        The Point-to-Point Protocol: A Proposal for Multi-Protocol
            Transmission of Datagrams Over Point-to-Point Links

                          Table of Contents

Status of this Memo

   This memo defines a proposed protocol for the Internet community.

   This proposal is the product of the Point-to-Point Protocol Working
   Group of the Internet Engineering Task Force (IETF). Comments on this
   memo should be submitted to the IETF Point-to-Point Protocol Working
   Group chair by January 15, 1990. Comments will be reviewed at the
   February 1990 IETF meeting, with the goal of advancing PPP to draft
   standard status. Distribution of this memo is unlimited.

Abstract

   The Point-to-Point Protocol (PPP) provides a method for transmitting
   datagrams over serial point-to-point links. PPP is composed of three
   parts:

      1. A method for encapsulating datagrams over serial links.

      2. An extensible Link Control Protocol (LCP).

      3. A family of Network Control Protocols (NCP) for establishing
         and configuring different network-layer protocols.

   This document defines the encapsulation scheme, the basic LCP, and an
   NCP for establishing and configuring the Internet Protocol (IP)
   (called the IP Control Protocol, IPCP).

   The options and facilities used by the LCP and the IPCP are defined
   in separate documents. Control protocols for configuring and
   utilizing other network-layer protocols besides IP (e.g., DECNET,
   OSI) are expected to be developed as needed.

1. Introduction

1.1. Motivation

   In the last few years, the Internet has seen explosive growth in the
   number of hosts supporting TCP/IP. The vast majority of these hosts
   are connected to Local Area Networks (LANs) of various types,
   Ethernet being the most common. Most of the other hosts are
   connected through Wide Area Networks (WANs) such as X.25 style Public
   Data Networks (PDNs). Relatively few of these hosts are connected
   with simple point-to-point (i.e., serial) links. Yet, point-to-point
   links are among the oldest methods of data communications and almost

every host supports point-to-point connections. For example,
asynchronous RS-232-C [1] interfaces are essentially ubiquitous.

One reason for the small number of point-to-point IP links is the
lack of a standard encapsulation protocol. There are plenty of non-
standard (and at least one defacto standard) encapsulation protocols
available, but there is not one which has been agreed upon as an
Internet Standard. By contrast, standard encapsulation schemes do
exist for the transmission of datagrams over most popular LANs.

One purpose of this memo is to remedy this problem. But even more
importantly, the Point-to-Point Protocol proposes more than just an
encapsulation scheme. Point-to-Point links tend to exacerbate many
problems with the current family of network protocols. For instance,
assignment and management of IP addresses, which is a problem even in
LAN environments, is especially difficult over switched point-to-
point circuits (e.g., dialups).

Some additional issues addressed by PPP include asynchronous
(start/stop) and bit-oriented synchronous encapsulation, network
protocol multiplexing, link configuration, link quality testing,
error detection, and option negotiation for such capabilities as
network-layer address negotiation and data compression negotiation.

PPP addresses these issues by providing an extensible Link Control
Protocol (LCP) and a family of Network Control Protocols (NCP) to
negotiate optional configuration parameters and facilities.

1.2. Overview of PPP

PPP has three main components:

    1. A method for encapsulating datagrams over serial links. PPP
       uses HDLC as a basis for encapsulating datagrams over point-
       to-point links.

    2. An extensible Link Control Protocol (LCP) to establish,
       configure, and test the data-link connection.

    3. A family of Network Control Protocols (NCP) for establishing
       and configuring different network-layer protocols. PPP is
       designed to allow the simultaneous use of multiple network-
       layer protocols.

In order to establish communications over a point-to-point link, the
originating PPP would first send LCP packets to configure and test
the data link. After the link has been establish and optional
facilities have been negotiated as needed by the LCP, the originating

PPP would send NCP packets to choose and configure one or more
network-layer protocols. Once each of the chosen network-layer
protocols has been configured, datagrams from each network-layer
protocol can be sent over the link.

The link will remain configured for communications until explicit LCP
or NCP packets close the link down, or until some external event
occurs (e.g., inactivity timer expires or user intervention).

1.3. Organization of the document

This memo is divided into several sections. Section 2 discusses the
physical-layer requirements of PPP. Section 3 describes the Data
Link Layer including the PPP frame format and data link encapsulation
scheme. Section 4 specifies the LCP including the connection
establishment and option negotiation procedures. Section 5 specifies
the IP Control Protocol (IPCP), which is the NCP for the Internet
Protocol, and describes the encapsulation of IP datagrams within PPP
packets. Appendix A summarizes important features of asynchronous
HDLC, and Appendix B describes an efficient table-lookup algorithm
for fast Frame Check Sequence (FCS) computation.

2. Physical Layer Requirements

The Point-to-Point Protocol is capable of operating across any
DTE/DCE interface (e.g., EIA RS-232-C, EIA RS-422, EIA RS-423 and
CCITT V.35). The only absolute requirement imposed by PPP is the
provision of a duplex circuit, either dedicated or switched, which
can operate in either an asynchronous (start/stop) or synchronous
bit-serial mode, transparent to PPP Data Link Layer frames. PPP does
not impose any restrictions regarding transmission rate, other than
those imposed by the particular DTE/DCE interface in use.

PPP does not require the use of modem control signals, such as
Request To Send (RTS), Clear To Send (CTS), Data Carrier Detect
(DCD), and Data Terminal Ready (DTR). However, using such signals
when available can allow greater functionality and performance.

3. The Data Link Layer

The Point-to-Point Protocol uses the principles, terminology, and
frame structure of the International Organization For
Standardization's (ISO) High-level Data Link Control (HDLC)
procedures (ISO 3309-1979 [2]), as modified by ISO 3309:1984/PDAD1
"Addendum 1: Start/stop transmission" [5]. ISO 3309-1979 specifies
the HDLC frame structure for use in synchronous environments. ISO
3309:1984/PDAD1 specifies proposed modifications to ISO 3309-1979 to
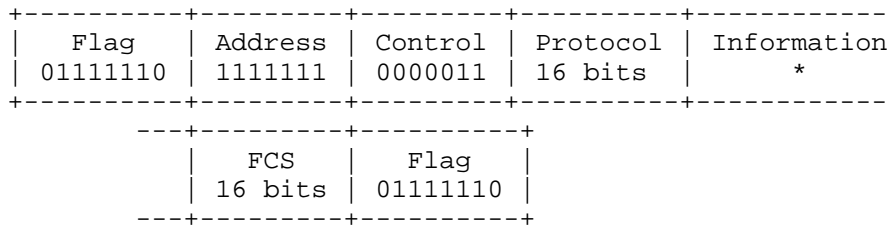allow its use in asynchronous environments.

The PPP control procedures use the definitions and Control field
encodings standardized in ISO 4335-1979 [3] and ISO 4335-
1979/Addendum 1-1979 [4]. The PPP frame structure is also consistent
with CCITT Recommendation X.25 LAPB [6], since that too is based on
HDLC.

    Note: ISO 3309:1984/PDAD1 is a Proposed Draft standard. At
    present, it seems that ISO 3309:1984/PDAD1 is stable and likely to
    become an International Standard. Therefore, we feel comfortable
    about using it before it becomes an International Standard. The
    progress of this proposal should be tracked and encouraged by the
    Internet community.

The purpose of this memo is not to document what is already
standardized in ISO 3309. We assume that the reader is already
familiar with HDLC, or has access to a copy of [2] or [6]. Instead,
this paper attempts to give a concise summary and point out specific
options and features used by PPP. Since "Addendum 1: Start/stop
transmission", is not yet standardized and widely available, it is
summarized in Appendix A.

3.1. Frame Format

A summary of the standard PPP frame structure is shown below. The
fields are transmitted from left to right.

```
    +----------+---------+---------+----------+------------
    |   Flag   | Address | Control | Protocol | Information
    | 01111110 | 1111111 | 0000011 | 16 bits  |     *
    +----------+---------+---------+----------+------------
          ---+---------+----------+
             |   FCS   |   Flag   |
             | 16 bits | 01111110 |
          ---+---------+----------+
```

This figure does not include start/stop bits (for asynchronous links)
or any bits or octets inserted for transparency. When asynchronous
links are used, all octets are transmitted with one start bit, eight
bits of data, and one stop bit. There is no provision in either PPP
or ISO 3309:1984/PDAD1 for seven bit asynchronous links.

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 (i.e., network bit order).
Keep this in mind when comparing this document with the international

standards documents.

Flag Sequence

   The Flag Sequence is a single octet and indicates the beginning or
   end of a frame. The Flag Sequence consists of the binary sequence
   01111110 (hexadecimal 0x7e).

Address Field

   The Address field is a single octet and contains the binary
   sequence 11111111 (hexadecimal 0xff), the All-Stations address.
   PPP does not assign individual station addresses. The All-
   Stations address should always be recognized and received. Frames
   with other Addresses should be silently discarded.

Control Field

   The Control field is a single octet and contains the binary
   sequence 00000011 (hexadecimal 0x03), the Unnumbered Information
   (UI) command with the P/F bit is set to zero. Frames with other
   Control field values should be silently discarded.

Protocol Field

   The Protocol field is two octets and its value identifies the
   protocol encapsulated in the Information field of the frame. The
   most up-to-date values of the Protocol field are specified in the
   most recent "Assigned Numbers" RFC [11]. Initial values are also
   listed below.

   Protocol field values in the "cxxx" range identify datagrams as
   belonging to the Link Control Protocol (LCP) or associated
   protocols. Values in the "8xxx" range identify datagrams belonging
   to the family of Network Control Protocols (NCP). Values in the
   "0xxx" range identify the network protocol of specific datagrams.

   This Protocol field is defined by PPP and is not a field defined
   by HDLC. However, the Protocol field is consistent with the ISO
   3309 extension mechanism for Address fields. All Protocols MUST be
   odd; the least significant bit of the least significant octet MUST
   equal "1". Also, all Protocols MUST be assigned such that the
   least significant bit of the most significant octet equals "0".
   Frames received which don't comply with these rules should be
   considered as having an unrecognized Protocol, and should be
   handled as specified by the LCP. The Protocol field is
   transmitted and received most significant octet first.

The Protocol field is initially assigned as follows:

```
Value (in hex)          Protocol

0001 to 001f            reserved (transparency inefficient)
0021                    Internet Protocol
0023                  * ISO CLNP
0025                  * Xerox NS IDP
0027                  * DECnet Phase IV
0029                  * Appletalk
002b                  * Novell IPX
002d                  * Van Jacobson Compressed TCP/IP 1
002f                  * Van Jacobson Compressed TCP/IP 2

8021                    Internet Protocol Control Protocol
8023                  * ISO CLNP Control Protocol
8025                  * Xerox NS IDP Control Protocol
8027                  * DECnet Phase IV Control Protocol
8029                  * Appletalk Control Protocol
802b                  * Novell IPX Control Protocol
802d                  * Reserved
802f                  * Reserved

c021                    Link Control Protocol
c023                  * User/Password Authentication Protocol
```

        * Reserved for future use; not described in this document.

  Information Field

    The Information field is zero or more octets. The Information
    field contains the datagram for the protocol specified in the
    Protocol field. The end of the Information field is found by
    locating the closing Flag Sequence and allowing two octets for the
    Frame Check Sequence field. The default maximum length of the
    Information field is 1500 octets. By prior agreement, consenting
    PPP implementations may use other values for the maximum
    Information field length.

    On transmission, the Information field may be padded with an
    arbitrary number of octets up to the maximum length. It is the
    responsibility of each protocol to disambiguate padding characters
    from real information.

  Frame Check Sequence (FCS) Field

    The Frame Check Sequence field is normally 16 bits (two octets).
    By prior agreement, consenting PPP implementations may use a 32-

bit (four-octet) FCS for improved error detection.

The FCS field is calculated over all bits of the Address, Control,
Protocol and Information fields not including any start and stop
bits (asynchronous) and any bits (synchronous) or octets
(asynchronous) inserted for transparency. This does not include
the Flag Sequences or FCS field. The FCS is transmitted with the
coefficient of the highest term first.

For more information on the specification of the FCS, see ISO 3309
or CCITT X.25.

   Note: A fast, table-driven implementation of the 16-bit FCS
   algorithm is shown in Appendix B. This implementation is based
   on [7] and [8].

Modifications to the Basic Frame Format

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

4. The PPP Link Control Protocol (LCP)

The Link Control Protocol (LCP) provides a method of establishing,
configuring, maintaining and terminating the point-to-point
connection. LCP goes through four distinct phases:

Phase 1: Link Establishment and Configuration Negotiation

Before any network-layer datagrams (e.g., IP) may be exchanged,
LCP must first open the connection through an exchange of
Configure packets. This exchange is complete, and the Open state
entered, once a Configure-Ack packet (described below) has been
both sent and received. Any non-LCP packets received before this
exchange is complete are silently discarded.

It is important to note that LCP handles configuration only of the
link; LCP does not handle configuration of individual network-
layer protocols. In particular, all Configuration Parameters
which are independent of particular network-layer protocols are
configured by LCP. All Configuration Options are assumed to be at
default values unless altered by the configuration exchange.

Phase 2: Link Quality Determination

LCP allows an optional Link Quality Determination phase following
transition to the LCP Open state. In this phase, the link is

tested to determine if the link quality is sufficient to bring up
network-layer protocols. This phase is completely optional. LCP
may delay transmission of network-layer protocol information until
this phase is completed.

The procedure for Link Quality Determination is unspecified and
may vary from implementation to implementation, or because of
user-configured parameters, but only so long as the procedure
doesn't violate other aspects of LCP. One suggested method is to
use LCP Echo-Request and Echo-Reply packets.

What is important is that this phase may persist for any length of
time. Therefore, implementations should avoid fixed timeouts when
waiting for their peers to advance to phase 3.

   Phase 3: Network-Layer Protocol Configuration Negotiation

Once LCP has finished the Link Quality Determination phase,
network-layer protocols may be separately configured by the
appropriate Network Control Protocols (NCP), and may be brought up
and taken down at any time. If LCP closes the link, it informs
the network-layer protocols so that they may take appropriate
action.

   Phase 4: Link Termination

LCP may terminate the link at any time. This will usually be done
at the request of a human user, but may happen because of a
physical event such as the loss of carrier, or the expiration of
an idle-period timer.

4.1. The LCP Automation

4.1.1. Overview

   LCP is specified by a number of packet formats and a finite-state
   automation. This section presents an overview of the LCP automation,
   followed by a representation of it as both a state diagram and a
   state transition table.

   There are three classes of LCP packets:

   1. Link Establishment packets used to establish and configure a
      link, (e.g., Configure-Request, Configure-Ack, Configure-Nak
      and Configure-Reject)

   2. Link Termination packets used to terminate a link, (e.g.,
      Terminate-Request and Terminate-Ack)

   3. Link Maintenance packets used to manage and debug a link,
      (e.g., Code-Reject, Protocol-Reject, Echo-Request, Echo-Reply
      and Discard-Request)

   The finite-state automation is defined by events, state transitions
   and actions. Events include receipt of external commands such as
   Open and Close, expiration of the Restart timer, and receipt of
   packets from a LCP peer. Actions include the starting of the Restart
   timer and transmission of packets.

4.1.2. State Diagram

   The state diagram which follows describes the sequence of events for
   reaching agreement on Configuration Options (opening the PPP
   connection) and for later closing of the connection. The state
   machine is initially in the Closed state (1). Once the Open state
   (6) has been reached, both ends of the link have met the requirement
   of having both sent and received a Configure-Ack packet.

   In the state diagram, events are shown above horizontal lines.
   Actions are shown below horizontal lines. Two types of LCP packets -
   Configure-Naks and Configure-Rejects - are not differentiated in the
   state diagram. As will be described later, these packets do indeed
   serve different, though similar, functions. However, at the level of
   detail of this state diagram, they always cause the same transition.

   Since a more detailed specification of the LCP automation is given in
   a state transition table in the following section, implementation
   should be done by consulting it rather than this state diagram.

```
                            +-----------------------------+
                            |                             |
                            V                             |
      +---2---+       PO +---1---+        RTA +---7---+    |
      |       |<-------------|       |<-----------|       |    |
      |Listen |    C         |Closed |       PLD  |Closing|    |
RCR   |       |              |       |<---Any     |       |    |
+-----|       |----->+------>|       |    State   +-------+ |<--+ |
|scr  +-------+      ^       +-------+            ^    | TO |    |
|              +----------+  AO  |                 |    +---->+    |
|              |          |  --- |                 |     str ^    |
|         C    |   RCN/TO    SCR |    C            |              |
|         --   | +-------->+<--------+ +-------------+              |
|              | | scr     |    | str |                            |
|      +---3---+      V  TO +---4---+      +-------+               |
|      |       |<-----+<------|       |<-----------|       |       |
|      | Req-  |       | scr  | Ack-  |    scn     | Good  |       |
|      | Sent  | RCA          | Rcvd  | RCR        | Req?  |       |
|      |       |------------->|       |----------->|       |       |
|      +-------+              +-------+            +-------+       |
|        | ^                                          |           |
|    RCR | +<--------+                                |           |
|    --- | |         |    TO       RCN      ---       |           |
|        | |   ---   +---------+  +-----+   sca       V           |
|        V | scn     scr |   | scr |              +---6---+ C |    |
|    +-------+          +---5---+  |            |       |---+ |    |
+--->|       |------------->|       |<--+        | Open  | str |    |
    | Good  | sca          | Ack-  |  |         |       |      |    |
    | Req?  |     RCR      | Sent  | RCA        |       |      |    |
    |       |<-------------|       |----------->|       |      |    |
    +-------+              +-------+            +-------+      |    |
        ^                                        |     |      |    |
        |                                 RCR |   | RTR  |    |
        +----------------------------------------+   +--------+    |
                                             scr       sta
```
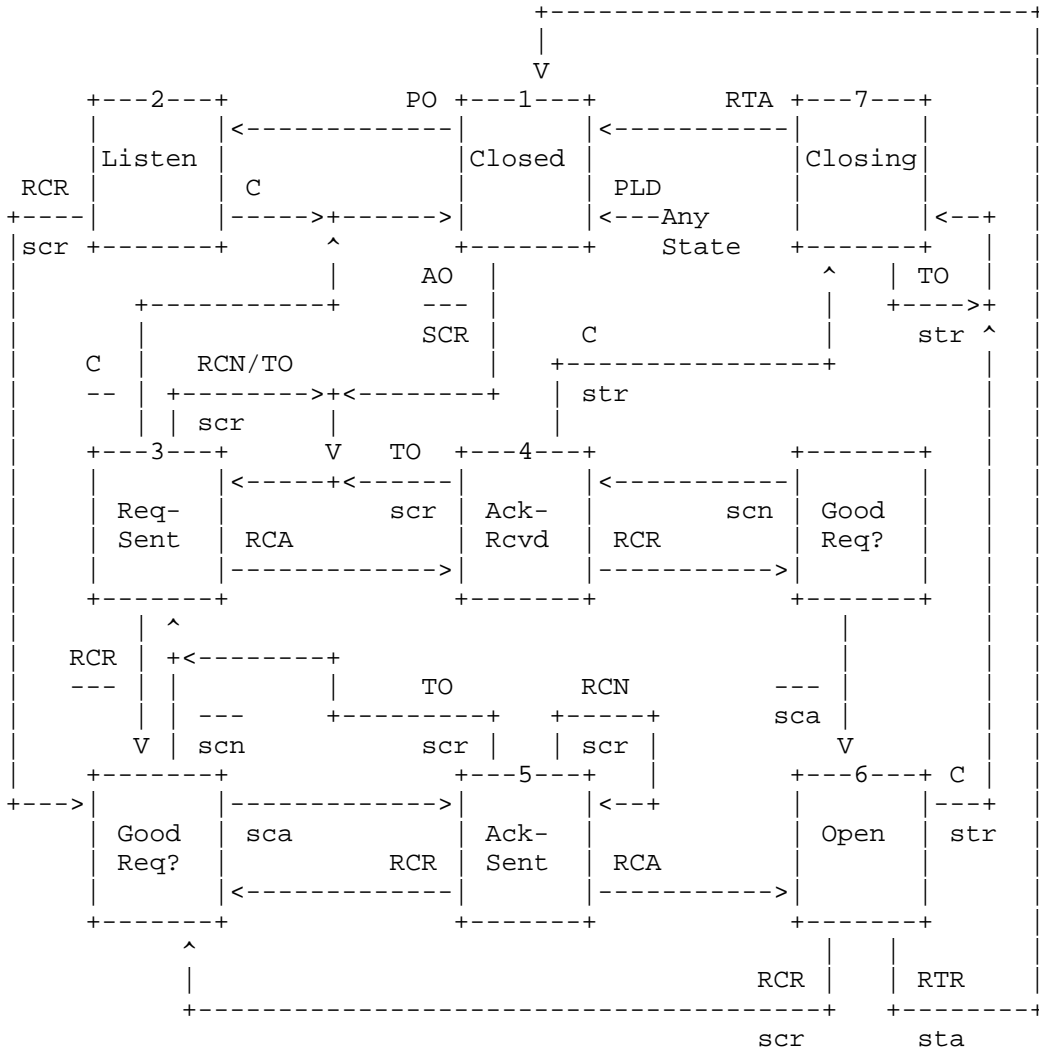
Events   Actions
RCR - Receive-Configure-Request   scr - Send Configure-Request
RCA - Receive-Configure-Ack   sca - Send Configure-Ack
RCN - Receive-Configure-Nak or Reject scn - Send Configure-Nak or
RTR - Receive-Terminate-Req   Reject
RTA - Receive-Terminate-Ack   str - Send Terminate-Req
AO - Active-Open   sta - Sent Terminate-Ack
PO - Passive-Open
C   - Close
TO - Timeout
PLD - Physical-Layer-Down

4.1.3. State Transition Table

   The complete state transition table follows. States are indicated
   horizontally, and events are read vertically. State transitions and
   actions are represented in the form action/new-state. Two actions
   caused by the same event are represented as action1&action2.

```
         | State
         |  1         2         3         4         5         6          7
 Events| Closed    Listen    Req-Sent  Ack-Rcvd  Ack-Sent  Open       Closing
 ------+-------------------------------------------------------------------
   AO  | scr/3     scr/3       3         4         5         6        scr/3
   PO  |   2         2         2*        4         5         6        sta/3*
   C   |   1         1         1*        1       str/7     str/7        7
   TO  |   1         2       scr/3     scr/3     scr/3       6        str/7*
  PLD  |   1         1         1         1         1         1          1
  RCR+ | sta/1   scr&sca/5   sca/5     sca/6     sca/5    scr&sca/5     7
  RCR- | sta/1   scr&scn/3   scn/3     scn/4     scn/3    scr&scn/3     7
  RCA  | sta/1    sta/2        4       scr/3       6       scr/3        7
  RCN  | sta/1    sta/2      scr/3     scr/3     scr/5     scr/3        7
  RTR  | sta/1    sta/2      sta/3     sta/3     sta/3     sta/1      sta/7
  RTA  |   1         2         3         3         3         1          1
  RCJ  |   1         2         1         1         1         1          1
  RUC  | scj/1    scj/1     scj/1     scj/1     scj/1     scj/1    1 scj/1
  RER  | sta/1    sta/2        3         4         5       ser/1        7
```

   Notes:
       RCR+ - Receive-Configure-Request (Good)
       RCR- - Receive-Configure-Request (Bad)
       RCJ - Receive-Code-Reject
       RUC - Receive-Unknown-Code
       RER - Receive-Echo-Request
       scj - Send-Code-Reject
       ser - Send-Echo-Reply
        *   - Special attention necessary, see detailed text

4.1.4. Events

   Transitions and actions in the LCP state machine are caused by
   events. Some events are caused by commands executed at the local end
   (e.g., Active-Open, Passive-Open, and Close), others are caused by
   the receipt of packets from the remote end (e.g., Receive- Configure-
   Request, Receive-Configure-Ack, Receive-Configure-Nak, Receive-
   Terminate-Request and Receive-Terminate-Ack), and still others are
   caused by the expiration of the Restart timer started as the result
   of other events (e.g., Timeout).

   Following is a list of LCP events.

Active-Open (AO)

   The Active-Open event indicates the local execution of an Active-
   Open command by the network administrator (human or program).
   When this event occurs, LCP should immediately attempt to open the
   connection by exchanging configuration packets with the LCP peer.

Passive-Open (PO)

   The Passive-Open event is similar to the Active-Open event.
   However, instead of immediately exchanging configuration packets,
   LCP should wait for the peer to send the first packet. This will
   only happen after an Active-Open event in the LCP peer.

Close (C)

   The Close event indicates the local execution of a Close command.
   When this event occurs, LCP should immediately attempt to close
   the connection.

Timeout (TO)

   The Timeout event indicates the expiration of the LCP Restart
   timer. The LCP Restart timer is started as the result of other
   LCP events.

   The Restart timer is used to time out transmissions of Configure-
   Request and Terminate-Request packets. Expiration of the Restart
   timer causes a Timeout event, which triggers the corresponding
   Configure-Request or Terminate-Request packet to be retransmitted.
   The Restart timer MUST be configurable, but should default to
   three (3) seconds.

Receive-Configure-Request (RCR)

   The Receive-Configure-Request event occurs when a Configure-
   Request packet is received from the LCP peer. The Configure-
   Request packet indicates the desire to open a LCP connection and
   may specify Configuration Options. The Configure-Request packet
   is more fully described in a later section.

Receive-Configure-Ack (RCA)

   The Receive-Configure-Ack event occurs when a valid Configure-Ack
   packet is received from the LCP peer. The Configure-Ack packet is
   a positive response to a Configure-Request packet.

Receive-Configure-Nak (RCN)

   The Receive-Configure-Nak event occurs when a valid Configure-Nak
   or Configure-Reject packet is received from the LCP peer. The
   Configure-Nak and Configure-Reject packets are negative responses
   to a Configure-Request packet.

Receive-Terminate-Request (RTR)

   The Receive-Terminate-Request event occurs when a Terminate-
   Request packet is received from the LCP peer. The Terminate-
   Request packet indicates the desire to close the LCP connection.

Receive-Terminate-Ack (RTA)

   The Receive-Terminate-Ack event occurs when a Terminate-Ack packet
   is received from the LCP peer. The Terminate-Ack packet is a
   response to a Terminate-Request packet.

Receive-Code-Reject (RCJ)

   The Receive-Code-Reject event occurs when a Code-Reject packet is
   received from the LCP peer. The Code-Reject packet communicates
   an error that immediately closes the connection.

Receive-Unknown-Code (RUC)

   The Receive-Unknown-Code event occurs when an un-interpretable
   packet is received from the LCP peer. The Code-Reject packet is a
   response to an unknown packet.

Receive-Echo-Request (RER)

   The Receive-Echo-Request event occurs when a Echo-Request, Echo-
   Reply, or Discard-Request packet is received from the LCP peer.
   The Echo-Reply packet is a response to a Echo-Request packet.
   There is no reply to a Discard-Request.

Physical-Layer-Down (PLD)

   The Physical-Layer-Down event occurs when the Physical Layer
   indicates that it is down.

4.1.5. Actions

   Actions in the LCP state machine are caused by events and typically
   indicate the transmission of packets and/or the starting or stopping
   of the Restart timer. Following is a list of LCP actions.

Send-Configure-Request (scr)

   The Send-Configure-Request action transmits a Configure-Request
   packet. This indicates the desire to open a LCP connection with a
   specified set of Configuration Options. The Restart timer is
   started after the Configure-Request packet is transmitted, to
   guard against packet loss.

Send-Configure-Ack (sca)

   The Send-Configure-Ack action transmits a Configure-Ack packet.
   This acknowledges the receipt of a Configure-Request packet with
   an acceptable set of Configuration Options.

Send-Configure-Nak (scn)

   The Send-Configure-Nak action transmits a Configure-Nak or
   Configure-Reject packet, as appropriate. This negative response
   reports the receipt of a Configure-Request packet with an
   unacceptable set of Configuration Options. Configure-Nak packets
   are used to refuse a Configuration Option value, and to suggest a
   new, acceptable value. Configure-Reject packets are used to
   refuse all negotiation about a Configuration Option, typically
   because it is not recognized or implemented. The use of
   Configure-Nak vs. Configure-Reject is more fully described in the
   section on LCP Packet Formats.

Send-Terminate-Req (str)

   The Send-Terminate-Request action transmits a Terminate-Request
   packet. This indicates the desire to close a LCP connection. The
   Restart timer is started after the Terminate-Request packet is
   transmitted, to guard against packet loss.

Send-Terminate-Ack (sta)

   The Send-Terminate-Request action transmits a Terminate-Ack
   packet. This acknowledges the receipt of a Terminate-Request
   packet or otherwise confirms the belief that a LCP connection is
   Closed.

Send-Code-Reject (scj)

   The Send-Code-Reject action transmits a Code-Reject packet. This
   indicates the receipt of an unknown type of packet. This is an
   unrecoverable error which causes immediate transitions to the
   Closed state on both ends of the link.

   Send-Echo-Reply (ser)

      The Send-Echo-Reply action transmits an Echo-Reply packet. This
      acknowledges the receipt of an Echo-Request packet.

4.1.6. States

   Following is a more detailed description of each LCP state.

   Closed (1)

      The initial and final state is the Closed state. In the Closed
      state the connection is down and there is no attempt to open it;
      all connection requests from peers are rejected. Physical-Layer-
      Down events always cause an immediate transition to the Closed
      state.

      There are two events which cause a transition out of the Closed
      state, Active-Open and Passive-Open. Upon an Active-Open event, a
      Configure-Request is transmitted, the Restart timer is started,
      and the Request-Sent state is entered. Upon a Passive-Open event,
      the Listen state is entered immediately. Upon receipt of any
      packet, with the exception of a Terminate-Ack, a Terminate-Ack is
      sent. Terminate-Acks are silently discarded to avoid creating a
      loop.

      The Restart timer is not running in the Closed state.

      The Physical Layer connection may be disconnected at any time when
      in the LCP Closed state.

   Listen (2)

      The Listen state is similar to the Closed state in that the
      connection is down and there is no attempt to open it. However,
      peer connection requests are no longer rejected.

      Upon receipt of a Configure-Request, a Configure-Request is
      immediately transmitted and the Restart timer is started. The
      received Configuration Options are examined and the proper
      response is sent. If a Configure-Ack is sent, the Ack-Sent state
      is entered. Otherwise, if a Configure-Nak or Configure-Reject is
      sent, the Request-Sent state is entered. In either case, LCP
      exits its passive state, and begins to actively open the
      connection. Terminate-Ack packets are sent in response to either
      Configure-Ack or Configure-Nak packets,

      The Restart timer is not running in the Listen state.

   Request-Sent (3)

      In the Request-Sent state an active attempt is made to open the
      connection. A Configure-Request has been sent and the Restart
      timer is running, but a Configure-Ack has not yet been received
      nor has one been sent.

      Upon receipt of a Configure-Ack, the Ack-Received state is
      immediately entered. Upon receipt of a Configure-Nak or
      Configure-Reject, the Configure-Request Configuration Options are
      adjusted appropriately, a new Configure-Request is transmitted,
      and the Restart timer is restarted. Similarly, upon the
      expiration of the Restart timer, a new Configure-Request is
      transmitted and the Restart timer is restarted. Upon receipt of a
      Configure-Request, the Configuration Options are examined and if
      acceptable, a Configure-Ack is sent and the Ack-Sent state is
      entered. If the Configuration Options are unacceptable, a
      Configure-Nak or Configure-Reject is sent as appropriate.

      Since there is an outstanding Configure-Request in the Request-
      Sent state, special care must be taken to implement the Passive-
      Open and Close events; otherwise, it is possible for the LCP peer
      to think the connection is open. Processing of either event
      should be postponed until there is reasonable assurance that the
      peer is not open. In particular, the Restart timer should be
      allowed to expire.

   Ack-Received (4)

      In the Ack-Received state, a Configure-Request has been sent and a
      Configure-Ack has been received. The Restart timer is still
      running since a Configure-Ack has not yet been transmitted.

      Upon receipt of a Configure-Request with acceptable Configuration
      Options, a Configure-Ack is transmitted, the Restart timer is
      stopped and the Open state is entered. If the Configuration
      Options are unacceptable, a Configure-Nak or Configure-Reject is
      sent as appropriate. Upon the expiration of the Restart timer, a
      new Configure-Request is transmitted, the Restart timer is
      restarted, and the state machine returns to the Request-Sent
      state.

   Ack-Sent (5)

      In the Ack-Sent state, a Configure-Ack and a Configure-Request
      have been sent but a Configure-Ack has not yet been received. The
      Restart timer is always running in the Ack-Sent state.

Upon receipt of a Configure-Ack, the Restart timer is stopped and
the Open state is entered. Upon receipt of a Configure-Nak or
Configure-Reject, the Configure-Request Configuration Options are
adjusted appropriately, a new Configure-Request is transmitted,
and the Restart timer is restarted. Upon the expiration of the
Restart timer, a new Configure-Request is transmitted, the Restart
timer is restarted, and the state machine returns to the Request-
Sent state.

Open (6)

In the Open state, a connection exists and data may be
communicated over the link. The Restart timer is not running in
the Open state.

In normal operation, only two events cause transitions out of the
Open state. Upon receipt of a Close command, a Terminate-Request
is transmitted, the Restart timer is started, and the Closing
state is entered. Upon receipt of a Terminate-Request, a
Terminate-Ack is transmitted and the Closed state is entered.
Upon receipt of an Echo-Request, an Echo-Reply is transmitted.
Similarly, Echo-Reply and Discard-Request packets are silently
discarded or processed as expected. All other events cause
immediate transitions out of the Open state and should be handled
as if the state machine were in the Listen state.

Closing (7)

In the Closing state, an active attempt is made to close the
connection. A Terminate-Request has been sent and the Restart
timer is running, but a Terminate-Ack has not yet been received.

Upon receipt of a Terminate-Ack, the Closed state is immediately
entered. Upon the expiration of the Restart timer, a new
Terminate-Request is transmitted and the Restart timer is
restarted. After the Restart timer has expired Max-Restart times,
this action may be skipped, and the Closed state may be entered.
Max-Restart MUST be a configurable parameter.

Since there is an outstanding Terminate-Request in the Closing
state, special care must be taken to implement the Passive-Open
event; otherwise, it is possible for the LCP peer to think the
connection is open. Processing of the Passive-Open event should
be postponed until there is reasonable assurance that the peer is
not open. In particular, the implementation should wait until the
state machine would normally transition to the Closed state
because of a Receive-Terminate-Ack event or Max-Restart Timeout
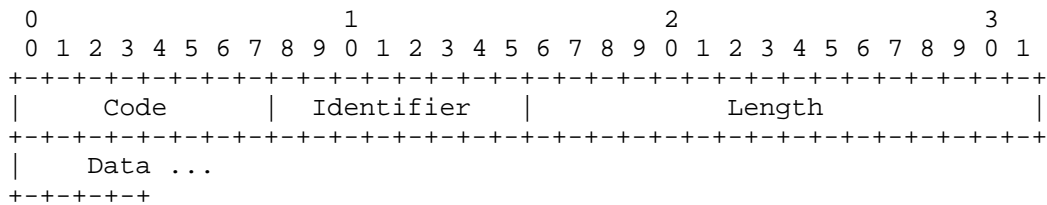events.

4.2. Loop Avoidance

   Note that the protocol makes a reasonable attempt at avoiding
   Configuration Option negotiation loops. However, the protocol does
   NOT guarantee that loops will not happen. As with any negotiation,
   it is possible to configure two PPP implementations with conflicting
   policies that will never converge. It is also possible to configure
   policies which do converge, but which take significant time to do so.
   Implementors should keep this in mind and should implement loop
   detection mechanisms or higher level timeouts. If a timeout is
   implemented, it MUST be configurable.

   For example, implementations could take care to avoid Configure-
   Request or Terminate-Request livelocks by using a Max-Retries
   counter. A Configure-Request livelock could occur when an
   originating PPP sends and re-sends a C-R without receiving a reply
   (e.g., the receiving PPP entity may have died). A Terminate-Request
   livelock could occur when the originating PPP sends and re-sends a
   T-R without receiving a Terminate-Ack (e.g., the T-A may have been
   lost, but the remote PPP may have already terminated). Max-Retries
   indicates the number of packet retransmissions that are allowed
   before there is reasonable assurance that a livelock situation
   exists. Max-Retries MUST also be configurable, but should default to
   ten (10) retransmissions.

4.3 Packet Format

   Exactly one Link Control Protocol packet is encapsulated in the
   Information field of PPP Data Link Layer frames where the Protocol
   field indicates type hex c021 (Link Control Protocol).

   A summary of the Link Control Protocol packet 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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |     Code      |  Identifier   |            Length             |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |    Data ...
   +-+-+-+-+
```

   Code

      The Code field is one octet and identifies the kind of LCP packet.
      LCP Codes are assigned as follows:

```
         1          Configure-Request
         2          Configure-Ack
         3          Configure-Nak
         4          Configure-Reject
         5          Terminate-Request
         6          Terminate-Ack
         7          Code-Reject
         8          Protocol-Reject
         9          Echo-Request
         10         Echo-Reply
         11         Discard-Request
```

Identifier

   The Identifier field is one octet and aids in matching requests
   and replies.

Length

   The Length field is two octets and indicates the length of the LCP
   packet including the Code, Identifier, Length and Data fields.
   Octets outside the range of the Length field should be treated as
   Data Link Layer padding and should be ignored on reception.

Data

   The Data field is zero or more octets as indicated by the Length
   field. The format of the Data field is determined by the Code
   field.

Regardless of which Configuration Options are enabled, all LCP
packets are always sent in the full, standard form, as if no
Configuration Options were enabled. This ensures that LCP
Configure-Request packets are always recognizable even when one end
of the link mistakenly believes the link to be Open.

This document describes Version 1 of the Link Control Protocol. In
the interest of simplicity, there is no version field in the LCP
packet. If a new version of LCP is necessary in the future, the
intention is that a new Data Link Layer Protocol field value should
be used to differentiate Version 1 LCP from all other versions. A
correctly functioning Version 1 LCP implementation will always
respond to unknown Protocols (including other versions) with an
easily recognizable Version 1 packet, thus providing a deterministic
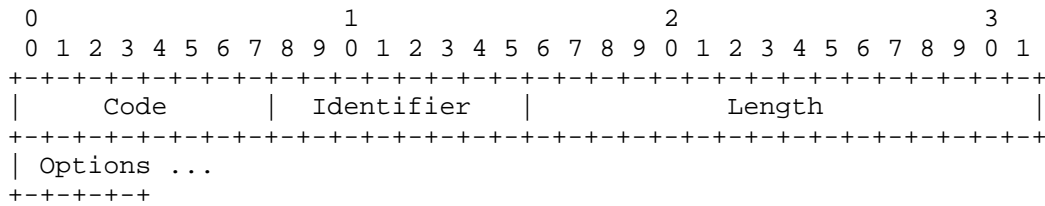fallback mechanism for implementations of other versions.

4.3.1. Configure-Request

   Description

      A LCP implementation wishing to open a connection MUST transmit a
      LCP packet with the Code field set to 1 (Configure-Request) and
      the Options field filled with any desired changes to the default
      link Configuration Options.

      Upon reception of a Configure-Request, an appropriate reply MUST
      be transmitted.

   A summary of the Configure-Request packet 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Code      |  Identifier   |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Options ...
+-+-+-+-+-+
```

   Code

      1 for Configure-Request.

   Identifier

      The Identifier field should be changed on each transmission. On
      reception, the Identifier field should be copied into the
      Identifier field of the appropriate reply packet.

   Options

      The options field is variable in length and contains the list of
      zero or more Configuration Options that the sender desires to
      negotiate. All Configuration Options are always negotiated
      simultaneously. The format of Configuration Options is further
      described in a later section.

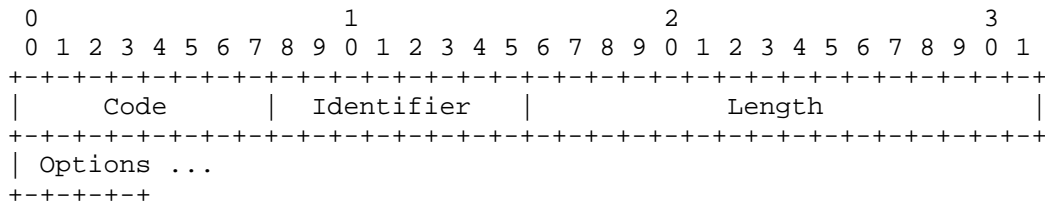4.3.2. Configure-Ack

   Description

      If every Configuration Option received in a Configure-Request is
      both recognizable and acceptable, then a LCP implementation should
      transmit a LCP packet with the Code field set to 2 (Configure-

Ack), the Identifier field copied from the received Configure-
Request, and the Options field copied from the received
Configure-Request. The acknowledged Configuration Options MUST
NOT be reordered or modified in any way.

On reception of a Configure-Ack, the Identifier field must match
that of the last transmitted Configure-Request, or the packet is
invalid. Additionally, the Configuration Options in a Configure-
Ack must match those of the last transmitted Configure-Request, or
the packet is invalid. Invalid packets should be silently
discarded.

Reception of a valid Configure-Ack indicates that all
Configuration Options sent in the last Configure-Request are
acceptable.

A summary of the Configure-Ack packet 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Code      |   Identifier  |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Options ...
+-+-+-+-+
```

Code

   2 for Configure-Ack.

Identifier

   The Identifier field is a copy of the Identifier field of the
   Configure-Request which caused this Configure-Ack.

Options

   The Options field is variable in length and contains the list of
   zero or more Configuration Options that the sender is
   acknowledging. All Configuration Options are always acknowledged
   simultaneously.

4.3.3. Configure-Nak

Description

   If every element of the received Configuration Options is

recognizable but some are not acceptable, then a LCP
implementation should transmit a LCP packet with the Code field
set to 3 (Configure-Nak), the Identifier field copied from the
received Configure-Request, and the Options field filled with only
the unacceptable Configuration Options from the Configure-Request.
All acceptable Configuration Options should be filtered out of the
Configure-Nak, but otherwise the Configuration Options from the
Configure-Request MUST NOT be reordered. Each of the nak'd
Configuration Options MUST be modified to a value acceptable to
the Configure-Nak sender. Finally, an implementation may be
configured to require the negotiation of a specific option. If
that option is not listed, then that option may be appended to the
list of nak'd Configuration Options in order to request the remote
end to list that option in its next Configure-Request packet. The
appended option must include a value acceptable to the Configure-
Nak sender.

On reception of a Configure-Nak, the Identifier field must match
that of the last transmitted Configure-Request, or the packet is
invalid and should be silently discarded.

Reception of a valid Configure-Nak indicates that a new
Configure-Request should be sent with the Configuration Options
modified as specified in the Configure-Nak.

A summary of the Configure-Nak packet 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Code      |  Identifier   |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Options ...
+-+-+-+-+
```

Code

   3 for Configure-Nak.

Identifier

   The Identifier field is a copy of the Identifier field of the
   Configure-Request which caused this Configure-Nak.

Options

   The Options field is variable in length and contains the list of

zero or more Configuration Options that the sender is nak'ing.
All Configuration Options are always nak'd simultaneously.

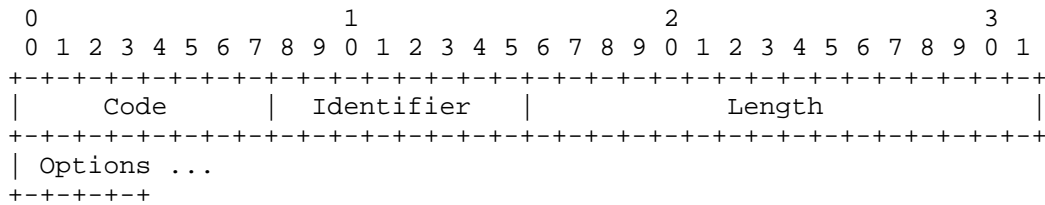4.3.4. Configure-Reject

Description

   If some Configuration Options received in a Configure-Request are
   not recognizable or are not acceptable for negotiation (as
   configured by a network manager), then a LCP implementation should
   transmit a LCP packet with the Code field set to 4 (Configure-
   Reject), the Identifier field copied from the received Configure-
   Request, and the Options field filled with only the unrecognized
   Configuration Options from the Configure-Request. All
   recognizable and negotiable Configuration Options must be filtered
   out of the Configure-Reject, but otherwise the Configuration
   Options MUST not be reordered.

   On reception of a Configure-Reject, the Identifier field must
   match that of the last transmitted Configure-Request, or the
   packet is invalid. Additionally, the Configuration Options in a
   Configure-Reject must be a proper subset of those in the last
   transmitted Configure-Request, or the packet is invalid. Invalid
   packets should be silently discarded.

   Reception of a Configure-Reject indicates that a new Configure-
   Request should be sent which does not include any of the
   Configuration Options listed in the Configure-Reject.

 A summary of the Configure-Reject packet 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Code      |  Identifier   |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Options ...
+-+-+-+-+
```

Code

   4 for Configure-Reject.

Identifier

   The Identifier field is a copy of the Identifier field of the
   Configure-Request which caused this Configure-Reject.

   Options

      The Options field is variable in length and contains the list of
      zero or more Configuration Options that the sender is rejecting.
      All Configuration Options are always rejected simultaneously.

4.3.5. Terminate-Request and Terminate-Ack
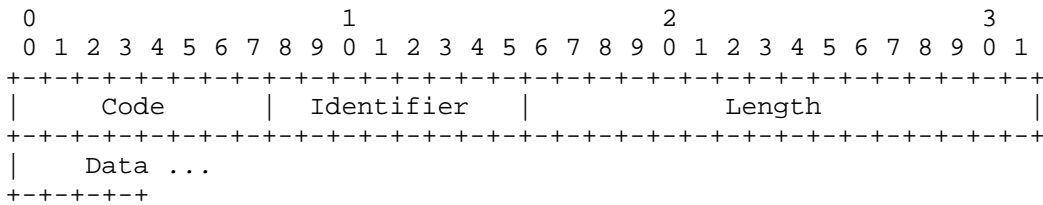
   Description

      LCP includes Terminate-Request and Terminate-Ack Codes in order to
      provide a mechanism for closing a connection.

      A LCP implementation wishing to close a connection should transmit
      a LCP packet with the Code field set to 5 (Terminate-Request) and
      the Data field filled with any desired data. Terminate-Request
      packets should continue to be sent until Terminate-Ack is
      received, the Physical Layer indicates that it has gone down, or a
      sufficiently large number have been transmitted such that the
      remote end is down with reasonable certainty.

      Upon reception of a Terminate-Request, a LCP packet MUST be
      transmitted with the Code field set to 6 (Terminate-Ack), the
      Identifier field copied from the Terminate-Request packet, and the
      Data field filled with any desired data.

      Reception of an unelicited Terminate-Ack indicates that the
      connection has been closed.

   A summary of the Terminate-Request and Terminate-Ack packet formats
   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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Code      |  Identifier   |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Data ...
+-+-+-+-+
```

   Code

      5 for Terminate-Request;

      6 for Terminate-Ack.

Identifier

   The Identifier field is one octet and aids in matching requests
   and replies.

Data

   The Data field is zero or more octets and contains uninterpreted
   data for use by the sender. The data may consist of any binary
   value and may be of any length from zero to the established value
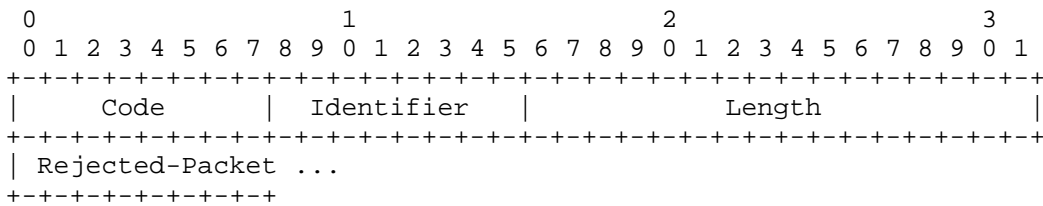   for the peer's MRU.

4.3.6. Code-Reject

Description

   Reception of a LCP packet with an unknown Code indicates that one
   of the communicating LCP implementations is faulty or incomplete.
   This error MUST be reported back to the sender of the unknown Code
   by transmitting a LCP packet with the Code field set to 7 (Code-
   Reject), and the inducing packet copied to the Rejected-Packet
   field.

   Upon reception of a Code-Reject, a LCP implementation should make
   an immediate transition to the Closed state, and should report the
   error, since it is unlikely that the situation can be rectified
   automatically.

A summary of the Code-Reject packet 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Code      |  Identifier   |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Rejected-Packet ...
+-+-+-+-+-+-+-+-+-+
```

Code

   7 for Code-Reject.

Identifier

   The Identifier field is one octet and is for use by the
   transmitter.

Rejected-Packet

   The Rejected-Packet field contains a copy of the LCP packet which
   is being rejected. It begins with the rejected Code field; it
   does not include any PPP Data Link Layer headers. The Rejected-
   Packet should be truncated to comply with the established value of
   the peer's MRU.

4.3.7. Protocol-Reject

   Description

      Reception of a PPP frame with an unknown Data Link Layer Protocol
      indicates that the remote end is attempting to use a protocol
      which is unsupported at the local end. This typically occurs when
      the remote end attempts to configure a new, but unsupported
      protocol. If the LCP state machine is in the Open state, then
      this error MUST be reported back to the sender of the unknown
      protocol by transmitting a LCP packet with the Code field set to 8
      (Protocol-Reject), the Rejected-Protocol field set to the received
      Protocol, and the Data field filled with any desired data.

      Upon reception of a Protocol-Reject, a LCP implementation should
      stop transmitting frames of the indicated protocol.

      Protocol-Reject packets may only be sent in the LCP Open state.
      Protocol-Reject packets received in any state other than the LCP
      Open state should be discarded and no further action should be
      taken.

   A summary of the Protocol-Reject packet 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Code      |  Identifier   |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       Rejected-Protocol       |    Rejected-Information ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
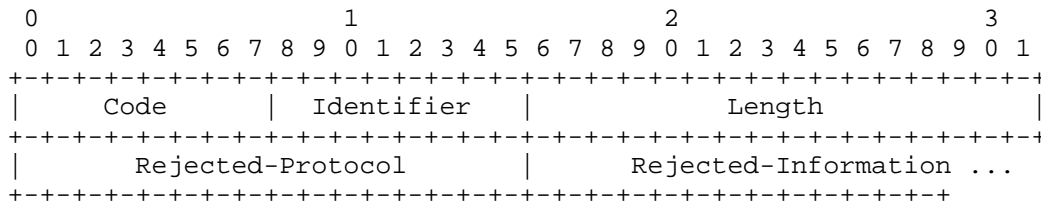
   Code

      8 for Protocol-Reject.

   Identifier

      The Identifier field is one octet and is for use by the

      transmitter.

   Rejected-Protocol

      The Rejected-Protocol field is two octets and contains the
      Protocol of the Data Link Layer frame which is being rejected.

   Rejected-Information

      The Rejected-Information field contains a copy from the frame
      which is being rejected. It begins with the Information field,
      and does not include any PPP Data Link Layer headers or the FCS.
      The Rejected-Information field should be truncated to comply with
      the established value of the peer's MRU.

4.3.8. Echo-Request and Echo-Reply

   Description

      LCP includes Echo-Request and Echo-Reply Codes in order to provide
      a Data Link Layer loopback mechanism for use in exercising both
      directions of the link. This is useful as an aid in debugging,
      link quality determination, performance testing, and for numerous
      other functions.

      An Echo-Request sender transmits a LCP packet with the Code field
      set to 9 (Echo-Request) and the Data field filled with any desired
      data, up to but not exceeding the receivers established MRU.

      Upon reception of an Echo-Request, a LCP packet MUST be
      transmitted with the Code field set to 10 (Echo-Reply), the
      Identifier field copied from the received Echo-Request, and the
      Data field copied from the Echo-Request, truncating as necessary
      to avoid exceeding the peer's established MRU.

      Echo-Request and Echo-Reply packets may only be sent in the LCP
      Open state. Echo-Request and Echo-Reply packets received in any
      state other than the LCP Open state should be discarded and no
      further action should be taken.

   A summary of the Echo-Request and Echo-Reply packet formats 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Code      |  Identifier   |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Magic-Number                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Data ...
+-+-+-+-+-+
```
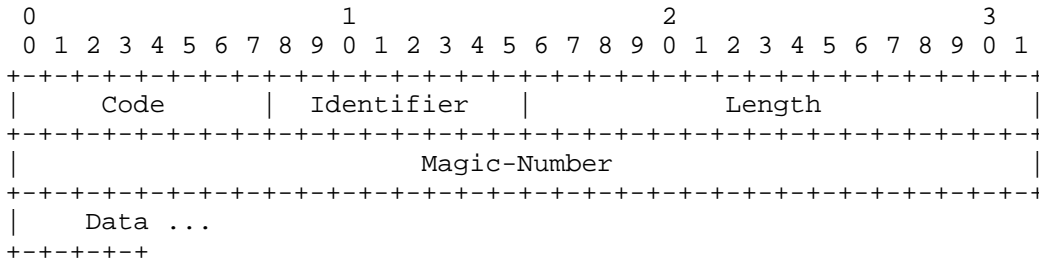
Code

   9 for Echo-Request;

   10 for Echo-Reply.

Identifier

   The Identifier field is one octet and aids in matching Echo-
   Requests and Echo-Replies.

Magic-Number

   The Magic-Number field is four octets and aids in detecting
   loopbacked links. Unless modified by a Configuration Option, the
   Magic-Number MUST always be transmitted as zero and MUST always be
   ignored on reception. Further use of the Magic-Number is beyond
   the scope of this discussion.

Data

   The Data field is zero or more octets and contains uninterpreted
   data for use by the sender. The data may consist of any binary
   value and may be of any length from zero to the established value
   for the peer's MRU.

4.3.9. Discard-Request

Description

   LCP includes a Discard-Request Code in order to provide a Data
   Link Layer data sink mechanism for use in exercising the local to
   remote direction of the link. This is useful as an aid in
   debugging, performance testing, and and for numerous other
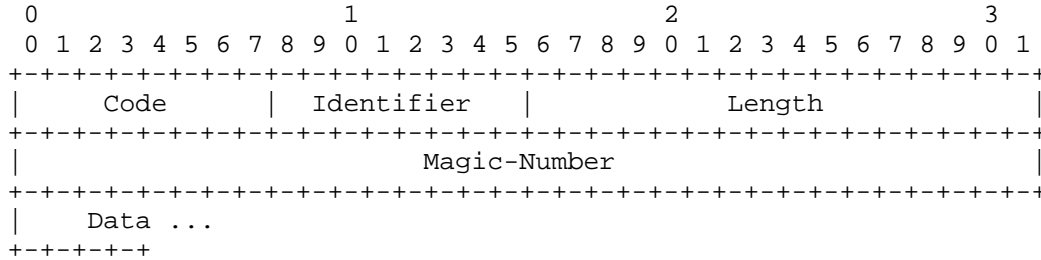   functions.

   A discard sender transmits a LCP packet with the Code field set to
   11 (Discard-Request) and the Data field filled with any desired

data, up to but not exceeding the receivers established MRU.

A discard receiver MUST simply throw away an Discard-Request that it receives.

Discard-Request packets may only be sent in the LCP Open state.

A summary of the Discard-Request packet formats 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Code      |  Identifier   |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Magic-Number                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Data ...
+-+-+-+-+
```

Code

    11 for Discard-Request.

Identifier

    The Identifier field is one octet and is for use by the Discard-
    Request transmitter.

Magic-Number

    The Magic-Number field is four octets and aids in detecting
    loopbacked links. Unless modified by a configuration option, the
    Magic-Number MUST always be transmitted as zero and MUST always be
    ignored on reception. Further use of the Magic-Number is beyond
    the scope of this discussion.

Data

    The Data field is zero or more octets and contains uninterpreted
    data for use by the sender. The data may consist of any binary
    value and may be of any length from zero to the established value
    for the peer's MRU.

4.4. Configuration Options

   LCP Configuration Options allow modifications to the standard
   characteristics of a point-to-point link to be negotiated.

Negotiable modifications include such things as the maximum receive
unit, async control character mapping, the link authentication
method, the link encryption method, etc.. The Configuration Options
themselves are described in separate documents. If a Configuration
Option is not included in a Configure-Request packet, the default
value for that Configuration Option is assumed.

The end of the list of Configuration Options is indicated by the end
of the LCP packet.

Unless otherwise specified, a specific Configuration Options should
be listed no more than once in a Configuration Options list.
Specific Configuration Options may override this general rule and may
be listed more than once. The effect of this is Configuration Option
specific and is specified by each such Configuration Option.

Also unless otherwise specified, all Configuration Options apply in a
half-duplex fashion. When negotiated, they apply to only one
direction of the link, typically in the receive direction when
interpreted from the point of view of the Configure-Request sender.

4.4.1. Format

A summary of the Configuration Option format is shown below. The
fields are transmitted from left to right.

```
 0                   1
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |    Length     |    Data ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Type

   The Type field is one octet and indicates the type of
   Configuration Option. The most up-to-date values of the Type
   field are specified in the most recent "Assigned Numbers" RFC
   [11].

Length

   The Length field is one octet and indicates the length of this
   Configuration Option including the Type, Length and Data fields.
   If a negotiable Configuration Option is received in a Configure-
   Request but with an invalid Length, a Configure-Nak should be
   transmitted which includes the desired Configuration Option with
   an appropriate Length and Data.

Data

   The Data field is zero or more octets and indicates the value or
   other information for this Configuration Option. The format and
   length of the Data field is determined by the Type and Length
   fields.

5. A PPP Network Control Protocol (NCP) for IP

   The IP Control Protocol (IPCP) is responsible for configuring,
   enabling, and disabling the IP protocol modules on both ends of the
   point-to-point link. As with the Link Control Protocol, this is
   accomplished through an exchange of packets. IPCP packets may not be
   exchanged until LCP has reached the network-layer Protocol
   Configuration Negotiation phase. Likewise, IP datagrams may not be
   exchanged until IPCP has first opened the connection.

   The IP Control Protocol is exactly the same as the Link Control
   Protocol with the following exceptions:

   Data Link Layer Protocol Field

      Exactly one IP Control Protocol packet is encapsulated in the
      Information field of PPP Data Link Layer frames where the Protocol
      field indicates type hex 8021 (IP Control Protocol).

   Code field

      Only Codes 1 through 7 (Configure-Request, Configure-Ack,
      Configure-Nak, Configure-Reject, Terminate-Request, Terminate-Ack
      and Code-Reject) are used. Other Codes should be treated as
      unrecognized and should result in Code-Rejects.

   Timeouts

      IPCP packets may not be exchanged until the Link Control Protocol
      has reached the network-layer Protocol Configuration Negotiation
      phase. An implementation should be prepared to wait for Link
      Quality testing to finish before timing out waiting for a
      Configure-Ack or other response. It is suggested that an
      implementation give up only after user intervention or a
      configurable amount of time.

   Configuration Option Types

      The IPCP has a separate set of Configuration Options. The most
      up-to-date values of the type field are specified in the most
      recent "Assigned Numbers" RFC [11].

5.1. Sending IP Datagrams

   Before any IP packets may be communicated, both the Link Control
   Protocol and the IP Control Protocol must reach the Open state.

   Exactly one IP packet is encapsulated in the Information field of PPP
   Data Link Layer frames where the Protocol field indicates type hex
   0021 (Internet Protocol).

   The maximum length of an IP packet transmitted over a PPP link is the
   same as the maximum length of the Information field of a PPP data
   link layer frame. Larger IP datagrams must be fragmented as
   necessary. If a system wishes to avoid fragmentation and reassembly,
   it should use the TCP Maximum Segment Size option [12], or a similar
   mechanism, to discourage others from sending large datagrams.

A. Asynchronous HDLC

   This appendix summarizes the modifications to ISO 3309-1979 proposed
   in ISO 3309:1984/PDAD1. These modifications allow HDLC to be used
   with 8-bit asynchronous links.

   Transmission Considerations

      Each octet is delimited by a start and a stop element.

   Flag Sequence

      The Flag Sequence is a single octet and indicates the beginning or
      end of a frame. The Flag Sequence consists of the binary sequence
      01111110 (hexadecimal 0x7e).

   Transparency

      On asynchronous links, a character stuffing procedure is used.
      The Control Escape octet is defined as binary 01111101
      (hexadecimal 0x7d) where the bit positions are numbered 87654321
      (not 76543210, BEWARE).

      After FCS computation, the transmitter examines the entire frame
      between the two Flag Sequences. Each Flag Sequence, Control
      Escape octet and octet with value less than hexadecimal 0x20 is
      replaced by a two character sequence consisting of the Control
      Escape octet and the original octet with bit 6 complemented (i.e.,
      exclusive-or'd with hexadecimal 0x20).

      Prior to FCS computation, the receiver examines the entire frame
      between the two Flag Sequences. For each Control Escape octet,

that octet is removed and bit 6 of the following octet is
complemented. A Control Escape octet immediately preceding the
closing Flag Sequence indicates an invalid frame.

   Note: The inclusion of all octets less than hexadecimal 0x20
   allows all ASCII control characters [10] excluding DEL (Delete)
   to be transparently communicated through almost all known data
   communications equipment.

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

   0x7e is encoded as 0x7d, 0x5e.
   0x7d is encoded as 0x7d, 0x5d.
   0x01 is encoded as 0x7d, 0x21.

Aborting a Transmission

   On asynchronous links, frames may be aborted by transmitting a "0"
   stop bit where a "1" bit is expected (framing error) or by
   transmitting a Control Escape octet followed immediately by a
   closing Flag Sequence.

Inter-frame Time Fill

   On asynchronous links, inter-octet and inter-frame time fill
   should be accomplished by transmitting continuous "1" bits (mark-
   hold state).

   Note: On asynchronous links, 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 close 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 transmit 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). Transmitters should avoid this by
   transmitting an open Flag Sequence whenever "appreciable time"
   has elapsed since the prior closing Flag Sequence. 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. The maximum value for "appreciable
time" is likely to be no greater than the typing rate of a slow
to average typist, say 1 second.

B. Fast Frame Check Sequence (FCS) Implementation

B.1. FCS Computation Method

The following code provides a table lookup computation for
calculating the Frame Check Sequence as data arrives at the
interface. The table is created by the code in section 2.

```
/*
 * FCS lookup table as calculated by the table generator in section 2.
 */
static unsigned short 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 PPPINITFCS      0xffff  /* Initial FCS value */
    #define PPPGOODFCS      0xf0b8  /* Good final FCS value */


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

        return (fcs);
    }
```

B.2. Fast FCS table generator

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

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

    /*
     * The HDLC polynomial: x**0 + x**5 + x**12 + x**16 (0x8408).
     */
    #define P        0x8408


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

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

```
            printf("0);

        v = b;
        for (i = 8; i--; )
            v = v & 1 ? (v >> 1) ^ P : v >> 1;

        printf("0x%04x", v & 0xFFFF);

        if (++b == 256)
            break;
        printf(",");
    }
    printf("0;0);
  }
```

References

  [1]   Electronic Industries Association, "Interface Between Data
        Terminal Equipment and Data Communications Equipment Employing
        Serial Binary Data Interchange", EIA Standard RS-232-C, August
        1969.

  [2]   International Organization For Standardization, "Data
        Communication - High-level Data Link Control Procedures - Frame
        Structure", ISO Standard 3309-1979, 1979.

  [3]   International Organization For Standardization, "Data
        Communication - High-level Data Link Control Procedures -
        Elements of Procedures", ISO Standard 4335-1979, 1979.

  [4]   International Organization For Standardization, "Data
        Communication - High-Level Data Link Control Procedures -
        Elements of Procedures - Addendum 1", ISO Standard 4335-
        1979/Addendum 1, 1979.

  [5]   International Organization For Standardization, "Information
        Processing Systems - Data Communication - High-level Data Link
        Control Procedures - Frame structure - Addendum 1: Start/stop
        Transmission", Proposed Draft International Standard ISO
        3309:1983/PDAD1, 1984.

  [6]   International Telecommunication Union, CCITT Recommendation X.25,
        "Interface Between Data Terminal Equipment (DTE) and Data Circuit
        Terminating Equipment (DCE) for Terminals Operating in the Packet
        Mode on Public Data Networks", CCITT Red Book, Volume VIII,
        Fascicle VIII.3, Rec. X.25., October 1984.

   [7] Perez, "Byte-wise CRC Calculations", IEEE Micro, June 1983.
        Morse, G., "Calculating CRC's by Bits and Bytes", Byte, September
        1986.

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

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

  [10] Postel, J., "Internet Protocol", RFC 791, USC/Information
        Sciences Institute, September 1981.

  [11] Reynolds, J.K., and J. Postel, "Assigned Numbers", RFC 1010,
        USC/Information Sciences Institute, May 1987.

  [12] Postel, J., "The TCP Maximum Segment Size Option and Related
        Topics", RFC 879, USC/Information Sciences Institute, November
        1983.

Security Considerations

   Security issues are not addressed in this memo.

Author's Address

   This proposal is the product of the Point-to-Point Protocol Working
   Group of the Internet Engineering Task Force (IETF). The working
   group can be contacted via the chair:

   Russ Hobby
   UC Davis
   Computing Services
   Davis, CA 95616

   Phone: (916) 752-0236

   EMail: rdhobby@ucdavis.edu