

Robust Header Compression (ROHC)

**53rd IETF
Minneapolis, March 2002**

Chairs:

Carsten Bormann <cabo@tzi.org>

Lars-Erik Jonsson <lars-erik.jonsson@ericsson.com>

Mailing List:

rohc@ietf.org

53rd IETF: Pre-Agenda

- **WG chair admonishments**
- **Real agenda**

- ✓ Blue sheets
- ✓ Scribe

Hello! This is an IETF Working Group

- **We are here to make the Internet work (Fred Baker)**
 - Together! (Harald Alvestrand)
- **Rough Consensus and Running Code (Dave Clark)**
- **Working Group is controlled by**
 - IETF Process (RFC2026, RFC2418) – *read it!*
 - Area Directors (ADs): Alison Mankin, Scott Bradner
 - Charter (<http://www.ietf.org/html.charters/rohc-charter.html>)
 - Working Group Chairs: Lars-Erik Jonsson, Carsten Bormann
 - Technical Advisor: Erik Nordmark
- **Work is done on email list: rohc@ietf.org**
 - And on IETF meetings, interim meetings, informal meetings
 - Mailing list is official channel, though

RFC 2026: Internet Standards Process

- **Standards track RFCs:**
 - **WG consensus (as judged by WG chairs)**
 - **WG last call**
 - **IESG approval (based on AD recommendation)**
 - Quality control!
 - **IETF last call**
- **Informational RFCs**
- **BCP (best current practice) RFCs**

RFC 2026: IPR issues (1)

- **(10.2) No contribution that is subject to any requirement of confidentiality or any restriction on its dissemination may be considered [...]**
- **Where the IESG knows of rights or claimed rights [...] the IETF Executive Director shall attempt to obtain from the claimant [...] a written assurance that upon approval by the IESG of the relevant Internet standards track specification(s), any party will be able to obtain the right to implement, use and distribute the technology [...] based upon the specific specification(s) under **openly specified, reasonable, non-discriminatory** terms.**

RFC 2026: IPR issues (2)

- **Contributions (10.3.1(6)):**
“The contributor represents that he has disclosed the existence of any proprietary or intellectual property rights in the contribution that are reasonably and personally known to the contributor.”
- **I.e., if you know of a patent application for a technology you are contributing, you have to tell.**
Or just shut up entirely!

53rd IETF: Agenda (09:00-11:30)

0900 - Chair admonishments and agenda	Bormann (10)
0910 - WG and document status update	Jonsson (15)
0925 - Signaling compression	
0925 - Architectural overview	Bormann (15)
0940 - UDVM, state	Price (25)
1005 - Extended operations	Hannu (15)
1020 - Security considerations	Bormann (20)
1040 - Open issues	(30)
1110 - Implementation status	Bormann (10)
1120 - Standardization status, what now?	Jonsson (10)

53rd IETF: Agenda (15:45-17:00)

1545 - Role of EPIC Lite in the ROHC work

- | | |
|---|---------------------|
| 1545 - Architectural role | Jonsson (10) |
| 1555 - Simple RTP profile example | West (10) |
| 1605 - EPIC-Lite notation, open issues | West (10) |
| 1615 - Way forward | Jonsson (5) |

1620 - TCP Profile

- | | |
|---|------------------|
| 1620 - Requirements and field behavior | West (15) |
| 1635 - TCP profile issues | Qian (25) |

53rd IETF: Agenda (17:00-18:00)

1700 - SCTP profile

1700 - Requirements document

Schmidt (5)

1705 - Profile design issues

West (10)

1715 - RTP issues

1715 - Implementation & impl. guide

Jonsson (10)

1720 - MIB

Quittek (15)

1735 - LLA implementation examples

Pelletier (5)

1740 - UDP Lite profiles, initial discussions

Pelletier (15)

WG Status, Goals and Milestones 1(3)

- ✓ I-D on Requirements for IP/UDP/RTP header compression.
- ✓ I-D of layer-2 design guidelines.
- ✓ I-D(s) proposing IP/UDP/RTP header compression schemes.
- ✓ I-D of Requirements for IP/TCP header compression.
- ✓ Requirements for IP/UDP/RTP header compression submitted to IESG for publication as Informational.
- ✓ Requirements for IP/TCP header compression submitted to IESG for publication as Informational.
- ✓ Resolve possibly multiple IP/UDP/RTP compression schemes into a single scheme.
- ✓ Submit I-D on IP/TCP header compression scheme.
- ✓ IP/UDP/RTP header compression scheme submitted to IESG for publication as Proposed Standard.
- ✓ Possible recharter of WG to develop additional compression schemes



WG Status, Goals and Milestones 2(3)

- **Jan 02 - Requirements and assumptions for signaling compression**
- † *Jan 02 - Initial draft on general signaling compression security analysis.*
- **Jan 02 - Signaling compression scheme submitted to IESG for publication as Proposed Standard, including security approach for SIP compression usage.**
- ✓ **Jan 02 - Layer-2 design guidelines submitted to IESG for publication as Informational.**
- **Apr 02 - LLA mapping examples submitted to IESG for publication as Informational.**
- **Apr 02 - I-Ds of ROHC IP/UDP/RTP bis, framework and profiles separated.**
- † *May 02 - General signaling compression security analysis submitted to IESG for publication as Informational.*
- **May 02 - ROHC MIB submitted to IESG for publication as Proposed Standard.**



WG Status, Goals and Milestones 3(3)

- **Aug 02 - ROHC UDP Lite schemes submitted to IESG for publication as Proposed Standard.**
- **Sep 02 - ROHC IP/UDP/RTP schemes submitted to IESG for publication as Draft Standard.**
- **Sep 02 - Requirements for IP/TCP compression submitted to IESG for publication as Informational.**
- **Sep 02 - ROHC framework submitted to IESG for publication as Draft Standard.**
- **Sep 02 - IP/TCP compression scheme submitted to IESG for publication as Proposed Standard.**
- **Dec 02 - Requirements for IP/SCTP compression submitted to IESG for publication as Informational.**
- **Dec 02 - IP/SCTP compression scheme submitted to IESG for publication as Proposed Standard.**
- **Dec 02 - Possible recharter of WG to develop additional compression schemes.**



Document status update

- **Published:**
 - **RFC3095: Framework and four profiles**
(was: draft-ietf-rohc-rtp-09.txt)
 - **RFC3096: RTP requirements**
(was: draft-ietf-rohc-rtp-requirements-05.txt)
- **Approved (in RFC editor queue):**
 - **draft-ietf-rohc-over-ppp-04.txt**
 - **draft-ietf-rohc-rtp-0-byte-requirements-02.txt**
 - **draft-ietf-rohc-rtp-lla-03.txt**
- **Submitted to IESG (passed IESG last-call):**
 - **draft-ietf-rohc-rtp-lower-layer-guidelines-03.txt**
 - **draft-ietf-rohc-rtp-lla-r-mode-02.txt**

New ROHC mail list

- rohc@cdt.luth.se has been closed
- New ROHC list, rohc@ietf.org
- All old subscribers were subscribed to the new list
- New archive at:
<http://www1.ietf.org/mail-archive/working-groups/rohc/>
- Old archive will stay at:
<http://www.cdt.luth.se/rohc>
- The new list is “subscriber-post”, with manual moderation of other postings
- Addresses can be subscribed with disabled delivery

Signaling Compression: Overview

Carsten Bormann

cabo@tzi.org

based on slides from:

Richard Price

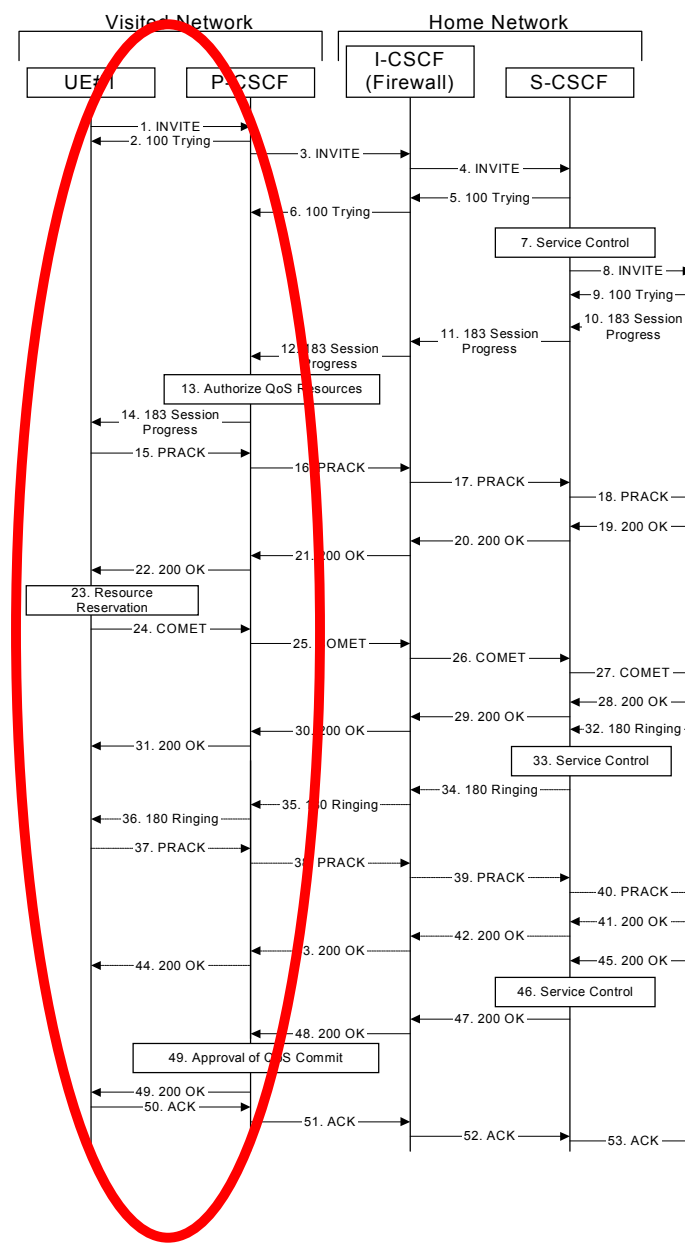
Richard.Price@roke.co.uk

Hans Hannu

Hans.Hannu@epl.ericsson.se

Why?

- ◆ Minimize **connection setup delay** in complex 3GPP SIP interactions
- ◆ Minimize **bandwidth stealing** for in-call usage of SIP
- ◆ The point is *not* saving raw bandwidth (although it does help the network!)
- ◆ [draft-ietf-rohc-signaling-req-assump-04.txt](#)



What are the messages to be compressed?

- ◆ **SIP:**
 - ▲ Largely a lock-step protocol
 - ▲ Essentially RFC822 (Text)
 - ▲ Can carry MIME payload
- ◆ **SDP:**
 - ▲ v=2 m=audio etc. (Text)
 - ▲ Other MIME payloads are possible (SDPng!)
- ◆ **Either *could* be encrypted, possibly partially**

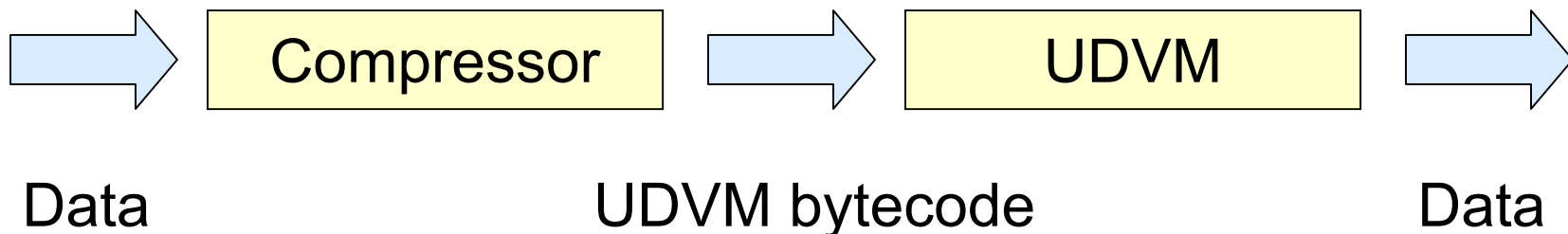
- ◆ **RTSP (for streaming), also carrying SDP**
- ◆ **DNS, RSVP, ... ???**

But which compression algorithm?

- ◆ **Hard to decide on a standard default algorithm**
- ◆ **Why not have the compressor tell the decompressor?**
 - ▲ **Quite usual approach in the compression industry**
Example: DEFLATE can download Huffman tables
- ◆ **Universal Decompressor**
 - ▲ **Virtual machine optimized for decompression**
 - ▲ **Gets executable decompressor spec from compressor**
 - ▲ **No compression schemes in standards**
 - ▲ **Full interoperability with any compressor**

Universal Decompressor Virtual Machine

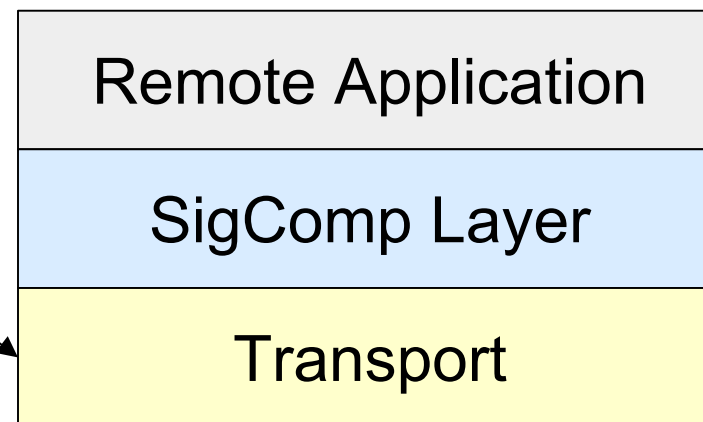
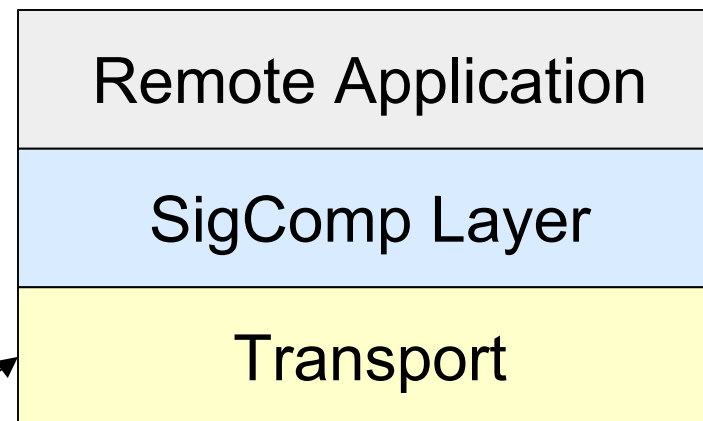
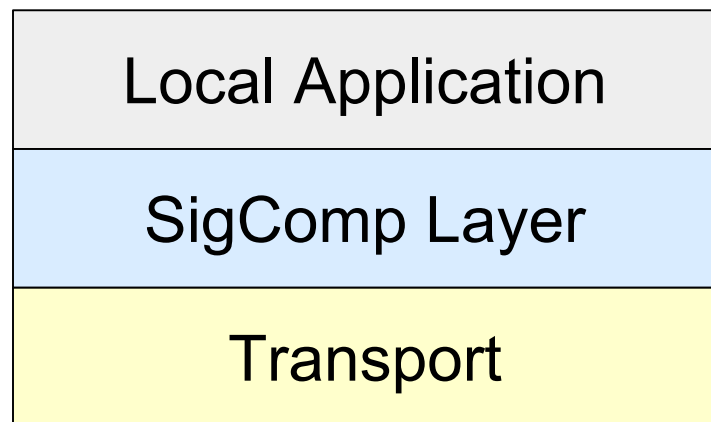
- ◆ **UDVM can be visualized in two ways**
 - ▲ Highly adaptive variant of DEFLATE
 - ▲ Like a Java Virtual Machine, but highly optimized for decompression
- ◆ **Compressed data is bytecode for the UDVM**
 - ▲ Algorithm is implementation decision at compressor



- ◆ **Instruction set selected to minimize code size**
 - ▲ Typical decompressor code requires 50 – 200 bytes

SigComp Architecture

- ◆ Offered as a *shim layer* between application and transport



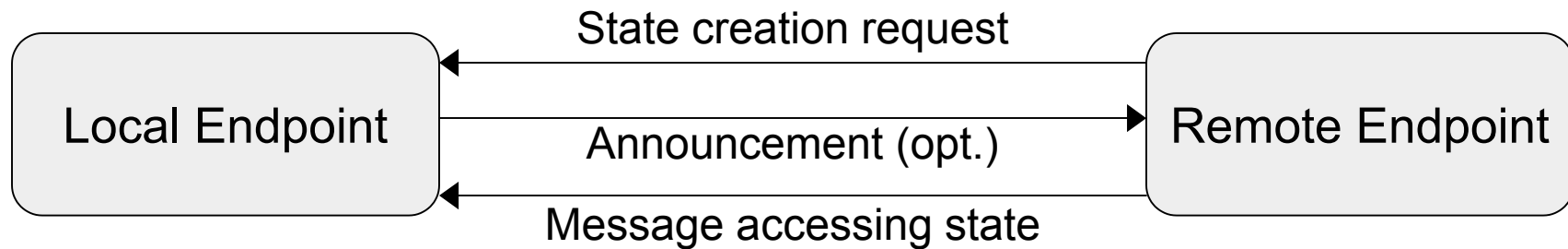
- ◆ Supports arbitrary transports (TCP, UDP, SCTP, TLS)

Minimal Protocol

- ◆ **UDP: per-packet, TCP: per-stream compression**
- ◆ **Can start out from scratch or with state reference**
 - ▲ Decompressor spec
 - ▲ Initial dictionary
- ◆ **Can use application knowledge to know that state is there**
 - ▲ Loading dictionary with INVITE is likely good enough
- ◆ **Extended versions can use ACKs and compressor-decompressor state sharing**
 - ▲ IETF has been notified about IPR claims in this area

State Handler

- ◆ **SigComp can save state after decompressing a message**
- ◆ **Two types of state information are available**
 - ▲ **Item of state created at the local endpoint**
 - ▲ **Announcement information for a remote endpoint**



- ◆ **State can be used over an unsecure transport**
 - ▲ **State is saved with permission from the application**
 - ▲ **State identifiers are hashes over the state information**

UDVM vs Algorithm Negotiation

- ◆ **Sigcomp: Negotiation not needed (just tell)**
- ◆ **Further optimize based on previous knowledge**
 - ▲ Preload an endpoint with well-known algorithms
 - ▲ Announce the corresponding state identifiers
- ◆ **Merits of algorithm announcement are questionable**
 - ▲ Downloading bytecode is efficient (66 bytes for LZW)
 - ▲ Additional RTT needed for announcements to arrive
 - ▲ Announcement costs more bandwidth than it saves
- ◆ **However defining *mandatory* state can be useful**
 - ▲ E.g., predefined static dictionary for SIP/SDP
 - ▲ Similar to the pre-defined Huffman codes of DEFLATE
 - ▲ Good for short-lived flows and for stateless endpoints
 - ▲ Only problem is choosing the state!

UDVM vs Algorithm Negotiation

- ◆ **Sigcomp: Negotiation not needed (just tell)**
- ◆ **Further optimize based on previous knowledge**
 - ▲ Preload an endpoint with well-known algorithms
 - ▲ Announce the corresponding state identifiers
- ◆ **Merits of algorithm announcement are questionable**
 - ▲ Downloading bytecode is efficient (66 bytes for LZW)
 - ▲ Additional RTT needed for announcements to arrive
 - ▲ Announcement costs more bandwidth than it saves
- ◆ **However defining *mandatory* state can be useful**
 - ▲ E.g., predefined state
 - ▲ Sim
 - ▲ Good
 - ▲ Only

Define this in the application!

SigComp Draft Structure and Concept

SigComp IETF draft structure comprises two different documents:

- ◆ **SigComp itself is the baseline solution, normative**
 - ▲ **Signaling Compression (SigComp), draft-ietf-rohc-sigcomp-05.txt**
 - ▲ **Explains the service of the underlying transport plus *per-message compression*.**
- ◆ **SigComp Extended Operations shows how to provide features significantly improving the compression efficiency**
 - ▲ **SigComp-Extended Operations, draft-ietf-rohc-sigcomp-extended-02.txt**
 - ▲ **Explains *acknowledgements, shared and dynamic compression*; significantly increasing the compression ratio.**
 - ▲ **Informative, as implementation is in compressor and UDVM code (but important as a proof-of-concept!)**
- ◆ **Later: UDVM user guide**
 - ▲ **E.g., bytecode for common algorithms**
 - ▲ **Explains implicit acknowledgements (may have fewer IPR issues)**

SigComp Draft Structure and Concept

SigComp IETF draft structure comprises two different documents:

- ◆ **SigComp itself is the baseline solution, normative**
 - ▲ **Signaling Compression (SigComp), draft-ietf-rohc-sigcomp-05.txt**
 - ▲ **Explains the service of the underlying transport plus *per-message compression*.**
- ◆ **SigComp Extended Operations shows how to provide features significantly improving the compression efficiency**
 - ▲ **SigComp-Extended Operations, draft-ietf-rohc-sigcomp-extended-01.txt**
 - ▲ **Explains *acknowledgement* and *per-message compression*; significantly improved efficiency.**
 - ▲ **Information on compressor and UDVM code (but in concept!)**
- ◆ **Later: UDVM**
 - ▲ **E.g., byte stream for common algorithms**
 - ▲ **Explains implicit acknowledgements (may have fewer IPR issues)**

The code is in the compressor/UDVM!

UDVM user guide: plan

- ◆ **Define assembly language**
 - ▲ Not needed for interoperability, but good for talking about UDVM
 - ▲ Point to resources (example code for an assembler)
- ◆ **Provide example decompressors**
 - ▲ LZ77 simple, DEFLATE, LZW; more coming
 - ▲ Assembly code and example input
 - ▲ More “bag-o-tricks” stuff
- ◆ **Explain how to put sigcomp into new environments**
 - ▲ I.e., what needs to be defined for sigcomp to be usable
 - ▲ UDVM usage modes (e.g., flexible vs. pre-loaded)
- ◆ **Hints for UDVM implementers**
 - ▲ How to get a fast/cheap UDVM implementation
 - ▲ Point to resources (example code for a UDVM)

Sigcomp Architecture Issues

- ◆ **Discovery vs. Sigcomp**
 - ▲ How do you find out whether to use Sigcomp in the first place
- ◆ **Multiplexing (compressed vs. uncompressed)**
 - ▲ How do you distinguish sigcomp from original app packets
- ◆ **Integration into app protocol**
 - ▲ What is needed to marry app protocol and sigcomp
- ◆ **Application interface**
 - ▲ What is needed in a system between app and sigcomp impl
- ◆ **Editorial/Timeline**
 - ▲ When are we done?
- ◆ **... Richard: UDVM, state; Hans: -extended**

Discovery vs. Sigcomp

How to find out whether to use Sigcomp in the first place?

- ◆ **General Internet Discovery Issue: Hard**
 - ▲ Need to interface to existing application discovery architecture
 - ▲ Can't just overload URI schemes (sips:) and ports ad infinitum
 - ▲ Combinatorial explosion: Security, compression, what next...
 - ▲ New architecture needed! (Says IESG.)
 - ◆ **For 3GPP UE↔P-CSCF usage, non-issue**
 - ▲ Network can define Sigcomp as mandatory
- ➔ **Don't solve it in Sigcomp now!**
(but keep issue active)

Multiplexing (compressed vs. uncompressed)

How to distinguish sigcomp from original app packets?

- ◆ **Can use different ports (as section 3.1 says)**
 - ▲ Even if the compressed port is not a well-known port
 - ➔ finding the port is a discovery issue
- ◆ **For text-based signaling, can use first byte**
 - ▲ 1111xxx for sigcomp, 0xxxxxxx for ASCII
 - ▲ Distinguishable even with UTF-8-based protocols
 - ▲ [Help for RTP/UDP ROHC: Distinguishable from RTP (10xxxxxx)]
- ➔ **Sigcomp should**
 - ▲ be open to a number of ways to do the multiplexing
 - ▲ not prescribe any of these (linked to discovery, anyway)

Integration into app protocol

What is needed to marry app protocol and sigcomp

- ◆ **Discovery/multiplexing... ➔ not now**
- ◆ **App-specific static dictionary is extremely useful**
 - ▲ **But don't change that every week**
 - ▲ **Will define this for SIP now**
(could go into appendix for –06 or in separate document)
- ◆ **Default values for sigcomp application parameters**
 - ▲ **Will reduce number of these parameters (–06)**
 - ▲ **“Good” values might become quite obvious then**

Application interface

What is needed in a system between app and sigcomp?

- ◆ **App needs to decide whether state is authorized**
 - ▲ **Yes/no**
- ◆ **If yes, app needs to provide endpoint ID**
 - ▲ **Maybe wrong term (context ID?)**
 - ▲ **Used as the unit of granularity in apportioning state memory**
 - ▲ **Used as a compartment ID for state FIFOing**
 - ▲ **Used as a compartment ID with STATE-FREE (➡ -06)**

Editorial/Timeline

When are we done?

- ◆ **Really: what are the document dependencies?**
- ◆ **-extended planned to be available in time**
- ◆ **Refer to UDVM user guide in general terms**
 - ▲ **Planned for later...**

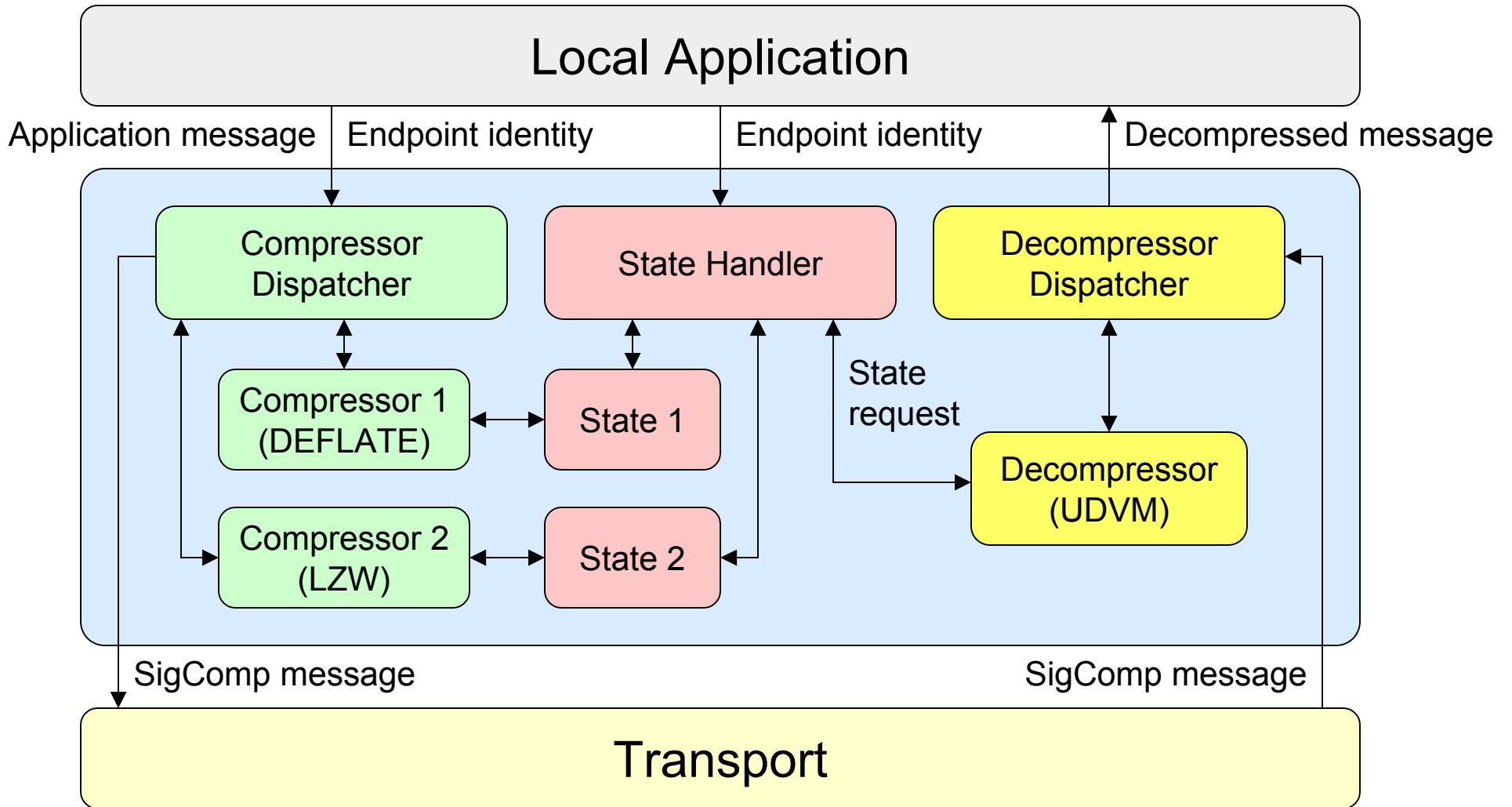
- ◆ **Timeline: Apr 05 for –06, -extended–03**
 - ▲ **Decide then whether to last-call or do –07**

SigComp

Compression for Signaling Applications

Richard Price
(richard.price@roke.co.uk)

Architecture for a SigComp Endpoint



Supported Algorithms (UDVM)

- Arbitrary algorithms can be run on the UDVM
 - Given enough processing and memory
- Simple algorithms can be programmed directly

LZ77

LZSS

LZW

DEFLATE

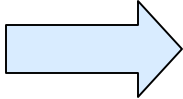
LZJH

- More complex algorithms require a compiler such as EPIC
 - Fast creation of “application-aware” bytecode



Changes in SigComp-05 (UDVM)

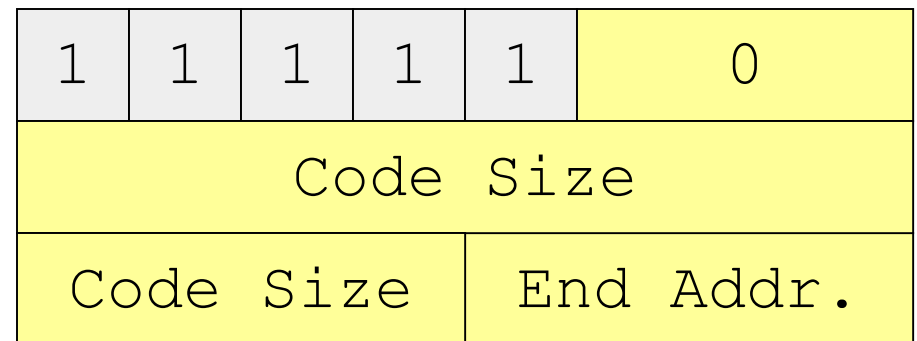
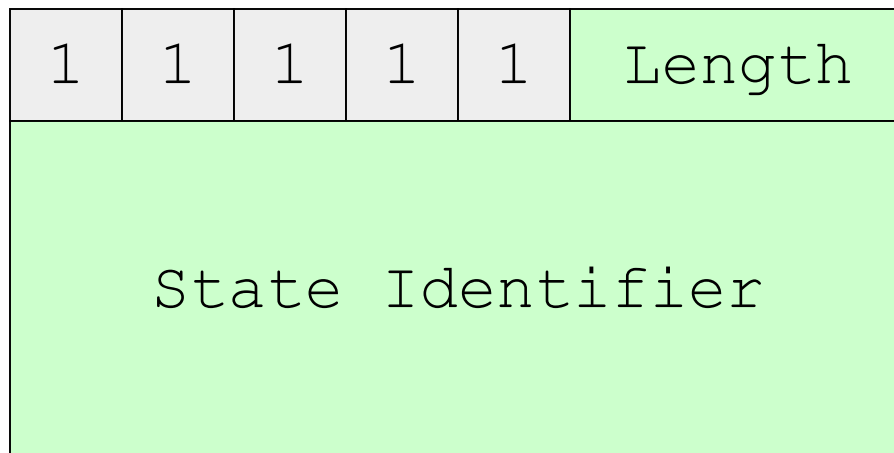
- Added variable-length encoding for operands

LZ77:	96 bytes		47 bytes
LZW:	132 bytes		66 bytes

- Compressed and uncompressed data is now buffered externally to the UDVM
- Added UDVM instructions to create and access state
- Added some further general-purpose instructions
 - Bit manipulation (AND, OR, NOT)
 - Initialization (LOAD, MULTILOAD)

Proposed Changes (UDVM)

- Updated SigComp header



- Header should also be decompressed by the UDVM
 - Currently done by the dispatcher
 - Not compatible with SigComp architecture
 - UDVM bytecode cannot access header information

Proposed Changes (UDVM)

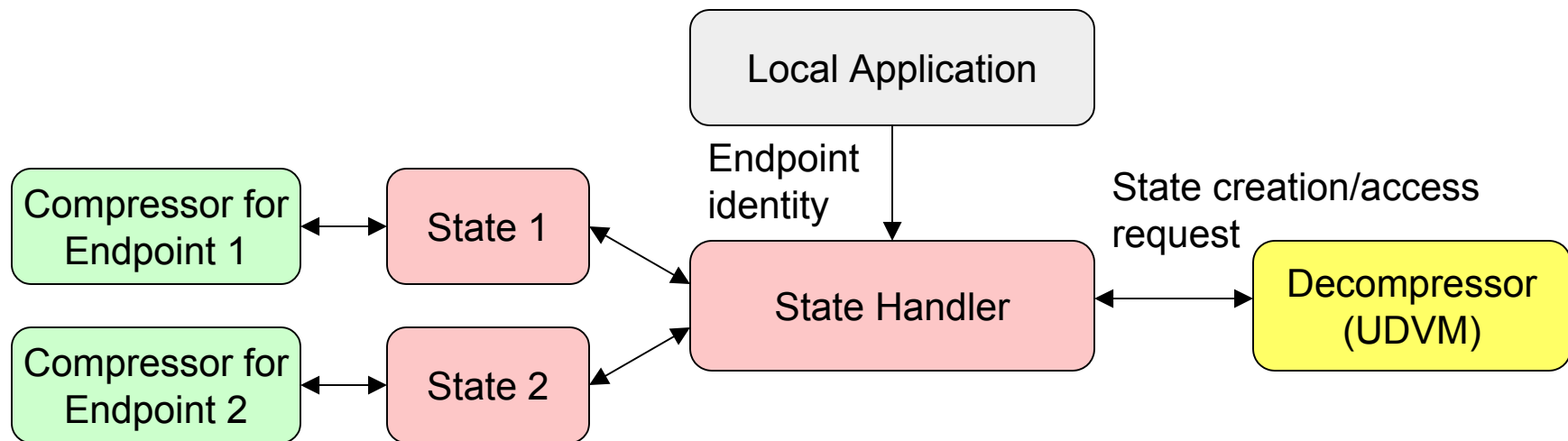
- Fix values for several application-defined parameters
 - No need to change on a per-application basis
 - `maximum_expansion_size`
 - `maximum_compressed_size`
 - `maximum_uncompressed_size`
 - `minimum_hash_size`
 - `cycles_per_message`
- Change some fields in the announcement data
 - `maximum_state_size` should be announced
 - `cycles_per_message` should not be announced
- `byte_copy_right` should point to first byte after buffer

Open Issues (UDVM)

- Are there any missing instructions (SHIFT, MODULO)?
- Should INPUT-BYTECODE be able to retrieve the entire compressed message?
- Should there be separate LSB and MSB instructions to cope with different bit orders within a byte?
- STATE-REFERENCE vs. STATE-EXECUTE – can these (should these) be unified?
- Are more CRCs required?
- How many values should there be for UDVM parameters (e.g. fix six or seven values for UDVM_memory_size)?

Changes in SigComp-05 (State Handler)

- Added secure state reference mechanism



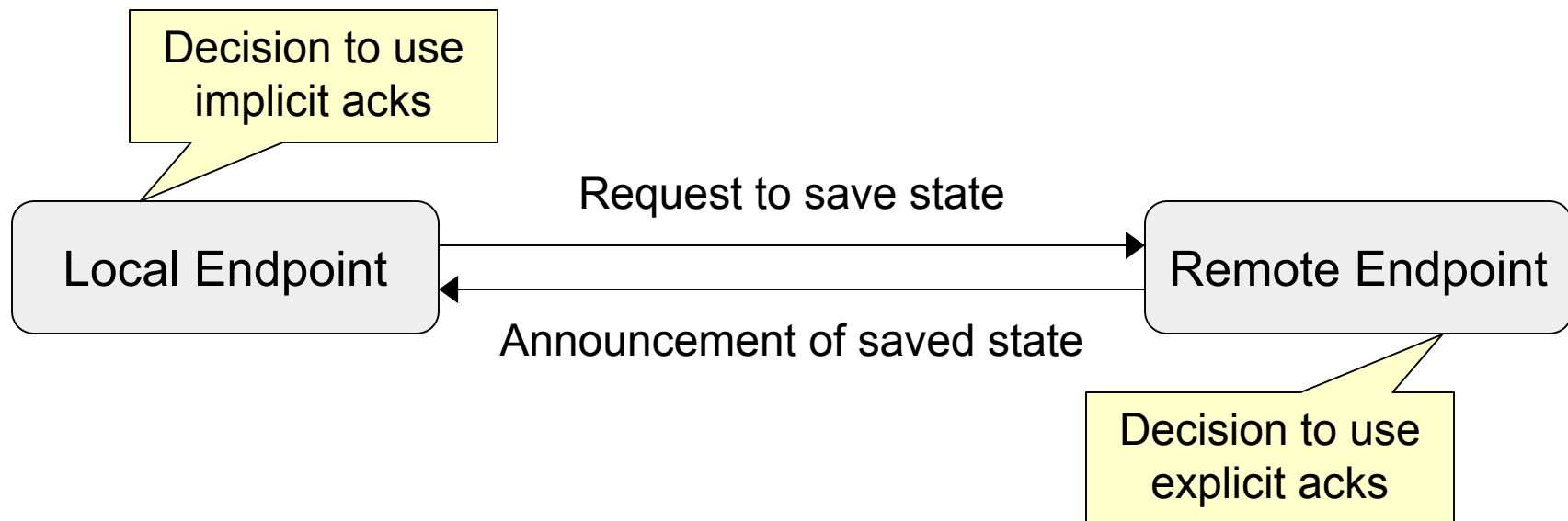
- State can be used over an unsecure transport
 - State is saved with permission from the application
 - State identifiers are hashes over the state information

Proposed Changes (State Handler)

- Flexible state identifiers
 - All state identifiers should be saved as 16-byte hashes
 - Minimum reference length chosen when creating state
- State identifier hash should be calculated over entire state item (not just over `state_value`)

Open Issues (State Handler)

- State can be acknowledged implicitly or explicitly
- Schemes are currently invoked at different endpoints



- Should the local endpoint be able to request explicit acks?

Open Issues (State Handler)

- Should the compressor be able to choose the hash function when creating state?
- Is a STATE-CREATE instruction needed?
- Should there be a “state-class” operand in END-MESSAGE for state management/(rollback)?
- STATE-FREE – is it needed, can it be made secure?

SigComp – Extended operations

Hans Hannu

Ericsson

Hans.Hannu@epl.ericsson.se

SigComp – Extended Operations

[<draft-ietf-rohc-sigcomp-extended-02.txt>](#)

- Progress since IETF-52 Salt Lake City
 - Before draft-submission (-> March 1st 2002)
 - After draft-submission (-> March 19th 2002)
- Overview
 - Explicit acknowledgements
 - Shared compression
 - State management/(Rollback)

The authors believe that there might be IPR issues related to the extended operation mechanisms. For more information refer to:

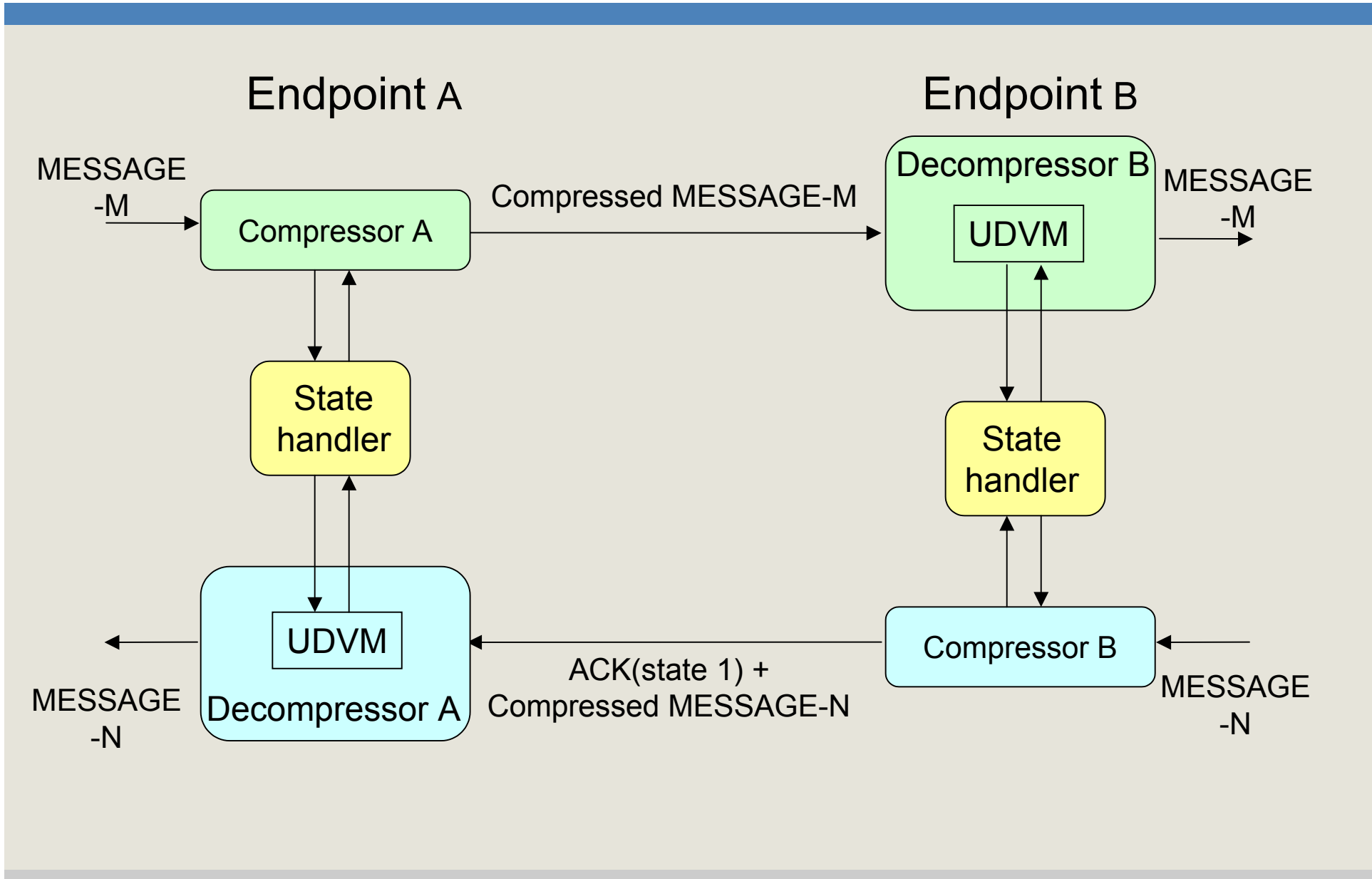
<http://www.ietf.org/ipr.html>

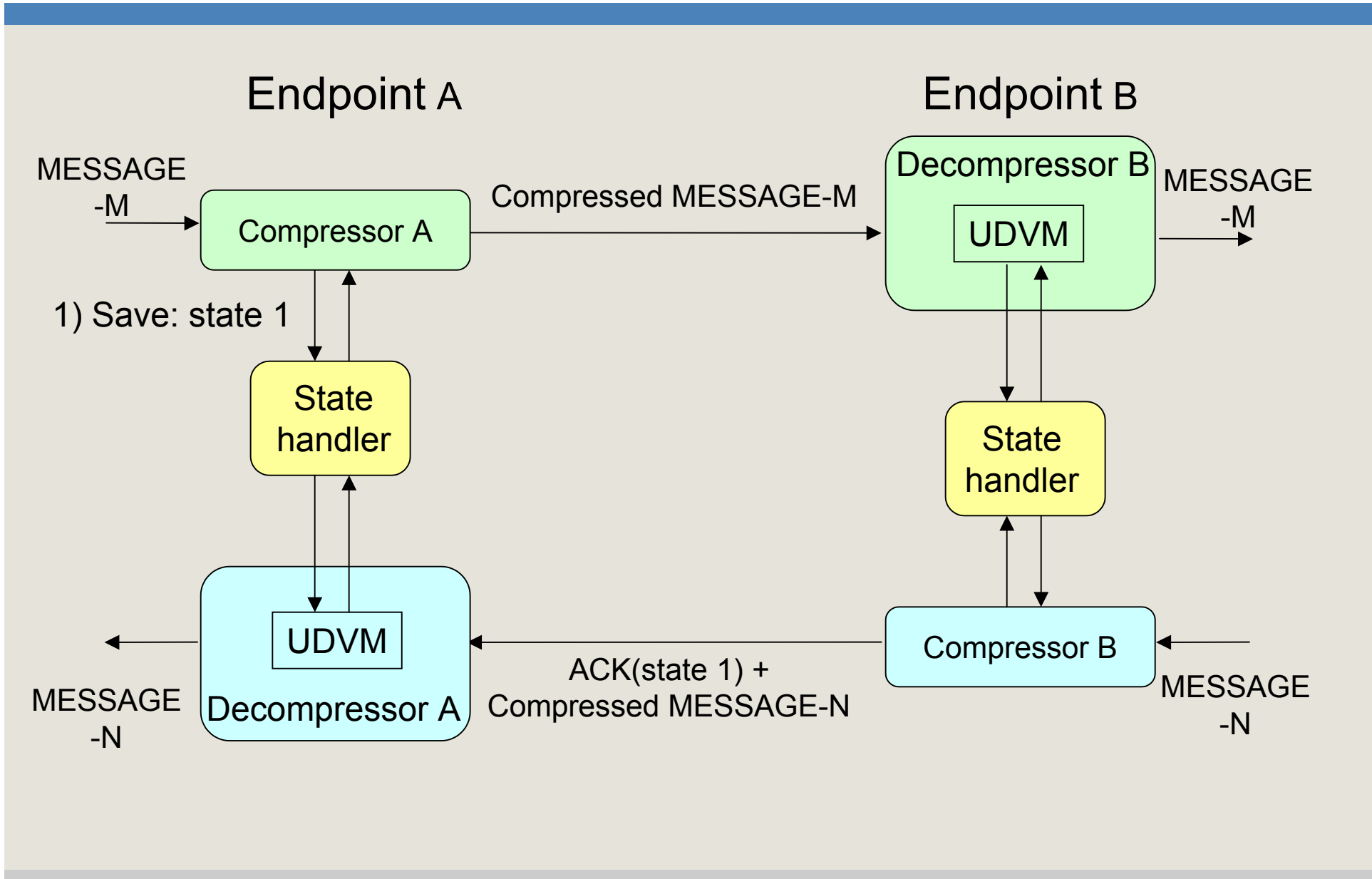
Progress since IETF-52 Salt Lake City

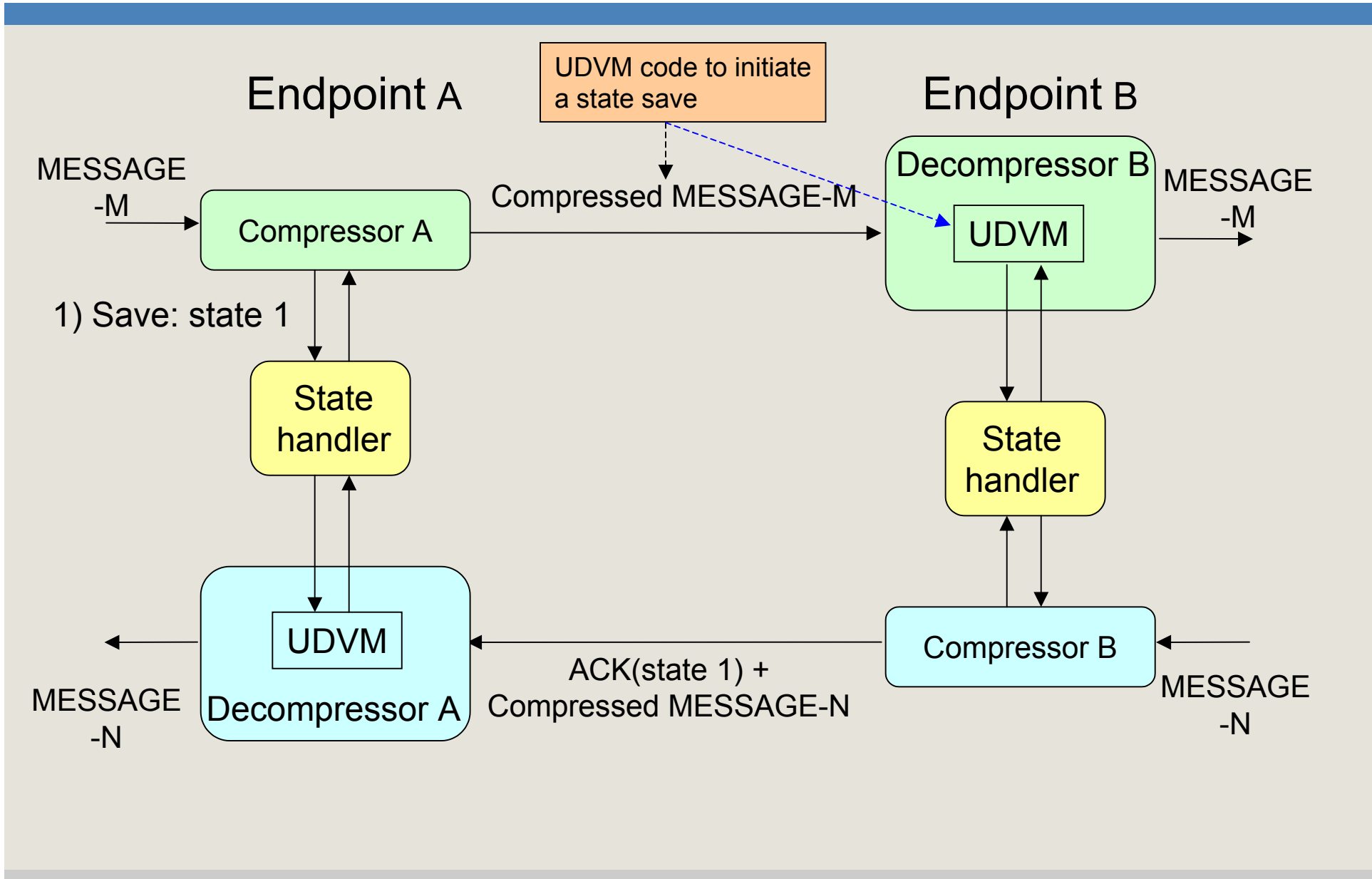
- Before draft-submission (-> March 1st 2002)
 - The defined SigComp message format is now used for both SigComp-basic and SigComp-extended
 - Architectural view for explicit acknowledgements is introduced
 - “New” mechanisms/concepts added e.g.:
 - User-specific dictionary
 - State management/(Rollback)
- After draft-submission (-> March 19th 2002)
 - No change to the announcement information is needed to provide for *explicit acknowledgements and shared compression*.
 - Code for providing the above mechanisms are in the compressor/UDVM,
 - i.e. UDVM instructions can be combined to provide support for *explicit acknowledgements and shared compression*.

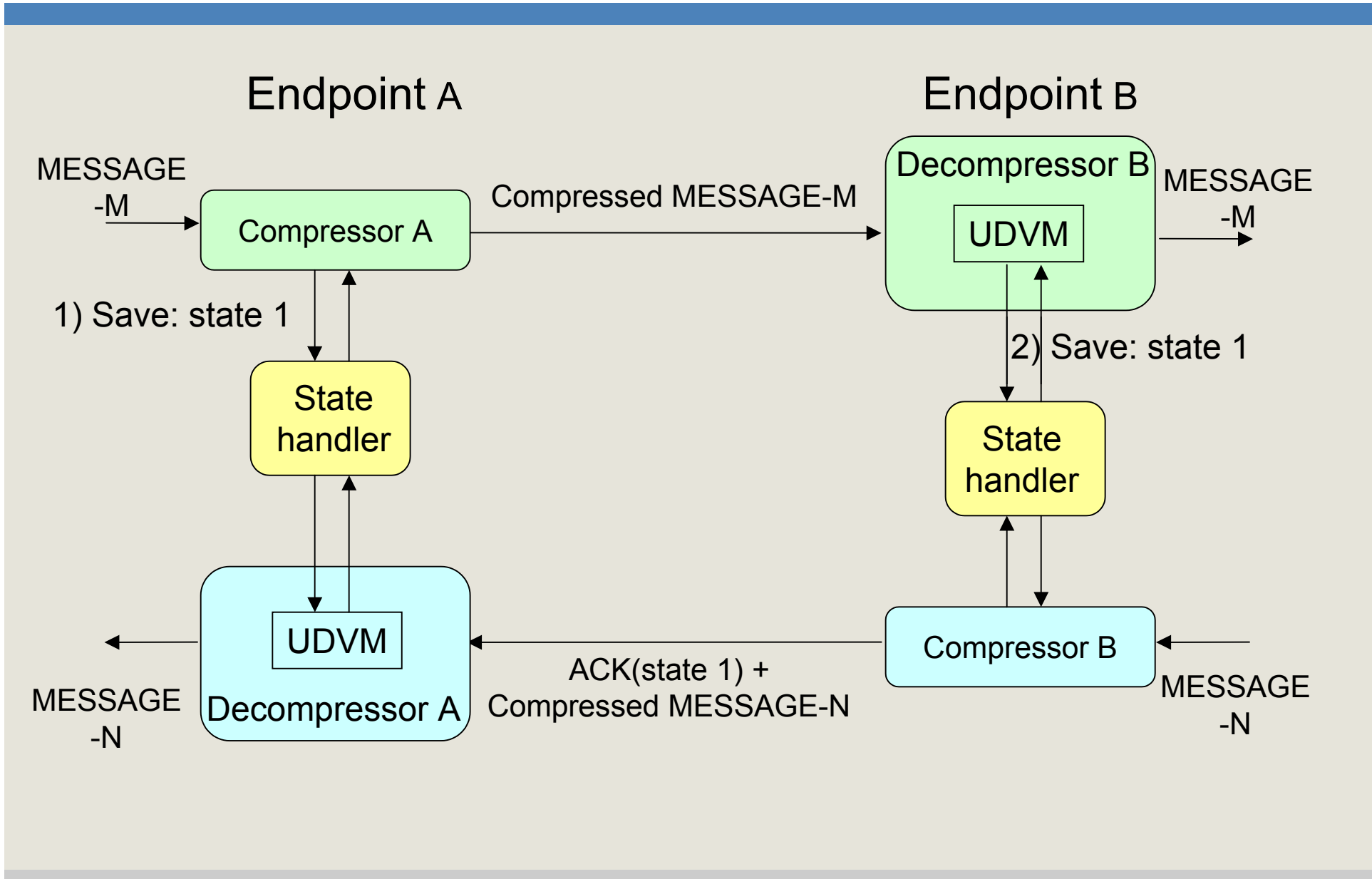
SigComp Extended – Explicit acknowledgements

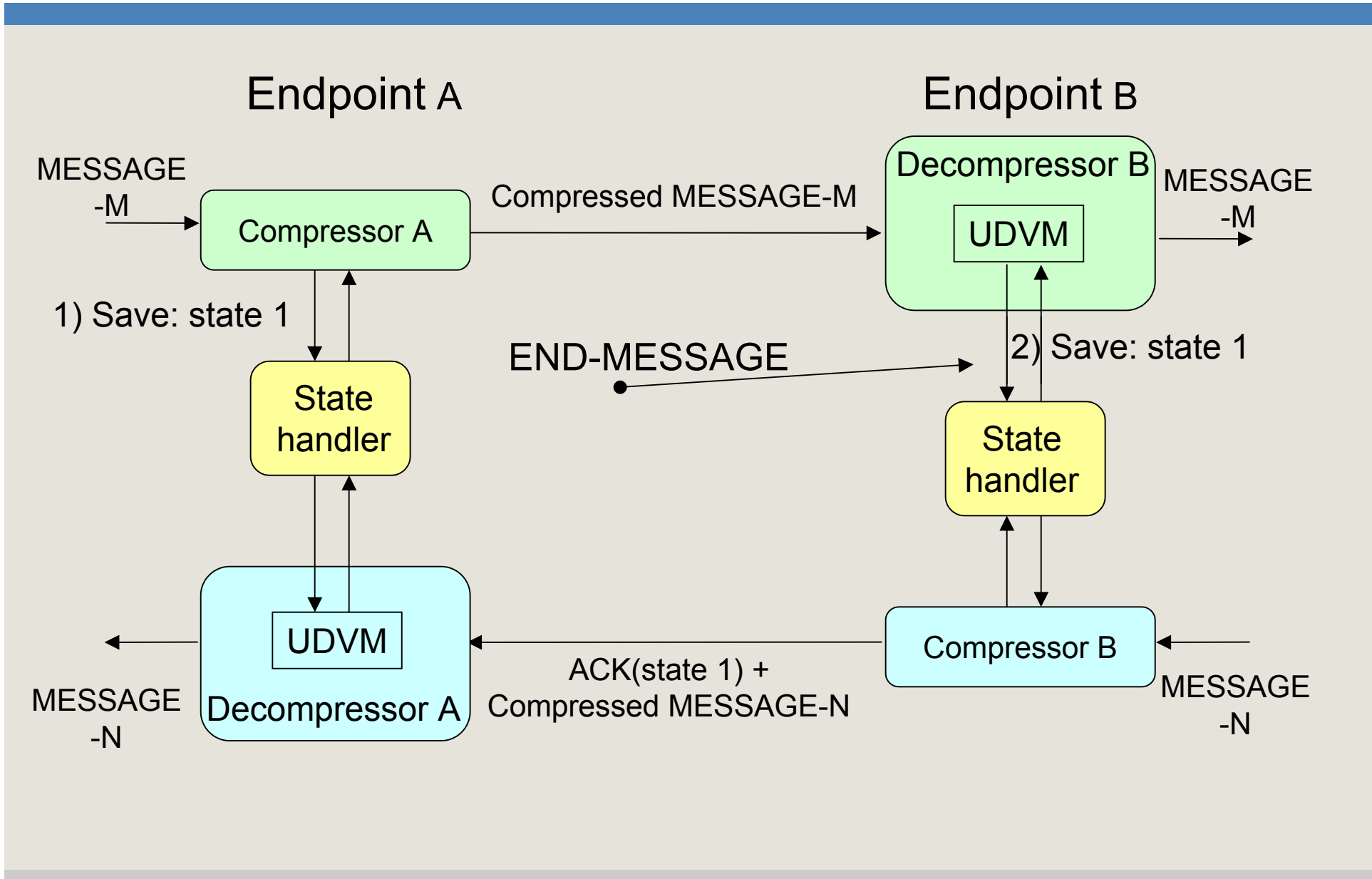
- Explicit acknowledgements
 - Endpoint B may decide to acknowledge established states, created by messages received from endpoint A.
 - The acknowledgements are passed to endpoint A through endpoint A's UDVM using the END-MESSAGE instruction with the announcement location pointer referencing the explicit acknowledgement feedback.

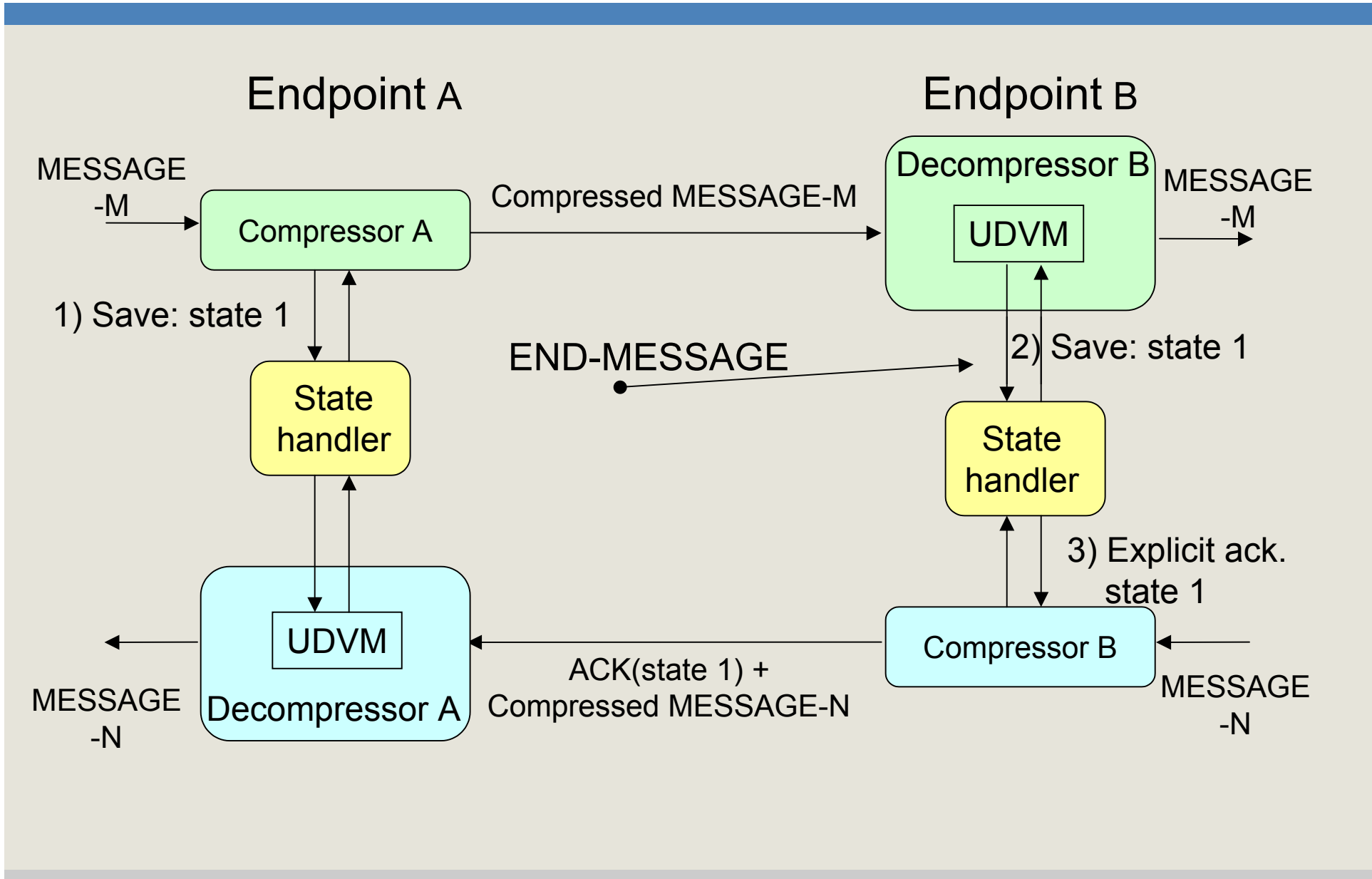


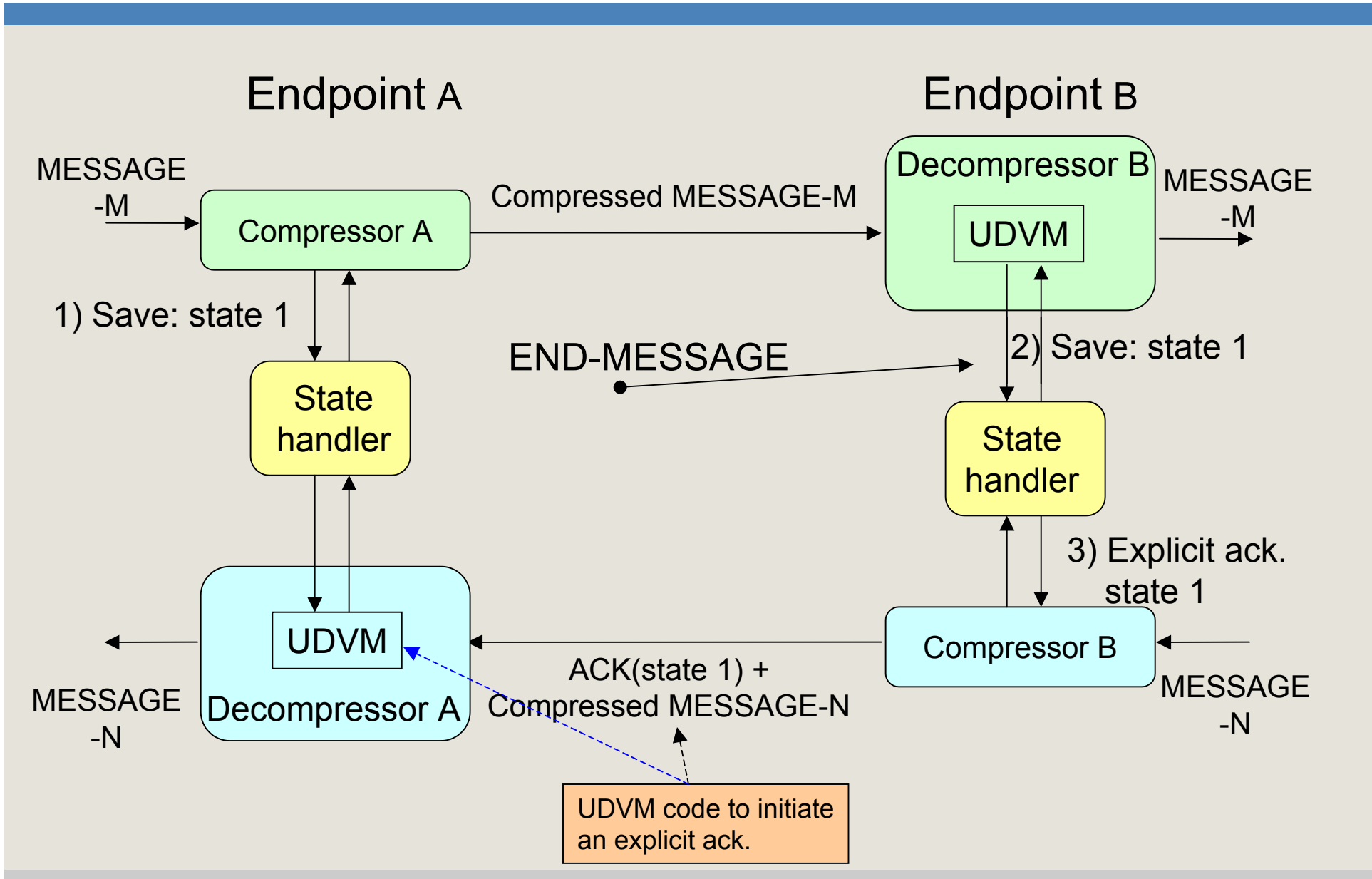


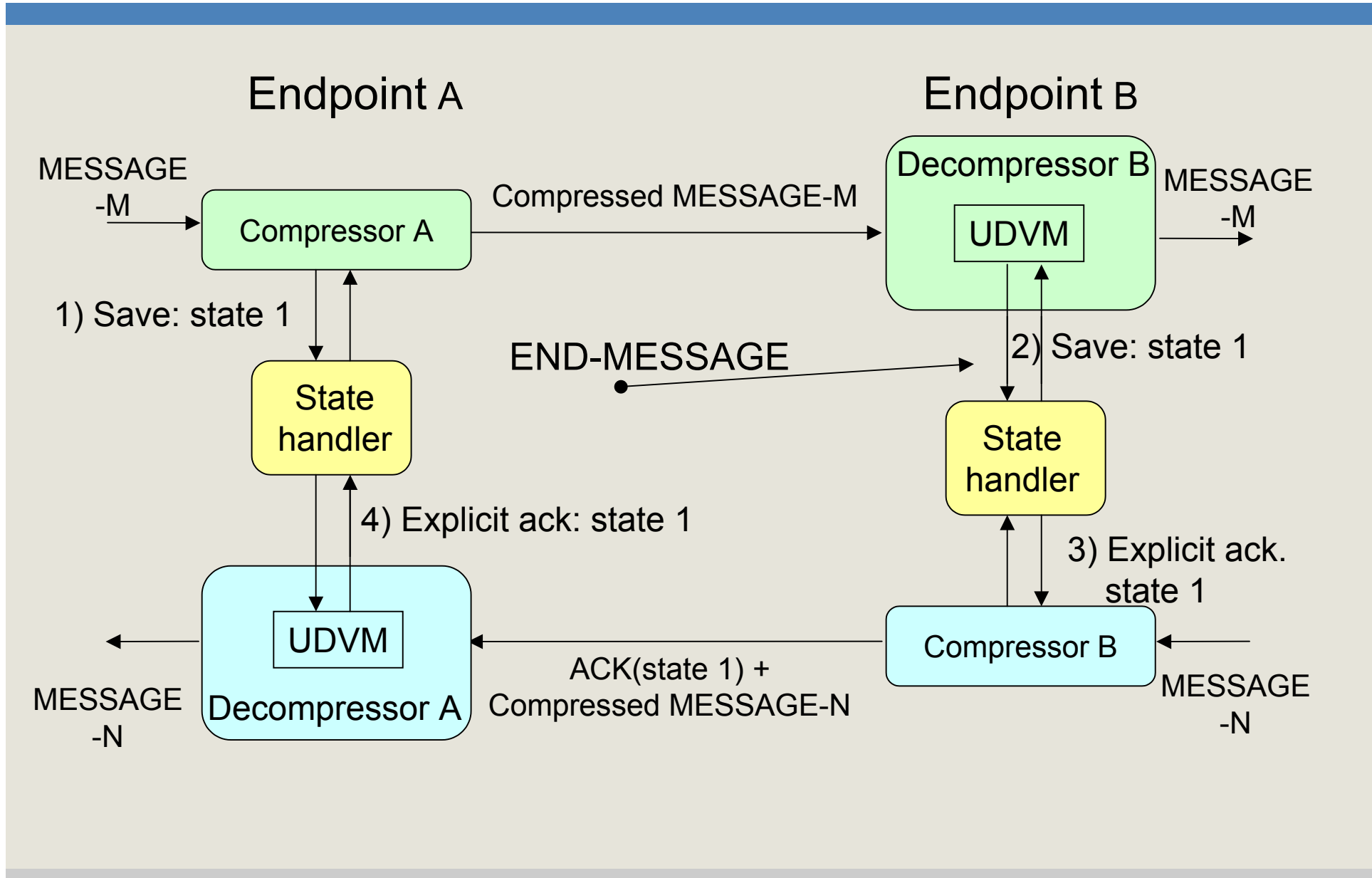


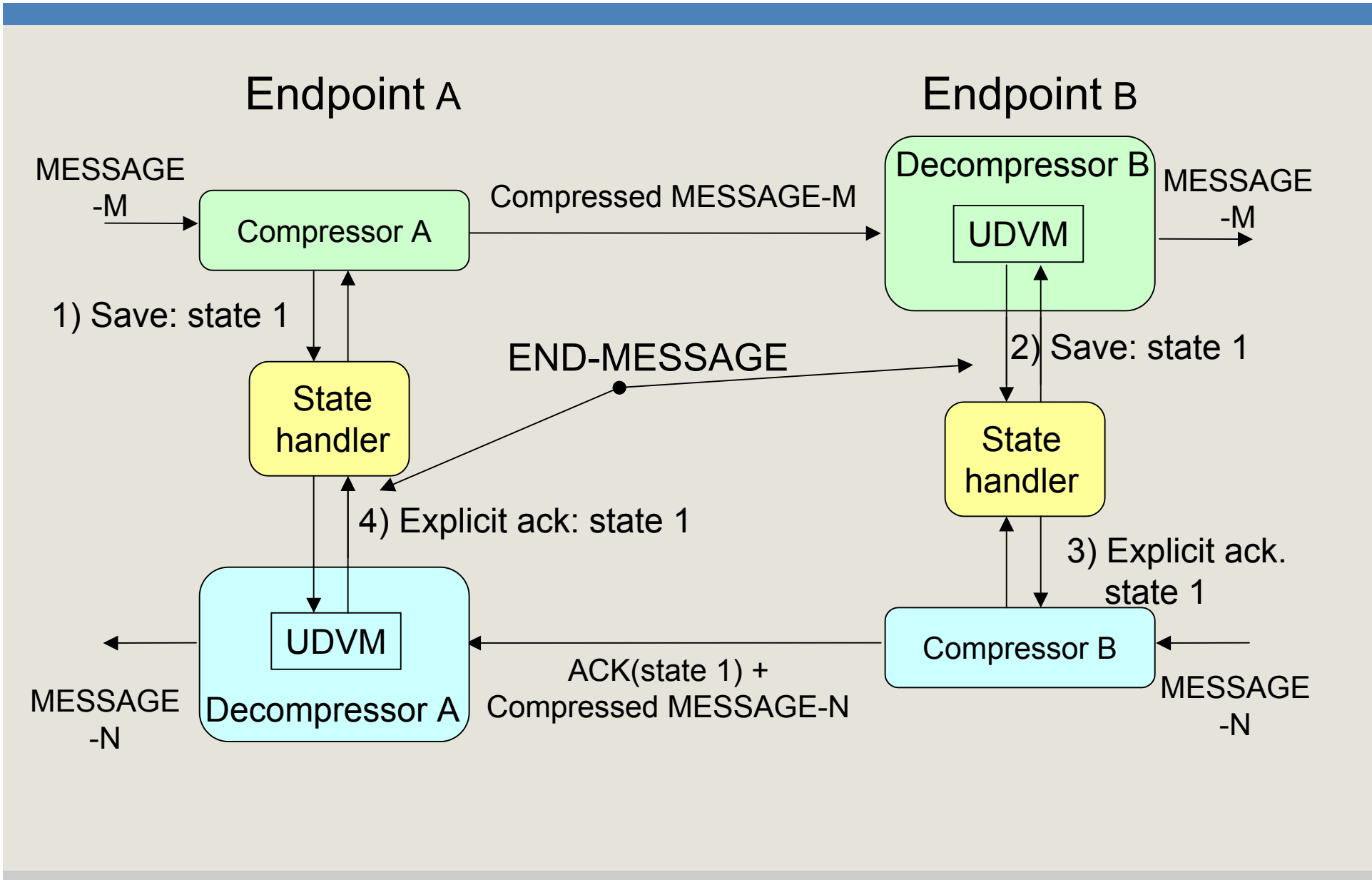






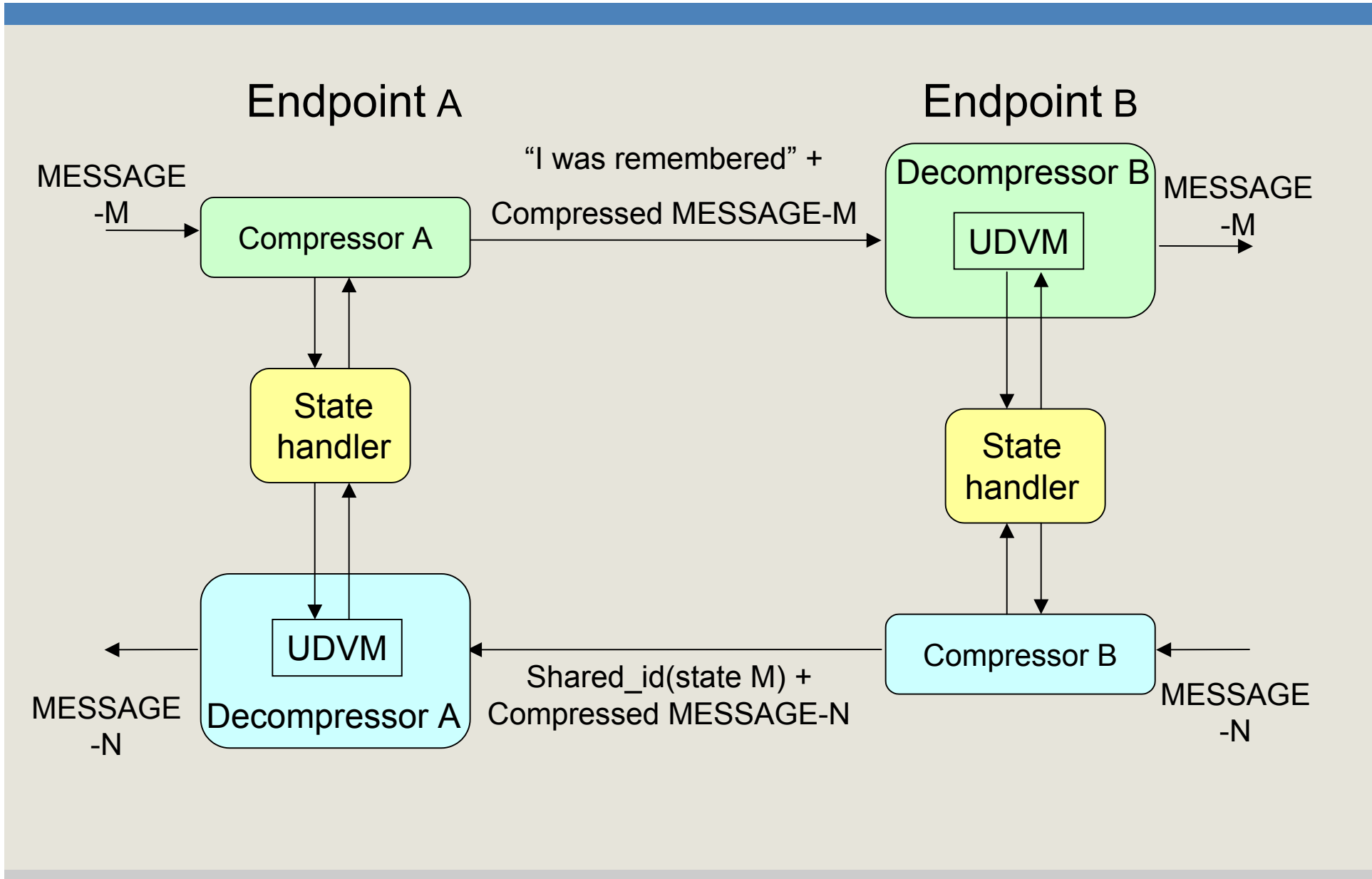


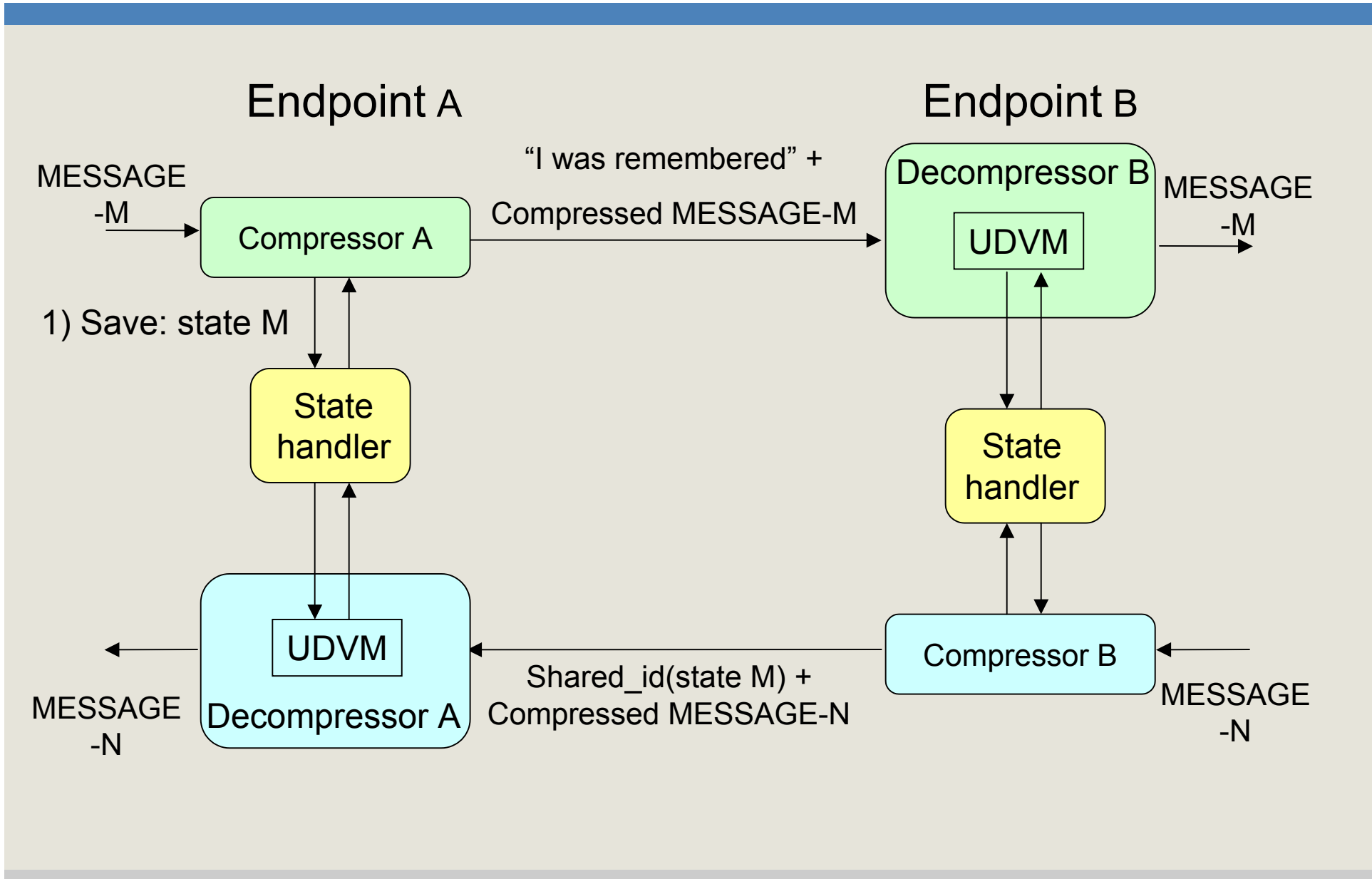


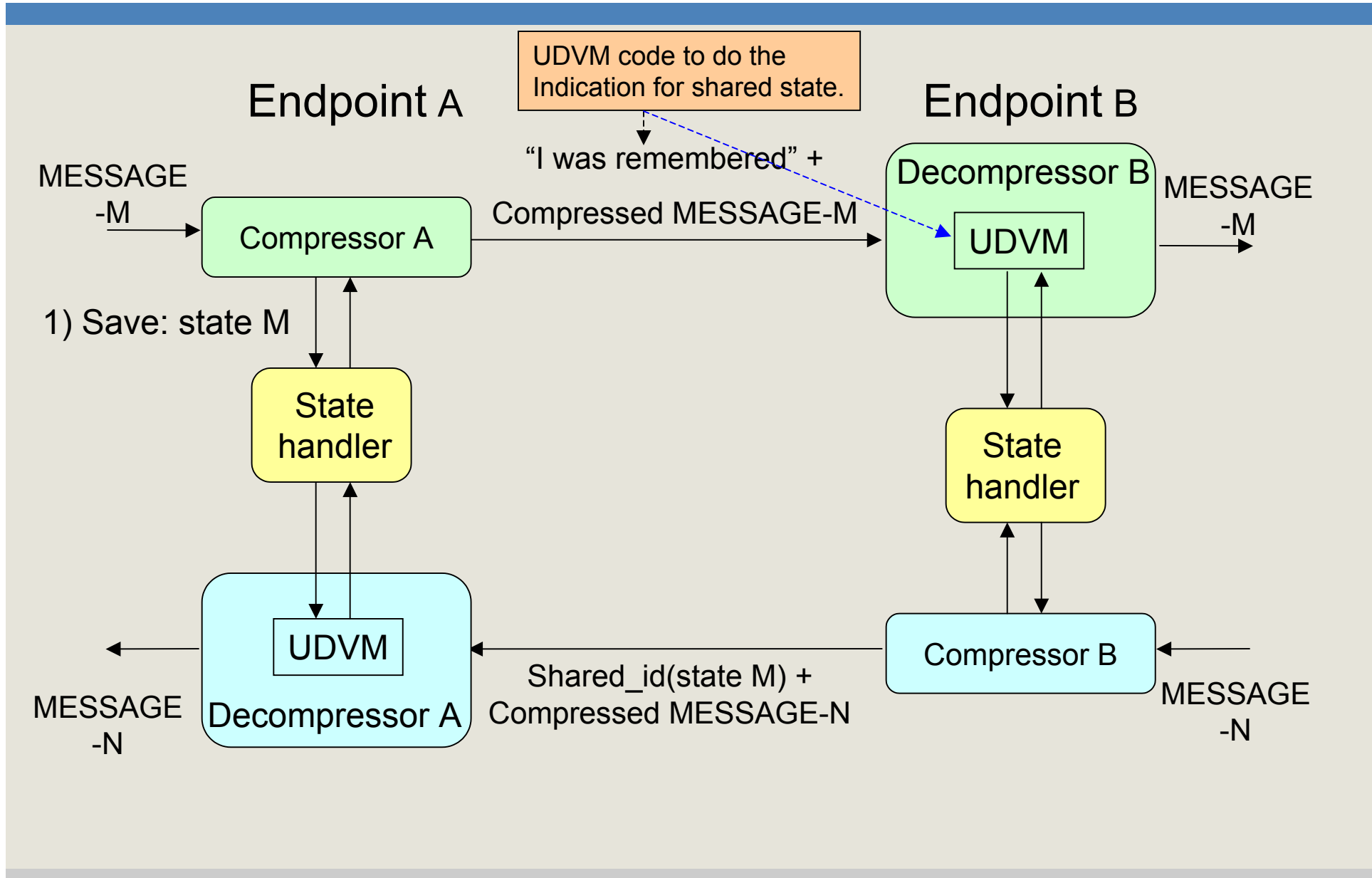


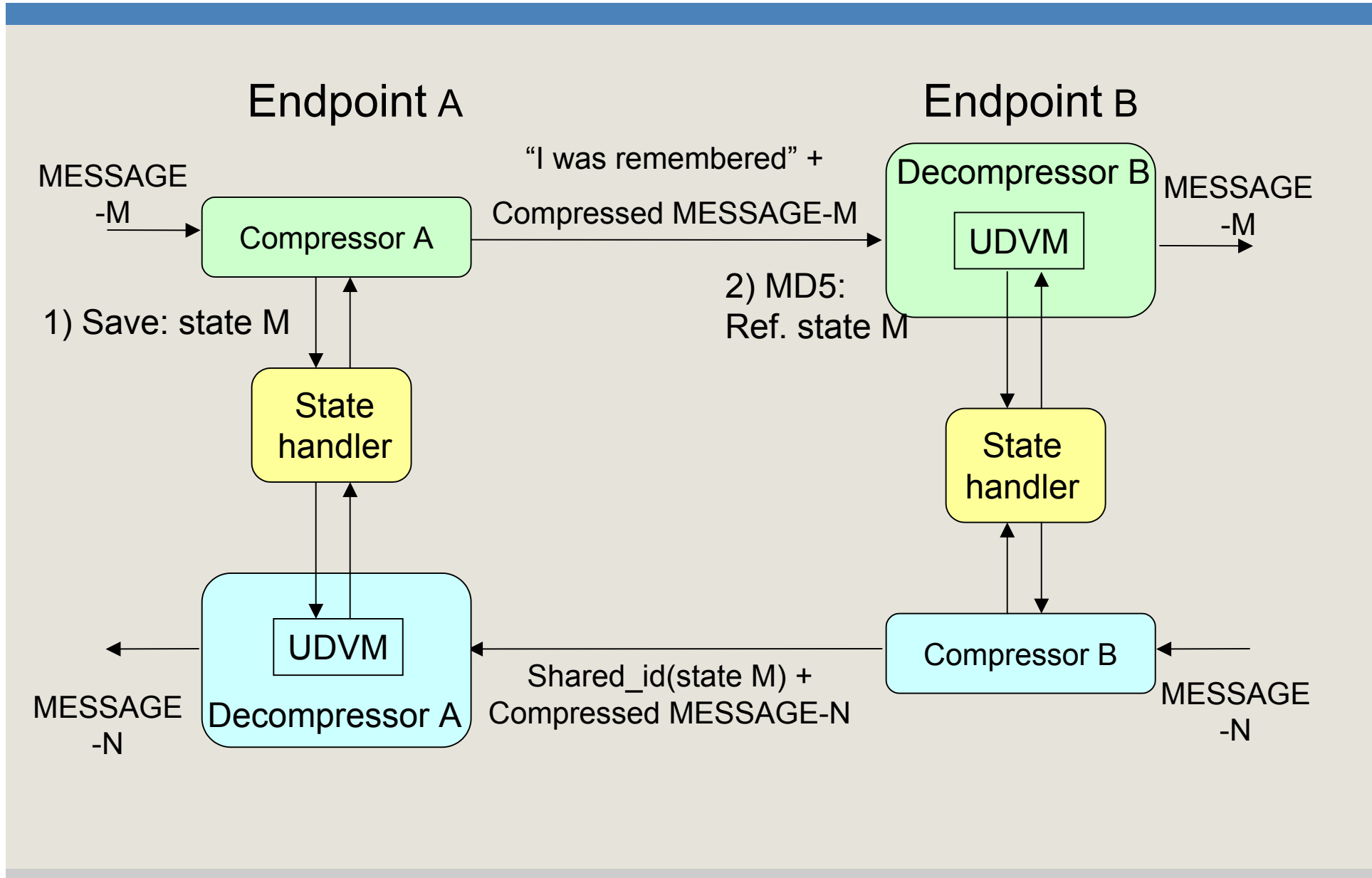
SigComp Extended – Shared compression

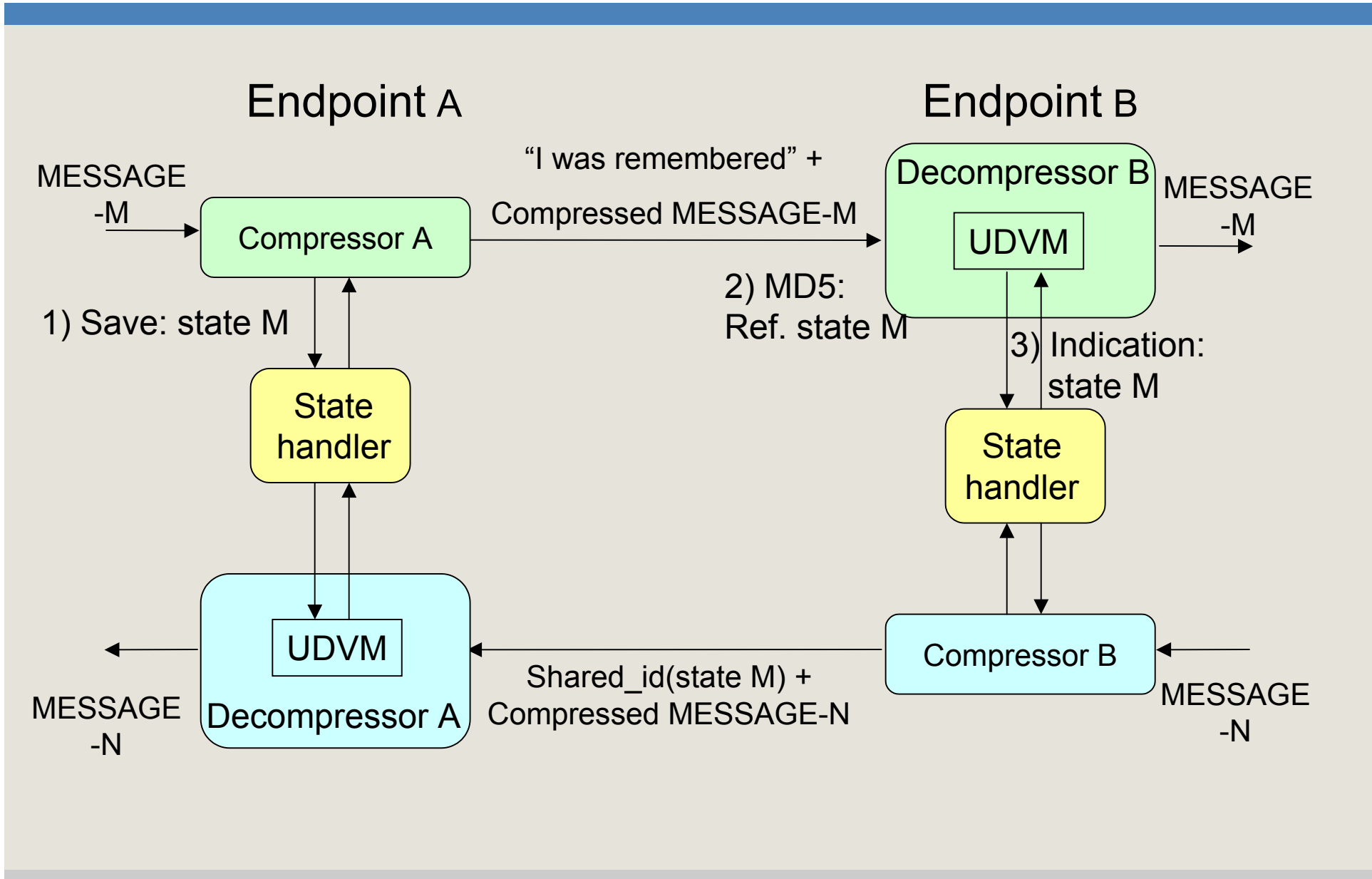
- Shared compression
 - Endpoint A indicates to endpoint B that a state corresponding to the uncompressed version of the message is saved and can be accessed by the local UDVM.
 - The indication to endpoint B is done by placing the state reference at the location of the announcement information

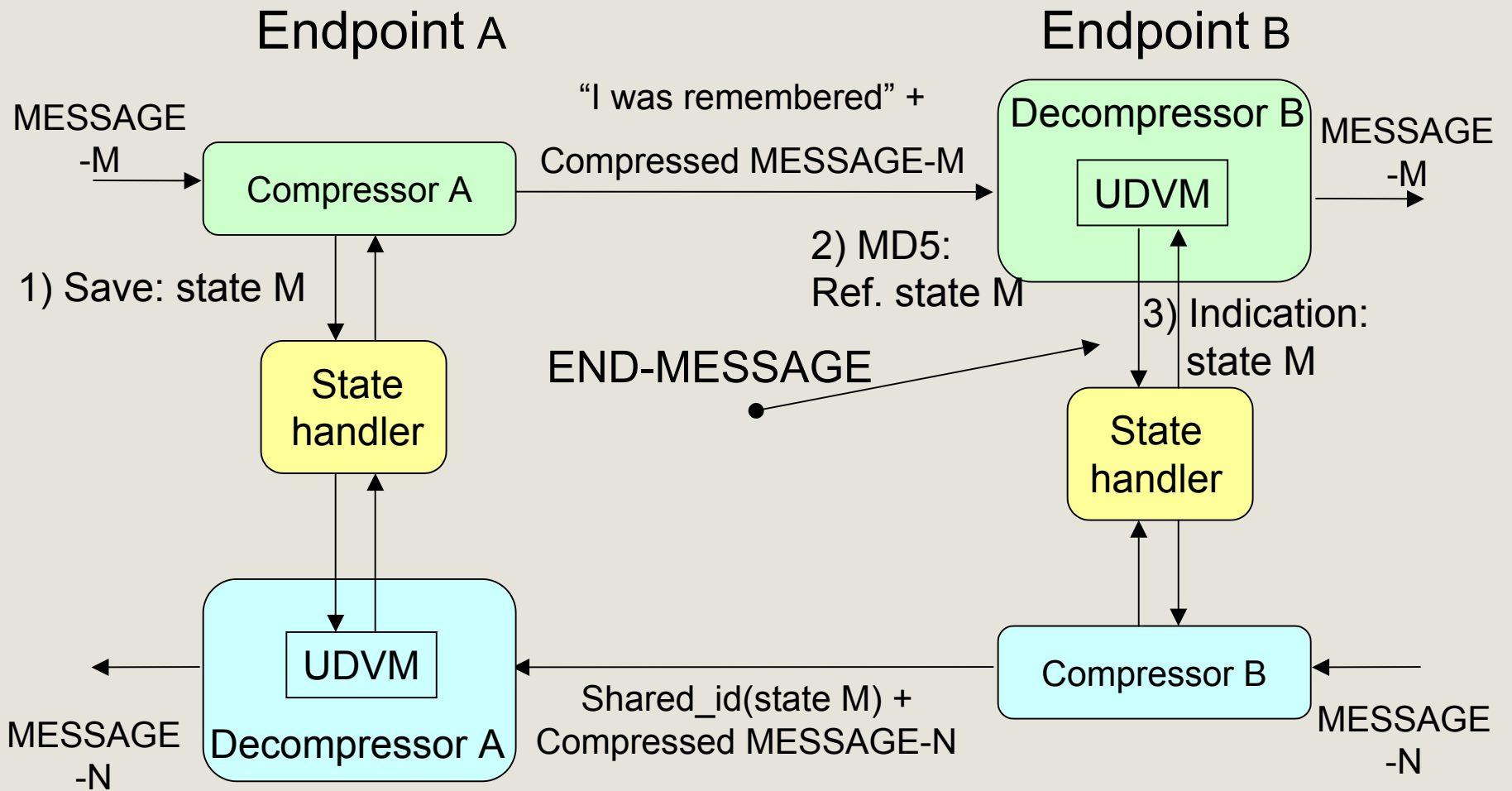


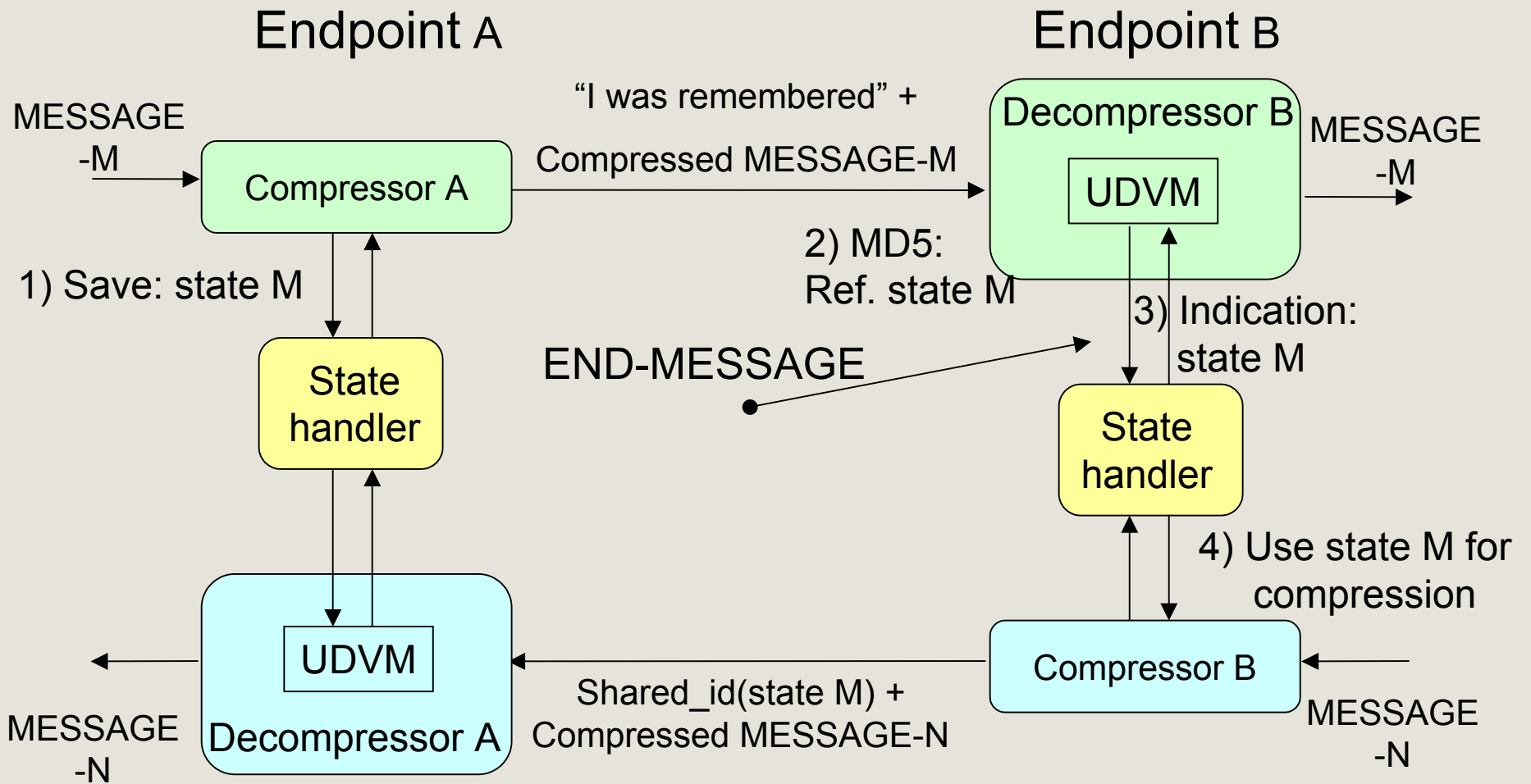


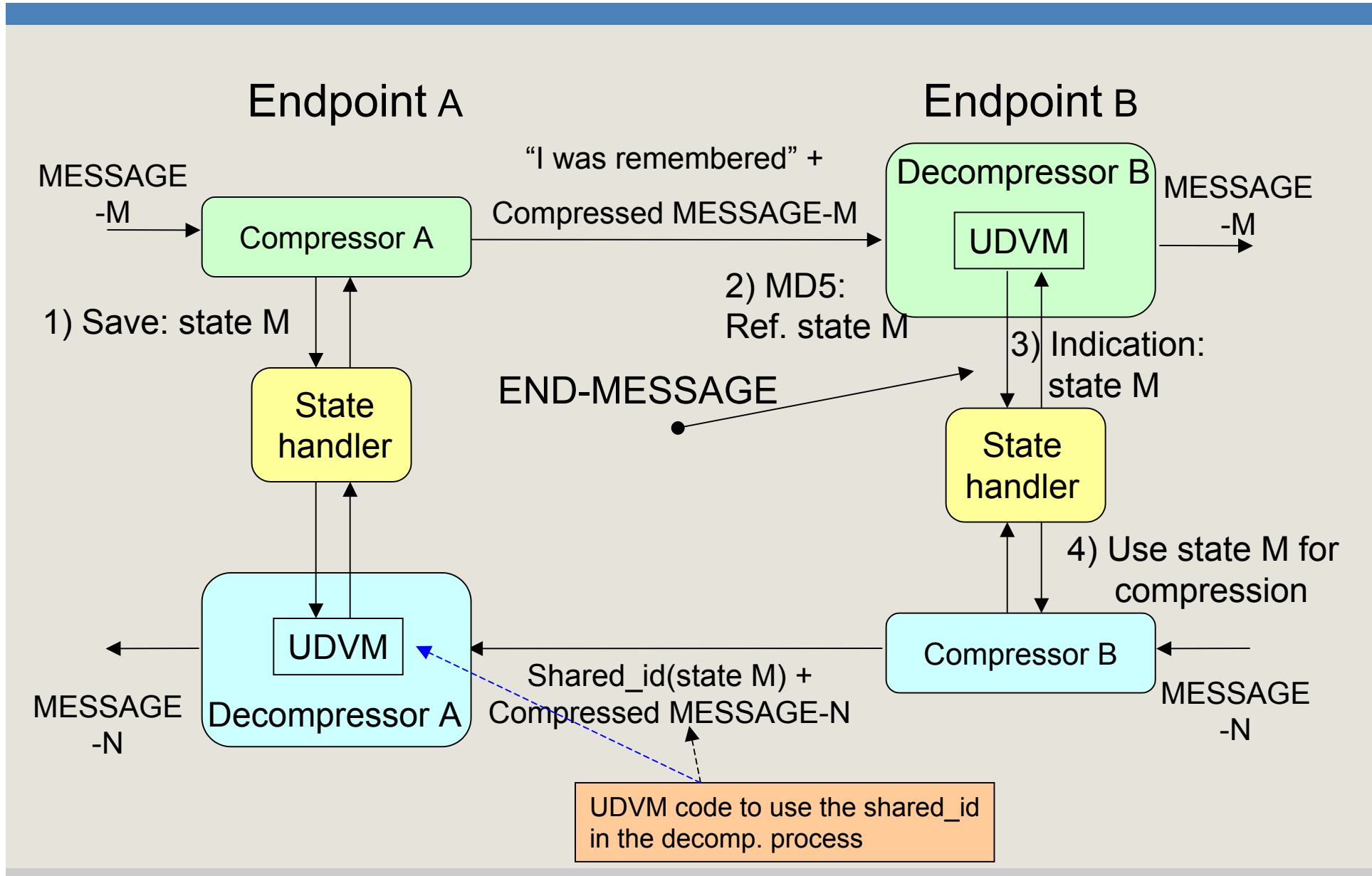






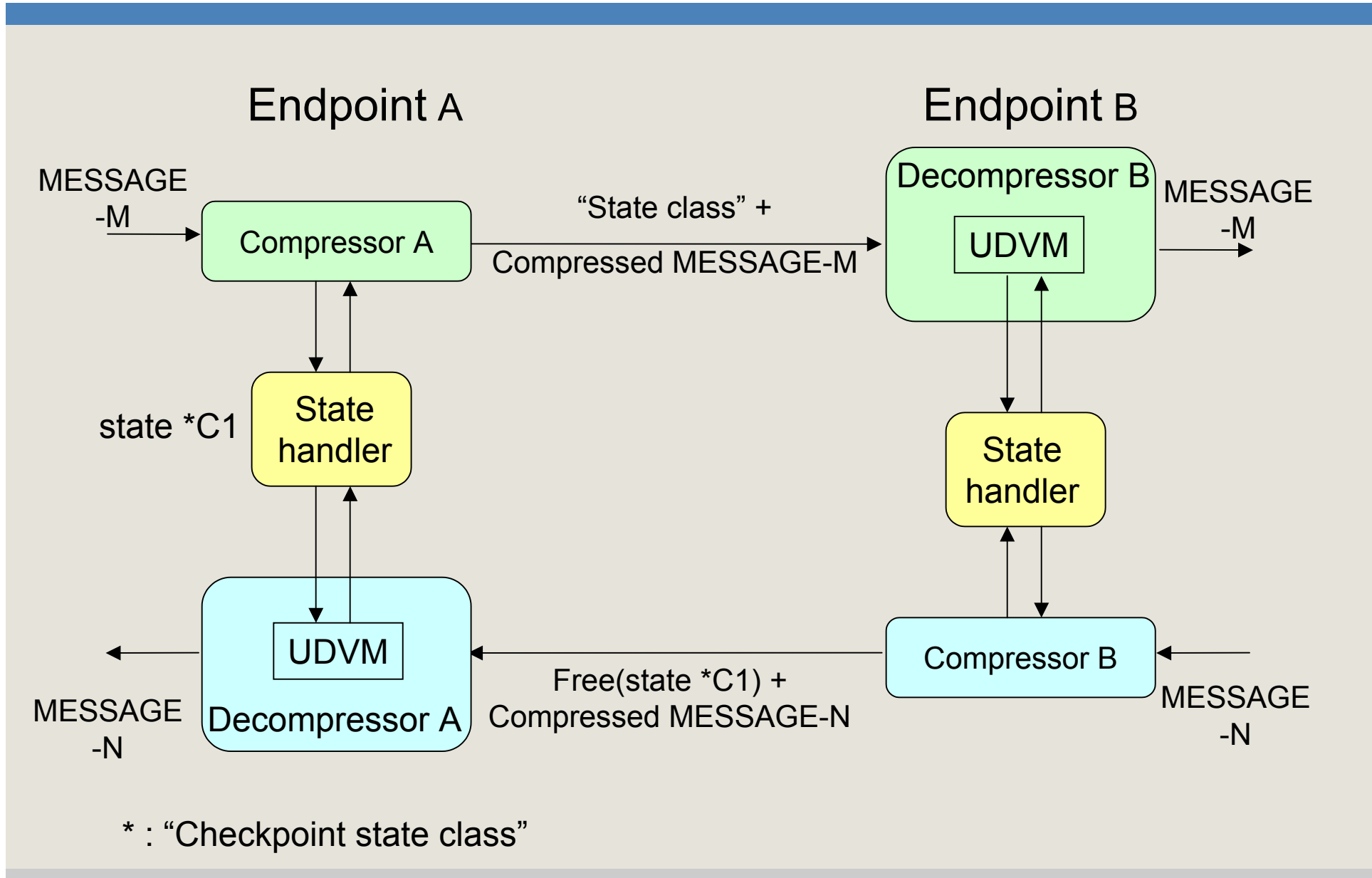


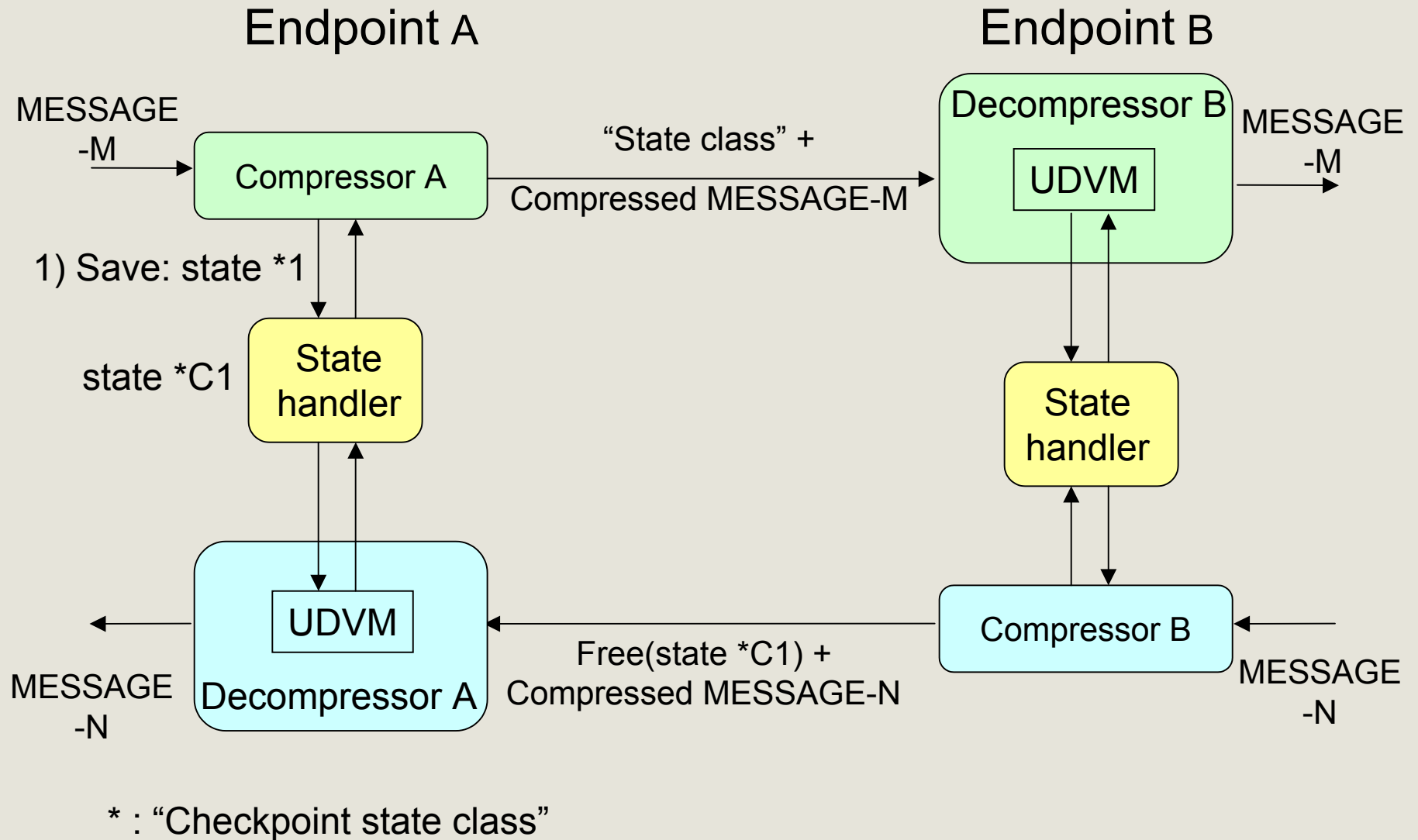


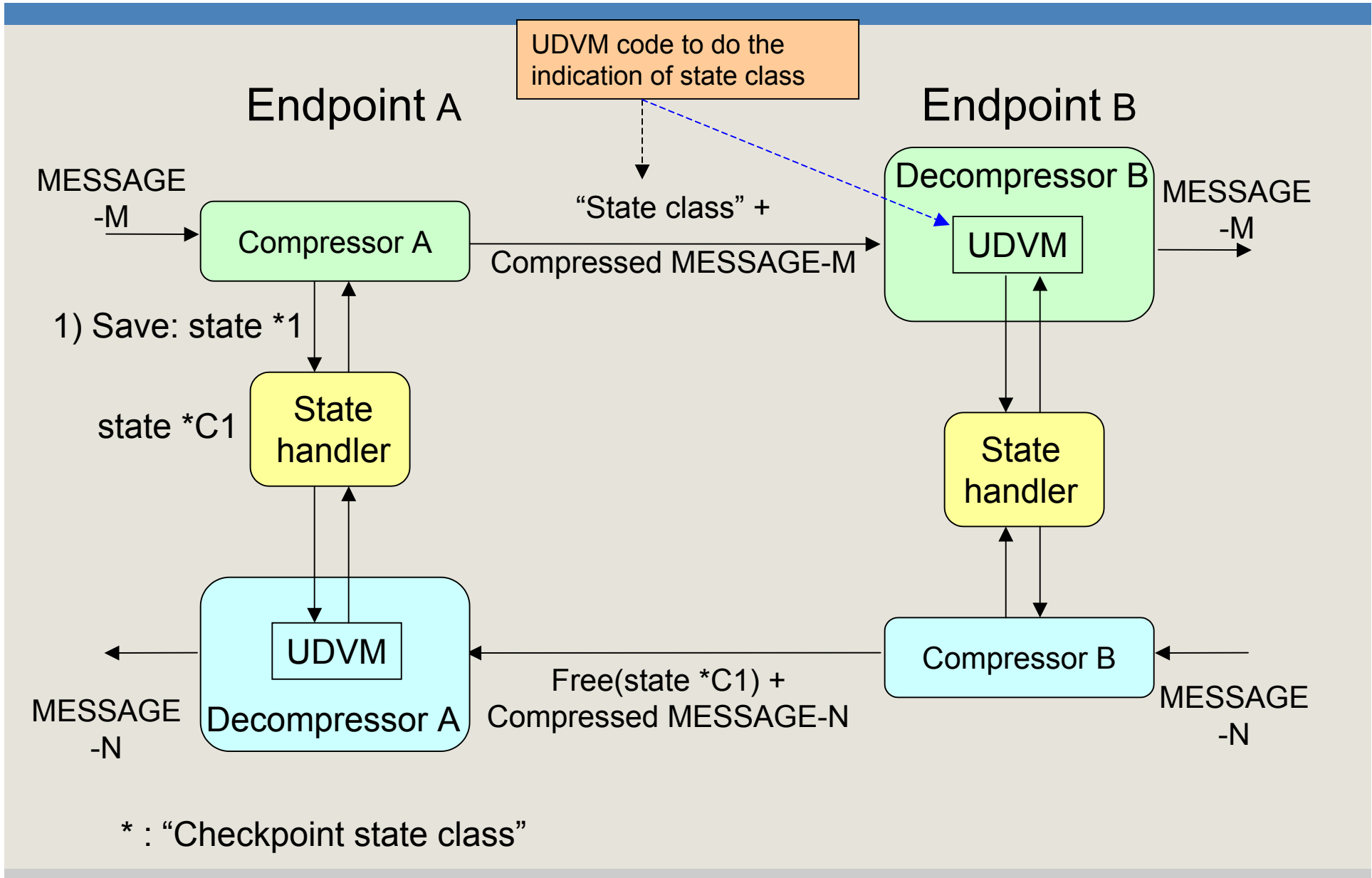


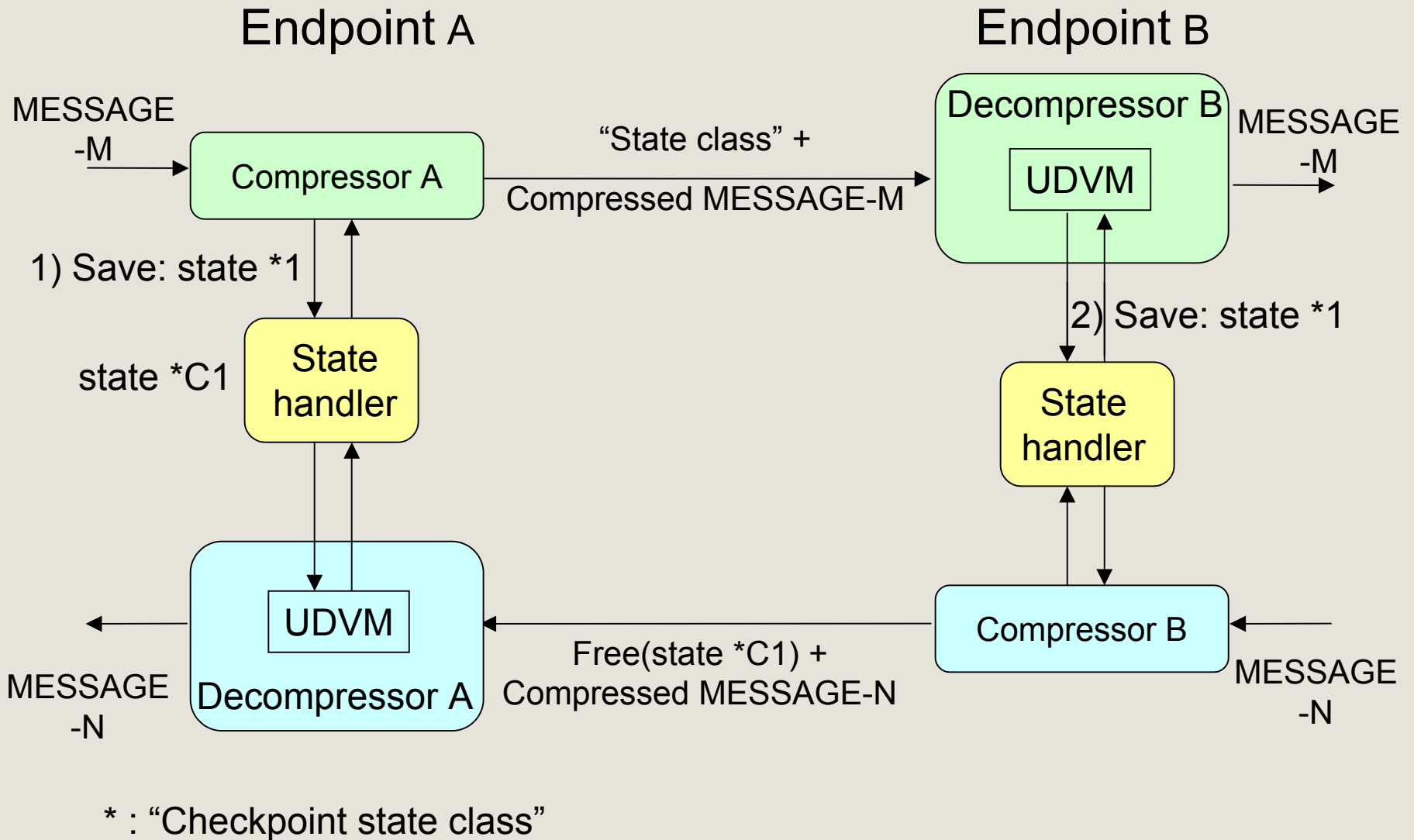
SigComp Extended – State management/(Rollback)

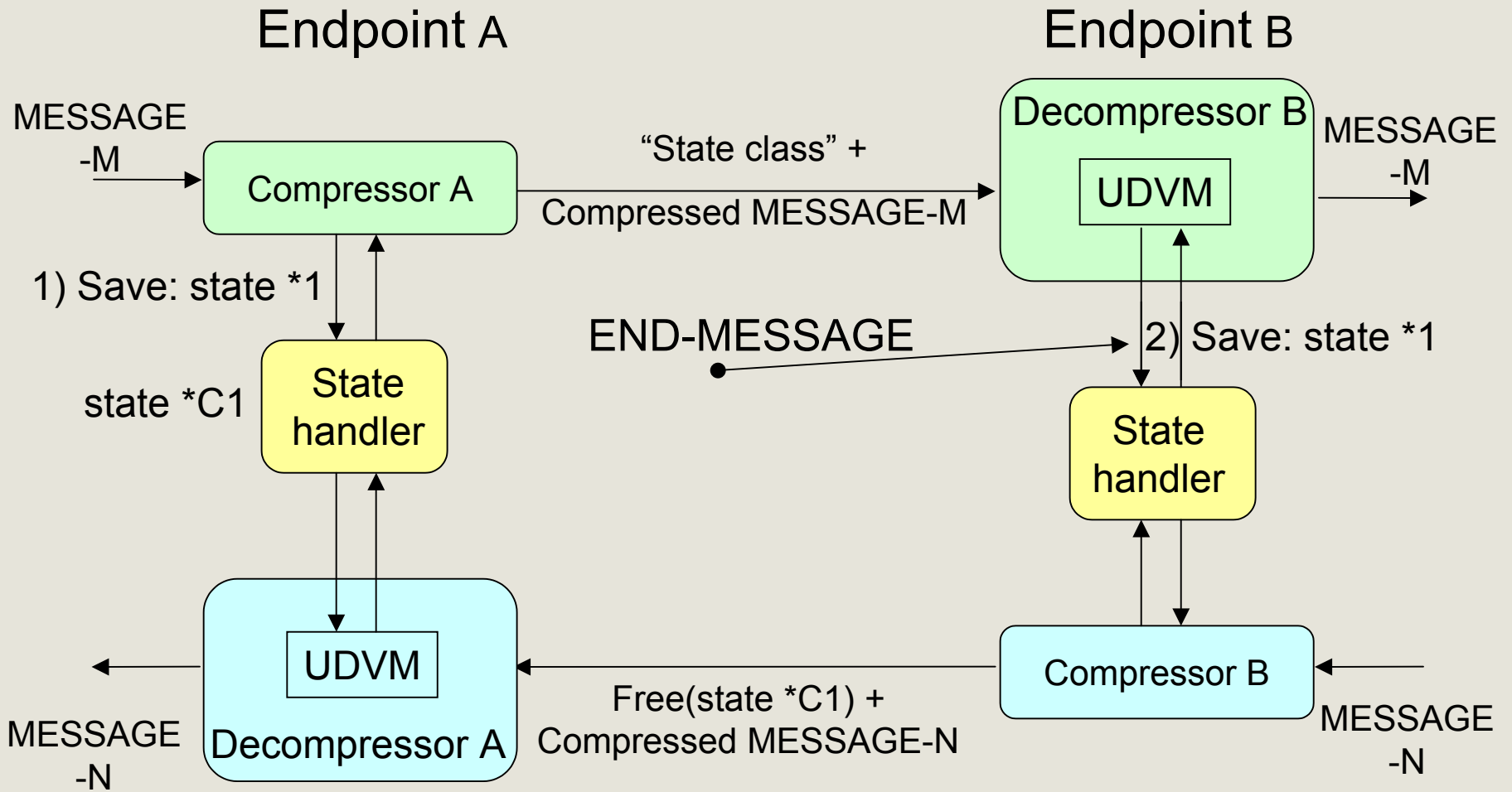
- State management/(Rollback)
 - Endpoint A indicates to endpoint B that this state should not be deleted until it is explicitly instructed by endpoint A to do so.
 - See open issue...
 - Endpoint A uses the STATE-FREE instruction to indicate to endpoint B that the state will no longer be used by endpoint A.
 - See open issue...



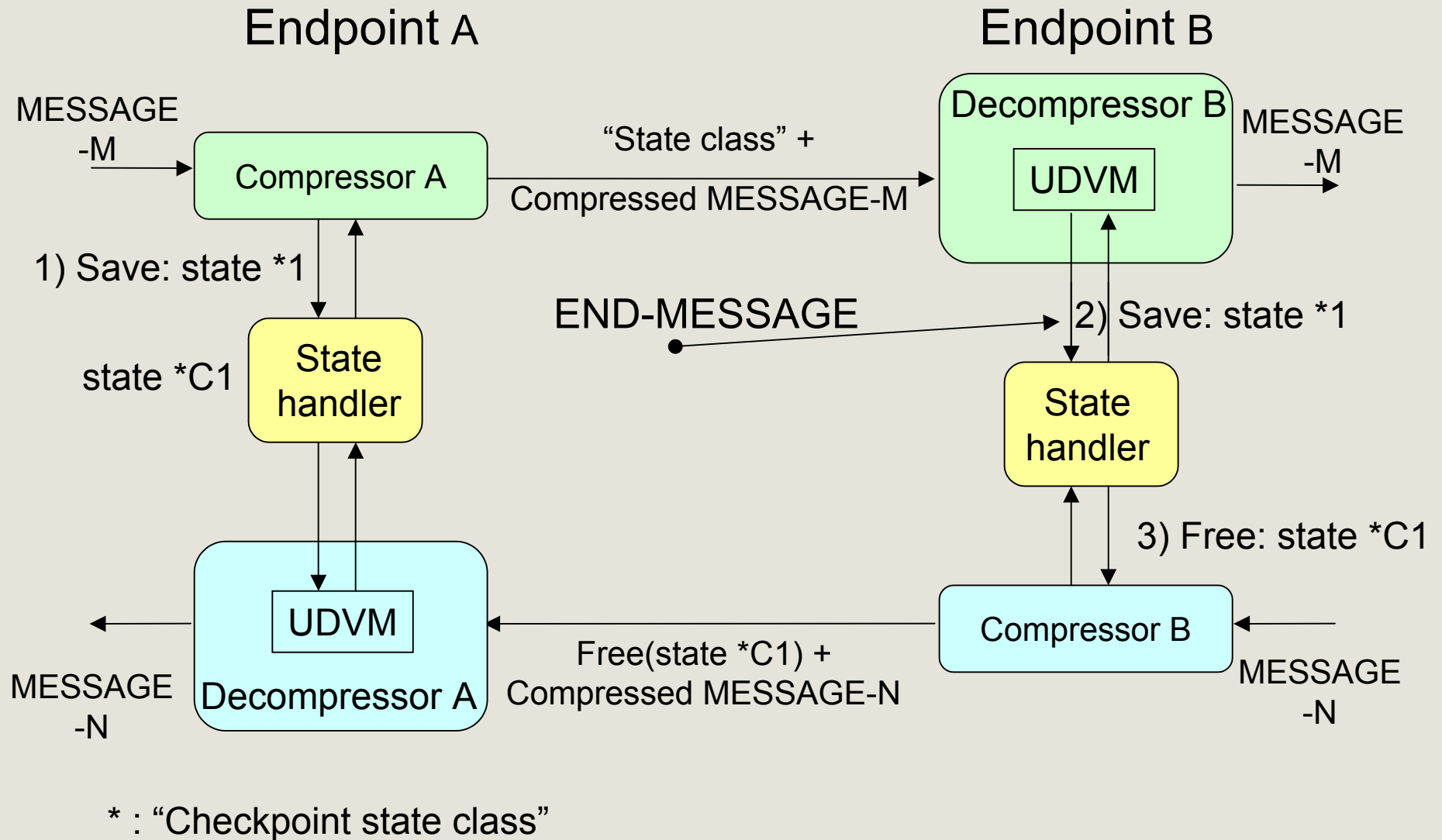


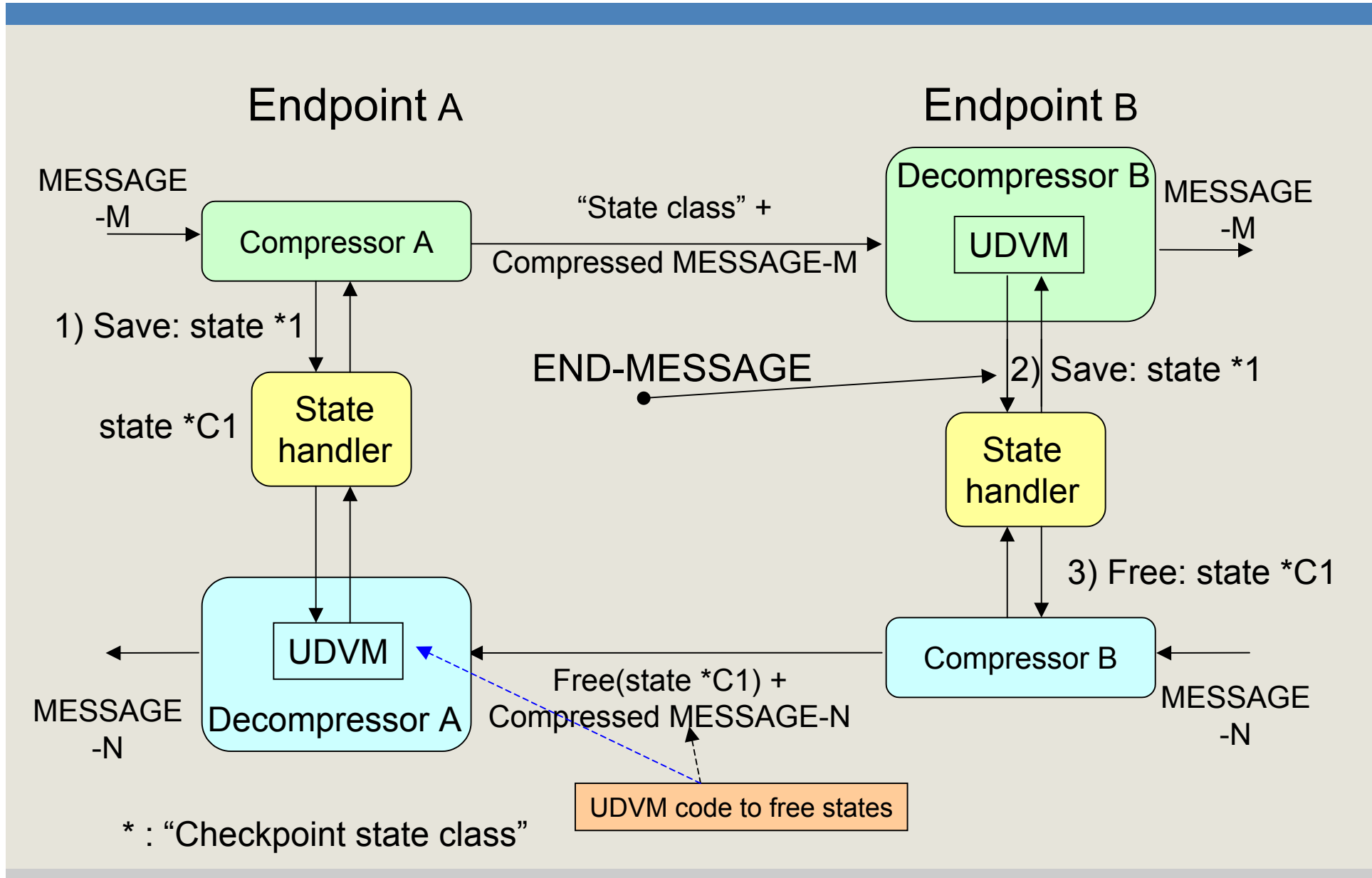


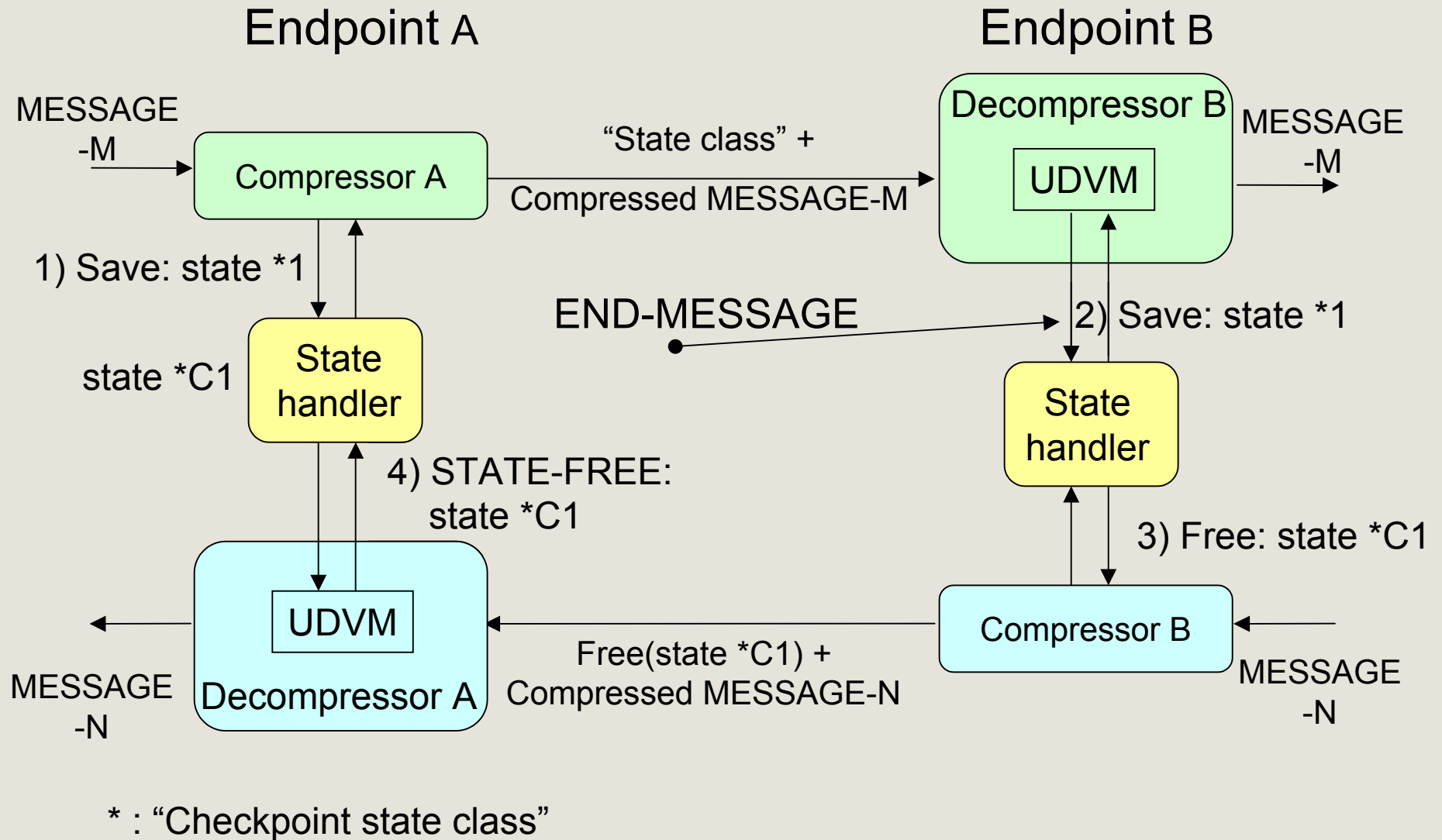




* : "Checkpoint state class"







Open issues

- As SigComp is defined now it is the remote endpoint that decides whether explicit acknowledgements is to be provided or not.
 - Opinions?
- Any new instructions or changes to existing ones to provide for State management/(Rollback)?
 - Operand to the END-MESSAGE instruction that indicates “state class”

Sigcomp Security Goals

- ◆ **Do not create *new* risks**
 - ▲ I.e., risks that are in addition to those already present in the application protocols.
 - ▲ No intention for SigComp to *enhance* the security of the protocols
 - ▲ Attacker could always circumvent by not using compression
- ◆ **do not worsen security of existing application protocol**
- ◆ **do not create any new security issues**
- ◆ **do not hinder deployment of application security**

Confidentiality risks

Attacking SigComp by snooping into state of other users

- ◆ **State can only be accessed using a state identifier**
 - ▲ (prefix of a) **cryptographic hash** of the state being referenced
 - ▲ referencing packet already needs knowledge about the state
 - ▲ default reference length of 72 bits (9 bytes)
 - ▲ Can use 48 bits (6 bytes) where birthday issue can be ruled out
 - ▲ Can use up to 96 bits (12 bytes) for additional security
 - ▲ minimizes the probability of an accidental state collision
- ◆ **Snooping state identifiers (e.g., passive attacks)?**
 - ▲ provide knowledge about the state referenced as easily
 - ▲ no new vulnerability results
- ◆ **App needs to handle state ids with the same care it would handle the state itself**

Integrity risks

- ◆ **Sigcomp itself is not making any contributions to integrity protection, but might jeopardize it:**

Attacking SigComp by faking state or making unauthorized changes to state:

- ◆ **State cannot be destroyed* or changed by a malicious sender – it can only add new state. Faking state is only possible if the hash allows intentional collision.**
- ◆ ***) limited memory ➡ could lose information from FIFO**
 - ▲ **Rely on endpoint identification provided by application!**

Availability risks (avoid DoS vulnerabilities) (1)

Use of SigComp as a tool in a DoS attack to another target

- ◆ **SigComp as an amplifier in a reflection attack?**
 - ▲ **SigComp only generates one decompressed message per incoming compressed message.**
 - ▲ **This packet is then handed to the application; the utility as a reflection amplifier is therefore limited by the utility of the application.**
- ◆ **However: SigComp can be used to generate larger packets as input to the application than have to be sent from the malicious sender;**
 - ▲ **Attacker can send smaller packets (at a lower bandwidth) than are delivered to the application.**
 - ▲ **Depending on the reflection characteristics of the application, this can be considered a mild form of amplification.**
 - ▲ **The application MUST limit the number of packets reflected to a potential target – even if SigComp is used to generate a large amount of information from a small incoming attack packet.**

Availability risks (avoid DoS vulnerabilities) (2)

Attacking SigComp as the DoS target by filling it with state

- ◆ **Excessive state can only be installed by a malicious sender (or a set of malicious senders) with the consent of the application.**
- ◆ **SigComp + application are approximately as vulnerable as the application itself, unless it allows the installation of state from a message where it would not have installed state itself (“gratuitous state”)**
- ◆ **Might be desirable to increase the compression ratio**
 - ▲ **mitigate by adding feedback at the application level that indicates whether the state requested was actually installed**
 - ▲ **allows system under attack to gracefully degrade by no longer installing gratuitous state**

Availability risks (avoid DoS vulnerabilities) (3)

Attacking the UDVM by faking state or making unauthorized changes to state

- ◆ **(See "Integrity risks" above.)**

Attacking the UDVM by sending it looping code

- ◆ **App-defined upper limit to number of "CPU cycles" that can be used**
 - ▲ **E.g., 4 cycles per bit; add 1000 bits per compressed message**
- ◆ **Damage inflicted by sending packets with looping code is therefore limited, although this may still be substantial if a large number of CPU cycles are offered by the UDVM**
- ◆ **(This would be true for any decompressor that can receive packets from anywhere)**

Sigcomp: Implementation status

- ◆ **Implementations in progress at**
 - ▲ dynamicsoft
 - ▲ Roke Manor
 - ▲ TZI <http://www.dmn.tzi.org/ietf/rohc/udvm/>
 - ▲ Assembler: 250 lines,
UDVM: 520 lines (~ 750 est. when complete)
- ◆ **Interoperable code exchanged for**
 - ▲ Simple LZ77, DEFLATE, LZW
 - ▲ More in progress
- ◆ **Need to do interop testing on state management**
 - ▲ Exercise –extended code
- ◆ **Need to do testing in real setting (e.g., TCP record mark)**

SigComp – What now? 1(2)

- **SigComp discovery**

1. SigComp messages can be distinguished from uncompressed messages
2. Apart from that, this is a non-SigComp issue
3. In 3GPP, mandated SigComp support together with 1.) above makes the problem avoidable in that environment

SigComp DONE!

- **Static dictionary**

- Not a SigComp, but a “SigComp for SIP” standardization issue
- Must be done now for 3GPP SIP compression
- ROHC/SIPPING cooperation

SigComp DONE!

SigComp – What now? 2(2)

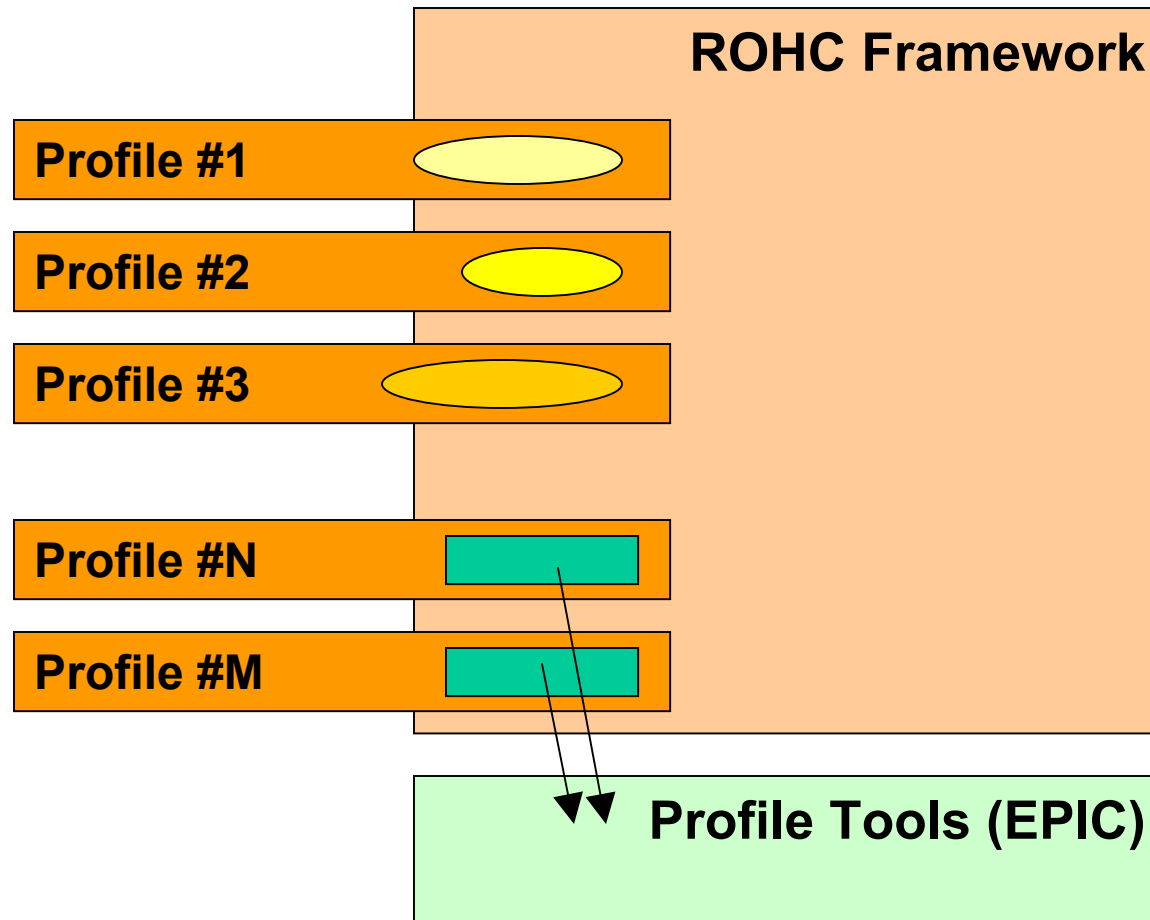
- **Additional instructions? Remove instructions?**
 - **No major decisions left, most cosmetics!**
- **The security issues have been addressed**
- **We have RUNNING CODE!!**
- **All issues that have been clearly (by [name-NN], e.g. [cabo-15]) raised on the list will be tracked and solutions or clarifications to these will be provided**
- **SigComp–06 and Extended-03 on April 5th (submit)**
- **WG last-call these documents on **April 8th****

Role of EPIC in the ROHC work, 1(2)

Taken from the SLC presentation by Mark West

- **What EPIC is...**
 - **EPIC is about generating packet formats**
 - Allows the packets between compressor and decompressor to be described at a higher level
 - Automatically generates highly efficient formats
- **What EPIC is not...**
 - **It is not a complete framework for header compression**

Role of EPIC in the ROHC work, 2(2)



An RTP Profile for EPIC(-lite)

Mark West
(mark.a.west@roke.co.uk)

Why do an RTP profile?

- EPIC-lite profiles have been accused of being obscure!
- With a profile such as TCP it is necessary to
 - Understand the structure of the profile
 - Understand what it says about TCP
- Both of these can be challenging...

- RTP compression is well-understood and documented in depth in RFC 3095
- Provide a subset of this functionality as an EPIC profile
- Make an easier basis for comparison

What the profile *isn't*

- It is not a complete RTP compression solution
 - It only handles IPv4/UDP/RTP
 - It only describes the equivalent of UO mode packets
- It does not attempt to be bit-wise compatible with (any of) the headers in RFC 3095
 - Shares the same prefix bits for IR and IR-DYN packets

A quick tour

- Are there any particularly interesting bits of the profile?!
- What issues are raised by these?
 - 3-bit vs 7-bit CRC selection
 - IP ID byte-ordering
 - UDP checksum used / not-used
 - Offset encoding
 - CSRC list

CRC length

- The profile includes 2 branches for compressed packets, one with a 3-bit CRC and one with a 7-bit CRC
- This is not an exact match for RFC 3095
- However, it is a close match
- Ensures that packets which update significant amounts of context use 7-bit CRCs

non-random-ip-id-co3 = STATIC (70.71%) | LSB (5, 0, 29.29%)

non-random-ip-id-co7 = STATIC (63.1%) | LSB (5, 0, 26.14%) |
LSB (9, 0, 10.76%)

IP-ID byte ordering

- Mentioned because of much debate on mailing list
- NBO processing is (nearly) orthogonal to other compression issues
(As is scaling)
- Therefore, decided to split these functions out
- NBO decision (or scaling) must be performed on a field before it is encoded

```
ip-id    =    NBO(16) ; check the byte-order  
  
compress-ip-id  
  
STATIC(99.9%) | IRREGULAR(1,0.1%) ; NBO flag
```

UDP checksum

- This demonstrates a use of the 'FORMAT' construct
- Flows will either never use the UDP checksum or always use it
- Point of FORMAT is that changes are expected only rarely
- The selection of the FORMAT is based on state information in the context, rather than per-packet indicator

```
udp-checksum-co          =      FORMAT (no-checksum,with-checksum)
```

```
                          STATIC (100%) ; FORMAT index
```

```
no-checksum             =      VALUE (16, 0, 100%)
```

```
with-checksum          =      IRREGULAR (16, 100%)
```

Offset Encoding

- There is a slight difference from the way that RFC 3095 describes (for example) IP ID encoding
- When offset encoding is used in EPIC, it is applied consistently
 - So, with IP ID, it is *always* the offset from the RTP SN that is encoded
 - This does not affect efficiency

CSRC list

- Makes use of EPIC LIST-based encoding
- LIST has a number of possible entries
- Individual entries are sent along with ‘presence’ flags and ‘order’ information
- Interesting to compare with RFC 3095 list-based compression
 - But we haven’t done this yet!

Performance

- We have run this profile through our EPIC interpreter
- Run several of the test flows used in interoperability tests to check that it works
- Compression performance is directly comparable with that offered by RFC 3095 on the same flows
- Don't yet have a compiled version of the code to test processing load

Average CO packet sizes

	ROHC-RTP	EPIC-lite
Call-2 RTP voice call with talk-spurts	1.3	1.2
Call-3 As above, but with IP-ID jitter	2.0	1.7
Call-5 IP TOS change	1.6	1.4

Conclusions

- Can write an RTP profile for EPIC
- Essentially equivalent to the packets defined in RFC 3095
- Useful as a discussion point for EPIC notation / processing

EPIC-lite Status and Open Issues

Mark West
(mark.a.west@roke.co.uk)

Status – of the draft

- Basic definition and pseudo-code is believed to be stable
 - Some minor clarifications and updates required as a result of implementation effort...
- May benefit from additional clarification of where EPIC fits in to overall header compression

Status - implementations

- We have a functional test-bed implementation of the pseudo-code
- Demonstrates
 - Tree-building for EPIC-lite packet formats
 - Compression
 - Decompression
- Have run profiles for test protocols, RTP, TCP and SCTP

Status - Implementations

- Alternative implementation
 - University of Split
- Have just (this week!) achieved basic interoperability
- Built the same packet formats from a simple profile
- Exchanged compressed packets
- Verified correct decompression
- Moving on now to more complex profiles with greater variety of encoding methods

Open Issues

- Where does EPIC fit in?
- Profile complexity
- Encoding methods
 - General
 - FORMAT
 - LIST
 - Stack manipulation
- Further interoperability
- Performance

Where does it fit?

- Need to clarify where EPIC(-lite) fits into header compression solutions
- Think more about structure of the solution and interfaces to external components

Profile Complexity

- Again, topic of discussion on mailing list
- Profile structure is actually quite simple
 - Based on BNF
 - Just 'and' and 'or' combinations
- Profiles are quite 'dense' with respect to information content, however
- Try to mitigate this through clear structure and more comments in the profiles
- Also bear in mind trade-offs between complexity and efficiency

Encoding Methods

- In general, it is designed to be easy to add new encodings, where necessary
- Each encoding has 3 basic methods to support
 - Tree building
 - Compression
 - Decompression
- Currently methods are defined in pseudo-code

FORMAT

- There have been discussions about the use of FORMAT
- FORMAT is designed to capture changes behaviour based on context rather than per-packet indicators
- Need to be sure that use is appropriate
- Cost of FORMAT change is sending IR-DYN packets
 - ‘Horizon effect’ – only worth changing if flow will continue long enough to make it worthwhile
 - But could defer FORMAT change until an IR-DYN was going to be sent anyway, for example
 - This is one example of state-model interaction
 - Another is where FORMAT triggers a state change

FORMAT

- FORMAT is clearly useful for handling cases such as
 - ECN used
 - No ECN used
- Less clear in cases such as TCP flow behavior
- However, still thought to be worthwhile (but with some clarification)
- Stress that choosing to change format is typically a compressor-local optimisation decision

LIST

- Main concerns about LIST encoding is efficiency
- Two aspects to this
 - Information about the presence of LIST entries
 - Information about the order of LIST entries
- Presence information can easily be encoded to take account of ‘SYN only’ options, for example
- Order information is more complex
 - Previous list encoding aimed for long-term efficiency
 - LISTs such as used for TCP options and SCTP chunks may raise slightly different considerations
 - Most LISTs are ‘sparse’ and order information should reflect this
 - More thought required...

Stack Manipulation

- Used to process header fields
- Allows interactions between fields to be processed
- But, drifts into “solution space” rather than just defining how the stack behaves
- So, we may want to think about this notation

Interoperability

- Initial interoperability results are encouraging
- Have shown that same packet formats and identifiers can be built from a profile
- More tests are obviously needed
 - Extend coverage of encoding methods
- If anyone else wants to join in, we'd be delighted to accommodate them!

Performance

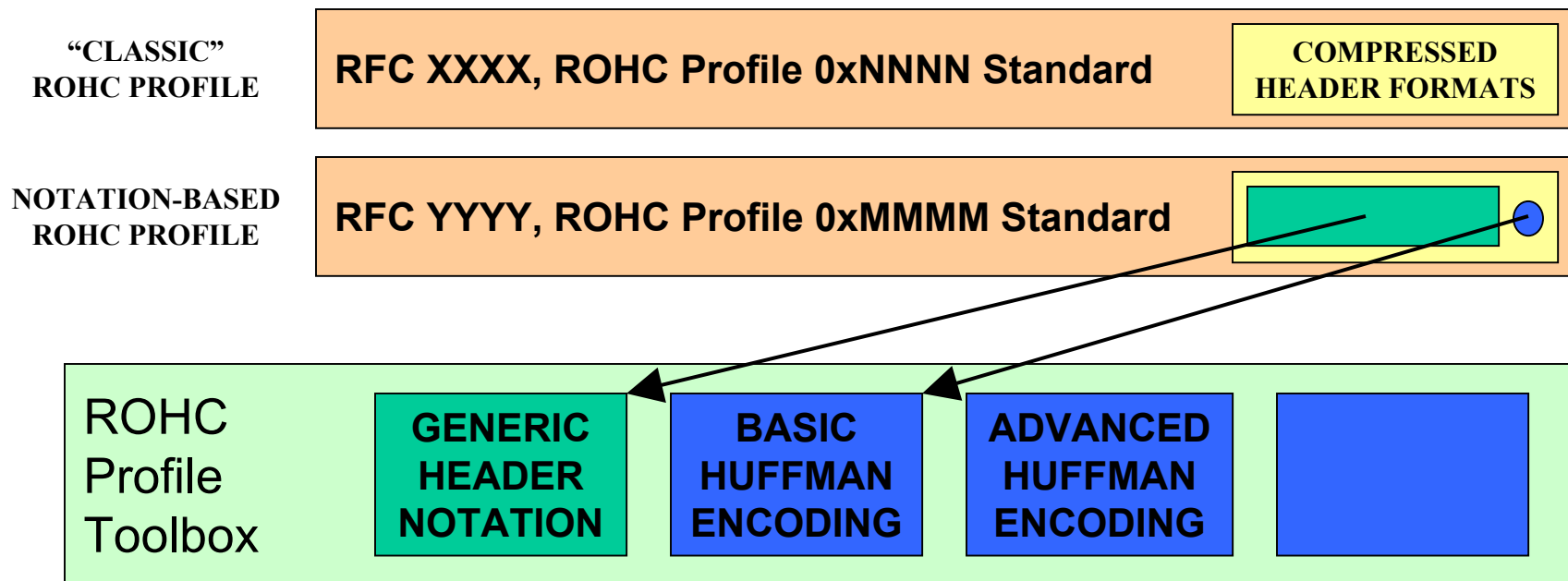
- We can't easily get CPU performance metrics from our implementation
(we're running it in Python and it works ok!)
- We're starting to run profiles to get compression figures
 - U-mode only at the moment
- Hopefully compiled implementations (e.g. from Univ. Split) will give useful results

Generic Notation - Way forward, 1(2)

- 1. Create a new WG document:**
 - “ROHC: Generic Compressed Header Notation”
 - Based on the notation part of the EPIC-Lite document
- 2. Evolve the notation language by applying it in the development of TCP and SCTP profiles**
- 3. Develop one or several compressed header encoding mechanisms that use the generic notation as input**

Generic Notation - Way forward, 2(2)

- **New profiles could then be defined with the generic header notation instead of written header formats, each referring to one specific encoding method**



TCP Requirements and Field Behavior

Mark West
(mark.a.west@roke.co.uk)

Requirements

- Requirements are stable
- Most important issues now under consideration
- Main missing element from profile
 - IPv6 encoding
 - Tunnel headers

Behavior Draft

- Included many comments, suggestions and corrections to –00
 - Thanks!
- May still be some more points to capture
 - Not clear how useful it is to go much further
- How slavishly should this map to a profile?
 - Some recent discussion suggests that this might become counter-productive
 - Excessive profile complexity
 - May not increase compression efficiency

Issues

- Short-lived connections
- Degree of robustness
- Implicit acknowledgements
- Use of master sequence number
- Re-ordering Channels

Short-Lived Connections

- Context replication is the obvious way to exploit inter-flow redundancy
- However, still issues to resolve
 - What degree of granularity is appropriate?
 - What happens if the base-context (to be copied) is not available (or is corrupt) at the decompressor?
- Should short-lived connection efficiency be achieved at the expense of long-lived connection efficiency?

Degree of Robustness

- We have broadly agreed on a sensible level of robustness
- However, ‘packet-centric’ view is hard to quantify for TCP
- Will see more clearly when we start running some tests

Implicit Acknowledgements

- One commonly considered implicit ack is the first packet following a SYN
 - Since this implies that the SYN has been received
- This is a good hint
- However, it can only be considered 'safe' as an ack if there is no other path for the SYN to have been retransmitted...
- Need to think carefully about where it is sensible to use this

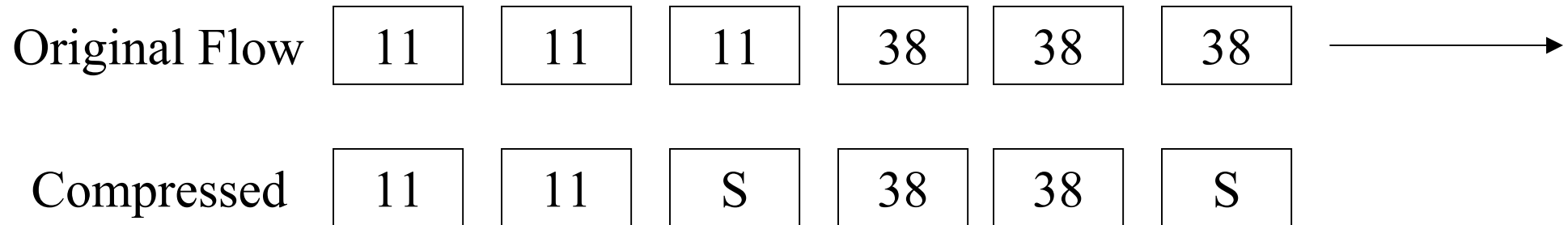
Use of MSN

- In TCP this is clearly only needed for acknowledging packets
- We probably only need to acknowledge certain packets
- Thus the MSN will only be used on some packets
- Considered in more detail in the TCP compression solution

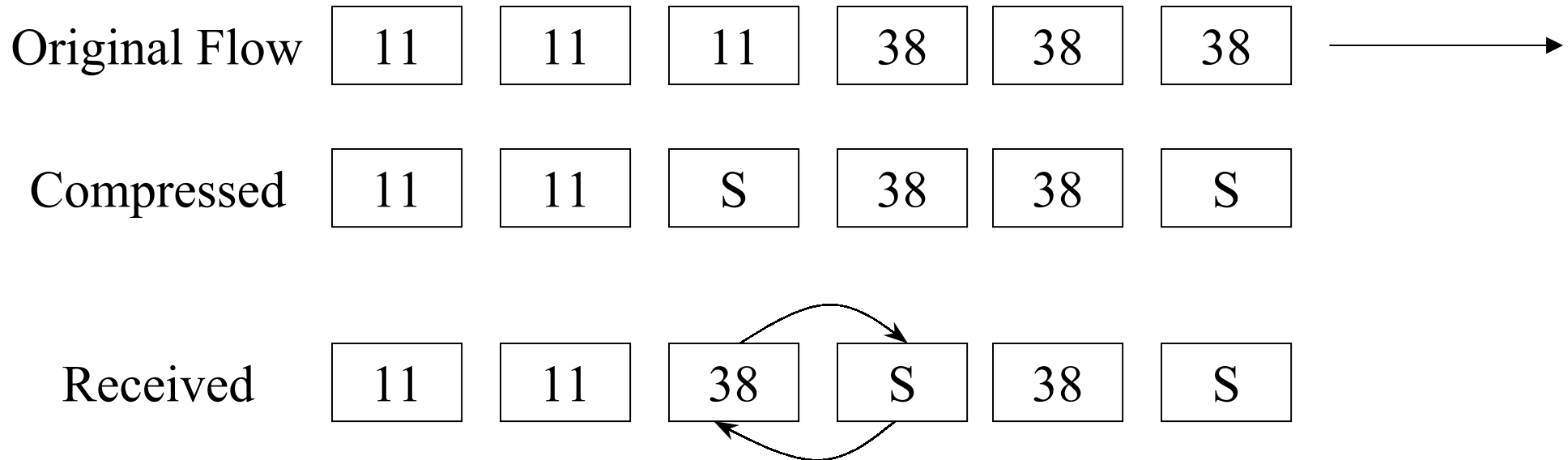
Re-Ordering Channels

- Is this part of the header compression framework?
- Essential requirement is for an external sequence number (from the 'link' layer)
 - Cannot use MSN (since this is not available until after successful decompression!)
- Decompressor need to maintain a context history
- Sequence number determines which historical context at decompressor is used as base
- This appears to be a decompressor-local decision (in conjunction with the link)

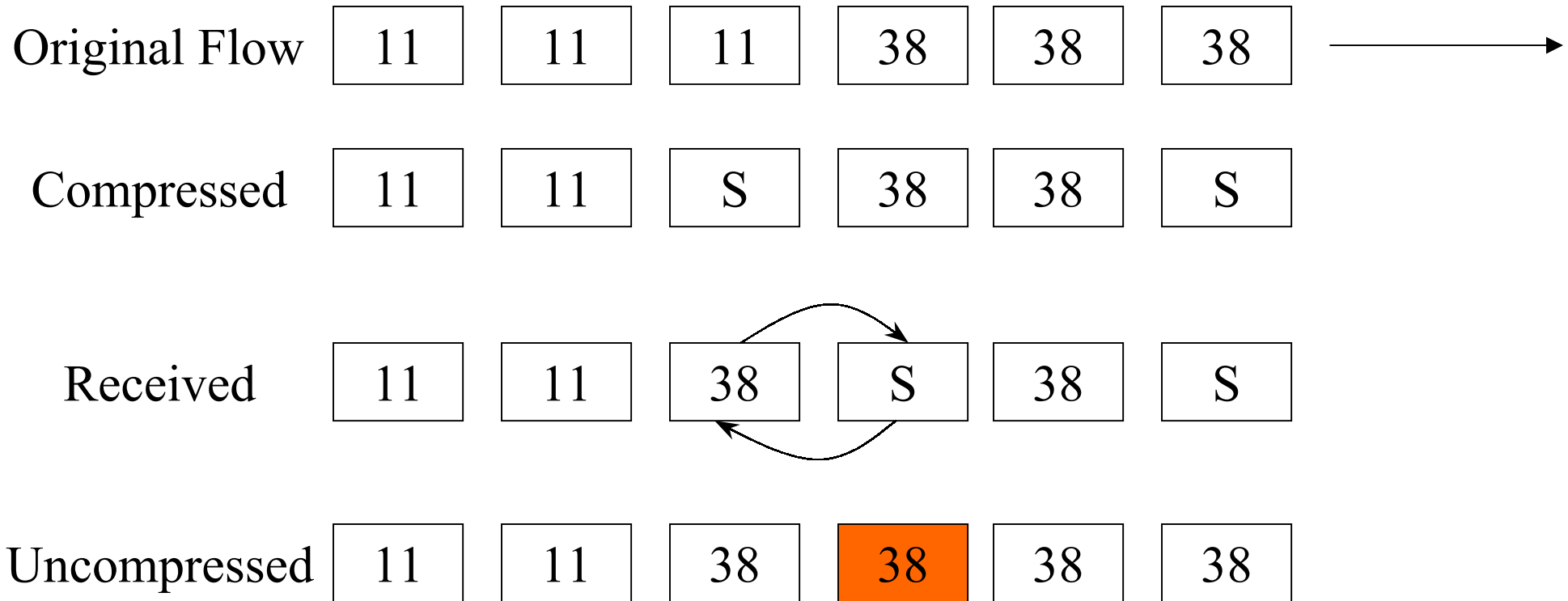
Re-ordering example



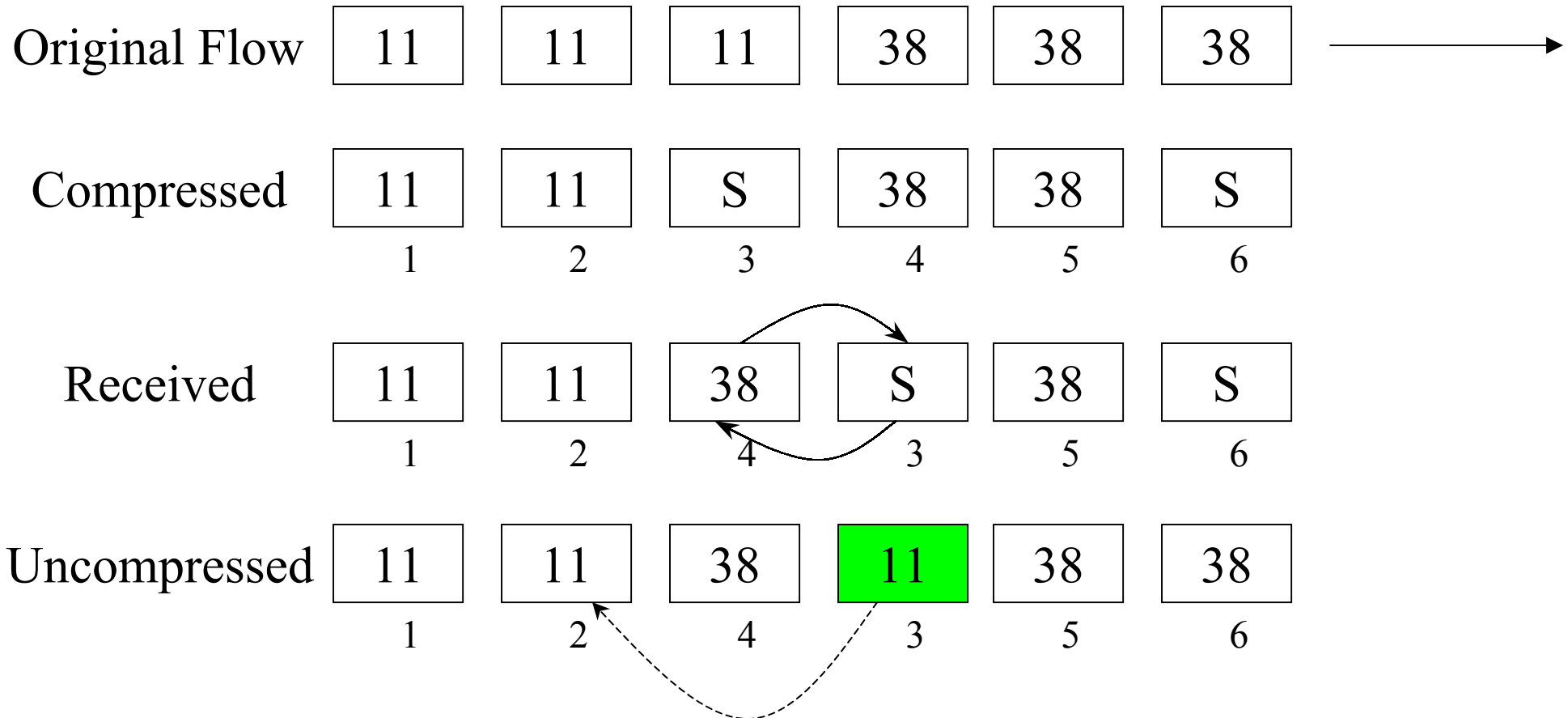
Re-ordering example



Re-ordering example



Re-ordering example



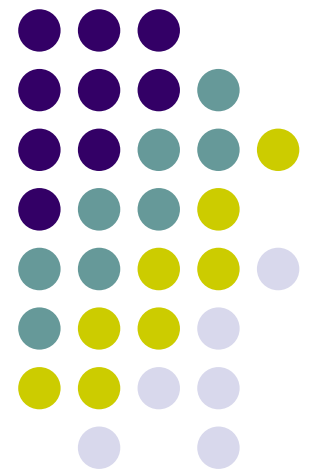
Overall

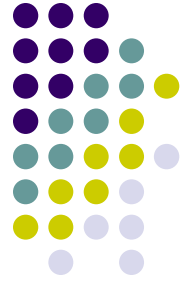
- Think we have a good handle on TCP behavior
- TCP requirements are stable
- Need to work more on mapping the behavior into a profile
- Need to ensure that we have a solution that meets the requirements

ROHC-TCP: TCP/IP Header Compression for ROHC

Qian Zhang
Microsoft Research

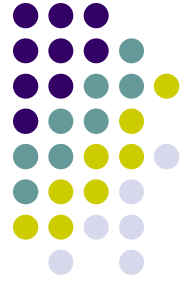
[draft-ietf-rohc-tcp-00a.txt](#)
[draft-ietf-rohc-tcp-00.txt](#)





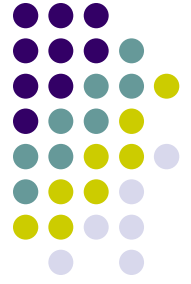
ROHC-TCP

- Robustness/ Efficiency
 - Refine states, modes, operations in different modes
- Support for TCP options
 - MSS, WSopt, SACK-permitted (SYN packet)
 - Timestamp, SACK
- Short-lived TCP transfers
 - Context replication/updating
- Packet format generation
 - Further analysis for TCP behavior
 - Some issues (correlation and option support) in EPIC-LITE



Framework for ROHC-TCP (1)

- Guideline for compressor's state transition
 - Variation in packet headers
 - Positive feedback from decompressor (ACK)
 - Negative feedback from decompressor (NACK)
 - **Robustness confidence level**
- Operation modes
 - U-mode and B-mode (O-mode)
 - No need to send feedback in per-packet base
 - Header formats are almost the same for U and B modes
 - MSN (master sequence number) support in some packets in B mode



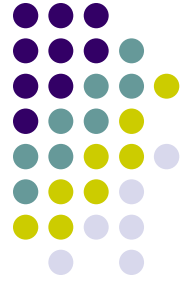
Framework for ROHC-TCP (2)

- Verification on the decompressor
 - Packet loss may occur, residual BER is quite small
 - Cons for TCP checksum
 - E2E rather for hop-by-hop
 - Computation complexity
 - CRC for different types of packets
 - 3~7 bits CRC for compressed packet
 - RTP case: 3 for smallest packet (?)
 - 8bit for IR/IR-DYN/IR-UPDATE



Protocol for ROHC-TCP (1)

- Robustness and Efficiency maintenance
 - Error avoidance (key for ROHC)
 - W-LSB for most TCP/IP fields
 - Context window, the number of context value, indicates the robustness
 - Error detection and recovery
 - TCP protocol itself
 - Control context window to achieve the balance of robustness and efficiency
 - Feedback in B-mode provide a way to control
 - TCP congestion window provide implicit feedback in U/B-mode
 - How to estimation TCP congestion window is an implementation option (optimization)



Protocol of ROHC-TCP (2)

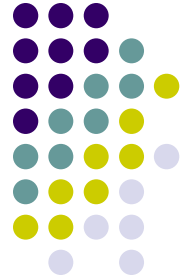
- Operation in U-mode
 - Upwards transitions (Optimistic / F-optimistic)
 - Robustness confidence level
 - Downward transitions (Update / F-update)
 - Robustness confidence level
 - Optional enhanced operations
 - Speed up IR to FO transition if SYN packet pass through the compressor
 - Optional operation in IR state
 - Full (partial) packets pattern: slow start
 - Optional operation in all the states
 - Use estimated TCP congestion window to control context window for transition more efficiently



Protocol for ROHC-TCP (3)

- Operation in B-mode
 - Upwards transition (Optimistic/ACK, F-optimistic/ACK)
 - Robustness confidence level + ACK
 - Downward transition (NACK / Update, F-update)
 - Robustness confidence level + NACK
 - MSN for feedback
 - Add MSN as an additional field for packets in B-mode
 - Optional enhanced operations
 - Optional operation in all the states
 - TCP Congestion window to control context window for transition
 - Optional operation for MSN
 - May not necessary to append a MSN for each packet

Protocol for ROHC-TCP (4)

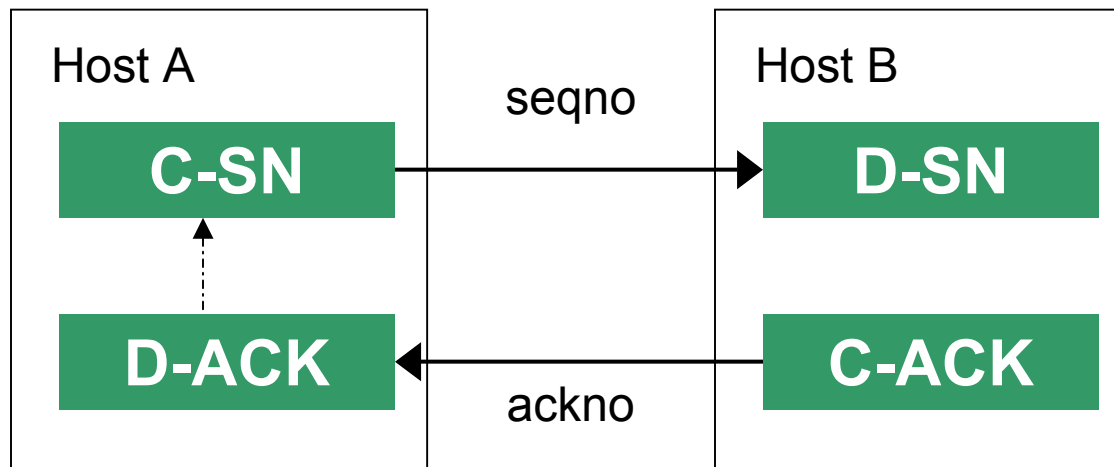


- Feedback and MSN related issues
 - Should ROHC-TCP support feedback in U-mode?
 - Current version: not necessary, so not supported
 - No MSN issue involve
 - MSN for feedback to acknowledge packet
 - No candidate to offer MSN for TCP/IP case
 - IP-ID in IPv4
 - ? (combination of seqno and ackno) for IPv6
 - Provide an additional field for MSN in some packets
 - Format for feedback?
 - Prefix “11110” had been reserved in ROHC
 - No need to use EPIC-LITE to generate the corresponding feedback packet formats

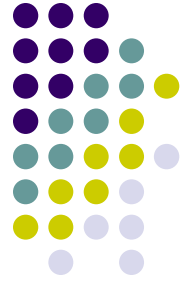


Protocol for ROHC-TCP (5)

- Implementation Options
 - Tracking-based TCP congestion window estimation
 - Bi-directional deployment
 - forward and reverse paths of the same TCP connection share the same link



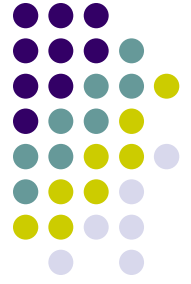
Shrink the context window size based on feedback from D-ACK to C-SN



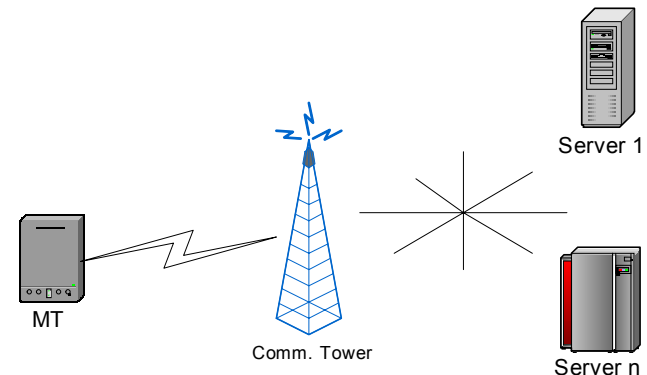
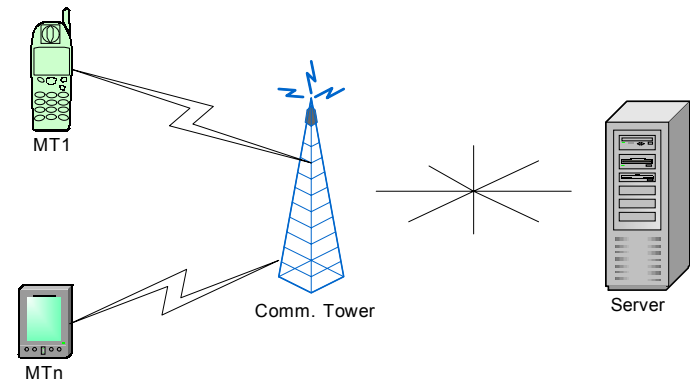
TCP Options

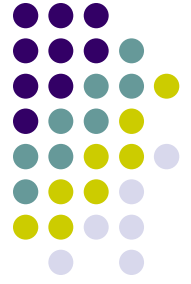
- May occur in any order
- Options may be sent only in SYN segment
 - Maximum segment size (MSS)
 - Window Scale Option (WSopt)
 - SACK-permitted
- Option with undetermined pattern and occurrence (No-operation)
- Timestamp
- SACK (may have 1-4 sack blocks)

Short-lived TCP Transfers (1)



- Three scenarios
 - Multiple connections between same source and destination
 - Multiple mobile terminals download web from the same server over cellular links
 - One mobile terminal send requests to multiple web servers over cellular links
- Criteria to determine contexts shareable / replicate-able?
 - Simple solution: same source-IP and short time interval

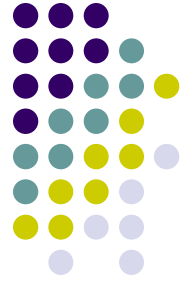




Short-lived TCP Transfers (2)

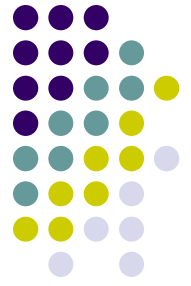
- Shareable analysis for TCP/IP fields
 - IP-ID, destination address/port, SYN-related options, Timestamp, TOS, TTL
- Context replication to improve performance
 - Re-initialize a new context from an existing one and overwrite some of values to create a new context
 - Possible encoding method for shareable fields
 - LSB for IP-ID, Timestamp
 - Delta (to original context) for port
 - How to make sure the correctness of the context that to be replicated, especially in U-mode?
 - A simple solution: send the IR packet when jump to IR state
 - Format: IR-UPDATE (“11111101”)

Further Discussion on TCP/IP Behavior



- Further discussion on TCP/IP behavior
 - “Coarse-correlation” among several TCP/IP fields
 - Seqno/ackno
 - most TCP connections only have one-way traffic
 - only Sequence Number changes and Acknowledgement Number remains constant at most time, or
 - only Acknowledgement Number changes and Sequence Number remain constant at most time
 - Options in SYN packets
 - May not occur in SYN, but can not occur in other packets
 - Undetermined order and presence for options

Compressed Format Generator



- Issues related to EPIC-LITE
 - List encoding for TCP options
 - Padding for No-operation in each option
 - Call for more efficient representation for order
 - A default encoding method is provided for Format
 - Other method is only the enhancement in the compressor
 - Separate control of state machine with format generator
 - Decision of Format selection should be take only in IR state
 - Simplify stack control mechanism to make sure the consistent of profile
 - A simple TCP/IP profile should be given first



Conclusion

- A refined state machine
- Further analysis about TCP behavior
 - Coarse-correlation
 - Shareable characteristic
- Interact with EPIC-LITE for more efficient supporting
- Some issues for MSN, context replication
- Plan:
 - Revise draft
 - Further discussion on the above issues and TCP/IP profile in mailing list

Requirements for SCTP compression

(Stream Control Transmission Protocol)

Christian Schmidt

53. IETF / RoHC in Minneapolis
19.03.2002

History of SCTP compression in RoHC

50. IETF in Mineapolis 03/2001:
Initial EPIC profile for SCTP compression: draft-price-rohc-epic-sctp-00.txt

51. IETF in London 08/2001:
No requirement specification for SCTP compression available.

52. IETF in Salt Lake City 12/2001:
Requirements Spezifikation draft-schmidt-rohc-sctp-requirements-00.txt

53. IETF in Minneapolis 03/2002:
Upgraded Requirement Spec draft-ietf-rohc-sctp-requirements-00.txt
Upgraded EPIC profile draft-west-sctp-epic-00.txt

Requirement for SCTP multi-streaming

Requirement: Keep SCTP multi-streaming quality of SCTP, that mean decompression errors affecting a stream should not influence other streams much.

Error case: SCTP Packet 2 lost and SCTP Packet 3 decompression failed
Different results for chunk with ts7: with / without compressed link.

Updated Requirement:

„Multi-streaming function of SCTP has to be kept in most of the cases.“

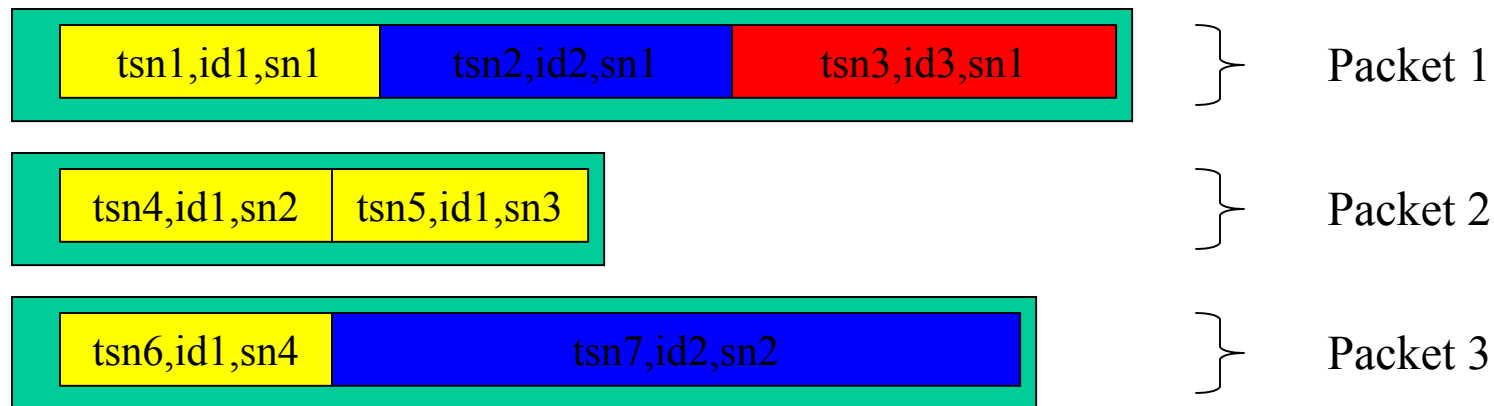
Requirement for SCTP multi-streaming

Requirement: Keep SCTP multi-streaming quality of SCTP, that mean decompression errors affecting a stream should not influence other streams much.

Error case: SCTP Packet 2 lost and SCTP Packet 3 decompression failed
Different results for chunk with tsn7: with / without compressed link.

Updated Requirement:

„Multi-streaming function of SCTP has to be kept in most of the cases.“



Further proceeding

- No open issue know today
- Final discussions for SCTP requirements on the mailing list
- WG last call for SCTP requirement specification.

Attempt at an SCTP Profile for EPIC

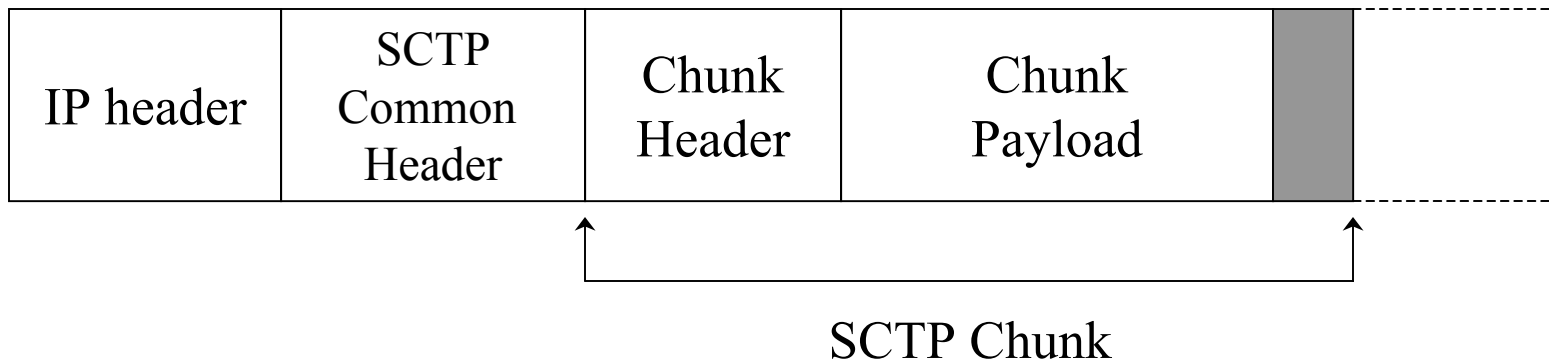
Mark West
(again)

Why have we done this?

- SCTP is a profile that we are interested in compressing
- It is also useful to see how EPIC can cope with this
- SCTP is quite a different protocol from RTP or TCP in many respects
- Give an early view as to what can be achieved with SCTP compression

What have we done?

- Assess the basic structure of an SCTP packet
- Handle the common header
- Handle the overall chunk structure
 - Note that this requires the addition of a new encoding method to handle the padding between chunks



SCTP Compression

- Slightly unusual for header compression
- The entire packet is processed
- Each chunk has a header, which can be compressed
- Chunk payload must be handled in order to get to the next chunk
 - Payload is typically not compressible
 - But perhaps in some cases...

What are the limitations

- It is a first attempt, so there are a number of things that are not captured
- Main issues are
 - Doesn't exploit the rules on combining of chunks in SCTP packets
 - Limitations on the number of times that chunks can occur in a packet
 - Doesn't exploit sharing of information between different chunk-types for the same flow

Conclusions

- We have a starting point for SCTP compression
 - Currently only from the perspective of an EPIC profile
 - No thought given to states and modes, for example
- Next step is to run the profile on some sample SCTP flows and assess the performance
- Also check requirements and ensure that profile addresses these points

ROHC RTP Implementation

- **ROHC RTP Implementer's guide**
 - *draft-ietf-rohc-rtp-impl-guide-00.txt*
 - **Optimized mode transitions corrected**
 - **Clarification on packet decoding during mode transfer**
 - **Clarification on aspects regarding initiation of mode transfer**
 - **Note on extension-3 in UO-1* packets: **Should go away?****

ROHC RTP Implementation

- **Third ROHC bake-off - “Arctic ROHC”**
 - **Where: Luleå, Sweden**
 - **When: April 17-23**
 - **Host: Ericsson**
 - **All implementations are welcome, especially “new-comers”**
 - **<http://standards.ericsson.net/rohc>**
 - **If you have an implementation and wants to participate, please let us know not later than April 3rd**



ROHC-MIB-RTP

<draft-ietf-rohc-mib-rtp-01.txt>

Juergen Quittek <quittek@ccrle.nec.de>

Hannes Hartenstein <hartenst@ccrle.nec.de>

Martin Stiernerling <stiernerling@ccrle.nec.de>

NEC Europe Ltd.

Overview

- MIB Structure: Object Groups
 - Instance, Channel
 - Compressor, Decompressor, Statistics
- Changes from -00 to -01
- Discussion Points
 - Architectural assumptions
 - Statistics
 - Openness (beyond RTP)
 - Conformance

MIB Structure: 5 Object Groups

- Instance group (`rohcInstanceGroup`)
 - merger of interface group and header group
- Channel group (`rohcChannelGroup`)
- Compressor group (`rohcCompressorGroup`)
- Decompressor group (`rohcDecompressorGroup`)
- Statistics group (`rohcStatisticsGroup`)

Instance Group

- Description of running instances of rohc at a managed network node
- Instance properties (manufacturer, version, ...)
- Instance parameters (clock resolution)
- Supported headers types
- IP interfaces served by the instance

Channel Group

- Table of all channels per IP interface
 - Properties of cannels:
 - Large CIDs
 - FeedbackFor
 - MRRU
 - Flow counter
 - ...
- Table of supported profiles per channel

Compressor Group

- Table of all compressor contexts per channel
 - CID, state, mode, profile,
 - compression ratio,
 - packet counters, (N)ACK counters
 - ...
- Table of allowed packet sizes per compressor
- Table of payload sizes per compressor

Decompressor Group

- Table of all decompressor contexts per channel
 - CID, state, mode, profile
 - depth of reverse compression
 - packet counters
 - (N)ACK counters
 - ...

Statistics Group

- Table of outgoing packet counters per header type and per compressor
- Table of incoming packet counters per header type and per decompressor
- Table of Error counters per error type and decompressor

Changes from -00 to -01

- Added Section 3 “Architectural Assumptions”
- Split interface table into instance table and interface table
- Merged interface group and header group into instance group
- Added CID state for compressors and decompressors
 - unused, active, expired, terminated

Architectural Assumptions

Which are reasonable and appropriate?

- Concurrent instances of ROHC
- Single instance per interface
- Multiple interfaces per instance
- Channels may be bi-directional
- Channel used by single instance only
- Feedback Channel at Same Interface

Statistics

- Are there suggestions for more concrete error types?
- Better having counters per repair strategy instead of per error type?
- Should there be more counters per channel?
 - ... and less per context?
 - contexts might be short lived
- Are there ideas for more / less / modified statistics?
 - **Packet counter per header type supported**
 - **Packet counter per profile not supported yet**

Openness (beyond RTP)

- How to be extensible concerning ROHC for TCP, SCTP, ... ?
 - independent MIB modules for each transport protocol?
 - basic module and individual extension modules?
 - open generic approach probably capable of integrating foreseeable future extensions?

Conformance

- Which of the groups should be
 - mandatory?
 - optional?

**Is anyone planning
to implement the MIB?**

IETF 53 – RObust Header Compression WG

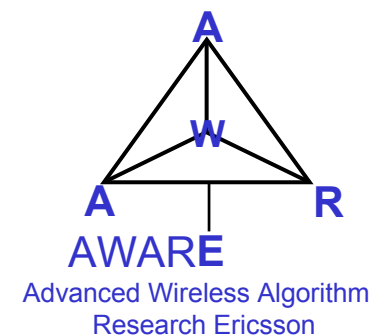
ROHC: Profiles for UDP Lite

Introducing
<draft-pelletier-rohc-udplite-00.txt>

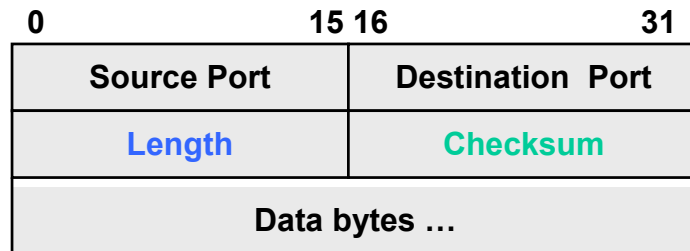


Ericsson Erisoft AB
Ghyslain.Pelletier@epl.ericsson.se
+46 920 20 24 32

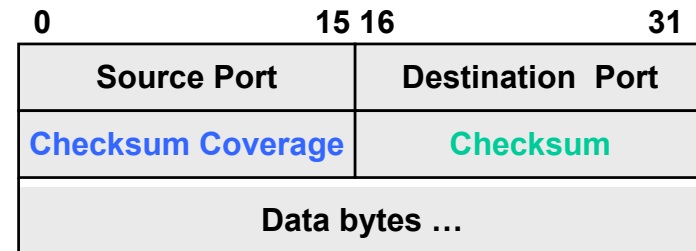
March 19st 2002, Minneapolis



Classic UDP vs UDP Lite



Classic UDP



UDP Lite

Classic UDP and UDP Lite

- Do not share protocol identifier (independent protocols)
- UDP Lite redefines the semantics for the Length and Checksum fields

UDP Lite redefines the semantics of the classic UDP Checksum and Length fields:

- Length field -> Checksum Coverage field (cannot always be inferred)
- The Checksum value depends on the Checksum Coverage field, and may exclude datagram payload
- The 16 bits checksum and checksum coverage are applied on a per-packet basis and minimally covers the UDP Lite header.

Overview of the draft

Different approaches to header compression are possible when the UDP Lite checksum is enabled. Specifically:

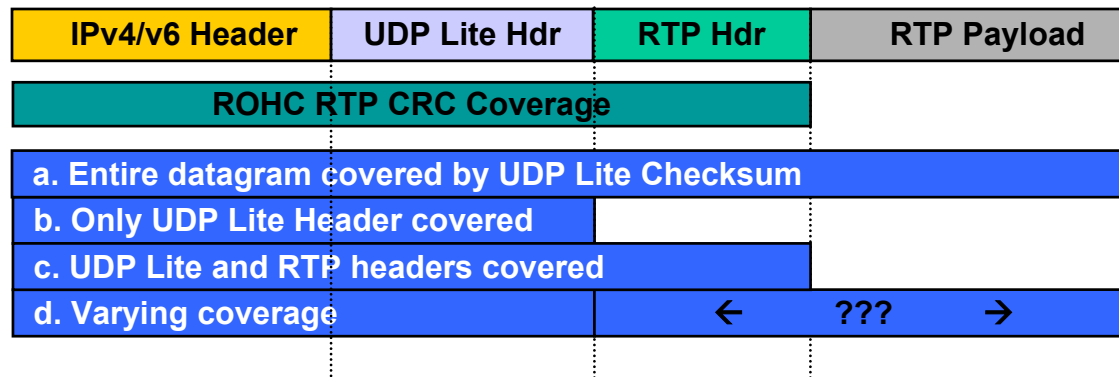
- 1) **Use RFC-3095 profiles for UDP almost as-is.** Checksum field is sent uncompressed (2 octets). Checksum Coverage field might be compressible or sent uncompressed (1 - 2 octets).
- 2) **Use the semantics and properties of UDP Lite Checksum Coverage and Checksum** to improve efficiency (save up to 3 octets in some cases?)

This draft suggests using the second approach.

The objectives of draft <draft-pelletier-rohc-udplite-00.txt>

- **Motivate the work for new profiles for UDP Lite**
- **Offers a possible direction**
 - Define UDP Lite profiles as modifications to the UDP profiles of RFC-3095
 - Suggest specific cases where it may be possible to replace the transport-layer checksum with a stronger mechanism, without violating the end-to-end nature of this checksum

Possible value assignments for the Checksum Coverage



The checksum coverage field may be:

- a.** Set to the UDP datagram length (or to 0)
- b.** Set to the UDP Lite header size
- c.** Set to the combined UDP Lite/RTP headers size
- d.** Set to an unpredictable value, varying between the UDP Lite (or UDP Lite/RTP) header size and the entire datagram length.

If these cases can be segregated from each other, additional compression gains may be possible for cases a-b-c.

Replacing the transport-layer checksum

Based on the semantics of the UDP Lite Checksum and the ROHC CRC functionality, can we achieve compression efficiency gains?

- **When only the UDP Lite header only is covered**

Basically, no information needs to be sent other than the checksum.

“It seems rather silly to protect the transmission of information that isn’t being sent” [RFC-1144]

- **When the UDP Lite/RTP headers only are covered**

Very few information bits are being sent as part of the header compressed flow. It may be acceptable to *replace* the transport-layer checksum with a CRC with the following properties:

- equal or stronger than the transport-layer checksum
- protects all bits covered by the transport-layer checksum
- offers equal or stronger robustness than header compression CRC
- protects the original transport-layer checksum

In those cases, it might be acceptable to recalculate the UDP Lite Checksum locally when decompressing the header and then validate using the transmitted CRC.

Discussion points

Questions to the group:

- Is replacing the UDP Lite checksum and the ROHC CRC with a stronger CRC for some well-defined cases acceptable?
- Shall we pursue the approach described in this draft?

Open questions:

- Which CRC polynomial would qualify? How many bits?
- How much of the UDP profiles need to be modified?
- ???