Internet Engineering Task Force (IETF)
Request for Comments: 8054
Category: Standards Track
ISSN: 2070-1721
January 2017

K. Murchison
Carnegie Mellon University
J. Elie

Network News Transfer Protocol (NNTP) Extension for Compression

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 is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc8054.

Copyright Notice

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, similar to a security layer like the one negotiated by STARTTLS [RFC4642]. Therefore, compression can be beneficial 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. It is currently addressed only partially by unstandardized commands like XZVER, XZHDR, XFEATURE COMPRESS, or MODE COMPRESS. Yet, these commands are not wholly satisfactory because they enable compression only for the responses sent by the news server. In comparison, the COMPRESS command permits the compression of 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 disabling compression for certain commands, if needed. 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 lossless data compression is already 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 keeping 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:

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

- o 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", "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.

2. The COMPRESS Extension

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

This extension provides a new COMPRESS command and has the 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; it MUST be supported and listed in the advertisement of the COMPRESS extension.

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

Example:

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

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].

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 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, in case no other generic response code representing the situation applies, 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 method [RFC3749]). A client MUST NOT attempt to activate compression (for instance, via the COMPRESS command) or negotiate a TLS security layer (because STARTTLS [RFC4642] may activate TLS-level compression) 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 MUST NOT be attempted after a successful use of the COMPRESS command. Consequently, a server MUST either list the AUTHINFO capability with no arguments or not advertise it at all, in response to a CAPABILITIES command received from an unauthenticated client after a successful use of the COMPRESS command, and such a client MUST NOT attempt to utilize any AUTHINFO [RFC4643] commands. This implies that a server MUST reply with a 502 response code if a syntactically valid AUTHINFO command is received after a successful use of the COMPRESS command. (Note that this specification does not change the behavior of AUTHINFO as described in [RFC4643] independently of TLS-level compression. Authentication is therefore still allowed, even though TLS-level compression is active.)

For DEFLATE [RFC1951] (as for many other compression algorithms), the sending compressor can trade speed against compression ratio. The receiving 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. Besides, all data that was submitted for compression MUST be included in the compressed output, and appropriately flushed so as to ensure that the receiving decompressor can completely decompress it.

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 (with a possible additional TLS-level compression). When receiving data, the processing order MUST be reversed. This ensures that before sending, data is compressed before it is encrypted.

When compression is active and either the client or the server receives invalid or corrupted compressed data, the receiving end immediately closes the connection, in response to which the sending end will do the same.

2.2.3. Examples

Example of layering a TLS security layer 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]
[C] CAPABILITIES
[S] 101 Capability list:
[S] VERSION 2
[S] READER
[S] POST
[S] LIST ACTIVE NEWSGROUPS
[S] .
```

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 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 security 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 4).

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

NNTP responses that are not related to article header/body retrieval are one type. Compressing NNTP simple responses (e.g., in answer to CHECK [RFC4644], DATE, GROUP, LAST, NEXT, STAT, etc.) generally does not save many bytes, unless repeated several times in the same NNTP session. On the contrary, most of the NNTP multi-line responses (e.g., in answer to LIST, LISTGROUP, NEWGROUPS, NEWNEWS, etc.) are highly compressible; using its least CPU-intensive setting, zlib compresses typical responses to 25-40% of their original size.

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

A third type is article body text (as retrieved, for instance, 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, for instance, via the BODY or ARTICLE commands) are transmitted in encoded form, usually Base64 [RFC4648], UUencode [IEEE.1003.1-2008], or yEnc [yEnc].

When such non-text article bodies 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 approximately 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.

4. 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, thus reducing interoperability. deflateParams() can be used to improve compression rate and resource use. Regarding flush operations, the Z_FULL_FLUSH argument to deflate() permits to clear the dictionary, which generally results in compression that is less effective than performing a Z_PARTIAL_FLUSH. As a matter of fact, keeping the 32 KB dictionary from previous data, no matter how unrelated, can be of help (if there are no matching strings in there, then it is simply not referenced).

A server can improve downstream compression and the CPU efficiency of both 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). This mechanism prevents the server from trying to compress incompressible attachments.

A very simple strategy is to change the compression level to 0 at the start of an incompressible multi-line data block, for instance when encoded using yEnc [yEnc], and to keep it at 1-5 the rest of the time. More complex strategies are, of course, possible and encouraged.

5. Augmented BNF Syntax for the COMPRESS Extension

This section describes the formal syntax of the COMPRESS extension using ABNF [RFC7405] and [RFC5234]. It extends the syntax in Section 9 of [RFC3977], and non-terminals not defined in this document are defined there. The NNTP ABNF [RFC3977] should be imported first, before attempting to validate these rules.

5.1. Commands

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

```
command =/ compress-command

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

5.2. 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)
```

5.3. General Non-terminals

```
algorithm = %s"DEFLATE" / 1*20alg-char  ; case-sensitive
alg-char = UPPER / DIGIT / "-" / "_"
```

6. Summary of Response Codes

This section defines the following new response code. It is not multi-line and has no arguments.

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

7. 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 MUST NOT be attempted after a successful use of COMPRESS. Therefore, when a client wants to authenticate, compress data, and negotiate a TLS security layer (without TLS-level compression) in the same NNTP connection, it MUST 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 divulge confidential information as long as only public Netnews newsgroups and articles are accessed. That is why this specification only prohibits the use of AUTHINFO after COMPRESS. 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 TLS [RFC5246] or SASL [RFC4422] security layer is active. As a matter of fact, adversaries that observe the length of the compressed data might be able to derive information about it, when public data (that adversaries know is read) and confidential data are compressed in the same compression session.

Additionally, it is preferable not to compress the contents of two distinct confidential articles together if it can be avoided, as adversaries might be able to derive information about them (for instance, if they have a few header fields or body lines in common). This can be achieved, for instance, with DEFLATE by clearing the compression dictionary each time a confidential article is sent. More complex implementations are, of course, possible and encouraged.

Implementations are encouraged to unconditionally allow compression when no security layer is active, and to support an option to enable or disable compression when a security layer is active. Such an option could, for instance, have global scope or be server/connection-based. Besides, as compression may in general weaken the confidentiality of a security layer, implementations SHOULD NOT automatically enable compression when a security layer is active unless the user explicitly enabled it with this knowledge.

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

Last but not least, careful consideration should be given to protections against implementation errors that introduce security risks with regards to compression algorithms. See, for instance, the part of Section 6 of [RFC3749] about compression algorithms that can occasionally expand, rather than compress, input data.

8. IANA Considerations

8.1. "NNTP Compression Algorithms" Registry

The "NNTP Compression Algorithms" registry is maintained by IANA. The registry is available at <http://www.iana.org/assignments/nntp-parameters>.

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 Algorithms" registry (in which case it is a "registered NNTP compression algorithm"). Different entries in the registry MUST use different names.

Private algorithms with unregistered names are allowed, but SHOULD NOT be used because it is difficult to achieve interoperability with them.

The 206, 403, and 502 response codes that a news server answers to the COMPRESS command using a private compression algorithm MUST have the same meaning as the one documented in Section 2.2 of this document.

The procedure detailed in Section 8.1.1 is to be used for registration of a value naming a specific individual compression algorithm.

Any name that conforms to the syntax of an NNTP compression algorithm name (Section 5.3) can be used. Especially, NNTP compression algorithms are named by strings, from 1 to 20 characters in length, consisting of uppercase letters, digits, hyphens, and/or underscores.

Comments may be included in the registry as discussed in Section 8.1.2 and may be changed as discussed in Section 8.1.3.

8.1.1. 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 not perform a 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 Z

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.

8.1.2. 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 NNTPEXT working group (<ietf-nntp@lists.eyrie.org>) of the IETF.

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.

8.1.3. 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.

8.2. Registration of the DEFLATE Compression Algorithm

This section gives a formal definition of the DEFLATE compression algorithm as required by Section 8.1.1 for the IANA registry.

NNTP compression algorithm name: DEFLATE

Security considerations: See Section 7 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

This registration appears as follows in the "NNTP Compression Algorithms" registry:

Algorithm Name	Intended Usage	Comment	Change Controller	Reference
DEFLATE	COMMON	Mandatory to implement	IESG	RFC 8054

8.3. 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 MUST either be listed with no arguments or not advertised at all after a 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 and AUTHINFO commands MUST 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>

This registration will appear as follows in the "NNTP Capability Labels" registry contained in the "Network News Transfer Protocol (NNTP) Parameters" registry:

Label	Meaning	Reference
COMPRESS	Supported compression algorithms	RFC 8054

9. References

9.1. Normative References

[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>.

[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>.

[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>.

[RFC7405] Kyzivat, P., "Case-Sensitive String Support in ABNF", RFC 7405, DOI 10.17487/RFC7405, December 2014, <http://www.rfc-editor.org/info/rfc7405>.

9.2. Informative References

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

[IEEE.1003.1-2008] IEEE, "Information Technology - Portable Operating System Interface (POSIX(R))", IEEE Standard 1003.1-2008, DOI 10.1109/IEEESTD.2016.7582338, 2008, <https://standards.ieee.org/findstds/standard/1003.1-2008.html>.

[MNP] Held, G., "The Complete Modem Reference", Second Edition, John Wiley & Sons, Inc., May 1994.

[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>.

[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>.

[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, <http://www.itu.int/rec/T-REC-V.42bis>.

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

Acknowledgments

This document draws heavily on ideas in [RFC4978] by Arnt Gulbrandsen; 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 reviewing this specification: Mark Adler, Russ Allbery, Stephane Bortzmeyer, Francis Dupont, Angel Gonzalez, Barry Leiba, John Levine, and Brian Peterson.

Special thanks to our Document Shepherd, Michael Baeuerle, who significantly helped to increase the quality of this specification, and to Stephen Farrell for his encouragement to pursue the efforts in standardizing this NNTP extension.

Many thanks to the Responsible Area Director, Alexey Melnikov, for reviewing and sponsoring this document.

Authors' Addresses

Kenneth Murchison
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
United States of America

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/