            Signaling Compression (SigComp) - Extended Operations

Status of this Memo

Copyright Notice

Abstract

   This document describes how to implement certain mechanisms in
   Signaling Compression (SigComp), RFC 3320, which can significantly
   improve the compression efficiency compared to using simple per-
   message compression.

   SigComp uses a Universal Decompressor Virtual Machine (UDVM) for
   decompression, and the mechanisms described in this document are
   possible to implement using the UDVM instructions defined in RFC
   3320.

Table of Contents

1. Introduction

   This document describes how to implement mechanisms with [SIGCOMP] to
   significantly improve the compression efficiency compared to per-
   message compression.

   One such mechanism is to use previously sent messages in the SigComp
   compression process, referred to as dynamic compression. In order to
   utilize information from previously sent messages, it is necessary
   for a compressor to gain knowledge about the reception of these
   messages. For a reliable transport, such as TCP, this is guaranteed.
   For an unreliable transport however, the SigComp protocol can be used
   to provide such a functionality itself. That functionality is
   described in this document and is referred to as explicit
   acknowledgement.

   Another mechanism that will improve the compression efficiency of
   SigComp, especially when SigComp is applied to protocols that are of
   request/response type, is shared compression. This involves using
   received messages in the SigComp compression process. In particular
   the compression of the first few messages will gain from shared
   compression. Shared compression is described in this document.

   For better understanding of this document the reader should be
   familiar with the concept of [SIGCOMP].

2. Terminology

   The reader should consult [SIGCOMP] for definitions of terminology,
   since this document uses the same terminology. Further terminology
   is defined below.

     Compressor

       Entity that encodes application messages using a certain
       compression algorithm and keeps track of state that can be used
       for compression. The compressor is responsible for ensuring that
       the messages it generates can be decompressed by the remote UDVM.

     Decompressor

       The decompressor is responsible for converting a SigComp message
       into uncompressed data. Decompression functionality is provided
       by the UDVM.

     Dynamic compression

       Compression relative to messages sent prior to the current
       compressed message.

     Explicit acknowledgement

       Acknowledgement for a state. The acknowledgment is explicitly
       sent from a decompressor to its remote compressor. The
       acknowledgement should be piggybacked onto a SigComp message in
       order not to create additional security risks.

     Shared compression

       Compression relative to messages received by the local endpoint
       prior to the current compressed message.

     Shared state

       A state used for shared compression consists only of an
       uncompressed message. This makes the state independent of the
       compression algorithm.

State identifier

Reference used to access a previously created item of state.

- shared_state_id

State identifier of a shared state.

- acked_state_id

State identifier of a state that is acknowledged as
successfully saved by the decompressor.

3. Architectural View of Feedback

SigComp has a request/response mechanism to provide feedback between
endpoints, see Figure 1. This particular functionality of SigComp is
used in this document to provide support for the mechanisms described
in this document.

```
       +-------------------+               +-------------------+
       |    Endpoint 1     |               |    Endpoint 2     |
       | +-------------+   |               | +-------------+   |
       | | Compressor 1|   |               | |Decompressor 2|  |
       | | [-----------+--+-------------+--+--]    *        |   |
       | +-|-------^----+   |               | +--|---|-------+   |
       |   |       |        |               |    |   |           |
       |   |       |        |               |    |   |           |
       |   |       |        |               |    |   |           |
       | +-|-------|----+   |               | +--v---|-------+   |
       | | *       [----+--+-------------+--+------]  |        |   |
       | |Decompressor 1|  |               | | Compressor 2 |  |
       | +-------------+   |               | +-------------+   |
       +-------------------+               +-------------------+
```
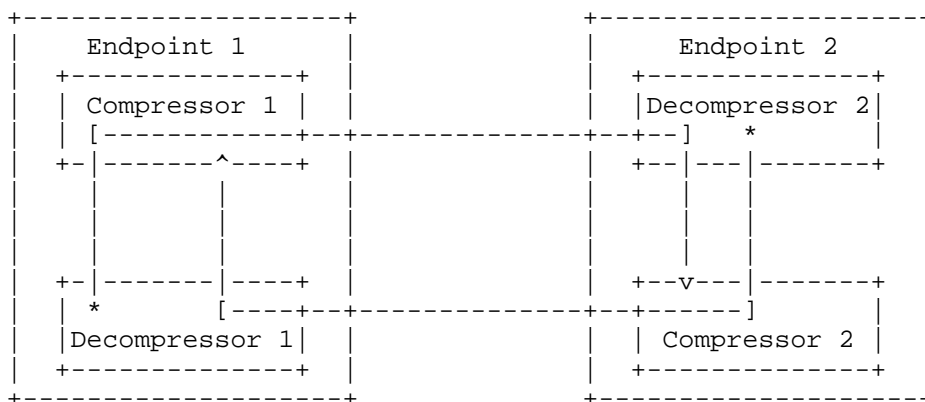
Figure 1. Architectural view

The feedback functionality of SigComp is used in this document to
provide a mechanism for a SigComp endpoint to confirm which states
have been established by its remote SigComp endpoint during the
lifetime of a SigComp compartment. The established state
confirmations are referred to as acknowledgments. Depending on the
established states this particular type of feedback may or may not be
used to increase the compression efficiency.

The following sections describe how the SigComp functionality of
providing feedback information is used to support the mechanisms
described in this document. Section 4 describes the state reference
model of SigComp. Section 5 continues with a general description of
the mechanisms and Section 6 describes the implications of some of
the mechanisms on basic SigComp.

4. State Reference Model

A UDVM may want to save the status of its memory, and this status is
referred to as a state. As explained in [SIGCOMP] a state save
request may or may not be granted by the application. For later
reference to a saved state, e.g., if the UDVM is to be loaded with
this state, a reference is needed to locate the specific state. This
reference is called a state identifier.

4.1. Overview of State Reference with Dynamic Compression

When compressor 1 compresses a message m it uses the information
corresponding to a SigComp state that its remote decompressor 2 has
established and acknowledged. If compressor 1 wishes to use the new
state for compression of later messages it must save the new state.
The new state contains information from the former state and from m.
When an acknowledgement is received for this new state, compressor 1
can utilize the new state in the compression process. Below is an
overview of the model together with an example of a message flow.

Saved state(s)

  A state which is expected to be used for compression/decompression
  of later messages.

Acked state(s)

  An acked state is a saved state for which the compressor has
  received an acknowledgement, i.e., the state has been established
  at the remote decompressor. The compressor must only use states
  that are established at the remote decompressor, otherwise a
  decompression failure will occur. For this reason,
  acknowledgements are necessary, at least for unreliable transport.

```
              Compressor 1                         Decompressor 2
                +---+                                 +---+
                | C |                                 | D |
                +---+                                 +---+

       Saved        Acked    |              |   Saved
      State(s)     State(s)  |              |  State(s)
     --------------------+-----------+-----------------
     s0            s0        |              |    s0
     s1=s0+m1                | --m1(s0)-->|
                             | <--ack(s1) |   s0,s1
     s0,s1        s0,s1      |              |

     s0,s1        s0,s1      | --m2(s1)-->|   (m2 Lost)
     s2=s1+m1                |              |

     s0-s2        s0,s1      |              |
     s3=s1+m3                | --m3(s1)-->|   s0,s1

                             |              |
                             | <--ack(s3) |   s0,s1,s3=s1+m3
     s0-s3      s0,s1,s3     |              |
```
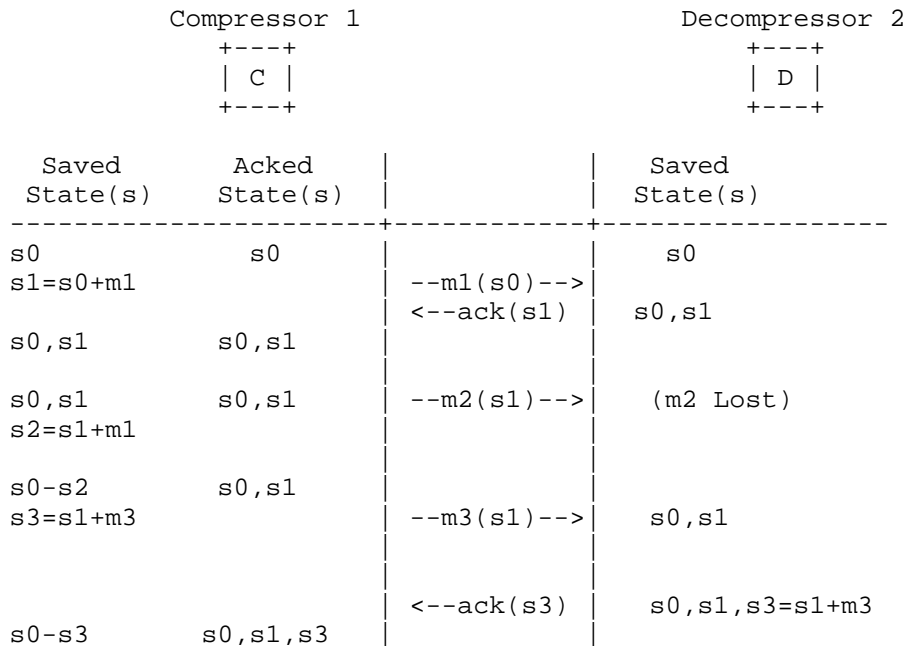
              Figure 2. Example of message flow for dynamic compression

   Legend: Message 1 compressed making use of state s0 is denoted
   m1(s0). The notation s1=s0+m1 means that state s1 is created using
   information from state s0 and message m1. ack(s1) means that the
   creation of state s1 is acknowledged through piggybacking on a
   message traveling in the reverse direction (which is not shown in the
   figure).

5. Extended Mechanisms

   The following subsections give a general description of the extended
   mechanisms.

5.1. Explicit Acknowledgement Scheme

   For a compressor to be able to utilize a certain state it must know
   that the remote decompressor has access to this state.

   In the case where compressed messages can be lost or misordered on
   the path between compressor and decompressor, an acknowledgement
   scheme must be used to notify the remote compressor that a certain
   state has been established.

Explicit acknowledgements can be initiated either by UDVM-code
uploaded to the decompressor by the remote compressor or by the
endpoint where the states have been established. These two cases
will be explained in more detail in the following two sections.

5.1.1. Remote Compressor Initiated Acknowledgements

This is the case when e.g., compressor 1 has uploaded UDVM bytecode
to decompressor 2. The UDVM bytecode will use the requested feedback
field in the announcement information and the returned feedback field
in the SigComp header to obtain knowledge about established states at
endpoint 2.

   Consider Figure 3. An event flow for successful use of remote
   compressor initiated acknowledgements can be as follows:

   (1): Compressor 1 saves e.g., state(A).
   (2): The UDVM bytecode to initiate a state save for state(A) is
        either carried in the compressed message, or can be retrieved by
        decompressor 2 from a state already saved at endpoint 2.
   (3): As compressor 1 is the initiator of this acknowledgement it can
        use an arbitrary identifier to be returned to indicate that
        state(A) has been established. The identifier needs to consist
        of enough bits to avoid acknowledgement of wrong state.
        To avoid padding of the feedback items and for simplicity a
        minimum of 1 octet should be used for the identifier.
        The identifier is placed at the location of the
        requested_feedback_item [SIGCOMP].
        The END-MESSAGE instruction is used to indicate the location of
        the requested_feedback_item to the state handler.
   (4): The requested feedback data is now called returned feedback data
        as it is placed into the SigComp message at compressor 2.
   (5): The returned feedback item is carried in the SigComp message
        according to Figure 4: see Section 6.1 and [SIGCOMP].
   (6): The returned feedback item is handled according to: Section 7
        of [SIGCOMP]

```
     +--------------+            (2)          +--------------+
     | Compressor 1 |-------------------------->|Decompressor 2|
     +------^-------+                         +-------^------+
         |     (1)                              (3)    |
       +---v---+                               +---v---+
       |State  |                               |State  |
       |handler|                               |handler|
       +---^---+                               +---^---+
         |     (6)                              (4)    |
     +------v-------+            (5)          +-------v------+
     |Decompressor 1|<--------------------------| Compressor 2 |
     +--------------+                         +--------------+
```
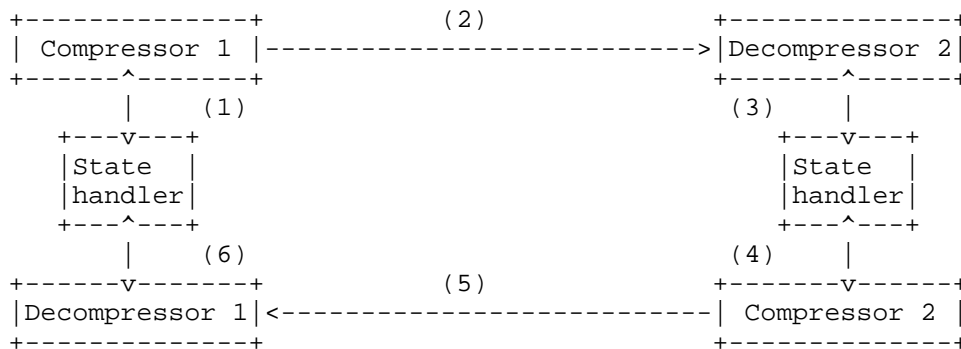
            Figure 3. Simplified SigComp endpoints

5.1.2. Local Endpoint Initiated Acknowledgements

   When explicit acknowledgements are provided by an endpoint, the
   SigComp message will also carry acknowledgements, so-called
   acked_state_id: see Section 2. Consider Figure 3, an event flow for
   successful use of explicit endpoint initiated acknowledgements can be
   as follows:

   (1): Compressor 1 saves e.g., state(A).
   (2): The UDVM bytecode to initiate a state save for state(A) is
        either carried in the compressed message, or can be retrieved by
        decompressor 2 from a state already saved at endpoint 2.
   (3): A save state request for state(A) is passed to the state handler
        using the END-MESSAGE instruction. The application may then
        grant the state handler permission to save state(A): see
        [SIGCOMP].
   (4): Endpoint 2 decides to acknowledge state(A) to endpoint 1. The
        state identifier (acked_state_id) for state(A) is placed in
        the SigComp message sent from compressor 2 to decompressor 1.
   (5): The UDVM bytecode to initiate (pass) the explicit
        acknowledgement to endpoint 1 is either carried in the
        compressed message, or can be retrieved by decompressor 1 from a
        state already saved at endpoint 1.
   (6): The acked_state_id for state(A) is passed to the state handler
        by placing the acked_state_id at the location of the
        "returned SigComp parameters" [SIGCOMP], whose location is given
        to the state handler using the END-MESSAGE instruction.

   Note: When the requested feedback length is non-zero endpoint
   initiated acknowledgements should not be used, due to possible waste
   of bandwidth. When deciding to implement this mechanism one should
   consider whether this is worth the effort as all SigComp
   implementations will support the feedback mechanism and thus have the
   possibility to implement the mechanism of Section 5.1.1.

5.2. Shared Compression

   To make use of shared compression a compressing endpoint saves the
   uncompressed version of the compressed message as a state (shared
   state). As described in Section 2 the reference to a shared state is
   referred to as shared_state_id. The shared state's parameters
   state_address and state_instruction must be set to zero. The
   state_retention_priority must be set to 65535, and the other state
   parameters are set according to [SIGCOMP]. This is because different
   compression algorithms may be used to compress application messages
   traveling in different directions. The shared state is also created
   on a per-compartment basis, i.e., the shared state is stored in the
   same memory as the states created by the particular remote

compressor. The choice of how to divide the state memory between
"ordinary" states and shared states is an implementation decision at
the compressor. Note that new shared state items must not be created
unless the compressor has made enough state memory available (as
decompression failure could occur if the shared state pushed existing
state out of the state memory buffer).

A compressing endpoint must also indicate to the remote compressor
that the shared state is available, but only if the local
decompressor can retrieve the shared state. The retrieval of the
shared state is done according to the state retrieval instruction of
the UDVM.

Consider Figure 3. An event flow for successful use of shared
compression can be as follows:

(1): Compressor 1 saves e.g., state(M), which is the uncompressed
     version of the current application message to be compressed and
     sent.
(2): The UDVM bytecode to indicate the presence of state(M) at
     endpoint 1 is either carried in the compressed message, or can
     be retrieved by decompressor 2 from a state already saved at
     endpoint 2.
(3): The SHA-1 instruction is used at endpoint 2 to calculate the
     shared_state_id for state(M). The indication is passed to the
     state handler, by placing the shared identifier at the location
     of the "returned SigComp parameters" [SIGCOMP]. The location of
     the "returned SigComp parameters" is given to the state handler
     using the END-MESSAGE instruction.
(4): If endpoint 2 uses shared compression, it compares the state
     identifier values in the "returned SigComp parameters"
     information with the value it has calculated for the current
     decompressed message received from endpoint 1. If there is a
     match then endpoint 2 uses the shared state together with the
     state it would normally use if shared compression is not
     supported to compress the next message.
(5): The UDVM bytecode that will use the shared state (state(M)) in
     the decompression process at decompressor 1 is either carried
     in the compressed message, or can be retrieved by decompressor 1
     from a state already saved at endpoint 1.

5.3. Maintaining State Data Across Application Sessions

Usually, signaling protocols (e.g., SIP) employ the concept of
sessions. However, from the compression point of view, the messages
sent by the same source contain redundancies beyond the session
boundary. Consequently, it is natural to maintain the state data
from the same source across sessions so that high performance can be

achieved and maintained, with the overhead amortized over a much
longer period of time than one application session.

Maintaining states across application sessions can be achieved simply
by making the lifetime of a compartment longer than the time duration
of a single application session. Note that the states here are
referring to those stored on a per-compartment basis, not the locally
available states that are stored on a global basis (i.e., not
compartment specific).

5.4. Use of User-Specific Dictionary

The concept of the user-specific dictionary is based on the
observation that for protocols such as SIP, a given user/device
combination will produce some messages containing fields that are
always populated with the same data.

Take SIP as an example. Capabilities of the SIP endpoints are
communicated during session initiation, and tend not to change unless
the capabilities of the device change. Similarly, user-specific
information such as the user's URL, name, and e-mail address will
likely not change on a frequent basis, and will appear regularly in
SIP signaling exchanges involving a specific user.

Therefore, a SigComp compressor could include the user-specific
dictionary as part of the initial messages to the decompressor, even
before any time critical signaling messages are generated from a
particular application. This enables an increase in compression
efficiency once the messages start to flow.

Obviously, the user-specific dictionary is a state item that would be
good to have as a cross-session state: see Section 5.3.

5.5. Checkpoint State

The following mechanism can be used to avoid decompression failure
due to reference to a non-existent state. This may occur in three
cases: a) a state is not established at the remote SigComp endpoint
due to the loss of a SigComp message; b) a state is not established
due to insufficient memory; c) a state has been established but was
deleted later due to insufficient memory.

When a compressor sends a SigComp message that will create a new
state on the decompressor side, it can indicate that the newly
created state will be a checkpoint state by setting
state_retention_priority [SIGCOMP] to the highest value sent by the
same compressor. In addition, a checkpoint state must be explicitly
acknowledged by the receiving decompressor to the sending compressor.

Consider Figure 3. An event flow for this kind of state management
can be as follows:

(1): Compressor 1 saves e.g., state(A), which it would like to have
     as a checkpoint state at decompressor 2.
(2): The UDVM bytecode to indicate the state priority ([SIGCOMP]
     state_retention_priority) of state(A) and initiate a state save
     for state(A) is either carried in the compressed message, or can
     be retrieved by decompressor 2 from a state already saved at
     endpoint 2.
(3): A save state request for state(A) is passed to the state handler
     using the END-MESSAGE instruction, including the indication of
     the state priority. The application grants the saving of
     state(A): see [SIGCOMP].
(4): An acknowledgement for state(A) (the checkpoint state) is
     returned to endpoint 2 using one of the mechanisms described in
     Section 5.1.

Note: To avoid using a state that has been deleted due to
insufficient memory a compressor must keep track of the memory
available for saving states at the remote endpoint. The SigComp
parameter state_memory_size which is announced by the SigComp
feedback mechanism can be used to infer if a previous checkpoint
state has been deleted (by a later checkpoint state creation request)
due to lack of memory.

5.6. Implicit Deletion for Dictionary Update

Usually a state consists of two parts: UDVM bytecode and dictionary.
When dynamic compression is applied, new content needs to be added to
the dictionary. To keep an upper bound of the memory consumption
such as in the case for a low end mobile terminal, existing content
of the dictionary must be deleted to make room for the new content.

Instead of explicitly signaling which parts of the dictionary need to
be deleted on a per message basis, an implicit deletion approach may
be applied. Specifically, some parts of the dictionary are chosen to
be deleted according to a well-defined algorithm that is known and
applied in the same way at both compressor and decompressor. For
instance, the algorithm can be part of the predefined UDVM bytecode
that is agreed between the two SigComp endpoints. As input to the
algorithm, one provides the total number of bytes to be deleted. The
algorithm then specifies which parts of the dictionary are to be
deleted. Since the same algorithm is applied at both SigComp
endpoints, there is no need for explicit signaling on a per message
basis. This may lead to higher compression efficiency due to the
avoidance of

signaling overhead. It also means more robustness as there are no
signaling bits on the wire that are subject to possible transmission
errors/losses.

6. Implications on SigComp

The extended features will have implications on the SigComp messages
sent between the compressor and its remote decompressor, and on how
to interpret e.g., returned SigComp parameters [SIGCOMP]. However,
except for the mandatory bytes of the SigComp messages [SIGCOMP], the
final message formats used are implementation issues. Note that an
implementation that does not make use of explicit acknowledgements
and/or shared compression is not affected, even if it receives this
kind of feedback.

6.1. Implications on SigComp Messages

To support the extended features, SigComp messages must carry the
indications and information addressed in Section 5. For example to
support shared compression and explicit acknowledgements the SigComp
messages need to convey the following information:

- The acked_state_id as described in Sections 2 and 5.1.
- The shared_state_id as described in Sections 2 and 5.2.

Figure 4 depicts the format of a SigComp message according to
[SIGCOMP]:

```
   0   1   2   3   4   5   6   7       0   1   2   3   4   5   6   7
 +---+---+---+---+---+---+---+---+   +---+---+---+---+---+---+---+---+
 | 1   1   1   1   1 | T |  len  |   | 1   1   1   1   1 | T |   0   |
 +---+---+---+---+---+---+---+---+   +---+---+---+---+---+---+---+---+
 |                               |   |                               |
 :     returned feedback item    :   :     returned feedback item    :
 |                               |   |                               |
 +---+---+---+---+---+---+---+---+   +---+---+---+---+---+---+---+---+
 |                               |   |            code_len           |
 :     partial state identifier  :   +---+---+---+---+---+---+---+---+
 |                               |   |   code_len    |  destination  |
 +---+---+---+---+---+---+---+---+   +---+---+---+---+---+---+---+---+
 |                               |   |                               |
 :     remaining SigComp message :   :     uploaded UDVM bytecode    :
 |                               |   |                               |
 +---+---+---+---+---+---+---+---+   +---+---+---+---+---+---+---+---+
                                     |                               |
                                     :     remaining SigComp message :
                                     |                               |
                                     +---+---+---+---+---+---+---+---+
```
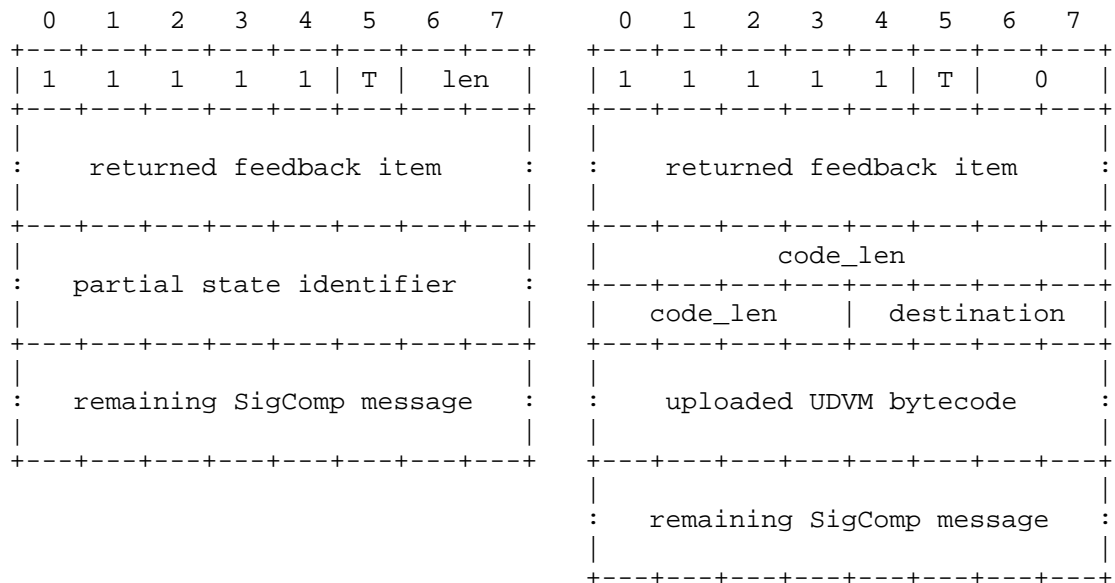
Figure 4. Format of a SigComp message

The format of the field "remaining SigComp message" is an
implementation decision by the compressor which supplies the UDVM
bytecode. Therefore there is no need to specify a message format to
carry the information necessary for the extended features described
in this document.

Figure 5 depicts an example of what the "remaining SigComp message"
with support for shared compression and explicit acknowledgements,
could look like. Note that this is only an example; the format is an
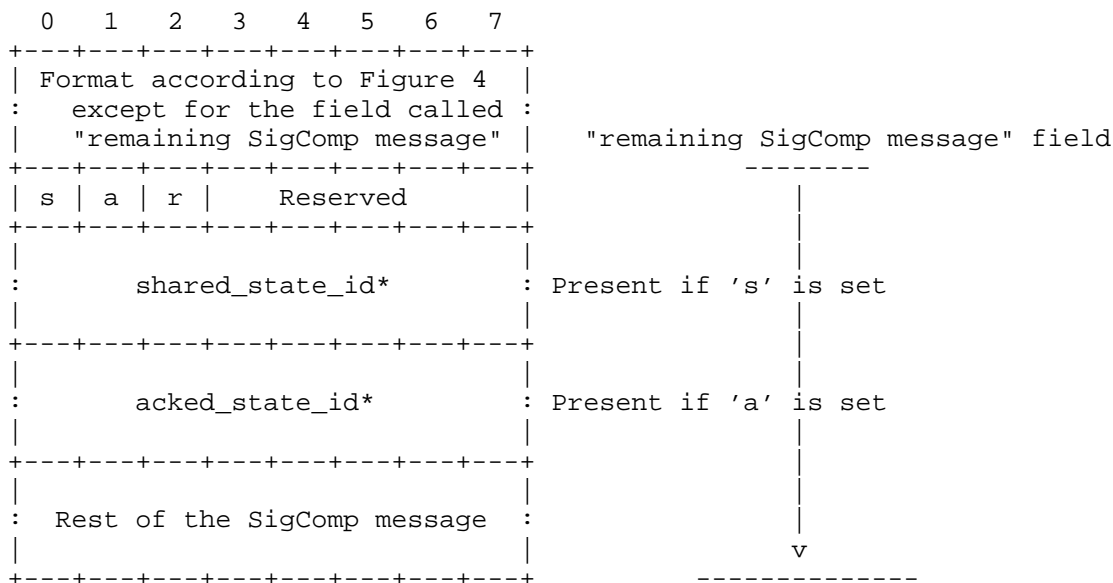implementation decision.

```
   0   1   2   3   4   5   6   7
 +---+---+---+---+---+---+---+---+
 | Format according to Figure 4  |
 :   except for the field called :
 |   "remaining SigComp message" |     "remaining SigComp message" field
 +---+---+---+---+---+---+---+---+            --------
 | s | a | r |     Reserved      |                   |
 +---+---+---+---+---+---+---+---+                   |
 |                               |                   |
 :         shared_state_id*      :   Present if 's' is set
 |                               |                   |
 +---+---+---+---+---+---+---+---+                   |
 |                               |                   |
 :         acked_state_id*       :   Present if 'a' is set
 |                               |                   |
 +---+---+---+---+---+---+---+---+                   |
 |                               |                   |
 :  Rest of the SigComp message  :                   |
 |                               |                   v
 +---+---+---+---+---+---+---+---+            --------------
```

        Figure 5. Example of SigComp message for some of the extended
                features.

   'r' : If set, then a state corresponding to the decompressed
         version of this compressed message (shared state) was saved at
         the compressor.
    * : The length of the shared_state_id and acked_state_id fields
         are of the same length as the partial state identifier.

6.2. Extended SigComp Announcement/Feedback Format

   This section describes how the "returned_SigComp_parameters"
   [SIGCOMP] information is interpreted to provide feedback according to
   Section 5.1 and 5.2.

   The partial_state_identifiers correspond to the hash_value for states
   that have been established at the remote endpoint after the reception
   of SigComp messages, i.e., these are acknowledgements for established
   states and may be used for compression. The
   partial_state_identifiers may also announce "global state" that is
   not mapped to any particular compartment and is not established upon
   the receipt of a SigComp message.

It is up to the implementation to deduce what kind of state each
partial_state_identifier refers to, e.g., an acknowledged state or a
shared state. In case a SigComp message that includes state
identifiers for shared states and/or acknowledged states is received
by a basic SigComp implementation, these identifiers will be ignored.

The I-bit of the requested feedback format is provided to switch off
the list of locally available state items. An endpoint that wishes
to receive shared_state_id must not set the I-bit to 1. The endpoint
storing shared states and sending the list of locally available
states to its remote endpoint must be careful when taking the
decision whether to exclude or include different types of the locally
available states (i.e., shared states or states of e.g., well-known
algorithms) from/to the list.

6.3. Acknowledgement Optimization

If shared compression is used between two endpoints (see Figure 1)
then there exists an optimization, which, if implemented, makes an
acked_state_id in the SigComp message unnecessary:

Compressor 1 saves a shared state(M), which is the uncompressed
version of the current compressed message (message m) to be sent.
Compressor 1 also sets bit 'r' (see Figure 5), to signal that
state(M) can be used by endpoint 2 in the compression process. The
acked_state_id for state(S), which was created at endpoint 2 upon the
decompression of message m, may not have to be explicitly placed in
the compressed messages from compressor 2 if the shared state(M) is
used in the compression process.

When endpoint 1 notices that shared state(M) is requested by
decompressor 1, it implicitly knows that state(S) was created at
endpoint 2. This follows since:

* Compressor 1 has instructed decompressor 2 to save state(S).
* The indication of shared state(M) would never have been received by
  compressor 2 if state(S) had not been successfully saved, because
  if a state save request is denied then the corresponding
  announcement information is discarded by the state handler.

Note: Endpoint 1's state handler must maintain a mapping between
state(M) and state(S) for this optimization to work.

Note: The only state that is acknowledged by this feature is the
state that was created by combining the state used for compression of
the message and the message itself. For any other case the
acked_state_id has to be used.

Note: There is a possibility that state(S) is discarded due to lack
of state memory even though the announcement information is
successfully forwarded. This possibility must be taken into account
(otherwise a decompression failure may occur); this can be done by
using the SigComp parameter state_memory_size which is announced by
the SigComp feedback mechanism. The endpoint can use this parameter
to infer if a state creation request has failed due to lack of
memory.

7. Security Considerations

   The features in this document are believed not to add any security
   risks to the ones mentioned in [SIGCOMP].

8. IANA Considerations

   This document does not require any IANA involvement.

9. Acknowledgements

   Thanks to Carsten Bormann, Christopher Clanton, Miguel Garcia, Lars-
   Erik Jonsson, Khiem Le, Mats Nordberg, Jonathan Rosenberg and Krister
   Svanbro for valuable input.

10. Intellectual Property Right Considerations

   The IETF has been notified of intellectual property rights claimed in
   regard to some or all of the specification contained in this
   document. For more information consult the online list of claimed
   rights.

11. References

   [SIP]   Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston,
               A., Peterson, J., Sparks, R., Handley, M. and E.
               Schooler, "SIP: Session Initiation Protocol", RFC 3261,
               June 2002.

   [SIGCOMP]   Price R., Bormann, C., Christoffersson, J., Hannu, H.,
               Liu, Z. and J. Rosenberg, "Signaling Compression
               (SigComp)", RFC 3320, January 2003.

12. Authors' Addresses

   Hans Hannu
   Box 920
   Ericsson AB
   SE-971 28 Lulea, Sweden

   Phone: +46 920 20 21 84
   EMail: hans.hannu@epl.ericsson.se

   Jan Christoffersson
   Box 920
   Ericsson AB
   SE-971 28 Lulea, Sweden

   Phone: +46 920 20 28 40
   EMail: jan.christoffersson@epl.ericsson.se

   Stefan Forsgren

   EMail: StefanForsgren@alvishagglunds.se

   Ka-Cheong Leung
   Department of Computer Science
   Texas Tech University
   Lubbock, TX 79409-3104
   United States of America

   Phone: +1 806 742-3527
   EMail: kcleung@cs.ttu.edu

   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
   Roke Manor Research Ltd
   Romsey, Hants, SO51 0ZN, United Kingdom

   Phone: +44 1794 833681
   EMail: richard.price@roke.co.uk

13. Full Copyright Statement

Acknowledgement