Internet Draft
Document: draft-preston-ftpext-deflate-04.txt
Intended status: Standards Track
Expires: December 30, 2010

J. Preston
NSC
TJ Saunders
June 30, 2010

Deflate transmission mode for FTP
draft-preston-ftpext-deflate

Abstract

This document defines an optional extension to RFC 959, "FILE TRANSFER PROTOCOL (FTP)" (October 1985). It specifies a new "deflate" transmission mode designed to increase network bandwidth by compressing data using existing techniques.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on December 30, 2010.

Copyright Notice

Table of Contents

1. Introduction

As the Internet grows, modern devices and networking environments create new performance challenges for the File Transfer Protocol (FTP) [1]. One solution to this problem, which is addressed in the FTP "compress" transmission mode, is to compress file and system data to maximize network resources. However, the original system is designed to reduce ASCII text with repetitive characters and is unsuitable in many applications because it can add significant network overhead to binary transfers. This document enhances the capabilities of FTP by introducing a new "deflate" transmission mode that:

* increases network throughput and decreases transfer time

* effectively compresses ASCII and binary data

* requires a minimum amount of control information

* provides error recovery and data integrity options

* includes a mechanism to negotiate compression parameters to balance CPU, memory and data requirements

* is extensible to accommodate future compression techniques

2. Document Conventions

This document makes use of the conventions defined in BCP 14 [2] which includes the explanation of capitalized imperative words MUST, SHOULD, MAY, SHOULD NOT and MUST NOT. Any syntax is defined using Augmented BNF (ABNF) as specified in RFC 2234 [3].

The terms "reply", "user", "file", "pathname", "FTP commands", "DTP", "user-FTP process", "user-PI", "user-DTP", "server-FTP

process", "server-PI", "server-DTP", "mode", "type", "NVT", "control connection", "data connection", "transmission mode", "binary" and "ASCII" are all used here as defined in STD 9 [1].

In addition, this specification makes use of the terms "compression engine" and "compression method." A compression engine is a hardware or software component that implements a compression method.

The compression method is a process that reduces the size of computer data.

2.1 Basic Tokens

This document imports the core definitions given in Appendix A of RFC 2234 [3] which includes the ABNF elements like ALPHA, DIGIT, SP, etc. The following terms are added for use in this document:

```
TCHAR  = VCHAR / SP / HTAB     ; visible plus white space
RCHAR  = ALPHA / DIGIT / "," / "." / ":" / "!" /
         "@" / "#" / "$" / "%" / "^" /
         "&" / "(" / ")" / "-" / "_" /
         "+" / "?" / "/" / "\" / "'" /
         DQUOTE   ; <"> -- double quote character (%x22)
```

```
SCHAR   = RCHAR / "=" ;
```

The VCHAR (from [3]), RCHAR, SCHAR, and TCHAR types give basic character types from varying sub-sets of the ASCII character set for use in various commands and responses.

```
token   = 1*RCHAR
```

A "token" is a string whose precise meaning depends upon the context in which it is used. In some cases it will be a value from a set of possible values and in others it might be a string invented by one party for an FTP conversation.

Note that in ABNF, string literals are case insensitive. That convention is preserved in this document, and implies that FTP commands added by this specification have names that can be represented in any case. For example, "MODE" is the same as "mode" and "Mode". However, ALPHA characters are case sensitive which implies a token can have an exact value. That implication is correct, except where explicitly stated to the contrary in this document, or in some other specification which defines the values this document specifies be used in a particular context.

3. Deflate Transmission Mode

The deflate extension introduces a fourth transmission mode to FTP by updating the transfer mode (MODE) command. It employs general purpose compression methods to reduce data for efficient transfers.

The following codes are assigned for transfer modes:

S Stream
B Block
C Compressed
Z Deflate

The default transfer mode remains Stream.

3.1 Client-server Interaction

The user-FTP process sends the MODE Z command to request compressed

data transfers. If the server-FTP process accepts the request, then deflate transmission mode will be used for all data transfers until the client switches to another mode.

3.2 Overview

In deflate transmission mode, data is compressed and transmitted as a stream of octets (8 bit bytes). The sender and receiver rely on a compression engine to perform compression operations (deflate/inflate) and maintain state. There is no restriction on the representation type used; record structures are allowed.

Since there is no fixed compression format, both FTP hosts MUST process data until the compression engine reports an end-of-file (EOF) state or data error. Closing the data connection is not a sufficient method to end transfers because there may be pending information.

If an FTP process encounters an error while compressing or decompressing the data stream, it SHOULD discard all information after that point and cancel the transfer using the procedures described in STD 9 [1].

3.3 Compression Engine

Each compression engine generates a unique data stream that MAY consist of the following parts: header and control information, compressed data, integrity checkpoints and end-of-file (EOF) markers. The compression engine MUST support an EOF mechanism and MUST NOT send non-essential structures like version headers.

3.3.1 ZLIB Compression Engine

All FTP processes that support deflate transmission mode MUST support the ZLIB compressed data format specified in RFC 1950 [5].

The ZLIB compression method, an LZ77 variant called deflation, provides a lossless compressed data format that:

* is independent of CPU type, operating system, file system and character set and is therefore ideal for network communications

* provides a number of different compression settings (ratios are in the order of 2:1 to 5:1) that accommodates a wide range of CPU, memory and data requirements

* minimizes control data overhead (approximately 0.02% for large data streams)

* provides integrity checks

* can be implemented readily in a manner not covered by patents, and hence can be practiced freely

In the worst case, ZLIB reverts to stored (uncompressed) blocks making the deflate data stream analogous to STREAM transmission mode.

By default, compliant FTP processes MUST support compression method 8 and transmit the CMF, FLG and ADLER32 information in the data stream. If bandwidth or processing requirements are a concern, these restrictions can be negotiated with the OPTS command.

3.4 Syntax

The deflate extension modifies the MODE command by adding the parameter "Z":

Request:

```
mode              = "MODE" SP "Z" CRLF
```

Response:

```
mode-response     = mode-good / mode-bad
mode-good         = "200" SP response-message CRLF
mode-bad          = "451" SP response-message CRLF /
                  = "501" SP response-message CRLF
response-message  = *TCHAR
```

A "mode-good" response (200 reply) MUST be sent when the "Z" parameter is recognized and the current compression settings are appropriate. An "mode-bad" response is sent in other cases. The 451 reply should be used when the current compression settings or

some other temporary condition at the server prevent the command from being accepted; but a changed environment for the server-FTP process may permit the command to succeed. A 501 reply is appropriate for a permanent error.

3.5 FEAT Response

If the server-FTP process supports the feature (FEAT) command specified in RFC 2389 [4], then it MUST include a "MODE Z" feature line. This string indicates required support for the extension and lists the names of additional compression engines:

```
mode-feat   = SP "MODE" SP "Z" [SP eng-list] CRLF
eng-list    = *(eng-desc ",")
eng-desc    = 1*(eng-name eng-opts)
eng-name    = ALPHA*(ALPHA / DIGIT / "-" / ".")
eng-opts    = *("(" opts-list ")")
opts-list   = 1*(opt-name ",")
opt-name    = ALPHA*(ALPHA / DIGIT)
```

If eng-list is not present, then the server-FTP process is informing the client that ZLIB is the only engine available.

The "MODE Z" feature line string is not case sensitive, but SHOULD be transmitted in upper case.

In the case where the server-FTP process does not support the FEAT command, the user-FTP can negotiate the deflate extension by sending the "MODE Z" request. The server would respond with a positive (200) reply and the default compression settings would be effective.

3.5.1 FEAT Examples

The following examples contrast three servers with deflate support. The first server advertises ZLIB and BZIP2 capabilities, while the second indicates the required ZLIB engine and the third implements ZLIB and a proprietary compression engine.

```
C> FEAT
S> 211-Extensions supported:
S>  =85
S>  MODE Z BZIP2(LEVEL,BLOCKSIZE)
S>  =85
S> 211 End.
```

and

```
C> FEAT
S> 211-Extensions supported:
```

```
S>  =85
S>  MODE Z
S>  =85
S> 211 End.
```

and

```
C> FEAT
S> 211-Extensions supported:
S> =85
S>  MODE Z ENG(SETTING1,SETTING2)
S> =85
S> 211 End.
```

The ellipses indicate place holders where other features may be included, and are not required. A one space indentation of the feature line is mandatory [4].

3.6 OPTS Features

The user-FTP process may specify alternate compression settings with the OPTS command [4]. All subsequent transfers will use these settings until another OPTS request is sent. The format is specified by:

```
opts  = "OPTS" SP "MODE" SP "Z" [ SP 1*(name SP value ",") ]
name  = ALPHA*(ALPHA / DIGIT)
value = 1*RCHAR
```

When the client sends an "OPTS MODE Z" command, the server will examine each opt-name and opt-value pair and update the compression engine. An OPTS request with no parameters will cause the server-FTP process to revert to the default compression settings outlined in this document. If the server encounters an invalid or unsupported opt-name or opt-value the OPTS request will be rejected.

Note the server-FTP process MUST reject any MODE Z requests during data transfers.

If the server-FTP process accepts an OPTS request, it MUST respond with a positive (200) reply. Otherwise, a negative (501) response should be sent.

The deflate extension does not require the server to support all of the opt-name and opt-value parameters defined in this documented. If the server-FTP process encounters an invalid or unsupported option, it SHOULD return the opt-name in the error reply. In the worst case, where all OPTS requests are rejected, the FTP processes revert to the default compression settings.

If new opt-name parameters are required, the server-FTP process MUST include the labels in the feature string enclosed in brackets and separated by commas. For example, "MODE Z ENG(SETTING1,SETTING2)." The apparatus defined in this specification should be able to handle any routine compression setting.

3.6.1 Standard Opt-names

This document defines a standard set of opt-names as follows: ENGINE, METHOD, LEVEL, EXTRA and BLOCKSIZE. Each opt-name is case insensitive, or in other words, "ENGINE" is the same as "Engine" and "engine".

3.6.1.1 ENGINE Syntax

The syntax of the ENGINE option follows:

```
eng-option  = eng-label SP eng-value
eng-label   = "ENGINE"
eng-value   = ALPHA*(ALPHA / DIGIT / "-" / ".")
```

3.6.1.2 METHOD Syntax

The METHOD option allows the FTP processes to negotiate the compression method. The syntax of the METHOD option follows:

```
mth-option  = mth-label SP mth-value
mth-label   = "METHOD"
mth-value   = 1*DIGIT
```

3.6.1.3 LEVEL Syntax

The LEVEL option allows the FTP processes to negotiate the

compression level. It will influence the processing requirements and length of the compressed stream. The syntax of the LEVEL option follows:

```
lvl-option  = lvl-label SP lvl-value
lvl-label   = "LEVEL"
lvl-value   = 1*DIGIT
```

3.6.1.4 EXTRA Syntax

The EXTRA option allows the FTP processes to negotiate the transmission of non-essential compression information (like version headers and trailers). This option does not apply to all compression

engines. The syntax of the EXTRA option follows:

```
ext-option  = ext-label SP ext-value
ext-label   = "EXTRA"
ext-value   = ext-enable / ext-disable
ext-enable  = "ON"
ext-disable = "OFF"
```

3.6.1.5 BLOCKSIZE Syntax

The BLOCKSIZE option is for compression engines that use block sorting algorithms. It influences the compression ratio and processing requirements. The syntax of the BLOCKSIZE option follows:

```
blk-option  = blk-label SP blk-value
blk-label   = "BLOCKSIZE"
blk-value   = 1*DIGIT
```

The blk-value is specified in octets.

3.6.2 OPTS Examples

The following examples illustrate how a client would change the ZLIB compression options and configure a new compression engine.

```
C> OPTS MODE Z LEVEL 9
```

```
S> 200 MODE Z LEVEL set to 9.
C> OPTS MODE Z BLOCKSIZE 8192
S> 501 MODE Z BLOCKSIZE is not available.
C> OPTS MODE Z METHOD 9
S> 501 MODE Z METHOD 9 is invalid.
C> OPTS MODE Z ENGINE ZLIB LEVEL 9 EXTRA OFF
S> 200- MODE Z ENGINE set to ZLIB.
S> 200- MODE Z LEVEL set to 9.
S> 200  MODE Z EXTRA set to OFF.
C> OPTS MODE Z ENGINE ZLIB LEVEL 9 METHOD 15
S> 501 MODE Z METHOD 15 is invalid.
```

and

```
C> OPTS MODE Z ENGINE BZIP2 BLOCKSIZE 8192
S> 200- MODE Z ENGINE set to BZIP2.
S> 200  MODE Z BLOCKSIZE set to 8192.
C> OPTS MODE Z ENGINE ZLIB BLOCKSIZE 8192
S> 501 MODE Z BLOCKSIZE is not available.
```

3.7 Error Recovery and Restart

In deflate transmission mode, it is not possible to insert restart markers into the data stream because they would be indistinguishable from compressed data, and the user-FTP can alter the data representation by changing compression settings between transfers. However, it is possible to define a restart mechanism by specifying a byte offset into the uncompressed data stream.

The logic for this system is similar to the restart mechanisms specified in RFC 1123 [7] and other Internet Drafts [8] for STREAM transmission mode with the following addendum. In a compressed data stream, the output will always be exactly the same as the input, thus an offset will always represent the same position within a file.

If the user-FTP process plans to restart a retrieve (RETR) request, it will directly calculate the restart marker, and send the uncompressed offset in the restart (REST) command. The server will skip to the specified file position. When the transfer continues, both FTP processes will operate with a new compressed data stream.

The store (STOR) process works in the same manner as the retrieve system. However, the sender must determine how much data was previously received and expanded, with the SIZE [8] command or an alternate method, before resuming the transfer.

4. Security Considerations

The deflate extension does not introduce any protocol related security issues. However, some compression settings may impose a considerable load on the FTP server, which could lead to denial of service attacks, and compression engines not described in this document may contain security vulnerabilities. If these operational risks are a concern, then implementers should consider limiting server resources or denying problematic settings.

In some cases, deflate transmission mode can reduce the demands on the server. For example, in a secure FTP session, the combined process of compressing and encrypting data is less expensive than sending raw encrypted data; and fewer secure renegotiations are required because of the shorter transfer times.

A general discussion of issues related to the security of FTP can be found in RFC 2577 [9].

5. References

5.1. Normative References

[1] Postel, J., Reynolds, J., "File Transfer Protocol (FTP)", STD 9, RFC 959, October 1985

[2] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997

[3] Crocker, D., Overell, P., "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997

[4] Hethmon, P., Elz, R., "Feature negotiation mechanism for the File Transfer Protocol", RFC 2389, August 1998

[5] Deutsch, P., "ZLIB Compressed Data Format Specification version

3.3", RFC 1950, May 1996

[6] Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, May 1996

[7] Braden, R., "Requirements for Internet Hosts -- Application and Support", RFC 1123, October 1989

[8] P., Elz, R., Hethmon, "Extensions to FTP", Internet Draft (draft-ietf-ftpext-mlst-16), September 2002

[9] Allman, M., Ostermann, S., "FTP Security Considerations", RFC 2577, May 1999

5.2. Informative References

[RFC5797] Klensin, J. and A. Hoenes, "FTP Command and Extension Registry", RFC 5797, March 2010.

6. Authors' Addresses

Jeff Preston
NSC
23 Fielding Drive
Aurora, Ontario. L4G 4Z4

EMail: jpreston@nsctech.com

TJ Saunders
23525 24th Ave W
Brier, WA 98036

EMail: tj@castaglia.org