Network Working Group
Request for Comments: 1661
STD: 51
Obsoletes: 1548
Category: Standards Track

W. Simpson, Editor
Daydreamer
July 1994

The Point-to-Point Protocol (PPP)

Status of this Memo

Abstract

The Point-to-Point Protocol (PPP) provides a standard method for transporting multi-protocol datagrams over point-to-point links. PPP is comprised of three main components:

1. A method for encapsulating multi-protocol datagrams.

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 organization and methodology, and the PPP encapsulation, together with an extensible option negotiation mechanism which is able to negotiate a rich assortment of configuration parameters and provides additional management functions. The PPP Link Control Protocol (LCP) is described in terms of this mechanism.

Table of Contents

1. Introduction

The Point-to-Point Protocol is designed for simple links which transport packets between two peers. These links provide full-duplex simultaneous bi-directional operation, and are assumed to deliver packets in order. It is intended that PPP provide a common solution for easy connection of a wide variety of hosts, bridges and routers [1].

Encapsulation

The PPP encapsulation provides for multiplexing of different network-layer protocols simultaneously over the same link. The PPP encapsulation has been carefully designed to retain compatibility with most commonly used supporting hardware.

Only 8 additional octets are necessary to form the encapsulation when used within the default HDLC-like framing. In environments where bandwidth is at a premium, the encapsulation and framing may be shortened to 2 or 4 octets.

To support high speed implementations, the default encapsulation uses only simple fields, only one of which needs to be examined for demultiplexing. The default header and information fields fall on 32-bit boundaries, and the trailer may be padded to an arbitrary boundary.

Link Control Protocol

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, detect a looped-back link and other common misconfiguration errors, and terminate the link. Other optional facilities provided are authentication of the identity of its peer on the link, and determination when a link is functioning properly and when it is failing.

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 companion documents.

Configuration

It is intended that PPP links be easy to configure. By design, the standard defaults handle all common configurations. The implementor can 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 are designed to be used by other control protocols, especially 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 must be understood and carefully weighed before choosing a different course.

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

1.2. Terminology

This document frequently uses the following terms:

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

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

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

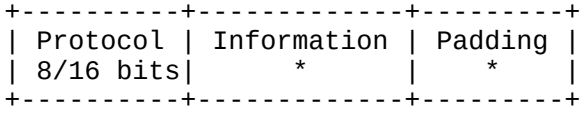
peer The other end of the point-to-point link.

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

2. PPP Encapsulation

The PPP encapsulation is used to disambiguate multiprotocol datagrams. This encapsulation requires framing to indicate the beginning and end of the encapsulation. Methods of providing framing are specified in companion documents.

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



Protocol Field

The Protocol field is one or two octets, and its value identifies the datagram encapsulated in the Information field of the packet. The field is transmitted and received most significant octet first.

The structure of this 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 treated as having an unrecognized Protocol.

Protocol field values in the "0***" to "3***" range identify the network-layer protocol of specific packets, and values in the "8***" to "b***" range identify packets 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 packets as link-layer Control Protocols (such as LCP).

Up-to-date values of the Protocol field are specified in the most recent "Assigned Numbers" RFC [2]. This specification reserves the following values:

Value (in hex)	Protocol Name
0001	Padding Protocol
0003 to 001f	reserved (transparency inefficient)
007d	reserved (Control Escape)
00cf	reserved (PPP NLPID)
00ff	reserved (compression inefficient)
8001 to 801f	unused
807d	unused
80cf	unused
80ff	unused
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 maximum length for the Information field, including Padding, but not including the Protocol field, is termed the Maximum Receive Unit (MRU), which defaults to 1500 octets. By negotiation, consenting PPP implementations may use other values for the MRU.

Padding

On transmission, the Information field MAY be padded with an arbitrary number of octets up to the MRU. It is the responsibility of each protocol to distinguish padding octets from real information.

3. PPP Link Operation

3.1. Overview

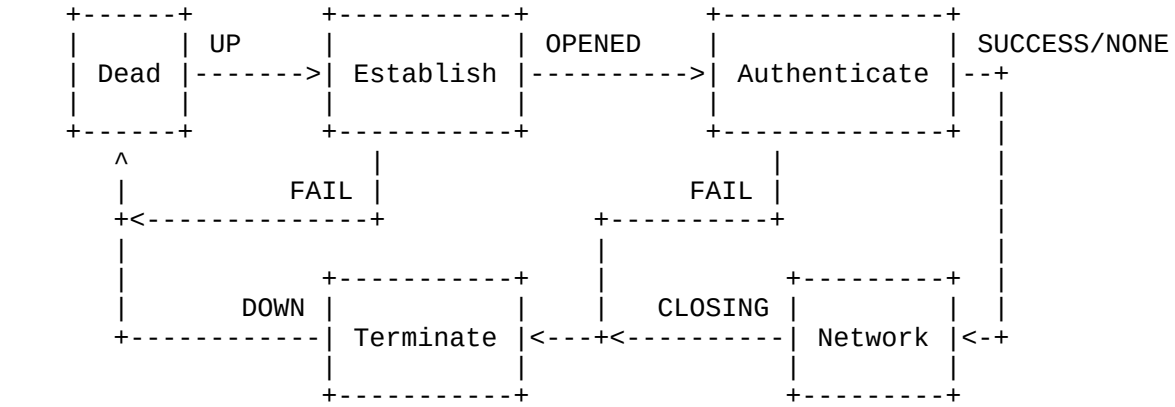
In order to establish communications over a point-to-point link, each end of the PPP link MUST first send LCP packets to configure and test the data link. After the link has been established, the peer MAY be authenticated.

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).

3.2. Phase Diagram

In the process of configuring, maintaining and terminating the point-to-point link, the PPP link goes through several distinct phases which are specified in the following simplified state diagram:



Not all transitions are specified in this diagram. The following semantics MUST be followed.

3.3. 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 later) will be in the Initial or Starting states. The transition to the Link Establishment phase will signal an Up event to the LCP 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 link, this phase may be extremely short -- merely long enough to detect the presence of the device.

3.4. 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 later) has been both sent and received.

All Configuration Options are assumed to be at default values unless altered by the configuration exchange. See the chapter 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.

Any non-LCP packets received during this phase MUST be silently discarded.

The receipt of the LCP Configure-Request causes a return to the Link Establishment phase from the Network-Layer Protocol phase or Authentication phase.

3.5. 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 mandatory. If an implementation desires that the peer authenticate with some specific authentication protocol, then it MUST request 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 authentication has completed. If authentication fails, the authenticator SHOULD proceed instead to the Link Termination phase.

Only Link Control Protocol, authentication protocol, and link quality monitoring packets are allowed during this phase. All other packets received during this phase MUST be silently discarded.

Implementation Notes:

An implementation SHOULD NOT fail authentication simply due to timeout or lack of response. The authentication SHOULD allow some method of retransmission, and proceed to the Link Termination phase only after a number of authentication attempts has been exceeded.

The implementation responsible for commencing Link Termination phase is the implementation which has refused authentication to its peer.

3.6. Network-Layer Protocol Phase

Once PPP has finished the previous phases, each network-layer protocol (such as IP, IPX, or AppleTalk) 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 supported network-layer protocol packets received when the corresponding NCP is not in the Opened state MUST be silently discarded.

Implementation Note:

> While LCP is in the Opened state, any protocol packet which is unsupported by the implementation MUST be returned in a Protocol-Reject (described later). Only protocols which are supported are silently discarded.

During this phase, link traffic consists of any possible combination of LCP, NCP, and network-layer protocol packets.

3.7. Link Termination Phase

PPP can terminate the link at any time. This might happen because of the loss of carrier, authentication failure, link quality failure, the expiration of an idle-period timer, or the administrative closing of the link.

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.

Any non-LCP packets received during this phase MUST be silently discarded.

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 one NCP has Closed is not sufficient reason to cause the termination of the PPP link, even if that NCP was the only NCP currently in the Opened state.

4. 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-Started
Close= administrative Close	tlf = This-Layer-Finished
TO+ = Timeout with counter > 0	irc = Initialize-Restart-Count
TO- = Timeout with counter expired	zrc = Zero-Restart-Count
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

4.1. 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; multiple actions may be implemented in any convenient order. The state may be followed by a letter, which indicates an explanatory footnote. The dash ('-') indicates an illegal transition.

	State					
	0	1	2	3	4	5
Events	Initial	Starting	Closed	Stopped	Closing	Stopping
Up	2	irc,scr/6	-	-	-	-
Down	-	-	0	tls/1	0	1
Open	tls/1	1	irc,scr/6	3r	5r	5r
Close	0	tlf/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

	State			
	6	7	8	9
Events	Req-Sent	Ack-Rcvd	Ack-Sent	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	scj/9
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-Count actions start or re-start the Restart timer. The Restart timer is stopped when transitioning from any state where the timer is running to a state where the timer is not running.

The events and actions are defined according to a message passing architecture, rather than a signalling architecture. If an action is desired to control specific signals (such as DTR), additional actions are likely to be required.

[p] Passive option; see Stopped state discussion.

[r] Restart option; see Open event discussion.

[x] Crossed connection; see RCA event discussion.

4.2. 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, the Closed state is 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 running, since a Configure-Ack has not yet been received.

Opened

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

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.

4.3. Events

Transitions and actions in the automaton are caused by events.

Up

This event occurs when a lower layer indicates that it is ready to carry packets.

Typically, this event is used by a modem handling or calling process, or by some other coupling of the PPP link to the physical media, to signal LCP that the link is entering Link Establishment phase.

It also can be used by LCP to signal each NCP that the link is entering Network-Layer Protocol phase. That is, the This-Layer-Up action from LCP triggers the Up event in the NCP.

Down

This event occurs when a lower layer indicates that it is no

longer ready to carry packets.

Typically, this event is used by a modem handling or calling process, or by some other coupling of the PPP link to the physical media, to signal LCP that the link is entering Link Dead phase.

It also can be used by LCP to signal each NCP that the link is leaving Network-Layer Protocol phase. That is, the This-Layer-Down action from LCP triggers the Down event in the NCP.

Open

This 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 is necessary.

Implementation Option:

Experience has shown that users will execute an additional Open command when they want to renegotiate the link. This might indicate that new values are to be negotiated.

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. Care must be taken that an intervening Down event cannot occur from another source.

The Down followed by an Up will cause an orderly renegotiation of the link, by progressing through the Starting to the Request-Sent state. This will cause the renegotiation of the link, without any harmful side effects.

Close

This 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. Futher attempts to re-configure the link are denied until a new Open event occurs.

Implementation Note:

When authentication fails, the link SHOULD be terminated, to prevent attack by repetition and denial of service to other users. Since the link is administratively available (by definition), this can be accomplished by simulating a Close event to the LCP, immediately followed by an Open event. Care must be taken that an intervening Close event cannot occur from another source.

The Close followed by an Open will cause an orderly termination of the link, by progressing through the Closing to the Stopping state, and the This-Layer-Finished action can disconnect the link. The automaton waits in the Stopped or Starting states for the next connection attempt.

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)

This 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 occurance 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)

This 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)

This 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)

This 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 an Echo-Request packet. There is no reply to an Echo-Reply or Discard-Request packet.

4.4. 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 cannot occur in a properly implemented automaton. The implementation has an internal error, which should be reported and logged. No transition is taken, and the implementation SHOULD NOT reset or freeze.

This-Layer-Up (tlu)

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

Typically, this action is 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 network layer traffic.

This-Layer-Down (tld)

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

Typically, this action is 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 network layer traffic.

This-Layer-Started (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 results of this action are highly implementation dependent.

This-Layer-Finished (tlf)

This action indicates to the lower layers that the automaton is entering the Initial, Closed or Stopped 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 results of this action are highly implementation dependent.

Initialize-Restart-Count (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.

Implementation Note:

In addition to setting the Restart counter, the implementation MUST set the timeout period to the initial value when Restart timer backoff is used.

Zero-Restart-Count (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, allowing traffic to be processed by the peer. In addition to zeroing the Restart counter, the implementation MUST set the timeout period to an appropriate value.

Send-Configure-Request (scr)

A Configure-Request packet is transmitted. 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)

A Configure-Ack packet is transmitted. This acknowledges the reception of a Configure-Request packet with an acceptable set of Configuration Options.

Send-Configure-Nak (scn)

A Configure-Nak or Configure-Reject packet is transmitted, 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 chapter on LCP Packet Formats.

Send-Terminate-Request (str)

A Terminate-Request packet is transmitted. 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)

A Terminate-Ack packet is transmitted. This acknowledges the reception of a Terminate-Request packet or otherwise serves to synchronize the automatons.

Send-Code-Reject (scj)

A Code-Reject packet is transmitted. This indicates the reception of an unknown type of packet.

Send-Echo-Reply (ser)

An Echo-Reply packet is transmitted. This acknowledges the reception of an Echo-Request packet.

4.5. 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.

4.6. 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 SHOULD 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 (2,400 to 9,600 bps), high switching latency links (typical telephone lines). Higher speed links, or links with low switching latency, SHOULD have correspondingly faster retransmission times.

Instead of a constant value, the Restart timer MAY begin at an initial small value and increase to the configured final value. Each successive value less than the final value SHOULD be at least twice the previous value. The initial value SHOULD be large enough to account for the size of the packets, twice the round trip time for transmission at the link speed, and at least an additional 100 milliseconds to allow the peer to process the packets before responding. Some circuits add another 200 milliseconds of satellite delay. Round trip times for modems operating at 14,400 bps have been measured in the range of 160 to more than 600 milliseconds.

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 for peer requested options are converted to Configure-Reject packets, and locally desired options are no longer appended. Max-Failure MUST be configurable, but SHOULD default to five (5) transmissions.

5. 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).

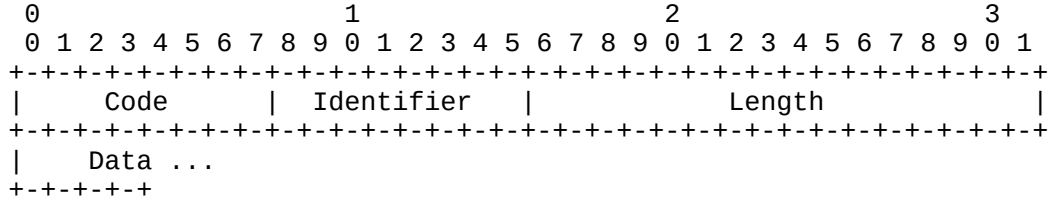
3. Link Maintenance packets used to manage and debug a link (Code-Reject, Protocol-Reject, Echo-Request, Echo-Reply, and Discard-Request).

In the interest of simplicity, there is no version field in the LCP packet. A correctly functioning LCP implementation will always respond to unknown Protocols and Codes with an easily recognizable LCP 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 as if no Configuration Options were negotiated. In particular, each Configuration Option specifies a default value. This ensures that such LCP packets are always recognizable, even when one end of the link mistakenly believes the link to be open.

Exactly one LCP packet is encapsulated in the PPP Information field, where the PPP 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.



Code

The Code field is one octet, and identifies the kind of LCP

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

Up-to-date values of the LCP Code field are specified in the most recent "Assigned Numbers" RFC [2]. This document concerns the following values:

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

Identifier

The Identifier field is one octet, and aids in matching requests and replies. When a packet is received with an invalid Identifier field, the packet is silently discarded without affecting the automaton.

Length

The Length field is two octets, and indicates the length of the LCP packet, including the Code, Identifier, Length and Data fields. The Length MUST NOT exceed the MRU of the link.

Octets outside the range of the Length field are treated as padding and are ignored on reception. When a packet is received with an invalid Length field, the packet is silently discarded without affecting the automaton.

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.

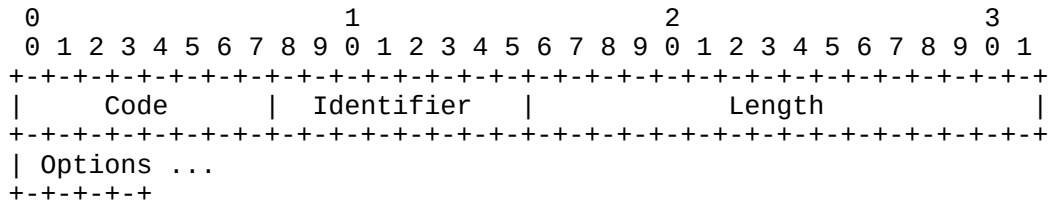
5.1. Configure-Request

Description

An implementation wishing to open a connection MUST transmit a Configure-Request. The Options field is filled with any desired changes to the link defaults. Configuration Options SHOULD NOT be included with default values.

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.



Code

1 for Configure-Request.

Identifier

The Identifier field MUST be changed whenever the contents of the Options field changes, and whenever a valid reply has been received for a previous request. For retransmissions, the Identifier MAY remain unchanged.

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 chapter.

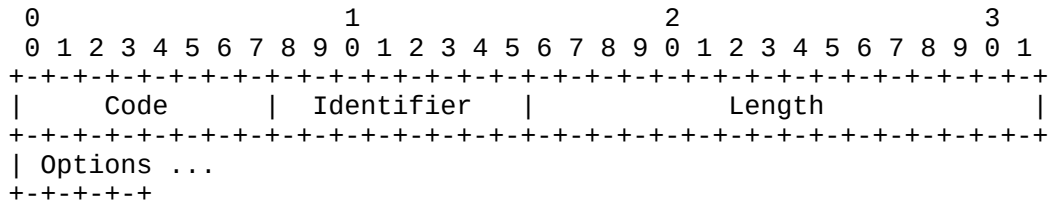
5.2. Configure-Ack

Description

If every Configuration Option received in a Configure-Request is recognizable and all values are acceptable, then the implementation MUST transmit a Configure-Ack. 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.



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.

5.3. Configure-Nak

Description

If every instance of the received Configuration Options is recognizable, but some values are not acceptable, then the implementation MUST transmit a Configure-Nak. The Options field is 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.

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

Each Configuration Option which is allowed only a single instance MUST be modified to a value acceptable to the Configure-Nak sender. The default value MAY be used, when this differs from the requested value.

When a particular type of Configuration Option can be listed more than once with different values, the Configure-Nak MUST include a list of all values for that option which are acceptable to the Configure-Nak sender. This includes acceptable values that were present in the Configure-Request.

Finally, an implementation may be configured to request the negotiation of a specific Configuration Option. If that option is not listed, then that option MAY be appended to the list of Nak'd Configuration Options, in order to prompt the peer to include that option in its next Configure-Request packet. Any value fields for the option MUST indicate values acceptable to the Configure-Nak sender.

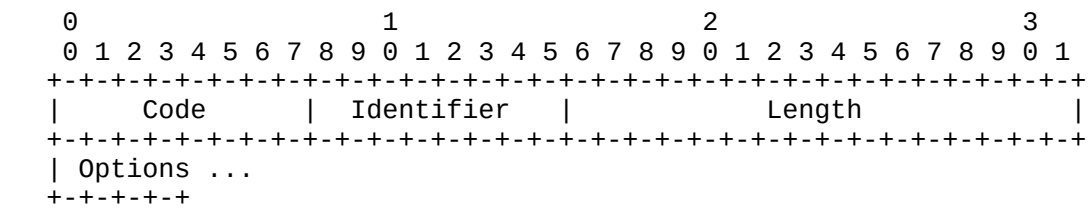
On reception of a Configure-Nak, the Identifier field MUST match that of the last transmitted Configure-Request. Invalid packets are silently discarded.

Reception of a valid Configure-Nak indicates that when a new Configure-Request is sent, the Configuration Options MAY be modified as specified in the Configure-Nak. When multiple instances of a Configuration Option are present, the peer SHOULD select a single value to include in its next Configure-Request packet.

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.



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.

5.4. Configure-Reject

Description

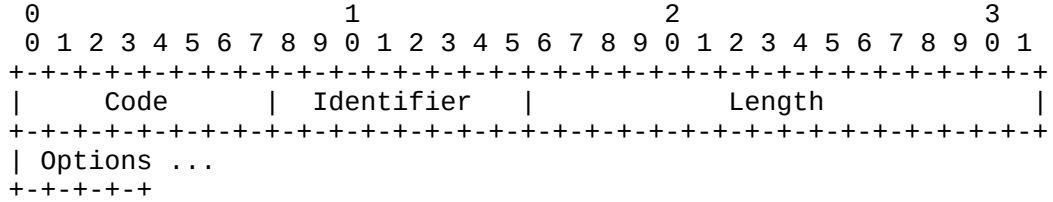
If some Configuration Options received in a Configure-Request are not recognizable or are not acceptable for negotiation (as configured by a network administrator), then the implementation MUST transmit a Configure-Reject. The Options field is filled with only the unacceptable Configuration Options from 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 when a new Configure-Request is sent, it MUST 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.



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.

5.5. Terminate-Request and Terminate-Ack

Description

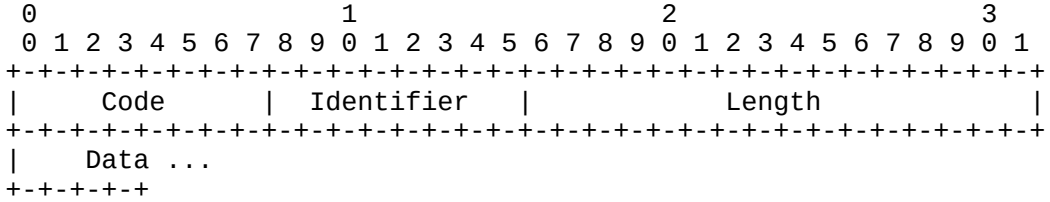
LCP includes Terminate-Request and Terminate-Ack Codes in order to provide a mechanism for closing a connection.

An implementation wishing to close a connection SHOULD transmit a Terminate-Request. 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 Terminate-Ack MUST be transmitted.

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.



Code

5 for Terminate-Request;

6 for Terminate-Ack.

Identifier

On transmission, the Identifier field MUST be changed whenever the content of the Data field changes, and whenever a valid reply has been received for a previous request. For retransmissions, the Identifier MAY remain unchanged.

On reception, the Identifier field of the Terminate-Request is copied into the Identifier field of the Terminate-Ack packet.

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. The end of the field is indicated by the Length.

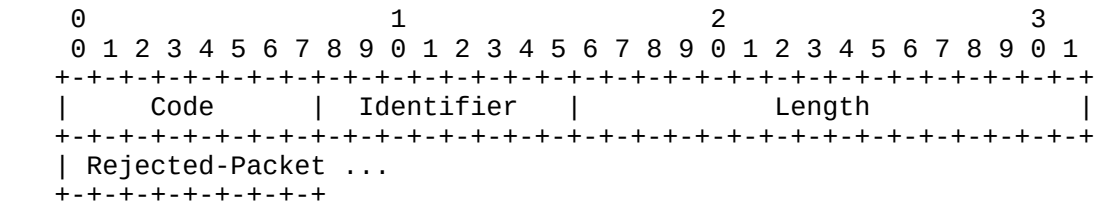
5.6. Code-Reject

Description

Reception of a LCP packet with an unknown Code indicates that the peer is operating with a different version. This MUST be reported back to the sender of the unknown Code by transmitting a Code-Reject.

Upon reception of the Code-Reject of a code which is fundamental to this version of the protocol, the implementation SHOULD report the problem and drop the connection, 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.



Code

7 for Code-Reject.

Identifier

The Identifier field MUST be changed for each Code-Reject sent.

Rejected-Packet

The Rejected-Packet field contains a copy of the LCP packet which is being rejected. It begins with the Information field, and does not include any Data Link Layer headers nor an FCS. The Rejected-Packet MUST be truncated to comply with the peer's

established MRU.

5.7. Protocol-Reject

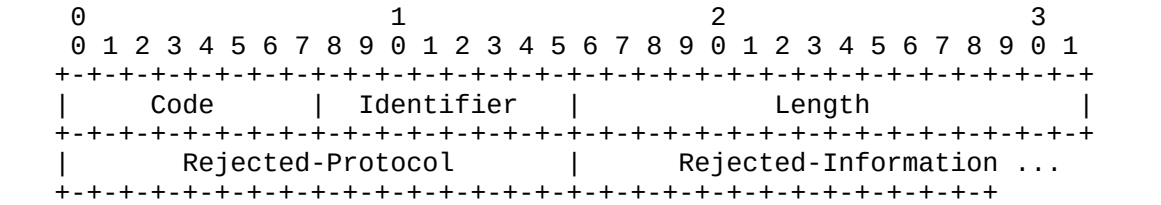
Description

Reception of a PPP packet with an unknown Protocol field 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 automaton is in the Opened state, then this MUST be reported back to the peer by transmitting a Protocol-Reject.

Upon reception of a Protocol-Reject, the implementation MUST stop sending packets of the indicated protocol at the earliest opportunity.

Protocol-Reject packets can 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.



Code

8 for Protocol-Reject.

Identifier

The Identifier field MUST be changed for each Protocol-Reject sent.

Rejected-Protocol

The Rejected-Protocol field is two octets, and contains the PPP Protocol field of the packet which is being rejected.

Rejected-Information

The Rejected-Information field contains a copy of the packet which is being rejected. It begins with the Information field, and does not include any Data Link Layer headers nor an FCS. The Rejected-Information MUST be truncated to comply with the peer's established MRU.

5.8. Echo-Request and Echo-Reply

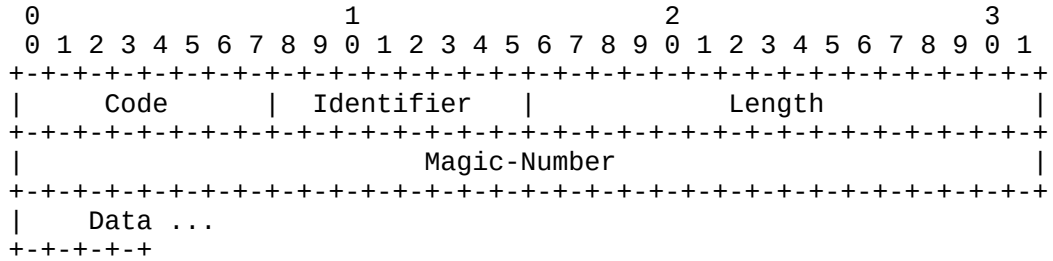
Description

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.

Upon reception of an Echo-Request in the LCP Opened state, an Echo-Reply MUST be transmitted.

Echo-Request and Echo-Reply packets MUST 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.



Code

9 for Echo-Request;

10 for Echo-Reply.

Identifier

On transmission, the Identifier field MUST be changed whenever the content of the Data field changes, and whenever a valid reply has been received for a previous request. For retransmissions, the Identifier MAY remain unchanged.

On reception, the Identifier field of the Echo-Request is copied into the Identifier field of the Echo-Reply packet.

Magic-Number

The Magic-Number field is four octets, and aids in detecting links which are in the looped-back condition. Until the Magic-Number Configuration Option has been successfully negotiated, the Magic-Number MUST be transmitted as zero. 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. The end of the field is indicated by the Length.

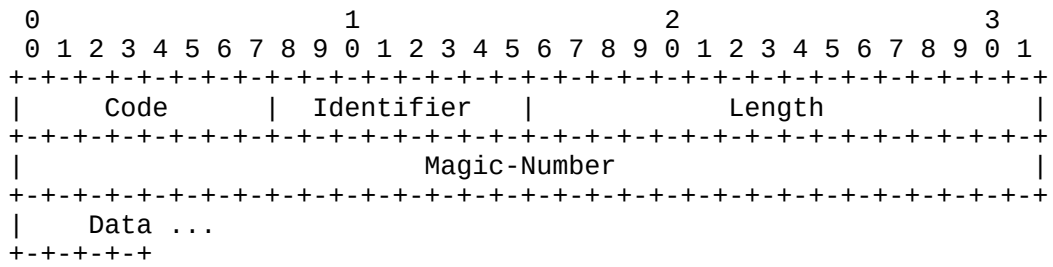
5.9. Discard-Request

Description

LCP includes a Discard-Request Code in order to provide a Data Link Layer 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.

Discard-Request packets MUST only be sent in the LCP Opened state. On reception, the receiver MUST silently discard any Discard-Request that it receives.

A summary of the Discard-Request packet format is shown below. The fields are transmitted from left to right.



Code

11 for Discard-Request.

Identifier

The Identifier field MUST be changed for each Discard-Request sent.

Magic-Number

The Magic-Number field is four octets, and aids in detecting links which are in the looped-back condition. Until the Magic-Number Configuration Option has been successfully negotiated, the Magic-Number MUST be transmitted as zero. 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. The end of the field is indicated by the Length.

6. LCP Configuration Options

LCP Configuration Options allow negotiation of modifications to the default characteristics of a point-to-point link. If a Configuration Option is not included in a Configure-Request packet, the default value for that Configuration Option is assumed.

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 description. (None of the Configuration Options in this specification can be listed more than once.)

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

Unless otherwise specified, all Configuration Options apply in a half-duplex fashion; typically, in the receive direction of the link from the point of view of the Configure-Request sender.

Design Philosophy

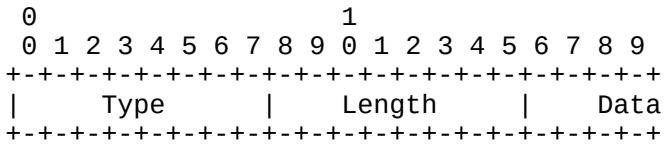
The options indicate additional capabilities or requirements of the implementation that is requesting the option. An implementation which does not understand any option SHOULD interoperate with one which implements every option.

A default is specified for each option which allows the link to correctly function without negotiation of the option, although perhaps with less than optimal performance.

Except where explicitly specified, acknowledgement of an option does not require the peer to take any additional action other than the default.

It is not necessary to send the default values for the options in a Configure-Request.

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



Type

The Type field is one octet, and indicates the type of Configuration Option. Up-to-date values of the LCP Option Type field are specified in the most recent "Assigned Numbers" RFC [2]. This document concerns the following values:

0	RESERVED
1	Maximum-Receive-Unit
3	Authentication-Protocol
4	Quality-Protocol
5	Magic-Number
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 or unrecognized 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 contains information specific to the Configuration Option. The format and length of the Data field is determined by the Type and Length fields.

When the Data field is indicated by the Length to extend beyond the end of the Information field, the entire packet is silently discarded without affecting the automaton.

6.1. Maximum-Receive-Unit (MRU)

Description

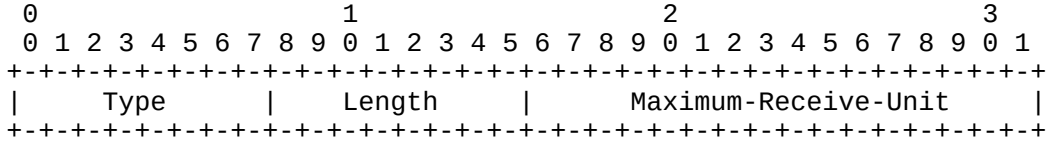
This Configuration Option may be sent to inform the peer that the implementation can receive larger packets, or to request that the peer send smaller packets.

The default value is 1500 octets. If smaller packets are requested, an implementation MUST still be able to receive the full 1500 octet information field in case link synchronization is lost.

Implementation Note:

This option is used to indicate an implementation capability. The peer is not required to maximize the use of the capacity. For example, when a MRU is indicated which is 2048 octets, the peer is not required to send any packet with 2048 octets. The peer need not Configure-Nak to indicate that it will only send smaller packets, since the implementation will always require support for at least 1500 octets.

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



Type

1

Length

4

Maximum-Receive-Unit

The Maximum-Receive-Unit field is two octets, and specifies the maximum number of octets in the Information and Padding fields. It does not include the framing, Protocol field, FCS, nor any transparency bits or bytes.

6.2. 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.

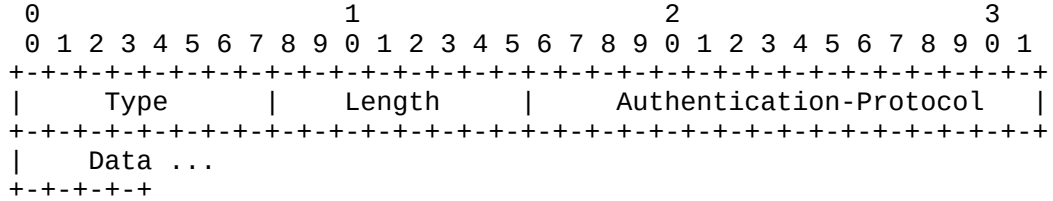
This Configuration Option provides a method to negotiate the use of a specific protocol for authentication. By default, authentication is not required.

An implementation MUST 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 Configure-Nak'd, then the implementation SHOULD attempt the next most desirable protocol in the next Configure-Request.

The implementation sending the Configure-Request is indicating that it expects authentication from its peer. If an implementation sends a Configure-Ack, then it is agreeing to authenticate with the specified protocol. An implementation receiving a Configure-Ack 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.



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 Protocol field values for that same authentication protocol.

Up-to-date values of the Authentication-Protocol field are specified in the most recent "Assigned Numbers" RFC [2]. 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.

6.3. 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.

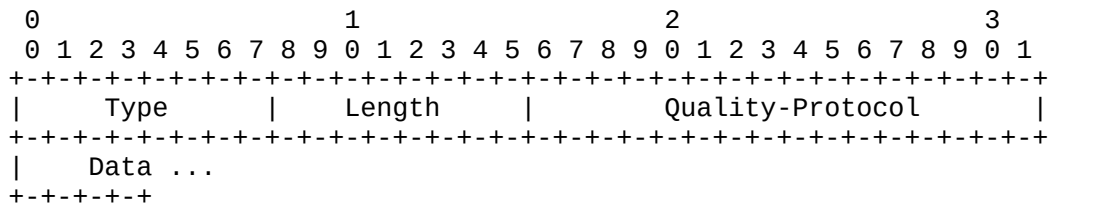
This Configuration Option provides a method to negotiate the use of a specific protocol for link quality monitoring. By default, link quality monitoring is disabled.

The implementation sending the Configure-Request is indicating that it expects to receive monitoring information from its peer. If an implementation sends a Configure-Ack, then it is agreeing to send the specified protocol. An implementation receiving a Configure-Ack SHOULD expect the peer to send the acknowledged protocol.

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.



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 Protocol field values for that same monitoring protocol.

Up-to-date values of the Quality-Protocol field are specified in the most recent "Assigned Numbers" RFC [2]. 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.

6.4. Magic-Number

Description

This Configuration Option provides a method 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 Quality-Protocol Configuration Option. By default, the Magic-Number is not negotiated, and zero is inserted where a Magic-Number might otherwise be used.

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 a 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 MUST 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 (that is, 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 MUST 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 E-10
2	1/2**32**2 = 5.4 E-20
3	1/2**32**3 = 1.3 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, looped-back links 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 when it receives 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).

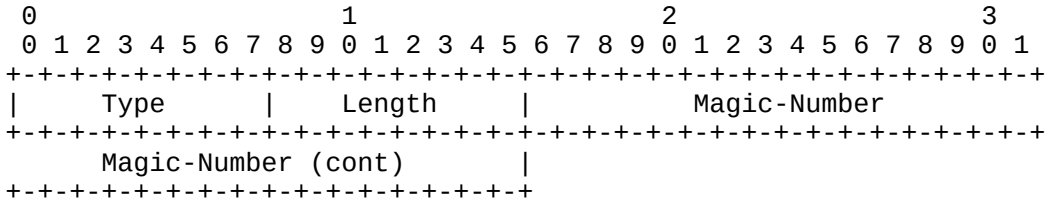
The Magic-Number also may be used to detect looped-back links during normal operation, as well as during Configuration Option negotiation. All LCP Echo-Request, Echo-Reply, and Discard-Request packets have a Magic-Number field. If Magic-Number has been successfully negotiated, an implementation MUST 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 a Magic-Number.

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, 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 looped-back condition is terminated, and Magic-Numbers are successfully negotiated. A more optimistic procedure (in the case of a looped-back link) is to begin transmitting LCP Echo-Request packets until an appropriate Echo-Reply is received, indicating a termination of the looped-back condition.

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



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.

6.5. Protocol-Field-Compression (PFC)

Description

This Configuration Option provides a method to negotiate the compression of the PPP Protocol field. By default, all implementations MUST transmit packets with two octet PPP Protocol fields.

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.

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 binary representations for 3, 00000011 and 00000000 00000011).

When using low speed links, it is desirable to conserve bandwidth by sending as little redundant data as possible. The Protocol-Field-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 packets are compressible since data protocols are typically assigned with Protocol field values less than 256.

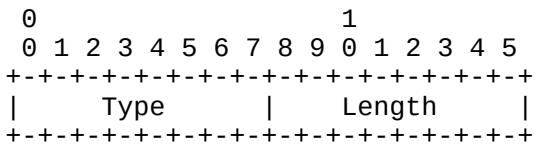
Compressed Protocol fields MUST NOT be transmitted unless this Configuration Option has been negotiated. When negotiated, PPP implementations MUST accept PPP packets with either double-octet or single-octet Protocol fields, and MUST NOT distinguish between them.

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.



Type

7

Length

2

6.6. Address-and-Control-Field-Compression (ACFC)

Description

This Configuration Option provides a method 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 appropriate to the link framing.

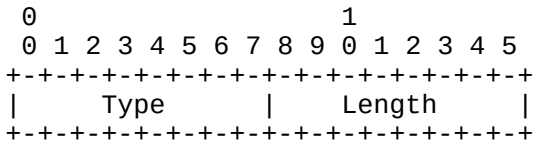
Since these fields usually have constant values for point-to-point links, they are easily compressed. This Configuration Option is sent to inform the peer that the implementation can receive compressed Address and Control fields.

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.



Type

8

Length

2

Security Considerations

Security issues are briefly discussed in sections concerning the Authentication Phase, the Close event, and the Authentication-Protocol Configuration Option.

References

[1] Perkins, D., "Requirements for an Internet Standard Point-to-Point Protocol", RFC 1547, Carnegie Mellon University, December 1993.

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

Acknowledgements

This document is the product of the Point-to-Point Protocol Working Group of the Internet Engineering Task Force (IETF). Comments should be submitted to the ietf-ppp@merit.edu mailing list.

Much of the text in this document is taken from the working group requirements [1]; and RFCs 1171 & 1172, by Drew Perkins while at Carnegie Mellon University, and by Russ Hobby of the University of California at Davis.

William Simpson was principally responsible for introducing consistent terminology and philosophy, and the re-design of the phase and negotiation state machines.

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, Ken Adelman, Fred Baker, Mike Ballard, Craig Fox, Karl Fox, Phill Gross, Kory Hamzeh, former WG chair Russ Hobby, David Kaufman, former WG chair Steve Knowles, Mark Lewis, former WG chair Brian Lloyd, John LoVerso, Bill Melohn, Mike Patton, former WG chair Drew Perkins, Greg Satz, John Shriver, Vernon Schryver, and Asher Waldfogel.

Special thanks to Morning Star Technologies for providing computing resources and network access support for writing this specification.

Chair's Address

The working group can be contacted via the current chair:

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

fbaker@acc.com

Editor's Address

Questions about this memo can also be directed to:

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

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

~~~~~~
~~~~~~

Network Working Group
Request for Comments: 1662
STD: 51
Obsoletes: 1549
Category: Standards Track

W. Simpson, Editor
Daydreamer

July 1994

PPP in HDLC-like Framing

Status of this Memo

Abstract

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

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

Table of Contents

1. Introduction

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

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

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

1.1. Specification of Requirements

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

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

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

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

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

1.2. Terminology

This document frequently uses the following terms:

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

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

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

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

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

2. Physical Layer Requirements

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

Interface Format

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

Transmission Rate

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

Control Signals

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

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

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

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

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

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

3. The Data Link Layer

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

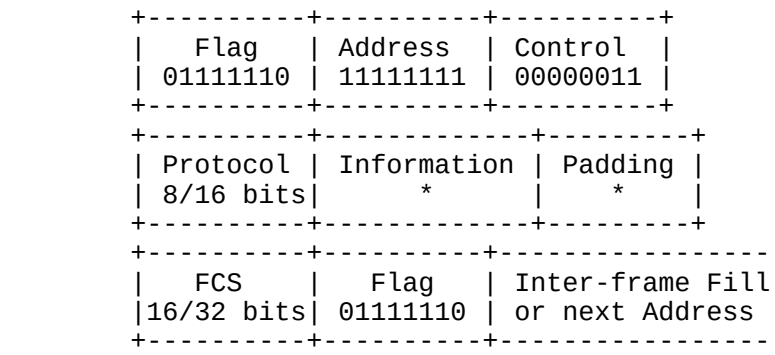
The PPP control procedures use the Control field encodings described in ISO 4335-1979 HDLC elements of procedures, most recently the fourth edition 4335:1991 [4].

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

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

3.1. Frame Format

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



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

Flag Sequence

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

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

Address Field

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

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

Control Field

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

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

Frame Check Sequence (FCS) Field

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

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

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

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

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

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

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

3.2. Modification of the Basic Frame

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

Address-and-Control-Field-Compression

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

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

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

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

4. Octet-stuffed framing

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

4.1. Flag Sequence

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

4.2. Transparency

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

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

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

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

Receiving implementations MUST correctly process all Control Escape sequences.

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

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

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

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

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

4.3. Invalid Frames

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

4.4. Time Fill

4.4.1. Octet-synchronous

There is no provision for inter-octet time fill.

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

4.4.2. Asynchronous

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

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

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

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

4.5. Transmission Considerations

4.5.1. Octet-synchronous

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

4.5.2. Asynchronous

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

5. Bit-stuffed framing

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

5.1. Flag Sequence

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

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

5.2. Transparency

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

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

5.3. Invalid Frames

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

5.4. Time Fill

There is no provision for inter-octet time fill.

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

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

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

5.5. Transmission Considerations

All octets are transmitted least significant bit first.

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

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

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

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

6. Asynchronous to Synchronous Conversion

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

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

7. Additional LCP Configuration Options

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

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

2 Async-Control-Character-Map

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

Description

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

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

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

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

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

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

Note that this specification differs in detail from later amendments, such as 3309:1991/Amendment 2 [3]. However, such "extended transparency" is applied only by "prior agreement". Use of the transparency methods in this specification constitute a prior agreement with respect to PPP.

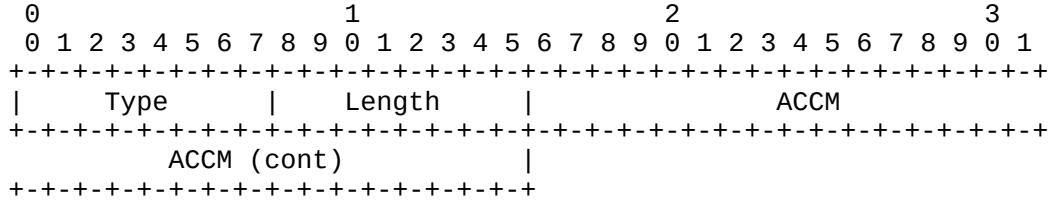
For compatibility with 3309:1991/Amendment 2, the transmitter MAY escape DEL and ACCM equivalents with the 8th (most significant) bit set. No change is required in the receiving algorithm.

Following ACCM negotiation, the transmitter SHOULD cease escaping DEL.

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

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

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



Type

2

Length

6

ACCM

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

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

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

A. Recommended LCP Options

The following Configurations Options are recommended:

High Speed links

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

Low Speed or Asynchronous links

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

B. Automatic Recognition of PPP Frames

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

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

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

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

C. Fast Frame Check Sequence (FCS) Implementation

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

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

C.1. FCS table generator

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

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

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


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

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

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

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

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

C.2. 16-bit FCS Computation Method

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

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

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

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

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

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

       return (fcs);
   }

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

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

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

C.3. 32-bit FCS Computation Method

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

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

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

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

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

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

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

    return (fcs);
    }

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

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

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

Security Considerations

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

References

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

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

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

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

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

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

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

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

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

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

Acknowledgements

This document is the product of the Point-to-Point Protocol Working Group of the Internet Engineering Task Force (IETF). Comments should be submitted to the ietf-ppp@merit.edu mailing list.

This specification is based on previous RFCs, where many contributions have been acknowleged.

The 32-bit FCS example code was provided by Karl Fox (Morning Star Technologies).

Special thanks to Morning Star Technologies for providing computing resources and network access support for writing this specification.

Chair's Address

The working group can be contacted via the current chair:

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

fbaker@acc.com

Editor's Address

Questions about this memo can also be directed to:

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

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