE.

If the inner IP destination address of the sprite request was the same as the TFE's current sprite address, the TFE creates soft state per Section 5.3.2 if possible. The TFE maintains the soft state as long as it continues to receive sprite requests from the TNE that include an inner destination address that matches its current sprite address. When the flow of requests ceases, the TFE schedules the soft state entry for deletion.

If the TFE is unable to maintain soft state, e.g., due to resource limitations, it sets the inner IP destination address in its sprite replys to a different link local IP address than the one included in the inner IP source address of the sprite request. The TNE must accept this as an indication that the TFE is not currently maintaining soft state.

5.6. Sending Packets

Inner IP packets forwarded by upper layers that are larger than the tunnel virtual interface MTU are dropped with an ICMP Packet Too Big (PTB) sent back to the original source, as for any IP interface. Other inner IP packets are forwarded into the tunnel interface, which will encapsulate and send them on the underlying tunnel and/or return an internally-generated PTB when necessary.

TEs that implement the sprite-mtu protocol use the specifications for sending packets found in the following sections:

5.6.1. Conceptual Sending Algorithm

TEs use the conceptual sending algorithm in Figure 2 for sending packets that are forwarded into the tunnel virtual interface by upper layers:

```
Fragment inner packet if necessary (Section 5.5.2)
foreach inner packet/fragment
    Encapsulate as an outer packet (Section 5.5.3).
    Fragment outer packet if necessary (Sect. 5.5.4).
    foreach outer packet/fragment
        Send packet/fragment (Section 5.5.5).
    endforeach
    Send PTB appropriate to the inner protocol if
    necessary (Section 5.5.6).
end foreach
```

Figure 2: Conceptual Sending Algorithm

5.6.2. Inner Packet Fragmentation

The TE considers an inner IPv4 packet as fragmentable IFF the DF bit is set to 0 in the inner IPv4 header, and assumes that the original source has learned through some end-to-end means that the final destination is able to reassemble a packet of this size. The TE uses IPv4 fragmentation to break fragmentable inner packets into fragments no larger than (576 - ENCAPS) before encapsulation; these inner fragments will ultimately be reassembled by the final destination.

The TE is not permitted to fragment inner IPv6 packets, therefore an inner IPv6 packet is never fragmentable.

5.6.3. Encapsulation and Trailers

TEs encapsulate inner IP packets according to the specific IP/*/IP document. When 'isQualified' is TRUE, the TE includes a trailer with a correct non-zero checksum in all packets that may incur outer fragmentation (see: Section 5.5.4). For other packets, the TE can either: 1) include a trailer with a correct non-zero checksum, 2) include a trailer with a zero checksum, or 3) omit the trailer.

For outer packets that will include a trailer during encapsulation, the TE includes zero or more padding bytes plus a 4-byte trailing checksum immediately following the inner IP packet. The TE increments the innermost '*' header length field by the number of trailer bytes added before applying the outermost */IP encapsulation(s). For example, it increments the UDP length field for IP/UDP/IP tunnels ('*' = UDP), the IPv4 length field for IP/IPv4 tunnels ('*' = NULL), etc. The encapsulation is shown in Figure 3:

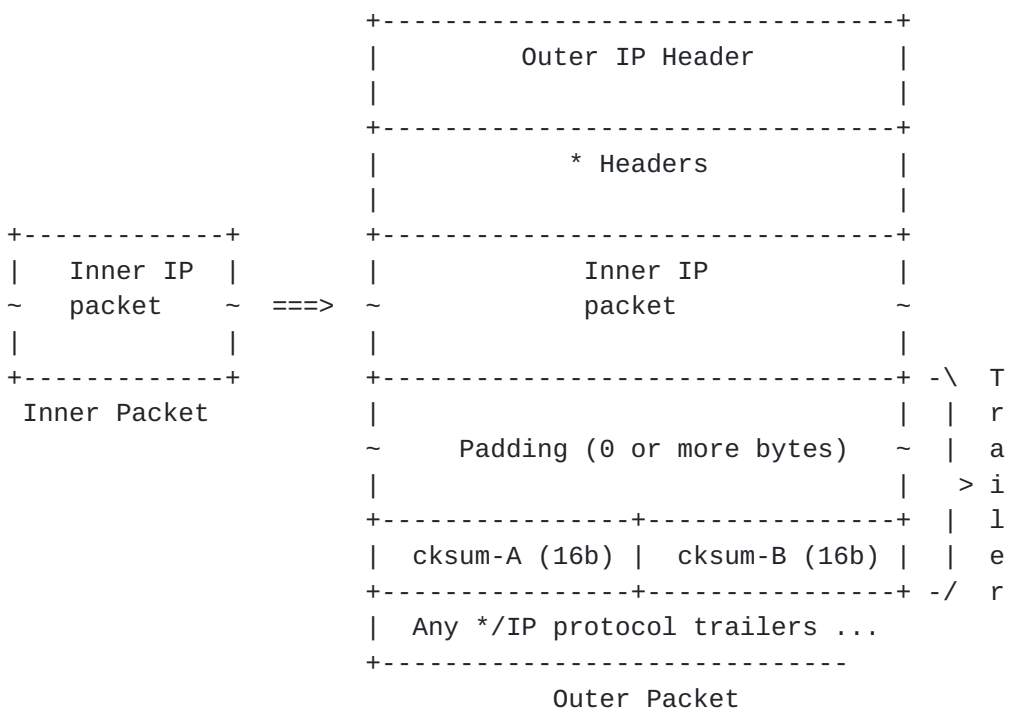

Figure 3: Encapsulation Format with Trailer

For all packets that will include a trailer, the TE appends any padding bytes as necessary to extend the packet to a specific length then calculates the checksum as specified in Section 5.10. It then appends the results in the A and B fields of the trailing checksum. (The TE instead writes the value 0 in these fields if the trailer is to include a zero checksum). The checksum is byte-aligned only, i.e., it need not be aligned on an even word/longword/etc. boundary.

The TE SHOULD NOT include a trailer during encapsulation when 'isQualified' is FALSE.

5.6.4. Outer Packet Fragmentation and Setting DF

The TE is not permitted to fragment outer */IPv6 packets using IPv6 fragmentation.

The TE considers an outer */IPv4 packet as fragmentable IFF:

- for IPv6/*/IPv4 tunnels, 'pathMTU' is less than (1280 bytes + ENCAPS) and the inner IPv6 packet is no larger than 1280 bytes, or:

- for IPv4/*/IPv4 tunnels, 'pathMTU' is less than MIN(EMTU_R, 1280) bytes and the inner IPv4 packet is no larger than MIN(EMTU_R, 1280) minus ENCAPS. (When EMTU_R for the TFE is not known, 576

bytes must be assumed.)

The TE's */IPv4 encapsulation layer(s) MAY fragment a fragmentable outer packet before admitting it into the tunnel. The TE SHOULD set DF=1 in the outer IPv4 header of each fragment if 'pathMTU' is known; otherwise, the TE MAY set DF=0 if there is assurance that the TFE can receive non-initial IPv4 fragments. These outer fragments will be reassembled by the TFE.

The TE MUST set DF=1 in the outer IPv4 header of all unfragmentable outer packets.

5.6.5. Admitting Packets into the Tunnel

For IP/*/IPv4 tunnels, when 'pathMTU' is smaller than the minimum values listed in Section 5.5.4 and the TFE is not maintaining soft state, the TE MUST institute pacing and AQM to minimize IPv4 reassembly misassociations and/or congestion at the TFE. The TE MAY relax this pacing when the TFE indicates that it is maintaining soft state, but MUST resume pacing if it subsequently detects congestion at the TFE (see: Section 5.8).

The TE admits outer packets into the tunnel subject to pacing and TTL restrictions. For unfragmentable outer packets that are larger than 'pathMTU', the TE admits the packet but also sends a PTB message appropriate to the inner IP protocol with an MTU size of ('pathMTU' - ENCAPS).

5.7. Receiving Packets

TEs that implement the sprite-mtu protocol use the specifications for receiving packets found in the following sections:

5.7.1. IPv4 Reassembly Cache Management

IP/*/IPv4 TEs SHOULD perform AQM in their IPv4 reassembly cache. In particular, they should actively discard "stale" reassemblies that have no apparent opportunity for successful completion, i.e., even before the packets have reached the normal reassembly timeout expiration recommended in [RFC1122], Section 3.3.2.

5.7.2. Decapsulation

When the TE receives (and, if necessary, reassembles) an encapsulated packet, and the packet includes a trailing checksum, the TE accepts the packet if the checksum is correct and drops the packet if the checksum is incorrect. Otherwise, the TE decapsulates the packet exactly as specified in the appropriate IP/*/IP document and discards

the trailer.

During decapsulation, it the TE is maintaining soft state it also records 'rx*' statistics in its soft state entry for the tunnel, i.e., it increments 'rxPackets' and 'rxBytes' for packets received with no errors, and increments 'rxDropped' for packets dropped due to checksum errors, congestion, etc.

5.7.3. Receiving Packet Too Big (PTB) Errors

TNEs may receive PTB errors in response to any packets they admit into the tunnel. When the TNE receives a PTB with an MTU value smaller than 'pathMTU', it SHOULD record the new 'pathMTU' in its soft state entry for the tunnel.

5.7.4. Receiving Other ICMP Errors

TEs SHOULD observe the specifications in [RFC2003][RFC2473][RFC4213] when they receive other ICMP errors from within the tunnel, but are advised that ICMP denial-of-service attacks are possible.

5.8. MTU Probing and Black Hole Detection

When 'isQualified' is TRUE, the TE can send sprite requests to the TFE with trailing padding added during encapsulation to create an MTU probe of the desired length. In particular, when the TE sends a data packet into the tunnel with a size that exceeds the current 'pathMTU', it MAY also send a padded sprite request of the same size into the tunnel. If the TE receives a matching sprite reply, it advances 'pathMTU' to the size of the request.

The TE MAY send additional sprite requests into the tunnel to determine a maximum 'pathMTU' independent of any data packets sent into the tunnel, to detect a smaller 'pathMTU' due to routing changes and/or to detect MTU-related black holes.

For IP/*/IPv4 tunnels, the TE MUST set DF=1 in any sprite request used for the purpose of 'pathMTU' probing.

5.9. Congestion Control

The TNE MUST set the ECN field in the inner IP header of sprite requests used for soft state maintenance to either ECT(0) or ECT(1) [RFC3168] and MUST also examine the ECN field in the inner IP headers of sprite replys. When the TNE begins to observe the CE codepoint in the ECN field in the inner IP headers of successive sprite replys at a rate that exceeds a high water threshold, it institutes pacing per Section 5.5.5. The TNE MAY relax pacing when the rate falls below a

low water threshold.

When soft state management is requested by the TNE, the TFE MUST track the rate at which packets received from the TNE are dropped due to, e.g., checksum errors, congestion, etc. When the rate exceeds a high water threshold, the TFE MUST begin setting the CE codepoint in the ECN field in the inner IP headers of sprite replys sent in response to requests with the ECN field set to other than not-ECT. The TFE MUST also set the CE codepoint in ordinary data packets with the ECN field set to other than not-ECT when it forwards them to the next IP hop. When the rate of dropped packets falls below a low water threshold, the TFE MAY relax CE codepoint marking.

When the TFE is temporarily unable to maintain soft state, it includes a link local address in the inner IP destination address of its sprite replies that is different than the inner IP source address that the TNE included in its sprite request. The TNE must interpret this as an indication that pacing should resume.

5.10. sprite-mtu Checksum Calculation

This specification uses a 16-bit variation of the Fletcher Checksum [RFC1146][STONE1][STONE2] called: the "sprite-mtu checksum" which provides a lightweight integrity check with different properties than those used by common link layers and upper layer protocols.

The TE calculates the sprite-mtu checksum by summing every 10th byte of the packet beginning with the inner IP header up to the end of the inner packet including trailing padding, but not including the trailing checksum field itself. The TE calculates the checksum according to the algorithm below, which represents a slight variation of that found in [RFC1146]:

> The sprite-mtu checksum is calculated over a sequence of unsigned data octets (call them D[0] through D[N-1]) by maintaining unsigned 1's-complement 16-bit accumulators A and B whose contents are initially zero, and performing the following loop where i ranges from 1 to N:
>
> ```
> A := A + D[i]
> B := B + A
> i := i + 10
> ```
>
> If, at the end of the loop, either or both of the A and B accumulators encode the value 0x0000, invert the value in the accumulator(s) to 0xffff.

Note that faster algorithms are possible and may be used instead of

the algorithm above (see: [RFC1146]). Note also that for '*/IP' encapsulations that include an additional checksum, the sprite-mtu checksum can be calculated in parallel.

6. Updated Specifications

This document updates the following specifications, and possibly others:

- RFC1853 (IP-in-IP)
- RFC2003 (IP-in-IP)
- RFC2473 (Generic packet tunneling in IPv6)
- RFC2529 (6over4)
- RFC2661 (L2TP)
- RFC2784 (GRE)
- RFC3056 (6to4)
- RFC3378 (ETHERIP)
- RFC3884 (IPSec Transport Mode for Dynamic Routing)
- RFC4023 (MPLS-in-IP)
- RFC4213 (Basic IPv6 Transition Mechanisms)
- RFC4214 (ISATAP)
- RFC4301 (IPSec)
- RFC4302 (AH)
- RFC4303 (ESP)
- RFC4380 (TEREDO)
- LISP
- IPAE
- others....

7. IANA Considerations

A new UDP port number for the "sprite-udp" service is requested.

8. Security Considerations

A possible denial of service attack vector involves an off-path attacker sending sprite replys with spoofed source addresses, however the 8-byte nonce serves as an effective mitigation.

A possible resource exhaustion attack vector exist when TEs use well-known and/or time-invariant addresses. Sprite addresses serve as an effective mitigation for bidirectional tunnels, as specified in Section 5.9.

Security considerations for specific IP/*/IP tunneling mechanisms are specified in their respective documents.

9. Acknowledgments

This work has benefited from discussions with Fred Baker, Iljitsch van Beijnum, Brian Carpenter, Steve Casner, Remi Denis-Courmont, Aurnaud Ebalard, Gorry Fairhurst, John Heffner, Bob Hinden, Christian Huitema, Joe Macker, Matt Mathis, Dave Thaler, Joe Touch, Magnus Westerlund, Robin Whittle, and James Woodyatt.

This work is dedicated to the editor's family.

10. References

10.1. Normative References

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC1122] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.

[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.

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

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[RFC2473] Conta, A. and S. Deering, "Generic Packet Tunneling in IPv6 Specification", RFC 2473, December 1998.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.

[RFC3927] Cheshire, S., Aboba, B., and E. Guttman, "Dynamic Configuration of IPv4 Link-Local Addresses", RFC 3927, May 2005.

[RFC4213] Nordmark, E. and R. Gilligan, "Basic Transition Mechanisms for IPv6 Hosts and Routers", RFC 4213, October 2005.

[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.

[RFC4941] Narten, T., Draves, R., and S. Krishnan, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", RFC 4941, September 2007.

10.2. Informative References

[RFC1146] Zweig, J. and C. Partridge, "TCP alternate checksum options", RFC 1146, March 1990.

[RFC1958] Carpenter, B., "Architectural Principles of the Internet", RFC 1958, June 1996.

[RFC2923] Lahey, K., "TCP Problems with Path MTU Discovery", RFC 2923, September 2000.

[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.

[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.

[RFC4861] Narten, T., Nordmark, E., Simpson, W., and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)", RFC 4861, September 2007.

[RFC4963] Heffner, J., Mathis, M., and B. Chandler, "IPv4 Reassembly Errors at High Data Rates", RFC 4963, July 2007.

[STONE1] Stone, J., "Checksums in the Internet (Stanford Doctoral Dissertation)", August 2001.

[STONE2] Stone, J., Greenwald, M., Partridge, C., and J. Hughes, "Performance of Checksums and CRC's over Real Data, IEEE/ACM Transactions on Networking, Vol 6, No. 5", October 1998.

Author's Address

Fred L. Templin (editor)
Boeing Phantom Works
P.O. Box 3707
Seattle, WA 98124
USA

Email: fred.l.templin@boeing.com

Full Copyright Statement

Intellectual Property

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).