
CACE Technologies

Per-Packet Information Header Specification

Version 1.0

Revision History

Date	Version	Description	Author
04/16/2007	0.6	Updated 802.11 common and 802.11n header, split now into 802.11n basic/MAC and 802.11n extended/MAC+PHY	Gianluca Varenni
04/18/2007	0.7	Renamed from Airhead to PPI. Minor fixes and updates	Gerald Combs
05/09/2007	0.8	Fixed the invalid values for MCS, dBm values and RSSI.	Gianluca Varenni
05/18/2007	0.9	Added abbreviations and acronyms section. Added member element to the Channel-Flags element in both 802.11-Common and 802.11n MAC+PHY fields. General cleanup. Filled in purpose section to clarify PPI's intent.	Dustin Johnson
6/11/2007	0.95	Update the "Purpose" section. Explicitly refer to NTAR	Gerald Combs
6/12/2007	1.0	Bump to 1.0.	Gerald Combs
6/15/2007	1.0.1	Fix remaining "Airhead" references	Gerald Combs

Table of Contents

1.	Introduction	4
1.1	Purpose	4
1.2	Scope	4
1.3	Definitions, Acronyms and Abbreviations	4
1.4	Notational Conventions	5
1.5	References	5
1.6	Overview	5
2.	Overall Description	5
3.	PPI Header Format	5
3.1	PPI Packet Header Structure	5
3.1.1	pph_version	6
3.1.2	pph_flags	6
3.1.3	pph_len	6
3.1.4	pph_dlt	6
3.2	PPI Field Structure	6
3.2.1	pfh_type	6
3.2.2	pfh_data_len	6
3.3	Field Processing	7
4.	General-Purpose Field Types	7
4.1	Field Descriptions	7
4.1.2	802.11-Common	7
4.1.3	802.11n MAC Extension (basic)	8
4.1.4	802.11n MAC+PHY Extension (Extended)	9
4.1.5	Spectrum-Map	11
4.1.6	Process-Info	12
4.1.7	Capture-Info	12
4.1.8	Host-Name-Info	12
4.1.9	Signature	12
4.1.10	Privacy	12
5.	Vendor-Specific Field Types	12

1. Introduction

When capturing live network data, it is often useful to collect out-of-band information and provide it along with in-band packet data. The traditional method of doing this is to prefix each PDU with a meta-information header (often called a pseudoheader). Current implementations include such information as 802.11 radio information, access server user IDs, and point-to-point link direction.

The Per-Packet Information (PPI) Header is a general and extensible meta-information header format originally developed to provide 802.11n radio information, but can handle other information as well.

1.1 Purpose

PPI is intended to supplement individual packets received from a hardware or software engine with interesting data in a light-weight fashion.

Often such data of interest pertains to the PHY layer, but this need not always be the case. Whatever the carried information may be, the intent is that the information contained is only added to data captured in real-time, not stored packets.

It is not intended to add arbitrary data to packets, such as annotations. That task is better suited to NTAR.

1.2 Scope

This document defines the general format of the PPI header, along with the formats of several fields. Performance and security are outside the scope of this document.

1.3 Definitions, Acronyms and Abbreviations

ASCII	American Standard Code for Information Interchange
A-MPDU	aggregate MAC protocol data unit
A-MSDU	aggregate MAC service data unit
CCK	complementary code keying
CRC	cyclic redundancy check
CR-LF	carriage return-line feed
DLT	data link type
EVM	error vector magnitude
FCS	frame check sequence
FHSS	frequency-hopping spread spectrum
GFSK	gaussian frequency shift key or keying
GID	group identifier
HT	high throughput
MAC	medium access control
MPDU	MAC protocol data unit
OFDM	orthogonal frequency division multiplexing
PHY	physical layer
PPI	per-packet information
RF	radio frequency

RSSI	receive signal strength indicator
RX	receive or receiver
SIG	short guard interval
TSF	timing synchronization function
TSFT	timing synchronization function timer
UID	unique identifier
UTF	unicode transformation format

1.4 Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

1.5 References

Radiotap manual page: http://netbsd.gw.com/cgi-bin/man-cgi?ieee80211_radiotap+9+NetBSD-current

NTAR documentation: <http://www.winpcap.org/ntar/>

RFC 2119: <http://www.ietf.org/rfc/rfc2119.txt>

1.6 Overview

Section 2 provides a description of the PPI header, along with an explanation of its necessity. Section 3 defines the structure of the header, complete with C and C++-compatible data structures. Section 4 defines each data type.

2. Overall Description

Existing header formats are typically made up of static data structures filled in by the capture mechanism and passed to user space. They suffer from the following problems:

- Limited scope. They are restricted to specific elements within a single domain.
- Rigidity. It is either impossible or very difficult to add new elements.
- Fixed DLTs. Each format only supports one encapsulated data link type.

PPI attempts to address each of these issues in a clean, consistent manner. Data elements are formatted as type-length-value (TLV) triplets, which allows for future expansion of the header while providing backward compatibility. Per-packet DLTs can be implemented by using an "empty" PPI header.

3. PPI Header Format

Each PPI packet header is made up of a packet header followed by zero or more fields. Each field is a type-length-value triplet.

Packet Header	Field Header	Field Data	Field Header	Field Data
---------------	--------------	------------	--------------	------------

Multi-byte integers in the packet header and field headers MUST be stored as little-endian. The endianness of field data may be either big- or little-endian, and MUST be noted in the field description. The total length of the packet header plus all field headers and field data MUST be padded to a 32-bit boundary.

3.1 PPI Packet Header Structure

The PPI packet header provides a version, indicator flags, and the header length:

```
typedef struct ppi_packetheader {
```

```

    u_int8_t pph_version;    /* Version.  Currently 0 */
    u_int8_t pph_flags;      /* Flags.          */
    u_int16_t pph_len;       /* Length of entire message,
                           * including this header and TLV
                           * payload. */
    u_int32_t pph_dlt        /* Data Link Type of the captured
                           * packet data. */
} ppi_packetheader_t;

```

3.1.1 *pph_version*

The version of the PPI header. MUST be set to zero (0).

3.1.2 *pph_flags*

An 8-bit mask that defines the behavior of the header. The following values are defined:

Bits (Bit 1 = LSB)	Values
1	Alignment. 32-bit aligned = 1, non-aligned = 0 Explained further in section 3.3
2-8	Reserved. MUST be 0.

3.1.3 *pph_len*

The length of the entire PPI header, including the packet header and fields. It MUST be between 4 and 65,532 inclusive.

3.1.4 *pph_dlt*

This MUST contain a valid data link type as defined in `pcap-bpf.h` from the libpcap distribution. If an official DLT registry is ever created by the libpcap development team, then it will supersede this list.

A capture facility can implement per-packet DLTs by setting `pph_version` to 0, `pph_flags` to 0, `pph_len` to 8, and `pph_dlt` to the DLT of the encapsulated packet.

3.2 PPI Field Structure

Each PPI field includes a type and length:

```

typedef struct ppi_fieldheader {
    u_int16_t pfh_type;    /* Type */
    u_int16_t pfh_datalen; /* Length of data */
} ppi_fieldheader_t;

```

3.2.1 *pfh_type*

The type of data following the field header MUST be a valid type value as defined below:

Range	Possible Values
0-29,999	General-purpose field. Defined in section 4.
30,000-65,535	Vendor-specific fields. Defined in section 5.

If an unknown field value is encountered, it MUST be skipped according to the length rule in section 3.3. Implementations MAY mark it as “unknown” as appropriate.

3.2.2 *pfh_datalen*

The length of the data, in bytes, that follows MUST be between 0 and 65,520 inclusive. The end of the data MUST NOT exceed the total header length.

3.3 Field Processing

The first field header immediately follows the packet header (that is, if the packet header starts at byte 0, the first field header starts at byte 8). The starting point of each subsequent field header is defined by the “padding” bit in pph_flags:

- If the “padding” bit in pph_flags is set to 1 AND pph_dataalen in field n is not a multiple of 4, then field $n+1$ will start at the next multiple of 4. For example, if pph_dataalen in field 3 is 9, then the next three bytes MUST be considered padding, and field 4 will begin at byte 12.
- If the “padding” bit in pph_flags is 0, then field $n+1$ will start at the next byte offset following the data in field n .

All padding bytes MUST be set to 0 in order to keep from exposing kernel memory to user space.

4. General-Purpose Field Types

The following general-purpose fields are currently defined. Further general-purpose fields will be defined in later revisions of this document. Vendor-specific fields may be defined externally.

Type	Length (Bytes)	Description
0-1		RESERVED
2	20	802.11-Common. Common (pre-n and .11n) radio information.
3	12	802.11n MAC Extensions. Extended (.11n) radio information.
4	48	802.11n MAC+PHY Extensions. Extended (.11n) radio information.
5	22-65,520	Spectrum-Map. Radio frequency spectrum information.
6	19-65,520	Process-Info. Process information, e.g. UID and GID
7	???	Capture-Info. Capture information, e.g. interface, drop counts, etc.
8 – 29,999	-	RESERVED

4.1 Field Descriptions

4.1.2 802.11-Common

Zero or one 802.11-Common fields may be present in a single header. All fields are little-endian.

The 802.11-Common field is loosely based on the existing Radiotap header format. It contains data common to both pre-n and 802.11n. Total length is 20 bytes.

Field Name	Semantics	Value Type	Length
TSF-Timer	7.3.1.10 and 11.1 of IEEE 802.11-1999 Invalid value = 0	Unsigned integer	8 bytes
Flags	Packet flags LSB = bit 0. Bits: Bit 0 = If set, FCS present Bit 1 = If set to 1, the TSF-timer is in ms, if set to 0 the TSF-timer is in us Bit 2 = If set, the FCS is not valid Bit 3 = If set, there was a PHY error receiving the packet. If this bit is set, Bit 2 is not relevant	Unsigned integer	2 bytes

Rate	Data rate in multiples of 500 Kbps Invalid value = 0x0000	Unsigned integer	2 bytes
Channel-Freq	Radiotap-formatted channel frequency, in MHz Invalid value = 0x0000	Unsigned integer	2 bytes
Channel-Flags	Radiotap-formatted channel flags: Bit 0-3 = Reserved Bit 4 = Turbo Bit 5 = Complementary Code Keying (CCK) Bit 6 = Orthogonal Frequency-Division Multiplexing (OFDM) Bit 7 = 2 GHz spectrum Bit 8 = 5 GHz spectrum Bit 9 = Only passive scan allowed Bit 10 = Dynamic CCK-OFDM Bit 11 = Gaussian Frequency Shift Keying (GFSK) (FHSS PHY) Bit 12-15 = Reserved	Unsigned integer	2 bytes
FHSS-Hopset	Radiotap-formatted Frequency-hopping spread spectrum (FHSS) hopset	Unsigned integer	1 byte
FHSS-Pattern	Radiotap-formatted Frequency-hopping spread spectrum (FHSS) pattern	Unsigned integer	1 byte
dBm-Antsignal	RF signal power at antenna Invalid value = -128	Signed integer	1 byte
dBm-Antnoise	RF noise at antenna Invalid value = -128	Signed integer	1 byte

Unlike Radiotap, these fields are packed without any padding or alignment.

4.1.3 802.11n MAC Extension (basic)

Zero or one 802.11-Common fields may be present in a single header. Correct parsing the 802.11n-MAC Extension field depends on values from the 802.11-Common field. If present, it **MUST** be immediately preceded by an 802.11-Common field. All 802.11n MAC Extension fields are little-endian.

The 802.11n MAC Extension field contains radio information specific to 802.11n. Total length is 27 bytes.

Field Name	Semantics	Value Type	Length
Flags	LSB = bit 0. Bits: Bit 0 = Greenfield Bit 1 = HT20 (0) or HT40 (1) indicator Bit 2 = RX short guard interval (SGI) Bit 3 = Duplicate RX Bit 4 = Aggregate	Unsigned integer	4 bytes

	Bit 5 = More aggregates Bit 6 = Aggregate delimiter CRC error after this frame		
A-MPDU-ID	Unique A-MPDU ID used for A-MPDU reassembly	Unsigned integer	4 bytes
Num-Delimiters	Number of zero-length pad delimiters	Unsigned integer	1 byte
Reserved		Unsigned integer	3 bytes

If the aggregate flag (bit 4 of the Flags field) is set, then each MPDU in a particular A-MPDU MUST have the same A-MPDU-ID. The A-MPDU-ID SHOULD be randomly assigned in order to prevent improper reassembly if capture files are merged.

4.1.4 802.11n MAC+PHY Extension (Extended)

Zero or one 802.11-Common fields may be present in a single header. Correct parsing the 802.11n-MAC+PHY Extension field depends on values from the 802.11-Common field. If present, it MUST be immediately preceded by an 802.11-Common field. All 802.11n MAC+PHY Extension fields are little-endian.

The 802.11n MAC+PHY Extension field contains radio information specific to 802.11n. Total length is 48 bytes.

Field Name	Semantics	Value Type	Length
Flags	LSB = bit 0. Bits: Bit 0 = Greenfield Bit 1 = HT20 (0) or HT40 (1) indicator Bit 2 = RX guard interval Bit 3 = Duplicate RX Bit 4 = Aggregate Bit 5 = More aggregates Bit 6 = Aggregate delimiter CRC error after this frame	Unsigned integer	4 bytes
A-MPDU-ID	Unique A-MPDU ID used for A-MPDU reassembly	Unsigned integer	4 bytes
Num-Delimiters	Number of zero-length pad delimiters	Unsigned integer	1 byte
MCS	Modulation Coding Scheme Invalid Value = 255	Unsigned integer	1 byte
Num-Streams	Number of spatial streams. 0 (zero) means that the information is not available	Unsigned integer	1 byte
RSSI-Combined	Combined Received Signal Strength Indication (RSSI) value from all the active antennas and channels. Invalid value = 255	Unsigned integer	1 byte
RSSIAnt0Ctl	Received Signal Strength Indication (RSSI) value for the antenna 0, control channel Invalid value = 255	Unsigned integer	1 byte

RSSIAnt1Ctl	Received Signal Strength Indication (RSSI) value for the antenna 1, control channel Invalid value = 255	Unsigned integer	1 byte
RSSIAnt2Ctl	Received Signal Strength Indication (RSSI) value for the antenna 2, control channel Invalid value = 255	Unsigned integer	1 byte
RSSIAnt3Ctl	Received Signal Strength Indication (RSSI) value for the antenna 3, control channel Invalid value = 255	Unsigned integer	1 byte
RSSIAnt0Ext	Received Signal Strength Indication (RSSI) value for the antenna 0, extension channel Invalid value = 255	Unsigned integer	1 byte
RSSIAnt1Ext	Received Signal Strength Indication (RSSI) value for the antenna 0, extension channel Invalid value = 255	Unsigned integer	1 byte
RSSIAnt2Ext	Received Signal Strength Indication (RSSI) value for the antenna 0, extension channel Invalid value = 255	Unsigned integer	1 byte
RSSIAnt3Ext	Received Signal Strength Indication (RSSI) value for the antenna 0, extension channel Invalid value = 255	Unsigned integer	1 byte
Extension Channel-Freq	Radiotap-formatted extension channel frequency, in Channel Frequency in MHz Invalid value = 0x0000. The frequency of the control channel is stored in the Channel-Freq field of the 802.11 Common header.	Unsigned integer	2 bytes
Extension Channel-Flags	Radiotap-formatted extension channel flags. The flags of the control channel are stored in the Channel-Flags field of the 802.11 Common header. Bit 0-3 = Reserved Bit 4 = Turbo Bit 5 = Complementary Code Keying (CCK) Bit 6 = Orthogonal Frequency-Division Multiplexing (OFDM) Bit 7 = 2 GHz spectrum Bit 8 = 5 GHz spectrum Bit 9 = Only passive scan allowed Bit 10 = Dynamic CCK-OFDM Bit 11 = Gaussian Frequency Shift Keying	Unsigned integer	2 bytes

	(GFSK) (FHSS PHY) Bit 12-15 = Reserved		
dBm-Ant0signal	RF signal power at antenna 0 Invalid value = -128	Signed integer	1 byte
dBm-Ant0noise	RF noise at antenna 0 Invalid value = -128	Signed integer	1 byte
dBm-Ant1signal	RF signal power at antenna 1 Invalid value = -128	Signed integer	1 byte
dBm-Ant1noise	RF noise at antenna 1 Invalid value = -128	Signed integer	1 byte
dBm-Ant2signal	RF signal power at antenna 2 Invalid value = -128	Signed integer	1 byte
dBm-Ant2noise	RF noise at antenna 2 Invalid value = -128	Signed integer	1 byte
dBm-Ant3signal	RF signal power at antenna 3 Invalid value = -128	Signed integer	1 byte
dBm-Ant3noise	RF noise at antenna 3 Invalid value = -128	Signed integer	1 byte
EVM0	Error vector magnitude for Chain 0 Invalid value = 0	Unsigned integer	4 bytes
EVM1	Error vector magnitude for Chain 1 Invalid value = 0	Unsigned integer	4 bytes
EVM2	Error vector magnitude for Chain 2 Invalid value = 0	Unsigned integer	4 bytes
EVM3	Error vector magnitude for Chain 3 Invalid value = 0	Unsigned integer	4 bytes

4.1.5 Spectrum-Map

Zero or more Spectrum-Map fields may be present in a single header. If one or more Spectrum-Map field is present, each field MUST have a unique Spectra-Type value. All fields are little-endian.

The Spectrum-Map field is intended to be compatible with

Field Name	Semantics	Value Type	Length
Time-Start	Start time of the sample	timeval struct	4 bytes
Time-End	End time of the sample	timeval struct	4 bytes
Spectra-Type	The type of data contained in the map (e.g. dB, dBm)	Unsigned integer	4 bytes

Start-kHz	Starting frequency in kHz	Unsigned integer	4 bytes
End-kHz	End frequency in kHz	Unsigned integer	4 bytes
Num-Samples	Number of samples	Unsigned integer	2 bytes
Sample-Data	Array of unsigned bytes. Length is Num-Samples.	Unsigned integer	variable

This information should be suitable for generating histograms.

The following values are defined for Spectra-Type:

Value	Data type
0	dBm
1	dB
2	RSSI

4.1.6 Process-Info

Zero or one Process-Info fields may be present in a single header. All fields are little-endian.

Field Name	Semantics	Value Type	Length
Process-ID	Process ID	Unsigned integer	4 bytes
Thread-ID	Thread ID	Unsigned integer	4 bytes
Process-Path-Len	Length of the process name	Unsigned integer	1 byte
Process-Path	Path and filename of the process	UTF-8 string	variable
User-ID	User ID	Unsigned integer	4 bytes
User-Name-Len	Length of user name	Unsigned integer	1 byte
User-Name	User name	UTF-8 string	variable
Group-ID	Primary group ID	Unsigned integer	4 bytes
Group-Name-Len	Length of primary group name	Unsigned integer	1 byte
Group-Name	Primary group name	UTF-8 string	variable

4.1.7 Capture-Info

To be defined.

5. Vendor-Specific Field Types

Type values 30,000 to 65,535 are reserved for vendor-specific applications. Vendor numbers are assigned by the WinPcap development team, and assignment may be handed over to a formal standards body in the future. To request a vendor number, send an email to winpcap-users@winpcap.org.

A vendor-specific type MUST NOT be used without first obtaining an assignment from the WinPcap development team. The intent behind such a large space for vendor-specific types is to allow easy and unadulterated registration, thus maintaining some sanity in version tracking and parsing.

Type	Length	Description
30,000	-	Intel Corporation

30,001 – 59,917	-	Unassigned.
51,918	-	Reserved for CACE. ($51918_{10} = CACE_{16}$)
51,919 – 65,535	-	Unassigned.