

Network Working Group  
Request for Comments: 1331  
Obsoletes: RFCs 1171, 1172

W. Simpson  
Daydreamer  
May 1992

The Point-to-Point Protocol (PPP)  
for the  
Transmission of Multi-protocol Datagrams  
over Point-to-Point Links

Status of this Memo

This RFC specifies an IAB standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "IAB Official Protocol Standards" for the standardization state and status of this protocol. 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 comprised of three main components:

1. A method for encapsulating datagrams over serial links.
2. A Link Control Protocol (LCP) for establishing, configuring, and testing the data-link connection.
3. A family of Network Control Protocols (NCPs) for establishing and configuring different network-layer protocols.

This document defines the PPP encapsulation scheme, together with the PPP Link Control Protocol (LCP), an extensible option negotiation protocol which is able to negotiate a rich assortment of configuration parameters and provides additional management functions.

This RFC is a 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-ppp@ucdavis.edu](mailto:ietf-ppp@ucdavis.edu) mailing list.

## Table of Contents

1.	Introduction .....	1
1.1	Specification of Requirements .....	3
1.2	Terminology .....	3
2.	Physical Layer Requirements .....	4
3.	The Data Link Layer .....	5
3.1	Frame Format .....	6
4.	PPP Link Operation .....	10
4.1	Overview .....	10
4.2	Phase Diagram .....	10
4.3	Link Dead (physical-layer not ready) .....	10
4.4	Link Establishment Phase .....	11
4.5	Authentication Phase .....	11
4.6	Network-Layer Protocol Phase .....	12
4.7	Link Termination Phase .....	12
5.	The Option Negotiation Automaton .....	14
5.1	State Diagram .....	15
5.2	State Transition Table .....	16
5.3	States .....	18
5.4	Events .....	20
5.5	Actions .....	24
5.6	Loop Avoidance .....	26
5.7	Counters and Timers .....	27
6.	LCP Packet Formats .....	28
6.1	Configure-Request .....	30
6.2	Configure-Ack .....	31
6.3	Configure-Nak .....	32
6.4	Configure-Reject .....	33
6.5	Terminate-Request and Terminate-Ack .....	35
6.6	Code-Reject .....	36
6.7	Protocol-Reject .....	38
6.8	Echo-Request and Echo-Reply .....	39
6.9	Discard-Request .....	40
7.	LCP Configuration Options .....	42
7.1	Format .....	43
7.2	Maximum-Receive-Unit .....	44
7.3	Async-Control-Character-Map .....	45
7.4	Authentication-Protocol .....	47
7.5	Quality-Protocol .....	49
7.6	Magic-Number .....	51

7.7	Protocol-Field-Compression .....	54
7.8	Address-and-Control-Field-Compression .....	56
APPENDICES .....		58
A.	Asynchronous HDLC .....	58
B.	Fast Frame Check Sequence (FCS) Implementation .....	61
B.1	FCS Computation Method .....	61
B.2	Fast FCS table generator .....	63
C.	LCP Recommended Options .....	64
SECURITY CONSIDERATIONS .....		65
REFERENCES .....		65
ACKNOWLEDGEMENTS .....		66
CHAIR'S ADDRESS .....		66
AUTHOR'S ADDRESS .....		66

## 1. Introduction

### 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.

### Encapsulation

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 de facto 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.

PPP provides an encapsulation protocol over both bit-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. PPP uses HDLC as a basis for the encapsulation.

PPP has been carefully designed to retain compatibility with most commonly used supporting hardware. In addition, 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.

The PPP encapsulation also provides for multiplexing of different network-layer protocols simultaneously over the same link. It is intended that PPP provide a common solution for easy connection of a wide variety of hosts, bridges and routers.

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.

By default, only 8 additional octets are necessary to form the encapsulation. In environments where bandwidth is at a premium, the encapsulation may be shortened to as few as 2 octets. To support high speed hardware implementations, PPP provides that the default encapsulation header and information fields fall on 32-bit boundaries, and allows the trailer to be padded to an arbitrary boundary.

### Link Control Protocol

More importantly, the Point-to-Point Protocol defines more than just an encapsulation scheme. In order to be sufficiently versatile to be portable to a wide variety of environments, PPP provides a Link Control Protocol (LCP). The LCP is used to automatically agree upon the encapsulation format options, handle varying limits on sizes of packets, authenticate the identity of its peer on the link, determine when a link is functioning properly and when it is defunct, detect a looped-back link and other common misconfiguration errors, and terminate the link.

### Network Control Protocols

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 circuit-switched point-to-point links (such as dial-up modem servers). These problems are handled by a family of Network Control Protocols (NCPs), which each manage the specific needs required by their respective network-layer protocols. These NCPs are defined in other documents.

### Configuration

It is intended that PPP be easy to configure. By design, the standard defaults should handle all common configurations. The implementor may specify improvements to the default configuration, which are automatically communicated to the peer without operator intervention. Finally, the operator may explicitly configure options for the link which enable the link to operate in environments where it would otherwise be impossible.

This self-configuration is implemented through an extensible option negotiation mechanism, wherein each end of the link describes to the other its capabilities and requirements. Although the option negotiation mechanism described in this

document is specified in terms of the Link Control Protocol (LCP), the same facilities may be used by the Internet Protocol Control Protocol (IPCP) and others in the family of NCPs.

### 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 should 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:

#### peer

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

#### silently discard

This means 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

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 full-duplex circuit, either dedicated or circuit-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 any particular synchronous encoding, such as FM, NRZ, or NRZI.

### Implementation Note:

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.

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).

### Implementation Note:

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 Option Negotiation Automaton (described below).

### 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.

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.

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.



### 3.1. Frame Format

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

Flag 01111110	Address 11111111	Control 00000011	Protocol 16 bits	Information *
FCS 16 bits		Flag 01111110	Inter-frame Fill or next Address	

#### Inter-frame Time Fill

For asynchronous links, inter-frame time fill SHOULD be accomplished in the same manner as inter-octet time fill, by transmitting continuous "1" bits (mark-hold state).

For synchronous links, the Flag Sequence SHOULD be transmitted during inter-frame time fill. There is no provision for inter-octet time fill.

#### Implementation Note:

Mark idle (continuous ones) SHOULD NOT be used for idle synchronous inter-frame time fill. However, certain types of circuit-switched links require the use of mark idle, particularly those that calculate accounting based on bit activity. When mark idle is used on a synchronous link, the implementation MUST ensure at least 15 consecutive "1" bits between Flags, and that the Flag Sequence is generated at the beginning and end of a frame.

#### 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).

The Flag is a frame separator. Only one Flag is required between two frames. Two consecutive Flags constitute an empty frame, which is ignored.

#### Implementation Note:

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.

#### 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 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, and reported through the normal network management facility.

#### 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 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.

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 MUST be considered as having an unrecognized Protocol, and handled as specified by the LCP. The Protocol field is transmitted and received most significant octet first.

Protocol field values in the "0---" to "3---" range identify the network-layer protocol of specific datagrams, and values in the "8---" to "b---" range identify datagrams belonging to the associated Network Control Protocols (NCPs), if any.

Protocol field values in the "4---" to "7---" range are used for protocols with low volume traffic which have no associated NCP. Protocol field values in the "c---" to "f---" range identify

datagrams as link-layer Control Protocols (such as LCP).

The most up-to-date values of the Protocol field are specified in the most recent "Assigned Numbers" RFC [11]. Current values are assigned as follows:

Value (in hex)	Protocol Name
0001 to 001f	reserved (transparency inefficient)
0021	Internet Protocol
0023	OSI Network Layer
0025	Xerox NS IDP
0027	DECnet Phase IV
0029	Appletalk
002b	Novell IPX
002d	Van Jacobson Compressed TCP/IP
002f	Van Jacobson Uncompressed TCP/IP
0031	Bridging PDU
0033	Stream Protocol (ST-II)
0035	Banyan Vines
0037	reserved (until 1993)
00ff	reserved (compression inefficient)
0201	802.1d Hello Packets
0231	Luxcom
0233	Sigma Network Systems
8021	Internet Protocol Control Protocol
8023	OSI Network Layer 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
8031	Bridging NCP
8033	Stream Protocol Control Protocol
8035	Banyan Vines Control Protocol
c021	Link Control Protocol
c023	Password Authentication Protocol
c025	Link Quality Report
c223	Challenge Handshake Authentication Protocol

Developers of new protocols MUST obtain a number from the Internet Assigned Numbers Authority (IANA), at IANA@isi.edu.

## 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 negotiation, 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 octets from real information.

## Frame Check Sequence (FCS) Field

The Frame Check Sequence field is normally 16 bits (two octets). 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 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 the FCS field itself. The FCS is transmitted with the coefficient of the highest term first.

Note: When octets are received which are flagged in the Async-Control-Character-Map, they are discarded before calculating the FCS. See the description in Appendix A.

For more information on the specification of the FCS, see ISO 3309 [2] or CCITT X.25 [6].

Note: A fast, table-driven implementation of the 16-bit FCS algorithm is shown in Appendix B. This implementation is based on [7], [8], and [9].

## 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. PPP Link Operation

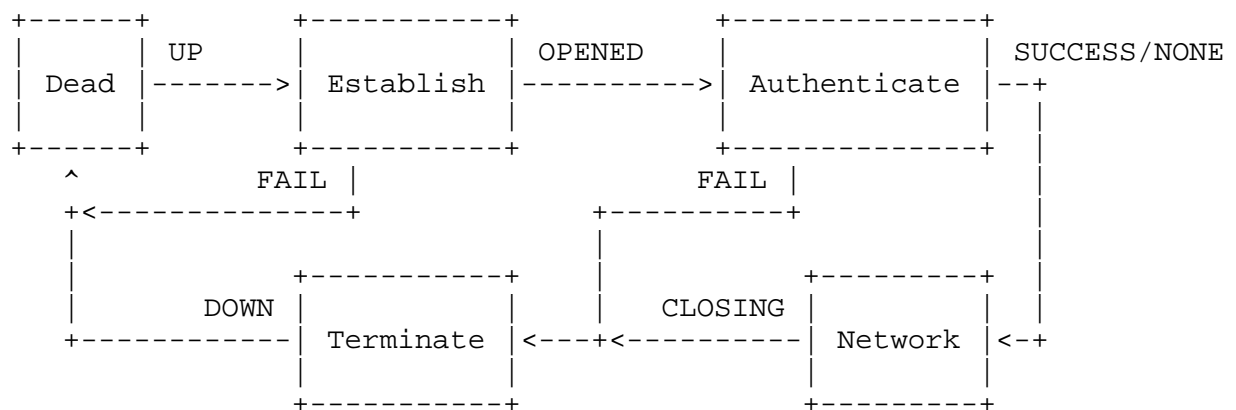
### 4.1. Overview

In order to establish communications over a point-to-point link, each end of the PPP link must first send LCP packets to configure and test the data link. After the link has been established, the peer may be authenticated. Then, PPP must 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 (an inactivity timer expires or network administrator intervention).

### 4.2. Phase Diagram

In the process of configuring, maintaining and terminating the point-to-point link, the PPP link goes through several distinct phases:



### 4.3. Link Dead (physical-layer not ready)

The link necessarily begins and ends with this phase. When an external event (such as carrier detection or network administrator configuration) indicates that the physical-layer is ready to be used, PPP will proceed to the Link Establishment phase.

During this phase, the LCP automaton (described below) will be in the Initial or Starting states. The transition to the Link Establishment phase will signal an Up event to the automaton.

#### Implementation Note:

Typically, a link will return to this phase automatically after the disconnection of a modem. In the case of a hard-wired line, this phase may be extremely short -- merely long enough to detect the presence of the device.

#### 4.4. Link Establishment Phase

The Link Control Protocol (LCP) is used to establish the connection through an exchange of Configure packets. This exchange is complete, and the LCP Opened state entered, once a Configure-Ack packet (described below) has been both sent and received. Any non-LCP packets received during this phase MUST be silently discarded.

All Configuration Options are assumed to be at default values unless altered by the configuration exchange. See the section on LCP Configuration Options for further discussion.

It is important to note that only Configuration Options which are independent of particular network-layer protocols are configured by LCP. Configuration of individual network-layer protocols is handled by separate Network Control Protocols (NCPs) during the Network-Layer Protocol phase.

#### 4.5. Authentication Phase

On some links it may be desirable to require a peer to authenticate itself before allowing network-layer protocol packets to be exchanged.

By default, authentication is not necessary. If an implementation requires that the peer authenticate with some specific authentication protocol, then it MUST negotiate the use of that authentication protocol during Link Establishment phase.

Authentication SHOULD take place as soon as possible after link establishment. However, link quality determination MAY occur concurrently. An implementation MUST NOT allow the exchange of link quality determination packets to delay authentication indefinitely.

Advancement from the Authentication phase to the Network-Layer Protocol phase MUST NOT occur until the peer is successfully authenticated using the negotiated authentication protocol. In the event of failure to authenticate, PPP SHOULD proceed instead to the Link Termination phase.

#### 4.6. Network-Layer Protocol Phase

Once PPP has finished the previous phases, each network-layer protocol (such as IP) **MUST** be separately configured by the appropriate Network Control Protocol (NCP).

Each NCP may be Opened and Closed at any time.

##### Implementation Note:

Because an implementation may initially use a significant amount of time for link quality determination, implementations **SHOULD** avoid fixed timeouts when waiting for their peers to configure a NCP.

After a NCP has reached the Opened state, PPP will carry the corresponding network-layer protocol packets. Any network-layer protocol packets received when the corresponding NCP is not in the Opened state **SHOULD** be silently discarded.

During this phase, link traffic consists of any possible combinations of LCP, NCP, and network-layer protocol packets. Any NCP or network-layer protocol packets received during any other phase **SHOULD** be silently discarded.

##### Implementation Note:

There is an exception to the preceding paragraphs, due to the availability of the LCP Protocol-Reject (described below). While LCP is in the Opened state, any protocol packet which is unsupported by the implementation **MUST** be returned in a Protocol-Reject. Only supported protocols are silently discarded.

#### 4.7. Link Termination Phase

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

LCP is used to close the link through an exchange of Terminate packets. When the link is closing, PPP informs the network-layer protocols so that they may take appropriate action.

After the exchange of Terminate packets, the implementation **SHOULD** signal the physical-layer to disconnect in order to enforce the termination of the link, particularly in the case of an authentication failure. The sender of the Terminate-Request **SHOULD**

disconnect after receiving a Terminate-Ack, or after the Restart counter expires. The receiver of a Terminate-Request SHOULD wait for the peer to disconnect, and MUST NOT disconnect until at least one Restart time has passed after sending a Terminate-Ack. PPP SHOULD proceed to the Link Dead phase.

Implementation Note:

The closing of the link by LCP is sufficient. There is no need for each NCP to send a flurry of Terminate packets. Conversely, the fact that a NCP has Closed is not sufficient reason to cause the termination of the PPP link, even if that NCP was the only currently NCP in the Opened state.



## 5. The Option Negotiation Automaton

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

Some types of packets -- Configure-Naks and Configure-Rejects, or Code-Rejects and Protocol-Rejects, or Echo-Requests, Echo-Replies and Discard-Requests -- are not differentiated in the automaton descriptions. As will be described later, these packets do indeed serve different functions. However, they always cause the same transitions.

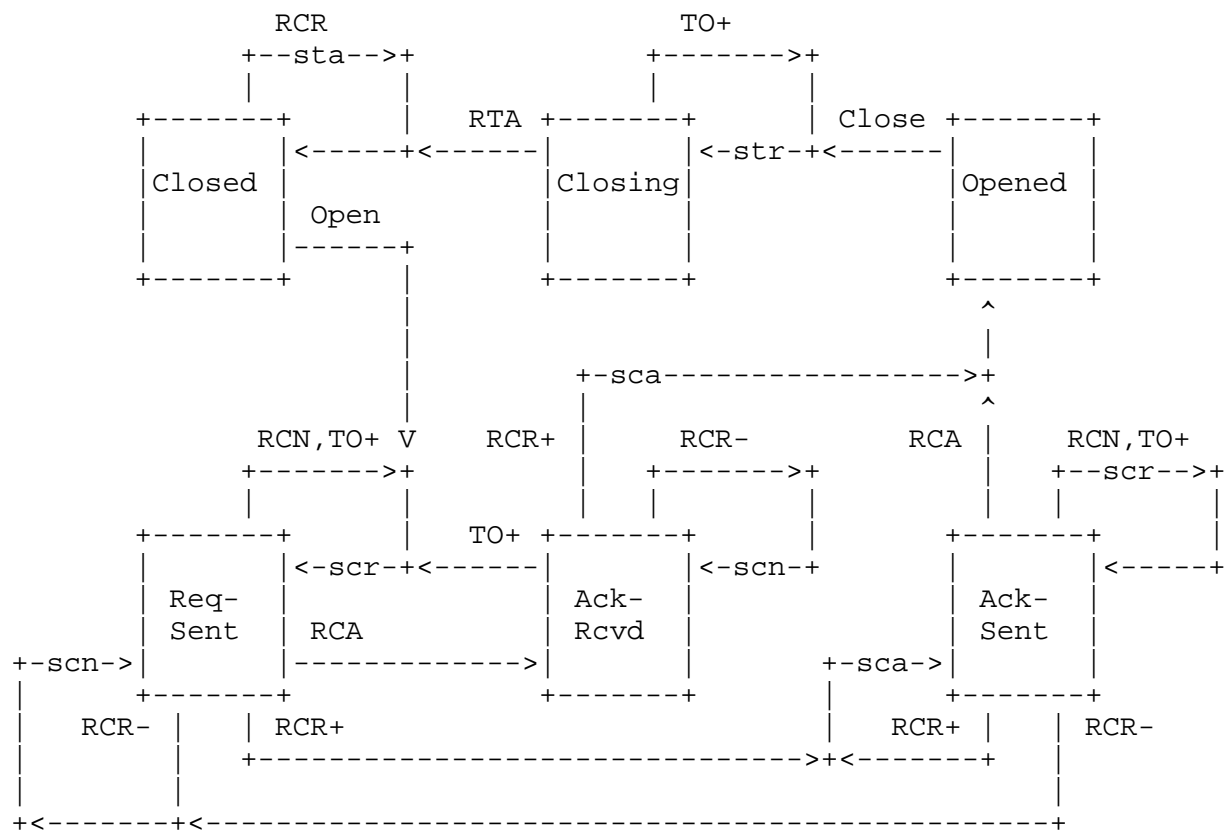
Events	Actions
Up = lower layer is Up	tlu = This-Layer-Up
Down = lower layer is Down	tld = This-Layer-Down
Open = administrative Open	tls = This-Layer-Start
Close = administrative Close	tlf = This-Layer-Finished
TO+ = Timeout with counter > 0	irc = initialize restart counter
TO- = Timeout with counter expired	zrc = zero restart counter
RCR+ = Receive-Configure-Request (Good)	scr = Send-Configure-Request
RCR- = Receive-Configure-Request (Bad)	
RCA = Receive-Configure-Ack	sca = Send-Configure-Ack
RCN = Receive-Configure-Nak/Rej	scn = Send-Configure-Nak/Rej
RTR = Receive-Terminate-Request	str = Send-Terminate-Request
RTA = Receive-Terminate-Ack	sta = Send-Terminate-Ack
RUC = Receive-Unknown-Code	scj = Send-Code-Reject
RXJ+ = Receive-Code-Reject (permitted) or Receive-Protocol-Reject	
RXJ- = Receive-Code-Reject (catastrophic) or Receive-Protocol-Reject	
RXR = Receive-Echo-Request or Receive-Echo-Reply or Receive-Discard-Request	ser = Send-Echo-Reply
	- = illegal action

## 5.1. State Diagram

The simplified state diagram which follows describes the sequence of events for reaching agreement on Configuration Options (opening the PPP link) and for later termination of the link.

This diagram is not a complete representation of the automaton. Implementation MUST be done by consulting the actual state transition table.

Events are in upper case. Actions are in lower case. For these purposes, the state machine is initially in the Closed state. Once the Opened state has been reached, both ends of the link have met the requirement of having both sent and received a Configure-Ack packet.



## 5.2. 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. Multiple actions are separated by commas, and may continue on succeeding lines as space requires. The state may be followed by a letter, which indicates an explanatory footnote.

### Rationale:

In previous versions of this table, a simplified non-deterministic finite-state automaton was used, with considerable detailed information specified in the semantics. This lead to interoperability problems from differing interpretations.

This table functions similarly to the previous versions, with the up/down flags expanded to explicit states, and the active/passive paradigm eliminated. It is believed that this table interoperates with previous versions better than those versions themselves.

Events	State					
	0 Initial	1 Starting	2 Closed	3 Stopped	4 Closing	5 Stopping
Up	2	irc,scr/6	-	-	-	-
Down	-	-	0	tls/1	0	1
Open	tls/1	1	irc,scr/6	3r	5r	5r
Close	0	0	2	2	4	4
TO+	-	-	-	-	str/4	str/5
TO-	-	-	-	-	tlf/2	tlf/3
RCR+	-	-	sta/2	irc,scr,sca/8	4	5
RCR-	-	-	sta/2	irc,scr,scn/6	4	5
RCA	-	-	sta/2	sta/3	4	5
RCN	-	-	sta/2	sta/3	4	5
RTR	-	-	sta/2	sta/3	sta/4	sta/5
RTA	-	-	2	3	tlf/2	tlf/3
RUC	-	-	scj/2	scj/3	scj/4	scj/5
RXJ+	-	-	2	3	4	5
RXJ-	-	-	tlf/2	tlf/3	tlf/2	tlf/3
RXR	-	-	2	3	4	5

Events	State			
	6 Req-Sent	7 Ack-Rcvd	8 Ack-Sent	9 Opened
Up	-	-	-	-
Down	1	1	1	tld/1
Open	6	7	8	9r
Close	irc,str/4	irc,str/4	irc,str/4	tld,irc,str/4
TO+	scr/6	scr/6	scr/8	-
TO-	tlf/3p	tlf/3p	tlf/3p	-
RCR+	sca/8	sca,tlu/9	sca/8	tld,scr,sca/8
RCR-	scn/6	scn/7	scn/6	tld,scr,scn/6
RCA	irc/7	scr/6x	irc,tlu/9	tld,scr/6x
RCN	irc,scr/6	scr/6x	irc,scr/8	tld,scr/6x
RTR	sta/6	sta/6	sta/6	tld,zrc,sta/5
RTA	6	6	8	tld,scr/6
RUC	scj/6	scj/7	scj/8	tld,scj,scr/6
RXJ+	6	6	8	9
RXJ-	tlf/3	tlf/3	tlf/3	tld,irc,str/5
RXR	6	7	8	ser/9

The states in which the Restart timer is running are identifiable by the presence of TO events. Only the Send-Configure-Request, Send-Terminate-Request and Zero-Restart-Counter actions start or re-start the Restart timer. The Restart timer SHOULD be stopped when transitioning from any state where the timer is running to a state where the timer is not running.

[p] Passive option; see Stopped state discussion.

[r] Restart option; see Open event discussion.

[x] Crossed connection; see RCA event discussion.

### 5.3. States

Following is a more detailed description of each automaton state.

#### Initial

In the Initial state, the lower layer is unavailable (Down), and no Open has occurred. The Restart timer is not running in the Initial state.

#### Starting

The Starting state is the Open counterpart to the Initial state. An administrative Open has been initiated, but the lower layer is still unavailable (Down). The Restart timer is not running in the Starting state.

When the lower layer becomes available (Up), a Configure-Request is sent.

#### Closed

In the Closed state, the link is available (Up), but no Open has occurred. The Restart timer is not running in the Closed state.

Upon reception of Configure-Request packets, a Terminate-Ack is sent. Terminate-Acks are silently discarded to avoid creating a loop.

#### Stopped

The Stopped state is the Open counterpart to the Closed state. It is entered when the automaton is waiting for a Down event after the This-Layer-Finished action, or after sending a Terminate-Ack. The Restart timer is not running in the Stopped state.

Upon reception of Configure-Request packets, an appropriate response is sent. Upon reception of other packets, a Terminate-Ack is sent. Terminate-Acks are silently discarded to avoid creating a loop.

#### Rationale:

The Stopped state is a junction state for link termination, link configuration failure, and other automaton failure modes. These potentially separate states have been combined.

There is a race condition between the Down event response (from

the This-Layer-Finished action) and the Receive-Configure-Request event. When a Configure-Request arrives before the Down event, the Down event will supercede by returning the automaton to the Starting state. This prevents attack by repetition.

#### Implementation Option:

After the peer fails to respond to Configure-Requests, an implementation MAY wait passively for the peer to send Configure-Requests. In this case, the This-Layer-Finished action is not used for the TO- event in states Req-Sent, Ack-Rcvd and Ack-Sent.

This option is useful for dedicated circuits, or circuits which have no status signals available, but SHOULD NOT be used for switched circuits.

#### Closing

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

Upon reception of a Terminate-Ack, the Closed state is 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-Terminate times, this action may be skipped, and the Closed state may be entered.

#### Stopping

The Stopping state is the Open counterpart to the Closing state. A Terminate-Request has been sent and the Restart timer is running, but a Terminate-Ack has not yet been received.

#### Rationale:

The Stopping state provides a well defined opportunity to terminate a link before allowing new traffic. After the link has terminated, a new configuration may occur via the Stopped or Starting states.

#### Request-Sent

In the Request-Sent state an attempt is made to configure 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.

#### Ack-Received

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 sent.

#### Ack-Sent

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

#### Opened

In the Opened state, a Configure-Ack has been both sent and received. The Restart timer is not running in the Opened state.

When entering the Opened state, the implementation SHOULD signal the upper layers that it is now Up. Conversely, when leaving the Opened state, the implementation SHOULD signal the upper layers that it is now Down.

### 5.4. Events

Transitions and actions in the automaton are caused by events.

#### Up

The Up event occurs when a lower layer indicates that it is ready to carry packets. Typically, this event is used to signal LCP that the link is entering Link Establishment phase, or used to signal a NCP that the link is entering Network-Layer Protocol phase.

#### Down

The Down event occurs when a lower layer indicates that it is no longer ready to carry packets. Typically, this event is used to signal LCP that the link is entering Link Dead phase, or used to signal a NCP that the link is leaving Network-Layer Protocol phase.

#### Open

The Open event indicates that the link is administratively available for traffic; that is, the network administrator (human

or program) has indicated that the link is allowed to be Opened. When this event occurs, and the link is not in the Opened state, the automaton attempts to send configuration packets to the peer.

If the automaton is not able to begin configuration (the lower layer is Down, or a previous Close event has not completed), the establishment of the link is automatically delayed.

When a Terminate-Request is received, or other events occur which cause the link to become unavailable, the automaton will progress to a state where the link is ready to re-open. No additional administrative intervention should be necessary.

#### Implementation Note:

Experience has shown that users will execute an additional Open command when they want to renegotiate the link. Since this is not the meaning of the Open event, it is suggested that when an Open user command is executed in the Opened, Closing, Stopping, or Stopped states, the implementation issue a Down event, immediately followed by an Up event. This will cause the renegotiation of the link, without any harmful side effects.

#### Close

The Close event indicates that the link is not available for traffic; that is, the network administrator (human or program) has indicated that the link is not allowed to be Opened. When this event occurs, and the link is not in the Closed state, the automaton attempts to terminate the connection. Further attempts to re-configure the link are denied until a new Open event occurs.

#### Timeout (TO+,TO-)

This event indicates the expiration of the Restart timer. The Restart timer is used to time responses to Configure-Request and Terminate-Request packets.

The TO+ event indicates that the Restart counter continues to be greater than zero, which triggers the corresponding Configure-Request or Terminate-Request packet to be retransmitted.

The TO- event indicates that the Restart counter is not greater than zero, and no more packets need to be retransmitted.

#### Receive-Configure-Request (RCR+,RCR-)

This event occurs when a Configure-Request packet is received from



the peer. The Configure-Request packet indicates the desire to open a connection and may specify Configuration Options. The Configure-Request packet is more fully described in a later section.

The RCR+ event indicates that the Configure-Request was acceptable, and triggers the transmission of a corresponding Configure-Ack.

The RCR- event indicates that the Configure-Request was unacceptable, and triggers the transmission of a corresponding Configure-Nak or Configure-Reject.

#### Implementation Note:

These events may occur on a connection which is already in the Opened state. The implementation MUST be prepared to immediately renegotiate the Configuration Options.

#### Receive-Configure-Ack (RCA)

The Receive-Configure-Ack event occurs when a valid Configure-Ack packet is received from the peer. The Configure-Ack packet is a positive response to a Configure-Request packet. An out of sequence or otherwise invalid packet is silently discarded.

#### Implementation Note:

Since the correct packet has already been received before reaching the Ack-Rcvd or Opened states, it is extremely unlikely that another such packet will arrive. As specified, all invalid Ack/Nak/Rej packets are silently discarded, and do not affect the transitions of the automaton.

However, it is not impossible that a correctly formed packet will arrive through a coincidentally-timed cross-connection. It is more likely to be the result of an implementation error. At the very least, this occurrence should be logged.

#### Receive-Configure-Nak/Rej (RCN)

This event occurs when a valid Configure-Nak or Configure-Reject packet is received from the peer. The Configure-Nak and Configure-Reject packets are negative responses to a Configure-Request packet. An out of sequence or otherwise invalid packet is silently discarded.

#### Implementation Note:

Although the Configure-Nak and Configure-Reject cause the same state transition in the automaton, these packets have significantly different effects on the Configuration Options sent in the resulting Configure-Request packet.

#### Receive-Terminate-Request (RTR)

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

#### Implementation Note:

This event is not identical to the Close event (see above), and does not override the Open commands of the local network administrator. The implementation MUST be prepared to receive a new Configure-Request without network administrator intervention.

#### Receive-Terminate-Ack (RTA)

The Receive-Terminate-Ack event occurs when a Terminate-Ack packet is received from the peer. The Terminate-Ack packet is usually a response to a Terminate-Request packet. The Terminate-Ack packet may also indicate that the peer is in Closed or Stopped states, and serves to re-synchronize the link configuration.

#### Receive-Unknown-Code (RUC)

The Receive-Unknown-Code event occurs when an un-interpretable packet is received from the peer. A Code-Reject packet is sent in response.

#### Receive-Code-Reject, Receive-Protocol-Reject (RXJ+,RXJ-)

This event occurs when a Code-Reject or a Protocol-Reject packet is received from the peer.

The RXJ+ event arises when the rejected value is acceptable, such as a Code-Reject of an extended code, or a Protocol-Reject of a NCP. These are within the scope of normal operation. The implementation MUST stop sending the offending packet type.

The RXJ- event arises when the rejected value is catastrophic, such as a Code-Reject of Configure-Request, or a Protocol-Reject of LCP! This event communicates an unrecoverable error that

terminates the connection.

Receive-Echo-Request, Receive-Echo-Reply, Receive-Discard-Request (RXR)

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

## 5.5. Actions

Actions in the automaton are caused by events and typically indicate the transmission of packets and/or the starting or stopping of the Restart timer.

Illegal-Event (-)

This indicates an event that SHOULD NOT occur. The implementation probably has an internal error.

This-Layer-Up (tlu)

This action indicates to the upper layers that the automaton is entering the Opened state.

Typically, this action MAY be used by the LCP to signal the Up event to a NCP, Authentication Protocol, or Link Quality Protocol, or MAY be used by a NCP to indicate that the link is available for its traffic.

This-Layer-Down (tld)

This action indicates to the upper layers that the automaton is leaving the Opened state.

Typically, this action MAY be used by the LCP to signal the Down event to a NCP, Authentication Protocol, or Link Quality Protocol, or MAY be used by a NCP to indicate that the link is no longer available for its traffic.

This-Layer-Start (tls)

This action indicates to the lower layers that the automaton is entering the Starting state, and the lower layer is needed for the link. The lower layer SHOULD respond with an Up event when the lower layer is available.

This action is highly implementation dependent.

#### This-Layer-Finished (tlf)

This action indicates to the lower layers that the automaton is entering the Stopped or Closed states, and the lower layer is no longer needed for the link. The lower layer SHOULD respond with a Down event when the lower layer has terminated.

Typically, this action MAY be used by the LCP to advance to the Link Dead phase, or MAY be used by a NCP to indicate to the LCP that the link may terminate when there are no other NCPs open.

This action is highly implementation dependent.

#### Initialize-Restart-Counter (irc)

This action sets the Restart counter to the appropriate value (Max-Terminate or Max-Configure). The counter is decremented for each transmission, including the first.

#### Zero-Restart-Counter (zrc)

This action sets the Restart counter to zero.

#### Implementation Note:

This action enables the FSA to pause before proceeding to the desired final state. In addition to zeroing the Restart counter, the implementation MUST set the timeout period to an appropriate value.

#### Send-Configure-Request (scr)

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

#### Send-Configure-Ack (sca)

The Send-Configure-Ack action transmits a Configure-Ack packet. This acknowledges the reception 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 reception 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 versus Configure-Reject is more fully described in the section on LCP Packet Formats.

### Send-Terminate-Request (str)

The Send-Terminate-Request action transmits a Terminate-Request packet. This indicates the desire to close a connection. The Restart timer is started when the Terminate-Request packet is transmitted, to guard against packet loss. The Restart counter is decremented each time a Terminate-Request is sent.

### Send-Terminate-Ack (sta)

The Send-Terminate-Ack action transmits a Terminate-Ack packet. This acknowledges the reception of a Terminate-Request packet or otherwise serves to synchronize the state machines.

### Send-Code-Reject (scj)

The Send-Code-Reject action transmits a Code-Reject packet. This indicates the reception of an unknown type of packet.

### Send-Echo-Reply (ser)

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

## 5.6. Loop Avoidance

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.

## 5.7. Counters and Timers

### Restart Timer

There is one special timer used by the automaton. The Restart timer is used to time transmissions of Configure-Request and Terminate-Request packets. Expiration of the Restart timer causes a Timeout event, and retransmission of the corresponding Configure-Request or Terminate-Request packet. The Restart timer MUST be configurable, but MAY default to three (3) seconds.

#### Implementation Note:

The Restart timer SHOULD be based on the speed of the link. The default value is designed for low speed (19,200 bps or less), high switching latency links (typical telephone lines). Higher speed links, or links with low switching latency, SHOULD have correspondingly faster retransmission times.

### Max-Terminate

There is one required restart counter for Terminate-Requests. Max-Terminate indicates the number of Terminate-Request packets sent without receiving a Terminate-Ack before assuming that the peer is unable to respond. Max-Terminate MUST be configurable, but should default to two (2) transmissions.

### Max-Configure

A similar counter is recommended for Configure-Requests. Max-Configure indicates the number of Configure-Request packets sent without receiving a valid Configure-Ack, Configure-Nak or Configure-Reject before assuming that the peer is unable to respond. Max-Configure MUST be configurable, but should default to ten (10) transmissions.

### Max-Failure

A related counter is recommended for Configure-Nak. Max-Failure indicates the number of Configure-Nak packets sent without sending a Configure-Ack before assuming that configuration is not converging. Any further Configure-Nak packets are converted to Configure-Reject packets. Max-Failure MUST be configurable, but should default to ten (10) transmissions.

## 6. LCP Packet Formats

There are three classes of LCP packets:

1. Link Configuration packets used to establish and configure a link (Configure-Request, Configure-Ack, Configure-Nak and Configure-Reject).
2. Link Termination packets used to terminate a link (Terminate-Request and Terminate-Ack).
3. Link Maintenance packets used to manage and debug a link (Code-Reject, Protocol-Reject, Echo-Request, Echo-Reply, and Discard-Request).

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 will 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.

Regardless of which Configuration Options are enabled, all LCP Link Configuration, Link Termination, and Code-Reject packets (codes 1 through 7) 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.

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. When a packet is received with an invalid Code field, a Code-Reject packet is transmitted.

The most up-to-date values of the LCP Code field are specified in the most recent "Assigned Numbers" RFC [11]. Current values 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
12	RESERVED

## Identifier

The Identifier field is one octet and aids in matching requests and replies. When a packet is received with an invalid Identifier field, the packet is silently discarded.

## 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. When a packet is received with an invalid Length field, the packet is silently discarded.

## 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.



## 6.1. 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.

## 6.2. 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. Additionally, the Configuration Options in a Configure-Ack must exactly match those of the last transmitted Configure-Request. Invalid packets are silently discarded.

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.

### 6.3. 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 are 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. Options which have no value fields (boolean options) use the Configure-Reject reply instead.

Finally, an implementation may be configured to request 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 peer to list that option in its next Configure-Request packet. Any value fields for the option MUST indicate values acceptable to the Configure-Nak sender.

On reception of a Configure-Nak, the Identifier field must match that of the last transmitted Configure-Request. Invalid packets are silently discarded.

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

Some Configuration Options have a variable length. Since the Nak'd Option has been modified by the peer, the implementation MUST be able to handle an Option length which is different from the original Configure-Request.

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.

### 6.4. 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 administrator), 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 unacceptable Configuration Options from the Configure-Request. All recognizable and negotiable Configuration Options are filtered out of the Configure-Reject, but otherwise the Configuration Options MUST NOT be reordered or modified in any way.

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

Reception of a valid 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.

## 6.5. 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 lower layer indicates that it has gone down, or a sufficiently large number have been transmitted such that the peer 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 peer is in the Closed or Stopped states, or is otherwise in need of re-negotiation.

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 peer's established maximum Information field length minus four.

## 6.6. 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-Information field.

Upon reception of a Code-Reject, the implementation 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-Information

The Rejected-Information field contains a copy of the LCP packet which is being rejected. It begins with the Information field, and does not include any PPP Data Link Layer headers nor the FCS.

The Rejected-Information MUST be truncated to comply with the peer's established maximum Information field length.



## 6.7. Protocol-Reject

### Description

Reception of a PPP frame with an unknown Data Link Layer Protocol indicates that the peer is attempting to use a protocol which is unsupported. This usually occurs when the peer attempts to configure a new protocol. If the LCP state machine is in the Opened state, then this error MUST be reported back to the peer 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 inducing packet copied to the Rejected-Information field.

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 Opened state. Protocol-Reject packets received in any state other than the LCP Opened state SHOULD be silently discarded.

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 nor the FCS. The Rejected-Information MUST be truncated to comply with the peer's established maximum Information field length.

## 6.8. 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), the Identifier field set, the local Magic-Number inserted, and the Data field filled with any desired data, up to but not exceeding the peer's established maximum Information field length minus eight.

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, the local Magic-Number inserted, and the Data field copied from the Echo-Request, truncating as necessary to avoid exceeding the peer's established maximum Information field length.

Echo-Request and Echo-Reply packets may only be sent in the LCP Opened state. Echo-Request and Echo-Reply packets received in any state other than the LCP Opened state SHOULD be silently discarded.

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 links which are in the looped-back condition. Unless modified by a Configuration Option, the Magic-Number MUST be transmitted as zero and MUST be ignored on reception. See the Magic-Number Configuration Option for further explanation.

### 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 peer's established maximum Information field length minus eight.

## 6.9. 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 for numerous other functions.

A discard sender transmits a LCP packet with the Code field set to 11 (Discard-Request) the Identifier field set, the local Magic-Number inserted, and the Data field filled with any desired data, up to but not exceeding the peer's established maximum Information field length minus eight.

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

Discard-Request packets may only be sent in the LCP Opened 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 links which are in the looped-back condition. Unless modified by a configuration option, the Magic-Number MUST be transmitted as zero and MUST be ignored on reception. See the Magic-Number Configuration Option for further explanation.

#### 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 peer's established maximum Information field length minus four.

## 7. LCP 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, etc. 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 length of the LCP packet.

Unless otherwise specified, each Configuration Option is not listed more than once in a Configuration Options list. Some Configuration Options 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.

### 7.1. 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 LCP Option Type field are specified in the most recent "Assigned Numbers" RFC [11]. Current values are assigned as follows:

- |   |                                       |
|---|---------------------------------------|
| 1 | Maximum-Receive-Unit                  |
| 2 | Async-Control-Character-Map           |
| 3 | Authentication-Protocol               |
| 4 | Quality-Protocol                      |
| 5 | Magic-Number                          |
| 6 | RESERVED                              |
| 7 | Protocol-Field-Compression            |
| 8 | Address-and-Control-Field-Compression |

#### 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.

## 7.2. Maximum-Receive-Unit

### Description

This Configuration Option may be sent to inform the peer that the implementation can receive larger frames, or to request that the peer send smaller frames. If smaller frames are requested, an implementation **MUST** still be able to receive 1500 octet frames in case link synchronization is lost.

A summary of the Maximum-Receive-Unit 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										Maximum-Receive-Unit																			

### Type

1

### Length

4

### Maximum-Receive-Unit

The Maximum-Receive-Unit field is two octets and indicates the new maximum receive unit. The Maximum-Receive-Unit covers only the Data Link Layer Information field. It does not include the header, padding, FCS, nor any transparency bits or bytes.

### Default

1500

### 7.3. Async-Control-Character-Map

#### Description

This Configuration Option provides a way to negotiate the use of control character mapping on asynchronous links. By default, PPP maps all control characters into an appropriate two character sequence. However, it is rarely necessary to map all control characters and often it is unnecessary to map any characters. A PPP implementation may use this Configuration Option to inform the peer which control characters must remain mapped and which control characters need not remain mapped when the peer sends them. The peer may still send these control characters in mapped format if it is necessary because of constraints at the peer.

There may be some use of synchronous-to-asynchronous converters (some built into modems) in Point-to-Point links resulting in a synchronous PPP implementation on one end of a link and an asynchronous implementation on the other. It is the responsibility of the converter to do all mapping conversions during operation. To enable this functionality, synchronous PPP implementations **MUST** always accept a Async-Control-Character-Map Configuration Option (it **MUST** always respond to an LCP Configure-Request specifying this Configuration Option with an LCP Configure-Ack). However, acceptance of this Configuration Option does not imply that the synchronous implementation will do any character mapping, since synchronous PPP uses bit-stuffing rather than character-stuffing. Instead, all such character mapping will be performed by the asynchronous-to-synchronous converter.

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	2	3	4	5	6	7	8	9										
Type										Length										Async-Control-Character-Map																													
ACCM (cont)																																																	

Type

2



Length

6

Async-Control-Character-Map

The Async-Control-Character-Map field is four octets and indicates the new async control character map. The map is encoded in big-endian fashion where each numbered bit corresponds to the ASCII control character of the same value. If the bit is cleared to zero, then that ASCII control character need not be mapped. If the bit is set to one, then that ASCII control character must remain mapped. E.g., if bit 19 is set to zero, then the ASCII control character 19 (DC3, Control-S) may be sent in the clear.

Note: The bit ordering of the map is as described in section 3.1, Most Significant Bit to Least Significant Bit. 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.

Default

All ones (0xffffffff).

#### 7.4. Authentication-Protocol

##### Description

On some links it may be desirable to require a peer to authenticate itself before allowing network-layer protocol packets to be exchanged. This Configuration Option provides a way to negotiate the use of a specific authentication protocol. By default, authentication is not necessary.

An implementation SHOULD NOT include multiple Authentication-Protocol Configuration Options in its Configure-Request packets. Instead, it SHOULD attempt to configure the most desirable protocol first. If that protocol is Rejected, then the implementation could attempt the next most desirable protocol in the next Configure-Request.

An implementation receiving a Configure-Request specifying Authentication-Protocols MAY choose at most one of the negotiable authentication protocols and MUST send a Configure-Reject including the other specified authentication protocols. The implementation MAY reject all of the proposed authentication protocols.

If an implementation sends a Configure-Ack with this Configuration Option, then it is agreeing to authenticate with the specified protocol. An implementation receiving a Configure-Ack with this Configuration Option SHOULD expect the peer to authenticate with the acknowledged protocol.

There is no requirement that authentication be full duplex or that the same protocol be used in both directions. It is perfectly acceptable for different protocols to be used in each direction. This will, of course, depend on the specific protocols negotiated.

A summary of the Authentication-Protocol 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										Authentication-Protocol																			
Data ...																																							

Type

3

Length

>= 4

Authentication-Protocol

The Authentication-Protocol field is two octets and indicates the authentication protocol desired. Values for this field are always the same as the PPP Data Link Layer Protocol field values for that same authentication protocol.

The most up-to-date values of the Authentication-Protocol field are specified in the most recent "Assigned Numbers" RFC [11]. Current values are assigned as follows:

Value (in hex)	Protocol
c023	Password Authentication Protocol
c223	Challenge Handshake Authentication Protocol

Data

The Data field is zero or more octets and contains additional data as determined by the particular protocol.

Default

No authentication protocol necessary.

## 7.5. Quality-Protocol

### Description

On some links it may be desirable to determine when, and how often, the link is dropping data. This process is called link quality monitoring.

This Configuration Option provides a way to negotiate the use of a specific protocol for link quality monitoring. By default, link quality monitoring is disabled.

There is no requirement that quality monitoring be full duplex or that the same protocol be used in both directions. It is perfectly acceptable for different protocols to be used in each direction. This will, of course, depend on the specific protocols negotiated.

A summary of the Quality-Protocol 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      |      Quality-Protocol      |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|      Data ...  |
+---+---+---+

```

### Type

4

### Length

>= 4

### Quality-Protocol

The Quality-Protocol field is two octets and indicates the link quality monitoring protocol desired. Values for this field are always the same as the PPP Data Link Layer Protocol field values for that same monitoring protocol.

The most up-to-date values of the Quality-Protocol field are specified in the most recent "Assigned Numbers" RFC [11]. Current values are assigned as follows:

Value (in hex)	Protocol
c025	Link Quality Report

#### Data

The Data field is zero or more octets and contains additional data as determined by the particular protocol.

#### Default

None

## 7.6. Magic-Number

### Description

This Configuration Option provides a way to detect looped-back links and other Data Link Layer anomalies. This Configuration Option MAY be required by some other Configuration Options such as the Monitoring-Protocol Configuration Option.

Before this Configuration Option is requested, an implementation must choose its Magic-Number. It is recommended that the Magic-Number be chosen in the most random manner possible in order to guarantee with very high probability that an implementation will arrive at a unique number. A good way to choose a unique random number is to start with an unique seed. Suggested sources of uniqueness include machine serial numbers, other network hardware addresses, time-of-day clocks, etc. Particularly good random number seeds are precise measurements of the inter-arrival time of physical events such as packet reception on other connected networks, server response time, or the typing rate of a human user. It is also suggested that as many sources as possible be used simultaneously.

When a Configure-Request is received with a Magic-Number Configuration Option, the received Magic-Number is compared with the Magic-Number of the last Configure-Request sent to the peer. If the two Magic-Numbers are different, then the link is not looped-back, and the Magic-Number should be acknowledged. If the two Magic-Numbers are equal, then it is possible, but not certain, that the link is looped-back and that this Configure-Request is actually the one last sent. To determine this, a Configure-Nak should be sent specifying a different Magic-Number value. A new Configure-Request should not be sent to the peer until normal processing would cause it to be sent (i.e., until a Configure-Nak is received or the Restart timer runs out).

Reception of a Configure-Nak with a Magic-Number different from that of the last Configure-Nak sent to the peer proves that a link is not looped-back, and indicates a unique Magic-Number. If the Magic-Number is equal to the one sent in the last Configure-Nak, the possibility of a looped-back link is increased, and a new Magic-Number should be chosen. In either case, a new Configure-Request should be sent with the new Magic-Number.

If the link is indeed looped-back, this sequence (transmit Configure-Request, receive Configure-Request, transmit Configure-Nak, receive Configure-Nak) will repeat over and over again. If the link is not looped-back, this sequence might occur a few

times, but it is extremely unlikely to occur repeatedly. More likely, the Magic-Numbers chosen at either end will quickly diverge, terminating the sequence. The following table shows the probability of collisions assuming that both ends of the link select Magic-Numbers with a perfectly uniform distribution:

Number of Collisions	Probability
-----	-----
1	$1/2^{**32} = 2.3 \text{ E-10}$
2	$1/2^{**32**2} = 5.4 \text{ E-20}$
3	$1/2^{**32**3} = 1.3 \text{ E-29}$

Good sources of uniqueness or randomness are required for this divergence to occur. If a good source of uniqueness cannot be found, it is recommended that this Configuration Option not be enabled; Configure-Requests with the option SHOULD NOT be transmitted and any Magic-Number Configuration Options which the peer sends SHOULD be either acknowledged or rejected. In this case, loop-backs cannot be reliably detected by the implementation, although they may still be detectable by the peer.

If an implementation does transmit a Configure-Request with a Magic-Number Configuration Option, then it MUST NOT respond with a Configure-Reject if its peer also transmits a Configure-Request with a Magic-Number Configuration Option. That is, if an implementation desires to use Magic Numbers, then it MUST also allow its peer to do so. If an implementation does receive a Configure-Reject in response to a Configure-Request, it can only mean that the link is not looped-back, and that its peer will not be using Magic-Numbers. In this case, an implementation should act as if the negotiation had been successful (as if it had instead received a Configure-Ack).

The Magic-Number also may be used to detect looped-back links during normal operation as well as during Configuration Option negotiation. All LCP Echo-Request, Echo-Reply, and Discard-Request packets have a Magic-Number field which MUST normally be zero, and MUST normally be ignored on reception. If Magic-Number has been successfully negotiated, an implementation MUST transmit these packets with the Magic-Number field set to its negotiated Magic-Number.

The Magic-Number field of these packets SHOULD be inspected on reception. All received Magic-Number fields MUST be equal to either zero or the peer's unique Magic-Number, depending on whether or not the peer negotiated one.

Reception of a Magic-Number field equal to the negotiated local

Magic-Number indicates a looped-back link. Reception of a Magic-Number other than the negotiated local Magic-Number or the peer's negotiated Magic-Number, or zero if the peer didn't negotiate one, indicates a link which has been (mis)configured for communications with a different peer.

Procedures for recovery from either case are unspecified and may vary from implementation to implementation. A somewhat pessimistic procedure is to assume a LCP Down event. A further Open event will begin the process of re-establishing the link, which can't complete until the loop-back condition is terminated and Magic-Numbers are successfully negotiated. A more optimistic procedure (in the case of a loop-back) is to begin transmitting LCP Echo-Request packets until an appropriate Echo-Reply is received, indicating a termination of the loop-back condition.

A summary of the Magic-Number 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      |      Magic-Number      |
+-----+-----+-----+-----+-----+-----+-----+-----+
|      Magic-Number (cont)      |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Type

5

Length

6

Magic-Number

The Magic-Number field is four octets and indicates a number which is very likely to be unique to one end of the link. A Magic-Number of zero is illegal and MUST always be Nak'd, if it is not Rejected outright.

Default

None.



## 7.7. Protocol-Field-Compression

### Description

This Configuration Option provides a way to negotiate the compression of the Data Link Layer Protocol field. By default, all implementations MUST transmit standard PPP frames with two octet Protocol fields. However, PPP Protocol field numbers are chosen such that some values may be compressed into a single octet form which is clearly distinguishable from the two octet form. This Configuration Option is sent to inform the peer that the implementation can receive such single octet Protocol fields. Compressed Protocol fields MUST NOT be transmitted unless this Configuration Option has been negotiated.

As previously mentioned, the Protocol field uses an extension mechanism consistent with the ISO 3309 extension mechanism for the Address field; the Least Significant Bit (LSB) of each octet is used to indicate extension of the Protocol field. A binary "0" as the LSB indicates that the Protocol field continues with the following octet. The presence of a binary "1" as the LSB marks the last octet of the Protocol field. Notice that any number of "0" octets may be prepended to the field, and will still indicate the same value (consider the two representations for 3, 00000011 and 00000000 00000011).

In the interest of simplicity, the standard PPP frame uses this fact and always sends Protocol fields with a two octet representation. Protocol field values less than 256 (decimal) are prepended with a single zero octet even though transmission of this, the zero and most significant octet, is unnecessary.

However, when using low speed links, it is desirable to conserve bandwidth by sending as little redundant data as possible. The Protocol Compression Configuration Option allows a trade-off between implementation simplicity and bandwidth efficiency. If successfully negotiated, the ISO 3309 extension mechanism may be used to compress the Protocol field to one octet instead of two. The large majority of frames are compressible since data protocols are typically assigned with Protocol field values less than 256.

In addition, PPP implementations must continue to be robust and MUST accept PPP frames with either double-octet or single-octet Protocol fields, and MUST NOT distinguish between them.

The Protocol field is never compressed when sending any LCP packet. This rule guarantees unambiguous recognition of LCP packets.

When a Protocol field is compressed, the Data Link Layer FCS field is calculated on the compressed frame, not the original uncompressed frame.

A summary of the Protocol-Field-Compression 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
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           Type           |      Length      |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

Type

7

Length

2

Default

Disabled.

## 7.8. Address-and-Control-Field-Compression

### Description

This Configuration Option provides a way to negotiate the compression of the Data Link Layer Address and Control fields. By default, all implementations MUST transmit frames with Address and Control fields and MUST use the hexadecimal values 0xff and 0x03 respectively. Since these fields have constant values, they are easily compressed. This Configuration Option is sent to inform the peer that the implementation can receive compressed Address and Control fields.

Compressed Address and Control fields are formed by simply omitting them. By definition the first octet of a two octet Protocol field will never be 0xff, and the Protocol field value 0x00ff is not allowed (reserved) to avoid ambiguity.

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.

If a compressed frame is received when Address-and-Control-Field-Compression has not been negotiated, the implementation MAY silently discard the frame.

The Address and Control fields MUST NOT be compressed when sending any LCP packet. This rule guarantees unambiguous recognition of LCP packets.

When the Address and Control fields are compressed, the Data Link Layer FCS field is calculated on the compressed frame, not the original uncompressed frame.

A summary of the Address-and-Control-Field-Compression 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
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|      Type      |      Length      |
+---+---+---+---+---+---+---+---+---+---+---+---+

```

Type

8

Length

2

Default

Not compressed.

## A. Asynchronous HDLC

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

### Transmission Considerations

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.

### 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 which is flagged in the Remote Async-Control-Character-Map is replaced by a two octet 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. Each octet with value less than hexadecimal 0x20 is checked. If it is flagged in the Local Async-Control-Character-Map, it is simply removed (it may have been inserted by intervening data communications equipment). For each Control Escape octet, that octet is also removed, but 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.

The transmitter may also send octets with value 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.

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.

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.  
0x13 is encoded as 0x7d, 0x33.  
0x91 is encoded as 0x7d, 0xb1.  
0x93 is encoded as 0x7d, 0xb3.

#### 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.

#### Time Fill

On asynchronous links, inter-octet and inter-frame time fill MUST 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. This implementation is based on [7], [8], and [9]. The table is created by the code in section B.2.

```

/*
 * ul6 represents an unsigned 16-bit number. Adjust the typedef for
 * your hardware.
 */
typedef unsigned short ul6;

/*
 * FCS lookup table as calculated by the table generator in section
 * B.2.
 */
static ul6 fcstab[256] = {
    0x0000, 0x1189, 0x2312, 0x329b, 0x4624, 0x57ad, 0x6536, 0x74bf,
    0x8c48, 0x9dc1, 0xaf5a, 0xbcd3, 0xca6c, 0xdb5e, 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,
    0xbdc3, 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,
    0xdec3, 0xcf44, 0xfddf, 0xec56, 0x98e9, 0x8960, 0xbbfb, 0xaa72,
    0x6306, 0x728f, 0x4014, 0x519d, 0x2522, 0x34ab, 0x0630, 0x17b9,
    0xef4e, 0xfec7, 0xcc5c, 0xdd5d, 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.
 */
ul6 pppfcs(fcs, cp, len)
    register ul6 fcs;
    register unsigned char *cp;
    register int len;
{
    ASSERT(sizeof (ul6) == 2);
    ASSERT(((ul6) -1) > 0);
    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("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("0x%04x", v & 0xFFFF);
        if (++b == 256)
            break;
        printf(",");
    }
    printf("\n};\n");
}
```

### C. LCP Recommended Options

The following Configurations Options are recommended:

#### SYNC LINES

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

#### ASync LINES

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

## Security Considerations

Security issues are briefly discussed in sections concerning the Authentication Phase, and the Authentication-Protocol Configuration Option. Further discussion is planned in a separate document entitled PPP Authentication Protocols.

## References

- [1] Electronic Industries Association, EIA Standard RS-232-C, "Interface Between Data Terminal Equipment and Data Communications Equipment Employing Serial Binary Data Interchange", August 1969.
- [2] International Organization For Standardization, ISO Standard 3309-1979, "Data communication - High-level data link control procedures - Frame structure", 1979.
- [3] International Organization For Standardization, ISO Standard 4335-1979, "Data communication - High-level data link control procedures - Elements of procedures", 1979.
- [4] International Organization For Standardization, ISO Standard 4335-1979/Addendum 1, "Data communication - High-level data link control procedures - Elements of procedures - Addendum 1", 1979.
- [5] International Organization For Standardization, Proposed Draft International Standard ISO 3309:1983/PDAD1, "Information processing systems - Data communication - High-level data link control procedures - Frame structure - Addendum 1: Start/stop transmission", 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.
- [8] Morse, G., "Calculating CRC's by Bits and Bytes", Byte, September 1986.
- [9] LeVan, J., "A Fast CRC", Byte, November 1987.
- [10] American National Standards Institute, ANSI X3.4-1977, "American National Standard Code for Information Interchange",

1977.

- [11] Reynolds, J., and J. Postel, "Assigned Numbers", RFC 1060, USC/Information Sciences Institute, March 1990.

#### Acknowledgments

Much of the text in this document is taken from the WG Requirements (unpublished), and RFCs 1171 & 1172, by Drew Perkins of Carnegie Mellon University, and by Russ Hobby of the University of California at Davis.

Many people spent significant time helping to develop the Point-to-Point Protocol. The complete list of people is too numerous to list, but the following people deserve special thanks: Rick Adams (UUNET), Ken Adelman (TGV), Fred Baker (ACC), Mike Ballard (Telebit), Craig Fox (NSC), Karl Fox (Morning Star Technologies), Phill Gross (NRI), former WG chair Russ Hobby (UC Davis), David Kaufman (Proteon), former WG chair Steve Knowles (FTP Software), John LoVerso (Xylogics), Bill Melohn (Sun Microsystems), Mike Patton (MIT), former WG chair Drew Perkins (CMU), Greg Satz (cisco systems) and Asher Waldfogel (Wellfleet).

#### Chair's Address

The working group can be contacted via the current chair:

Brian Lloyd  
Lloyd & Associates  
3420 Sudbury Road  
Cameron Park, California 95682

Phone: (916) 676-1147

EMail: [brian@ray.lloyd.com](mailto:brian@ray.lloyd.com)

#### Author's Address

Questions about this memo can also be directed to:

William Allen Simpson  
Daydreamer  
Computer Systems Consulting Services  
P O Box 6205  
East Lansing, MI 48826-6025

EMail: [bsimpson@ray.lloyd.com](mailto:bsimpson@ray.lloyd.com)

