Layer Two Tunneling Protocol - Version 4 (L2TPv4)
draft-bhati-l2ptext-l2tpv4-00.txt

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

This Internet-Draft will expire on April 25, 2009.
Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the
document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal
Provisions Relating to IETF Documents
(http://trustee.ietf.org/license-info) in effect on the date of
publication of this document. Please review these documents
carefully, as they describe your rights and restrictions with respect
to this document. Code Components extracted from this document must
include Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.
Abstract

This document describes the Layer Two Tunneling protocol (L2TPv4). L2TPv4 defines implementation friendly and more secured encapsulation headers for carrying multiple Layer 2 connections between two IP nodes.

Table of Contents

1. Introduction...................................................4
   1.1. Changes from RFC 3931....................................4
2. Conventions used in this document............................4
3. L2TPv4 data packet header format............................5
   3.1. Discussion................................................8
4. L2TPv4 control packet header format...........................9
   4.1. Discussion...............................................11
5. Security Considerations.....................................12
6. IANA Considerations........................................12
7. Conclusions..................................................12
8. References....................................................13
   8.1. Normative References....................................13
   8.2. Informative References................................13
9. Acknowledgments.............................................13
Appendix A. Received Packet Processing........................14
Appendix B. Packet Header Encapsulation Processing...........17
1. Introduction

L2TP as originally defined in RFC 2661, is a standard method for tunneling Point-To-Point (PPP) [RFC 1661] sessions. L2TP is then modified in L2TPv3 for adoption of tunneling a number of other L2 protocols. This document describes L2TPv4 encapsulation headers (for data packets and control packets) that is implementation friendly and provides more security features. Irrespective of whether we carry L2TPv4 payload as IP layer payload [IP + L2TP + layer-2 header + payload] or transport layer payload [IP + UDP + L2TP + layer-2 header + payload], L2TPv4 encapsulation header format will be same.

The following header formats are described in following sections.

- L2TPv4 data packet header over IP or UDP
- L2TPv4 control packet header over IP or UDP

1.1. Changes from RFC 3931

Everything that is described in RFC 3931 will remain intact except the L2TP header formats for data packets and control packets. This draft document proposes to use common L2TPv4 data packet header format irrespective of the PSNs over which L2TPv4 is operating. This draft document also proposes to use common L2TPv4 control packet header format irrespective of the PSNs over which L2TPv4 is operating. L2TPv3 data and control header formats are different in cases when L2TPv3 is operating over IP or over UDP. This draft document proposes to replace those differences with similarities across PSNs.

L2TPv4 implementations MUST support L2TP over IP and should support L2TP over UDP for better NAT and firewall traversal, and for easier migration from previous L2TP implementations.

2. Conventions used in this document

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 RFC 2119 [RFC2119].
3. L2TPv4 data packet header format

L2TPv4 data packet header format is described in the following figure. This header format is same irrespective of PSNs [over IP, over UDP] over which L2TP is operating.

```
+------------------+-
|V|V|V|V|T|C|S|X|X|X|X|X|    L2TP Header Checksum       |
+------------------+-
|     Proto    |Proto Hdr Len  |    L2TP Payload Length        |
+------------------+-
|                     L2TP Session Id                           |
+------------------+-
|         Session Cookie (Optional, maximum 64 bits)            |
+------------------+-
```

Figure 1 L2TPv4 data packet header

VVVV:

A 4 bit field which indicates version of L2TP header. Its value is equal to 0100 for L2TPv4 data packet.

T:

A 1 bit field which indicates whether it is a data packet or control packet. Its value is equal to 0 for data packets.
C:
A 1 bit field which indicates header checksum is present inside L2TP header and needs to be validated. Checksum calculation is similar to IPv4 header checksum calculation as described in [RFC 1071]. For L2TP data packets directly operating over IP, it is recommended to set this bit and update the 16-bit checksum value inside L2TP header. This bit can be set to 0 in case L2TP is operating over UDP because UDP checksum can be used to provide integrity to L2TP header and its payload. If C bit is zero, checksum field MUST contain 0x0000 as 16 bit value. Checksum covers all bytes starting from first bit of L2TP header till last byte of session cookie value (if present).

S:
A 1 bit field which indicates session cookie value is present inside L2TP header and a comparison is required between this value and cookie present inside L2TP session context.

L2TP Header Checksum:
A 16 bit field which contains checksum value and covers entire L2TP header. This value should be written to all zeros before actually doing the checksum calculation on header bytes.

Proto:
An 8 bit field which contains the protocol value which L2TP is carrying inside as a payload. As described in [RFC 3931], this is equal to the pseudo-wire type. For example: If L2TP is used to carry PPP frames, the Pseudo-wire type PS-WIRE-PPP should be set as proto value inside L2TP header. This value of protocol is covered inside L2TP header integrity checksum.
Proto Header Length:

An 8 bit field which carries length of protocol header which is encapsulated inside L2TP protocol. As described in [RFC 3931], this is equal to the length of pseudo-wire header. For example: If L2TP is used to carry HDLC frames, the Pseudo-wire type PS-WIRE-HDLC should be set as proto value inside L2TP header and proto header length should be set to 4. For any encapsulation sequence, we always know how many bytes of pseudo-wire header are added before encapsulation of L2TP headers. This value of protocol header length is covered inside L2TP header integrity checksum.

L2TP Payload Length:

A 16 bit field which carries total length of L2TP payload. This value does not include L2TP header length. If L2TP is used to carry PPP frames, then L2TP payload length value is equal to sum of PPP header and PPP payload bytes.

L2TP Session ID:

A 32 bit field containing a non-zero identifier for a session as described in [RFC 3931].

L2TP Session Cookie:

The optional cookie field contains a variable length value (maximum 64 bits) used to check the association of a received data message with the session identified by the Session ID. It is as described in [RFC 3931].
3.1. Discussion

The above mentioned format of L2TP data message header shall be used irrespective of whether L2TP is operating over IP or over UDP. The main reason for not having L2TP header length inside L2TP header is to avoid guessing the actual session cookie length by intermediate entities. The prediction of session cookie by intermediate entities can compromise the L2TP security features.

If IPv4 protocol value is 115, and first 5 bits after end of IPv4 header are equal to 01000, then we MUST interpret the packet as L2TPv4 data packet.

If IPv4 protocol value is 17, then 8 bytes of UDP header will be present after IPv4 header. If operating over UDP, L2TP peers usually agree on a UDP port range. If UDP destination port falls in between that range and first 5 bits after UDP header are equal to 01000, then we MUST interpret the packet as L2TPv4 data packet.

Addition of version bits inside L2TP header flags makes it easier to define future versions of L2TP protocol. It is also easier for the implementations to switch between different versions dynamically when L2TP entity receives L2TP packets of different versions. This scenario can happen when multiple LAC with different L2TP versions establish tunnel with LNS which supports all versions of L2TP protocol.
4. L2TPv4 control packet header format

L2TPv4 data packet header format is described in the following figure. This header format is same irrespective of PSNs [over IP, over UDP] over which L2TP is operating.

```
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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V|V|V|V|T|C|S|X|X|X|X|X|X|X|X|X|    L2TP Header Checksum      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Length               |    Reserved                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Control Connection Id                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Ns                 |         Nr                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2 L2TPv4 control packet header

VVVV:

A 4 bit field indicates the version of L2TP header. Its value is equal to 0100 for L2TPv4 control packet.

T:

A 1 bit field indicates whether it is a data packet or control packet. Its value is equal to 1 for control packets.
C:
A 1 bit field which indicates header checksum is present inside L2TP header and needs to be validated. Checksum calculation is similar to IPv4 header checksum calculation as described in [RFC 1071]. For L2TP data packets directly operating over IP, it is recommended to set this bit and update the 16-bit checksum value inside L2TP header. This bit can be set to 0 in case L2TP is operating over UDP because UDP checksum can be used to provide security to L2TP header and its payload. Checksum covers all bytes starting from first bit of L2TP header till last bit of Nr field.

S:
A 1 bit field indicates sequence numbers (Ns and Nr) are present inside L2TP control packet header.

L2TP Header Checksum:
A 16 bit field contains checksum value for L2TPv4 control packet header. This value should be written to all zeros before actually doing the checksum calculation on header bytes.

Length:
A 16 bit field carries length of control packet bytes. This is equal to the sum of L2TPv4 control header bytes and L2TPv4 control payload. This length is covered inside the checksum calculation of L2TPv4 control packet, if C bit is 1.

Control Connection ID:
A 32 bit field which carries control connection identifier. This value is detailed in RFC 3931.
Ns:

A 16 bit field carries control message sequence number for the sender of control message. Interpretation of this field is governed by [RFC 3931]. This field is present only if S bit is set in L2TPv4 flags.

Nr:

A 16 bit field carries last received message number which is used to acknowledge messages received by a L2TP peer. Interpretation of this field is governed by [RFC 3931]. This field is present only if S bit is set in L2TPv4 flags.

4.1. Discussion

This format of L2TP control message header shall be used irrespective of whether L2TP is operating over IP or over UDP. If IP protocol value is 115, and first 5 bits of L2TP header are equal to 01001, then we shall interpret the packet as L2TP control packet.

When operating directly over IP, L2TP packets lose the ability to take advantage of UDP checksum as a simple measure for packet integrity check, which is of a particular concern to L2TP control messages. That is the main reason to include L2TP header checksum as an optional field inside L2TPv4 headers. The presence or absence of actual checksum is controlled by C bit in L2TPv4 header flags.

If IP protocol value is 17, then 8 bytes of UDP header will be present after IP header. If operating over UDP, L2TP peers usually agree on a UDP port range. If UDP destination port falls in between that range and first 5 bits after UDP header are equal to 01001, then we shall interpret the packet as L2TPv4 control packet.
5. Security Considerations

This section addresses data header integrity concern. Optional header checksum field is added inside data packet header which is calculated over L2TP data header including variable length of session cookie field. If session cookie flag is set in data packet header, it is recommended that header checksum flag MUST be set to 1 for additional level of integrity check.

6. IANA Considerations

This draft document does not request any IANA action.

7. Conclusions

This draft document proposes header formats for L2TPv4 data and control packets which are implementation friendly and provides additional level of integrity check via new fields inside headers. The header formats of data packet and control packet are common whether we carry L2TP over IP or L2TP over UDP.
8. References

8.1. Normative References


8.2. Informative References


9. Acknowledgments

Many of the protocol constructs were originally defined in RFC 2661, "L2TPv2". RFC 2661 authors are W. Townsley, A. Valencia, A. Rubens, G. Pall, G. Zorn, B. Palter. Thanks so much for defining a wonderful protocol.

Thanks to the authors of RFC 3931 "L2TPv3". RFC 3931 authors are J. Lau, M. Townsley, I. Goyret. Addition of multiple pseudo wire types is indeed an important update to L2TP protocol.

Big Thanks to J. Touch who has prepared the word template document to edit/write new RFC documents. It is really difficult to write a new RFC without this template. This document was prepared using 2-Word-v2.0.template.dot.
Appendix A. Received Packet Processing

```c
void rcvd_packet_process(uint8_t *pkt_ptr)
{
    ip_hdr_ptr = (ip_hdr_t *)(pkt_ptr);
    if(115 == ip_hdr_ptr->protocol)
    {
        l2tp_hdr_ptr = (uint8_t *)(ip_hdr_ptr + ip_hdr_length);
        first_two_bytes = *((uint16_t *)(l2tp_hdr_ptr));
        if(0x4000 == first_two_bytes & 0x4800)
        {
            // this is L2TPv4 data packet
        }
        else if(0x4800 == first_two_bytes & 0x4800)
        {
            // this is L2TPv4 control packet
        }
        else
        {
            // MUST discard this packet
        }
    }
}```
else if(17 == ip_hdr_ptr->protocol)
{
    udp_hdr_ptr = (udp_hdr_t *)(ip_hdr_ptr + 1);

    if(l2tp_session_ptr->udp_min < udp_hdr_ptr->dest_port &&
        l2tp_session_ptr->udp_max > udp_hdr_ptr->dest_port &&
        l2tp_session_ptr->l2tp_over_udp == 1)
    {
        l2tp_hdr_ptr = (l2tp_hdr_t *)(udp_hdr_ptr + 1);
        first_two_bytes = *((uint16_t *)(l2tp_hdr_ptr));
        if(0x4000 == first_two_bytes & 0x4800)
        {
            // this is L2TPv4 data packet
        }
        else if(0x4800 == first_two_bytes & 0x4800)
        {
            // this is L2TPv4 control packet
        }
        else
        {
            // MUST discard this packet
        }
    }
}
Copyright (c) 2016 IETF Trust and the persons identified as authors of the code. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- Neither the name of Internet Society, IETF or IETF Trust, nor the names of specific contributors, may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Appendix B. Packet Header Encapsulation Processing

```c
void l2tp_data_packet_sender(uint8_t *pkt_ptr,
                              uint16_t pkt_len,
                              void *l2tp_session_context)
{

  switch(l2tp_session_context->pw_type)
  {
    case 'PPP':
      l2_hdr_size = 4;
      l2_hdr_ptr = pkt_ptr - l2_hdr_size;
      *((uint32_t *)(l2tp_hdr_ptr)) = layer_2_hdr_value;
      break;
    case 'HDLC':
      l2_hdr_size = 4;
      l2_hdr_ptr = pkt_ptr - l2_hdr_size;
      *((uint32_t *)(l2tp_hdr_ptr)) = layer_2_hdr_value;
      break;
    case 'consider other cases also'
      break;
    default:
      break;
  }
}```
// Time to Add L2TPv4 data header

// 2 byte flags + 2 byte header checksum + 1 byte proto +
// 1 byte proto_len + 2 byte payload_len + 4 byte session id
// Total fixed size = 12 bytes

uint8_t l2tp_hdr_size = 12;
if(1 == l2tp_session_context->cookie_set)
{
    l2tp_hdr_size += l2tp_session_context->cookie_len;
}

l2tp_hdr_ptr = l2_hdr_ptr - l2tp_hdr_size;
l2tp_hdr_ptr->flags.version = 0x4; // L2TPv4
l2tp_hdr_ptr->flags.T_bit = 0; // data packet
if(l2tp_session_context->l2tp_over_ip)
{
    l2tp_hdr_ptr->flags.C_bit = 1;
}
else
{
    l2tp_hdr_ptr->flags.C_bit = 0; // UDP checksum is enough
}
if(l2tp_session_context->cookie_set)
{
    l2tp_hdr_ptr->flags.S_bit = 1;
}

l2tp_hdr_ptr->checksum = 0x0000;
l2tp_hdr_ptr->proto = l2tp_session_context->pw_type;
l2tp_hdr_ptr->proto_hdr_len = l2_hdr_size;
l2tp_hdr_ptr->payload_len = pkt_len + l2_hdr_len;
l2tp_hdr_ptr->session_id = l2tp_session_context->session_id;
if(l2tp_hdr_ptr->flags.S_bit)
{
    memcpy(l2tp_hdr_ptr + 12,      // destination
            l2tp_session_ptr->cookie, // source
            l2tp_session_ptr->cookie_len); // length
}

if(l2tp_hdr_ptr->flags.C_bit)
{
    l2tp_hdr_ptr->checksum =
        checksum(l2tp_hdr_ptr, 12 + l2tp_session_ptr->cookie_len);
}
} // end of pseudo function
Copyright (c) 2016 IETF Trust and the persons identified as authors of the code. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

- Neither the name of Internet Society, IETF or IETF Trust, nor the names of specific contributors, may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Authors’ Addresses

ABHISHEK BHATI
SAMSUNG ELECTRONICS
SAMSUNG R&D INSTITUTE, BENGALURU, INDIA

Phone: +91-9686500752
Email: ABH.BHATI@SAMSUNG.COM / AB.BHATI@GMAIL.COM