Data Specification for AER over Ethernet "AEtheRnet"

Proposal version 0.3, 9 December 2014

Alex Rast, David Lester

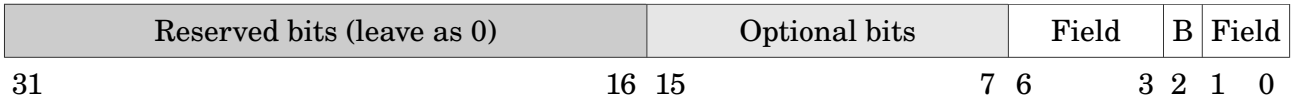
1 Introduction

This specification addresses the data format for spikes to be sent over Ethernet between compatible AER-generating and receiving devices. It is expected that the interface can be bidirectional: a device can both issue and receive spikes. However, it is not *required* that a given device must be able to do both; a device could be either a blind issuer of spikes or a passive receiver.

2 Definitions

Throughout this document several terms will be used repeatedly as defined below.
AER: Address-event representation: an event coded as a binary number indicating the event source.
AEtheRnet: The protocol this document describes.
Key: A valid address in the AER address space.
Packet: The basic bundling unit of the AEtheRnet protocol, including any transport-related framing.
Block: An amount of data, bundled into an AEtheRnet packet, not including any headers, all in the same format as specified in the AEtheRnet header.
Device: A hardware device or software process, treated as a black box, that can send and/or receive AEtheRnet packets.
Stream: An internal channel on a device, responding to AER packets in some particular expected way, capable of receiving and/or sending AER packets in their native internal format. A device could in principle support several different streams with possibly different interfaces.

Machine word descriptions: The text indicates the internal format of machine words using a graphical format as below. The highest bit-position indicates the expected word size. Optional bits are assumed to be detected using a flag which if not set causes the bits to be ignored.



3 Supported Content Formats

The protocol supports 3 different types of content:
A. Packed AER addresses; these may be either 16- or 32-bit.
B. Packed AER addresses with payloads; either 16+16 or 32+32-bit.
C. Device-specific internal commands with internal format determined by the device.

4 General Comments on Data Handling

Transport: This specification does not specify the transport. In principle, any Ethernet-

compatible transport could be used. Designers of interfaces should consider the facilities provided by the transport when developing AEtheRnet interfaces. Note that some transports will normally result in data words possibly misaligned according to machine word boundaries. The specification provides a mechanism to address this.

Packet Type Handling: It is not required that all devices be able to service all packet types. For example, a device that had only 16 bit internal registers would not be required to service 32-bit packets. However, the interface must be designed such that receipt of a packet of unsupported type not cause the device to go into an invalid state; i.e. if it receives an unsupported type it should discard the packet and continue to service supported packets. Implementers must specify clearly in interface documentation which packet types are supported.

Endianness: The assumption is that in normal use, packets will be transported over the wire in big-endian format, but represented on the device in little-endian format. All machine words are described as in the device representation, i.e. little-endian.

Errors: AEtheRnet does not attempt to detect or correct errors. If an error does occur, the behaviour is UNDEFINED. Designers of interfaces may use transport facilities (e.g. CRC) to recover from errors if desired, but this is not required. In particular there is no provision for what processing occurs if an invalid AEtheRnet header is received.

Acknowledgements: No devices acknowledge receipt of AEtheRnet packets; similar to hardware AER, the protocol is fire-and-forget. No command sent using a command packet may expect a reply packet of any type.

Command Processing: The interpretation of data in a command is entirely device-specific. AEtheRnet does not impose any restrictions on the format of data in this field, including endianness and error handling, so the comments related to errors and endianness above may not apply in the case of a command.

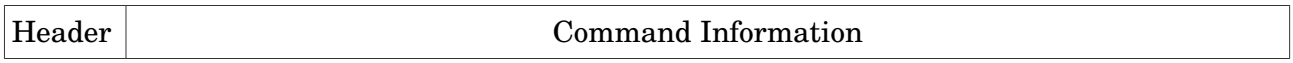
Latency: For real-time traffic, latency before a packet is sent will be a concern. A device which buffers AER spikes before forwarding them in either direction must ensure that no spike is forwarded later than the time when it should have been received by the receiving target. Therefore devices must send partially-filled AEtheRnet packets as soon as the earliest latency period of any of its members is exceeded. Once a packet is full (contains 256 items or is as large as the device's internal buffer size), it must be sent immediately. Communicating devices can use Command packets, with a suitable protocol, if necessary, to inform each other of their latency requirements and buffering limits.

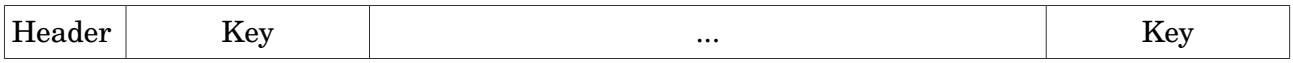
Packet Ordering: Packets must be issued in sequential time order. If packet payloads contain timestamps then the value of the timestamp must be strictly increasing with packet sequence number. Receiving devices must discard any spikes whose timestamps are received out of strict sequential order.

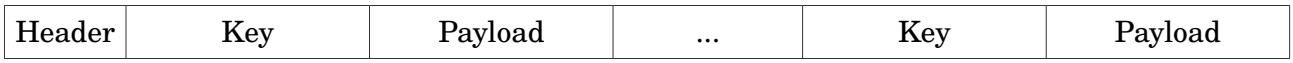
5 Packet Format

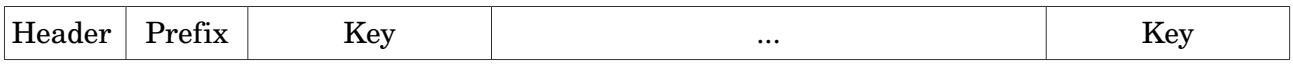
AEtheRnet packets consist of a header and data, packed into possibly variable-length blocks of up to 2K bytes. Each packet contains a single data type (with the exception of command packets). There are 5 different possible headers. The most fundamental is the Basic data header, a 16-bit halfword followed directly by the data. The Prefixed data header consists of 2

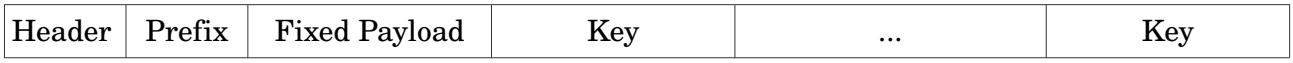
16-bit halfwords, the first with the same format as the Basic header, the second being a prefix to OR with all AER keys in the block (position-dependent) to assemble the compete key, and the Command header is a 16-bit halfword followed immediately by device-specific command information. Data (as opposed to command) AEtheRnet packets may also have a fixed payload base, to be ORed with sent payloads. If the packet does not send payloads then the fixed payload base is understood to be a constant payload with each AER key. The underlying word size of the packet determines the size of this payload base. A complete AEtheRnet packet therefore can have one of the following 9 structures:











All headers have as their 2 MSBs a pair of bits "P" (for "Prefixed") and "F" (for "Format") that identify the header type. Prefixed headers have either 10 or 11 depending on whether the prefix is to be ORed with the lower or upper halfword of the key respectively. If the protocol is sending 16-bit keys then a 1 in the F bit for Prefixed headers will extend the internal format of the keys to 32-bit. Likewise if the protocol is sending 32-bit keys a 0 in the F bit for Prefixed headers will truncate the key to 16-bit. Command headers always have 10 for their MSBs and Basic headers have 00 in their MSBs.

The rest of the Command header is the device-specific command code. For Basic and Prefixed headers, a flag"D" (for "Data") allow a fixed pattern to be ORed with payloads. If the packet has no payload the fixed pattern is used as a fixed payload. The data type of the packet (16 or 32 bit) sets the size of the expected field, which should always be the last header item before the start of keys. A second flag, "T" allows payloads to be interpreted as timestamps. If this bit is set, the receiver should interpret each spike as being sent at the time indicated by its payload. (If D and T are both set with no payload in the packet type, the system can efficiently send a block of spikes at the same time). The next 2 bits indicate the datatype in the packet, followed by 2 Tag bits that identify the stream, and finally 8 Count bits that give the number of data items, where an item is either a key or a key/payload pair.

The decode of the datatype flags is as follows:
P - Bit 15
  0 = No key prefix
  1 = With key prefix
F - Bit 14
  0 = Basic packet; OR prefix with lower halfword
  1 = Command packet; OR prefix with upper halfword
D - Bit 13
  0 = No payload prefix
  1 = With payload prefix
T - Bit 12
  0 = Payloads are not timestamps
  1 = Payloads are timestamps
Type - Bits 11:10
  00 = 16-bit key
  01 = 16-bit keys and 16-bit payloads, alternating.
  10 = 32-bit key
  11 = 32-bit keys and 32-bit payloads, alternating.

Thus the header formats have the following structure:

Formats of a data header:

P	F	D	T	Type	Tag	Count	Key Prefix (if P) or Data (not P)
15	14	13	12	11 10	9 8	7 0	15 0

0	0	1	T	Type	Tag	Count	Payload Prefix
15	14	13	12	11 10	9 8	7 0	15 0

1	F	1	T	Type	Tag	Count	Key Prefix	Payload Prefix
15	14	13	12	11 10	9 8	7 0	15 0	31/15 0

Format of a command header:

0	1	Command	(Device-specific)
15	14	13 0	...