[MS-SIPCOMP]:

Session Initiation Protocol (SIP) Compression Protocol

Revision Summary

Date	Revision History	Revision Class	Comments
4/4/2008	0.1	New	Initial version
4/25/2008	0.2	Minor	Updated based on feedback
6/27/2008	1.0	Major	Updated based on feedback
8/15/2008	1.01	Minor	Updated based on feedback
12/12/2008	2.0	Major	Updated with latest template bug fixes (redlined)
2/13/2009	2.01	Minor	Updated with latest template bug fixes (redlined)
3/13/2009	2.02	Minor	Updated with latest template bug fixes (redlined)
7/13/2009	2.03	Major	Revised and edited the technical content
8/28/2009	2.04	Editorial	Revised and edited the technical content
11/6/2009	2.05	Editorial	Revised and edited the technical content
2/19/2010	2.06	Editorial	Revised and edited the technical content
3/31/2010	2.07	Major	Updated and revised the technical content
4/30/2010	2.08	Editorial	Revised and edited the technical content
6/7/2010	2.09	Editorial	Revised and edited the technical content
6/29/2010	2.10	Editorial	Changed language and formatting in the technical content.
7/23/2010	2.10	None	No changes to the meaning, language, or formatting of the technical content.
9/27/2010	3.0	Major	Significantly changed the technical content.
11/15/2010	3.0	None	No changes to the meaning, language, or formatting of the technical content.
12/17/2010	3.0	None	No changes to the meaning, language, or formatting of the technical content.
3/18/2011	3.0	None	No changes to the meaning, language, or formatting of the technical content.
6/10/2011	3.0	None	No changes to the meaning, language, or formatting of the technical content.
1/20/2012	3.1	Minor	Clarified the meaning of the technical content.
4/11/2012	3.1	None	No changes to the meaning, language, or formatting of the technical content.
7/16/2012	3.1	None	No changes to the meaning, language, or formatting of the technical content.
10/8/2012	3.2	Minor	Clarified the meaning of the technical content.
2/11/2013	3.2.1	Editorial	Changed language and formatting in the technical content.
7/30/2013	3.2.1	None	No changes to the meaning, language, or formatting of the

Date	Revision History	Revision Class	Comments
			technical content.
11/18/2013	3.2.1	None	No changes to the meaning, language, or formatting of the technical content.
2/10/2014	3.2.1	None	No changes to the meaning, language, or formatting of the technical content.
4/30/2014	3.2.1	None	No changes to the meaning, language, or formatting of the technical content.
7/31/2014	3.3	Minor	Clarified the meaning of the technical content.
10/30/2014	3.4	Minor	Clarified the meaning of the technical content.
3/30/2015	4.0	Major	Significantly changed the technical content.
9/4/2015	4.0	None	No changes to the meaning, language, or formatting of the technical content.
7/15/2016	4.0	None	No changes to the meaning, language, or formatting of the technical content.
9/14/2016	4.0	None	No changes to the meaning, language, or formatting of the technical content.
9/29/2016	4.0	None	No changes to the meaning, language, or formatting of the technical content.
10/17/2016	4.0	None	No changes to the meaning, language, or formatting of the technical content.

Table of Contents

1 Introduction

The Session Initiation Protocol (SIP) Compression Protocol is the protocol for SIP signaling traffic compression. This protocol has two phases. The negotiation phase, which advertises and exchanges compression capabilities, and the transport phase that deals with encoding and decoding of the payload. This protocol is used by both the protocol client and the proxy.

Sections 1.5, 1.8, 1.9, 2, and 3 of this specification are normative. All other sections and examples in this specification are informative.

1.1 Glossary

This document uses the following terms:

200 OK: A response to indicate that the request has succeeded.

Augmented Backus-Naur Form (ABNF): A modified version of Backus-Naur Form (BNF), commonly used by Internet specifications. ABNF notation balances compactness and simplicity with reasonable representational power. ABNF differs from standard BNF in its definitions and uses of naming rules, repetition, alternatives, order-independence, and value ranges. For more information, see [RFC5234].

proxy: A computer, or the software that runs on it, that acts as a barrier between a network and the Internet by presenting only a single network address to external sites. By acting as a go-between that represents all internal computers, the proxy helps protects network identities while also providing access to the Internet.

Request-URI: A URI in an HTTP request message, as described in [RFC2616].

Session Initiation Protocol (SIP): An application-layer control (signaling) protocol for creating, modifying, and terminating sessions with one or more participants. **SIP** is defined in [RFC3261].

Transmission Control Protocol (TCP): A protocol used with the Internet Protocol (IP) to send data in the form of message units between computers over the Internet. TCP handles keeping track of the individual units of data (called packets) that a message is divided into for efficient routing through the Internet.

Transport Layer Security (TLS): A security protocol that supports confidentiality and integrity of messages in client and server applications communicating over open networks. **TLS** supports server and, optionally, client authentication by using X.509 certificates (as specified in [X509]). **TLS** is standardized in the IETF TLS working group. See [RFC4346].

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as defined in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

Links to a document in the Microsoft Open Specifications library point to the correct section in the most recently published version of the referenced document. However, because individual documents in the library are not updated at the same time, the section numbers in the documents may not match. You can confirm the correct section numbering by checking the Errata.

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information.

[MS-CONMGMT] Microsoft Corporation, "Connection Management Protocol".

[RFC2118] Pall, G., "Microsoft Point-To-Point Compression (MPPC) Protocol", RFC 2118, March 1997, http://www.ietf.org/rfc/rfc2118.txt

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

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and Schooler, E., "SIP: Session Initiation Protocol", RFC 3261, June 2002, http://www.ietf.org/rfc/rfc3261.txt

[RFC4346] Dierks, T., and Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, April 2006, http://www.ietf.org/rfc/rfc4346.txt

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

1.2.2 Informative References

[MS-SIPAE] Microsoft Corporation, "Session Initiation Protocol (SIP) Authentication Extensions".

1.3 Overview

This protocol provides a way to perform compression between the protocol client and its first hop **Session Initiation Protocol (SIP) proxy**. This protocol defines the usage of a modified form of the Microsoft Point-to-Point Compression (MPPC) protocol to perform compression of SIP data. This protocol also defines the protocol for negotiating compression capability. The protocol client and server can operate as the sender of compressed data.

1.3.1 Message Flow

The following figure shows the message flow for a typical compression session for this protocol.

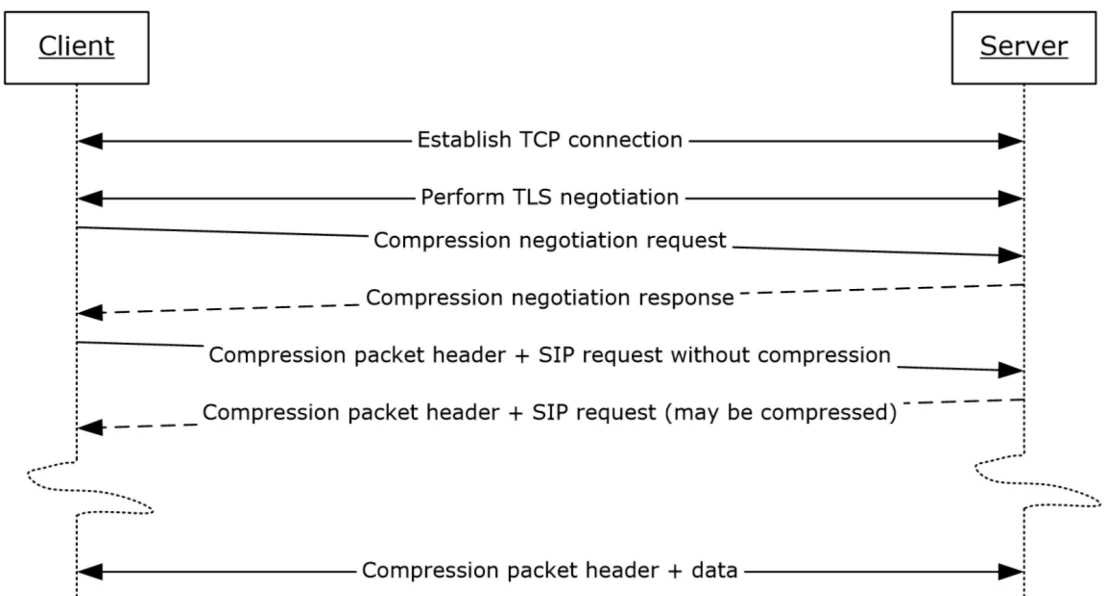

Figure 1: Typical message flow for this protocol

This protocol begins immediately following **Transport Layer Security (TLS)** negotiation. A protocol session has a negotiation phase and a transport phase. In the negotiation phase, the protocol client and server exchange a compression negotiation request and a compression negotiation response. In the transport phase, the protocol client and server exchange compression packet headers and data.

Some negotiation functions require the negotiation request to be sent before other SIP traffic begins to flow across the connection, and may also affect the format of subsequent packets. For example, Figure 2 illustrates a typical negotiation exchange that seeks to establish a compression session between the endpoints:

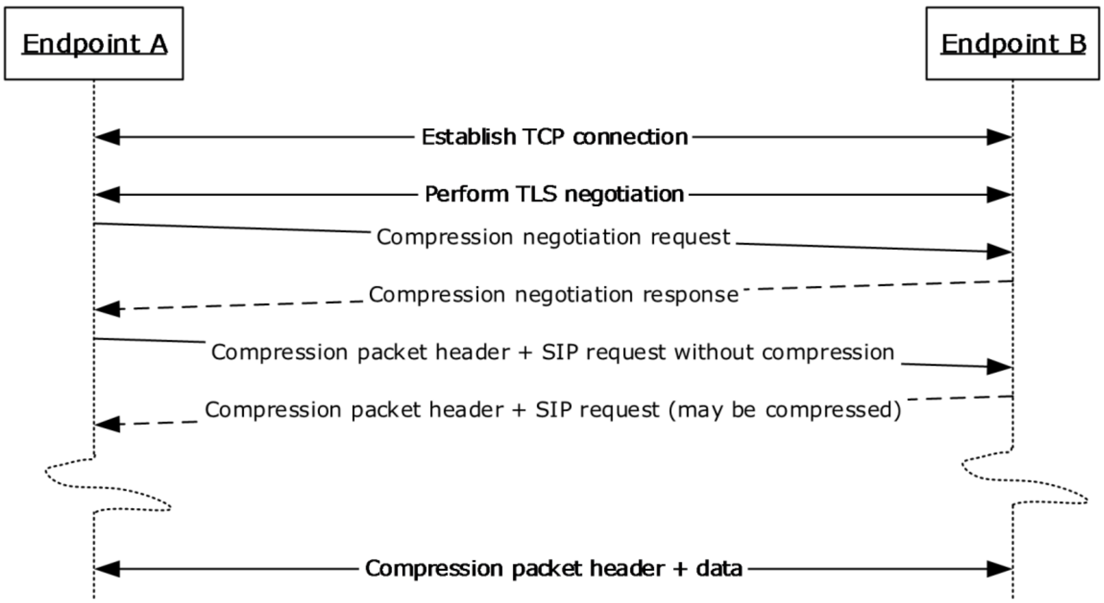

Figure 2: Typical message flow for compression negotiation

A successful compression negotiation response establishes a compression session, which alters the format of subsequent packets that flow across the connection by inserting a compression packet header ahead of the regular packet data. This session remains in effect until the connection is closed.

1.4 Relationship to Other Protocols

This protocol depends on the Microsoft Point-to-Point Compression (MPPC) protocol described in [RFC2118] for encoding and decoding compressed data. The compressed data is transported over a TLS channel.

The negotiation phase of the session determines whether data is compressed using this protocol or is sent uncompressed.

The following figure shows the logical relationship among the various protocols.

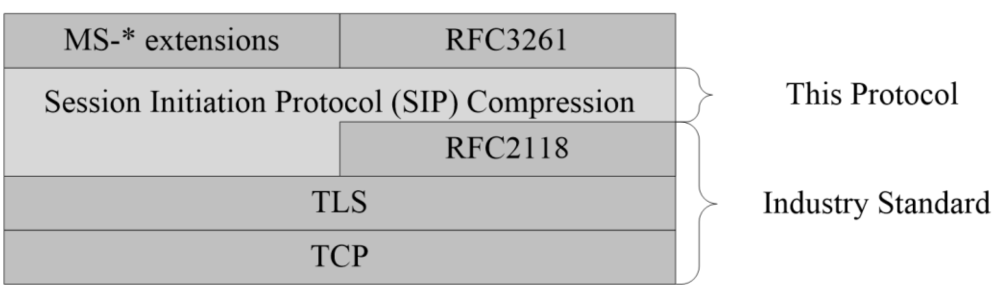

Figure 3: This protocol in relation to other protocols

1.5 Prerequisites/Preconditions

The TLS channel has to be established before this protocol starts the compression negotiation. In addition, the protocol client and server cannot have sent any SIP traffic on this connection before the compression negotiation.

1.6 Applicability Statement

This protocol is applicable when both the protocol client and the server support SIP and will use the enhancement offered by this protocol.

1.7 Versioning and Capability Negotiation

Protocol clients and servers supporting this protocol negotiate compression capability using the new **NEGOTIATE** method specified in section 2.2.1. The compression algorithm is negotiated using the **Compression** header field specified in section 2.2.3.

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

The negotiation messages and payload for this protocol MUST be transported over an established TLS channel.

2.2 Message Syntax

All of the message syntax specified in this document is described in both prose and an **Augmented Backus-Naur Form (ABNF)**, as defined in [RFC5234].

2.2.1 NEGOTIATE Request Message Format

This protocol extends [RFC3261] in defining a new SIP method for negotiation of compression. The capitalized **NEGOTIATE** token is an extension-method conforming to the method and extension-method grammar specified in [RFC3261] section 25.1 as follows:

```
Method             =  INVITEm / ACKm / OPTIONSm / BYEm
                      / CANCELm / REGISTERm
                      / extension-method
extension-method   =  token
```

The **NEGOTIATE** request MUST include the **CSeq**, **Via**, **Call-ID**, **From**, and **To** header fields constructed as specified in [RFC3261].

The **NEGOTIATE** request MUST<1> have a **Max-Forwards** header field value of 0. The **NEGOTIATE** method is not intended to be proxied beyond the first hop proxy.

The **NEGOTIATE** request MUST also include the **Compression** header field specified in section 2.2.3.

The **NEGOTIATE** request SHOULD NOT contain a **Content-Type** header field and it SHOULD NOT contain a message body.

2.2.2 Response to NEGOTIATE Request

The response for a **NEGOTIATE** request is constructed following the steps specified in [RFC3261] section 8.2.6.

In addition, the **200 OK** response for the **NEGOTIATE** request MUST contain a **Compression** header field, as specified in section 2.2.3.

2.2.3 Compression SIP Header Field Syntax

This protocol defines a new **Compression** SIP header field.

```
Compression          = "Compression" HCOLON compression-value
compression-value    = "LZ77-8K" / token
```

The **Compression** header field is used to exchange the compression algorithm to be used. Currently, "LZ77-8K" is the only supported value.

2.2.4 Compression Packet Header Format

Once compression capability is negotiated, a **Compression Packet** header MUST precede a data segment to be sent over the compression negotiated TLS channel, as specified in [RFC4346].

The size of the **Compression Packet** header MUST be 6 bytes. The **Compression Packet** header has the following format.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="4">flags</td><td colspan="4">type</td><td colspan="24">reserved</td></tr>
<tr><td colspan="16">Uncompressed size</td><td colspan="16">Data (variable, not part of the header)</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

flags (4 bits): The size of the flags MUST be 4 bits. The value is produced by performing a logical **OR** of the values in **PACKET_FLUSHED**, **PACKET_AT_FRONT**, and **PACKET_COMPRESSED**. The use of this value is further specified in section 3.2.5.1.1.

Name	Value	Description
PACKET_FLUSHED	0x8	If this flag is set, the data is not compressed and the receiver MUST reset the history buffer state. This flag MUST NOT be used in conjunction with **PACKET_COMPRESSED**.
PACKET_AT_FRONT	0x4	If this flag is set, uncompressed data is set at the beginning of the history buffer.
PACKET_COMPRESSED	0x2	If this flag is set, it indicates that the data is compressed. This flag MUST NOT be used in conjunction with **PACKET_FLUSHED**.
Undefined	0x1	This flag is not used. This flag MUST NOT be set.

type (4 bits): A 4-bit value used for the type of compression used. This value MUST be set to zero for this protocol. The server and client SHOULD ignore this value.

reserved (3 bytes): Three bytes that are not used. All bits MUST be set to zero by the sender, and MUST be ignored by the receiver.

Uncompressed size (2 bytes): The uncompressed size MUST be a 16-bit unsigned value containing the size of the original data before compression. An incorrect size MAY cause decompression to fail.

Data (variable): The data for the packet. This is not part of the header.

3 Protocol Details

3.1 Compression Negotiation Details

Both the protocol client and the server can operate as senders of compressed data. The protocol client and server advertise their compression capability and algorithm using the mechanism specified in this section.

3.1.1 Abstract Data Model

None.

3.1.2 Timers

After the protocol client sends the NEGOTIATE request, the protocol client MUST set timer F for the non-INVITE protocol client transaction, as specified in [RFC3261] section 17.1.2.2. However, instead of setting timer F to T1*64 seconds (with T1 having a default of 500ms as specified in [RFC3261] section 17.1.1.1), the protocol client SHOULD set timer F to 5 seconds. This smaller timer F value forces compression negotiation to complete within 5 seconds and shortens the maximum transport establishment delay between protocol client and server.

3.1.3 Initialization

The protocol client participating in this protocol MUST obtain the IP address of the first hop SIP proxy and the remote port with which the protocol client established the **Transmission Control Protocol (TCP)** connection and successfully negotiated the TLS channel. The first hop SIP proxy IP address and port is used to construct the **Request-URI** of the NEGOTIATE request.

3.1.4 Higher-Layer Triggered Events

3.1.4.1 Initiating Compression Negotiation

To participate in compression, a protocol client MUST send the compression negotiation request to the first hop SIP proxy after TLS negotiation is successfully finished and before sending any data on the TLS channel.

3.1.5 Message Processing Events and Sequencing Rules

This protocol uses the NEGOTIATE SIP non-INVITE transaction to negotiate compression capability. The NEGOTIATE request communicates the request to start compression. The NEGOTIATE is always sent from the protocol client to the server. The server MUST NOT start compression negotiation by sending a NEGOTIATE request to the protocol client.

3.1.5.1 Sending NEGOTIATE Request from the Client

The protocol client participating in compression MUST construct a NEGOTIATE message, as specified in section 2.2.1.

3.1.5.2 Processing NEGOTIATE Request in the Server

The server can receive a NEGOTIATE request after a TCP connection to a protocol client is established and TLS negotiation completes successfully. To participate in compression, the server MUST inspect the **Compression** header field and match the value "LZ77-8K". If the **Compression** header field does not contain "LZ77-8K", the server MUST respond to the NEGOTIATE request with a failure response code greater than or equal to 400.

If the server is unable to support compression negotiation for any reason, including internal causes such as resource limitations, the server MUST respond to the NEGOTIATE request with a failure response code greater than or equal to 400.

If the server receives a NEGOTIATE request with a **Max-Forwards** header field value greater than 0, it MUST<2> respond to the NEGOTIATE request with a failure response code greater than or equal to 400.

If the server receives a NEGOTIATE request with a **Content-Type** header field, it SHOULD ignore the header field.

If the server receives a NEGOTIATE request with a message body, it SHOULD ignore the message body. To proceed with compression negotiation, the server MUST construct a 200 OK response to the NEGOTIATE request, as specified in section 2.2.2.

The server MUST send a response to the NEGOTIATE request within 5 seconds, to prevent timer F in the protocol client from expiring.

3.1.5.3 Processing Response of NEGOTIATE Request in the Client

When the protocol client receives a response for the NEGOTIATE request, the protocol client MUST cancel the pending timer F. The protocol client then inspects the response code. Any response code other than 200 is treated as compression declined, and the protocol client and server MUST NOT start the transport phase of this protocol. If the response code is 200, the protocol client MUST inspect the **Compression** header field. If the header field value does not match "LZ77-8K", the server supports a compression algorithm that is different from the one used in this protocol. In this case, the protocol client MUST fail compression negotiation and tear down the TCP connection. If the header field value matches the expected value, the negotiation phase is successfully finished. This protocol then moves into the transport phase.

3.1.6 Timer Events

The protocol client's timer F for the NEGOTIATE non-INVITE transaction fires when the protocol client does not receive a response to the NEGOTIATE request. This is treated as compression declined, and the protocol client MUST reject any compressed data sent by the server, and MUST NOT start the transport phase of this protocol.

3.1.7 Other Local Events

If the established TCP connection is torn down on either the protocol client side or the server side, the negotiation phase is aborted and the connection is torn down, as specified in [MS-CONMGMT].

3.2 Compression Transport Details

Once the compression capability and algorithm are negotiated successfully, this protocol enters the transport phase. This protocol uses a modified form of the Point-to-Point Compression (MPPC) protocol specified in [RFC2118]. Unlike MPPC, instead of assuming an unreliable transport, this protocol compressed data is carried over a TLS channel on top of a TCP connection, which guarantees in-order transport.

Each data packet MUST include the compression packet header specified in section 2.2.4 when transporting over a connection on which this protocol has been successfully negotiated.

3.2.1 Abstract Data Model

This section describes a conceptual model of data organization that an implementation can maintain to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this protocol.

The shared state necessary to support the transmission and reception of compressed data between a protocol client and server requires a history buffer and a current offset into the history buffer (**HistoryOffset**). The size of the history buffer is 8 kilobytes. While compressing data, the sender inserts the uncompressed data that does not exceed 8 kilobytes at the position in the history buffer given by the **HistoryOffset**. After insertion, the **HistoryOffset** is advanced by the length of data added. If the data does not fit into the history buffer (the sum of the **HistoryOffset** and the size of the uncompressed data exceeds the size of the history buffer), the **HistoryOffset** MUST be reset to the start of the history buffer (offset 0).

As the receiver endpoint decompresses the data, it inserts the decompressed data at the position in the history buffer given by its local copy of **HistoryOffset**. If a reset occurs, the sender MUST notify the target receiver by setting the PACKET_FLUSHED flag in the compression packet header so it can reset its local state. After the data is decompressed, the receiver's history buffer and **HistoryOffset** are identical to the sender's history buffer and **HistoryOffset**.

Because the protocol client and server can send and receive compressed data, the protocol client and server MUST maintain two sets of state, one for sending and the other for receiving. Thus, the server maintains a history buffer and a **HistoryOffset** to send data to the protocol client, and a history buffer and a **HistoryOffset** to receive data from the protocol client. Similarly, the protocol client maintains a history buffer and a **HistoryOffset** to send data to the server, and a history buffer and a **HistoryOffset** to receive data from the server.

Both the protocol client and server SHOULD also maintain output buffers to store the output for compression and decompression operation.

3.2.2 Timers

None.

3.2.3 Initialization

The history buffer and **HistoryOffset** MUST both start initialized to zero.

3.2.4 Higher-Layer Triggered Events

None.

3.2.5 Message Processing Events and Sequencing Rules

The protocol server MAY start sending compressed data immediately after enters the transport phase. The protocol server SHOULD start sending compressed data only after it validates client identity to avoid committing memory and other resources for clients that were not yet validated. The server SHOULD use an authentication mechanism for client identity validation, such as the one those which are described in [MS-SIPAE] and MAY use any other mechanism of its choice.

The protocol client MUST NOT start compression before it receives the first compressed data from the server. The protocol client SHOULD start sending compressed data after it receives first compressed data from the server.

3.2.5.1 Compressing Data

The uncompressed data is first inserted into the local history buffer at the position indicated by the sender's **HistoryOffset**. The compressor then searches the uncompressed data for repeated series of characters, and produces output that is comprised of a sequence of literals (bytes to be sent uncompressed) and copy-tuples. Each copy-tuple represents a series of repeated characters, and consists of a <copy-offset, length-of-match> pair.

The copy-offset component of the copy-tuple is an index into the history buffer (counting backwards from the current byte towards the start of the buffer) to the most recent match of the data represented by the copy-tuple. The length-of-match component of the copy-tuple is the length of that match in bytes.

For example, consider the following string:

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

The compressor produces the following:

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

The <16,15> tuple is the compression of ".the.bell.tolls" and <40,4> is "for.", <19,3> gives "the".

The period (.) values indicate space characters.

After all data in the buffer is compressed into a sequence of literals and copy-tuples, it is then encoded using the MPPC protocol encoding scheme specified in [RFC2118] section 4.1 and section 4.2.

The tuple is constructed with the offset followed by the length-of-match.

According to [RFC2118], the offset in the tuple is to be encoded as follows:

- If the offset value is less than 64, the offset is encoded as 1111 followed by the lower 6 bits of the offset value.
- If the offset value is between 64 and 320, the offset is encoded as 1110 followed by the lower 8 bits of the offset value.
- If the offset value is between 320 and 8191, the offset is encoded as 110 followed by the lower 13 bits of the offset value.
- The offset value cannot be great than 8191 because the size of the history buffer is only 8 kilobytes.

According to [RFC2118], the length-of-match is to be encoded as follows:

- Bytes of a match of length less than 3 are encoded as literals.
- 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.

To use the preceding example, the <16,15> tuple is encoded as 1111010000110111 where the higher 10 bits 1111010000 represents the offset (16) and the lower 6 bits 110111 represents the length of match (15).

If the resulting data after encoding is greater than the original bytes (that is, expansion instead of compression results), this results in a flush and the data is sent uncompressed to avoid sending more data than the original uncompressed bytes.

3.2.5.1.1 Setting the Compression Flags

The sender MUST always specify the compression flags associated with a compressed payload. These flags MUST be set in the flags field in the compression packet header.

The compression flags are produced by performing a logical OR of the values in PACKET_FLUSHED, PACKET_AT_FRONT, and PACKET_COMPRESSED.

PACKET_FLUSHED: Indicates that the history buffer MUST be reinitialized. This value corresponds to the MPPC protocol bit A, as specified in [RFC2118] section 3.1. This flag MUST be set without setting any other flags.

This flag MUST be set if the compression generates an expansion of the data and the flag indicates to the decompressor that it needs to reset its history buffer, reset its **HistoryOffset** value, and then restart the reception of the next batch of compressed bytes. If this condition occurs, the data MUST be sent in uncompressed form.

PACKET_AT_FRONT: Indicates that the decompressed data MUST be placed at the beginning of the local history buffer. This value corresponds to the MPPC protocol bit B, as specified in [RFC2118] section 3.1. This flag MUST be set in conjunction with the PACKET_COMPRESSED (0x2) flag.

The following conditions on the compressor side generate this scenario:

- This is the first packet to be compressed.
- The data to be compressed will not fit at the end of the history buffer but, instead, needs to be placed at the start of the history buffer.

PACKET_COMPRESSED: Indicates that the data is compressed. This value corresponds to the MPPC protocol bit C, as specified in [RFC2118] section 3.1. This flag MUST be set when the data is compressed.

The following figure shows the general operation of the compressor and the production of the various flag values.

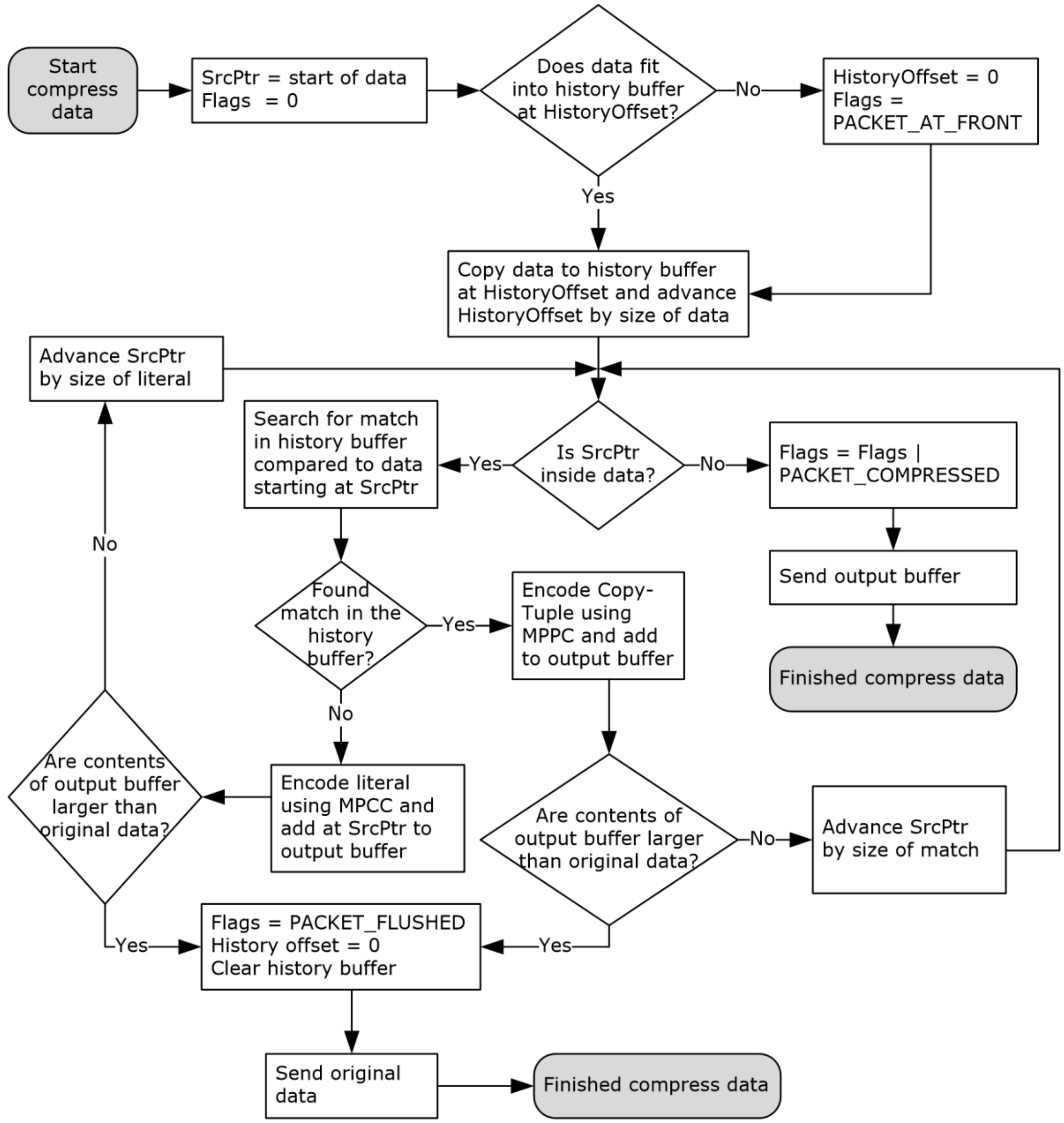

Figure 4: Compression flowchart

3.2.5.2 Decompressing Data

An endpoint which receives compressed data MUST decompress the data and store the resultant data at the end of the history buffer. The order of actions depends on the compression flags associated with the compressed data.

PACKET_FLUSHED: If this flag is set, the decompressor MUST reset its state by clearing the history buffer and resetting the **HistoryOffset** to 0.

PACKET_AT_FRONT: If this flag is set, the decompressor MUST start decompressing to the start of the history buffer, by resetting the **HistoryOffset** to 0. Otherwise, the decompressor MUST append the decompressed data to the end of the history buffer.

PACKET_COMPRESSED: If this flag is set, the decompressor MUST decompress the data, appending the decompressed data to the history buffer and advancing the **HistoryOffset**.

If the compression flags associated with the compressed data are inconsistent, the decompressor has reached an undefined state, and the receiving endpoint MUST tear down the TCP connection. Compression flags are inconsistent when PACKET_FLUSHED is set while PACKET_COMPRESSED is set.

The following diagram shows the general operation of the decompressor.

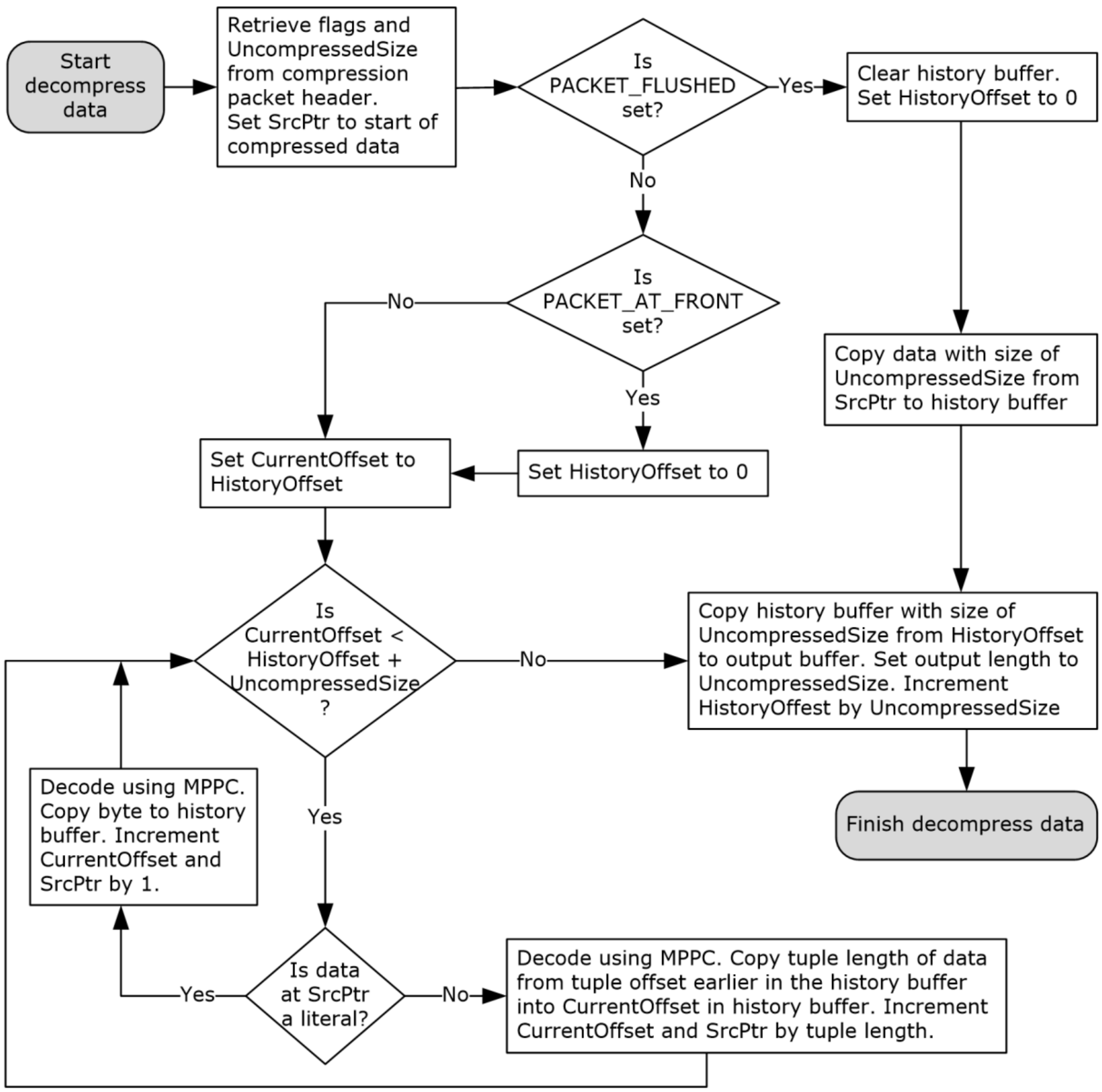

Figure 5: Decompression flowchart

3.2.6 Timer Events

None.

3.2.7 Other Local Events

None.

4 Protocol Examples

4.1 NEGOTIATE Request for Compression Negotiation

```
NEGOTIATE sip:192.0.0.1:5061 SIP/2.0
Via: SIP/2.0/TLS 192.0.0.2:2616
CSeq: 1 NEGOTIATE
Call-ID: 8d8b20f87c9c4221a732f3a70f57e9b8
From: <sip:192.0.0.2:2616>;tag=984721fb59b64e45b469c91aba8a9f8f
To: <sip:192.0.0.1:5061>
Compression: LZ77-8K
Max-Forwards: 0
Content-Length: 0
```

4.2 OK to the NEGOTIATE Request

```
SIP/2.0 200 OK
Compression: LZ77-8K
From: <sip:192.0.0.2:2616>;tag=984721fb59b64e45b469c91aba8a9f8f
To: <sip:192.0.0.1:5061>;tag=D76F601D7239923FBE84D78BF8821C85
Call-ID: 8d8b20f87c9c4221a732f3a70f57e9b8
CSeq: 1 NEGOTIATE
Via: SIP/2.0/TLS 192.0.0.2:2616;ms-received-port=2616;ms-received-cid=545400
Content-Length: 0
```

5 Security

5.1 Security Considerations for Implementers

None.

5.2 Index of Security Parameters

None.

6 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs.

- Microsoft Office Communications Server 2007
- Microsoft Office Communicator 2007
- Microsoft Office Communications Server 2007 R2
- Microsoft Office Communicator 2007 R2
- Microsoft Lync 2010
- Microsoft Lync Server 2010
- Microsoft Lync Client 2013/Skype for Business
- Microsoft Lync Server 2013
- Microsoft Skype for Business 2016
- Microsoft Skype for Business Server 2015

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms "SHOULD" or "SHOULD NOT" implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term "MAY" implies that the product does not follow the prescription.

<1> Section 2.2.1: Office Communications Server 2007 R2 can accept a **NEGOTIATE** request that does not have a **Max-Forwards** header, which is different from the specifications in [RFC3261] section 8.1.1.

<2> Section 3.1.5.2: Office Communications Server 2007 R2, Office Communicator 2007 R2: If the server receives a NEGOTIATE request with a Max-Forwards header field value greater than 0, it ignores the header field value.

7 Change Tracking

No table of changes is available. The document is either new or has had no changes since its last release.

8 Index

R

S

T

V