                        PPP in HDLC-like Framing


Status of this Memo

   This document specifies an Internet standards track protocol for the
   Internet community, and requests discussion and suggestions for
   improvements.  Please refer to the current edition of the "Internet
   Official Protocol Standards" (STD 1) for the standardization state
   and status of this protocol.  Distribution of this memo is unlimited.


Abstract

   The Point-to-Point Protocol (PPP) [1] provides a standard method for
   transporting multi-protocol datagrams over point-to-point links.

   This document describes the use of HDLC-like framing for PPP
   encapsulated packets.


Table of Contents

1.  Introduction

   This specification provides for framing over both bit-oriented and
   octet-oriented synchronous links, and asynchronous links with 8 bits
   of data and no parity.  These links MUST be full-duplex, but MAY be
   either dedicated or circuit-switched.

   An escape mechanism is specified to allow control data such as
   XON/XOFF to be transmitted transparently over the link, and to remove
   spurious control data which may be injected into the link by
   intervening hardware and software.

   Some protocols expect error free transmission, and either provide
   error detection only on a conditional basis, or do not provide it at
   all.  PPP uses the HDLC Frame Check Sequence for error detection.
   This is commonly available in hardware implementations, and a
   software implementation is provided.

1.1.  Specification of Requirements

   In this document, several words are used to signify the requirements
   of the specification.  These words are often capitalized.

   MUST       This word, or the adjective "required", means that the
              definition is an absolute requirement of the specification.

   MUST NOT   This phrase means that the definition is an absolute
              prohibition of the specification.

   SHOULD     This word, or the adjective "recommended", means that there
              may exist valid reasons in particular circumstances to
              ignore this item, but the full implications must be
              understood and carefully weighed before choosing a
              different course.

   MAY        This word, or the adjective "optional", means that this
              item is one of an allowed set of alternatives.  An
              implementation which does not include this option MUST be
              prepared to interoperate with another implementation which
              does include the option.


1.2.  Terminology

   This document frequently uses the following terms:

   datagram   The unit of transmission in the network layer (such as IP).
              A datagram may be encapsulated in one or more packets
              passed to the data link layer.

   frame      The unit of transmission at the data link layer.  A frame
              may include a header and/or a trailer, along with some
              number of units of data.

   packet     The basic unit of encapsulation, which is passed across the
              interface between the network layer and the data link
              layer.  A packet is usually mapped to a frame; the
              exceptions are when data link layer fragmentation is being
              performed, or when multiple packets are incorporated into a
              single frame.

   peer       The other end of the point-to-point link.

   silently discard
              The implementation discards the packet without further
              processing.  The implementation SHOULD provide the
              capability of logging the error, including the contents of
              the silently discarded packet, and SHOULD record the event
              in a statistics counter.

2.  Physical Layer Requirements

   PPP is capable of operating across most DTE/DCE interfaces (such as,
   EIA RS-232-E, EIA RS-422, and CCITT V.35).  The only absolute
   requirement imposed by PPP is the provision of a full-duplex circuit,
   either dedicated or circuit-switched, which can operate in either an
   asynchronous (start/stop), bit-synchronous, or octet-synchronous
   mode, transparent to PPP Data Link Layer frames.

   Interface Format

      PPP presents an octet interface to the physical layer.  There is
      no provision for sub-octets to be supplied or accepted.

   Transmission Rate

      PPP does not impose any restrictions regarding transmission rate,
      other than that of the particular DTE/DCE interface.

   Control Signals

      PPP does not require the use of control signals, such as Request
      To Send (RTS), Clear To Send (CTS), Data Carrier Detect (DCD), and
      Data Terminal Ready (DTR).

      When available, using such signals can allow greater functionality
      and performance.  In particular, such signals SHOULD be used to
      signal the Up and Down events in the LCP Option Negotiation
      Automaton [1].  When such signals are not available, the
      implementation MUST signal the Up event to LCP upon
      initialization, and SHOULD NOT signal the Down event.

      Because signalling is not required, the physical layer MAY be
      decoupled from the data link layer, hiding the transient details
      of the physical transport.  This has implications for mobility in
      cellular radio networks, and other rapidly switching links.

      When moving from cell to cell within the same zone, an
      implementation MAY choose to treat the entire zone as a single
      link, even though transmission is switched among several
      frequencies.  The link is considered to be with the central
      control unit for the zone, rather than the individual cell
      transceivers.  However, the link SHOULD re-establish its
      configuration whenever the link is switched to a different
      administration.

      Due to the bursty nature of data traffic, some implementations
      have choosen to disconnect the physical layer during periods of

inactivity, and reconnect when traffic resumes, without informing
the data link layer.  Robust implementations should avoid using
this trick over-zealously, since the price for decreased setup
latency is decreased security.  Implementations SHOULD signal the
Down event whenever "significant time" has elapsed since the link
was disconnected.  The value for "significant time" is a matter of
considerable debate, and is based on the tariffs, call setup
times, and security concerns of the installation.


3.  The Data Link Layer

PPP uses the principles described in ISO 3309-1979 HDLC frame
structure, most recently the fourth edition 3309:1991 [2], which
specifies modifications to allow HDLC use in asynchronous
environments.

The PPP control procedures use the Control field encodings described
in ISO 4335-1979 HDLC elements of procedures, most recently the
fourth edition 4335:1991 [4].

This should not be construed to indicate that every feature of the
above recommendations are included in PPP.  Each feature included
is explicitly described in the following sections.

To remain consistent with standard Internet practice, and avoid
confusion for people used to reading RFCs, all binary numbers in the
following descriptions are in Most Significant Bit to Least
Significant Bit order, reading from left to right, unless otherwise
indicated.  Note that this is contrary to standard ISO and CCITT
practice which orders bits as transmitted (network bit order).  Keep
this in mind when comparing this document with the international
standards documents.

3.1.  Frame Format

   A summary of the PPP HDLC-like frame structure is shown below.  This
   figure does not include bits inserted for synchronization (such as
   start and stop bits for asynchronous links), nor any bits or octets
   inserted for transparency.  The fields are transmitted from left to
   right.

```
          +----------+----------+----------+
          |   Flag   | Address  | Control  |
          | 01111110 | 11111111 | 00000011 |
          +----------+----------+----------+

          +---------+-------------+---------+
          | Protocol | Information | Padding |
          | 8/16 bits|     *       |    *    |
          +---------+-------------+---------+

          +---------+----------+-----------------
          |   FCS    |   Flag   | Inter-frame Fill
          |16/32 bits| 01111110 | or next Address
          +---------+----------+-----------------
```

   The Protocol, Information and Padding fields are described in the
   Point-to-Point Protocol Encapsulation [1].

Flag Sequence

   Each frame begins and ends with a Flag Sequence, which is the
   binary sequence 01111110 (hexadecimal 0x7e).  All implementations
   continuously check for this flag, which is used for frame
   synchronization.

   Only one Flag Sequence is required between two frames.  Two
   consecutive Flag Sequences constitute an empty frame, which is
   silently discarded, and not counted as a FCS error.

Address Field

   The Address field is a single octet, which contains the binary
   sequence 11111111 (hexadecimal 0xff), the All-Stations address.
   Individual station addresses are not assigned.  The All-Stations
   address MUST always be recognized and received.

   The use of other address lengths and values may be defined at a
   later time, or by prior agreement.  Frames with unrecognized
   Addresses SHOULD be silently discarded.

   Control Field

      The Control field is a single octet, which contains the binary
      sequence 00000011 (hexadecimal 0x03), the Unnumbered Information
      (UI) command with the Poll/Final (P/F) bit set to zero.

      The use of other Control field values may be defined at a later
      time, or by prior agreement.  Frames with unrecognized Control
      field values SHOULD be silently discarded.

   Frame Check Sequence (FCS) Field

      The Frame Check Sequence field defaults to 16 bits (two octets).
      The FCS is transmitted least significant octet first, which
      contains the coefficient of the highest term.

      A 32-bit (four octet) FCS is also defined.  Its use may be
      negotiated as described in "PPP LCP Extensions" [5].

      The use of other FCS lengths may be defined at a later time, or by
      prior agreement.

      The FCS field is calculated over all bits of the Address, Control,
      Protocol, Information and Padding fields, not including any start
      and stop bits (asynchronous) nor any bits (synchronous) or octets
      (asynchronous or synchronous) inserted for transparency.  This
      also does not include the Flag Sequences nor the FCS field itself.

         When octets are received which are flagged in the Async-
         Control-Character-Map, they are discarded before calculating
         the FCS.

      For more information on the specification of the FCS, see the
      Appendices.

   The end of the Information and Padding fields is found by locating
   the closing Flag Sequence and removing the Frame Check Sequence
   field.

3.2.  Modification of the Basic Frame

   The Link Control Protocol can negotiate modifications to the standard
   HDLC-like frame structure.  However, modified frames will always be
   clearly distinguishable from standard frames.

   Address-and-Control-Field-Compression

      When using the standard HDLC-like framing, the Address and Control
      fields contain the hexadecimal values 0xff and 0x03 respectively.
      When other Address or Control field values are in use, Address-
      and-Control-Field-Compression MUST NOT be negotiated.

      On transmission, compressed Address and Control fields are simply
      omitted.

      On reception, the Address and Control fields are decompressed by
      examining the first two octets.  If they contain the values 0xff
      and 0x03, they are assumed to be the Address and Control fields.
      If not, it is assumed that the fields were compressed and were not
      transmitted.

         By definition, the first octet of a two octet Protocol field
         will never be 0xff (since it is not even).  The Protocol field
         value 0x00ff is not allowed (reserved) to avoid ambiguity when
         Protocol-Field-Compression is enabled and the first Information
         field octet is 0x03.

4.  Octet-stuffed framing

   This chapter summarizes the use of HDLC-like framing with 8-bit
   asynchronous and octet-synchronous links.


4.1.  Flag Sequence

   The Flag Sequence indicates the beginning or end of a frame.  The
   octet stream is examined on an octet-by-octet basis for the value
   01111110 (hexadecimal 0x7e).


4.2.  Transparency

   An octet stuffing procedure is used.  The Control Escape octet is
   defined as binary 01111101 (hexadecimal 0x7d), most significant bit
   first.

   As a minimum, sending implementations MUST escape the Flag Sequence
   and Control Escape octets.

   After FCS computation, the transmitter examines the entire frame
   between the two Flag Sequences.  Each Flag Sequence, Control Escape
   octet, and any octet which is flagged in the sending Async-Control-
   Character-Map (ACCM), is replaced by a two octet sequence consisting
   of the Control Escape octet followed by the original octet
   exclusive-or'd with hexadecimal 0x20.

      This is bit 5 complemented, where the bit positions are numbered
      76543210 (the 6th bit as used in ISO numbered 87654321 -- BEWARE
      when comparing documents).

   Receiving implementations MUST correctly process all Control Escape
   sequences.

   On reception, prior to FCS computation, each octet with value less
   than hexadecimal 0x20 is checked.  If it is flagged in the receiving
   ACCM, it is simply removed (it may have been inserted by intervening
   data communications equipment).  Each Control Escape octet is also
   removed, and the following octet is exclusive-or'd with hexadecimal
   0x20, unless it is the Flag Sequence (which aborts a frame).

   A few examples may make this more clear.  Escaped data is transmitted
   on the link as follows:

```
   0x7e is encoded as 0x7d, 0x5e.    (Flag Sequence)
   0x7d is encoded as 0x7d, 0x5d.    (Control Escape)
   0x03 is encoded as 0x7d, 0x23.    (ETX)
```

Some modems with software flow control may intercept outgoing DC1 and
DC3 ignoring the 8th (parity) bit.  This data would be transmitted on
the link as follows:

```
   0x11 is encoded as 0x7d, 0x31.    (XON)
   0x13 is encoded as 0x7d, 0x33.    (XOFF)
   0x91 is encoded as 0x7d, 0xb1.    (XON with parity set)
   0x93 is encoded as 0x7d, 0xb3.    (XOFF with parity set)
```

4.3.  Invalid Frames

   Frames which are too short (less than 4 octets when using the 16-bit
   FCS), or which end with a Control Escape octet followed immediately
   by a closing Flag Sequence, or in which octet-framing is violated (by
   transmitting a "0" stop bit where a "1" bit is expected), are
   silently discarded, and not counted as a FCS error.

4.4.  Time Fill

4.4.1.  Octet-synchronous

   There is no provision for inter-octet time fill.

   The Flag Sequence MUST be transmitted during inter-frame time fill.

4.4.2.  Asynchronous

   Inter-octet time fill MUST be accomplished by transmitting continuous
   "1" bits (mark-hold state).

   Inter-frame time fill can be viewed as extended inter-octet time
   fill.  Doing so can save one octet for every frame, decreasing delay
   and increasing bandwidth.  This is possible since a Flag Sequence may
   serve as both a frame end and a frame begin.  After having received
   any frame, an idle receiver will always be in a frame begin state.

Robust transmitters should avoid using this trick over-zealously, since the price for decreased delay is decreased reliability.  Noisy links may cause the receiver to receive garbage characters and interpret them as part of an incoming frame.  If the transmitter does not send a new opening Flag Sequence before sending the next frame, then that frame will be appended to the noise characters causing an invalid frame (with high reliability).

It is suggested that implementations will achieve the best results by always sending an opening Flag Sequence if the new frame is not back-to-back with the last.  Transmitters SHOULD send an open Flag Sequence whenever "appreciable time" has elapsed after the prior closing Flag Sequence.  The maximum value for "appreciable time" is likely to be no greater than the typing rate of a slow typist, about 1 second.


4.5.  Transmission Considerations

4.5.1.  Octet-synchronous

The definition of various encodings and scrambling is the responsibility of the DTE/DCE equipment in use, and is outside the scope of this specification.


4.5.2.  Asynchronous

All octets are transmitted least significant bit first, with one start bit, eight bits of data, and one stop bit.  There is no provision for seven bit asynchronous links.

5.  Bit-stuffed framing

   This chapter summarizes the use of HDLC-like framing with bit-
   synchronous links.


5.1.  Flag Sequence

   The Flag Sequence indicates the beginning or end of a frame, and is
   used for frame synchronization.  The bit stream is examined on a
   bit-by-bit basis for the binary sequence 01111110 (hexadecimal 0x7e).

   The "shared zero mode" Flag Sequence "011111101111110" SHOULD NOT be
   used.  When not avoidable, such an implementation MUST ensure that
   the first Flag Sequence detected (the end of the frame) is promptly
   communicated to the link layer.  Use of the shared zero mode hinders
   interoperability with bit-synchronous to asynchronous and bit-
   synchronous to octet-synchronous converters.


5.2.  Transparency

   After FCS computation, the transmitter examines the entire frame
   between the two Flag Sequences.  A "0" bit is inserted after all
   sequences of five contiguous "1" bits (including the last 5 bits of
   the FCS) to ensure that a Flag Sequence is not simulated.

   On reception, prior to FCS computation, any "0" bit that directly
   follows five contiguous "1" bits is discarded.


5.3.  Invalid Frames

   Frames which are too short (less than 4 octets when using the 16-bit
   FCS), or which end with a sequence of more than six "1" bits, are
   silently discarded, and not counted as a FCS error.


5.4.  Time Fill

   There is no provision for inter-octet time fill.

   The Flag Sequence SHOULD be transmitted during inter-frame time fill.
   However, certain types of circuit-switched links require the use of

mark idle (continuous ones), particularly those that calculate
accounting based on periods of bit activity.  When mark idle is used
on a bit-synchronous link, the implementation MUST ensure at least 15
consecutive "1" bits between Flags during the idle period, and that
the Flag Sequence is always generated at the beginning of a frame
after an idle period.

   This differs from practice in ISO 3309, which allows 7 to 14 bit
   mark idle.


5.5.  Transmission Considerations

   All octets are transmitted least significant bit first.

   The definition of various encodings and scrambling is the
   responsibility of the DTE/DCE equipment in use, and is outside the
   scope of this specification.

   While PPP will operate without regard to the underlying
   representation of the bit stream, lack of standards for transmission
   will hinder interoperability as surely as lack of data link
   standards.  At speeds of 56 Kbps through 2.0 Mbps, NRZ is currently
   most widely available, and on that basis is recommended as a default.

   When configuration of the encoding is allowed, NRZI is recommended as
   an alternative, because of its relative immunity to signal inversion
   configuration errors, and instances when it MAY allow connection
   without an expensive DSU/CSU.  Unfortunately, NRZI encoding
   exacerbates the missing x1 factor of the 16-bit FCS, so that one
   error in 2**15 goes undetected (instead of one in 2**16), and triple
   errors are not detected.  Therefore, when NRZI is in use, it is
   recommended that the 32-bit FCS be negotiated, which includes the x1
   factor.

   At higher speeds of up to 45 Mbps, some implementors have chosen the
   ANSI High Speed Synchronous Interface [HSSI].  While this experience
   is currently limited, implementors are encouraged to cooperate in
   choosing transmission encoding.

6.  Asynchronous to Synchronous Conversion

   There may be some use of asynchronous-to-synchronous converters (some
   built into modems and cellular interfaces), resulting in an
   asynchronous PPP implementation on one end of a link and a
   synchronous implementation on the other.  It is the responsibility of
   the converter to do all stuffing conversions during operation.

   To enable this functionality, synchronous PPP implementations MUST
   always respond to the Async-Control-Character-Map Configuration
   Option with the LCP Configure-Ack.  However, acceptance of the
   Configuration Option does not imply that the synchronous
   implementation will do any ACCM mapping.  Instead, all such octet
   mapping will be performed by the asynchronous-to-synchronous
   converter.

7.  Additional LCP Configuration Options

    The Configuration Option format and basic options are already defined
    for LCP [1].

    Up-to-date values of the LCP Option Type field are specified in the
    most recent "Assigned Numbers" RFC [10].  This document concerns the
    following values:

        2          Async-Control-Character-Map

7.1.  Async-Control-Character-Map (ACCM)

    Description

        This Configuration Option provides a method to negotiate the use
        of control character transparency on asynchronous links.

        Each end of the asynchronous link maintains two Async-Control-
        Character-Maps.  The receiving ACCM is 32 bits, but the sending
        ACCM may be up to 256 bits.  This results in four distinct ACCMs,
        two in each direction of the link.

        For asynchronous links, the default receiving ACCM is 0xffffffff.
        The default sending ACCM is 0xffffffff, plus the Control Escape
        and Flag Sequence characters themselves, plus whatever other
        outgoing characters are flagged (by prior configuration) as likely
        to be intercepted.

        For other types of links, the default value is 0, since there is
        no need for mapping.

            The default inclusion of all octets less than hexadecimal 0x20
            allows all ASCII control characters [6] excluding DEL (Delete)
            to be transparently communicated through all known data
            communications equipment.

        The transmitter MAY also send octets with values in the range 0x40
        through 0xff (except 0x5e) in Control Escape format.  Since these
        octet values are not negotiable, this does not solve the problem
        of receivers which cannot handle all non-control characters.
        Also, since the technique does not affect the 8th bit, this does
        not solve problems for communications links that can send only 7-
        bit characters.

     Note that this specification differs in detail from later
     amendments, such as 3309:1991/Amendment 2 [3].  However, such
     "extended transparency" is applied only by "prior agreement".
     Use of the transparency methods in this specification
     constitute a prior agreement with respect to PPP.
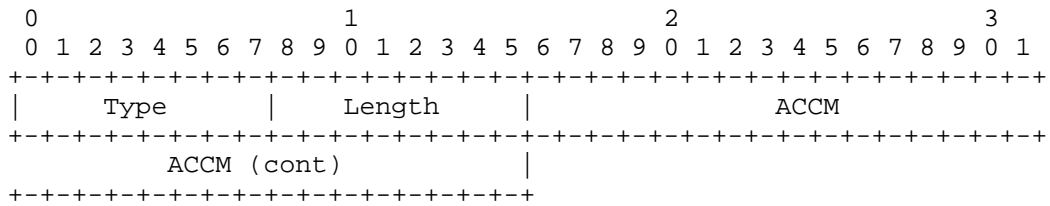
     For compatibility with 3309:1991/Amendment 2, the transmitter
     MAY escape DEL and ACCM equivalents with the 8th (most
     significant) bit set.  No change is required in the receiving
     algorithm.

     Following ACCM negotiation, the transmitter SHOULD cease
     escaping DEL.

  However, it is rarely necessary to map all control characters, and
  often it is unnecessary to map any control characters.  The
  Configuration Option is used to inform the peer which control
  characters MUST remain mapped when the peer sends them.

  The peer MAY still send any other octets in mapped format, if it
  is necessary because of constraints known to the peer.  The peer
  SHOULD Configure-Nak with the logical union of the sets of mapped
  octets, so that when such octets are spuriously introduced they
  can be ignored on receipt.

A summary of the Async-Control-Character-Map Configuration Option
format is shown below.  The fields are transmitted from left to
right.

```
 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      |    Length     |              ACCM
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        ACCM (cont)            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```


Type

    2

Length

    6

   ACCM

      The ACCM field is four octets, and indicates the set of control
      characters to be mapped.  The map is sent most significant octet
      first.

      Each numbered bit corresponds to the octet of the same value.  If
      the bit is cleared to zero, then that octet need not be mapped.
      If the bit is set to one, then that octet MUST remain mapped.  For
      example, if bit 19 is set to zero, then the ASCII control
      character 19 (DC3, Control-S) MAY be sent in the clear.

         Note: The least significant bit of the least significant octet
         (the final octet transmitted) is numbered bit 0, and would map
         to the ASCII control character NUL.

A.  Recommended LCP Options

    The following Configurations Options are recommended:

    High Speed links

        Magic Number
        Link Quality Monitoring
        No Address and Control Field Compression
        No Protocol Field Compression


    Low Speed or Asynchronous links

        Async Control Character Map
        Magic Number
        Address and Control Field Compression
        Protocol Field Compression



B.  Automatic Recognition of PPP Frames

    It is sometimes desirable to detect PPP frames, for example during a
    login sequence.  The following octet sequences all begin valid PPP
    LCP frames:

        7e ff 03 c0 21
        7e ff 7d 23 c0 21
        7e 7d df 7d 23 c0 21

    Note that the first two forms are not a valid username for Unix.
    However, only the third form generates a correctly checksummed PPP
    frame, whenever 03 and ff are taken as the control characters ETX and
    DEL without regard to parity (they are correct for an even parity
    link) and discarded.

    Many implementations deal with this by putting the interface into
    packet mode when one of the above username patterns are detected
    during login, without examining the initial PPP checksum.  The
    initial incoming PPP frame is discarded, but a Configure-Request is
    sent immediately.

C.  Fast Frame Check Sequence (FCS) Implementation

   The FCS was originally designed with hardware implementations in
   mind.  A serial bit stream is transmitted on the wire, the FCS is
   calculated over the serial data as it goes out, and the complement of
   the resulting FCS is appended to the serial stream, followed by the
   Flag Sequence.

   The receiver has no way of determining that it has finished
   calculating the received FCS until it detects the Flag Sequence.
   Therefore, the FCS was designed so that a particular pattern results
   when the FCS operation passes over the complemented FCS.  A good
   frame is indicated by this "good FCS" value.


C.1.  FCS table generator

   The following code creates the lookup table used to calculate the
   FCS-16.

```
/*
 * Generate a FCS-16 table.
 *
 * Drew D. Perkins at Carnegie Mellon University.
 *
 * Code liberally borrowed from Mohsen Banan and D. Hugh Redelmeier.
 */

/*
 * The FCS-16 generator polynomial: x**0 + x**5 + x**12 + x**16.
 */
#define P       0x8408


main()
{
    register unsigned int b, v;
    register int i;

    printf("typedef unsigned short u16;\n");
    printf("static u16 fcstab[256] = {");
    for (b = 0; ; ) {
        if (b % 8 == 0)
            printf("\n");

        v = b;
        for (i = 8; i--; )
```

```
              v = v & 1 ? (v >> 1) ^ P : v >> 1;

          printf("\t0x%04x", v & 0xFFFF);
          if (++b == 256)
              break;
          printf(",");
      }
   printf("\n};\n");
   }
```

C.2.  16-bit FCS Computation Method

   The following code provides a table lookup computation for
   calculating the Frame Check Sequence as data arrives at the
   interface.  This implementation is based on [7], [8], and [9].

```
   /*
    * u16 represents an unsigned 16-bit number.  Adjust the typedef for
    * your hardware.
    */
   typedef unsigned short u16;

   /*
    * FCS lookup table as calculated by the table generator.
    */
   static u16 fcstab[256] = {
      0x0000, 0x1189, 0x2312, 0x329b, 0x4624, 0x57ad, 0x6536, 0x74bf,
      0x8c48, 0x9dc1, 0xaf5a, 0xbed3, 0xca6c, 0xdbe5, 0xe97e, 0xf8f7,
      0x1081, 0x0108, 0x3393, 0x221a, 0x56a5, 0x472c, 0x75b7, 0x643e,
      0x9cc9, 0x8d40, 0xbfdb, 0xae52, 0xdaed, 0xcb64, 0xf9ff, 0xe876,
      0x2102, 0x308b, 0x0210, 0x1399, 0x6726, 0x76af, 0x4434, 0x55bd,
      0xad4a, 0xbcc3, 0x8e58, 0x9fd1, 0xeb6e, 0xfae7, 0xc87c, 0xd9f5,
      0x3183, 0x200a, 0x1291, 0x0318, 0x77a7, 0x662e, 0x54b5, 0x453c,
      0xbdcb, 0xac42, 0x9ed9, 0x8f50, 0xfbef, 0xea66, 0xd8fd, 0xc974,
      0x4204, 0x538d, 0x6116, 0x709f, 0x0420, 0x15a9, 0x2732, 0x36bb,
      0xce4c, 0xdfc5, 0xed5e, 0xfcd7, 0x8868, 0x99e1, 0xab7a, 0xbaf3,
      0x5285, 0x430c, 0x7197, 0x601e, 0x14a1, 0x0528, 0x37b3, 0x263a,
      0xdecd, 0xcf44, 0xfddf, 0xec56, 0x98e9, 0x8960, 0xbbfb, 0xaa72,
      0x6306, 0x728f, 0x4014, 0x519d, 0x2522, 0x34ab, 0x0630, 0x17b9,
      0xef4e, 0xfec7, 0xcc5c, 0xddd5, 0xa96a, 0xb8e3, 0x8a78, 0x9bf1,
      0x7387, 0x620e, 0x5095, 0x411c, 0x35a3, 0x242a, 0x16b1, 0x0738,
      0xffcf, 0xee46, 0xdcdd, 0xcd54, 0xb9eb, 0xa862, 0x9af9, 0x8b70,
      0x8408, 0x9581, 0xa71a, 0xb693, 0xc22c, 0xd3a5, 0xe13e, 0xf0b7,
      0x0840, 0x19c9, 0x2b52, 0x3adb, 0x4e64, 0x5fed, 0x6d76, 0x7cff,
      0x9489, 0x8500, 0xb79b, 0xa612, 0xd2ad, 0xc324, 0xf1bf, 0xe036,
      0x18c1, 0x0948, 0x3bd3, 0x2a5a, 0x5ee5, 0x4f6c, 0x7df7, 0x6c7e,
```

```
        0xa50a, 0xb483, 0x8618, 0x9791, 0xe32e, 0xf2a7, 0xc03c, 0xd1b5,
        0x2942, 0x38cb, 0x0a50, 0x1bd9, 0x6f66, 0x7eef, 0x4c74, 0x5dfd,
        0xb58b, 0xa402, 0x9699, 0x8710, 0xf3af, 0xe226, 0xd0bd, 0xc134,
        0x39c3, 0x284a, 0x1ad1, 0x0b58, 0x7fe7, 0x6e6e, 0x5cf5, 0x4d7c,
        0xc60c, 0xd785, 0xe51e, 0xf497, 0x8028, 0x91a1, 0xa33a, 0xb2b3,
        0x4a44, 0x5bcd, 0x6956, 0x78df, 0x0c60, 0x1de9, 0x2f72, 0x3efb,
        0xd68d, 0xc704, 0xf59f, 0xe416, 0x90a9, 0x8120, 0xb3bb, 0xa232,
        0x5ac5, 0x4b4c, 0x79d7, 0x685e, 0x1ce1, 0x0d68, 0x3ff3, 0x2e7a,
        0xe70e, 0xf687, 0xc41c, 0xd595, 0xa12a, 0xb0a3, 0x8238, 0x93b1,
        0x6b46, 0x7acf, 0x4854, 0x59dd, 0x2d62, 0x3ceb, 0x0e70, 0x1ff9,
        0xf78f, 0xe606, 0xd49d, 0xc514, 0xb1ab, 0xa022, 0x92b9, 0x8330,
        0x7bc7, 0x6a4e, 0x58d5, 0x495c, 0x3de3, 0x2c6a, 0x1ef1, 0x0f78
   };

   #define PPPINITFCS16    0xffff  /* Initial FCS value */
   #define PPPGOODFCS16    0xf0b8  /* Good final FCS value */

   /*
    * Calculate a new fcs given the current fcs and the new data.
    */
   u16 pppfcs16(fcs, cp, len)
       register u16 fcs;
       register unsigned char *cp;
       register int len;
   {
       ASSERT(sizeof (u16) == 2);
       ASSERT(((u16) -1) > 0);
       while (len--)
           fcs = (fcs >> 8) ^ fcstab[(fcs ^ *cp++) & 0xff];

       return (fcs);
   }

   /*
    * How to use the fcs
    */
   tryfcs16(cp, len)
       register unsigned char *cp;
       register int len;
   {
       u16 trialfcs;

       /* add on output */
       trialfcs = pppfcs16( PPPINITFCS16, cp, len );
       trialfcs ^= 0xffff;                 /* complement */
       cp[len] = (trialfcs & 0x00ff);      /* least significant byte first */
       cp[len+1] = ((trialfcs >> 8) & 0x00ff);
```

```
       /* check on input */
       trialfcs = pppfcs16( PPPINITFCS16, cp, len + 2 );
       if ( trialfcs == PPPGOODFCS16 )
           printf("Good FCS\n");
   }
```

C.3.  32-bit FCS Computation Method

   The following code provides a table lookup computation for
   calculating the 32-bit Frame Check Sequence as data arrives at the
   interface.

```
   /*
    * The FCS-32 generator polynomial: x**0 + x**1 + x**2 + x**4 + x**5
    *                     + x**7 + x**8 + x**10 + x**11 + x**12 + x**16
    *                     + x**22 + x**23 + x**26 + x**32.
    */

   /*
    * u32 represents an unsigned 32-bit number.  Adjust the typedef for
    * your hardware.
    */
   typedef unsigned long u32;

   static u32 fcstab_32[256] =
      {
      0x00000000, 0x77073096, 0xee0e612c, 0x990951ba,
      0x076dc419, 0x706af48f, 0xe963a535, 0x9e6495a3,
      0x0edb8832, 0x79dcb8a4, 0xe0d5e91e, 0x97d2d988,
      0x09b64c2b, 0x7eb17cbd, 0xe7b82d07, 0x90bf1d91,
      0x1db71064, 0x6ab020f2, 0xf3b97148, 0x84be41de,
      0x1adad47d, 0x6ddde4eb, 0xf4d4b551, 0x83d385c7,
      0x136c9856, 0x646ba8c0, 0xfd62f97a, 0x8a65c9ec,
      0x14015c4f, 0x63066cd9, 0xfa0f3d63, 0x8d080df5,
      0x3b6e20c8, 0x4c69105e, 0xd56041e4, 0xa2677172,
      0x3c03e4d1, 0x4b04d447, 0xd20d85fd, 0xa50ab56b,
      0x35b5a8fa, 0x42b2986c, 0xdbbbc9d6, 0xacbcf940,
      0x32d86ce3, 0x45df5c75, 0xdcd60dcf, 0xabd13d59,
      0x26d930ac, 0x51de003a, 0xc8d75180, 0xbfd06116,
      0x21b4f4b5, 0x56b3c423, 0xcfba9599, 0xb8bda50f,
      0x2802b89e, 0x5f058808, 0xc60cd9b2, 0xb10be924,
      0x2f6f7c87, 0x58684c11, 0xc1611dab, 0xb6662d3d,
      0x76dc4190, 0x01db7106, 0x98d220bc, 0xefd5102a,
      0x71b18589, 0x06b6b51f, 0x9fbfe4a5, 0xe8b8d433,
      0x7807c9a2, 0x0f00f934, 0x9609a88e, 0xe10e9818,
      0x7f6a0dbb, 0x086d3d2d, 0x91646c97, 0xe6635c01,
```

```
      0x6b6b51f4, 0x1c6c6162, 0x856530d8, 0xf262004e,
      0x6c0695ed, 0x1b01a57b, 0x8208f4c1, 0xf50fc457,
      0x65b0d9c6, 0x12b7e950, 0x8bbeb8ea, 0xfcb9887c,
      0x62dd1ddf, 0x15da2d49, 0x8cd37cf3, 0xfbd44c65,
      0x4db26158, 0x3ab551ce, 0xa3bc0074, 0xd4bb30e2,
      0x4adfa541, 0x3dd895d7, 0xa4d1c46d, 0xd3d6f4fb,
      0x4369e96a, 0x346ed9fc, 0xad678846, 0xda60b8d0,
      0x44042d73, 0x33031de5, 0xaa0a4c5f, 0xdd0d7cc9,
      0x5005713c, 0x270241aa, 0xbe0b1010, 0xc90c2086,
      0x5768b525, 0x206f85b3, 0xb966d409, 0xce61e49f,
      0x5edef90e, 0x29d9c998, 0xb0d09822, 0xc7d7a8b4,
      0x59b33d17, 0x2eb40d81, 0xb7bd5c3b, 0xc0ba6cad,
      0xedb88320, 0x9abfb3b6, 0x03b6e20c, 0x74b1d29a,
      0xead54739, 0x9dd277af, 0x04db2615, 0x73dc1683,
      0xe3630b12, 0x94643b84, 0x0d6d6a3e, 0x7a6a5aa8,
      0xe40ecf0b, 0x9309ff9d, 0x0a00ae27, 0x7d079eb1,
      0xf00f9344, 0x8708a3d2, 0x1e01f268, 0x6906c2fe,
      0xf762575d, 0x806567cb, 0x196c3671, 0x6e6b06e7,
      0xfed41b76, 0x89d32be0, 0x10da7a5a, 0x67dd4acc,
      0xf9b9df6f, 0x8ebeeff9, 0x17b7be43, 0x60b08ed5,
      0xd6d6a3e8, 0xa1d1937e, 0x38d8c2c4, 0x4fdff252,
      0xd1bb67f1, 0xa6bc5767, 0x3fb506dd, 0x48b2364b,
      0xd80d2bda, 0xaf0a1b4c, 0x36034af6, 0x41047a60,
      0xdf60efc3, 0xa867df55, 0x316e8eef, 0x4669be79,
      0xcb61b38c, 0xbc66831a, 0x256fd2a0, 0x5268e236,
      0xcc0c7795, 0xbb0b4703, 0x220216b9, 0x5505262f,
      0xc5ba3bbe, 0xb2bd0b28, 0x2bb45a92, 0x5cb36a04,
      0xc2d7ffa7, 0xb5d0cf31, 0x2cd99e8b, 0x5bdeae1d,
      0x9b64c2b0, 0xec63f226, 0x756aa39c, 0x026d930a,
      0x9c0906a9, 0xeb0e363f, 0x72076785, 0x05005713,
      0x95bf4a82, 0xe2b87a14, 0x7bb12bae, 0x0cb61b38,
      0x92d28e9b, 0xe5d5be0d, 0x7cdcefb7, 0x0bdbdf21,
      0x86d3d2d4, 0xf1d4e242, 0x68ddb3f8, 0x1fda836e,
      0x81be16cd, 0xf6b9265b, 0x6fb077e1, 0x18b74777,
      0x88085ae6, 0xff0f6a70, 0x66063bca, 0x11010b5c,
      0x8f659eff, 0xf862ae69, 0x616bffd3, 0x166ccf45,
      0xa00ae278, 0xd70dd2ee, 0x4e048354, 0x3903b3c2,
      0xa7672661, 0xd06016f7, 0x4969474d, 0x3e6e77db,
      0xaed16a4a, 0xd9d65adc, 0x40df0b66, 0x37d83bf0,
      0xa9bcae53, 0xdebb9ec5, 0x47b2cf7f, 0x30b5ffe9,
      0xbdbdf21c, 0xcabac28a, 0x53b39330, 0x24b4a3a6,
      0xbad03605, 0xcdd70693, 0x54de5729, 0x23d967bf,
      0xb3667a2e, 0xc4614ab8, 0x5d681b02, 0x2a6f2b94,
      0xb40bbe37, 0xc30c8ea1, 0x5a05df1b, 0x2d02ef8d
      };

   #define PPPINITFCS32  0xffffffff   /* Initial FCS value */
   #define PPPGOODFCS32  0xdebb20e3   /* Good final FCS value */
```

```
   /*
    * Calculate a new FCS given the current FCS and the new data.
    */
   u32 pppfcs32(fcs, cp, len)
       register u32 fcs;
       register unsigned char *cp;
       register int len;
       {
       ASSERT(sizeof (u32) == 4);
       ASSERT(((u32) -1) > 0);
       while (len--)
           fcs = (((fcs) >> 8) ^ fcstab_32[((fcs) ^ (*cp++)) & 0xff]);

       return (fcs);
       }

   /*
    * How to use the fcs
    */
   tryfcs32(cp, len)
       register unsigned char *cp;
       register int len;
   {
       u32 trialfcs;

       /* add on output */
       trialfcs = pppfcs32( PPPINITFCS32, cp, len );
       trialfcs ^= 0xffffffff;               /* complement */
       cp[len] = (trialfcs & 0x00ff);        /* least significant byte first */
       cp[len+1] = ((trialfcs >>= 8) & 0x00ff);
       cp[len+2] = ((trialfcs >>= 8) & 0x00ff);
       cp[len+3] = ((trialfcs >> 8) & 0x00ff);

       /* check on input */
       trialfcs = pppfcs32( PPPINITFCS32, cp, len + 4 );
       if ( trialfcs == PPPGOODFCS32 )
           printf("Good FCS\n");
   }
```

Security Considerations

   As noted in the Physical Layer Requirements section, the link layer
   might not be informed when the connected state of the physical layer
   has changed.  This results in possible security lapses due to over-
   reliance on the integrity and security of switching systems and
   administrations.  An insertion attack might be undetected.  An
   attacker which is able to spoof the same calling identity might be
   able to avoid link authentication.


References

   [1]   Simpson, W., Editor, "The Point-to-Point Protocol (PPP)",
         STD 50, RFC 1661, Daydreamer, July 1994.

   [2]   ISO/IEC 3309:1991(E), "Information Technology -
         Telecommunications and information exchange between systems -
         High-level data link control (HDLC) procedures - Frame
         structure", International Organization For Standardization,
         Fourth edition 1991-06-01.

   [3]   ISO/IEC 3309:1991/Amd.2:1992(E), "Information Technology -
         Telecommunications and information exchange between systems -
         High-level data link control (HDLC) procedures - Frame
         structure - Amendment 2: Extended transparency options for
         start/stop transmission", International Organization For
         Standardization, 1992-01-15.

   [4]   ISO/IEC 4335:1991(E), "Information Technology -
         Telecommunications and information exchange between systems -
         High-level data link control (HDLC) procedures - Elements of
         procedures", International Organization For Standardization,
         Fourth edition 1991-09-15.

   [5]   Simpson, W., Editor, "PPP LCP Extensions", RFC 1570,
         Daydreamer, January 1994.

   [6]   ANSI X3.4-1977, "American National Standard Code for
         Information Interchange", American National Standards
         Institute, 1977.

   [7]   Perez, "Byte-wise CRC Calculations", IEEE Micro, June 1983.

   [8]   Morse, G., "Calculating CRC's by Bits and Bytes", Byte,
         September 1986.

   [9]    LeVan, J., "A Fast CRC", Byte, November 1987.

   [10]   Reynolds, J., and J. Postel, "Assigned Numbers", STD 2, RFC
          1340, USC/Information Sciences Institute, July 1992.

Chair's Address

   The working group can be contacted via the current chair:

      Fred Baker
      Advanced Computer Communications
      315 Bollay Drive
      Santa Barbara, California  93117

      fbaker@acc.com


Editor's Address

   Questions about this memo can also be directed to:

      William Allen Simpson
      Daydreamer
      Computer Systems Consulting Services
      1384 Fontaine
      Madison Heights, Michigan  48071

      Bill.Simpson@um.cc.umich.edu
          bsimpson@MorningStar.com