Independent Submission
Internet-Draft
Intended status: Standards Track
Expires: December 20, 2016

K. Murchison
Carnegie Mellon University
J. Elie

June 18, 2016

Network News Transfer Protocol (NNTP) Extension for Compression
draft-murchison-nntp-compress-04

Abstract

This document defines an extension to the Network News Transport Protocol (NNTP) that allows a connection to be effectively and efficiently compressed between an NNTP client and server.

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 20, 2016.

Table of Contents

1. Introduction

The goal of COMPRESS is to reduce the bandwidth usage of NNTP.

Compared to PPP compression [RFC1962] and modem-based compression ([MNP] and [V42bis]), COMPRESS offers greater compression efficiency. COMPRESS can be used together with Transport Layer Security (TLS) [RFC5246], Simple Authentication and Security Layer (SASL) encryption [RFC4422], Virtual Private Networks (VPNs), etc.

The point of COMPRESS as an NNTP extension is to act as a compression layer, similarly to a security layer like the one negotiated by STARTTLS [RFC4642]. Compression can therefore benefit to all NNTP commands sent or received after the use of COMPRESS. This facility responds to a long-standing need for NNTP to compress data, that has partially been addressed by unstandardized commands like XZVER, XZHDR, XFEATURE COMPRESS, or MODE COMPRESS. These commands are not wholly satisfactory because they enable compression only for the responses sent by the news server. On the contrary, the COMPRESS command permits to compress data sent by both the client and the server, and removes the constraint of having to implement compression separately in each NNTP command. Besides, the compression level can be dynamically adjusted and optimized at any time during the connection, which even allows to disable compression for certain commands, if need be. If the news client wants to stop compression on a particular connection, it can simply use QUIT ([RFC3977] Section 5.4), and establish a new connection. For these reasons, using other NNTP commands than COMPRESS to enable compression is discouraged once COMPRESS is supported.

In order to increase interoperability, it is desirable to have as few different compression algorithms as possible, so this document specifies only one. The DEFLATE algorithm (defined in [RFC1951]) MUST be implemented as part of this extension. This compression algorithm is standard, widely available, and fairly efficient.

This specification should be read in conjunction with the NNTP base specification [RFC3977]. In the case of a conflict between these two documents, [RFC3977] takes precedence.

1.1. About TLS-level Compression

Though data compression is made possible via the use of TLS with NNTP [RFC4642], the best current practice is to disable TLS-level compression as explained in Section 3.3 of [RFC7525]. The COMPRESS command will permit to keep the compression facility in NNTP and control when it is available during a connection.

Compared to TLS-level compression [RFC3749], NNTP COMPRESS has the following advantages:

- COMPRESS can be implemented easily both by NNTP servers and clients.

- COMPRESS benefits from an intimate knowledge of the NNTP protocol's state machine, allowing for dynamic and aggressive optimization of the underlying compression algorithm's parameters.

- o COMPRESS can be activated after authentication has completed thus reducing the chances that authentication credentials can be leaked via for instance a CRIME attack ([RFC7457] Section 2.6).

1.2. Conventions Used in This Document

The notational conventions used in this document are the same as those in [RFC3977], and any term not defined in this document has the same meaning as it does in that one.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

In the examples, commands from the client are indicated with [C], and responses from the server are indicated with [S]. The client is the initiator of the NNTP connection; the server is the other endpoint.

1.3. Authors' Note

Please write the first letter of "Elie" and the penultimate letter of "allee" with an acute accent wherever possible -- they are respectively U+00C9 ("É" in XML) and U+00E9 ("é" in XML). Also, the letters "ae" in "Baeuerle" should be written as an a-umlaut (U+00E4, "ä" in XML), and the first letter of "Angel" as well as the fifth letter of "Gonzalez" should be written with an acute accent (respectively U+00C1 and U+00E1, that is to say "Á" and "á" in XML).

2. The COMPRESS Extension

The COMPRESS extension is used to enable data compression on an NNTP connection.

This extension provides a new COMPRESS command and has capability label COMPRESS.

2.1. Advertising the COMPRESS Extension

A server supporting the COMPRESS command as defined in this document will advertise the "COMPRESS" capability label in response to the CAPABILITIES command ([RFC3977] Section 5.2). However, this capability MUST NOT be advertised once a compression layer is active (see Section 2.2.2). This capability MAY be advertised both before and after any use of the MODE READER command ([RFC3977] Section 5.3), with the same semantics.

The COMPRESS capability label contains a whitespace-separated list of available compression algorithms. This document defines one compression algorithm: DEFLATE. This algorithm is mandatory to implement and MUST be supported in order to advertise the COMPRESS extension.

Future extensions may add additional compression algorithms to this capability. Unrecognized algorithms MUST be ignored by the client.

As the COMPRESS command is related to security because it can weaken encryption, cached results of CAPABILITIES from a previous session MUST NOT be relied on, as per Section 12.6 of [RFC3977].

Example:

```
[C] CAPABILITIES
[S] 101 Capability list:
[S] VERSION 2
[S] READER
[S] IHAVE
[S] COMPRESS DEFLATE X-SHRINK
[S] LIST ACTIVE NEWSGROUPS
[S] .
```

2.2. COMPRESS Command

2.2.1. Usage

This command MUST NOT be pipelined.

Syntax
  COMPRESS algorithm

Responses
  206 Compression active
  403 Unable to activate compression
  502 Command unavailable [1]

[1] If a compression layer is already active, COMPRESS is not a valid command (see Section 2.2.2).

Parameters
  algorithm = Name of compression algorithm (e.g. "DEFLATE")

2.2.2. Description

The COMPRESS command instructs the server to use the named compression algorithm ("DEFLATE" is the only one defined in this document) for all commands and/or responses after COMPRESS.

The client MUST NOT send any further commands until it has seen the result of COMPRESS.

If the requested compression algorithm is syntactically incorrect, the server MUST reject the COMPRESS command with a 501 response code ([RFC3977] Section 3.2.1). If the requested compression algorithm is invalid (e.g., is not supported), the server MUST reject the COMPRESS command with a 503 response code ([RFC3977] Section 3.2.1). If the server is unable to activate compression for any reason (e.g., a server configuration or resource problem), the server MUST reject the COMPRESS command with a 403 response code ([RFC3977] Section 3.2.1). Otherwise, the server issues a 206 response code and the compression layer takes effect for both client and server immediately following the CRLF of the success reply.

Additionally, the client MUST NOT issue a MODE READER command after activating a compression layer, and a server MUST NOT advertise the MODE-READER capability.

Both the client and the server MUST know if there is a compression layer active (for instance via the previous use of the COMPRESS command or the negotiation of a TLS-level compression [RFC3749]). A client MUST NOT attempt to activate compression or negotiate a TLS layer (for instance via the use of the COMPRESS or STARTTLS [RFC4642] commands) if a compression layer is already active. A server MUST NOT return the COMPRESS or STARTTLS capability labels in response to a CAPABILITIES command received after a compression layer is active, and a server MUST reply with a 502 response code if a syntactically valid COMPRESS or STARTTLS command is received while a compression layer is already active.

In order to help mitigate leaking authentication credentials via for instance a CRIME attack [CRIME], authentication SHOULD NOT be attempted when a compression layer is active. Consequently, a server SHOULD NOT return any arguments with the AUTHINFO capability label (or SHOULD NOT advertise it at all) in response to a CAPABILITIES command received from an unauthenticated client after a compression layer is active, and such a client SHOULD NOT attempt to utilize any AUTHINFO [RFC4643] commands. It implies that a server SHOULD reply with a 502 response code if a syntactically valid AUTHINFO command is received while a compression layer is already active.

For DEFLATE [RFC1951] (as for many other compression algorithms), the compressor can trade speed against quality. The decompressor MUST automatically adjust to the parameters selected by the sender. Consequently, the client and server are both free to pick the best reasonable rate of compression for the data they send.

When COMPRESS is combined with TLS [RFC5246] or SASL [RFC4422] security layers, the processing order of the three layers MUST be first COMPRESS, then SASL, and finally TLS. That is, before data is transmitted, it is first compressed. Second, if a SASL security layer has been negotiated, the compressed data is then signed and/or encrypted accordingly. Third, if a TLS security layer has been negotiated, the data from the previous step is signed and/or encrypted accordingly. When receiving data, the processing order MUST be reversed. This ensures that before sending, data is compressed before it is encrypted, independent of the order in which the client issues the COMPRESS, AUTHINFO SASL [RFC4643], and STARTTLS [RFC4642] commands.

When compression is active and either the client or the server receives invalid or corrupted compressed data, the receiving end SHOULD immediately close the connection. (Then the sending end will in turn do the same.)

2.2.3. Examples

Example of layering TLS and NNTP compression:

```
[C] CAPABILITIES
[S] 101 Capability list:
[S] VERSION 2
[S] READER
[S] STARTTLS
[S] AUTHINFO
[S] COMPRESS DEFLATE
[S] LIST ACTIVE NEWSGROUPS
[S] .
[C] STARTTLS
[S] 382 Continue with TLS negotiation
[TLS negotiation without compression occurs here]
[Following successful negotiation, all traffic is encrypted]
[C] CAPABILITIES
[S] 101 Capability list:
[S] VERSION 2
[S] READER
[S] AUTHINFO USER
[S] COMPRESS DEFLATE
[S] LIST ACTIVE NEWSGROUPS
[S] .
[C] AUTHINFO USER fred
[S] 381 Enter passphrase
[C] AUTHINFO PASS flintstone
[S] 281 Authentication accepted
[C] CAPABILITIES
[S] 101 Capability list:
[S] VERSION 2
[S] READER
[S] POST
[S] COMPRESS DEFLATE
[S] LIST ACTIVE NEWSGROUPS
[S] .
[C] COMPRESS DEFLATE
[S] 206 Compression active
[Henceforth, all traffic is compressed before being encrypted]
```

Example of a server failing to activate compression:

```
[C] CAPABILITIES
[S] 101 Capability list:
[S] VERSION 2
[S] IHAVE
[S] COMPRESS DEFLATE
[S] .
[C] COMPRESS DEFLATE
[S] 403 Unable to activate compression
```

Example of attempting to use an unsupported compression algorithm:

```
[C] CAPABILITIES
[S] 101 Capability list:
[S] VERSION 2
[S] IHAVE
[S] COMPRESS DEFLATE
[S] .
[C] COMPRESS X-SHRINK
[S] 503 Compression algorithm not supported
```

Example of a server refusing to compress twice:

```
[C] CAPABILITIES
[S] 101 Capability list:
[S] VERSION 2
[S] IHAVE
[S] STARTTLS
[S] COMPRESS DEFLATE
[S] .
[C] STARTTLS
[S] 382 Continue with TLS negotiation
[TLS negotiation with compression occurs here]
[Following successful negotiation, all traffic is encrypted]
[C] CAPABILITIES
[S] 101 Capability list:
[S] VERSION 2
[S] IHAVE
[S] .
[C] COMPRESS DEFLATE
[S] 502 Compression already active via TLS
```

Example of a server refusing to negotiate a TLS layer after compression has been activated:

```
[C] CAPABILITIES
[S] 101 Capability list:
[S] VERSION 2
[S] IHAVE
[S] STARTTLS
[S] COMPRESS DEFLATE
[S] .
[C] COMPRESS DEFLATE
[S] 206 Compression active
[Henceforth, all traffic is compressed]
[C] CAPABILITIES
[S] 101 Capability list:
[S] VERSION 2
[S] IHAVE
[S] .
[C] STARTTLS
[S] 502 DEFLATE compression already active
```

Example of a server not advertising AUTHINFO arguments after compression has been activated:

```
[C] CAPABILITIES
[S] 101 Capability list:
[S] VERSION 2
[S] READER
[S] AUTHINFO USER
[S] COMPRESS DEFLATE
[S] LIST ACTIVE NEWSGROUPS
[S] .
[C] COMPRESS DEFLATE
[S] 206 Compression active
[Henceforth, all traffic is compressed]
[C] CAPABILITIES
[S] 101 Capability list:
[S] VERSION 2
[S] READER
[S] AUTHINFO
[S] LIST ACTIVE NEWSGROUPS
[S] .
[C] AUTHINFO USER fred
[S] 502 DEFLATE compression already active
```

3. Compression Efficiency

This section is informative, not normative.

NNTP poses some unusual problems for a compression layer.

Upstream traffic is fairly simple. Most NNTP clients send the same few commands again and again, so any compression algorithm that can exploit repetition works efficiently. The article posting and transfer commands (e.g., POST, IHAVE, and TAKETHIS [RFC4644]) are exceptions; clients that send many article posting or transfer commands may want to surround large multi-line data blocks with a dictionary flush and/or, depending on the compression algorithm, a change of compression level in the same way as is recommended for servers later in this document (Section 3.1).

Downstream traffic has the unusual property that several kinds of data are sent, possibly confusing a dictionary-based compression algorithm.

One type is NNTP simple responses and NNTP multi-line responses not related to article header/body retrieval (e.g, CAPABILITIES, GROUP, LISTGROUP, LAST, NEXT, STAT, DATE, NEWNEWS, NEWGROUPS, LIST, CHECK [RFC4644], etc). These are highly compressible; zlib using its least CPU-intensive setting compresses typical responses to 25-40% of their original size.

Another type is article headers (as retrieved via the HEAD, HDR, OVER, or ARTICLE commands). These are equally compressible, and benefit from using the same dictionary as the NNTP responses.

A third type is article body text (as retrieved via the BODY or ARTICLE commands). Text is usually fairly short and includes much ASCII, so the same compression dictionary will do a good job here, too. When multiple messages in the same thread are read at the same time, quoted lines, etc. can often be compressed almost to zero.

Finally, non-text article bodies or attachments (as retrieved via the BODY and ARTICLE commands) are transmitted in encoded form, usually Base64 [RFC4648], UUencode [IEEE.1003-2.1992], or yEnc [yEnc].

When already compressed articles or attachments are retrieved, a compression algorithm may be able to compress them, but the format of their encoding is usually not NNTP-like, so the dictionary built while compressing NNTP does not help much. The compressor has to adapt its dictionary from NNTP to the attachment's encoding format, and then back.

When attachments are retrieved in Base64 or UUencode form, the Huffman coding usually compresses those to approximatively only 75% of their encoding size. 8-bit compression algorithms such as DEFLATE work well on 8-bit file formats; however, both Base64 and UUencode transform a file into something resembling 6-bit bytes, hiding most of the 8-bit file format from the compressor.

On the other end, attachments encoded using a compression algorithm that retains the full 8-bit spectrum, like yEnc, are much more likely to be incompressible.

3.1. DEFLATE Specificities

When using the zlib library (see [RFC1951]), the functions deflateInit2(), deflate(), inflateInit2(), and inflate() suffice to implement this extension. The windowBits value must be in the range -8 to -15 for deflateInit2(), or else it will use the wrong format. The windowBits value should be -15 for inflateInit2(), or else it will not be able to decompress a stream with a larger window size. deflateParams() can be used to improve compression rate and resource use. In order to improve compression efficiency, the Z_PARTIAL_FLUSH argument to deflate() should always be used to flush data. As far as DEFLATE is concerned, clearing the dictionary never improves compression over the other flushes. On the contrary, having the 32kB dictionary from previous data, no matter how unrelated, can only help. If there are no matching strings in there, then it is simply not referenced. Using Z_FULL_FLUSH clears the dictionary, and consequently always results in compression that is less effective than a Z_PARTIAL_FLUSH.

A server can improve downstream compression and the CPU efficiency both of the server and the client if it adjusts the compression level (e.g., using the deflateParams() function in zlib) at the start and end of large non-text multi-line data blocks (before and after 'content-lines' in the definition of 'multi-line-data-block' in [RFC3977] Section 9.8). It permits to avoid trying to compress incompressible attachments. Small multi-line data blocks are best left alone. A possible boundary is 5kB.

A very simple strategy is to change the compression level to 0 at the start of a multi-line data block provided the first two bytes are either 0x1F 0x8B (as in deflate-compressed files) or 0xFF 0xD8 (JPEG), and to keep it at 1-5 the rest of the time. More complex strategies are of course possible, and encouraged.

4. Augmented BNF Syntax for the COMPRESS Extension

This section describes the formal syntax of the COMPRESS extension using ABNF [RFC5234]. It extends the syntax in Section 9 of [RFC3977], and non-terminals not defined in this document are defined there. The [RFC3977] ABNF should be imported first, before attempting to validate these rules.

4.1. Commands

This syntax extends the non-terminal <command>, which represents an NNTP command.

```
command =/ compress-command

compress-command = "COMPRESS" WS algorithm
```

4.2. Capability Entries

This syntax extends the non-terminal <capability-entry>, which represents a capability that may be advertised by the server.

```
capability-entry =/ compress-capability

compress-capability = "COMPRESS" 1*(WS algorithm)
```

4.3. General Non-terminals

```
algorithm = "DEFLATE" / 1*20alg-char
alg-char = UPPER / DIGIT / "-" / "_"
```

5. Summary of Response Codes

This section contains a list of each new response code defined in this document and indicates whether it is multi-line, which commands can generate it, what arguments it has, and what its meaning is.

Response code 206
   Generated by: COMPRESS
   Meaning: compression layer activated

6. Security Considerations

Security issues are discussed throughout this document.

In general, the security considerations of the NNTP core specification ([RFC3977] Section 12) and the DEFLATE compressed data format specification ([RFC1951] Section 6) are applicable here.

Implementers should be aware that combining compression with encryption like TLS can sometimes reveal information that would not have been revealed without compression, as explained in Section 6 of [RFC3749]. As a matter of fact, adversaries that observe the length of the compressed data might be able to derive information about the corresponding uncompressed data. The CRIME and the BREACH attacks ([RFC7457] Section 2.6) are examples of such case.

In order to help mitigate leaking authentication credentials, this document states in Section 2.2.2 that authentication SHOULD NOT be attempted when a compression layer is active. Therefore, when a client wants to authenticate, compress data, and negotiate a TLS layer (without TLS-level compression) in the same NNTP connection, it SHOULD use the STARTTLS, AUTHINFO, and COMPRESS commands in that order. Of course instead of using the STARTTLS command, a client can also use implicit TLS, that is to say it begins the TLS negotiation immediately upon connection on a separate port dedicated to NNTP over TLS.

NNTP commands other than AUTHINFO are not believed to divulgate confidential information as long as only public Netnews newsgroups and articles are accessed. That is why this specification only adds a restriction to the use of AUTHINFO when a compression layer is active. In case confidential articles are accessed in private newsgroups, special care is needed: implementations SHOULD NOT compress confidential data together with public data when a security layer is active, for the same reasons as mentioned above in this Section.

Additionally, implementations MAY NOT compress together the contents of two distinct confidential articles. This can be achieved for instance with DEFLATE by clearing the compression dictionary before sending any article in these confidential newsgroups. More complex strategies are of course possible, and encouraged.

Implementations SHOULD use a default configuration with disabled compression when a security layer is active, and MUST support an option to allow compression to be enabled when a security layer is active. Such an option can be either with global scope or server/connection based. Implementations MAY unconditionally allow compression to be enabled when no security layer is active.

Future extensions to NNTP that define commands conveying confidential data SHOULD ensure to state that these confidential data SHOULD NOT be compressed together with public data when a security layer is active.

7. IANA Considerations

7.1. NNTP Compression Algorithm Registry

The NNTP Compression Algorithm registry will be maintained by IANA. The registry will be available at <http://www.iana.org/assignments/nntp-compression-algorithms>.

The purpose of this registry is not only to ensure uniqueness of values used to name NNTP compression algorithms, but also to provide a definitive reference to technical specifications detailing each NNTP compression algorithm available for use on the Internet.

An NNTP compression algorithm is either a private algorithm, or its name is included in the IANA NNTP Compression Algorithm registry (in which case it is a "registered NNTP compression algorithm"). Different entries in the registry MUST use different names.

Any name that conforms to the syntax of an NNTP compression algorithm name (Section 4.3) can be used. However, the name of a registered NNTP compression algorithm MUST NOT begin with "X".

To avoid the risk of a clash with a future registered NNTP compression algorithm, the names of private compression algorithms SHOULD begin with "X-".

If the server advertises a registered NNTP compression algorithm, it MUST implement the compression algorithm so as to fully conform with its related specification. If it does not implement the compression algorithm as specified, it MUST NOT list its registered name in the arguments list of the COMPRESS capability label. In that case, it MAY, but SHOULD NOT, provide a private algorithm (not listed, or listed with a different name) that implements the compression algorithm differently.

A server MUST NOT send different response codes to COMPRESS in response to the availability or use of a private compression algorithm.

The procedure detailed in Section 7.1.1 is to be used for registration of a value naming a specific individual compression algorithm.

Comments may be included in the registry as discussed in Section 7.1.2 and may be changed as discussed in Section 7.1.3.

7.1.1. Algorithm Name Registration Procedure

IANA will register new NNTP compression algorithm names on a First Come First Served basis, as defined in BCP 26 [RFC5226]. IANA has the right to reject obviously bogus registration requests, but will perform no review of claims made in the registration form.

Registration of an NNTP compression algorithm is requested by filling in the following template and sending it via electronic mail to IANA at <iana@iana.org>:

Subject: Registration of NNTP compression algorithm X

NNTP compression algorithm name:

Security considerations:

Published specification (recommended):

Contact for further information:

Intended usage: (One of COMMON, LIMITED USE, or OBSOLETE)

Owner/Change controller:

Note: (Any other information that the author deems relevant may be added here.)

While this registration procedure does not require expert review, authors of NNTP compression algorithms are encouraged to seek community review and comment whenever that is feasible. Authors may seek community review by posting a specification of their proposed algorithm as an Internet-Draft. NNTP compression algorithms intended for widespread use should be standardized through the normal IETF process, when appropriate.

7.1.2. Comments on Algorithm Registrations

Comments on a registered NNTP compression algorithm should first be sent to the "owner" of the algorithm and/or to the mailing list for the now concluded IETF NNTPEXT working group (<ietf-nntp@lists.eyrie.org>).

Submitters of comments may, after a reasonable attempt to contact the owner and/or the above mailing list, request IANA to attach their comment to the NNTP compression algorithm registration itself by sending mail to <iana@iana.org>. At IANA's sole discretion, IANA may attach the comment to the NNTP compression algorithm's registration.

7.1.3. Change Control

Once an NNTP compression algorithm registration has been published by IANA, the owner may request a change to its definition. The change request follows the same procedure as the initial registration request.

The owner of an NNTP compression algorithm may pass responsibility for the algorithm to another person or agency by informing IANA; this can be done without discussion or review.

The IESG may reassign responsibility for an NNTP compression algorithm. The most common case of this will be to enable changes to be made to algorithms where the owner of the registration has died, has moved out of contact, or is otherwise unable to make changes that are important to the community.

NNTP compression algorithm registrations MUST NOT be deleted; algorithms that are no longer believed appropriate for use can be declared OBSOLETE by a change to their "intended usage" field; such algorithms will be clearly marked in the registry published by IANA.

The IESG is considered to be the owner of all NNTP compression algorithms that are on the IETF standards track.

7.2. Registration of the DEFLATE Compression Algorithm

This section gives a formal definition of the DEFLATE compression algorithm as required by Section 7.1.1 for the IANA registry.

NNTP compression algorithm name: DEFLATE

Security considerations: See Section 6 of this document

Published specification: This document

Contact for further information: Authors of this document

Intended usage: COMMON

Owner/Change controller: IESG <iesg@ietf.org>

Note: This algorithm is mandatory to implement

7.3. Registration of the NNTP COMPRESS Extension

This section gives a formal definition of the COMPRESS extension as required by Section 3.3.3 of [RFC3977] for the IANA registry.

- The COMPRESS extension allows an NNTP connection to be effectively and efficiently compressed.
- The capability label for this extension is "COMPRESS", whose arguments list the available compression algorithms.
- This extension defines one new command, COMPRESS, whose behavior, arguments, and responses are defined in Section 2.2.

- This extension does not associate any new responses with pre-existing NNTP commands.
- This extension does affect the overall behavior of both server and client, in that after successful use of the COMPRESS command, all communication is transmitted in a compressed format.
- This extension does not affect the maximum length of commands or initial response lines.
- This extension does not alter pipelining, but the COMPRESS command cannot be pipelined.
- Use of this extension does alter the capabilities list; once the COMPRESS command has been used successfully, the COMPRESS capability can no longer be advertised by CAPABILITIES. Additionally, the STARTTLS and MODE-READER capabilities MUST NOT be advertised, and the AUTHINFO capability label SHOULD either return no arguments or no longer be advertised after successful execution of the COMPRESS command.
- This extension does not cause any pre-existing command to produce a 401, 480, or 483 response code.
- This extension is unaffected by any use of the MODE READER command; however, the MODE READER command MUST NOT be used in the same session following a successful execution of the COMPRESS command.
- The STARTTLS command MUST NOT be used, and the AUTHINFO command SHOULD NOT be used, in the same session following a successful execution of the COMPRESS command.
- Published Specification: This document.
- Contact for Further Information: Authors of this document.
- Change Controller: IESG <iesg@ietf.org>.

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3977] Feather, C., "Network News Transfer Protocol (NNTP)", RFC 3977, DOI 10.17487/RFC3977, October 2006, <http://www.rfc-editor.org/info/rfc3977>.

[RFC4642] Murchison, K., Vinocur, J., and C. Newman, "Using Transport Layer Security (TLS) with Network News Transfer Protocol (NNTP)", RFC 4642, DOI 10.17487/RFC4642, October 2006, <http://www.rfc-editor.org/info/rfc4642>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

8.2. Informative References

[CRIME] Rizzo, J. and T. Duong, "The CRIME Attack", Ekoparty Security Conference, 2012.

[IEEE.1003-2.1992] Institute of Electrical and Electronics Engineers, "Information Technology - Portable Operating System Interface (POSIX) - Part 2: Shell and Utilities (Vol. 1)", IEEE Standard 1003.2, 1992.

[MNP] Held, G., "The Complete Modem Reference", Second Edition, Wiley Professional Computing, May 1994.

[RFC1951] Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, DOI 10.17487/RFC1951, May 1996, <http://www.rfc-editor.org/info/rfc1951>.

[RFC1962] Rand, D., "The PPP Compression Control Protocol (CCP)", RFC 1962, DOI 10.17487/RFC1962, June 1996, <http://www.rfc-editor.org/info/rfc1962>.

[RFC3749] Hollenbeck, S., "Transport Layer Security Protocol Compression Methods", RFC 3749, DOI 10.17487/RFC3749, May 2004, <http://www.rfc-editor.org/info/rfc3749>.

[RFC4422] Melnikov, A., Ed. and K. Zeilenga, Ed., "Simple Authentication and Security Layer (SASL)", RFC 4422, DOI 10.17487/RFC4422, June 2006, <http://www.rfc-editor.org/info/rfc4422>.

[RFC4643] Vinocur, J. and K. Murchison, "Network News Transfer Protocol (NNTP) Extension for Authentication", RFC 4643, DOI 10.17487/RFC4643, October 2006, <http://www.rfc-editor.org/info/rfc4643>.

[RFC4644] Vinocur, J. and K. Murchison, "Network News Transfer Protocol (NNTP) Extension for Streaming Feeds", RFC 4644, DOI 10.17487/RFC4644, October 2006, <http://www.rfc-editor.org/info/rfc4644>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <http://www.rfc-editor.org/info/rfc4648>.

[RFC4978] Gulbrandsen, A., "The IMAP COMPRESS Extension", RFC 4978, DOI 10.17487/RFC4978, August 2007, <http://www.rfc-editor.org/info/rfc4978>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC7457] Sheffer, Y., Holz, R., and P. Saint-Andre, "Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS)", RFC 7457, DOI 10.17487/RFC7457, February 2015, <http://www.rfc-editor.org/info/rfc7457>.

[RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <http://www.rfc-editor.org/info/rfc7525>.

[V42bis] International Telecommunications Union, "Data compression procedures for data circuit-terminating equipment (DCE) using error correction procedures", ITU-T Recommendation V.42bis, January 1990.

[yEnc] Helbing, J., "yEnc - Efficient encoding for Usenet and eMail", March 2002, <http://www.yenc.org/>.

Appendix A. Acknowledgements

This document draws heavily on ideas in [RFC4978] by Arnt Gulbrandsen and a large portion of this text was borrowed from that specification.

The authors would like to thank the following individuals for contributing their ideas and support for writing this specification: Mark Adler, Russ Allbery, Michael Baeuerle, Angel Gonzalez, and Brian Peterson.

Appendix B. Document History (to be removed by RFC Editor before publication)

B.1. Changes since -03

- Added a naming convention for NNTP compression algorithms. Improve the wording of registered vs private compression algorithms.
- If a registered NNTP compression algorithm is advertised, it MUST fully conform with its related specification.
- Fixed the wording of security considerations to reflect that the threat appears when public and confidential data are compressed together inside a security layer. Thanks to Angel Gonzalez for pointing that.
- The default configuration SHOULD be disabled compression when a security layer is active.
- COMPRESS acts as a compression layer, not a transport layer.
- Minor editorial changes.

B.2. Changes since -02

- Added text stating that the receiving end SHOULD terminate the connection when receiving invalid or corrupted compressed data.
- Explained why COMPRESS permits to do better than existing unstandardized commands like XZVER, XZHDR, MODE COMPRESS, and XFEATURE GZIP.
- Added an example of AUTHINFO command when compression is active.
- The LIST capability label was missing in the examples when READER was also advertised.

- Improved an example to send CAPABILITIES after successful authentication.
- Mentioned that COMPRESS is related to security. CAPABILITIES is therefore sent again after COMPRESS.
- Re-added discussion of attachments in binary form and incompressible file formats. Improve the discussion about flushes, and add a specific section about DEFLATE.
- Changed a MUST NOT to SHOULD NOT for the use of AUTHINFO after COMPRESS.
- Algorithm names are case-insensitive.
- Mentioned the use of the 501 response code.
- Added the Security Considerations Section.
- Added Julien Elie as co-author of this document.
- Minor editorial changes.

B.3. Changes since -01

- Switched to using 206 response code when compression has been activated.
- Added text stating that TLS-level compression is susceptible to CRIME attack and current BCP is to disable it.
- Added text stating that AUTHINFO shouldn't be advertised or used after COMPRESS to prevent possible CRIME attack (with example).
- Added text stating that a windowBits value of -15 should be used for inflateInit2().
- Minor editorial changes.

B.4. Changes since -00

- Made DEFLATE the mandatory to implement compression algorithm.
- Removed the requirement that clients/servers implementing COMPRESS also implement TLS compression.
- Added an example of a client trying to use an unsupported compression algorithm.

- Rewrote Compression Efficiency (Section 3) as follows:
  - Included a sample listing of which NNTP commands produce which type of data to be compressed.
  - Removed discussion of attachments in binary form and incompressible file formats.
  - Mentioned UUencode and yEnc encoding of attachments.
- Added IANA registry of NNTP compression algorithms.
- Miscellaneous editorial changes submitted by Julien Elie.

Authors' Addresses

Kenneth Murchison
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
US

Phone: +1 412 268 1982
EMail: murch@andrew.cmu.edu

Julien Elie
10 allee Clovis
Noisy-le-Grand 93160
France

EMail: julien@trigofacile.com
URI: http://www.trigofacile.com/