IEEE P1722 AVBTP Fragmentation/Reassembly

Version 0.01, 2007-08-03

Alan K. Bartky alan@bartky.net

Bartky Networks www.bartky.net

Send comments to AVBTP@listserv.ieee.org

Notice of copyright release

- **Notice:**
  - This document has been prepared to assist the work of the IEEE P1722 and IEEE 802 Working Groups. It is offered as a basis for discussion and is not binding on the contributing individual(s) or organization(s). The material in this document is subject to change in form and content after further study. The contributor(s) reserve(s) the right to add, amend or withdraw material contained herein.
- **Copyright Release to IEEE:**
  - The contributor grants a free, irrevocable license to the IEEE to incorporate material contained in this contribution, and any modifications thereof, in the creation of an IEEE Standards publication; to copyright in the IEEE's name any IEEE Standards publication even though it may include portions of this contribution; and at the IEEE's sole discretion to permit others to reproduce in whole or in part the resulting IEEE Standards publication. The contributor also acknowledges and accepts that this contribution may be made public by the IEEE P1722 Working Group or the IEEE 802 Working Group.

Revision History

Rev	Date	Comments
0.01	2007-08-03	First version proposing another alternate fragmentation/reassembly mechanism to handle, 61883, IIDC and also future protocols.

Design Goals

- Fragmentation/Reassembly mechanism should be:
  - Efficient
    - only use one quadlet
  - Consistent
    - Common mechanism to serve 61883, IIDC and future applications)
    - Define standard format for fragment length, packet length, sequencing (for both packet loss and possible out of order condition)
  - Simple
    - Implementable by HW or SW
    - Make algorithm simple such that lower bit rate streams that do not need fragmentation (less than about 90 Mbits/sec)
  - Scalable
    - Solve problem from 100 megabit to 10 gigabit
    - Proposal to do all fragmentation, reassembly and sequencing based on quadlets
  - Functional
    - Provide additional functionality of a standard mechanism for sequencing.
    - Provide fields of use for future new protocols to run over AVBTP using the same mechanism.

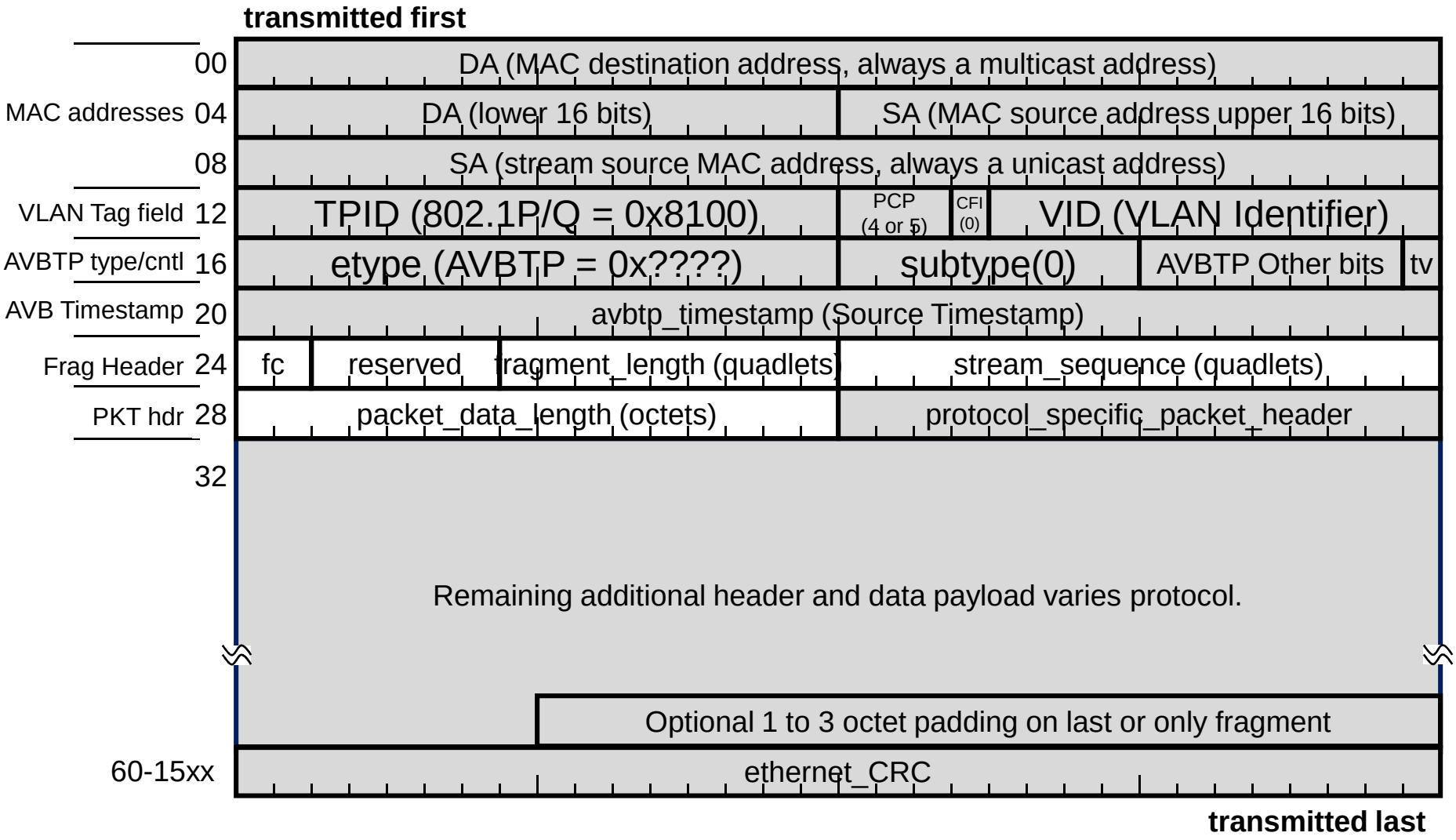
Proposal Summary

- Provide a fragmentation/reassembly service based on an integral number of quadlets
- Add one additional quadlet to AVBTP header for all stream data frames
- Standardize use of packet length field to use in conjunction with fragmentation and reassembly algorithms (e.g. used as sanity check for reassembly)
  - Use 1394 isochronous packet byte length field from previous proposals

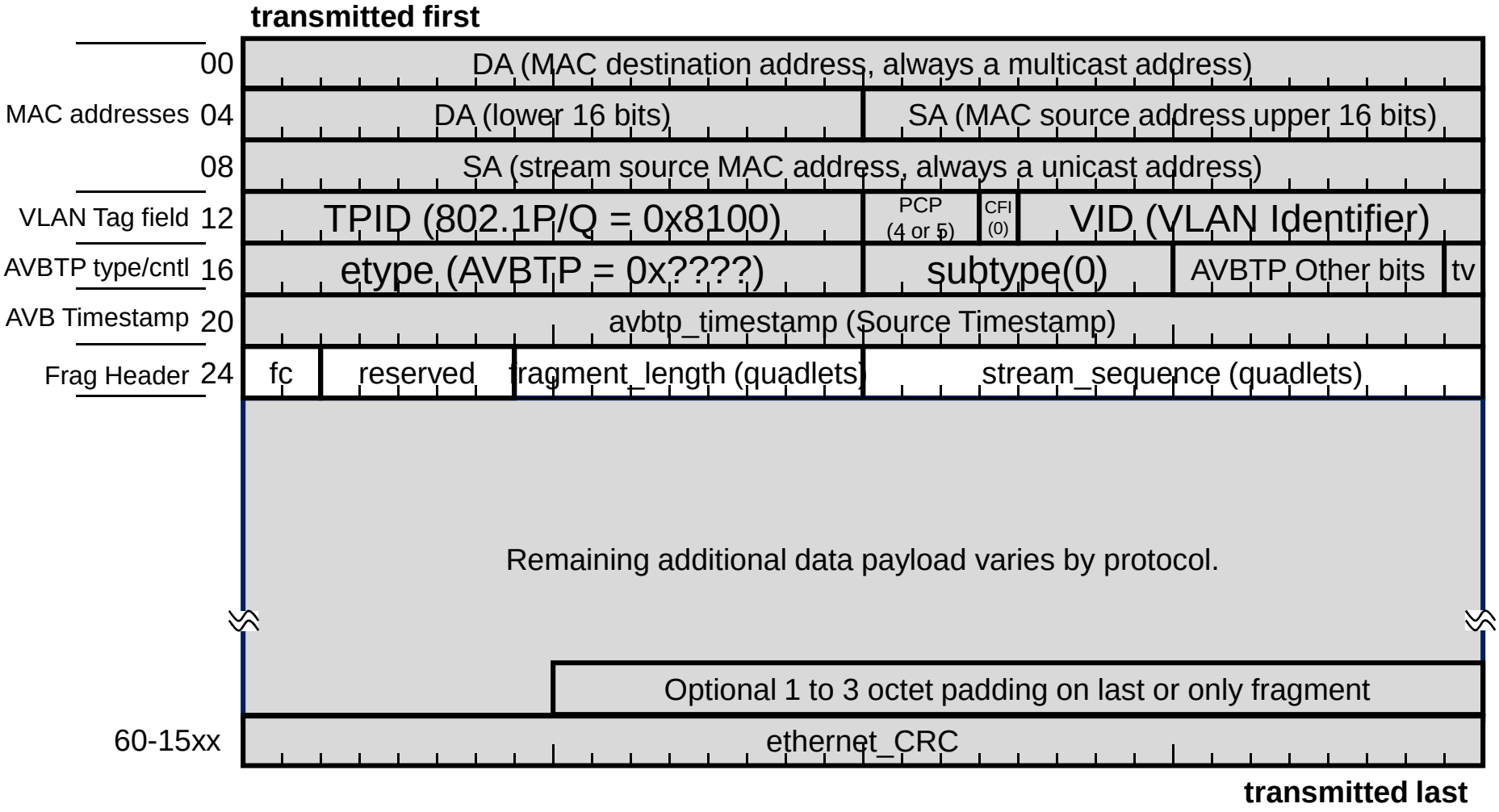
Proposal Summary

- Add the following new fields:
  - Fragment Length (quadlet count)
  - Fragment Control (First and Last bits)
  - Stream Sequence (quadlet count)
- Reuse the 1394 based packet length field, but standardize for all stream types.
- Mandate “smart fragmentation” for cases such as 61883 where packets are to be fragmented on source packet boundaries
  - Fragmentation rules passed to on a per stream basis
    - Per stream needed anyway to handle sequence number control
- Allow for delivery of received pipelined fragments and/or incomplete reassembled packets at AVBTP to upper layer.

First or only fragment stream packet format



Last or intermediate fragment stream packet format



Specific fields and rules

- fc: Fragmentation Control (2 bits)
  - NOTE: To make non-fragmented streams easier, a value of 0 is used to indicate first and last status
  - 00_2: Only Fragment (last = 0, first =0)
    - Used for all packets that are not fragmented
  - 10_2: First Fragment (last = 1, first = 0)
    - Indicates to receiver that this fragment contains a packet length and to start reassembly process
      - Store packet length for later sanity check
      - Store stream sequence for order and length checks of subsequent fragments
  - 11_2: Intermediate Fragment (last = 1, first = 1)
    - Indicates to receiver that this fragment is an intermediate fragment and to continue process
      - Check stream sequence if correct (Else packet lost or out of order fragment received)
      - Check if Packet MTU exceeded (Else packet too large, protocol error)
  - 01_2: Last Fragment (last = 0, first = 1)
    - Indicates to receiver that this fragment is an last packet fragment and to continue process
      - Check stream sequence if correct (Else packet lost or out of order fragment received)
      - Check if Packet MTU exceeded (Else packet too large, protocol error)
      - If all checks OK and not pipelined, OK to forward complete packet to upper layer and prepare hardware and or software for next stream packet.
      - If desired, the packet length can be checked as an additional sanity check.

Specific fields and rules

- fc: Fragmentation Control (2 bits)
  - NOTE: To make non-fragmented streams easier, a value of 0 is used to indicate first and last status
  - 00_2: Only Fragment (last = 0, first =0)
    - Used for all packets that are not fragmented
  - 10_2: First Fragment (last = 1, first = 0)
    - Indicates to receiver that this fragment contains a packet length and to start reassembly process
      - Store packet length for later sanity check
      - Store stream sequence for order and length checks of subsequent fragments
  - 11_2: Intermediate Fragment (last = 1, first = 1)
    - Indicates to receiver that this fragment is an intermediate fragment and to continue process
      - Check stream sequence if correct (Else packet lost or out of order fragment received)
      - Check if Packet MTU exceeded (Else packet too large, protocol error)
  - 01_2: Last Fragment (last = 0, first = 1)
    - Indicates to receiver that this fragment is an last packet fragment and to continue process
      - Check stream sequence if correct (Else packet lost or out of order fragment received)
      - Check if Packet MTU exceeded (Else packet too large, protocol error)
      - If all checks OK and not pipelined, OK to forward complete packet to upper layer and prepare hardware and or software for next stream packet.
      - If desired, the packet length can be checked as an additional sanity check.

General rules

- "If it fits, you must not split"
  - For packets that fit into a single Ethernet frame, the talker shall not break it up into fragments
    - Allows to handle streams up to ~90 Mbits/second without fragmentation/reassembly (keep cost down, possible simplifications, etc. for applications such as speakers, low res cameras, etc.)
- Fragmentation/Reassembly shall be based on quadlets, not bytes.
    - Forces alignment on 32 bit boundaries
    - Reduces bits needed in fragmentation/reassembly control field (one quadlet)
    - Should help it scale better to higher bandwidth streams for both hardware and software.
    - Standard packet octet based length field used if padding necessary.

Specific fields and rules

- stream_sequence(quadlets): 16 bits
  - Indicates current count of stream in quadlets.
  - Talker algorithm
    - Before any packets sent, transmit stream sequence counter is set to zero.
    - Each time a stream fragment (only, first, intermediate or last) is to be sent.
      - Fragment length (quadlets) is added to the stream sequence counter.
      - Fragment length and updated stream sequence counter value is put into the fragment.
      - Fragment is transmitted.
  - Listener algorithm
    - On reception of initial “first” or “only” fragment of a stream
      - Stream Sequence value from the fragment is copied to the receive sequence counter.
    - Each time a subsequent stream fragment (only, first, intermediate or last) is received
      - Fragment length (quadlets) from the frame is added to the receive stream sequence counter.
      - If receive stream sequence counter is not equal to the value in the received frame, then a fragment was lost, received out of order or there was a protocol error.
      - If frames are not pipelined, then receiver waits for next “first” or “only” packet and discards and resets the receive sequence counter when one of those frames is received.

Specific fields and rules

- packet_length (bytes): 16 bits
  - Same format as 1394, indicates number of octets including any subsequent headers (e.g. CIP) in this packet.
  - As fragmentation is done based on an integral number of quadlets, also used to ignore any optional padding octets at the end of the “last” or “only” fragment.
    - NOTE: 1394 based “portations” to AVBTP should already handle this case have this issue as they already handle this case as 1394 also provides a “quadlet based” service, and 61883-2 through -7 do not have any padding cases as currently defined.

Fragmentation control variables

- All variables are per stream
- packet_header_length:
  - 0 to n quadlets
  - Size in quadlets of packet header, starting at first quadlet past the packet length quadlet
- packet_payload_fragmentaton_unit_size:
  - 1 to n quadlets
  - Size in quadlets of “minimum sized chunks” past the packet_header_length quadlets to break the packet into fragments
- maximum_packet_size:
  - 1 to n quadlets
  - Maximum size for packets sent by talker or received by listener

Fragmentation control variables

- 61884-4 Example:
  - 61883-4 MPEG 192 byte source packets with a 2 quadlet CIP header per CIP packet
  - packet_header_length = 2 quadlets (8 bytes)
  - packet_payload_framentation_unit_size = 48 quadlets (192 bytes)
  - maximum_packet_size = as specified by the application, for this example, lets say max 10 MPEG packets per 125 microsecond cycle, so 2 + (48*10) = 482 quadlets
  - Fragmenter sends 1st fragment with CIP header and 7 MPEG packets and 2nd fragment with remaining 3 MPEG packets.

Fragmentation control variables

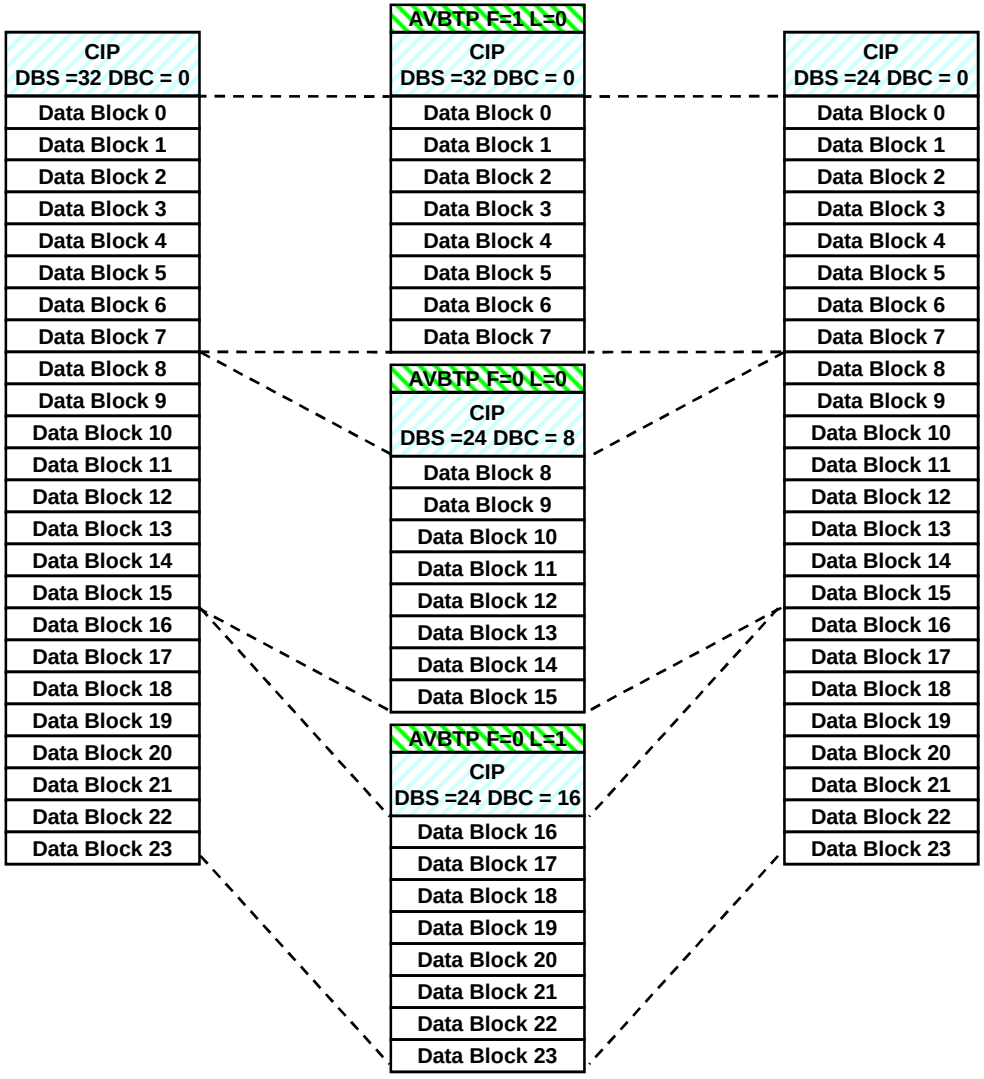
- IIDC Example:
  - IIDC with no CIP packet structure.
  - packet_header_length = 0 quadlets
  - packet_payload_framentation_unit_size = 1 quadlets (4 bytes)
  - maximum_packet_size = as specified by the application, for this example, lets say max of 2000 bytes (500 quadlets)
  - Fragmenter fills 1st fragment with 1488 bytes (372 quadlets) and 2nd fragment with 512 bytes (128 quadlets)

Backup

Previous proposal on fragmentation based on splitting up and creating multiple CIP packets.

Example Data Block Fragmentation

- Example fragmentation based on Data Blocks
- Use case, 61883-6, 192 kHz audio, 32 channels.
  - (24 samples per 8 kHz cycle) * (4 bytes per sample) * 32 channels = 3072 total bytes contained in 24 Data Blocks each 32 quadlets long.
- Need to break into 3 Ethernet Frames
  - First Frame: F bit = 1, L bit = 0, DBC = 0, SYT field copied from source.
  - Middle Frame: F bit =0, L bit = 0, DBC = 8, SYT field set to all ones.
  - Last Frame: F bit = 0, L bit = 1, DBC = 16, SYT field set to all ones
- Reassembly if needed is straight forward:
  - DBS, DBC, SYT copied from packet with F bit = 1,
  - Length calculated by adding lengths of all fragments minus CIP headers of intermediate or last packets.
  - Lost fragments detected by sequence number mismatch (just like for normal streams)



Example Source Packet Fragmentation

- Example fragmentation based on Source Packets
- Use case, 300 megabits/second 422P@HL MPEG-2 (1920x1080 @ 30Hz)
  - ~ 25 MPEG packets per 8 kHz cycle
  - 188 byte packets with 1 quadlet header(192 bytes per Source packet), 8 x 6 quadlet Data Blocks per source packer as per 61883-4 specification.
  - SYT field in source packet headers, not in CIP header.
- Need to break into 4 Ethernet Frames
  - First Frame: First Packet (F) bit = 1, Last Packet (L) bit = 0, DBC = 0
  - 2 Middle Frames: F bit =0, L bit = 0, DBC = 42 & 84
  - Last Frame: F bit = 0, L bit = 1, DBC = 126
- Reassembly if needed is straight forward:
  - DBS & DBC copied from packet with F bit = 1,
  - Length calculated by adding lengths of all fragments minus CIP headers of intermediate or last packets.
  - Lost fragments detected by DBC sequence number mismatch (just like for normal streams)