Overview of different encapsulation technologies

Compares main parameters of encapsulation technologies proposed for EFM copper

Vladimir Oksman
Broadcom Corporation
July 2002

Agenda

- This presentation describes two alternative encapsulation techniques - GFP and COBS.
  It also compares main parameters of these and the earlier proposed encapsulation techniques - HDLC and 64b/66b

- The goal of this presentation is to assist the selection of the appropriate encapsulation technique for EFM copper

COBS: brief introduction

- **COBS = Consistent Overhead Byte Stuffing**
  Introduced in 1999 [1] mostly as an alternative to HDLC

  - Build to operate in packet mode over a byte-synchronous channel

  - Another protocol using transparent bytes, but with significantly reduced statistical overhead

  - Can carry any byte-oriented payloads (Ethernet, IP, etc)

  - Simple, robust, and efficient

COBS: protocol

- **Structure:**

1. <u>Start-of-packet</u> byte (flag) = 0x00

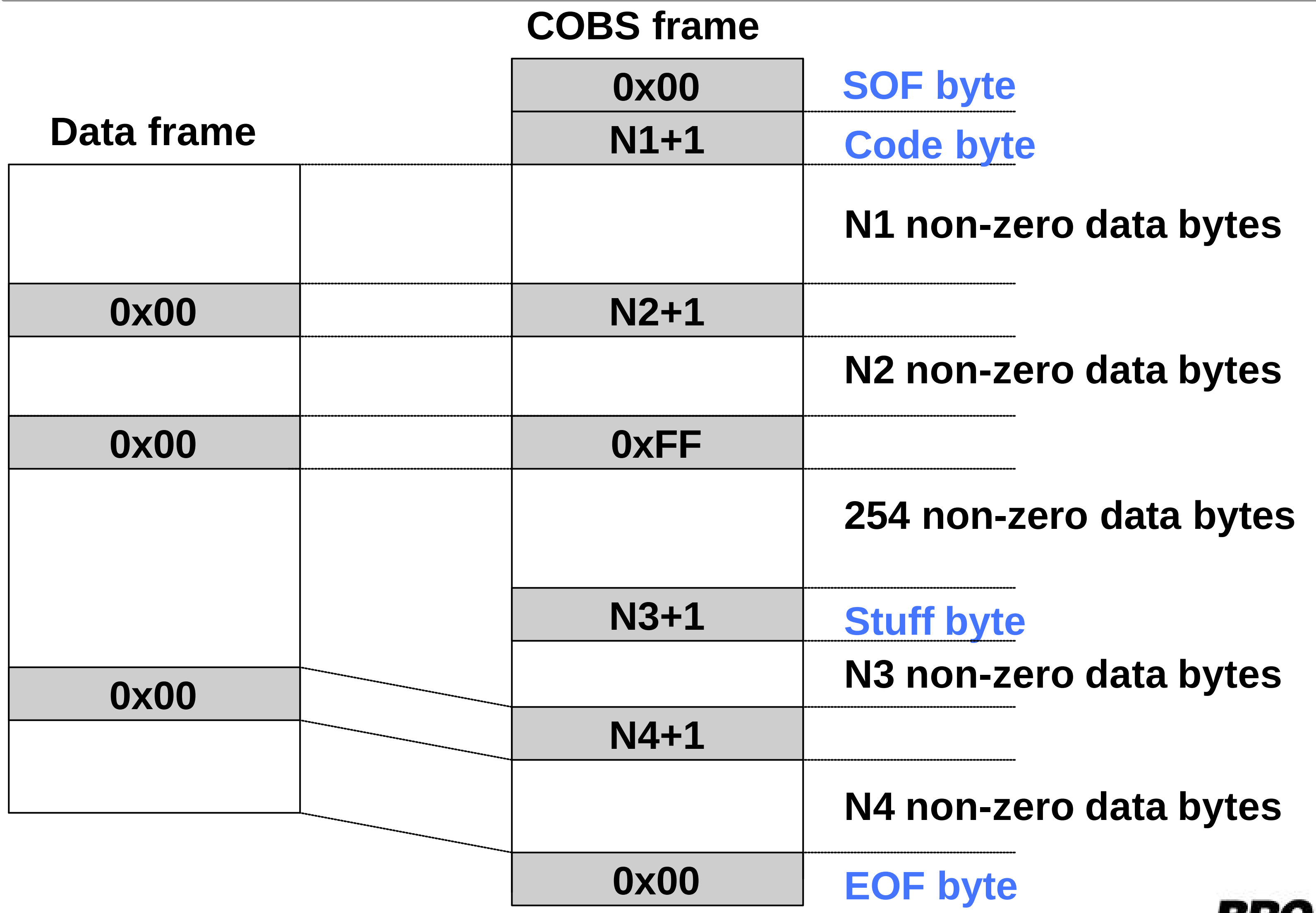
2. <u>End-of-packet</u> byte = 0x00, may be the flag of the next frame

3. <u>Code byte</u> - follows the flag and gets a value equal to the number of bytes to the first 0x00 data byte (if < 255) plus 1

4. All 0x00 in the transported data frame are changed to the values equal to the number of bytes to the next 0x00 data byte (if < 255) plus 1

5. If the number of bytes between the Code byte or any 0x00 byte and the following 0x00 byte is more then 254, the Code byte or relevant 0x00 byte is set to 0xFF and <u>Stuff byte</u> is introduced with a value equal to the number of bytes before the next 0x00 data byte (if < 255) plus 1

COBS: frame



COBS frame
Data frame
0x00
N1+1
N2+1
0xFF
N3+1
N4+1
0x00
0x00
0x00
0x00
SOF byte
Code byte
N1 non-zero data bytes
N2 non-zero data bytes
254 non-zero data bytes
Stuff byte
N3 non-zero data bytes
N4 non-zero data bytes
EOF byte

COBS: specifics

- Inter-frame gaps should be filled by Flags (0x00)
- Payload:
  - special header could be added to support loop bonding
  - CRC could be appended to provide the desired PUE and error monitoring
- Very low fixed overhead - only 2 overhead bytes per frame (SOP, Code byte)
- Very low statistical overhead - the maximum number of Stuff bytes is 1 per 255 data bytes
- Very simple synchronization - searching for Flag (0x00)

GFP: brief introduction

- **GFP = General Framing Procedure**
  Recently standardized by T1 and ITU as a generic adaptation protocol for multi-service broadband applications.

  - Build to operate both in packet mode (Frame-Mapped GFP) and continuous mode (Transparent-Mapped GFP) over a byte-synchronous channel

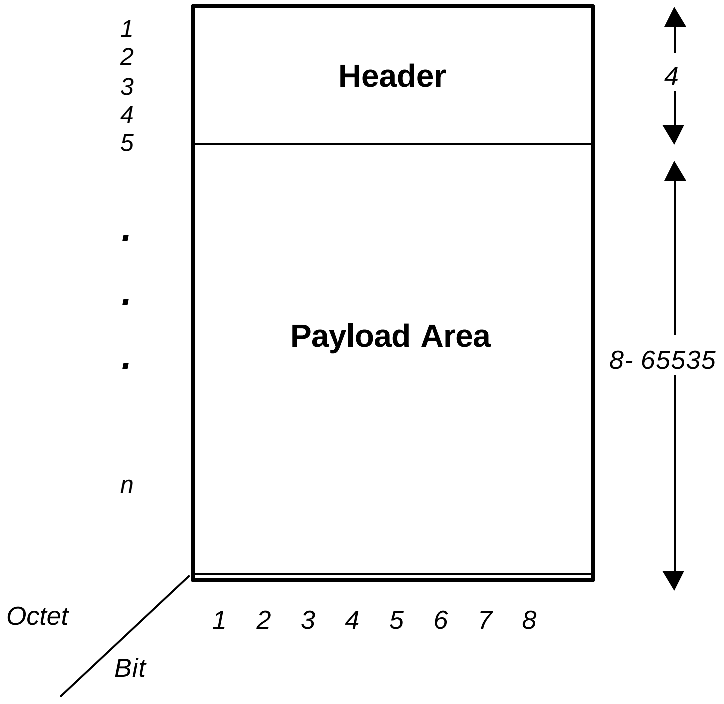
  - In packet mode intended to bear all variety of byte-oriented packet payloads (Ethernet, PPP, IP, MPLS, etc)

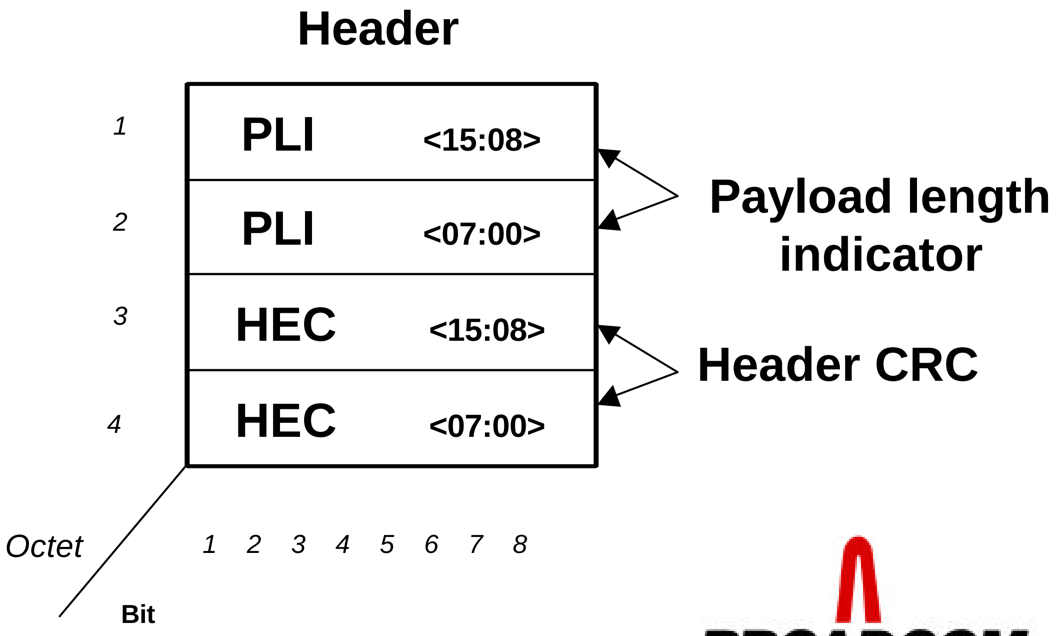
  - Simple, flexible, robust, and efficient

GFP: frame

- **GFP encapsulation includes:**

  - a 4-byte header for frame delineation
  - a up to 65535-byte payload (transmit data)
  - a 4-byte IDLE frame to fill up the inter-frame gaps





Frame-Mapped GFP: specifics

- Header
  - Provides single bit error correction and multi-error detection
  - IDLE header: PLI = 0x00
- Payload:
  - A 4-byte payload header to indicate the type of the payload, the carried protocol and other auxiliary information
  - A 4-byte optional FCS
- Low fixed overhead - only 4 overhead bytes per frame
- NO statistical overhead
- Synchronization - by searching for a valid header.
  Uses PLI to find the next frame boundary

GFP for EFM

- **Frame-Mapped GFP with possible simplifications:**

- **Header:**
  - Scrambling of the header may be not used since there is a separate scrambling in EFM PMD
  - Error correction in the header may not be performed (ineffective, as multiple errors are more probable)

- **Payload:**
  - special 3-4 byte optional header could be added instead a standard one to support loop bonding
  - 2-byte CRC could be appended instead a 4-byte FCS to provide the desired PUE and error monitoring

Main parameters

- The following parameters of the encapsulation technique were considered for comparison:
- Overhead (fixed and statistical)
- Synchronization (packet mode and continuous mode)
- Structure: byte-oriented or bit-oriented
- Probability of undetected error
- Complexity
- Field experience

Overhead computation

- **Probability of undetected error (PUE)**

  It was shown that standard Ethernet CRC doesn't provide the desired PUE. Additional means, such as 2-byte CRC or usage of FEC error indicator are necessary. Thus, <u>additional CRC is excluded from overhead computations</u> - it is assumed that it could be either used or not for any encapsulation technique

- **Additional headers**

  All additional headers intended for optional and/or auxiliary use (as Address/Control fields in HDLC, for instance) <u>are excluded from overhead computation</u>

HDLC (octet stuffing)

- **Overhead:**
  fixed: very low (1 bytes per frame, ~0.15% for an average frame)
  statistical: low in average (~ 3%), but 100% maximum value

- **Synchronization: very easy**
  Packet mode oriented - uses transparency mechanism. Frame is detected by searching for 1-byte flags

- **Structure: byte-oriented**

- **Complexity: very low**

- **Field experience: huge, widely used for different data networks**
  Current ITU-T standard for VDSL

HDLC - disadvantages

- A concern was raised on high maximum statistical overhead - up to 100%- although high values of the overhead appear with very low probability.

  Example: For a frame of average length (500 bytes) a probability of only 10%statistical overhead is 10^{-38}.

  This is much lower than even probability of an undetected error!

64b/66b

- **Overhead:**
  fixed: moderate (3.2% for long frames and 6.2% for short frames)
  statistical: low (< 10.6% for short frames and < 0.5% for long frames)

- **Synchronization: complex in packet mode**
  Uses Sync preambles distributed over the frame; build mostly for synchronized continuous mode. Not convenient for packet mode.

- **Structure: bit-oriented**

- **Complexity: moderate**

- **Field experience: IEEE standard for 10G**

Frame-Mapped GFP

- **Overhead:**
  fixed: low (4 bytes per frame)
  statistical: None

- **Synchronization: more complex**
  Build for packet mode, but frame alignment seems to be more complex than HDLC/COBS. Requires on-line search and processing of 2-byte code-words (similar to I.432)

- **Structure: byte-oriented**

- **Complexity: moderate (also requires length of the packet)**

- **Field experience: newly defined ANSI and ITU-T standard technology for packet transport in multi-protocol networks**

COBS

- **Overhead:**
  fixed: very low (2 bytes per frame)
  statistical: very low (< 0.4%)

- **Synchronization: very easy**
  Packet mode oriented - uses transparen bytes mechanism. Frame is detected by searching for 1-byte flags

- **Structure: byte-oriented**

- **Complexity: moderate (requires a 254-byte buffer)**

- **Field experience: limited usage**

Summary

Parameter	HDLC	64b/66b	GFP	COBS
Overhead	low, variable	moderate, almost fixed	low, fixed	low, almost fixed
Synchronization	very easy	complex	easy	very easy
Structure	byte-oriented	bit-oriented	byte-oriented	byte-oriented
Complexity	very low	moderate	moderate	moderate
Field experience	huge	10G	some	some
Standardized	ITU, IETF	IEEE.802	ITU, T1	---

Conclusions

- HDLC still seems to be a good candidate

- If HDLC can’t be adopted,

  GFP seems to be the more attractive than other considered technique due to:
  - proper synchronization in packet mode
  - simplicity of implementation
  - good international standard support

Possible GFP implementation for EFM

- **Header:**
  - A 4-byte standard header for frame delineation
  - Error correction - optional
  - Scrambling - optional

- **Payload (transported Ethernet frame):**
  - Preamble and SFD stripped
  - A special 3-4 byte optional header to support loop bonding etc.
  - Standard G.gfp scrambler to improve frame delineation
  - A 2-byte CRC appended to provide the desired PUE and error monitoring

- **Inter-frame gaps**
  - A standard IDLE GFP header (PLI=0x00)

Possible further simplification

- **Inter-frame gaps**
  - IDLE byte may be used instead IDLE header - reduces overhead and simplifies search for frame header if uses a value not used by PLI (0xFF, for instance, since for Ethernet PLI < 1538)