Network Working Group
INTERNET DRAFT
Document: draft-ietf-issll-framing-ext-00.txt
Expires: May 25, 1997

R. Andrades
isochrone, Inc.
F. Burg
AT&T
November 26, 1996

QOSPPP Framing Extensions to PPP
(draft-ietf-issll-framing-ext-00.txt)

Status of this Memo

Abstract

The Point-to-Point Protocol (PPP) [2] provides a standard method for transporting multi-protocol datagrams over point-to-point links. PPP datagrams are often encapsulated in HDLC frames [1] when used over standard analog modems.

This document describes the extensions to PPP encapsulation and HDLC framing to support Quality of Service (QoS) over low bandwidth links.

This document is a submission to the IETF ISSLL working group. Comments are solicited and should be addressed to the working group's mailing list at issll@mercury.lcs.mit.edu and/or the author.

This document is an update of the previous version which was named "draft-andrades-framing-ext-00.txt" and was revised as a result of discussions with Carsten Bormann, as well as the discussions at the September 30th meeting of the ISSLL working group.

Table of Contents

1. Introduction

QOSPPP provides an architectural framework for providing multimedia and other advanced services, requiring QoS over the Internet. Our current work is aimed at the consumer market which usually uses comparatively low bandwidth analog modems for Internet access. The links from the consumer's residences to their Internet Service Providers usually run the PPP protocol [2] encapsulated in asynchronous HDLC frames [1], and so our goal was to add QoS to this framing scheme. The current document describes our attempt to extend the PPP encapsulation in asynchronous HDLC framing to support QoS over these links. One goal was to maintain as much compatibility as possible with current PPP implementations and to fall back to the standard PPP/HDLC framing scheme if we detect that the extensions are not available on the peer. The framework also included a signalling engine which is used to negotiate parameters for the QoS connections, and a packet scheduler which contains the actual scheduling algorithms. The signalling protocol could be Q.2931 or RSVP or a variant of one of these. The term QOSPPP is used to refer to the entire framework and not just the framing. The complete QOSPPP architecture will be described in more detail in a future document.

Several Internet Services Providers still offer SLIP connections; however, we felt that this would very soon be obsolete and did not attempt to address it. At this stage we have not attempted to work out the framing extensions for synchronous HDLC, since it is not used in the market we are addressing; however, we feel that the current work can easily be adapted to that area.

Section 2 describes the current implementation and work in progress. Section 3 compares this framing scheme with those [5,6] proposed by Carsten Bormann. Section 4 considers merging the features of QOSPPP with [6].

2. Current Implementation

2.1 QOSPPP Extensions to HDLC framing

The aim of QOSPPP is to allow a customer to run a mix of applications with varying communications needs. Currently most PPP implementations offer a single class of service, best-effort, which is most suited for conventional data applications (e.g. Telnet, ftp, WWW, email). However, newer Internet applications such as packet telephony, video conferencing, etc. require a new class of service with bandwidth guarantees and upper bounds of the delay and jitter seen by their packets.

QOSPPP supports four classes of service, ABR, UBR, CBR and VBR. We use these terms in the same sense as defined by the ATM Forum [7]. However, the basic concepts are the same for any other definition of class of service.

ABR or Available Bit Rate supports traditional data applications, which do not need bandwidth guarantees, nor any strict bounds on their delay and jitter. They typically have variable sized packets. However, ABR applications ARE QoS aware and will specify their maximum datagram size, expected bandwidth usage, and maximum tolerable delays. The class of service is specified in the flowspec along with other parameters like bandwidth, delay and jitter. The application programming interface (API) MUST provide an interface by which the flowspec can be communicated by the application to the transport stack, but this is out of the scope of this document. While the network does not guarantee the latter two, it does use them to estimate buffer sizes and expected load. UBR or Unspecified Bit Rate is for legacy applications that are not QoS aware. ABR and UBR are equivalent to the framing layer and so are both referred to as ABR for the remainder of this document.

CBR or Constant Bit Rate is for applications that transmit data at regular intervals. The datagrams are usually small and of fixed length (though the latter is not a requirement). An example is a packet phone which does not do silence detection. They do have strict upper bounds on the delay and jitter they can tolerate as well as strict bandwidth requirements. VBR or Variable Bit Rate is similar to CBR, except that the rate of packet transmission is not fixed. The transmission rate may vary upto a maximum rate, and it also defines a long term average rate. CBR and VBR are equivalent to the framing layer and so are both referred to as QoS streams for the remainder of this document.

The framing layer then has to support two classes of datagrams, normal data applications and QoS. Most of this support is done by a packet scheduler and signalling engine at a layer higher than the framing layer, and so will not be discussed in this document. However, one aspect of QoS that is strongly influenced by the framing layer is the delay bounds of QoS streams. This is because the delay allowance of QoS streams tends to be in the range of tens of milliseconds. However, several popular data applications (e.g. WWW traffic) tend to use large size packets since they are more efficient. On a 28,800 link, a 1500 byte packet (which is the MTU used by most PPP implementations, and many data applications use the MTU) takes approximately half a second to transmit, making the link unavailable for that time. (In practice, the bandwidth available with a 28,800 modem is often less than 28,800, depending on the line conditions.) This clearly indicates that in order to

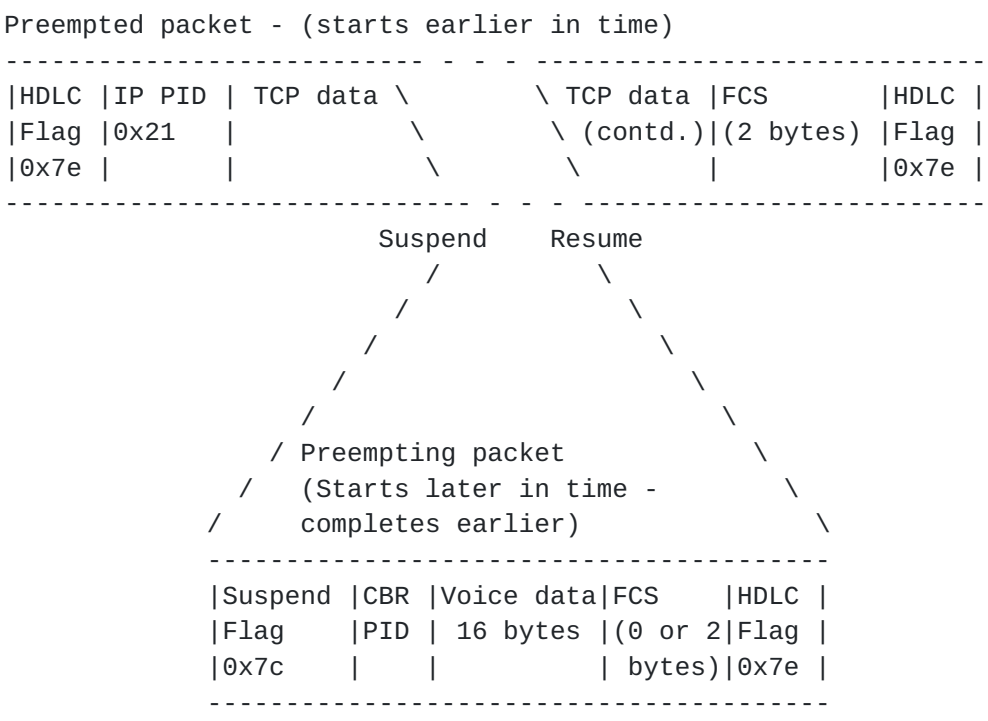
support QoS streams on low bandwidth links, some change in the standard PPP framing is required. One possible way to handle it would be to use a much smaller MTU. However, in order to support

delay bounds of around 20 ms for QoS streams, it would be necessary to restrict the MTU of ABR traffic to around 36 bytes, which is clearly unacceptable.

Another approach (which is what QOSPPP does) is to allow an ABR datagram that is currently being transmitted to be preempted by a QoS stream with stricter delay bounds. When the QoS packet is complete, the preempted ABR datagram will resume transmission. The difference between preemption and conventional fragmentation is that each resuming segment of the ABR datagram does NOT carry a header telling the receiving end how to put the pieces back together again. The preemption is indicated by stuffing a preemption flag byte which is defined as 0x7c. The end of preemption is indicated by transmitting a standard HDLC Flag byte. When preemption ends, the interrupted frame is automatically resumed at the point where it was suspended, with no extra header bytes. The current implementation supports a single level of preemption, however we are planning a new version with support for multiple levels of preemption, see section 2.5.

In this document we use the terms priority and preemption class interchangeably although that may not be the common usage.

The preemption is explained by the following diagram. Assume that there are two active streams, a TCP stream (maybe a Web browser) which is ABR, and a voice stream (the latter for a packet phone application) which is CBR (or VBR if silence detection is implemented). Initially, in the absence of any voice data, the framer begins transmission of a packet of the TCP stream. After some time, a voice packet is queued for transmission and the framer suspends transmission of the TCP stream to begin sending the voice packet. When transmission of the voice packet is complete, the framer resumes transmission of the suspended TCP packet. Note that one voice packet can not interrupt another voice packet. The payload size of the voice packet (16 bytes in the example), depend on the encoding algorithm used. Also note that each FCS in the following diagram protects only the frame it is a part of, (the one that is terminated by the HDLC Flag byte immediately after the FCS), and not any intermediate (preempting) frames, Each stream can negotiate the use of an FCS of a different strength (size). The new Suspend Flag byte needs to be added to the ACCM for transparency. Note that in the diagram, and elsewhere in this document, the term PID always refers to the PPP protocol ID.



As can be seen from the preceding diagram, the preempting packet's frame format is the same as the standard PPP frame except for the use of a new Suspend Flag (0x7c) instead of the HDLC Flag (0x7e).

One other difference is that the length of the FCS field can be different for different PIDs. PPP currently defines three different FCS', 16-bit, 32-bit and NULL FCS. We use the 16-bit FCS for all data streams (it is the default anyway). The NULL FCS can be negotiated for CBR streams that have a small MTU by the signalling engine. However, this choice, if desired, must be made by the application. The interface by which the application may negotiate the choice of an FCS is out of the scope of this document. In any examples in this document, the length of the FCS shown is just for illustration.

In the special case, when we have only a single active QoS stream, we omit sending the CBR PID in preempting frames, since it is implicit from the Suspend Flag, and reduces the Framing overhead by a byte.

2.2 QOSPPP Extensions to PPP encapsulation

All CBR and VBR streams are each assigned a unique PPP protocol ID (PID) by the signalling engine. These PID values are taken from the unused values as specified in [8]. We plan to get a range of these unused PIDs allocated for this purpose. Unlike the PIDs currently assigned to protocols by the IETF, the PIDs used by QOSPPP are not necessarily the same in both directions, because of the way the signalling engine works. We try to pick the PIDs from the 1-byte PID space (and since we do not expect too many streams to be simultaneously active over these low bandwidth links, it isn't difficult to find enough 1-byte PIDs).

2.3 QOSPPP Extensions to Link Establishment Protocol

PPP uses the Link Control Protocol (LCP) [2] to establish parameters for the link. Up-to-date values of the LCP Code field are specified in the most recent "Assigned Numbers" RFC [8].

QOSPPP adds the following configuration option:

Option	Length	version	Preemptive	Link Speed
0x55	= 8	(4 bytes)	scheduling	Monitoring
			(1 byte)	(1 byte)

Option: This is a new LCP configuration option code value.

Version: This field is currently set to 1. Future versions may set this field to other values to indicate other preemption schemes.

Preemptive scheduling: This is set to 0 to indicate that QoS streams are not currently supported (even though the peer apparently is QOSPPP-enabled). The current QOSPPP framing uses a value of 1 in this field. In the future, we plan to use this field to indicate the number of levels of preemption supported (Currently it is one level).

Link Speed Monitoring: This is currently set to 0 to indicate that Link Speed Monitoring is not required. The purpose of Link Speed Monitoring is to enable an implementation to track changes in the link bandwidth and adjust it's packet scheduling accordingly. No protocol has yet been defined for Link Speed Monitoring.

The node at one end sends a configuration message indicating that it supports preemption, and the maximum number of preemption levels it supports. If the other node responds with a reject (i.e. it is a non QoS-aware implementation and does not recognize the new LCP option code), then the sender will disable the preemption capability and send a new configuration message without the preemption option (and disable the preemption feature locally for the current session). If the other side responds with a NAK requesting a smaller number of levels of preemption, we will adjust our behavior accordingly and resend the configuration message reflecting the requested changes.

2.4 UDP header compression

QOSPPP uses UDP header compression. This consists of not transmitting the UDP and IP headers of any packet that is associated with a specific PID. The receiver automatically prepends the UDP & IP headers to every incoming packet.The information needed for the headers is transmitted when the PID is negotiated by the signalling engine (the checksum can be generated by the receiver on the reconstructed received packet - or we can use the optimization of setting the checksum to 0). Carsten proposes prefix elision [6] which is basically associating each class or PID with a prefix of bytes that begin every packet belong to that class or PID and then not transmitting those bytes. The receiver automatically prepends the prefix to every packet it receives. UDP header compression gives better reduction in overhead than prefix elision for UDP streams. It does not handle non-UDP streams, but we assume Van Jacobson does a fairly good job at that. Are there other non-UDP, non-TCP streams that we have to consider? If so, we can do prefix elision for these streams. (This will have to be negotiated by the signalling protocol on a per-stream basis).

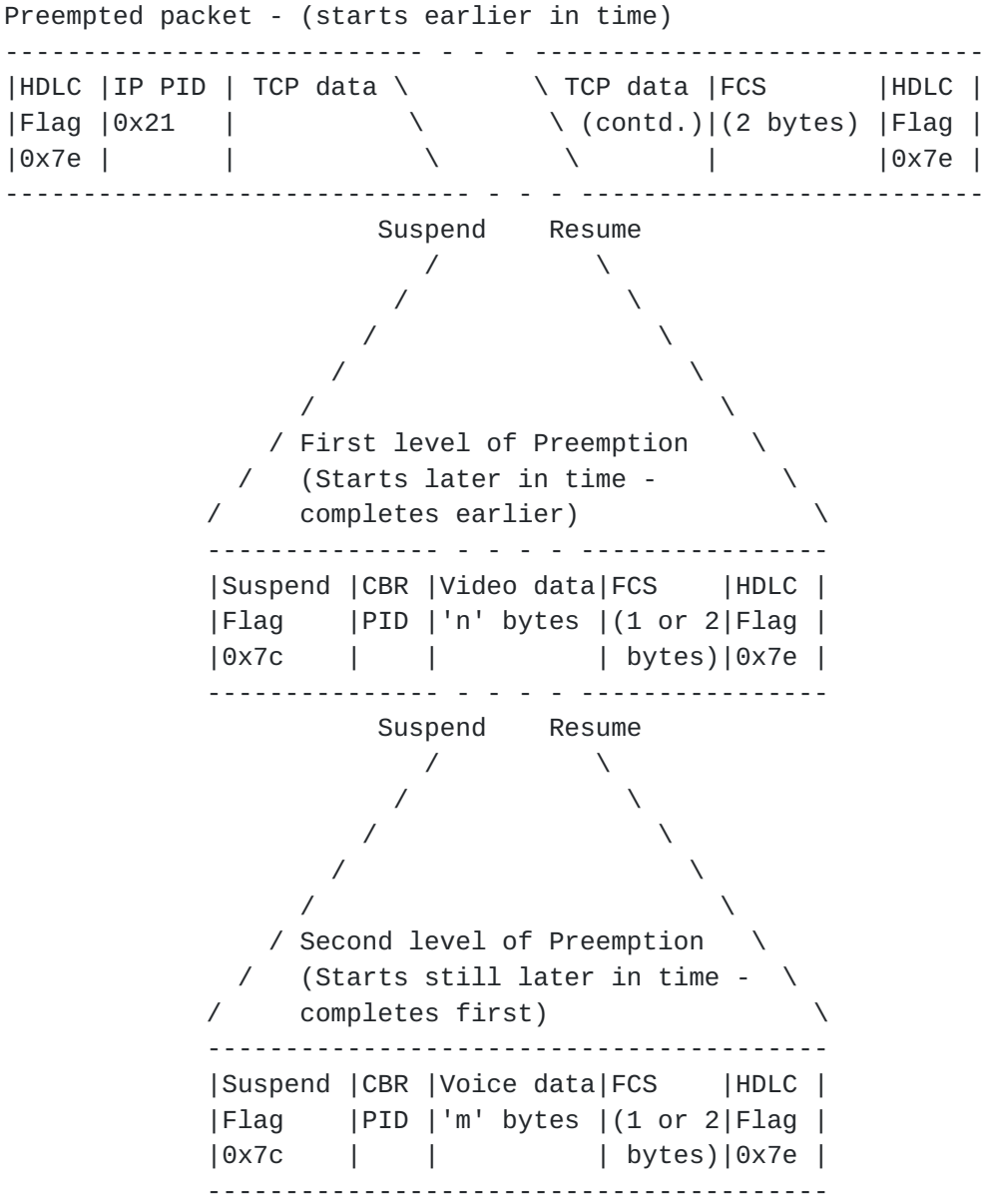
2.5 Planned QOSPPP Features

We will try to choose the PID values so that their Hamming distance is at least two, allowing single bit errors in the PID field to be detected.

In the QOSPPP Framing engine the FCS field can be turned off for individual CBR and VBR streams. We could also adopt Carsten's idea [6] of using an 8-bit CRC for CBR and VBR streams. Thus the effective Framing overhead can be reduced by a byte for CBR and VBR streams. We propose to use the 8-bit FCS described in the V.76 specification [9], unless the IETF already has a standard for an 8-bit FCS.

Another planned extension to QOSPPP is multiple levels of preemption. In this we will put different streams into different preemption classes and allow a packet of a stream of a higher preemption class to preempt a currently transmitting packet of a stream of a lower preemption class. Currently, we do not plan to support dynamic priorities (where two streams' relative priorities can change dynamically). The QOSPPP frame format can support multiple preemption levels without any change.

Multiple preemption levels are explained by the following diagram that shows two levels of preemption. Assume that there are three active streams, a TCP stream (maybe a Web browser), a video stream and a voice stream (the latter two for video conferencing). Assume that the video stream belongs to a higher preemption level than the TCP stream and the voice stream belongs to a higher preemption level than the video stream. Initially, in the absence of any voice or video data, the framer begins transmission of a packet of the TCP stream. After some time, a video packet is queued for transmission and the framer suspends transmission of the TCP stream to begin sending the video packet. Before transmission of the video packet is complete, a voice packet is queued for transmission. The framer suspends transmission of the video stream to begin sending the voice packet. When transmission of the voice packet is complete, the framer resumes transmission of the suspended video packet. When transmission of the video packet is complete, the framer resumes transmission of the suspended TCP packet. Note that a video packet can not interrupt a voice packet or another video packet, and one voice packet can not interrupt another voice packet.



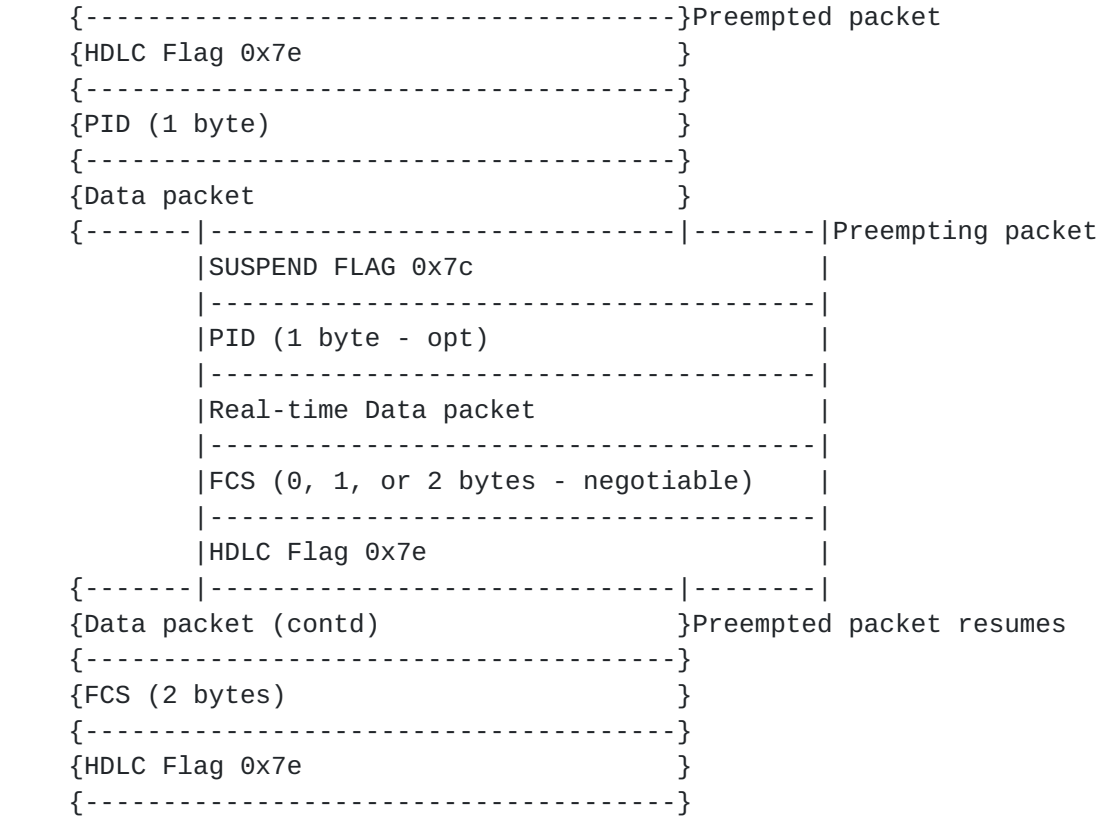
Note that the suspended packets have to be resumed in the reverse (stack-like or LIFO) order; i.e. if streamA preempts streamB preempts streamC, then when streamC completes, streamB must resume (and complete) before streamA is resumed. Currently we do not expect to support resumption of suspended packets in a different order (which, by the way, would necessarily imply implementing dynamic priorities).

3. Comparison with other proposals

In all the descriptions below we are assuming that the address-control field compression and protocol field compression

options have been negotiated by the link control protocol, and that all PIDs used for the real-time streams are 1 byte.

Case 0. The QOSPPP fragmentation format



This frame has between 5 and 2 bytes of overhead per preemption; the minimum of two bytes is in the case when there is only a single active stream of a high preemption class, and it has been negotiated not to require the use of a CRC; the maximum of five bytes is when there are multiple active streams of high preemption classes (either in the same class, or in different preemption classes), and they require the use of a 2 byte CRC. We could also consider using a 1 byte CRC (as suggested by Carsten Bormann), for streams whose packets are rather short. Note that there is no overhead on the interrupted packet. Therefore every low priority packet carries 5 bytes of framing overhead (regardless of whether it is preempted or not), and every higher priority packet carries 2 to 5 bytes of overhead.

There is no limit on the number of preemption levels, except the number of bits in a PID (though not all combinations are valid).

In the case of multiple levels of preemption, it assumes that suspended frames will be resumed in the reverse order, from that in which they were suspended.

In the absence of preemption, the frame format is exactly the same

as regular PPP.

We will consider the following proposals from Carsten Bormann and try to consider all the optimizations too.

Case 1. Using PPP Multi-Link as-is (short sequence number fragment format)

In this scheme, every lower priority packet needs to be sent in at least two MLPPP frames. (Since we do not know whether it is going to be interrupted or not, we must begin transmitting with the "E" bit set to "0". Therefore, even if it is not interrupted, we need to send a final (empty) fragment with the "E" bit set to "1" to terminate the packet). Now, the MLPPP frame has 6 bytes of framing overhead, therefore every lower priority packet has 6*2= 12 bytes

of framing overhead. However, the MLPPP packet actually carries a PPP packet within it adding an additional 1 byte overhead (for the

PPP PID) making the total framing overhead 13 bytes. We can save one byte by not transmitting two consecutive Flag bytes making the total framing overhead 12 bytes as opposed to 5 bytes for a normal PPP frame. For every preemption, the lower priority packet needs to be terminated with an MLPPP trailer (3 bytes) and restarted with an MLPPP header (3 bytes) adding an additional 6 bytes overhead. Additionally, the preempting packet needs it's PPP header of 5 bytes, giving a total framing overhead of 11 bytes per preemption. Dropping consecutive Flag bytes will save two bytes giving a total framing overhead of 9 bytes per preemption as opposed to 5 bytes for a normal PPP frame. In case the interrupted packet is resumed near it's end (e.g. it has fewer than say 7 bytes left to transmit), we can assume that it will not be interrupted again and can send the last fragment with the "E" bit set to "1", thus eliminating the 5 bytes of the empty MLPPP header at the end.

It supports a single level of preemption.

It can be used even if the other end does not support QoS. This however, assumes that the other end supports MLPPP; We are not sure how many implementations, and how many service providers support MLPPP for analog lines.

Case 2. Extending PPP Multi-Link to multiple class (short sequence number fragment format)

The difference between this scheme and case 1 is that it supports 4 levels of preemption. However, now preempting packets carry an MLPPP header (plus a PPP PID), instead of a normal PPP header (except for one of the preemption levels which uses normal PPP frames). Thus the preempting packets carry (6(MLPPP header) + 1(PPP PID) =) 7 bytes of framing overhead per packet. So the total framing overhead per preemption is (6 (preempted) + 7 (preempting)

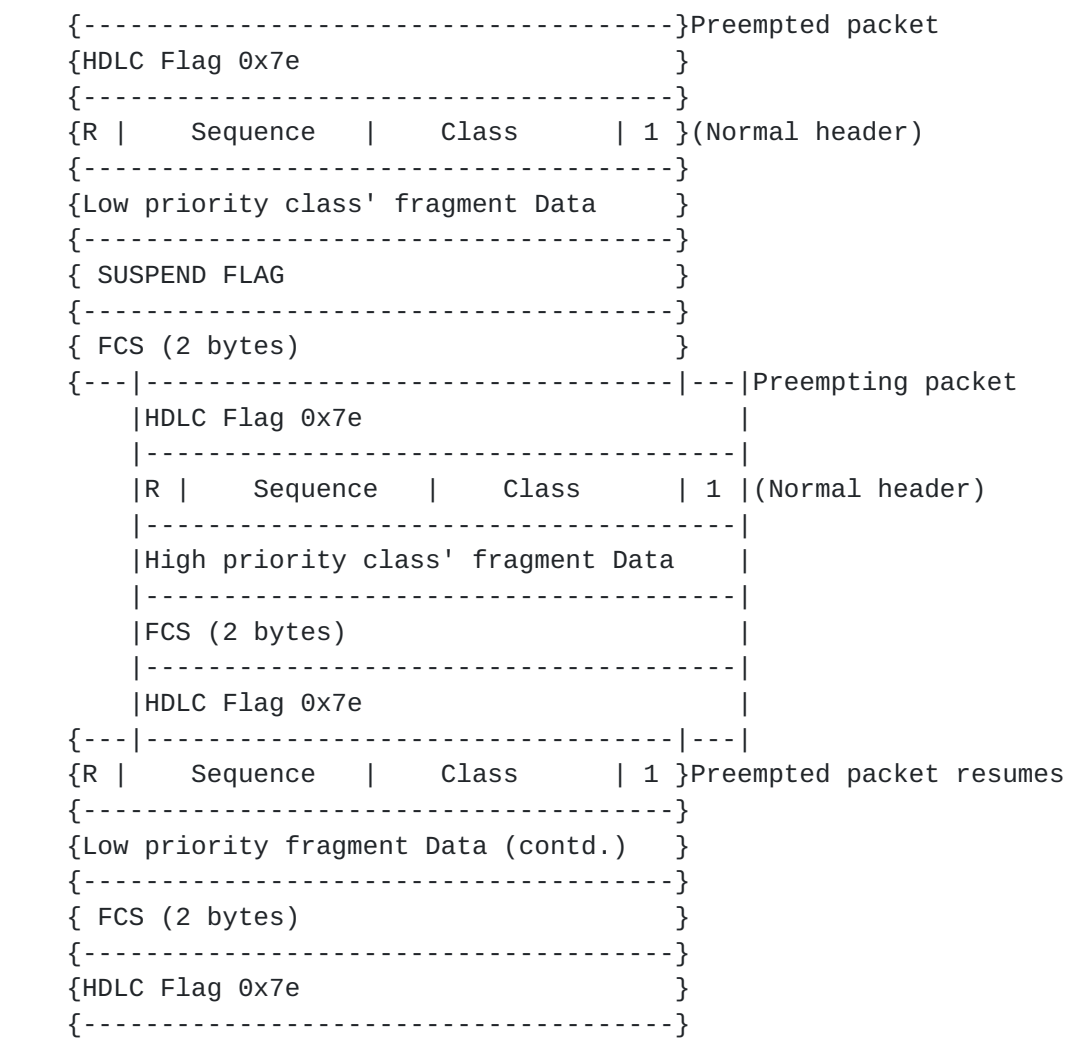
=) 13 bytes. Again, dropping consecutive Flag bytes will save two bytes giving a total framing overhead of 11 bytes per preemption

as opposed to 5 bytes for a normal PPP frame.

It is backward compatible to case 1; i.e. if the remote end does not support QoS, we can fall back to case 1, restricting ourselves to a single level of preemption.

In the case of multiple levels of preemption, suspended frames can be resumed in any order, not necessarily the reverse order from that in which they were suspended.

Case 3. The Compact Fragment Format (Normal header)



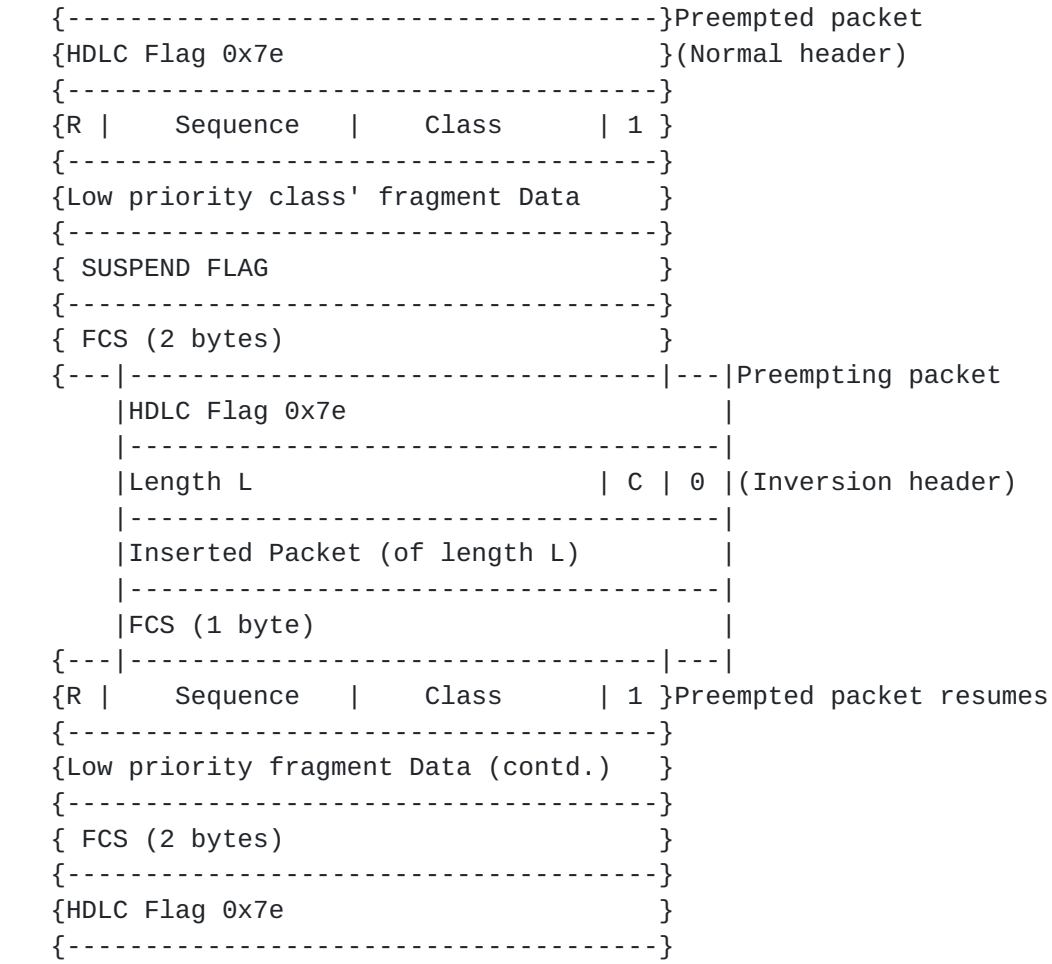
Note: In the diagram above, the PID bytes and the TERMINATE flag bytes have been dropped by optimizations.

It supports 7 levels of preemption.

In the case of multiple levels of preemption, suspended frames can be resumed in any order, not necessarily the reverse order from that in which they were suspended.

The preempting packet's header has 5 bytes of overhead which is the same as the normal PPP header. However the lower priority packet has terminated by an FCS and SUSPEND byte (3 bytes), and when it resumes, it will carry a 1 byte header, adding 4 bytes of overhead per preemption. Also the Higher priority packet will need 5 bytes of framing overhead, making the total framing overhead (5 + 4 =) 9 bytes per preemption. (This is assuming that it is not necessary to send consecutive Flag bytes.) So this scheme has a total framing overhead of 9 bytes per preemption as opposed to 5 bytes for a normal PPP frame.

Case 4. The Compact Fragment Format (Insertion Header)



It supports 2 levels of preemption.

The preempting packet's header has 3 bytes of overhead which is less than the normal PPP header. However the lower priority packet has terminated by an FCS and SUSPEND byte (3 bytes), and when it resumes, it will carry a 1 byte header, adding 4 bytes of overhead per preemption. Also the Higher priority packet will need 3 bytes of framing overhead, making the total framing overhead (3 + 4 =) 7 bytes per preemption. (This is assuming that it is not necessary to send consecutive Flag bytes.) So this scheme has a total framing overhead of 7 bytes per preemption as opposed to 5 bytes for a normal PPP frame.

It does have the restriction that the higher priority packet be not more than 64 bytes in length.

Comparison of Overhead

Cases 1 & 2 have more overhead than the rest. Case 1 is useful only if it is necessary to support (a single level of) preemption even for links where the peer does not support QoS. Even this is debatable for two reasons:

(1) How many implementations actually support MLPPP for analog lines, and,

(2) Preemption by itself is generally not sufficient to support QoS for analog lines, one also needs to do some form of header compression, especially considering the increased size of the MLPPP headers. Would the remote end which does not have support for QoS support the header compression scheme?

Case 2 does not seem useful considering that it requires the remote end to support a non-standard extension to MLPPP. If the remote end has to be modified to support this extension, one should question why it can not be modified to support some other, more efficient, extension. It does have the advantage of being able to gracefully fallback to case 1, but as mentioned above, the value of this advantage seems to be rather dubious. (At the September 30th ISSLL meeting in Billerica, it was agreed to make case 2 the baseline case and work from there towards a more optimized scheme.)

Compare cases 0, 3 & 4 in more detail.

One of the optimizations in case 4 that caused a 1 byte reduction in overhead (the smaller FCS) can also be applied to cases 0 and 3. The second optimization (dropping the intermediate Flag byte) is achieved mainly by putting a length field in the header and shrinking the class number to 1 bit. The former restricts the

length of high priority packets to 64 bytes which may be O.K., the latter restricts the number of high priority streams to 2, which makes it O.K. as an optimization of case 3 rather than as a separate case (which anyway, is what Carsten presents it as).

Let us examine cases 0 & 3. Consider the following scenarios for cases 0 & 3 (with case 4 as an optimization of case 3).

1. normal case, case 0 has 5 bytes overhead, case 3 has 9 bytes
2. 8-bit FCS, overhead reduces by 1 byte for both cases.
3. No FCS, overhead reduces by 2 bytes for both cases.
4. A single high priority stream, reduces case 0 overhead by 1 byte by dropping the PID. If the length of the packet is less than 64 bytes, case 3 reduces to case 4, saving 2 bytes.
5. Upto 2 high priority streams whose packet lengths are less than 64 bytes, and which use an 8-bit FCS can have their overhead reduced to 7 bytes for case 3 by using the case 4 optimizations. This compares with 4 bytes for case 0 under the same conditions, except that you can do it for a greater number of streams.
6. Case 0 overhead can be reduced by 1 byte by dropping the intermediate Flag byte, however this can be done only for streams that have fixed size packets, and it carries the danger of two packets (the preempted and the preempting,) being corrupted if any byte of the preempting packet is dropped.

As can be seen, there does not appear to be a clear advantage of one scheme over the other.

Note that the frame formats of cases 1 & 2 actually indicate fragmentation rather than preemption, since each frame carries a header telling the receiver how to put it back together. The idea behind preemption is precisely to avoid the overhead of this kind of header. However, although the frame format makes it look like fragmentation, calling it preemption is justifiable from the point of view of the actual operation of the protocol; i.e., with conventional fragmentation, the stack will decide on how to fragment a packet either when it is given a packet from the higher layer, or, when it decides to begin transmission. In preemption, the decision of how, when, etc. to "fragment" is made at the time a higher priority "preempting" packet becomes eligible for transmission. This distinction can only be made at the sending side, for the receiver it does look like fragmentation.

Comparison of error detection capability

Case 3 packets have a sequence number which will allow it to detect a lost fragment. Of course, even in case 0, lost fragments can be

caught by the FCS, but the sequence number provides an additional check.

Case 0 has the disadvantage that if a deeply nested preempting frame (i.e., one that has caused several other frames to be recursively preempted), is corrupted, you run the risk of being forced to discard the stack of preempted frames. This will happen if transmission or overrun errors cause any FLAG bytes to be lost or corrupted, in this case the receiver, on precessing a FLAG byte, may not be able to correctly match it with the corresponding SUSPEND byte. However, we suspect that this topic is not as straightforward as it seems and needs further analysis.

Comparison of number of levels of preemption

Consider cases 0 and 3 alone as case 4 is an optimization of case 3.

Case 0 supports an arbitrary number of levels of preemption, limited only by the PID space. Case 3 supports 7 levels of preemption. There does not seem to be much to choose between them here as 7 levels of preemption are probably enough.

Comparison of support for dynamic priorities

Case 3 appears to have slightly better support for dynamic priorities. However, let us see if this advantage is meaningful.

There is one way in which the use of dynamic priorities can affect the framing format. Consider the case where there are three stream A, B and C, in the increasing order of priorities. Assume that the priorities are dynamic so the priority ordering may change over time. Now consider a scenario where stream B has preempted stream A and stream C has preempted stream B. Suppose stream A gets a priority boost making it temporarily of higher priority that stream B (but lower than stream C). Therefore when stream C completes transmission of it's packet, there is an issue of whether we should resume transmission of stream A or stream B. Carsten's proposals (cases 2 & 3) give the implementation the choice in this matter, QOSPPP does not.

This does not mean that dynamic priorities are impossible with QOSPPP. Dynamic priorities can still be used in controlling the decision of preemption of one stream by another, they simply can not be used for making the resumption decision. Also, the scenario above could be considered farfetched by some. For that matter, the whole idea of dynamic priorities might be considered irrelevant for the applications we are considering; the alternative is to allow a slightly larger jitter, which might be perfectly acceptable.

Case 0, being limited to a strict stack-like sequence of suspends-resumes might be slightly simpler to implement.

4. Conclusions

Based on the discussion above, we suggest that any framing format adopted should attempt to borrow the best features of both the QOSPPP framing and Carsten's proposals. In particular, we suggest keeping the following features:

UDP header compression. This gives gives a significant reduction in overhead for UDP streams.

In the case of a single higher priority stream, keep the option of dropping the PID byte. This however, has to be negotiated by LCP.

The signalling protocol will negotiate the size of the FCS for each stream.

Use one of Carsten's values of 0xDE or 0xC3 for the Preemption flag. Maybe we should run tests over a PPP link (before the byte stuffing stage) rather than over an Ethernet as he did.

We feel that the requirement that packets be resumed in the reverse order from that in which they were suspended (even for Carsten's proposals), would aid the receiver in error detection.

The use or non-use of dynamic priorities is an independent decision which will not affect the frame format. Also, it may be possible to restrict the implementation of dynamic priorities to the sending side alone.

5. Security Considerations

This document does not raise any new security issues.

6. Acknowledgments

Much of the work in implementing the QOSPPP architecture and testing the concepts presented in this document was done by Murali Aravamudan and Kumar Vishwanathan of isochrone, Inc. Andreas Papanicolau and Khasha Mohammadi of AT&T also provided many helpful insights in the design of the architecture.

7. References

[1] Simpson, W., Editor, "PPP in HDLC Framing", RFC 1662, STD 51, Daydreamer, July 1994.

[2] Simpson, W., Editor, "The Point-to-Point Protocol (PPP)", RFC 1661, STD 51, Daydreamer, July 1994.

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

[4] McGregor, G., "The PPP Internet Control Protocol", RFC 1332, Merit, May 1992.

[5] Bormann, Carsten, "Providing integrated services over low-bitrate links", work in progress, Internet Draft (draft-ietf-issll-isslow-00.txt), Universitaet Bremen, June 1996.

[6] Bormann, Carsten, "The Multi-Class Extensions to Multi-Link PPP", work in progress, Internet Draft, (draft-ietf-issll-isslow-mcml-00.txt), Universitaet Bremen, September 1996.

[7] "ATM User-Network Interface (UNI) Specification Version 3.1", The ATM Forum, 1995.

[8] Reynolds, J., and J. Postel, "Assigned Numbers", STD 2, RFC 1700, USC/Information Sciences Institute, October 1994.

[9] Burg, F., editor, "Recommendation V.76 - Generic Multiplexer using V.42 LAPM-based procedures", International Telecommunication Union, April 1996.

8. Author's Address

Questions about this memo can be directed to:

Richard Andrades
isochrone, Inc. Phone: (908) 544 5508
One Main Street Fax: (908) 544 2059
Suite 511 Email: richard@isochrone.com
Eatontown, NJ 07724

Fred M. Burg
AT&T
307 Middletown-Lincroft Road Phone: (908) 576 4322

Room 3L209
Lincroft, NJ 07738

Fax: (908) 576 4689
Email: fred.burg@att.com