Applying Signaling Compression (SigComp) to the Session Initiation Protocol (SIP)
draft-ietf-rohc-sigcomp-sip-02.txt

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on August 18, 2006.

Copyright Notice

Copyright (C) The Internet Society (2006).

Abstract

This document describes some specifics that apply when Signaling
Compression (SigComp) is applied to the Session Initiation Protocol (SIP), such as default minimum values of SigComp parameters, compartment and state management, and a few issues on SigComp over TCP. Any implementation of SigComp for use with SIP must conform to this document, in addition to SigComp and support of the SIP and Session Description Protocol (SDP) static dictionary.

Table of Contents

1. Introduction ............................................. 3
2. Terminology ............................................. 3
3. Minimum Values of SigComp Parameters for SIP/SigComp ........ 3
   3.1. decompression_memory_size (DMS) for SIP/SigComp .......... 4
   3.2. state_memory_size (SMS) for SIP/SigComp ............... 4
   3.3. cycles_per_bit (CPB) for SIP/SigComp ................. 5
   3.4. SigComp_version (SV) for SIP/SigComp ............... 5
   3.5. locally available state (LAS) for SIP/SigComp .......... 5
4. Delimiting SIP Messages and SigComp Messages on the Same Port ............................................. 5
5. Continuous Mode over TCP .................................. 6
6. Compartment and State Management for SIP/SigComp .......... 7
   6.1. Remote Application Identifiers .......................... 7
   6.2. Compartment Opening and Closure ....................... 7
   6.3. Compartment Valid During a Transaction ............... 9
   6.4. Compartment Valid During a Registration ............. 9
   6.5. Compartment Valid During a Dialog .................... 10
7. Recommendations for Network Administrators ................. 10
8. Security Considerations ................................... 11
9. Private Agreements ........................................ 11
10. IANA Considerations ...................................... 11
11. Acknowledgements ........................................ 11
12. Normative References ..................................... 11
Appendix A. Shim header for sending uncompressed messages .... 12
Authors’ Addresses .......................................... 14
Intellectual Property and Copyright Statements ................. 15
1. Introduction

SigComp [RFC3320] is a solution for compressing messages generated by application protocols. Although its primary driver is to compress SIP [RFC3261] messages, the solution itself has been intentionally designed to be application agnostic so that it can be applied to any application protocol. (This is denoted as ANY/SigComp.) Consequently, many application dependent specifics are left out of the base standard. It is intended that a separate specification is used to describe those specifics when SigComp is applied to a particular application protocol.

This document binds SigComp and SIP (denoted as SIP/SigComp). Any SigComp implementation that is used for the compression of SIP messages must conform to this document, as well as to [RFC3320] and must support the SIP/SDP static dictionary as specified in [RFC3485].

Note: the mechanism of discovering SigComp support at the SIP layer is specified in [RFC3486].

2. Terminology

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 BCP 14 [RFC2119].

3. Minimum Values of SigComp Parameters for SIP/SigComp

In order to support a wide range of capabilities among endpoints implementing SigComp, SigComp defines a few parameters to describe SigComp behavior (see section 3.3 of [RFC3320]). For each parameter, [RFC3320] specifies a minimum value that any SigComp endpoint MUST support for ANY/SigComp. Those minimum values were determined with the consideration of all imaginable devices in which SigComp may be implemented. Scalability was also considered as a key factor.

However, some of the minimum values specified in [RFC3320] are too small to allow good performance for SIP message compression. Therefore, they are increased for SIP/SigComp as specified in the following sections. For completeness, those parameters that are the same for SIP/SigComp as they are for ANY/SigComp are also listed.

Note: the new minimum values are specific to SIP/SigComp. They do not apply to any other application protocols.

Note: a SigComp endpoint MAY offer additional resources if available;
these resources can be advertised to remote endpoints as described in section 9.4.9 of [RFC3320].

3.1. decompression_memory_size (DMS) for SIP/SigComp

Minimum value for ANY/SigComp: 2048 bytes, as specified in section 3.3.1 of [RFC3320].

Minimum value for SIP/SigComp: 8192 bytes.

Reason: a DMS of 2048 bytes is too small for SIP message compression, as it seriously limits the compression ratio and even makes compression impossible for certain messages. For example, the condition set by [RFC3320] for SigComp over UDP means: C + 2*B + R + 2*S + 128 < DMS (each term is described below). On the other hand, 8KB additional memory should not cause any problem for an endpoint that already implements SIP, SigComp, and applications that use SIP, as DMS is memory only temporarily needed during decompression of a SigComp message (the memory can be reclaimed when the message has been decompressed).

C  size of compressed application message, depending on R
B  size of bytecode (note: two copies -- one as part of the SigComp message and one in UDVM memory)
R  size of ring buffer in UDVM memory
S  any additional state uploaded other than that created from the content of the ring buffer at the end of decompression (similar to B, two copies of S are needed)
128 the smallest address in UDVM memory to copy bytecode to

3.2. state_memory_size (SMS) for SIP/SigComp

Minimum value for ANY/SigComp: 0 (zero) bytes, as specified in section 3.3.1 of [RFC3320].

Minimum value for SIP/SigComp: 2048 bytes.

Reason: a non-zero SMS allows an endpoint to upload a state in the first SIP message sent to a remote endpoint without the uncertainty of whether or not it can be created in the remote endpoint. A non-zero SMS obviously requires the SIP/SigComp implementation to keep state. Based on the observation that there is little gain from stateless SigComp compression, the assumption is that purely stateless SIP implementations are unlikely to provide a SigComp function. Stateful implementations should have little problem to keep 2K additional state for each compartment (see Section 6).

Note: SMS is a parameter that applies to each individual compartment.
An endpoint MAY offer different SMS values for different compartments as long as the SMS value is not less than 2048 bytes.

Compressors that make use of initial state memory MUST implement the SigComp Negative Acknowledgement (NACK) Mechanism [I-D.ietf-rohc-sigcomp-nack]. (Note that there is no such requirement on decompressors, but see also Section 6.) For this requirement, initial state memory is defined as the assumption of a non-zero SMS value before having received an advertisement of non-zero SMS (e.g., via returned parameters as specified in section 9.4.9 of [RFC3320]); ANY/SigComp as defined in [RFC3320] does not have initial state memory.

3.3. cycles_per_bit (CPB) for SIP/SigComp

Minimum value for ANY/SigComp: 16, as specified in section 3.3.1 of [RFC3320].

Minimum value for SIP/SigComp: 16 (same as above)

3.4. SigComp_version (SV) for SIP/SigComp

For ANY/SigComp: 0x01, as specified in section 3.3.2 of [RFC3320].

OPEN ISSUE: what SigComp version(s) should be required for SIP/SigComp?

   Version >= 0x01 (i.e. any SigComp)
   Version >= 0x02 (i.e. at least SigComp + NACK)
   Version == 0x01 | 0x02 (base SigComp or SigComp + NACK)
   Version == 0x02 (only SigComp with NACK)

For SIP/SigComp: 0x01 (same as above)

3.5. locally available state (LAS) for SIP/SigComp

Minimum LAS for ANY/SigComp: none, see section 3.3.3 of [RFC3320].

Minimum LAS for SIP/SigComp: the SIP/SDP static dictionary as defined in [RFC3485].

4. Delimiting SIP Messages and SigComp Messages on the Same Port

In order to limit the number of ports required by a SigComp-aware endpoint, it is possible to allow both SigComp messages and 'vanilla' SIP messages (i.e. uncompressed SIP messages with no SigComp header) to arrive on the same port.
For a message-based transport such as UDP or SCTP, this can be done per message. The receiving endpoint checks the first octet of the UDP/SCTP payload to determine whether the message has been compressed using SigComp. If the MSBs of the octet are "11111" then the message is considered to be a SigComp message and is parsed as per [RFC3320]. If the MSBs of the octet take any other value, then the message is assumed to be an uncompressed SIP message, and is passed directly to the application with no further effect on the SigComp layer.

For a stream-based transport such as TCP, the distinction is per connection. The receiving endpoint checks the first octet of the TCP data stream to determine whether the stream has been compressed using SigComp. If the MSBs of the octet are "11111" then the stream is considered to contain SigComp messages and is parsed as per [RFC3320]. If the MSBs of the octet take any other value, then the stream is assumed to contain uncompressed SIP messages, and is passed directly to the application with no further effect on the SigComp layer. Note that SigComp message delimiters MUST NOT be used if the stream contains uncompressed SIP messages.

Applications MUST NOT mix SIP messages and SigComp messages on a single TCP connection. If the TCP connection is used to carry SigComp messages then all messages sent over the connection MUST have a SigComp header and be delimited by the use of 0xFFFF as described in [RFC3320].

Note: Appendix A shows how to send uncompressed messages in a SigComp structured TCP connection using a "well-known shim header". Should it for any reason not be desirable to set up more than one TCP connection to a SIP implementation, but the flexibility to send both compressed and uncompressed SIP messages be required, the compressor can set up a SigComp structured connection and send any uncompressed SIP messages using the well-known shim header.

5. Continuous Mode over TCP

Continuous Mode is a special feature of SigComp, which is designed to improve the overall compression ratio for long-lived connections. Its use requires pre-agreement between the SigComp compressor and decompressor. Continuous mode is not used with SIP/SigComp.

Reason: continuous mode requires the transport itself to provide a certain level of protection against denial of service attacks. TCP alone is not considered to provide enough protection.
An application exchanging compressed traffic with a remote application has a compartment that contains state information needed to compress outgoing messages and to decompress incoming messages. To increase the compression efficiency, the application must assign distinct compartments to distinct remote applications.

6.1. Remote Application Identifiers

SIP/SigComp applications identify remote applications by their FQDN (Fully Qualified Domain Name) or by their IP address. For outgoing requests, the remote application identifier is the host part of the URI to which the request is sent. For incoming responses, the remote application identifier is the same as the one for the previously-sent request that initiated the transaction the response belongs to. For incoming requests and outgoing responses, the remote application identifier is the sent-by parameter of the top-most Via entry.

A given remote application identifier is mapped to a particular SigComp compartment ID following the rules given in the following sections.

OPEN ISSUE: this is an implicit way of identifying remote applications. It assumes that two remote applications are different if the host parts of their URIs are different. However, if a proxy farm shares dictionary state among its proxies and these proxies use different host parts (e.g., proxy1.example.com and proxy2.example.com), they will be considered like different remote applications, when they should have been considered a single remote application. If implementers intend to implement state sharing this way, we could use explicit application identifiers instead. These identifiers could be placed in a SIP URI parameter (e.g., sip:p1.example.net;id="12wsfeQ45") and in a Via parameter.

6.2. Compartment Opening and Closure

SIP applications need to know when to open a new compartment and when to close it. The lifetime of a compartment depends on how the SIP application obtained the remote application identifier (e.g., in a Record-Route header field of an incoming SIP message). There are compartments that are valid for the duration of a registration, of a dialog, and of a single transaction. The following sections specify how a SIP application decides the lifetime of a particular compartment.

If following the rules in the following sections, a SIP application
is supposed to open a compartment for a remote application identifier for which it already has a compartment, the SIP application MUST use the already existing compartment. That is, the SIP application MUST NOT open a new compartment. Additionally, the SIP application MUST adjust the closure time for the compartment so that it is only closed when the SIP application does not need it any longer.

For example, a SIP application may open a compartment valid for the duration of a registration for a particular remote application identifier. At a later point, the application is supposed to open a new compartment for the duration of a particular dialog for the same remote application identifier. Following the previous rule, the SIP application does not open a new compartment but use the already existing one for that remote application identifier. However, the SIP application must not close that compartment until both, the registration and the dialog are over. So, if the registration finishes before the dialog, the compartment will not be closed (because the dialog is still active) even though the compartment was originally open for the registration.

Usually, any states created during the lifetime of a compartment will be "logically" deleted when the compartment is closed. (As described in section 6.2 of [RFC3320], a logical deletion can become a physical deletion only when no compartment continues to exist that created the (same) state.)

A SigComp endpoint may offer to keep a state created upon request from a SigComp peer endpoint beyond the default lifetime of a compartment. This may be used to improve compression efficiency of subsequent SIP messages generated by the same remote application at the SigComp peer endpoint. To indicate that such state will continue to be available, the SigComp endpoint can inform its peer SigComp endpoint by announcing the (partial) state ID in the returned SigComp parameters at the end of the registration, dialog, or transaction that was supposed to limit the lifetime of the SigComp state. That signals the state will be maintained. As there is no way to signal any limit to the lifetime of this state, both decompressors that intend to offer state with possibly limited lifetimes as well as compressors that make use of such state SHOULD implement the SigComp Negative Acknowledgement (NACK) Mechanism [I-D.ietf-rohc-sigcomp-nack].

As an operational concern, bugs in the compartment management implementation are likely to lead to sporadic, hard to diagnose failures. Decompressors may therefore want to cache old state and, if still available, allow access while logging diagnostic information. Both compressors and decompressors are RECOMMENDED to implement the SigComp Negative Acknowledgement (NACK) Mechanism.
[I-D.ietf-rohc-sigcomp-nack], which facilitates recovery in a situation where such old state may no longer be available.

6.3. Compartment Valid During a Transaction

A SIP application that needs to send a compressed SIP request SHOULD open a compartment for the request’s remote application identifier. This compartment will be used to receive compressed responses for the request. The application can close the compartment when the transaction is over.

A SIP application receives a compressed SIP request SHOULD open a compartment for the request’s remote application identifier. This compartment will be used to send compressed responses for the request. The application SHOULD NOT close the compartment until the transaction is over.

The previous rules ensure that SIP applications always have an already existing compartment to send and receive responses.

6.4. Compartment Valid During a Registration

A REGISTER transaction can cause an application to open a new compartment to be valid for the duration of the registration established by the REGISTER transaction.

A 200 (OK) response for a register may contain a Path [RFC3327] and a Service-Route [RFC3308] header field. These header fields indicate the route future incoming and outgoing requests will follow.

On receiving a 200 (OK) response for a REGISTER, a SIP application that inserted itself in the Contact (i.e., because it is the user agent) or in the Path header field of the REGISTER, constructs the route future incoming requests will follow (using the Contact and the Path header fields) and the route future outgoing requests will follow (using the Contact and the Service-Route header fields). The application checks whether the URIs of its adjacent applications in both routes have the comp=sigcomp parameter. The application SHOULD open a new compartment for the remote application identifier of the URIs with that parameter. The application SHOULD NOT close the compartments until the registration is over.

Note that the route for incoming requests is typically the same (although traversed in the opposite direction) as the route for outgoing requests.

Note that some user agents use several registration in parallel to improve service reliability. Different registration typically have
different associated route vectors. Messages sent to different remote application identifiers will use different compartments, even if those messages are generated by the same user agent. It is assumed that the remote applications do not share SIP/SigComp state among them.

6.5. Compartment Valid During a Dialog

A transaction that establishes a dialog can cause an application to open a new compartment to be valid for the duration of the dialog established by the transaction.

A SIP message that establishes a dialog (e.g., a 2xx response for an INVITE) may contain a Record-Route header field. This header field indicates the route future requests within the dialog will follow.

On receiving such a SIP message, a SIP application that inserted itself in the Contact (i.e., because it is the user agent) or in the Record-Route header field of the request, constructs (using the Contact, and the Record-Route header fields) the route requests within the dialog will follow. The application checks whether the URIs of its adjacent applications in that route have the "comp=sigcomp" parameter. The application SHOULD open a new compartment for the remote application identifier of the URIs with that parameter. The application SHOULD NOT close the compartments until the dialog is over.

7. Recommendations for Network Administrators

Network administrators can configure their networks so that the compression efficiency achieved is increased. The following recommendations help network administrators perform their task.

For a given user agent, the route sets for incoming requests (created by a Path header field) and for outgoing requests (created by a Service-Route header field) are typically the same. However, registrars can, if they wish, insert proxies in the latter route that do not appear in the former route and vice versa. It is RECOMMENDED that registrars are configured so that proxies performing SigComp compression appear in both routes.

The routes described previously apply to requests sent outside a dialog. Requests inside a dialog follow a route constructed using Record-Route header fields. It is RECOMMENDED that the proxies performing SigComp that are in the route for requests outside a dialog are configured to place themselves (by inserting themselves in the Record-Route header fields) in the routes used for requests.
inside dialogs.

8. Security Considerations

The same security considerations as described in [RFC3320] apply to this document. Note that keeping SigComp states longer than the duration of a SIP dialog should not pose new security risks for two reasons: a) the state has been allowed to be created in the first place; and b) this is on voluntary basis and a SigComp endpoint can choose not to offer it.

9. Private Agreements

SIP/SigComp implementations that are subject to private agreements MAY deviate from this specification, if the private agreements unambiguously specify so. Plausible candidates for such deviations include:

- Minimum values (Section 3).
- Compartment definition (Section 6).
- Use of continuous mode (Section 5).

10. IANA Considerations

This specification does not require any actions from the IANA.

11. Acknowledgements

Abigail Surtees provided the code and text for Appendix A.

The authors would like to thank the following people for their comments and suggestions: Abigail Surtees, Jan Christoffersson, Joerg Ott, Mark West, Pekka Pessi, Robert Sugar, and Adam Roach.

12. Normative References


Appendix A. Shim header for sending uncompressed messages

This appendix presents bytecode that simply instructs the decompressor to output the entire message (effectively sending it uncompressed but within a SigComp message).

The mnemonic code is:


at (0)
:udvm_memory_size  pad (2)
:cycles_per_bit  pad (2)
:sigcomp_version  pad (2)
:partial_state_id_length  pad (2)
:state_length  pad (2)
:reserved  pad (2)

at (64)
:byte_copy_left  pad (2)
:byte_copy_right  pad (2)
:input_bit_order  pad (2)
:stack_location  pad (2)

; Simple loop
;      Read a byte
;      Output a byte
; Until there are no more bytes!

at (128)
:start
  INPUT-BYTES (1, byte_copy_left, end)
  OUTPUT (byte_copy_left, 1)
  JUMP (start)

:end
  END-MESSAGE (0,0,0,0,0,0,0)

which translates to give the following initial 13 bytes of the SigComp message (in hexadecimal):

f8 00 a1 1c 01 86 09 22 86 01 16 f9 23

As an implementation optimization, a SigComp implementation MAY compare the initial 13 bytes of each incoming message with the 13 bytes given (the "well-known shim header"), and, in case of a match, simply copy the SigComp message data that follow the shim header without even setting up a UDVM. (Note that, before a SigComp message is formed from the incoming TCP data, the record marking protocol defined in section 4.2.2 of [RFC3320] has to be performed.)

To obtain the maximum benefit from this optimization, compressors SHOULD employ exactly the well-known shim header given (and none of the other conceivable byte code sequences for just copying input to output) to send uncompressed data in a SigComp channel.
Authors’ Addresses

Carsten Bormann (editor)
Universitaet Bremen T2I
Postfach 330440
Bremen D-28334
Germany

Phone: +49 421 218 7024
Fax: +49 421 218 7000
Email: cabo@tzi.org

Zhigang Liu
Nokia Research Center
6000 Connection Drive
Irving, TX 75039
USA

Phone: +1 972 894-5935
Email: zhigang.c.liu@nokia.com

Richard Price
Cogent Defence and Security Networks
Queensway Meadows Industrial Estate
Meadows Road
Newport, Gwent NP19 4SS

Phone: +44 (0)1794 833681
Email: richard.price@cogent-dsn.com
URI: http://www.cogent-dsn.com

Gonzalo Camarillo
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: Gonzalo.Camarillo@ericsson.com
Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Copyright Statement

Copyright (C) The Internet Society (2006). This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.