Network Working Group
Request for Comments: 2118
Category: Informational

G. Pall
Microsoft Corporation

March 1997

Microsoft Point-To-Point Compression (MPPC) Protocol

Status of this Memo

This memo provides information for the Internet community. This memo does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Abstract

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

The PPP Compression Control Protocol [2] provides a method to negotiate and utilize compression protocols over PPP encapsulated links.

This document describes the use of the Microsoft Point to Point Compression protocol (also referred to as MPPC in this document) for compressing PPP encapsulated packets.

Table of Contents

1. Introduction 2
1.1 Licensing 2
1.2. Specification of Requirements 2
2. Configuration Option Format 3
3. MPPC Packets 4
3.1 Packet Format..... 5
4. Description of Compressor and Encoding 6
4.1 Literal Encoding 7
4.2 Copy Tuple Encoding 7
4.2.1 Offset Encoding 7
4.2.2 Length-of-Match Encoding 7
4.3 Synchronization 8
SECURITY CONSIDERATIONS 8
REFERENCES 9
ACKNOWLEDGEMENTS 9
CHAIR'S ADDRESS 9
AUTHORS' ADDRESS 9

1. Introduction

The Microsoft Point to Point Compression scheme is a means of representing arbitrary Point to Point Protocol (PPP) packets in a compressed form. The MPPC algorithm is designed to optimize processor utilization and bandwidth utilization in order to support large number of simultaneous connections. The MPPC algorithm is also optimized to work efficiently in typical PPP scenarios (1500 byte MTU, etc.).

The MPPC algorithm uses an LZ [3] based algorithm with a sliding window history buffer.

The MPPC algorithm keeps a continous history so that after 8192 bytes of data has been transmitted compressed there is always 8192 bytes of history to use for compressing, except when the history is flushed.

1.1. Licensing

MPPC can only be used in products that implement the Point to Point Protocol AND for the sole purpose of interoperating with other MPPC and Point to Point Protocol implementations.

Source and object licenses are available on a non-discriminatory basis from Stac Electronics. Please contact:

Cheryl Poland
Stac Electronics
12636 High Bluff Drive,
San Deigo, CA 92130
Phone: (619)794-4534
Email: cherylp@stac.com

1.2. 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.

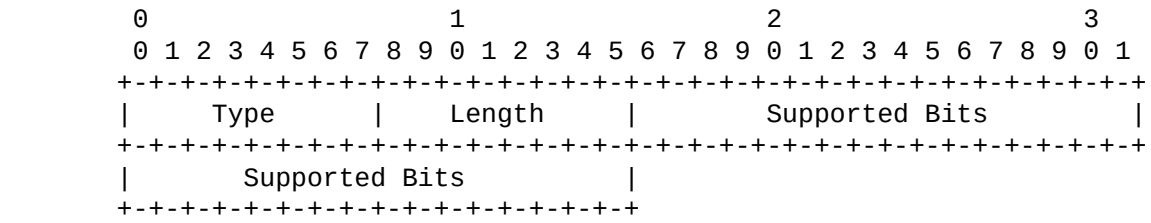
MAY This word, or the adjective "optional", means that this item is one of an allowed set of alternatives. An implementation which does not include this option MUST be prepared to interoperate with another implementation which does include the option.

2. Configuration Option Format

Description

The CCP Configuration Option negotiates the use of MPPC on the link. By default or ultimate disagreement, no compression is used.

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



Type

18

Length

6

Supported Bits

This field is 4 octets, most significant octet first. The least significant bit in the least significant octet set to 1 indicates desire to negotiate MPPC.

All other bits MUST be set to 0.

3. MPPC Packets

Before any MPPC packets may be communicated, PPP must reach the Network-Layer Protocol phase, and the CCP Control Protocol must reach the Opened state.

Exactly one MPPC datagram is encapsulated in the PPP Information field. The PPP Protocol field indicates type hex 00FD for all compressed datagrams.

The maximum length of the MPPC datagram transmitted over a PPP link is the same as the maximum length of the Information field of a PPP encapsulated packet. Since the history buffer is limited to 8192 bytes, this length cannot be greater than 8192 bytes.

Only packets with PPP Protocol numbers in the range hex 0021 to hex 00FA are compressed. Other packets are not passed thru the MPPC processor and are sent with their original PPP Protocol numbers.

Padding

It is recommended that padding not be used with MPPC since it defeats the purpose of compression. If the sender must use padding it MUST negotiate the Self-Describing-Padding Configuration option during LCP phase and use self-describing pads.

Reliability and Sequencing

The MPPC scheme does not require a reliable link. Instead, it relies on a 12 bit coherency count in each packet to keep the history buffers synchronized. If the receiver recognizes that the coherency count received in the packet does not match the count it is expecting, it sends a CCP Reset-Request packet to resynchronize its history buffer with the sender’s history buffer.

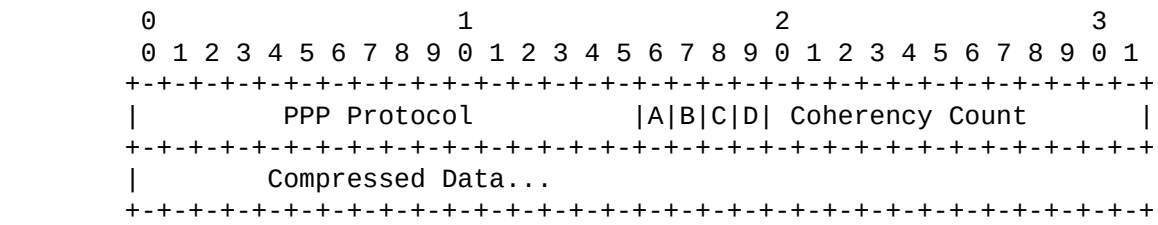
MPPC expects the packets to be delivered in sequence, otherwise history buffer re-synchronization will not occur.

MPPC MAY be used over a reliable link, as described in "PPP Reliable Transmision" [5], but this typically just adds unnecessary overhead since only the coherency count is required.

Data Expansion

If compressing the data results in data expansion, the original data is sent as an uncompressed MPPC packet. The sender must flush the history before compressing any more data and set the FLUSHED bit on the next outgoing packet.

3.1. Packet Format



PPP Protocol

The PPP Protocol field is described in the Point-to-Point Protocol Encapsulation [1].

When the MPPC compression protocol is successfully negotiated by the PPP Compression Control Protocol, the value is hex 00FD. This value MAY be compressed when Protocol-Field-Compression is negotiated.

Bit A

This bit indicates that the history buffer has just been initialized before this packet was generated. This packet can ALWAYS be decompressed because it is not based on any previous history. This bit is typically sent to inform the peer that the sender has initialized its history buffer before compressing the packet and that the receiving peer must initialize its history buffer before decompressing the packet. This bit is referred to as FLUSHED bit in this document.

Implementation Note: Compression and decompression histories are always initialized with all zeroes.

Bit B

This bit indicates that the packet was moved to the front of the history buffer typically because there was no room at the end of the history buffer. This bit is used to tell the decompressor to set its history pointer to the beginning of the history buffer.

Implementation Notes:

1. It is implied that this bit must be set at least once for every 8192 bytes of data that is sent compressed.
2. It is also implied that this bit can be set even if the sender's history buffer is not full. Initialized history that has not been used for compressing data must not be referred to in the compressed packets.

Bit C

This bit (if set) is used to indicate that the packet is compressed.

Bit D

This bit must be set to 0.

Coherency Count

The coherency count is used to assure that the packets are sent in proper order and that no packet has been dropped. This count starts at 0 and is always increased by 1 and NEVER decreases or goes back. When all bits are 1, the count returns to 0.

Compressed Data

The compressed data begins with the protocol field. For example, in case of an IP packet (0021 followed by an IP header), the compressor will first try to compress the 0021 protocol field and then compress the IP header.

If the packet contains header compression, the MPPC compressor is applied AFTER header compression is preformed and MUST be applied to the compressed header as well. For example, if a packet contained the protocol 002d for a compressed TCP/IP header, the compressor would first attempt to compress 002d and then it would attempt to compress the compressed Van-Jacobsen TCP/IP header.

4. Description of Compressor and Encoding

The compressor runs through the length of the frame producing as output a Literal (byte to be sent uncompressed) or a <Offset, Length-of-Match> Copy tuple, where Offset is the number of bytes before in the history where the match lies and Length-of-Match is the number of bytes to copy from the location indicated by Offset.

For example, comsider the following string:

```
0         1         2         3         4
01234567890123456789012345678901234567890123456789
for whom the bell tolls, the bell tolls for thee.
```

The compressor would produce:

for whom the bell tolls,<16,15> <40,4><19,3>e.

The Literal and Copy tuple tokens are then encoded according to the MPPC encoding scheme.

4.1 Literal Encoding

Literals are bytes sent uncompressed. If the value of the Literal is below hex 80, it is encoded with its value itself. If the Literal has value greater than hex 7F it is sent as bits 10 followed by the lower 7 bits of the Literal.

Example: Literal hex 56 is transmitted as 01010110
Literal hex E7 is transmitted as 101100111

4.2 Copy Tuple Encoding

Copy tuples represent compressed data. A tuple has two elements: the Offset and Length-of-Match. The Offset is encoded before the Length-of-Match.

4.2.1 Offset Encoding

Offset values less than 64 are encoded as bits 1111 followed by the lower 6 bits of the value.

Offset values between 64 and 320 are encoded as bits 1110 followed by the lower 8 bits of the computation (value - 64).

Offset values between 320 and 8191 are encoded as bits 110 followed by the lower 13 bits of the computation (value - 320).

Examples: Offset value of 3 is encoded as: 1111 000011
Offset value of 128 is encoded as: 1110 01000000
Offset value of 1024 is encoded as: 110 0001011000000

4.2.2 Length-of-Match Encoding

Length of 3 is encoded with bit 0.

Length values from 4 to 7 are encoded as 10 followed by lower 2 bits of the value.

Length values from 8 to 15 are encoded as 110 followed by lower 3 bits of the value.

Length values from 16 to 31 are encoded as 1110 followed by lower 4 bits of the value.

Length values from 32 to 63 are encoded as 11110 followed by lower 5 bits of the value.

Length values from 64 to 127 are encoded as 111110 followed by lower 6 bits of the value.

Length values from 128 to 255 are encoded as 1111110 followed by lower 7 bits of the value.

Length values from 256 to 511 are encoded as 11111110 followed by lower 8 bits of the value.

Length values from 512 to 1023 are encoded as 111111110 followed by lower 9 bits of the value.

Length values from 1024 to 2047 are encoded as 1111111110 followed by lower 10 bits of the value.

Length values from 2048 to 4095 are encoded as 11111111110 followed by lower 11 bits of the value.

Length values from 4096 to 8191 are encoded as 111111111110 followed by lower 12 bits of the value.

```
Examples: Length of 15 is encoded as:           110 111
          Length of 120 is encoded as:       111110 111000
          Length of 4097 is encoded as:111111111110 000000000001
```

The largest Length value that can be encoded is 8191.

4.3 Synchronization

Packets may be lost during transfer. If the decompressor maintained coherency count does not match the coherency count received in the compressed packet, the decompressor drops the packet and sends a CCP Reset-Request packet. The compressor on receiving this packet flushes the history buffer and sets the FLUSHED bit in the next packet it sends. The decompressor on receiving a packet with its FLUSHED bit set flushes its history buffer and sets its coherency count to the one transmitted by the compressor in that packet. Thus synchronization is achieved without a CCP Reset-Ack packet.

Security Considerations

Security issues are not discussed in this memo.

References

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

[2] Rand, D., "The PPP Compression Control Protocol (CCP)", RFC 1962, Novell, June 1996.

[3] Lempel, A. and Ziv, J., "A Universal Algorithm for Sequential Data Compression", IEEE Transactions On Information Theory, Vol. IT-23, No. 3, May 1977.

[4] Rand, D., "PPP Reliable Transmission", RFC 1663, Novell, July 1994.

Acknowledgments

Thomas Dimitri made significant contributions towards the design and development of Microsoft Point-To-Point Compression Protocol. Robert Friend of Stac Technology provided editoral input.

Chair's Address

The working group can be contacted via the current chair:

Karl F. Fox
Ascend Communications
3518 Riverside Dr., Suite 101
Columbus, Ohio 43221

(614) 451-1883

EMail: karl@ascend.Com

Author's Address

Questions about this memo can also be directed to:

Gurdeep Singh Pall
1, Microsoft Way,
Redmond, WA 98052

(206) 882-8080

Email: gurdeep@microsoft.com