INTERNET-DRAFT
Expires: January 1998

Carsten Bormann
Universitaet Bremen
July 1997

PPP in a real-time oriented HDLC-like framing
draft-ietf-issll-isslow-rtf-01.txt

Status of this memo

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

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

To learn the current status of any Internet-Draft, please check the ``1id-abstracts.txt'' listing contained in the Internet-Drafts Shadow Directories on ftp.is.co.za (Africa), nic.nordu.net (Europe), munnari.oz.au (Pacific Rim), ds.internic.net (US East Coast), or ftp.isi.edu (US West Coast).

Distribution of this document is unlimited.

Abstract

A companion document describes an architecture for providing integrated services over low-bitrate links, such as modem lines, ISDN B-channels, and sub-T1 links [1]. The main components of the architecture are: a real-time encapsulation format for asynchronous and synchronous low-bitrate links, a header compression architecture optimized for real-time flows, elements of negotiation protocols used between routers (or between hosts and routers), and announcement protocols used by applications to allow this negotiation to take place.

This document proposes the suspend/resume-oriented solution for the real-time encapsulation format part of the architecture. The general approach is to start from the PPP Multilink fragmentation protocol [2] and its multi-class extension [5] and add suspend/resume in a way that is as compatible to existing hard- and firmware as possible.

This document is a product of the IETF ISSLL working group. Comments are solicited and should be addressed to the two working groups' mailing lists at issll@mercury.lcs.mit.edu and ietf-ppp@merit.edu and/or the author.

1. Introduction

As an extension to the ``best-effort'' services the Internet is well-known for, additional types of services (``integrated services'') that support the transport of real-time multimedia information are being developed for and deployed in the Internet.

A companion document describes an architecture for providing integrated services over low-bitrate links, such as modem lines, ISDN B-channels, and sub-T1 links [1]. The main components of the architecture are: a real-time encapsulation format for asynchronous and synchronous low-bitrate links, a header compression architecture optimized for real-time flows, elements of negotiation protocols used between routers (or between hosts and routers), and announcement protocols used by applications to allow this negotiation to take place.

The present document defines the suspend/resume-oriented solution for the real-time encapsulation format part of the architecture. As described in more detail in the architecture document, a real-time encapsulation format is required as, e.g., a 1500 byte packet on a 28.8 kbit/s modem link makes this link unavailable for the transmission of real-time information for about 400 ms. This adds a worst-case delay that causes real-time applications to operate with round-trip delays on the order of at least a second -- unacceptable for real-time conversation.

A true suspend/resume-oriented approach can only be implemented on a type-1 sender [1], but provides the best possible delay performance to this type of senders. The format defined in this document may also be of interest to certain type-2-senders that want to exploit the better bit-efficiency of this format as compared to [5]. It can be implemented by both type-1 and type-2 receivers.

2. Requirements

The requirements for this document are similar to those listed in [5].

A suspend/resume-oriented solution can provide better worst-case latency than the pre-fragmenting-oriented solution defined in [5]. Also, as this solution requires a new encapsulation scheme, there is an opportunity to provide a slightly more efficient format.

Predictability, robustness, and cooperation with PPP and existing hard- and firmware installations are as important with suspend/resume as with pre-fragmenting. A good suspend/resume solution achieves good performance even with type-2 receivers [1] and is able to work with PPP hardware such as async-to-sync converters.

Finally, a partial non-requirement: While the format defined in this draft is based on the PPP multilink protocol ([2], also abbreviated

as MP), operation over multiple links is in many cases not required.

3. General Approach

As in [5], the general approach is to start out from PPP multilink and add multiple classes to obtain multiple levels of suspension. However, in contrast to [5], more significant changes are required to be able to suspend the transmission of a packet at any point and inject a higher priority packet.

The applicability of the multilink header for suspend/resume type implementations is limited, as the ``end'' bit is in the multilink header, which is the wrong place for suspend/resume operation. To make a big packet suspendable, it must be sent with the ``end'' bit off, and (unless the packet was suspended a small number of bytes before its end) an empty fragment has to be sent afterwards to ``close'' the packet. The minimum overhead for sending a suspendable packet thus is twice the multilink header size (six bytes, including a compressed multilink protocol field) plus one PPP framing (three bytes). Each suspension costs another six bytes (not counting the overhead of the framing for the intervening packet).

Also, the existing multi-link header is relatively large; as the frequency of small high-priority packets increases, the overhead becomes significant.

The general approach of this document is to start from PPP Multilink with classes and provide a number of extensions to add functionality and reduce the overhead of using PPP Multilink for real-time transmission.

This document introduces two new features:

1) A compact fragment format and header, and

2) a real-time frame format.

4. The Compact Fragment Format

This section describes an optional multilink fragment format that is more optimized towards single-link operation and frequent suspension (type 1 senders)/a small fragment size (type 2 senders), with optional support for multiple links.

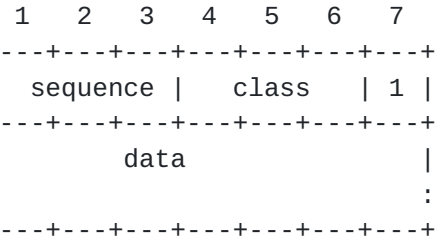
When operating over a single link, the Multilink sequence number is used only for loss detection. Even a 12-bit sequence number clearly is larger than required for this application on most kinds of links. We therefore define the following compact multilink header format

option with a three-bit sequence number.

As, with a compact header, there is little need for sending packets

outside the the multilink, we can provide an additional compression mechanism for this format: the MP protocol identifier is not sent with the compact fragment header. This obviously requires prior negotiation (similar to the way address and control field compression are negotiated), as well as a method for avoiding the bit combination 0xFF, as the start of a new LCP negotiation could otherwise not be reliably detected.

Figure 1: Compact Fragment Format



Having the least significant bit always be 1 helps with HDLC chips that operate specially on least significant bits in HDLC addresses. Headers with the least significant bit set to zero are reserved for future extensions.

The R bit is the inverted equivalent of the B bit in the other multilink fragment formats, i.e. R = 1 means that this fragment resumes a packet previous fragments of which have been sent already.

The following trick avoids the case of a header byte of 0xFF (which would mean R=1, sequence=7, and class=7): If the class field is set to 7, the R bit MUST never be set to one. I.e., class 7 frames by design cannot be suspended/resumed. (This is also the reason the sense of the B bit is inverted to an R bit in the compact fragment format -- class 7 would be useless otherwise, as a new packet could never be begun.)

As the sequence number is not particularly useful with the class field set to 7, it is used to distinguish eight more classes -- for some minor additional complexity, the applicability of prefix elision is significantly increased by providing more classes with possibly different elided prefixes.

For purposes of prefix elision, the actual class number of a fragment is computed as follows:

- If the class field is 0 to 6, the class number is 0 to 6,

- if the class field is 7 and the sequence field is 0 to 7, the class number is 7 to 14.

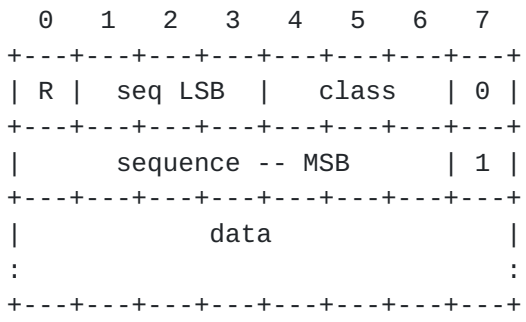
As a result of this scheme, the classes 0 to 6 can be used for suspendable packets, and classes 7 to 14 (where the class field is 7

and the R bit must always be off) can be used for non-suspendable high-priority classes, e.g., eight highly compressed voice streams.

5. The Extended Compact Fragment Format

For operation over multiple links, a three-bit sequence number will rarely be sufficient. Therefore, we define an optional extended compact fragment format. The option, when negotiated, allows both the basic compact fragment format and the extended compact fragment format to be used; each fragment indicates which format it is in.

Figure 1: Extended Compact Fragment Format



In the extended compact fragment format, the sequence number is composed of three least significant bits from the first octet of the fragment header and seven most significant bits from the second octet.

For prefix elision purposes, fragments with a class field of 7 can use the basic format to indicate classes 7 to 14 and the extended format to indicate classes 7 to 1030. Different classes may use different formats concurrently without problems. (This allows some classes to be spread over a multi-link and other classes to be confined to a single link with greater efficiency.) For class fields 0 to 6, i.e. suspendable classes, one of the two compact fragment formats SHOULD be used consistently within each class.

If the use of the extended compact fragment format has been negotiated, receivers MAY keep 10-bit sequence numbers for all classes to facilitate senders switching formats in a class. When a sender starts sending basic format fragments in a class that was using extended format fragments, the 3-bit sequence number can be taken as a modulo-8 version of the 10-bit sequence number, and no discontinuity need result. In the inverse case, if a 10-bit sequence number has been kept throughout and no major slips of the sequence number have occurred, no discontinuity will result, although this cannot be guaranteed in the presence of errors. (Discontinuity, in

this context, means that a receiver has to resynchronize sequence numbers by discarding fragments until a fragment with R=0 has been seen.)

6. Real-Time Frame Format

This section defines how fragments with compact fragment headers are mapped into real-time frames. This format has been designed to retain the overall HDLC based format of frames, so that existing synchronous HDLC chips and async to sync converters can be used on the link. Note that if the design could be optimized for async only operation, more design alternatives would be available [4]; with the advent of V.80 style modems, asynchronous communications is likely to decrease in importance, though.

The compact fragment format provides a compact rendition of the PPP multilink header with classes and a reduced sequence number space. However, it does not encode the E-bit of the PPP multilink header, which indicates whether the fragment at hand is the last fragment of a packet.

For a solution where packets can be suspended at any point in time, the E-bit needs to be encoded near the end of each fragment. The real-time frame format, to ensure maximum compatibility with type 2 receivers, encodes the E-bit in the following way: Any normal frame ending also ends the current fragment with E implicitly set to one. This ensures that packets that are ready for delivery to the upper layers immediately trigger a receive interrupt even at type-2 receivers.

Fragments of packets that are to be suspended are terminated within the HDLC frame by a special ``fragment suspend escape'' byte (FSE). The overall structure of the HDLC frame does not change; the detection and handling of FSE bytes is done at a layer above HDLC framing.

The suspend/resume format with FSE detection is an alternative to address/control field compression (ACFC, LCP option 8). It does not apply to frames that start with 0xFF, the standard PPP-in-HDLC address field; these frames are handled as defined in [6] and [7]. (This provision ensures that attempts to renegotiate LCP do not cause ambiguities.)

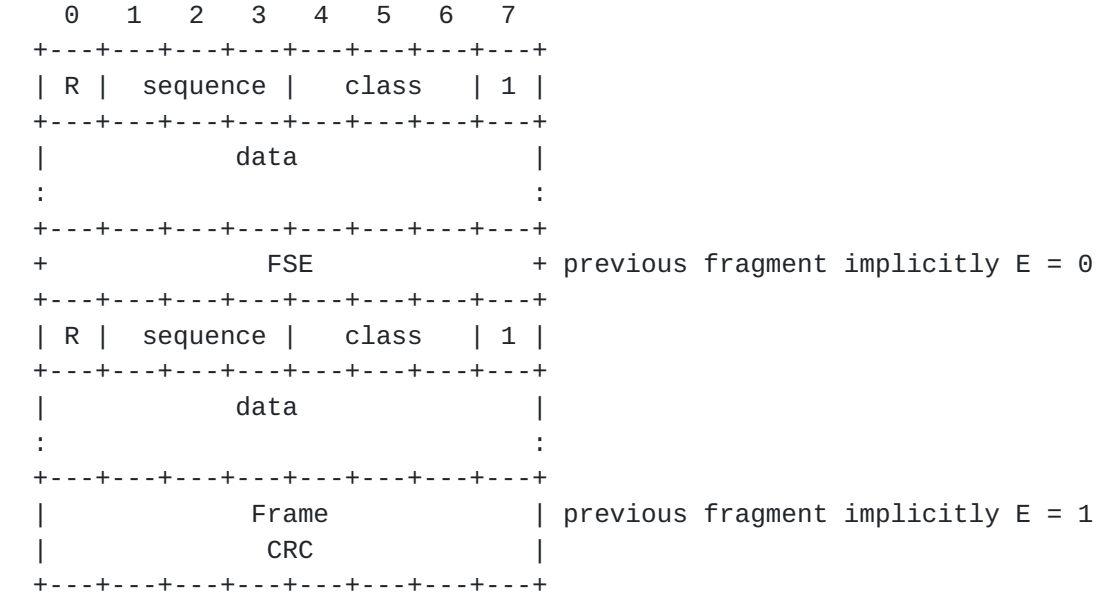
For frames that do not start with 0xFF, suspend/resume processing performs a scan of every HDLC frame received. The FCS of the HDLC frame is checked and stripped. Compact fragment format headers (both basic and extended) are handled without further FSE processing*. Within the remaining bytes of the HDLC frame, an FSE byte is used to indicate the end of the current fragment, with an E bit implicitly cleared. All fragments up to the last FSE are considered suspended; the final fragment is terminated, or, if it is empty, ignored (i.e., the data part of an HDLC frame can end in an FSE to indicate that the

* As the FSE byte was chosen such that it never occurs in compact fragment format headers, this does not require any specific code.

last fragment has E = 0).

Each fragment begins with a normal header, so the structure of a frame could be:

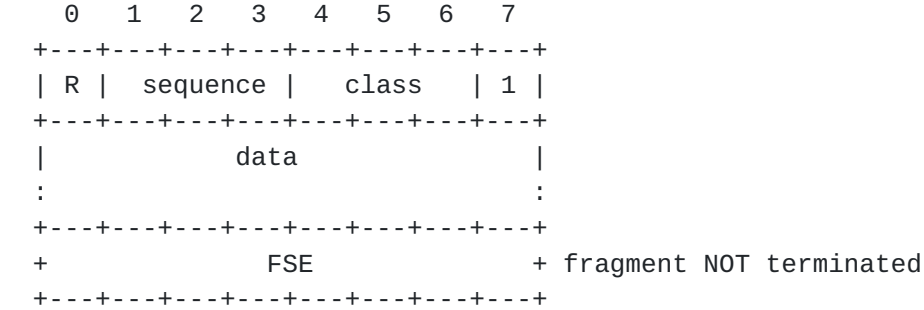
Figure 2: Example frame with FSE delimiter

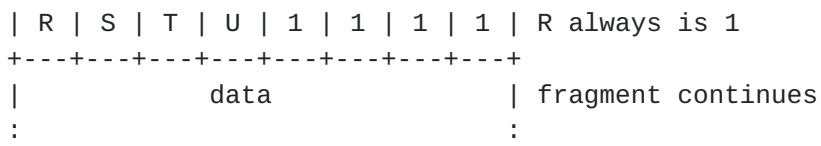


The value chosen for FSE is 0xDE (this is a relatively unlikely byte to occur in today's data streams, it does not trigger octet stuffing and triggers bit stuffing only for 1/8 of the possible preceding bytes).

The remaining problem is that of data transparency. In the scheme described so far, an FSE is always followed by a compact fragment header. In these headers, the combination of a class field set to 7 with R=1 is reserved. Data transparency is achieved by making the occurrence of an FSE byte followed by one of 0x8F, 0x9F to 0xFF special.

Figure 3: Data transparency with FSE bytes present




| R | S | T | U | 1 | 1 | 1 | 1 | R always is 1
data
| fragment continues

In a combination of FSE/0xnF (where n is the first four-bit field in the second byte, RSTU in Figure 3), the n field gives a sequence of four bits indicating where in the received data stream FSE bytes, which cannot simply be transmitted in the data stream, are to be added by the receiver:

0x8F: insert one FSE, back to data
0x9F: insert one FSE, copy two data bytes, insert one FSE, back to data
0xAF: insert one FSE, copy one data byte, insert one FSE, back to data
0xBF: insert one FSE, copy one data byte, insert two FSE bytes, back to data
0xCF: insert two FSE bytes, back to data
0xDF: insert two FSE bytes, copy one data byte, insert one FSE, back to data
0xEF: insert three FSE bytes, back to data
0xFF: insert four FSE bytes, back to data

The data bytes following the FSE/0xnF combinations and corresponding to the zero bits in the N field may not be FSE bytes.

This scheme limits the worst case expansion factor by FSE processing to about 25 %. Also, it is designed such that a single data stream can either trigger worst-case expansion by octet stuffing (or by bit stuffing) or worst-case FSE processing, but never both. Figure 4 illustrates the scheme in a few examples; FSE/0xnF pairs are written in lower case.

Figure 4: Data transparency examples

Data stream	FSE-stuffed stream
DD DE DF E0	DD de 8f DF E0
01 DE 02 DE 03	01 de af 02 03
DE DA DE DE DB	de bf DA DB

In summary, the real-time frame format is a HDLC-like frame delimited by flags and containing a final FCS as defined in [7], but without address and control fields, containing as data a sequence of FSE-stuffed fragments in compact fragment format, delimited by FSE bytes. As a special case, the final FSE may occur as the last byte of the data content (i.e. immediately before the FCS bytes) of the HDLC-like frame, to indicate that the last fragment in the frame is suspended and no final fragment is in the frame (e.g., because the desirable maximum size of the frame has been reached).

7. Implementation notes

7.1. MRU Issues

The LCP parameter MRU defines the maximum size of the packets sent on the link. Async-to-sync converters that are monitoring the LCP

negotiations on the link may interpret the MRU value as the maximum HDLC frame size to be expected.

Implementations of this specification should preferably negotiate a sufficiently large MRU to cover the worst-case 25 % increase in frame size plus the increase caused by suspended fragments. If that is not possible, the HDLC frame size should be limited by monitoring the HDLC frame sizes and possibly suspending the current fragment by sending an FSE with an empty final fragment (FSE immediately followed by the end of the information field, i.e. by CRC bytes and a flag) to be able to continue in a new HDLC frame. This strategy also helps minimizing the impact of lengthening the HDLC frame on the safety of the 16-bit FCS at the end of the HDLC frame.

7.2. Implementing octet-stuffing and FSE processing in one automaton

The simplest way to add real-time framing to an implementation may be to perform HDLC processing as usual and then, on the result, to perform FSE processing. A more advanced implementation may want to combine the two levels of escape character processing. Note, however, that FSE processing needs to wait until two bytes from the HDLC frame are available and followed by a third to ensure that the bytes are not the final HDLC FCS bytes, which are not subject to FSE processing. I.e., on the reception of normal data byte, look for an FSE in the second-to-previous byte, and, on the reception of a frame-end, look for an FSE as the last data byte.

8. Negotiable options

The following options are already defined by MP [2]:

- Multilink Maximum Received Reconstructed Unit
- Multilink Short Sequence Number Header Format
- Endpoint Discriminator

The following options is already defined by MCML [5]:

- Multilink Header Format
- Prefix Elision

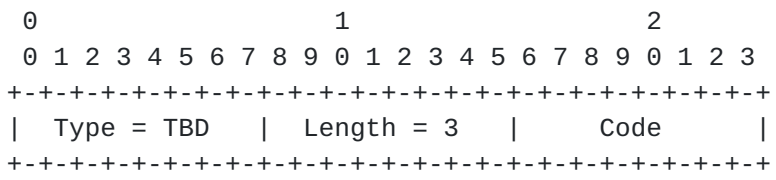
This document defines two new code points for the Multilink Header Format option.

8.1. Multilink header format option

The multilink header format option is defined in [5]. A summary of

the Multilink Header Format Option format is shown below. The fields are transmitted from left to right.

Figure 5: Multilink header format option



This option advises the peer that the implementation wishes to receive fragments with a format given by the code number. When this option is received, an implementation MUST either transmit all subsequent multilink packets on all links of the bundle with the multilink header format given or configure-NAK or configure-Reject the option. This specification defines one additional value for Code, in addition to those defined in [5]:

- This option present with code = 11: basic and extended compact real-time fragment format with classes, in FSE-encoded HDLC frame

- This option present with code = 15: basic compact real-time fragment format with classes, in FSE-encoded HDLC frame

An implementation MUST NOT request a combination of both the Short Sequence Number Header Format Option and this option, or of both LCP Address-and-Control-Field-Compression (ACFC) and the code values 11 or 15 for this option.

9. Acknowledgements

The participants in a lunch BOF at the Montreal IETF 1996 gave useful input on the design tradeoffs in various environments. Richard Andrades, Fred Burg, and Murali Aravamudan insisted that there should be a suspend/resume solution in addition to the pre-fragmenting one defined in [5]. The members of the ISSLL subgroup on low bitrate links (ISSLOW) have helped in coming up with a set of requirements that shaped this solution.

10. References

[1] C. Bormann, ``Providing integrated services over low-bitrate links'', Work in Progress (draft-ietf-issll-isslow-02.txt), May 1997.

[2] K. Sklower, B. Lloyd, G. McGregor, D. Carr, T. Coradetti, ``The PPP Multilink Protocol (MP)'', RFC 1990, August 1996 (obsoletes

RFC1717).

[3] W. Simpson, ``PPP in Frame Relay'', RFC 1973, June 1996.

[4] R. Andrades, F. Burg, ``QOSPPP Framing Extensions to PPP'', Work in Progress (draft-andrades-framing-ext-00.txt), September 1996.

[5] C. Bormann, ``The Multi-Class Extension to Multi-Link PPP'', Work in Progress (draft-ietf-issll-isslow-mcml-02.txt), March 1997.

[6] W. Simpson, Editor. ``The Point-to-Point Protocol (PPP)'', RFC 1661, July 1994 (Obsoletes RFC1548, also STD0051).

[7] W. Simpson, Editor. ``PPP in HDLC-like Framing'', RFC 1662, July 1994 (Obsoletes RFC1549, also STD0051)

11. Addresses

11.1. Working Group

The ISSLL working group can be contacted via the co-chairs, Eric Crawley <esc@baynetworks.com> and John Wroclawski <jtw@lcs.mit.edu>, or via its WG mailing list <issll@mercury.lcs.mit.edu>.

11.2. Author's address

Carsten Bormann
Universitaet Bremen FB3 TZI
Postfach 330440
D-28334 Bremen, GERMANY
cabo@tzi.uni-bremen.de
phone +49.421.218-7024
fax +49.421.218-7000