Signalling Compression (SigComp)

Introduction
Architecture
Implementation Requirements

adare GmbH
www.adare.de

Kai Tetzlaff
mail: kai.tetzlaff@adare.de

July 2005

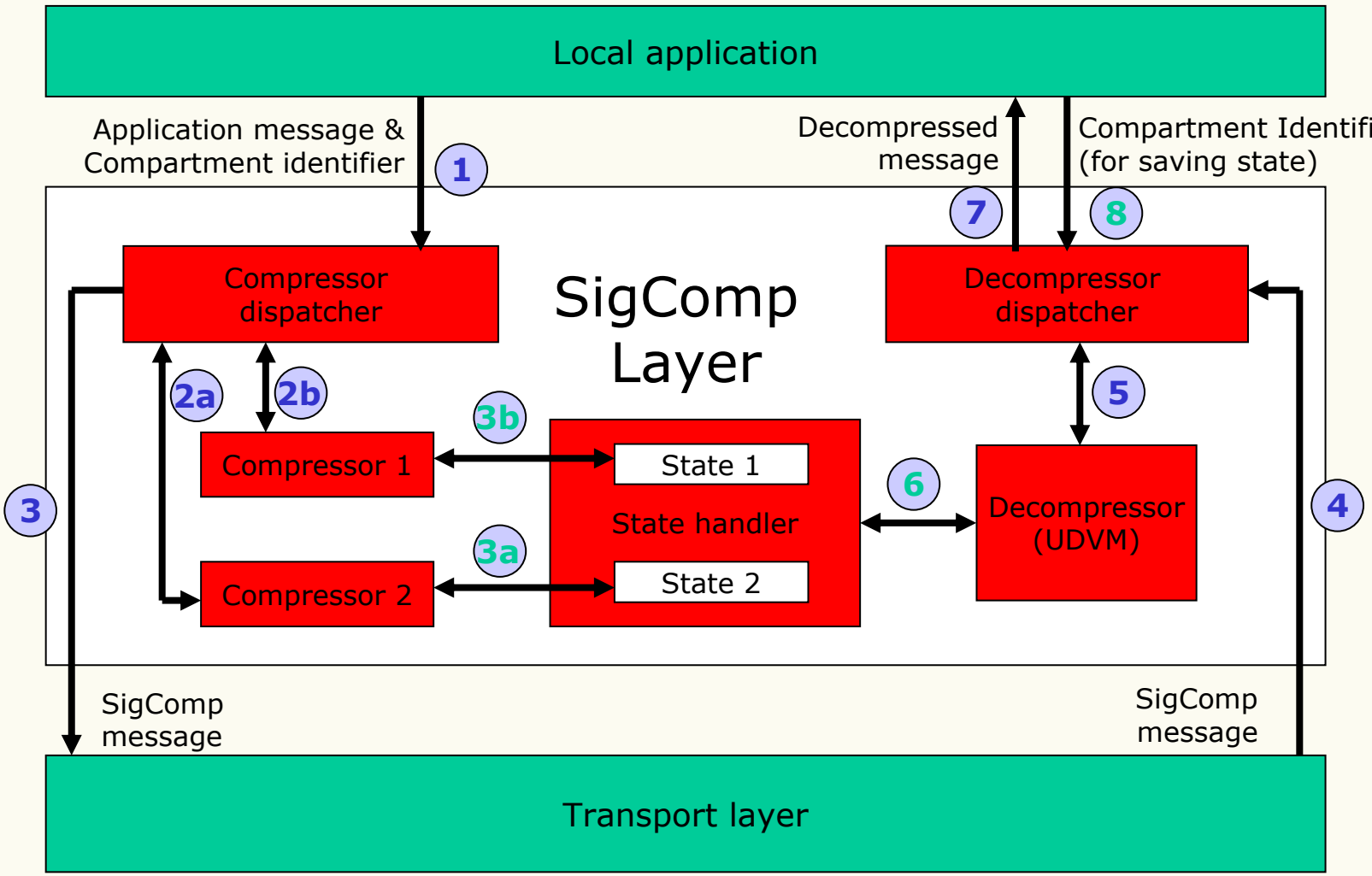
Agenda

- What is SigComp?
- Introduction
- Architecture
- UDVM
- SIP static SigComp dictionary
- SigComp implementation requirements
- SigComp performance

What is Signalling Compression (SigComp)?

- SigComp provides functionality for compressing messages of text based protocols
- Specified by Internet Engineering Task Force (IETF) ROHC working group
- SigComp has been designed to be application agnostic so that it can be applied to any text based application protocol
- SIP is the primary driver for SigComp development
- SigComp attempts to reduce the call setup time through the compression of SIP signalling messages
- SigComp is a mandatory part of 3GPP release 5 IP Multimedia Subsystem (IMS) – a new core network domain controlling voice and multimedia calls and sessions as well as interconnection to other networks
- Interface between application and SigComp is not specified.

Architecture Overview



Characteristics

- Compression in one direction does not imply compression in the reverse direction
- Appears as a new transport
  - similar behavior to the original transport
- Compression algorithm independent
- No application level negotiation
  - Predefined minimum capabilities
  - Configuration and algorithms are pushed, announced, or known in advance
- Minimal base protocol
  - Only one message type
  - Optional feedback mechanism

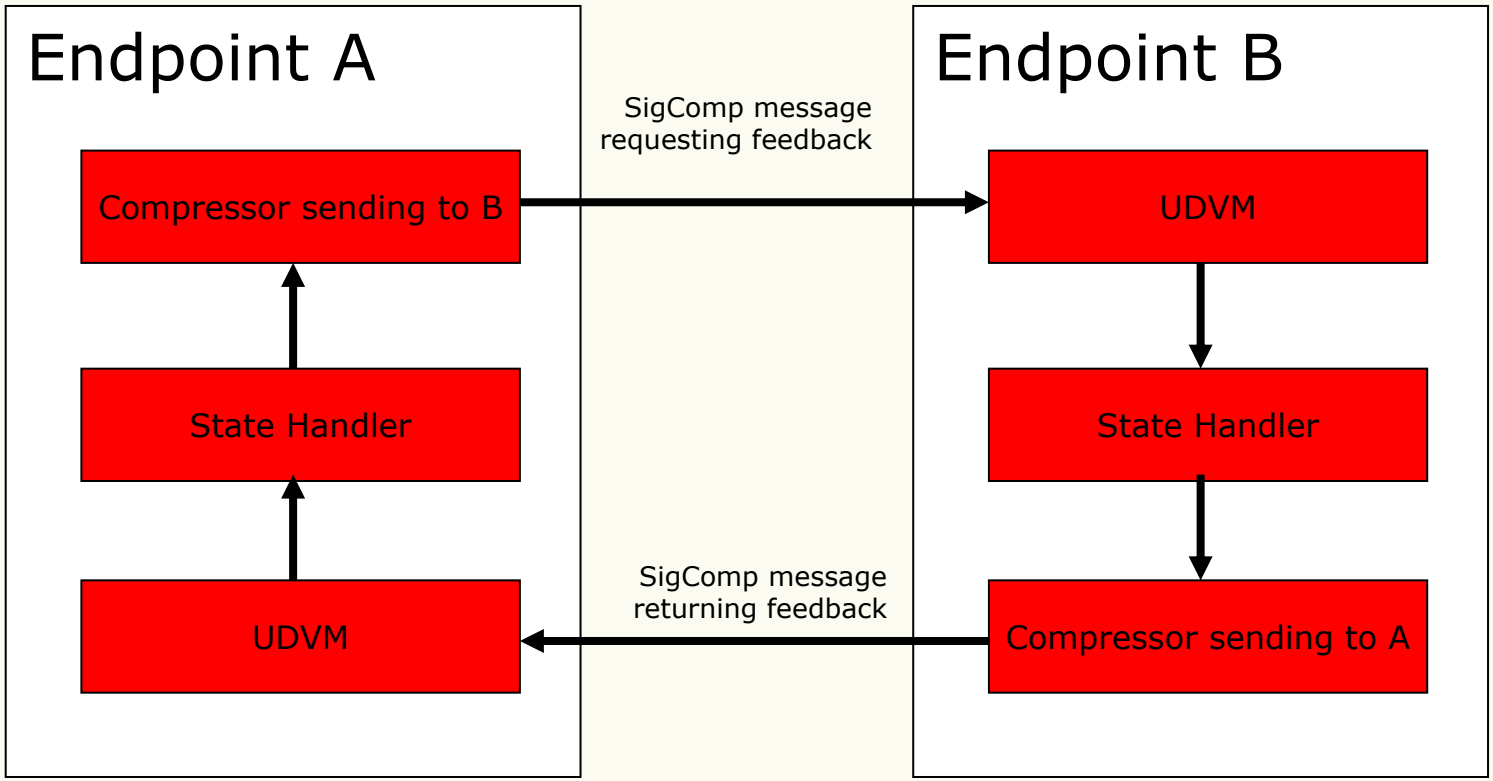
Key Concepts and Terms

- **Dispatcher:** Interface towards compressor, decompressor and application
- **Compressor:** Compress application messages
- **UDVM:** Universal Decompression Virtual Machine (concept similar to Java VM)
- **State Handler:** State management
- **Stateless Endpoint:** All information required for decompression is contained in message
- **Stateful Endpoint:** Stores session data to improve compression ratio
- **State:** ”snapshot” of UDVM memory
- **Compartment:** bundles messages e.g. to same session or connection

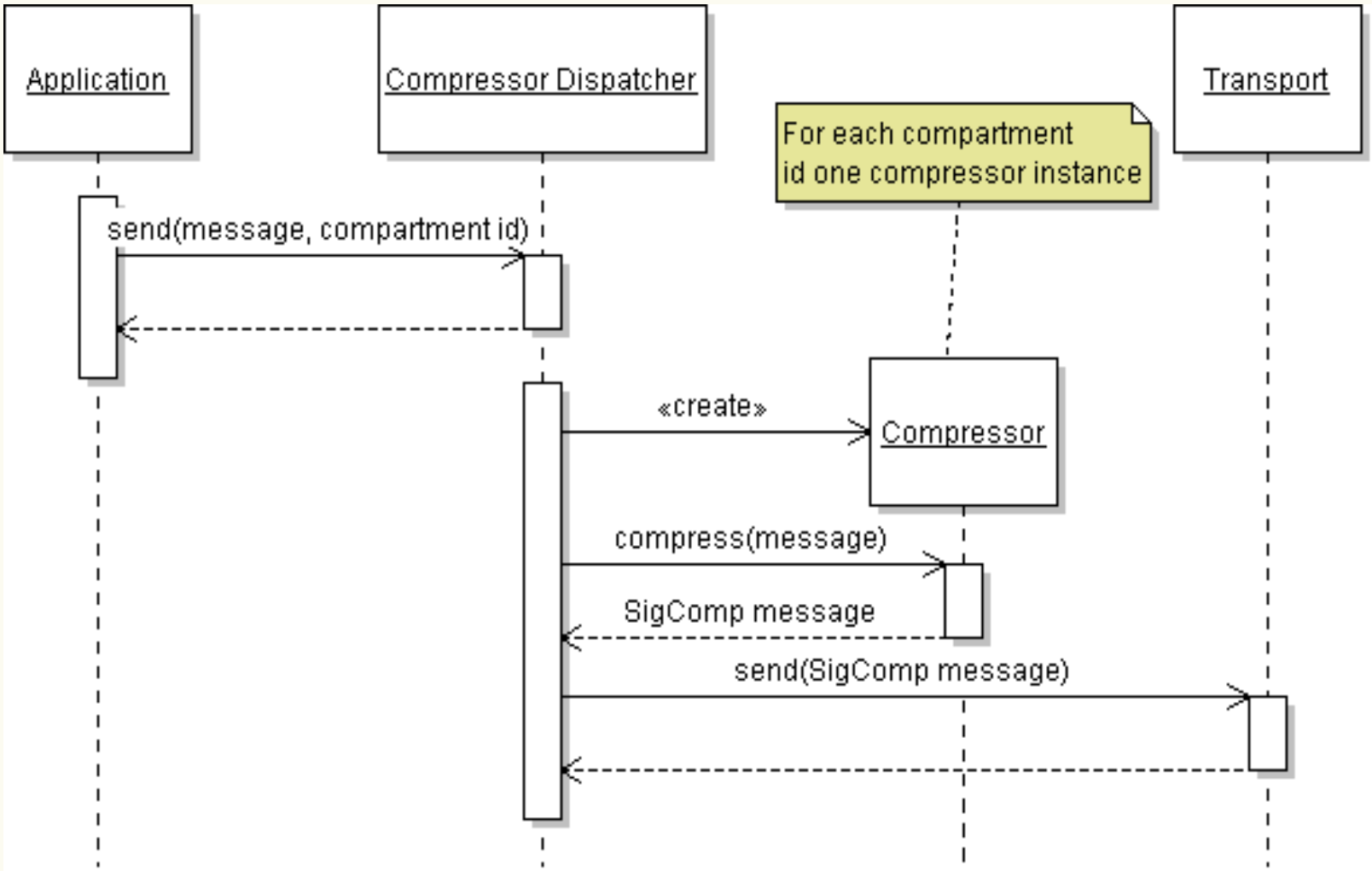
SigComp architecture

Feedback Mechanism

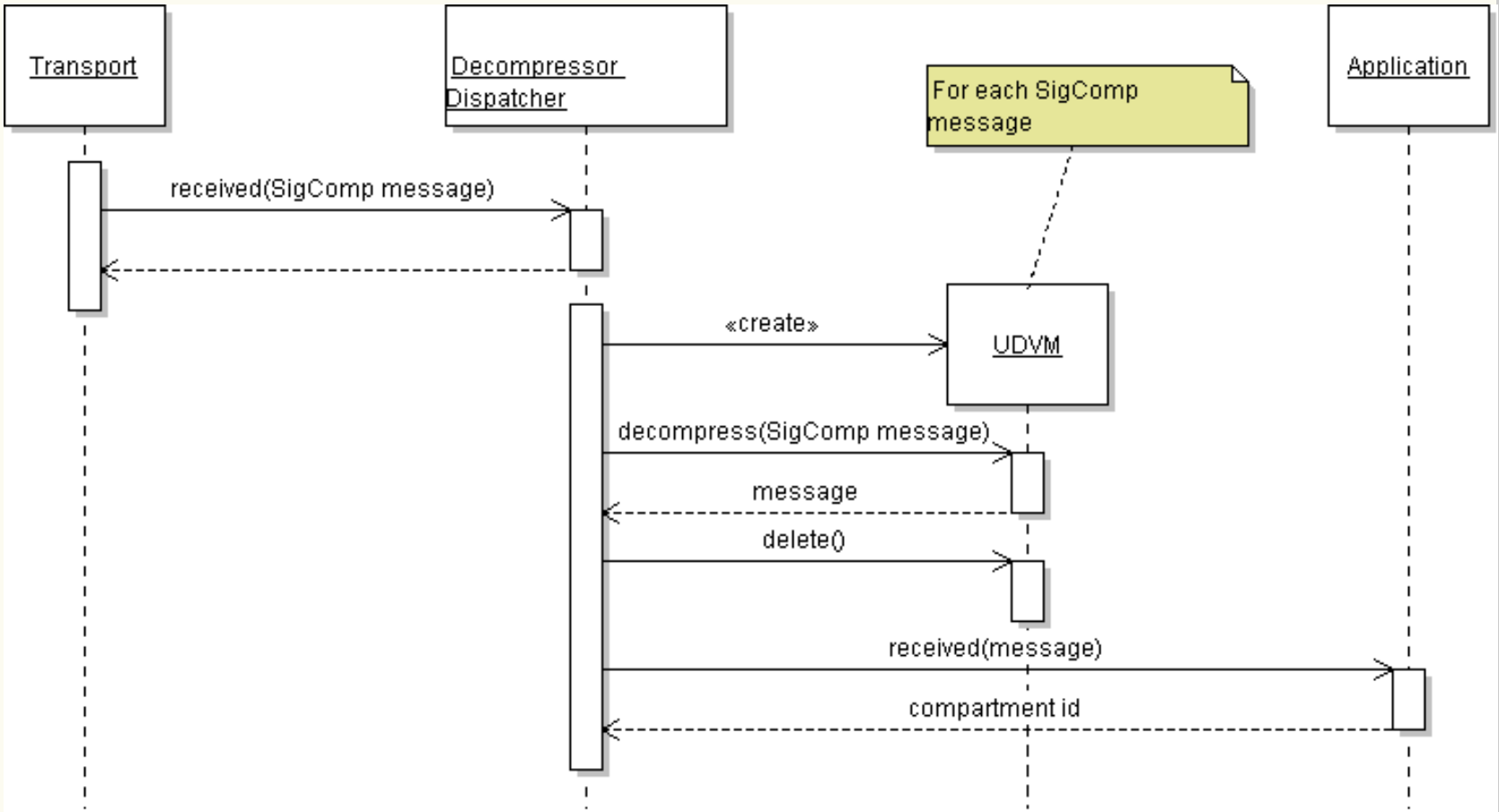
Feedback mechanism – always piggybacked
Used e.g. to announce available states, capabilities, and acknowledge saved states



Sending an Application Message



Receiving an Application Message



Compressor

- Compresses each application message to exactly one SigComp message.
- The compressor and its compression algorithm is hard coded.
- Requested feedback data is passed from the state handler to the compressor and returned with the next SigComp message.
- A compressor **MUST** be certain that all of the data needed to decompress a SigComp message is available at the receiving endpoint. One way to ensure this is to send all of the needed information in every SigComp message (including bytecode to decompress the message).

Compression Types

- Per-message compression (stateless endpoint)
  - Compression that does not reference data from previous messages. SigComp can decompress a message of this type using only the application-defined parameters and the data in the message itself.
- Dynamic compression (stateful endpoint)
  - Compression relative to messages sent prior to the current compressed message. SigComp stores and retrieves this data using the state handler.

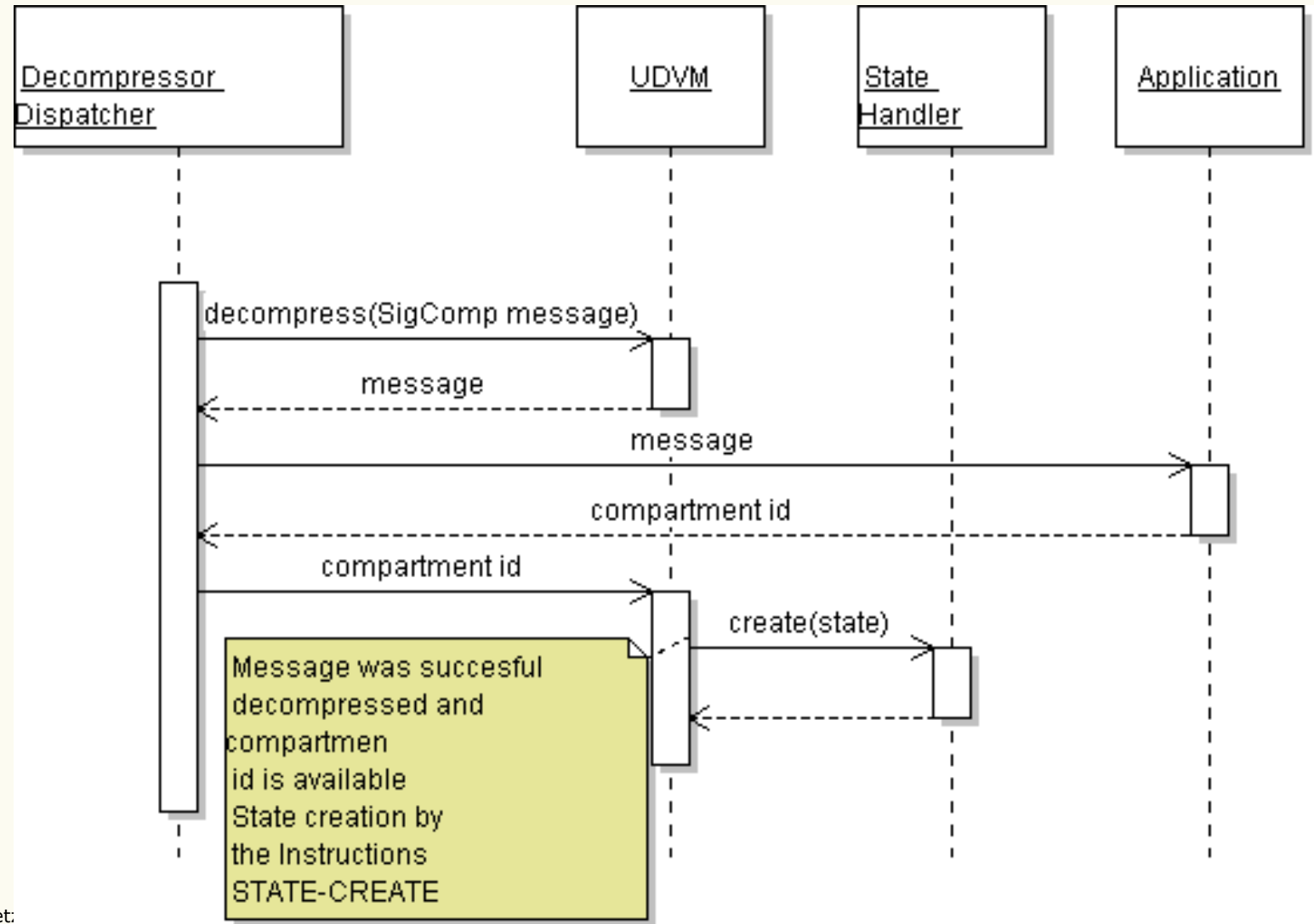
State

- State consists of data/information retained between messages
- State can be loaded into the UDVM
- There are 2 kinds of state:
  - Locally available state (for example the SIP static dictionary [5])
  - State received as part of a SigComp message
- State improves the compression ratio as the same information does not have to be uploaded on a per-message basis.

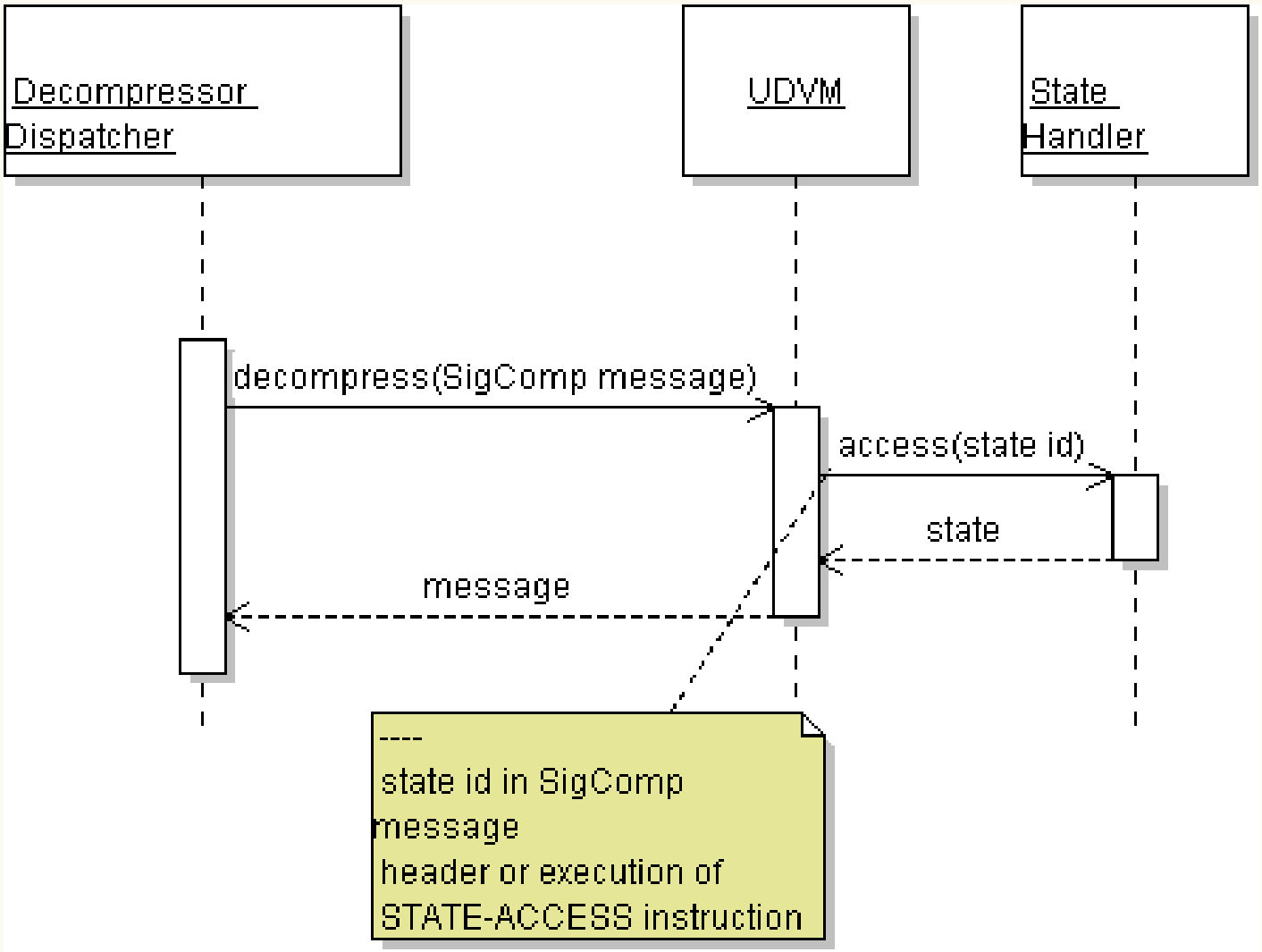
State Handler

- Retain information between received SigComp messages.
- Improves compression ratio by providing information about previously received messages (parts of messages must only be uploaded once).
- The UDVM can only create state with the UDVM instruction **STATE-CREATE** after complete decompression of a SigComp message and after the application has returned a compartment id.
- A State identifer is created for accessing state.
- The UDVM can access state by a supplied state identifier with a SigComp message or by executing bytecode with the **STATE-ACCESS** instruction.
- Manages the state memory on a per-compartment basis.

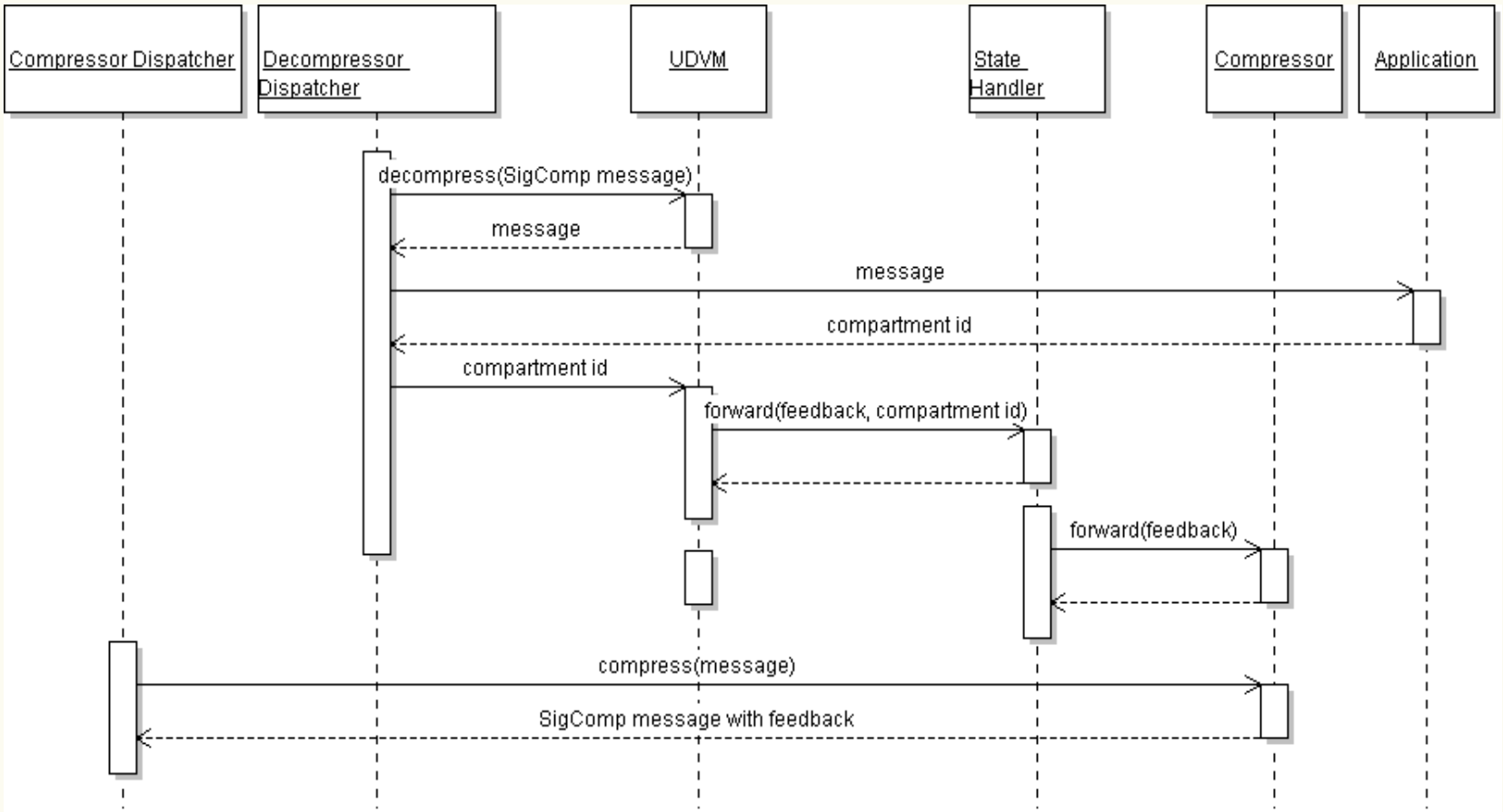
Stateful Endpoint: State Create



Stateful Endpoint: State Access



Providing Feedback



Message Format (1/3)

- **How to identify a SigComp message?**
- The 5 MSBs of the first byte of an SigComp message are set to one.

MSB							LSB
0	1	2	3	4	5	6	7
1	1	1	1	1	X	X	X

- This does not occur in UTF-8 encoded text messages.
- => possibility to multiplex SigComp and other messages on the same port.
- It is also possible to forward SigComp messages to a new special SigComp port.

Message Format (2/3)

- **A SigComp message takes one of two forms:**

0	1	2	3	4	5	6	7
1	1	1	1	1	T	len	
returned feedback item (optional)							
partial state identifier							
Application message/data (SDU)							

Message format 1:
Used for stateful communication. Byte code for decompression is available in locale state item and must therefore be accessed by the partial state identifier.

0	1	2	3	4	5	6	7
1	1	1	1	1	T	0	
returned feedback item (optional)							
code_len							
code_len				destination			
uploaded UDVM bytecode							
Application message/data (SDU)							

Message format 2:
Used for stateless communication. Used for passing the byte code to the remote peer.

T: If the T-Bit is set to 1 the SigComp message contains a returned feedback item.

Message Format (3/3)

- len: Determines the length of the partial state identifier
  - 00: no state identifier (stateless)
  - 01: 6 octets
  - 10: 9 octets
  - 11: 12 octets
- Original state identifier is 20 Byte long
- Feedback Item

Single octet

0	1	2	3	4	5	6	7
0	returned_feedback_field						

Multi octet

0	1	2	3	4	5	6	7
1	returned_feedback_length (size in byte 0-127)						
returned_feedback_field							

Message Format details (1/2)

The following parameters must be initialized with message format 1

0							7	8							15	
UDVM_memory_size (size in bytes modulo 2^16, 0 if memory size is 65535)																0 - 1
cycles_per_bit (16, 32, 64, 128)																2 - 3
SigComp_version (2 byte value)																4 - 5
partial_state_ID_length (length of partial state identifier)																6 - 7
state_length (number of bytes retrieved from the state item)																8 - 9
Reserved (initialized to 0)																10 - 31

Message format details (2/2)

The following parameters must be initialized with message format 2

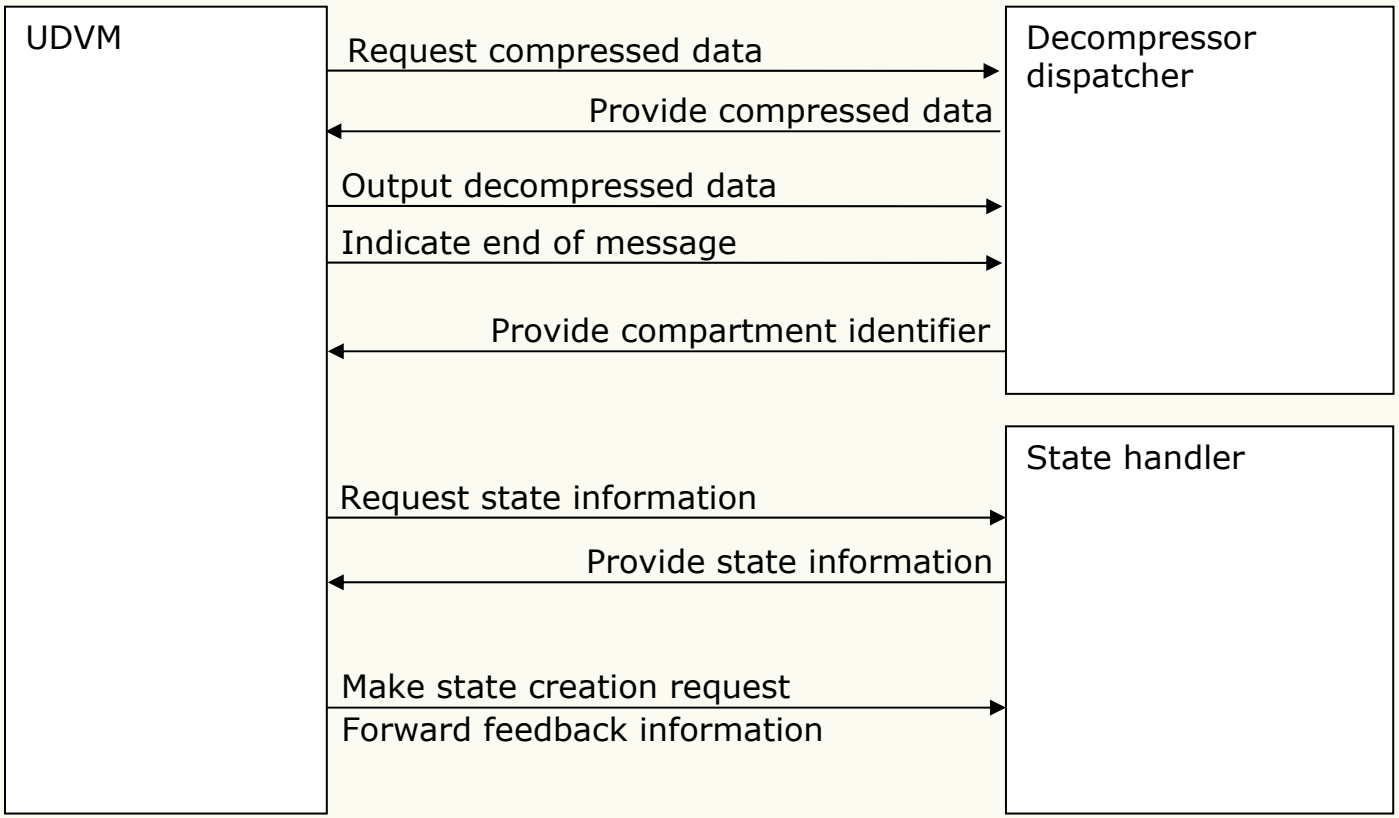
0							7	8							15	
UDVM_memory_size (size in bytes modulo 2^16, 0 if memory size is 65535)																0 - 1
cycles_per_bit (16, 32, 64, 128)																2 - 3
SigComp_version (2 byte value)																4 - 5
No initialization																6 - 7
No initialization																8 - 9
Reserved (initialized to 0)																10 - 31

Universal Decompression Virtual Machine (UDVM)

UDVM

- A Virtual Machine optimized for decompression algorithms.
  - The bytecode has 36 instructions
- A new UDVM gets instantiated and initialized for each received SigComp message
- Bytecode programs are small
  - LZW code is 66 bytes & DEFLATE <300 bytes
- Flexibility in how to compress a given application message
- Does not add significant extra processing or memory requirements compared to running a fixed preprogrammed compression algorithm.

UDVM – interfaces



UDVM – instructions

- Low and high-level instructions
- All instructions are encoded as a single byte to indicate the instruction type, followed by 0 or more bytes containing the operands required by the instruction.
  - ADD ($operand_1, %operand_2)

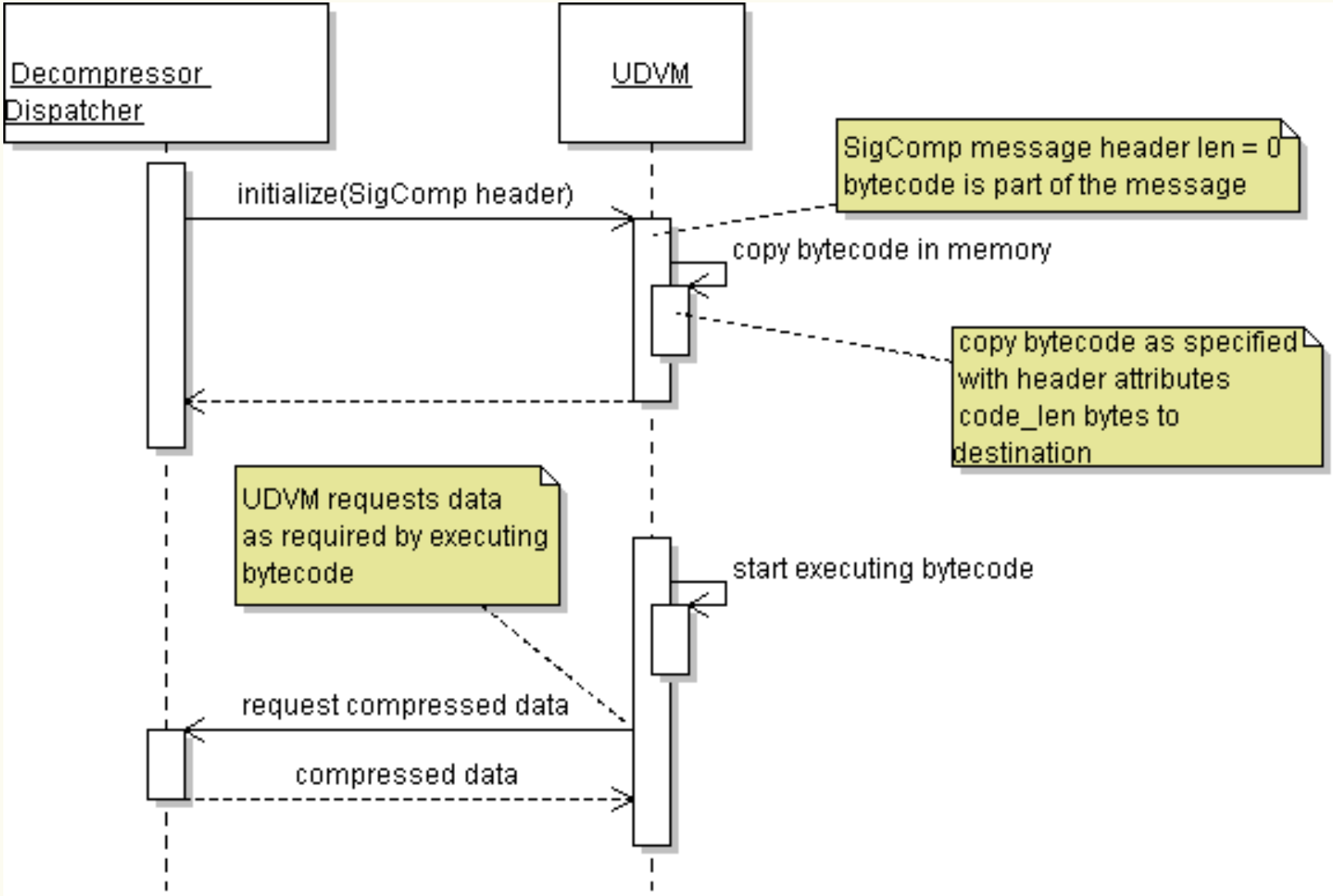
UDVM – operands

- The literal (#)
  - encodes a constant integer
- The reference ($)
  - is interpreted as the UDVM memory address containing the actual value of the operand.
- The multitype (%)
  - is used to encode both actual values and memory addresses. Offers efficient encoding for small integer values (both positive and negative) and for powers of 2.
- The address (@)
  - is decoded as a multitype operand followed by a further step: the memory address of the UDVM instruction containing the address operand is added to obtain the correct operand value. ensure that the UDVM bytecode is position independent code

UDVM – some instructions

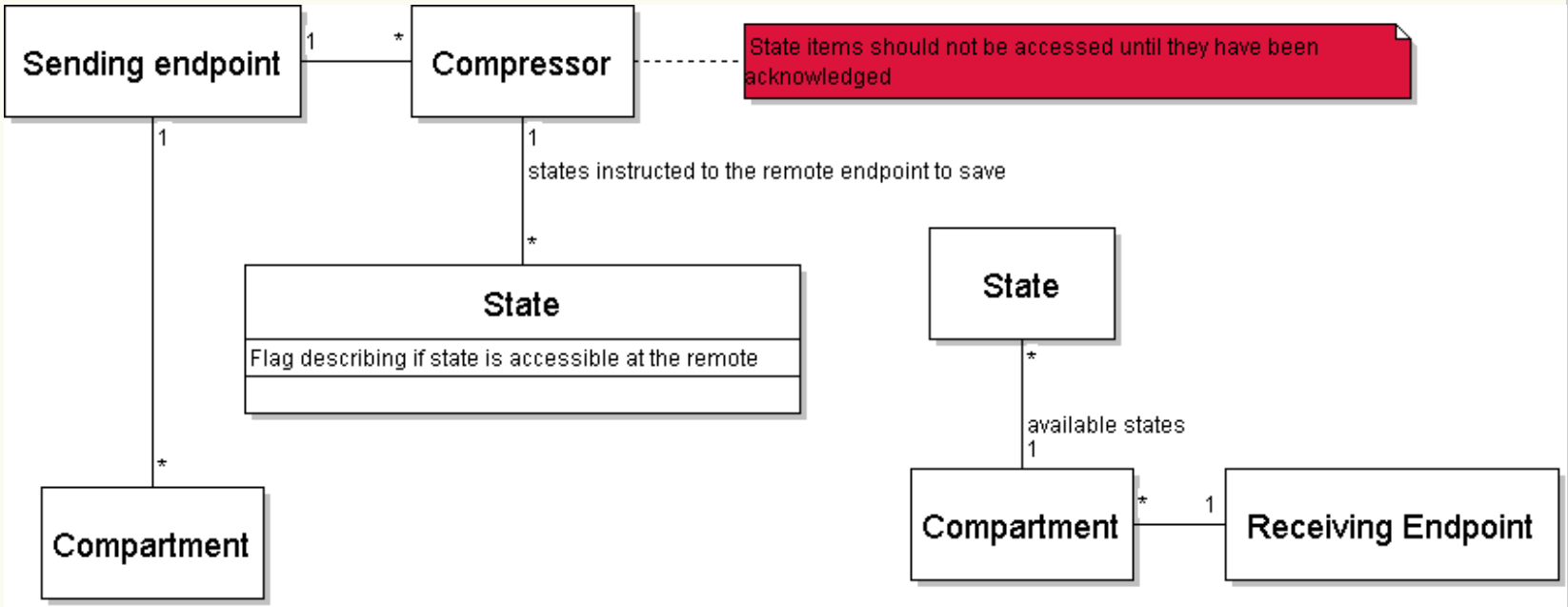
Instruction:	Bytecode value:	Cost in UDVM cycles:
DECOMPRESSION-FAILURE	0	1
AND	1	1
MULTIPLY	8	1
DIVIDE	9	1
REMAINDER	10	1
SORT-ASCENDING	11	1 + k * (ceiling(log2(k)) + n)
SORT-DESCENDING	12	1 + k * (ceiling(log2(k)) + n)
COPY	18	1 + length
COPY-LITERAL	19	1 + length
COPY-OFFSET	20	1 + length
MEMSET	21	1 + length
JUMP	22	1
COMPARE	23	1
CALL	24	1
RETURN	25	1
SWITCH	26	1 + n
CRC	27	1 + length
INPUT-BYTES	28	1 + length
INPUT-BITS	29	1
INPUT-HUFFMAN	30	1 + n

Example: SigComp message received by stateless endpoint

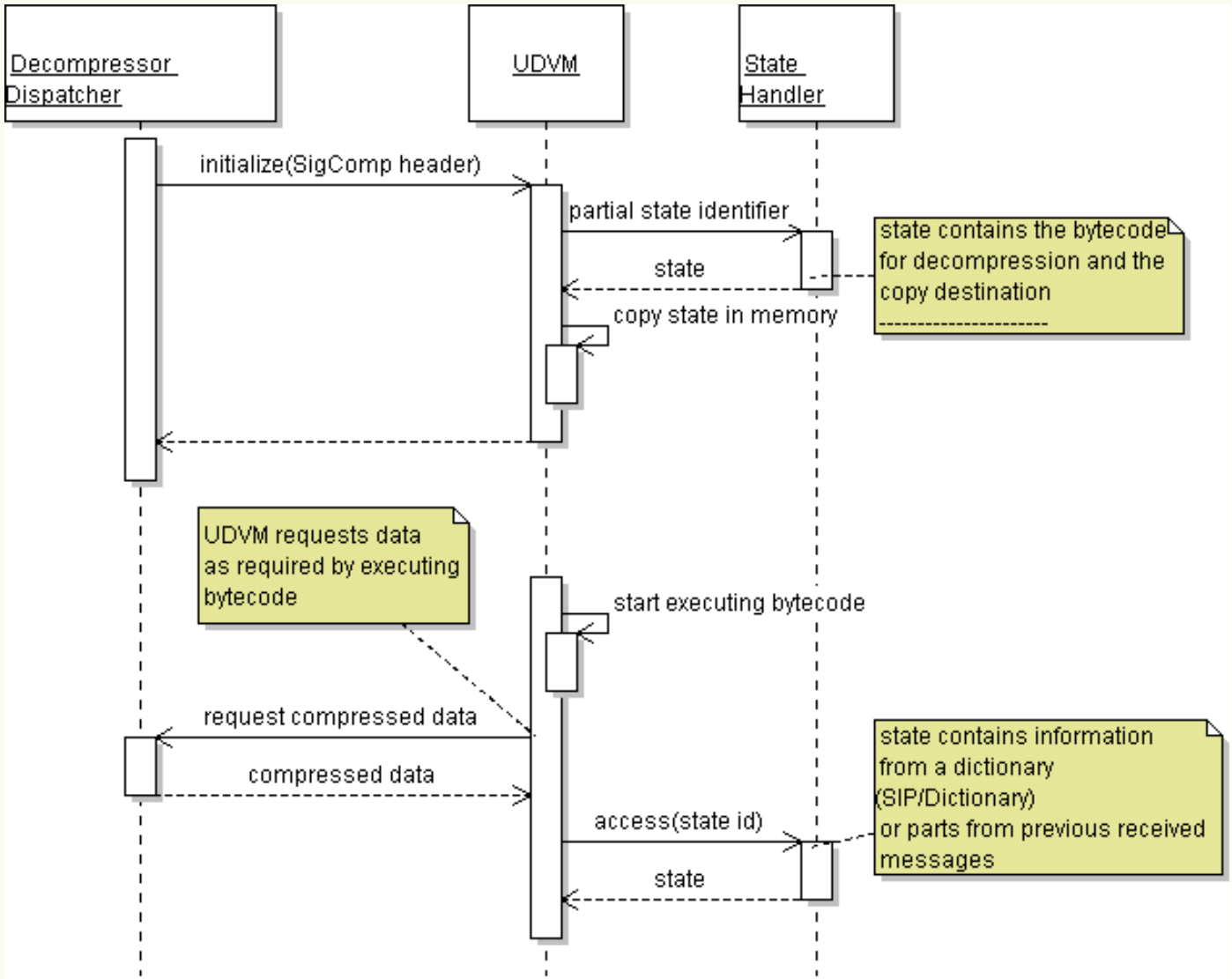


Requirement for stateful endpoints

The usage of state is only allowed if the compressor is sure about the state availability at the remote endpoint.



Example: SigComp message received by stateful endpoint



Example: Decompression with "Simplified LZ77" algorithm

- Simplified LZ77
  - 0x0f86 0389 8d89 1588 8800 011c 0420 0d13 5051 2222 5051 16f5 2300 0x00bf c086 a08b 06
- SigComp message
  - 0x0154 0001 0168 0001 0165 0001 0120 0001 0152 0001 0165 0001 0173 0x0002 0161 0001 0175 0001 0172 0001 0161 0001 016e 0001 0174 0001 0x0120 0001 0161 0001 020d 0002 0174 0001 0201 0003 0145 0001 016e 0x0001 0164 0001 0120 0001 016f 0001 0166 0001 0211 0005 0155 0001 0x016e 0001 0169 0001 0176 0001 0165 0001 0172 0002 0165 0001 010a 0x0001
- Decompressed message
  - The uncompressed message is "The Restaurant at the End of the Universe\n".

SIP static SigComp dictionary

SIP and SDP static dictionary for SigComp (1)

- RFC 3485
- The static SIP/SDP dictionary constitutes a SigComp state that can be referenced in the first SIP message that the compressor sends out.
- The static SIP/SDP dictionary is a collection of well-known strings that appear in most of the SIP and SDP messages.
- The static dictionary is unique and **MUST** be available in all SigComp implementations for SIP/SDP.

SIP and SDP static dictionary for SigComp(2)

- **State item**
  - state_identifier 0xfbe507dfe5e6aa5af2abb914ceaa05f99ce61ba5
  - state_length 0x12E4
  - state_address 0 (not relevant for the dictionary)
  - state_instruction 0 (not relevant for the dictionary)
  - minimum_access_length 6
  - state_value
- **Extract of the binary dictionary (state_value)**
  - 0000 0d0a 5265 6a65 6374 2d43 6f6e 7461 6374 ..Reject-Contact
  - 0010 3a20 0d0a 4572 726f 722d 496e 666f 3a20 : ..Error-Info:
  - 0020 0d0a 5469 6d65 7374 616d 703a 200d 0a43 ..Timestamp: ..C
  - 0030 616c 6c2d 496e 666f 3a20 0d0a 5265 706c all-Info: ..Repl
  - …
- **Extract of the SIP input Strings**

String	Pr	Off	Len	References
"sip:"	1	0CDD	0004	[3] 19.1.1
"sips:"	3	08AC	0005	[3] 19.1.1
"tel:"	3	08BD	0004	[7] 2.2

  - …
- **Accessing State with the “STATE-ACCESS” instruction**

SIP SigComp implementation requirements

Implementation requirements SIP/SigComp (1)

- **RFC and INTERNET-DRAFTS**
  - Any implementation must follow RFC3320, RFC 3485 and INTERNET-DRAFT "Applying Signalling Compression (SigComp) to the Session initiation Protocol (SIP).
- **SigComp parameters [RFC3320 3.3.1 – 3.3.3]**
  - Minimum decompression_memory_size
    - SIP/SigComp: 8192 bytes [INTERNET-DRAFT]
    - ANY/SigComp: 2048 bytes
  - Minimum state_memory_size
    - SIP/SigComp: 2048 bytes per compartment [INTERNET-DRAFT] (stateful) -> requires compressor too
    - ANY/SigComp: 0 bytes (**stateless**)
  - Minimum cycles_per_bit
    - SIP/SigComp and ANY/SigComp: 16
  - Minimum SigComp_version
    - SIP/SigComp and ANY/SigComp: 0x01
  - Minimum locally available state
    - SIP/SigComp: SIP/SDP-specific static dictionary
    - ANY/SigComp: none

Implementation requirements SIP/SigComp (2)

- **Compartment and State Management**
  - When receiving a request that can establish a dialog a compartment can be associated with the dialog. The Dialog ID can be used as compartment ID. All compartment related states will be closed when the dialog is terminated.

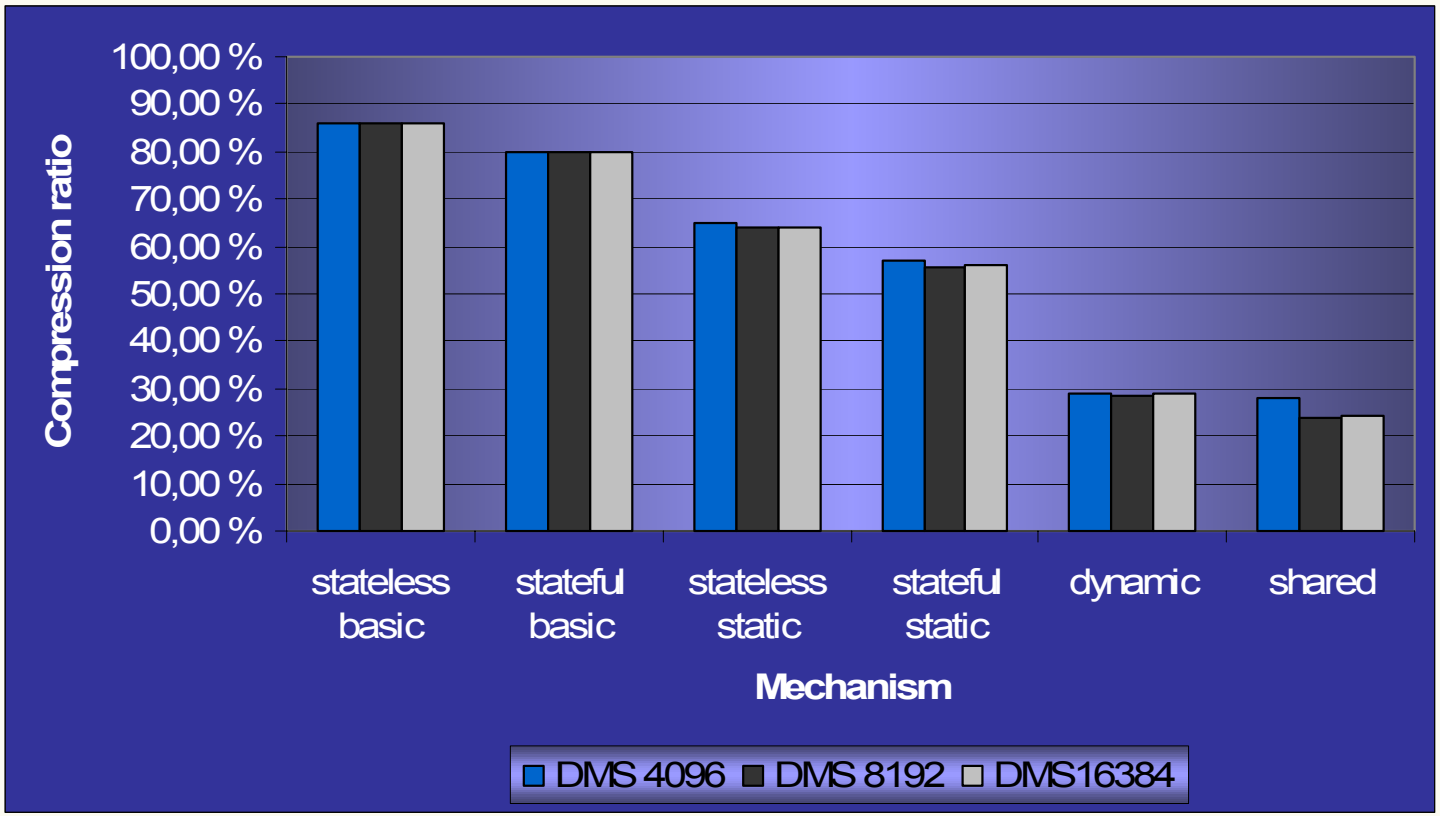
SigComp Performance

(Research by Jouni Mäenpää, Ericsson)

Compression Strategies

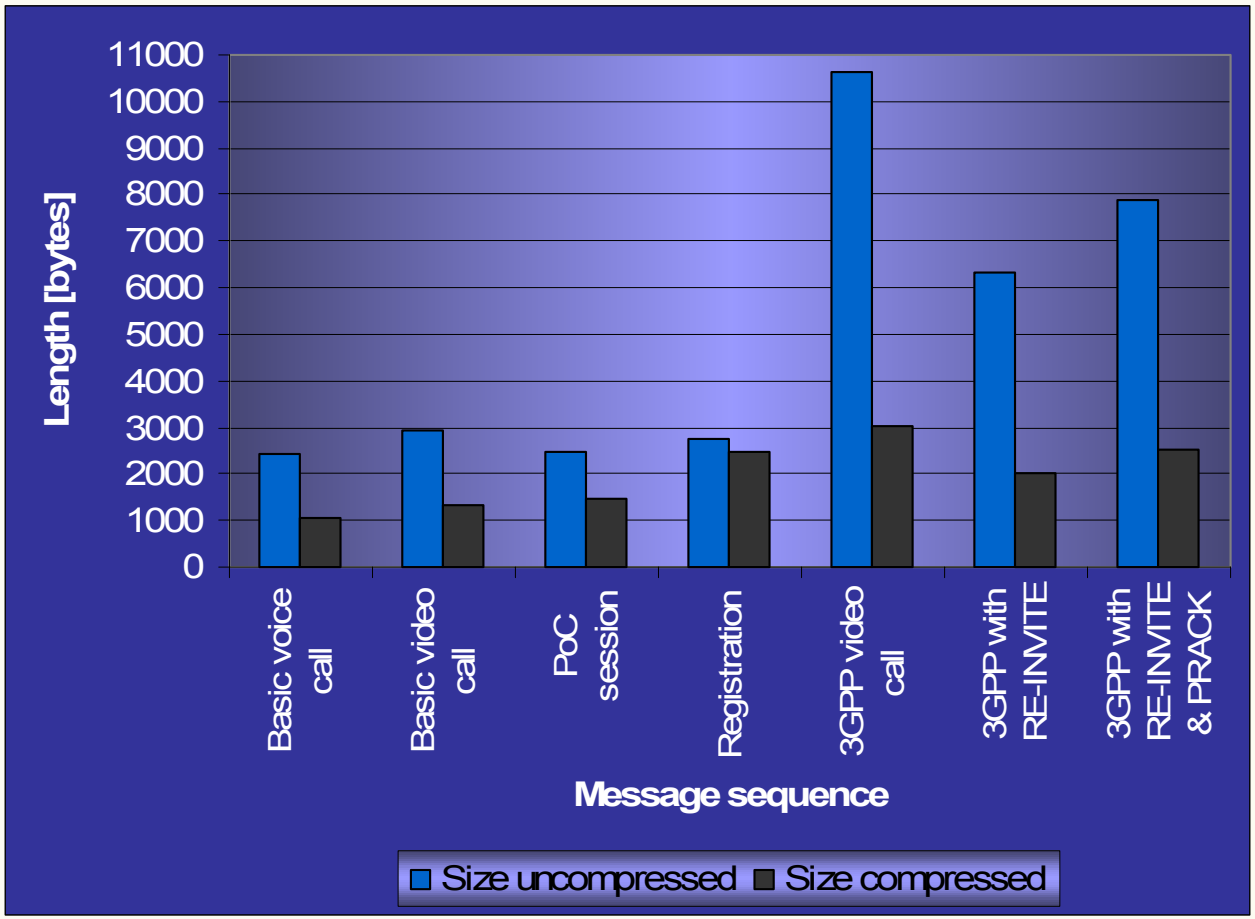
- Basic compression uses message-by-message compression
- In static compression, messages are compressed relative to the static SIP and Session Description Protocol (SDP) dictionary specified in RFC 3485
- In stateful versions of basic and static compressions, the bytecode is saved between messages
- In dynamic compression, previously sent messages are used in the compression process
- In shared compression, also received messages are utilised in the compression process

Compression Ratio vs. Compression Mechanism



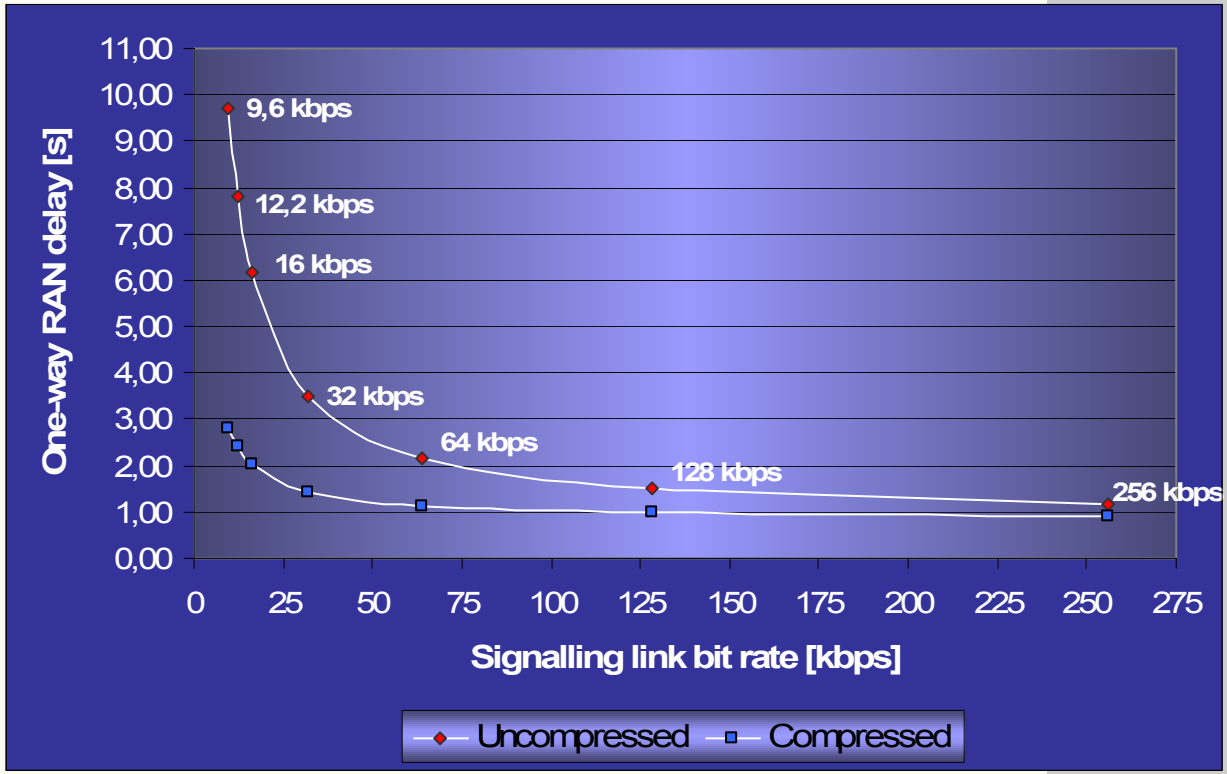
- Basic compression is useless
- Only dynamic and shared compressions offer satisfactory compression ratios

Compression Ratio vs. Application Data Types



RAN delay vs. Link Bandwidth

- The improvement SigComp offers is greatest when the bandwidth of the signalling link is low
- If the link has a high bandwidth (>64 kbps), the improvement SigComp offers may not be enough to justify the use of the protocol



References

- [1] RFC3320 Signalling Compression
- [2] RFC3321 Signalling Compression - Extended Operations
- [3] RFC3322 Signalling Compression - Requirements & Assumptions
- [4] RFC4077 A Negative Acknowledgement Mechanism for Signalling Compression
- [5] RFC4485 The Session Initiation Protocol (SIP) and Session Description Protocol (SDP) Static Dictionary for Signalling Compression (SigComp)
- [6] RFC4486 Compressing the Session Initiation Protocol (SIP)
- [7] draft-ietf-rohc-sigcomp-impl-guide-03.txt Implementer's Guide for SigComp
- [8] draft-ietf-rohc-sigcomp-sip-00.txt Applying Signalling Compression (SigComp) to the Session Initiation Protocol (SIP)
- [9] draft-ietf-rohc-sigcomp-torture-tests-03.txt SigComp Torture Tests
- [10] draft-ietf-rohc-sigcomp-user-guide-04.txt SigComp Users' Guide