Workgroup: Network Working Group
Internet-Draft: draft-tuexen-opsawg-pcapng-04
Published: 4 October 2021
Intended Status: Informational
Expires: 7 April 2022
Authors: M. Tuexen, Ed.
Muenster Univ. of Appl. Sciences
F. Risso — Politecnico di Torino
J. Bongertz — Airbus DS CyberSecurity
G. Combs — Wireshark
G. Harris
E. Chaudron — Red Hat
M. Richardson — Sandelman

PCAP Next Generation (pcapng) Capture File Format

Abstract

This document describes a format to record captured packets to a file. This format is extensible; Wireshark can currently read and write it, and libpcap can currently read some pcapng files.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 7 April 2022.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in

Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction
2. Terminology
   2.1. Acronyms
3. General File Structure
   3.1. General Block Structure
   3.2. Block Types
   3.3. Logical Block Hierarchy
   3.4. Physical File Layout
   3.5. Options
      3.5.1. Custom Options
   3.6. Data format
      3.6.1. Endianness
      3.6.2. Alignment
4. Block Definition
   4.1. Section Header Block
   4.2. Interface Description Block
   4.3. Enhanced Packet Block
      4.3.1. Enhanced Packet Block Flags Word
   4.4. Simple Packet Block
   4.5. Name Resolution Block
   4.6. Interface Statistics Block
   4.7. Decryption Secrets Block
   4.8. Custom Block
5. Vendor-Specific Custom Extensions
   5.1. Supported Use-Cases
   5.2. Controlling Copy Behavior
   5.3. Strings vs. Octets
   5.4. Endianness Issues
6. Recommended File Name Extension: .pcapng
7. Conclusions
8. Implementations
9. Security Considerations
10. IANA Considerations
   10.1. Standardized Block Type Codes
11. Contributors
12. Acknowledgments
13. References
   13.1. Normative References
   13.2. Informative References

Appendix A. Packet Block (obsolete!)

Authors' Addresses

1. Introduction

The problem of exchanging packet traces becomes more and more critical every day; unfortunately, no standard solutions exist for this task right now. One of the most accepted packet interchange formats is the one defined by libpcap, which is rather old and is lacking in functionality for more modern applications particularly from the extensibility point of view.

This document proposes a new format for recording packet traces. The following goals are being pursued:

Extensibility: It should be possible to add new standard capabilities to the file format over time, and third parties should be able to enrich the information embedded in the file with proprietary extensions, with tools unaware of newer extensions being able to ignore them.

Portability: A capture trace must contain all the information needed to read data independently from network, hardware and operating system of the machine that made the capture.

Merge/Append data: It should be possible to add data at the end of a given file, and the resulting file must still be readable.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2.1. Acronyms

The following acronyms are used throughout this document:

SHB: Section Header Block

IDB: Interface Description Block

ISB: Interface Statistics Block

EPB: Enhanced Packet Block

SPB: Simple Packet Block

NRB: Name Resolution Block

CB: Custom Block

3. General File Structure

3.1. General Block Structure

A capture file is organized in blocks, that are appended one to another to form the file. All the blocks share a common format, which is shown in Figure 1.

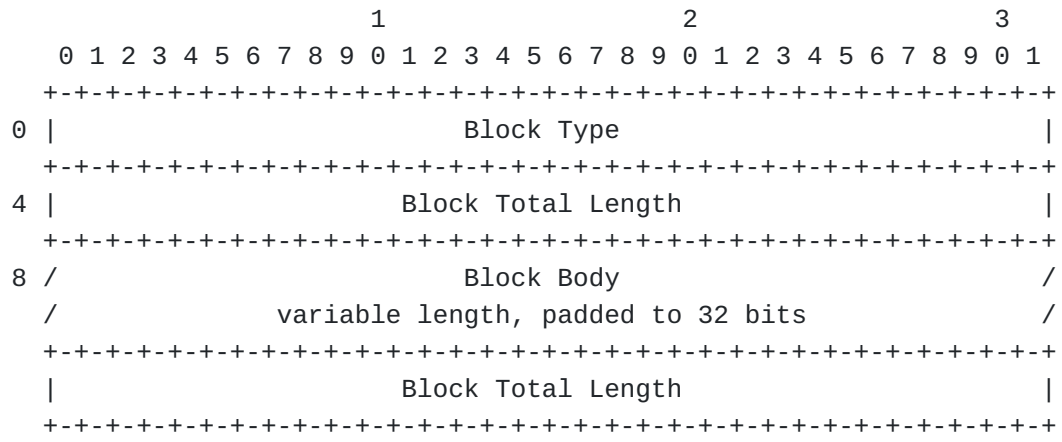


Figure 1: Basic block structure.

The fields have the following meaning:

- Block Type (32 bits): a unique unsigned value that identifies the block. Values whose Most Significant Bit (MSB) is equal to 1 are reserved for local use. They can be used to make extensions to the file format to save private data to the file. The list of currently defined types can be found in Section 10.1.

- Block Total Length (32 bits): an unsigned value giving the total size of this block, in octets. For instance, the length of a block that does not have a body is 12 octets: 4 octets for the Block Type, 4 octets for the initial Block Total Length and 4 octets for the trailing Block Total Length. This value MUST be a multiple of 4.

- Block Body: content of the block.

- Block Total Length: total size of this block, in octets. This field is duplicated to permit backward file navigation.

This structure, shared among all blocks, makes it easy to process a file and to skip unneeded or unknown blocks. Some blocks can contain other blocks inside (nested blocks). Some of the blocks are mandatory, i.e. a capture file is not valid if they are not present, other are optional.

The General Block Structure allows defining other blocks if needed. A parser that does not understand them can simply ignore their content.

3.2. Block Types

The currently standardized Block Type codes are specified in Section 10.1; they have been grouped in the following four categories:

The following MANDATORY block MUST appear at least once in each file:

* Section Header Block (Section 4.1): it defines the most important characteristics of the capture file.

The following OPTIONAL blocks MAY appear in a file:

* Interface Description Block (Section 4.2): it defines the most important characteristics of the interface(s) used for capturing traffic. This block is required in certain cases, as described later.

* Enhanced Packet Block (Section 4.3): it contains a single captured packet, or a portion of it. It represents an evolution of the original, now obsolete, Packet Block (Appendix A). If this appears in a file, an Interface Description Block is also required, before this block.

* Simple Packet Block (Section 4.4): it contains a single captured packet, or a portion of it, with only a minimal set of information about it. If this appears in a file, an Interface Description Block is also required, before this block.

* Name Resolution Block (Section 4.5): it defines the mapping from numeric addresses present in the packet capture and the canonical name counterpart.

* Interface Statistics Block (Section 4.6): it defines how to store some statistical data (e.g. packet dropped, etc) which can be useful to understand the conditions in which the capture has been made. If this appears in a file, an Interface Description Block is also required, before this block.

* Custom Block (Section 4.8): it contains vendor-specific data in a portable fashion.

The following OBSOLETE block SHOULD NOT appear in newly written files (but is documented in the Appendix for reference):

- Packet Block (Appendix A): it contains a single captured packet, or a portion of it. It is OBSOLETE, and superseded by the Enhanced Packet Block (Section 4.3).

The following EXPERIMENTAL blocks are considered interesting but the authors believe that they deserve more in-depth discussion before being defined:

- Alternative Packet Blocks
- Compression Block
- Encryption Block
- Fixed Length Block
- Directory Block
- Traffic Statistics and Monitoring Blocks
- Event/Security Blocks

Requests for new standardized Block Type codes should be made by creating a pull request to update this document as described in Section 10.1.

3.3. Logical Block Hierarchy

The blocks build a logical hierarchy as they refer to each other. Figure 2 shows the logical hierarchy of the currently defined blocks in the form of a "tree view":

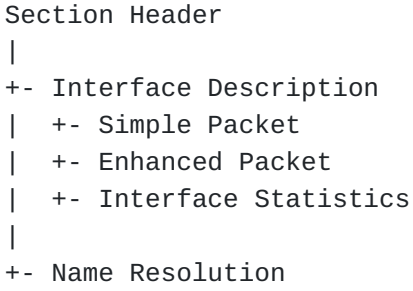


Figure 2: Logical Block Hierarchy of a pcapng File

For example: each captured packet refers to a specific capture interface, the interface itself refers to a specific section.

3.4. Physical File Layout

The file MUST begin with a Section Header Block. However, more than one Section Header Block can be present in the capture file, each one covering the data following it until the next one (or the end of file). A Section includes the data delimited by two Section Header Blocks (or by a Section Header Block and the end of the file), including the first Section Header Block.

In case an application cannot read a Section because of different version number, it MUST skip everything until the next Section Header Block. Note that, in order to properly skip the blocks until the next section, all blocks MUST have the fields Type and Length at the beginning. In order to properly skip blocks in the backward direction, all blocks MUST have the Length repeated at the end of the block. These are mandatory requirements that MUST be maintained in future versions of the block format.

Figure 3 shows a typical file layout, with a single Section Header that covers the whole file.



Figure 3: File structure example: Typical layout with a single Section Header Block

Figure 4 shows a file that contains three headers, and is normally the result of file concatenation. An application that understands only version 1.0 of the file format skips the intermediate section and restart processing the packets after the third Section Header.

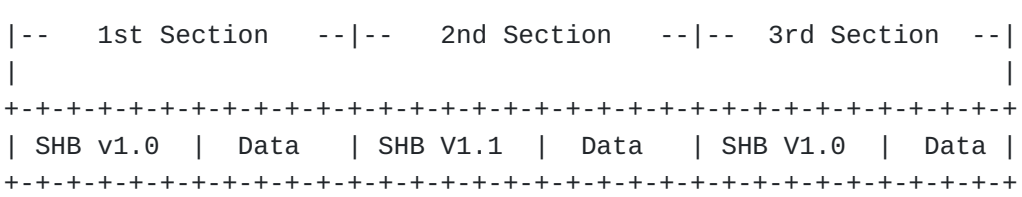


Figure 4: File structure example: three Section Header Blocks in a single file

Figure 5 shows a file comparable to a "classic libpcap" file - the minimum for a useful capture file. It contains a single Section Header Block (SHB), a single Interface Description Block (IDB) and a few Enhanced Packet Blocks (EPB).

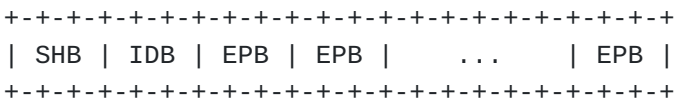


Figure 5: File structure example: a pcapng file similar to a classical libpcap file

Figure 6 shows a complex example file. In addition to the minimum file above, it contains packets captured from three interfaces, capturing on the third of which begins after packets have arrived on other interfaces, and also includes some Name Resolution Blocks (NRB) and an Interface Statistics Block (ISB).



Figure 6: File structure example: complex pcapng file

The last example should make it obvious that the block structure makes the file format very flexible compared to the classical libpcap format.

3.5. Options

All the block bodies MAY embed optional fields. Optional fields can be used to insert some information that may be useful when reading data, but that is not really needed for packet processing. Therefore, each tool can either read the content of the optional fields (if any), or skip some of them or even all at once.

A block that may contain options must be structured so that the number of octets of data in the Block Body that precede the options can be determined from that data; that allows the beginning of the options to be found. That is true for all standard blocks that support options; for Custom Blocks that support options, the Custom Data must be structured in such a fashion. This means that the Block Length field (present in the General Block Structure, see Section 3.1) can be used to determine how many octets of optional fields, if any, are present in the block. That number can be used to determine whether the block has optional fields (if it is zero, there are no optional fields), to check, when processing optional fields, whether any optional fields remain, and to skip all the optional fields at once.

Options are a list of Type - Length - Value fields, each one containing a single value:

- Option Type (16 bits): an unsigned value that contains the code that specifies the type of the current TLV record. Option types whose Most Significant Bit is equal to one are reserved for local use; therefore, there is no guarantee that the code used is unique among all capture files (generated by other applications), and is most certainly not portable. For cross-platform globally unique vendor-specific extensions, the Custom Option MUST be used instead, as defined in Section 3.5.1).

- Option Length (16 bits): an unsigned value that contains the actual length of the following 'Option Value' field without the padding octets.

- Option Value (variable length): the value of the given option, padded to a 32-bit boundary. The actual length of this field (i.e. without the padding octets) is specified by the Option Length field.

Requests for new standardized option codes for a given block should be made by creating a pull request to update this document as described in Section 10.1.

A given option may have a fixed length, in which case all instances of that option have a length that is equal to the specified fixed length, or a variable length, in which case the option has a minimum length and all instances of that option must have a length equal to or greater than the specified minimum length. The length of fixed-length options, and the minimum length of variable-length options, is specified in the description of the option; if the minimum length of a variable-length option is not specified, a zero-length option is valid. Software that reads these files SHOULD report options that have an invalid length as errors; the software MAY stop processing the file if it sees an option that has invalid length, or MAY ignore the option and continue processing it. Software that writes these files MUST NOT write files with options that have invalid lengths.

If an option's value is a string, the value is not necessarily zero-terminated. Software that reads these files MUST NOT assume that strings are zero-terminated, and MUST treat a zero-value octet as a string terminator.

Some options may be repeated several times; for example, a block can have multiple comments, and an Interface Description Block can give multiple IPv4 or IPv6 addresses for the interface if it has multiple IPv4 or IPv6 addresses assigned to it. Other options may appear at most once in a given block.

The option list is terminated by a option which uses the special 'End of Option' code (opt_endofopt). Code that writes pcapng files MUST put an opt_endofopt option at the end of an option list. Code that reads pcapng files MUST NOT assume an option list will have an opt_endofopt option at the end; it MUST also check for the end of the block, and SHOULD treat blocks where the option list has no opt_endofopt option as if the option list had an opt_endofopt option at the end.

The format of the optional fields is shown in Figure 7.

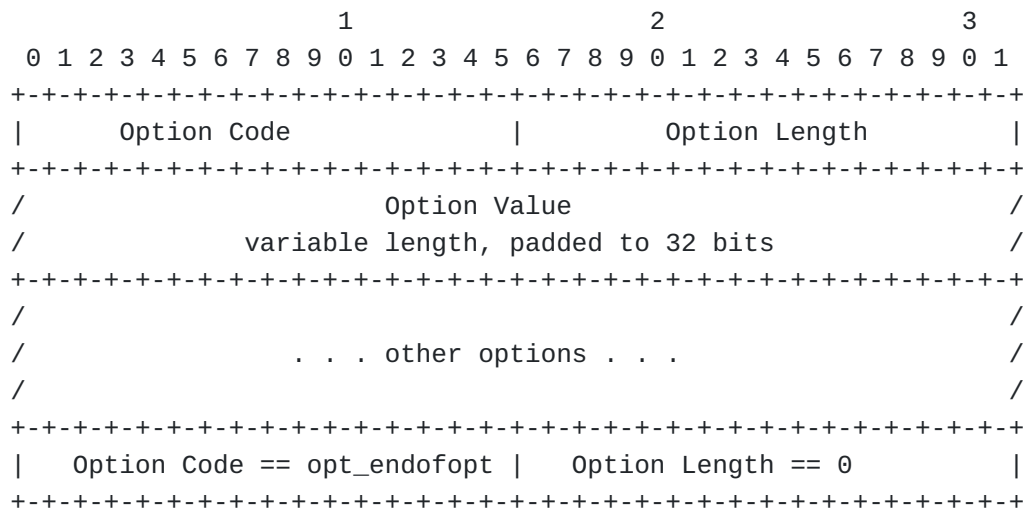


Figure 7: Options Format

The following codes can always be present in any optional field:

Name	Code	Length	Multiple allowed?
opt_endofopt	0	0	no
opt_comment	1	variable	yes
opt_custom	2988/2989/19372/19373	variable, minimum 4	yes

Table 1: Common Options

opt_endofopt:
The opt_endofopt option delimits the end of the optional fields. This option MUST NOT be repeated within a given list of options.

opt_comment:
The opt_comment option is a UTF-8 string containing human-readable comment text that is associated to the current block. Line separators SHOULD be a carriage-return + linefeed ('\r\n') or just linefeed ('\n'); either form may appear and be considered a line separator. The string is not zero-terminated.

Examples: "This packet is the beginning of all of our problems", "Packets 17-23 showing a bogus TCP retransmission!\r\n This is reported in bugzilla entry 1486.\nIt will be fixed in the future.".

opt_custom:
This option is described in detail in Section 3.5.1.

3.5.1. Custom Options

Customs Options are used for portable, vendor-specific data related to the block they're in. A Custom Option can be in any block type that can have options, can be repeated any number of times in a block, and may come before or after other option types - except the opt_endofopt option, which is always the last option. Different Custom Options, of different type codes and/or different Private Enterprise Numbers, may be used in the same pcapng file. See Section 5 for additional details.

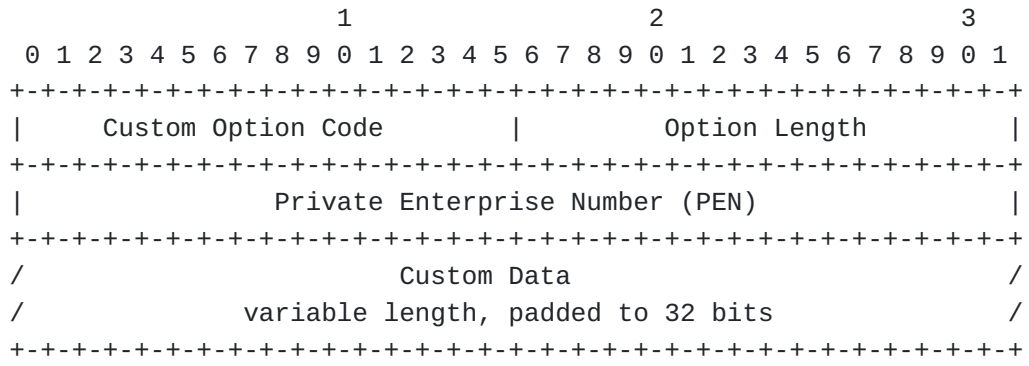


Figure 8: Custom Options Format

The Custom Option has the following fields:

*Custom Option Code: The code number for the Custom Option, which can be one of the following decimal numbers:

2988:
This option code identifies a Custom Option containing a UTF-8 string in the Custom Data portion. The string is not zero-terminated. This Custom Option can be safely copied to a new file if the pcapng file is manipulated by an application; otherwise 19372 should be used instead. See Section 5.2 for details.

2989:
This option code identifies a Custom Option containing binary octets in the Custom Data portion. This Custom Option can be safely copied to a new file if the pcapng file is manipulated

by an application; otherwise 19372 should be used instead. See Section 5.2 for details.

19372:
This option code identifies a Custom Option containing a UTF-8 string in the Custom Data portion. The string is not zero-terminated. This Custom Option should not be copied to a new file if the pcapng file is manipulated by an application. See Section 5.2 for details.

19373:
This option code identifies a Custom Option containing binary octets in the Custom Data portion. This Custom Option should not be copied to a new file if the pcapng file is manipulated by an application. See Section 5.2 for details.

*Option Length: as described in Section 3.1, this contains the length of the option's value, which includes the 4-octet Private Enterprise Number and variable-length Custom Data fields, without the padding octets.

*Private Enterprise Number: An IANA-assigned Private Enterprise Number identifying the organization which defined the Custom Option. See Section 5.1 for details. The PEN MUST be encoded using the same endianness as the Section Header Block it is within the scope of.

*Custom Data: the custom data, padded to a 32 bit boundary.

3.6. Data format

3.6.1. Endianness

Data contained in each section will always be saved according to the characteristics (little endian / big endian) of the capturing machine. This refers to all the fields that are saved as numbers and that span over two or more octets.

The approach of having each section saved in the native format of the generating host is more efficient because it avoids translation of data when reading / writing on the host itself, which is the most common case when generating/processing capture captures.

Please note: The endianness is indicated by the Section Header Block (Section 4.1). Since this block can appear several times in a pcapng file, a single file can contain both endianness variants.

3.6.2. Alignment

All fields of this specification use proper alignment for 16- and 32-bit values. This makes it easier and faster to read/write file contents if using techniques like memory mapped files.

The alignment octets (marked in this document e.g. with "padded to 32 bits") MUST be filled with zeroes.

Please note: 64-bit values are not aligned to 64-bit boundaries. This is because the file is naturally aligned to 32-bit boundaries only. Special care MUST be taken when reading and writing such values. (Note also that some 64-bit values are represented as a 64-bit integer in the endianness of the machine that wrote the file, and others are represented as 2 32-bit values, one containing the upper 32 bits of the value and one containing the lower 32 bits of the value, each written as 32-bit integers in the endianness of the machine that wrote the file. Neither of these formats guarantee 64-bit alignment.)

4. Block Definition

This section details the format of the blocks currently defined.

4.1. Section Header Block

The Section Header Block (SHB) is mandatory. It identifies the beginning of a section of the capture file. The Section Header Block does not contain data but it rather identifies a list of blocks (interfaces, packets) that are logically correlated. Its format is shown in Figure 9.

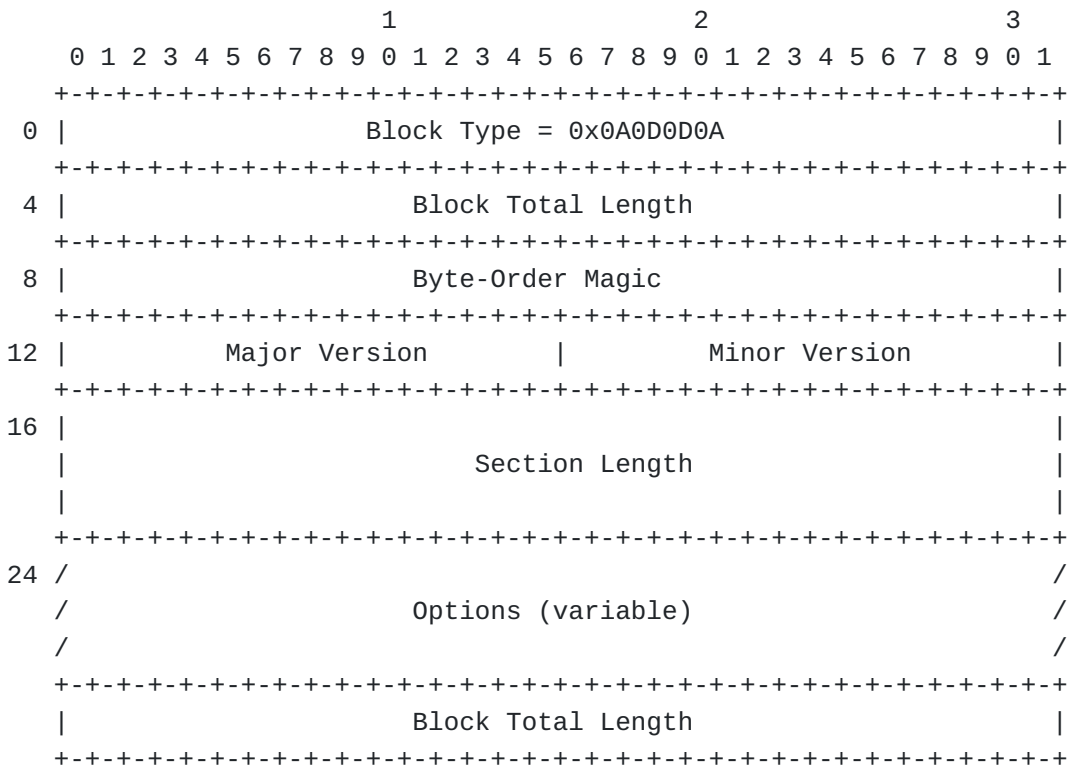


Figure 9: Section Header Block Format

The meaning of the fields is:

*Block Type: The block type of the Section Header Block is the integer corresponding to the 4-char string "\n\r\r\n" (0x0A0D0D0A). This particular value is used for 2 reasons:

1. This number is used to detect if a file has been transferred via FTP or HTTP from a machine to another with an inappropriate ASCII conversion. In this case, the value of this field will differ from the standard one ("\n\r\r\n") and the reader can detect a possibly corrupted file.

2. This value is palindromic, so that the reader is able to recognize the Section Header Block regardless of the endianness of the section. The endianness is recognized by reading the Byte Order Magic, which is located 8 octets after the Block Type.

*Block Total Length: total size of this block, as described in Section 3.1.

*Byte-Order Magic (32 bits): an unsigned magic number, whose value is the hexadecimal number 0x1A2B3C4D. This number can be used to distinguish sections that have been saved on little-endian

machines from the ones saved on big-endian machines, and to heuristically identify pcapng files.

*Major Version (16 bits): an unsigned value, giving the number of the current major version of the format. The value for the current version of the format is 1.

*Minor Version (16 bits): an unsigned value, giving the number of the current minor version of the format. The value for the current version of the format is 0.

*Section Length (64 bits): a signed value specifying the length in octets of the following section, excluding the Section Header Block itself. This field can be used to skip the section, for faster navigation inside large files. If the Section Length is -1 (0xFFFFFFFFFFFFFFFF), this means that the size of the section is not specified, and the only way to skip the section is to parse the blocks that it contains. Please note that if this field is valid (i.e. not negative), its value is always a multiple of 4, as all the blocks are aligned to and padded to 32-bit (4 octet) boundaries. Also, special care should be taken in accessing this field: since the alignment of all the blocks in the file is 32-bits, this field is not guaranteed to be aligned to a 64-bit boundary. This could be a problem on 64-bit processors.

*Options: optionally, a list of options (formatted according to the rules defined in Section 3.5) can be present.

Writers of pcapng files MUST NOT write SHBs with a Major Version other than 1 or a Minor Version other than 0. If they do so, they will write a file that many readers of pcapng files, such as programs using libpcap to read pcapng files (including, but not limited to, tcpdump), Wireshark, and possibly other programs not to be able to read their files.

Some pcapng file writers have used a minor version of 2, but the file format did not change incompatibly (new block types were added); Readers of pcapng files MUST treat a Minor Version of 2 as equivalent to a Minor Version of 0 (and, if they also write a pcapng file based on the results of reading one or more pcapng files, they MUST NOT, as per the previous sentence, write an SHB with a Minor Version of 2, even if they read an SHB with a Minor Version of 2). As indicated above, using a minor version number other than 0 when writing a section of a pcapng file will produce a section that most existing software will not be able to read; future versions of some of that software will be able to read sections with a version of 1.2, but older copies of that software that are not updated to the latest version will still not be able to read them.

The Major Version would be changed only if a new version of this specification, for a later major version of the file format, were created. Such a version would only be created if the format were to change in such a way that code that reads the new format could not read the old format (i.e., code to read both formats would have to check the version number and use different code paths for the two formats) and code that reads the old format could not read the new format. An incompatible change to the format of an existing block or an existing option would be such a change; the addition of a new block or a new option would not be such a change. An example of such an incomparible change would be the addition of an additional field to the Section Header Block, following the Minor Version field and before the Snaplen field; software expecting the new SHB format would not correctly read the old SHB format, and software expecting the old SHB format would not correctly read the new SHB format. (Note that a change to the SHB must leave the Block Type, Block Total Length, Byte-Order Magic, Major Version, and Minor Version fields at the same offsets from the beginning of the SHB and with the same lengths, must keep the value of the Block TYpe the same, must keep the two posssible values of the Byte-Order Magic the same, depending on the block's byte order, so that the rest of the SHB can be correctly interpreted.)

The Minor Version would be changed only if a new version of this specification, for a later minor version of the file format, were created. Such a version would only be created if the format were to change in such a way that code that reads the new format could read the old format without checking the version number but code that reads the old format could not read all files in the new format. A backward-compatible change to the format of an existing block or an existing opption would be such a change; the addition of a new block or a new option would not be such a change. An example of such a backward-compatible but not forward-compatible change would be a change to the Interface Description block (see below) to use the current Reserved field to indicate the presence of additional fields before the Options, with a zero value indicate no such fields are present.

I.e., adding new block types or options would not require that either the Major Version or the Minor Version be changed, as code that does not know about the block type or option should just skip it; only if skipping a block or option does not work should the minor version number be changed.

Aside from the options defined in Section 3.5, the following options are valid within this block:

Name	Code	Length	Multiple allowed?
shb_hardware	2	variable	no
shb_os	3	variable	no
shb_userappl	4	variable	no

Table 2: Section Header Block Options

shb_hardware:

The shb_hardware option is a UTF-8 string containing the description of the hardware used to create this section. The string is not zero-terminated.

Examples: "x86 Personal Computer", "Sun Sparc Workstation".

shb_os:

The shb_os option is a UTF-8 string containing the name of the operating system used to create this section. The string is not zero-terminated.

Examples: "Windows XP SP2", "openSUSE 10.2".

shb_userappl:

The shb_userappl option is a UTF-8 string containing the name of the application used to create this section. The string is not zero-terminated.

Examples: "dumpcap V0.99.7".

[Open issue: does a program which re-writes a capture file change the original hardware/os/application info?]

4.2. Interface Description Block

An Interface Description Block (IDB) is the container for information describing an interface on which packet data is captured.

Tools that write / read the capture file associate an incrementing unsigned 32-bit number (starting from '0') to each Interface Definition Block, called the Interface ID for the interface in question. This number is unique within each Section and identifies the interface to which the IDB refers; it is only unique inside the current section, so, two Sections can have different interfaces identified by the same Interface ID values. This unique identifier is referenced by other blocks, such as Enhanced Packet Blocks and Interface Statistic Blocks, to indicate the interface to which the block refers (such the interface that was used to capture the packet that an Enhanced Packet Block contains or to which the statistics in an Interface Statistic Block refer).

There must be an Interface Description Block for each interface to which another block refers. Blocks such as an Enhanced Packet Block or an Interface Statistics Block contain an Interface ID value referring to a particular interface, and a Simple Packet Block implicitly refers to an interface with an Interface ID of 0. If the file does not contain any blocks that use an Interface ID, then the file does not need to have any IDBs.

An Interface Description Block is valid only inside the section to which it belongs. The structure of a Interface Description Block is shown in Figure 10.

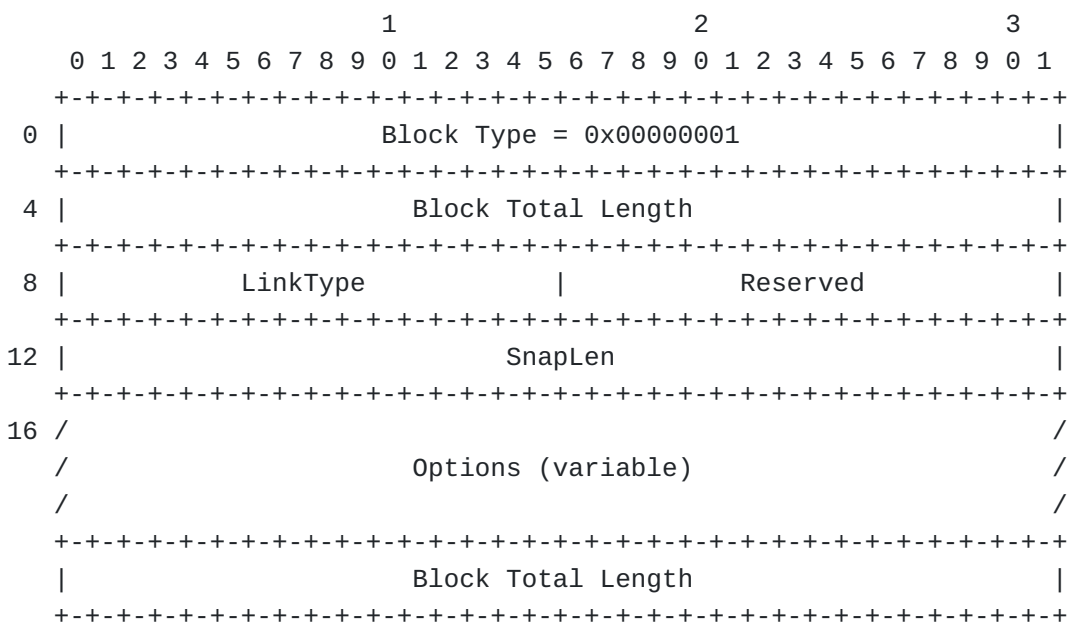


Figure 10: Interface Description Block Format

The meaning of the fields is:

- Block Type: The block type of the Interface Description Block is 1.

- Block Total Length: total size of this block, as described in Section 3.1.

- LinkType (16 bits): an unsigned value that defines the link layer type of this interface. The list of Standardized Link Layer Type codes is available in [LINKTYPES].

- Reserved (16 bits): not used - MUST be filled with 0 by pcapng file writers, and MUST be ignored by pcapng file readers.

- SnapLen (32 bits): an unsigned value indicating the maximum number of octets captured from each packet. The portion of each

packet that exceeds this value will not be stored in the file. A value of zero indicates no limit.

*Options: optionally, a list of options (formatted according to the rules defined in Section 3.5) can be present.

In addition to the options defined in Section 3.5, the following options are valid within this block:

Name	Code	Length	Multiple allowed?
if_name	2	variable	no
if_description	3	variable	no
if_IPv4addr	4	8	yes
if_IPv6addr	5	17	yes
if_MACaddr	6	6	no
if_EUIaddr	7	8	no
if_speed	8	8	no
if_tsresol	9	1	no
if_tzone	10	4	no
if_filter	11	variable, minimum 1	no
if_os	12	variable	no
if_fcslen	13	1	no
if_tsoffset	14	8	no
if_hardware	15	variable	no
if_txspeed	16	8	no
if_rxspeed	17	8	no

Table 3: Interface Description Block Options

if_name:

The if_name option is a UTF-8 string containing the name of the device used to capture data. The string is not zero-terminated.

Examples: "eth0", "\Device\NPF_{AD1CE675-96D0-47C5-ADD0-2504B9126B68}".

if_description:

The if_description option is a UTF-8 string containing the description of the device used to capture data. The string is not zero-terminated.

Examples: "Wi-Fi", "Local Area Connection", "Wireless Network Connection", "First Ethernet Interface".

if_IPv4addr:

The if_IPv4addr option is an IPv4 network address and corresponding netmask for the interface. The first four octets are the IP address, and the next four octets are the netmask. This option can be repeated multiple times within the same Interface Description Block when multiple IPv4 addresses are

assigned to the interface. Note that the IP address and netmask are both treated as four octets, one for each octet of the address or mask; they are not 32-bit numbers, and thus the endianness of the SHB does not affect this field's value.

Examples: '192 168 1 1 255 255 255 0'.

if_IPv6addr:
The if_IPv6addr option is an IPv6 network address and corresponding prefix length for the interface. The first 16 octets are the IP address and the next octet is the prefix length. This option can be repeated multiple times within the same Interface Description Block when multiple IPv6 addresses are assigned to the interface.

Example: 2001:0db8:85a3:08d3:1319:8a2e:0370:7344/64 is written (in hex) as '20 01 0d b8 85 a3 08 d3 13 19 8a 2e 03 70 73 44 40'.

if_MACaddr:
The if_MACaddr option is the Interface Hardware MAC address (48 bits), if available.

Example: '00 01 02 03 04 05'.

if_EUIaddr:
The if_EUIaddr option is the Interface Hardware EUI address (64 bits), if available.

Example: '02 34 56 FF FE 78 9A BC'.

if_speed:
The if_speed option is a 64-bit unsigned value indicating the interface speed, in bits per second.

Example: the 64-bit decimal number 100000000 for 100Mbps.

if_tsresol:
The if_tsresol option identifies the resolution of timestamps. If the Most Significant Bit is equal to zero, the remaining bits indicates the resolution of the timestamp as a negative power of 10 (e.g. 6 means microsecond resolution, timestamps are the number of microseconds since 1970-01-01 00:00:00 UTC). If the Most Significant Bit is equal to one, the remaining bits indicates the resolution as negative power of 2 (e.g. 10 means 1/1024 of second). If this option is not present, a resolution of 10^-6 is assumed (i.e. timestamps have the same resolution of the standard 'libpcap' timestamps).

Example: '6'.

if_tzone:
   The if_tzone option identifies the time zone for GMT support (TODO: specify better).

Example: TODO: give a good example.

if_filter:
   The if_filter option identifies the filter (e.g. "capture only TCP traffic") used to capture traffic. The first octet of the Option Data keeps a code of the filter used (e.g. if this is a libpcap string, or BPF bytecode, and more). More details about this format will be presented in Appendix XXX (TODO). (TODO: better use different options for different fields? e.g. if_filter_pcap, if_filter_bpf, ...)

Example: '00'"tcp port 23 and host 192.0.2.5".

if_os:
   The if_os option is a UTF-8 string containing the name of the operating system of the machine in which this interface is installed. This can be different from the same information that can be contained by the Section Header Block (Section 4.1) because the capture can have been done on a remote machine. The string is not zero-terminated.

Examples: "Windows XP SP2", "openSUSE 10.2".

if_fcslen:
   The if_fcslen option is an 8-bit unsigned integer value that specifies the length of the Frame Check Sequence (in bits) for this interface. For link layers whose FCS length can change during time, the Enhanced Packet Block epb_flags Option can be used in each Enhanced Packet Block (see Section 4.3.1).

Example: '4'.

if_tsoffset:
   The if_tsoffset option is a 64-bit signed integer value that specifies an offset (in seconds) that must be added to the timestamp of each packet to obtain the absolute timestamp of a packet. If the option is missing, the timestamps stored in the packet MUST be considered absolute timestamps. The time zone of the offset can be specified with the option if_tzone. TODO: won't a if_tsoffset_low for fractional second offsets be useful for highly synchronized capture systems?

Example: '1234'.

if_hardware:

The if_hardware option is a UTF-8 string containing the description of the interface hardware. The string is not zero-terminated.

Examples: "Broadcom NetXtreme", "Intel(R) PRO/1000 MT Network Connection", "NETGEAR WNA1000Mv2 N150 Wireless USB Micro Adapter".

if_txspeed:
The if_txrxspeeds option is a 64-bit unsigned value indicating the interface transmit speed in bits per second.

Example: the 64-bit decimal number 1024000 for 1024Kbps.

if_rxspeed:
The if_rxspeed option is a 64-bit unsigned value indicating the interface receive speed, in bits per second.

Example: the 64-bit decimal number 8192000 for 8192Kbps.

If the interface transmit speed and receive speed are the same, the if_speed option MUST be used and the if_txspeed and if_rxspeed options MUST NOT be used. If the transmit speed is unknown, the if_speed and if_txspeed options MUST NOT be used; if the receive speed is unknown, the if_speed and if_rxspeed options MUST NOT be used.

4.3. Enhanced Packet Block

An Enhanced Packet Block (EPB) is the standard container for storing the packets coming from the network. The Enhanced Packet Block is optional because packets can be stored either by means of this block or the Simple Packet Block, which can be used to speed up capture file generation; or a file may have no packets in it. The format of an Enhanced Packet Block is shown in Figure 11.

The Enhanced Packet Block is an improvement over the original, now obsolete, Packet Block (Appendix A):

- it stores the Interface Identifier as a 32-bit integer value. This is a requirement when a capture stores packets coming from a large number of interfaces;

- unlike the Packet Block (Appendix A), the number of packets dropped by the capture system between this packet and the previous one is not stored in the header, but rather in an option of the block itself.

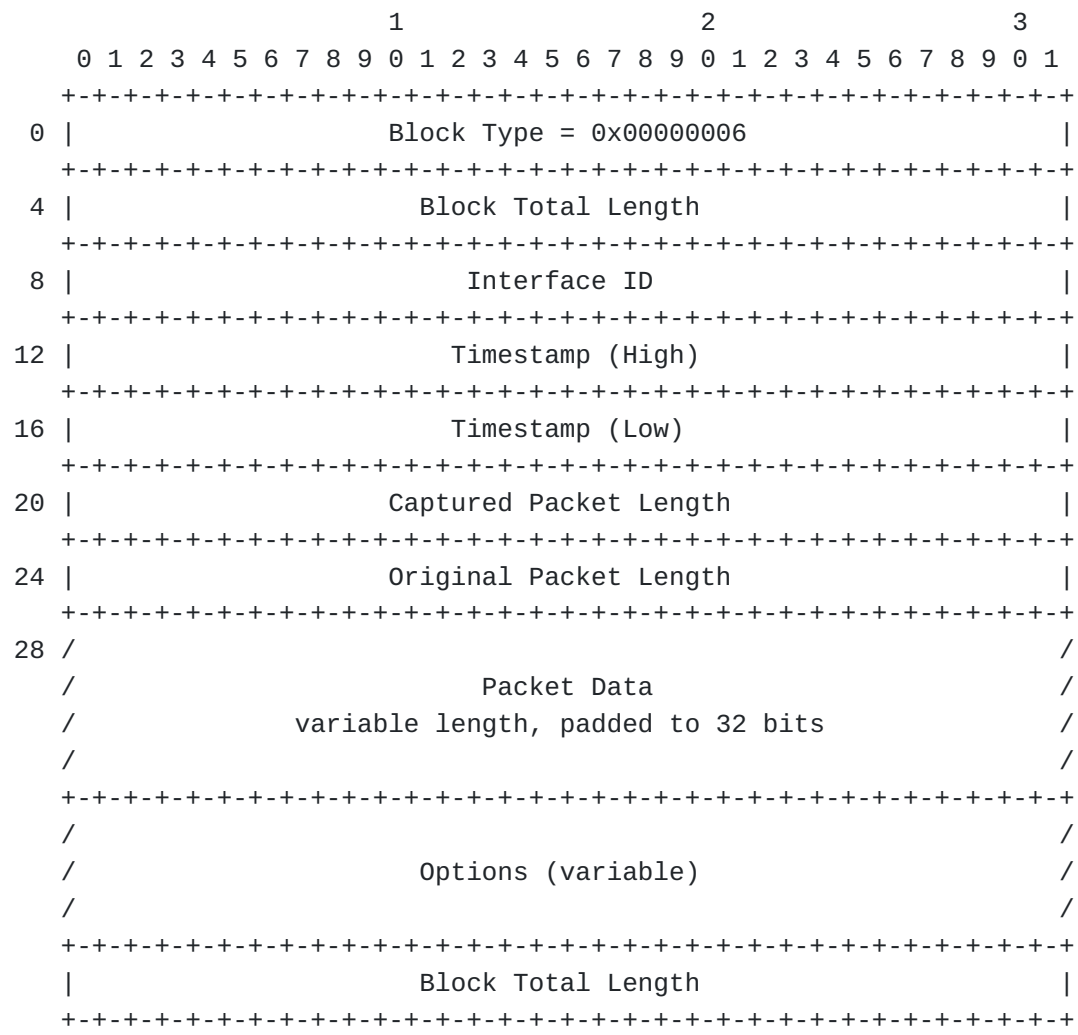


Figure 11: Enhanced Packet Block Format

The Enhanced Packet Block has the following fields:

*Block Type: The block type of the Enhanced Packet Block is 6.

*Block Total Length: total size of this block, as described in Section 3.1.

*Interface ID (32 bits): an unsigned value that specifies the interface on which this packet was received or transmitted; the correct interface will be the one whose Interface Description Block (within the current Section of the file) is identified by the same number (see Section 4.2) of this field. The interface ID MUST be valid, which means that an matching interface description block MUST exist.

*Timestamp (High) and Timestamp (Low): upper 32 bits and lower 32 bits of a 64-bit timestamp. The timestamp is a single 64-bit unsigned integer that represents the number of units of time that

have elapsed since 1970-01-01 00:00:00 UTC. The length of a unit of time is specified by the 'if_tsresol' option (see Figure 10) of the Interface Description Block referenced by this packet. Note that, unlike timestamps in the libpcap file format, timestamps in Enhanced Packet Blocks are not saved as two 32-bit values that represent the seconds and microseconds that have elapsed since 1970-01-01 00:00:00 UTC. Timestamps in Enhanced Packet Blocks are saved as two 32-bit words that represent the upper and lower 32 bits of a single 64-bit quantity.

*Captured Packet Length (32 bits): an unsigned value that indicates the number of octets captured from the packet (i.e. the length of the Packet Data field). It will be the minimum value among the Original Packet Length and the snapshot length for the interface (SnapLen, defined in Figure 10). The value of this field does not include the padding octets added at the end of the Packet Data field to align the Packet Data field to a 32-bit boundary.

*Original Packet Length (32 bits): an unsigned value that indicates the actual length of the packet when it was transmitted on the network. It can be different from the Captured Packet Length if the packet has been truncated by the capture process.

*Packet Data: the data coming from the network, including link-layer headers. The actual length of this field is Captured Packet Length plus the padding to a 32-bit boundary. The format of the link-layer headers depends on the LinkType field specified in the Interface Description Block (see Section 4.2) and it is specified in the entry for that format in [LINKTYPES].

*Options: optionally, a list of options (formatted according to the rules defined in Section 3.5) can be present.

In addition to the options defined in Section 3.5, the following options are valid within this block:

Name	Code	Length	Multiple allowed?
epb_flags	2	4	no
epb_hash	3	variable, minimum hash type-dependent	yes
epb_dropcount	4	8	no
epb_packetid	5	8	no
epb_queue	6	4	no
epb_verdict	7	variable, minimum verdict type-dependent	yes

Table 4: Enhanced Packet Block Options

epb_flags:
The epb_flags option is a 32-bit flags word containing link-layer information. A complete specification of the allowed flags can be found in Section 4.3.1.

Example: '0'.

epb_hash:
The epb_hash option contains a hash of the packet. The first octet specifies the hashing algorithm, while the following octets contain the actual hash, whose size depends on the hashing algorithm, and hence from the value in the first octet. The hashing algorithm can be: 2s complement (algorithm octet = 0, size = XXX), XOR (algorithm octet = 1, size=XXX), CRC32 (algorithm octet = 2, size = 4), MD-5 (algorithm octet = 3, size = 16), SHA-1 (algorithm octet = 4, size = 20), Toeplitz (algorithm octet = 5, size = 4). The hash covers only the packet, not the header added by the capture driver: this gives the possibility to calculate it inside the network card. The hash allows easier comparison/merging of different capture files, and reliable data transfer between the data acquisition system and the capture library.

Examples: '02 EC 1D 87 97', '03 45 6E C2 17 7C 10 1E 3C 2E 99 6E C2 9A 3D 50 8E'.

epb_dropcount:
The epb_dropcount option is a 64-bit unsigned integer value specifying the number of packets lost (by the interface and the operating system) between this packet and the preceding one for the same interface or, for the first packet for an interface, between this packet and the start of the capture process.

Example: '0'.

epb_packetid:
The epb_packetid option is a 64-bit unsigned integer that uniquely identifies the packet. If the same packet is seen by multiple interfaces and there is a way for the capture application to correlate them, the same epb_packetid value must be used. An example could be a router that captures packets on all its interfaces in both directions. When a packet hits interface A on ingress, an EPB entry gets created, TTL gets decremented, and right before it egresses on interface B another EPB entry gets created in the trace file. In this case, two packets are in the capture file, which are not identical but the epb_packetid can be used to correlate them.

Example: '0'.

epb_queue:

The epb_queue option is a 32-bit unsigned integer that identifies on which queue of the interface the specific packet was received.

Example: '0'.

epb_verdict:

The epb_verdict option stores a verdict of the packet. The verdict indicates what would be done with the packet after processing it. For example, a firewall could drop the packet. This verdict can be set by various components, i.e. Hardware, Linux's eBPF TC or XDP framework, etc. etc. The first octet specifies the verdict type, while the following octets contain the actual verdict data, whose size depends on the verdict type, and hence from the value in the first octet. The verdict type can be: Hardware (type octet = 0, size = variable), Linux_eBPF_TC (type octet = 1, size = 8 (64-bit unsigned integer), value = TC_ACT_* as defined in the Linux pck_cls.h include), Linux_eBPF_XDP (type octet = 2, size = 8 (64-bit unsigned integer), value = xdp_action as defined in the Linux pbf.h include).

Example: '02 00 00 00 00 00 00 00 02' for Linux_eBPF_XDP with verdict XDP_PASS.

4.3.1. Enhanced Packet Block Flags Word

The Enhanced Packet Block Flags Word is a 32-bit value that contains link-layer information about the packet.

The word is encoded as an unsigned 32-bit integer, using the endianness of the Section Header Block scope it is in. In the following table, the bits are numbered with 0 being the least-significant bit and 31 being the most-significant bit of the 32-bit unsigned integer. The meaning of the bits is the following:

Bit Number	Description
0-1	Inbound / Outbound packet (00 = information not available, 01 = inbound, 10 = outbound)
2-4	Reception type (000 = not specified, 001 = unicast, 010 = multicast, 011 = broadcast, 100 = promiscuous).
5-8	FCS length, in octets (0000 if this information is not available). This value overrides the if_fcslen option of the Interface Description Block, and is used with those link layers (e.g. PPP) where the length of the FCS can change during time.
9-15	Reserved (MUST be set to zero).
16-31	

Bit Number	Description
	link-layer-dependent errors (Bit 31 = symbol error, Bit 30 = preamble error, Bit 29 = Start Frame Delimiter error, Bit 28 = unaligned frame error, Bit 27 = wrong Inter Frame Gap error, Bit 26 = packet too short error, Bit 25 = packet too long error, Bit 24 = CRC error, other?? are 16 bit enough?).

Table 5

NOTE: in earlier versions of this specification, the bits were specified as being numbered with 0 being the most-significant bit and 31 being the least-significant bit of the 32-bit unsigned integer, rather than with 0 being the least-significant bit and 31 being the most-significant bit. Several implementations number the bits with 0 being the least-significant bit, and no known implementations number them with 0 being the most-significant bit, so the specification was changed to reflect that reality.

4.4. Simple Packet Block

The Simple Packet Block (SPB) is a lightweight container for storing the packets coming from the network. Its presence is optional.

A Simple Packet Block is similar to an Enhanced Packet Block (see Section 4.3), but it is smaller, simpler to process and contains only a minimal set of information. This block is preferred to the standard Enhanced Packet Block when performance or space occupation are critical factors, such as in sustained traffic capture applications. A capture file can contain both Enhanced Packet Blocks and Simple Packet Blocks: for example, a capture tool could switch from Enhanced Packet Blocks to Simple Packet Blocks when the hardware resources become critical.

The Simple Packet Block does not contain the Interface ID field. Therefore, it MUST be assumed that all the Simple Packet Blocks have been captured on the interface previously specified in the first Interface Description Block.

Figure 12 shows the format of the Simple Packet Block.

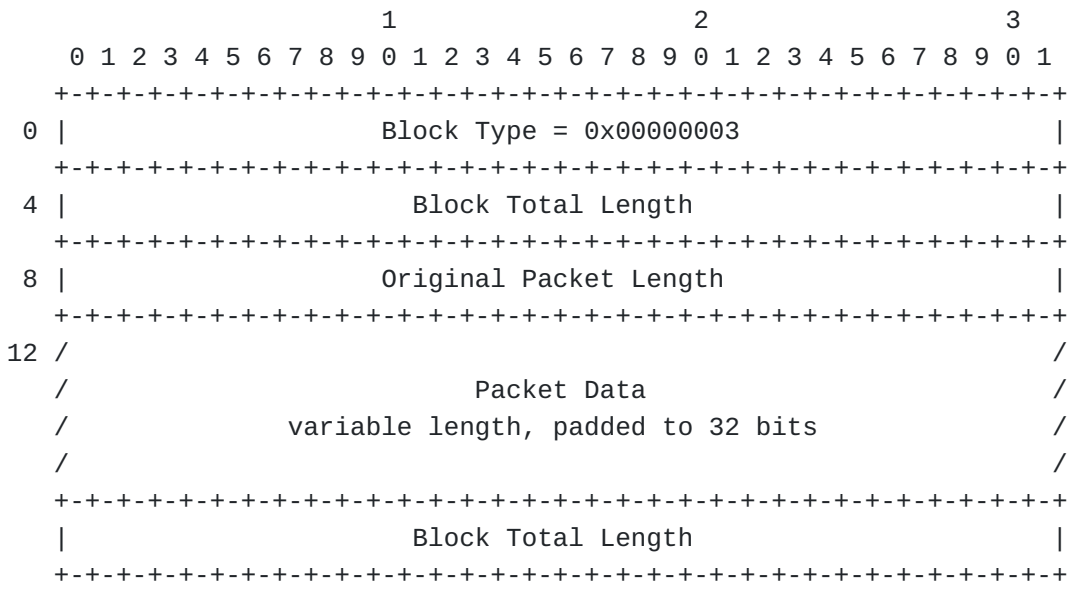


Figure 12: Simple Packet Block Format

The Simple Packet Block has the following fields:

- Block Type: The block type of the Simple Packet Block is 3.

- Block Total Length: total size of this block, as described in Section 3.1.

- Original Packet Length (32 bits): an unsigned value indicating the actual length of the packet when it was transmitted on the network. It can be different from length of the Packet Data field's length if the packet has been truncated by the capture process, in which case the SnapLen value in Section 4.2 will be less than this Original Packet Length value, and the SnapLen value MUST be used to determine the size of the Packet Data field length.

- Packet Data: the data coming from the network, including link-layer headers. The length of this field can be derived from the field Block Total Length, present in the Block Header, and it is the minimum value among the SnapLen (present in the Interface Description Block) and the Original Packet Length (present in this header). The format of the data within this Packet Data field depends on the LinkType field specified in the Interface Description Block (see Section 4.2) and it is specified in the entry for that format in [LINKTYPES].

The Simple Packet Block does not contain the timestamp because this is often one of the most costly operations on PCs. Additionally, there are applications that do not require it; e.g. an Intrusion Detection System is interested in packets, not in their timestamp.

A Simple Packet Block cannot be present in a Section that has more than one interface because of the impossibility to refer to the correct one (it does not contain any Interface ID field).

The Simple Packet Block is very efficient in term of disk space: a snapshot whose length is 100 octets requires only 16 octets of overhead, which corresponds to an efficiency of more than 86%.

4.5. Name Resolution Block

The Name Resolution Block (NRB) is used to support the correlation of numeric addresses (present in the captured packets) and their corresponding canonical names and it is optional. Having the literal names saved in the file prevents the need for performing name resolution at a later time, when the association between names and addresses may be different from the one in use at capture time. Moreover, the NRB avoids the need for issuing a lot of DNS requests every time the trace capture is opened, and also provides name resolution when reading the capture with a machine not connected to the network.

A Name Resolution Block is often placed at the beginning of the file, but no assumptions can be taken about its position. Multiple NRBs can exist in a pcapng file, either due to memory constraints or because additional name resolutions were performed by file processing tools, like network analyzers.

A Name Resolution Block need not contain any Records, except the nrb_record_end Record which MUST be the last Record. The addresses and names in NRB Records MAY be repeated multiple times; i.e., the same IP address may resolve to multiple names, the same name may resolve to the multiple IP addresses, and even the same address-to-name pair may appear multiple times, in the same NRB or across NRBs.

The format of the Name Resolution Block is shown in Figure 13.

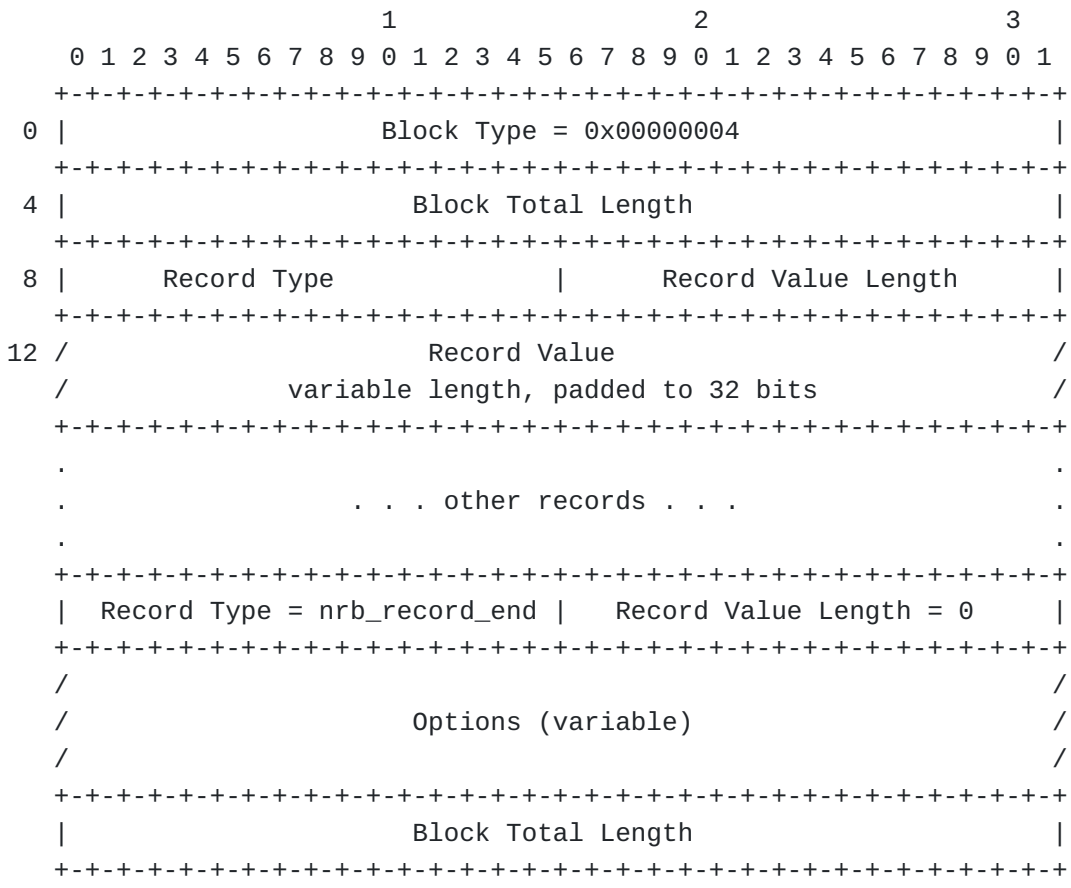


Figure 13: Name Resolution Block Format

The Name Resolution Block has the following fields:

*Block Type: The block type of the Name Resolution Block is 4.

*Block Total Length: total size of this block, as described in Section 3.1.

This is followed by zero or more Name Resolution Records (in the TLV format), each of which contains an association between a network address and a name. An nrb_record_end MUST be added after the last Record, and MUST exist even if there are no other Records in the NRB. There are currently three possible types of records:

Name	Code	Length
nrb_record_end	0x0000	0
nrb_record_ipv4	0x0001	variable
nrb_record_ipv6	0x0002	variable

Table 6: Name Resolution Block Records

nrb_record_end:

The nrb_record_end record delimits the end of name resolution records. This record is needed to determine when the list of name resolution records has ended and some options (if any) begin.

nrb_record_ipv4:
The nrb_record_ipv4 record specifies an IPv4 address (contained in the first 4 octets), followed by one or more zero-terminated UTF-8 strings containing the DNS entries for that address. The minimum valid Record Length for this Record Type is thus 6: 4 for the IP octets, 1 character, and a zero-value octet terminator. Note that the IP address is treated as four octets, one for each octet of the IP address; it is not a 32-bit word, and thus the endianness of the SHB does not affect this field's value.

Example: '127 0 0 1'"localhost".

[Open issue: is an empty string (i.e., just a zero-value octet) valid?]

nrb_record_ipv6:
The nrb_record_ipv6 record specifies an IPv6 address (contained in the first 16 octets), followed by one or more zero-terminated strings containing the DNS entries for that address. The minimum valid Record Length for this Record Type is thus 18: 16 for the IP octets, 1 character, and a zero-value octet terminator.

Example: '20 01 0d b8 00 00 00 00 00 00 00 00 12 34 56 78'"somehost".

[Open issue: is an empty string (i.e., just a zero-value octet) valid?]

Record Types other than those specified earlier MUST be ignored and skipped past. More Record Types will likely be defined in the future, and MUST NOT break backwards compatibility.

Each Record Value is aligned to and padded to a 32-bit boundary. The corresponding Record Value Length reflects the actual length of the Record Value; it does not include the lengths of the Record Type field, the Record Value Length field, any padding for the Record Value, or anything after the Record Value. For Record Types with name strings, the Record Length does include the zero-value octet terminating that string. A Record Length of 0 is valid, unless indicated otherwise.

After the list of Name Resolution Records, optionally, a list of options (formatted according to the rules defined in Section 3.5) can be present.

In addition to the options defined in Section 3.5, the following options are valid within this block:

Name	Code	Length	Multiple allowed?
ns_dnsname	2	variable	no
ns_dnsIP4addr	3	4	no
ns_dnsIP6addr	4	16	no

Table 7: Name Resolution Block Options

ns_dnsname:

The ns_dnsname option is a UTF-8 string containing the name of the machine (DNS server) used to perform the name resolution. The string is not zero-terminated.

Example: "our_nameserver".

ns_dnsIP4addr:

The ns_dnsIP4addr option specifies the IPv4 address of the DNS server. Note that the IP address is treated as four octets, one for each octet of the IP address; it is not a 32-bit word, and thus the endianness of the SHB does not affect this field's value.

Example: '192 168 0 1'.

ns_dnsIP6addr:

The ns_dnsIP6addr option specifies the IPv6 address of the DNS server.

Example: '20 01 0d b8 00 00 00 00 00 00 00 00 12 34 56 78'.

4.6. Interface Statistics Block

The Interface Statistics Block (ISB) contains the capture statistics for a given interface and it is optional. The statistics are referred to the interface defined in the current Section identified by the Interface ID field. An Interface Statistics Block is normally placed at the end of the file, but no assumptions can be taken about its position - it can even appear multiple times for the same interface.

The format of the Interface Statistics Block is shown in Figure 14.

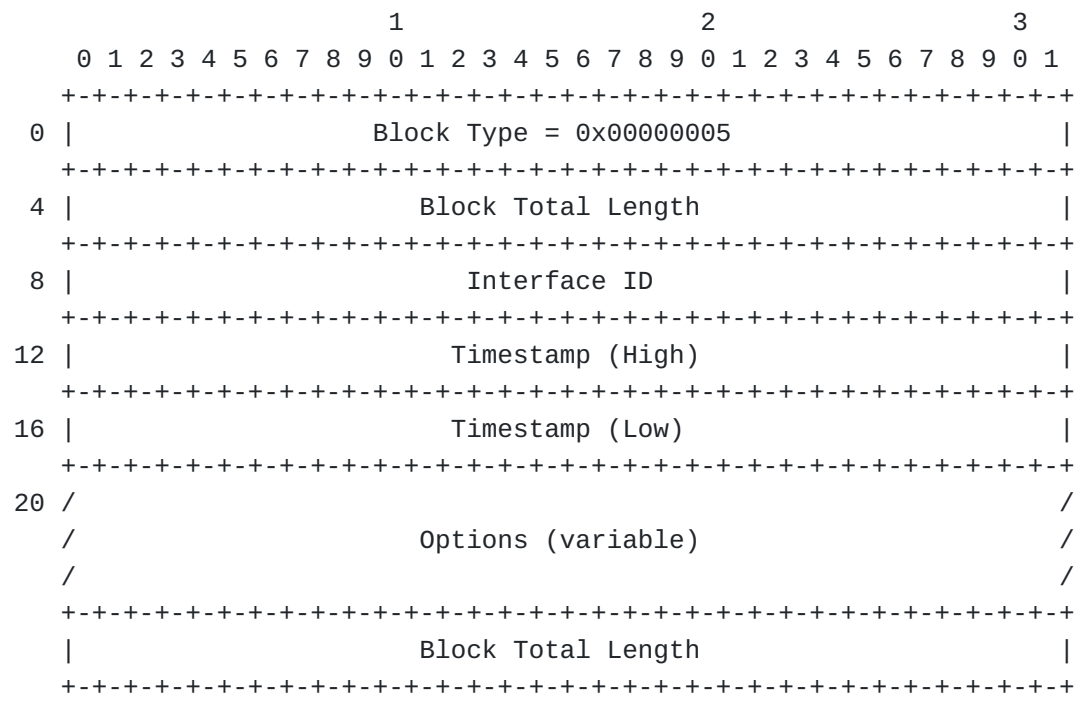


Figure 14: Interface Statistics Block Format

The fields have the following meaning:

- Block Type: The block type of the Interface Statistics Block is 5.

- Block Total Length: total size of this block, as described in Section 3.1.

- Interface ID: specifies the interface these statistics refers to; the correct interface will be the one whose Interface Description Block (within the current Section of the file) is identified by same number (see Section 4.2) of this field.

- Timestamp: time this statistics refers to. The format of the timestamp is the same already defined in the Enhanced Packet Block (Section 4.3); the length of a unit of time is specified by the 'if_tsresol' option (see Figure 10) of the Interface Description Block referenced by this packet.

- Options: optionally, a list of options (formatted according to the rules defined in Section 3.5) can be present.

All the statistic fields are defined as options in order to deal with systems that do not have a complete set of statistics. Therefore, In addition to the options defined in Section 3.5, the following options are valid within this block:

Name	Code	Length	Multiple allowed?
isb_starttime	2	8	no
isb_endtime	3	8	no
isb_ifrecv	4	8	no
isb_ifdrop	5	8	no
isb_filteraccept	6	8	no
isb_osdrop	7	8	no
isb_usrdeliv	8	8	no

Table 8: Interface Statistics Block Options

isb_starttime:

The isb_starttime option specifies the time the capture started; time will be stored in two blocks of four octets each. The format of the timestamp is the same as the one defined in the Enhanced Packet Block (Section 4.3); the length of a unit of time is specified by the 'if_tsresol' option (see Figure 10) of the Interface Description Block referenced by this packet.

Example: '96 c3 04 00 73 89 6a 65', in Little Endian, decodes to 2012-06-29 06:17:00.834163 UTC.

isb_endtime:

The isb_endtime option specifies the time the capture ended; time will be stored in two blocks of four octets each. The format of the timestamp is the same as the one defined in the Enhanced Packet Block (Section 4.3); the length of a unit of time is specified by the 'if_tsresol' option (see Figure 10) of the Interface Description Block referenced by this packet.

Example: '97 c3 04 00 aa 47 ca 64', in Little Endian, decodes to 2012-06-29 07:28:25.298858 UTC.

isb_ifrecv:

The isb_ifrecv option specifies the 64-bit unsigned integer number of packets received from the physical interface starting from the beginning of the capture.

Example: the decimal number 100.

isb_ifdrop:

The isb_ifdrop option specifies the 64-bit unsigned integer number of packets dropped by the interface due to lack of resources starting from the beginning of the capture.

Example: '0'.

isb_filteraccept:

The isb_filteraccept option specifies the 64-bit unsigned integer number of packets accepted by filter starting from the beginning of the capture.

Example: the decimal number 100.

isb_osdrop:

The isb_osdrop option specifies the 64-bit unsigned integer number of packets dropped by the operating system starting from the beginning of the capture.

Example: '0'.

isb_usrdeliv:

The isb_usrdeliv option specifies the 64-bit unsigned integer number of packets delivered to the user starting from the beginning of the capture. The value contained in this field can be different from the value 'isb_filteraccept - isb_osdrop' because some packets could still be in the OS buffers when the capture ended.

Example: '0'.

All the fields that refer to packet counters are 64-bit values, represented with the octet order of the current section. Special care must be taken in accessing these fields: since all the blocks are aligned to a 32-bit boundary, such fields are not guaranteed to be aligned on a 64-bit boundary.

4.7. Decryption Secrets Block

A Decryption Secrets Block (DSB) stores (session) secrets that enable decryption of packets within the capture file. The format of these secrets is defined by the Secrets Type.

Multiple DSBs can exist in a pcapng file, but they SHOULD be written before packet blocks that require those secrets. Tools MAY limit decryption to secrets that appear before packet blocks.

The structure of a Decryption Secrets Block is shown in Figure 15.

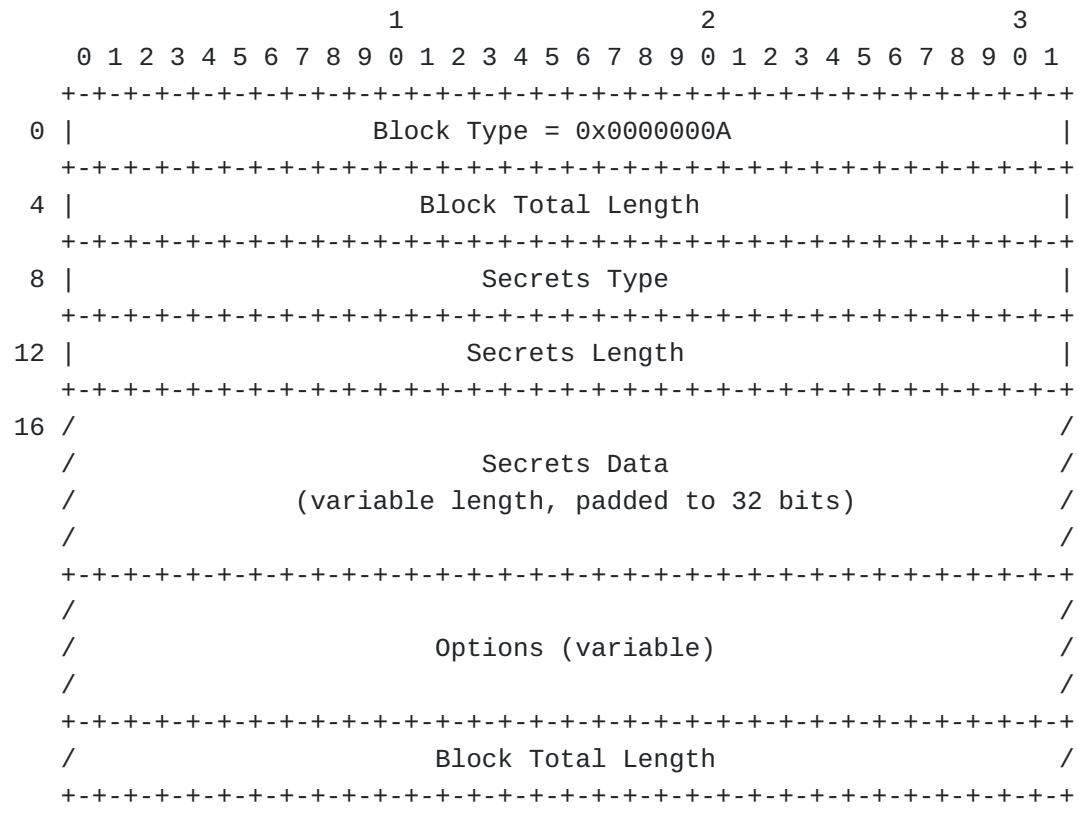


Figure 15: Decryption Secrets Block Format

The Decryption Secrets Block has the following fields.

*Block Type: The block type of the Decryption Secrets Block is 10.

*Block Total Length: total size of this block, as described in Section 3.1.

*Secrets Type (32 bits): an unsigned integer identifier that describes the format of the following Secrets field. Requests for new Secrets Type codes should be made by creating a pull request to update this document as described in Section 10.1.

*Secrets Length (32 bits): an unsigned integer that indicates the size of the following Secrets field, without any padding octets.

*Secrets Data: binary data containing secrets, padded to a 32 bit boundary.

*Options: optionally, a list of options (formatted according to the rules defined in Section 3.5) can be present. No DSB-specific options are currently defined.

The following is a list of Secrets Types.

0x544c534b:

TLS Key Log. This format is described at NSS Key Log Format. Every line MUST be properly terminated with either carriage return and linefeed ('\r\n') or linefeed ('\n'). Tools MUST be able to handle both line endings.

0x57474b4c:
WireGuard Key Log. Every line consists of the key type, equals sign ('='), and the base64-encoded 32-byte key with optional spaces before and in between. The key type is one of LOCAL_STATIC_PRIVATE_KEY, REMOTE_STATIC_PUBLIC_KEY, LOCAL_EPHEMERAL_PRIVATE_KEY, or PRESHARED|_KEY. This matches the output of extract-handshakes.sh, which is part of the WireGuard project. A PRESHARED_KEY line is linked to a session matched by a previous LOCAL_EPHEMERAL_PRIVATE_KEY line. Every line MUST be properly terminated with either carriage return and linefeed ('\r\n') or linefeed ('\n'). Tools MUST be able to handle both line endings.

Warning: LOCAL_STATIC_PRIVATE_KEY and potentially PRESHARED_KEY are long-term secrets, users SHOULD only store non-production keys, or ensure proper protection of the pcapng file.

0x5a4e574b:
ZigBee NWK Key and ZigBee PANID for that network. Network Key as described in the ZigBee Specification 05-3473-21 (R21) section 4.2.2. The NWK Key is a 16 octet binary AES-128 key used to secure NWK Level frames within a single PAN. The NWK key is immediately followed by the 2 octet (16 bit) network PANID in little endian format. If and when the NWK Key changes a new DSB will contain the new NWK Key.

0x5a415053:
ZigBee APS Key. Application Support Link Key as described in the ZigBee Specification 05-3473-21 (R21) section 4.4. Each 16 octet binary AES-128 key secures frames exchanged between a pair of network nodes. The APS Key is immediately followed by the 2 octet (16 bit) network PANID in little endian format. The PANID is followed by the 2 octet (16 bit) short addresses, in little endian format, of the nodes to which the APS Key applies. The numerically lower short address shall come first. There is a APS Key DSB for each node pair for which the Link Key is known. As new links are formed, new DSBs contain the new Keys. If the APS Key changes for an existing link, it is contained in a new DSB with the new APS Key.

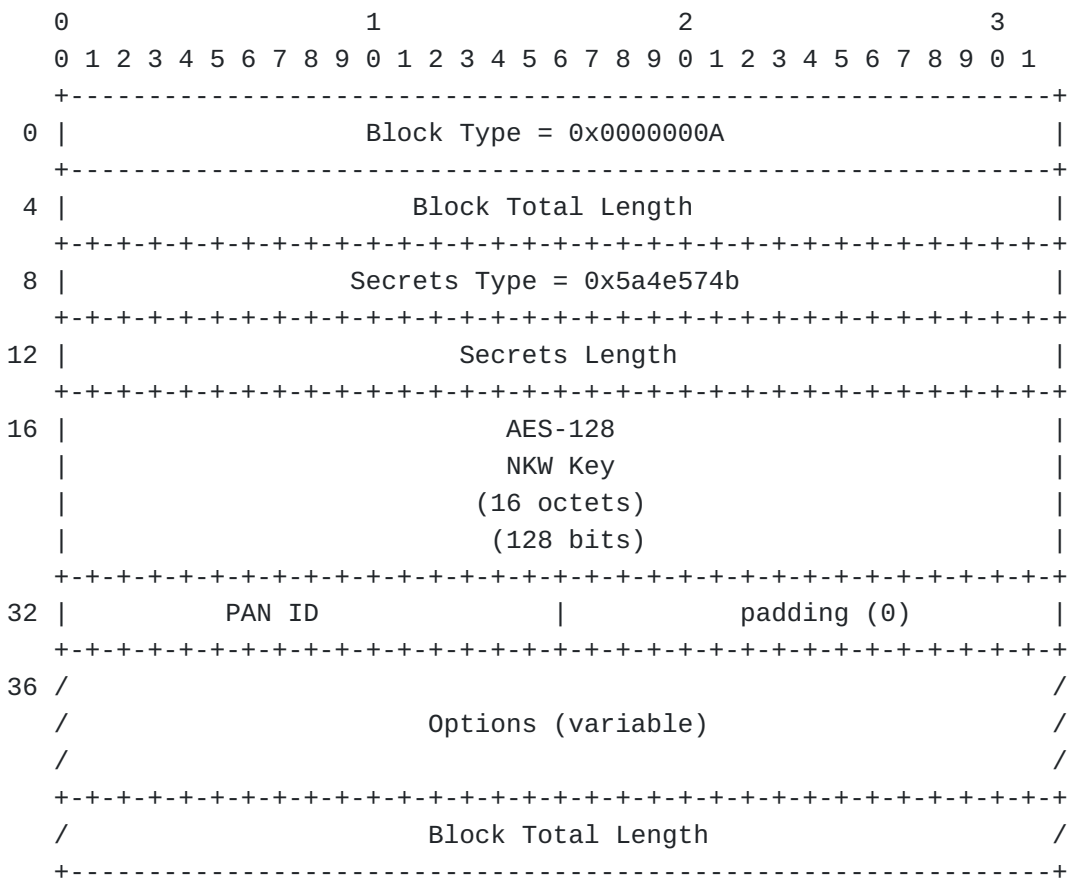


Figure 16: ZigBee NWK Key Data Format

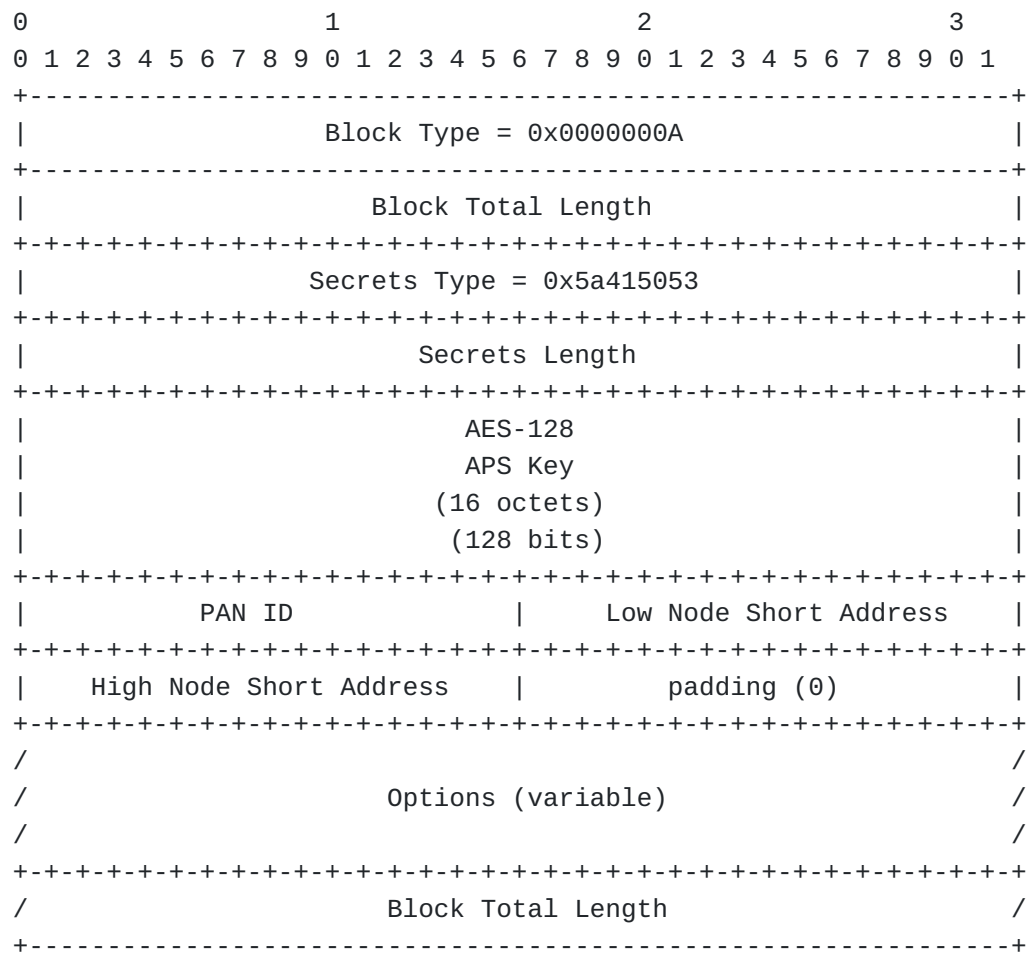


Figure 17: ZigBee APS Key Data Format

4.8. Custom Block

A Custom Block (CB) is the container for storing custom data that is not part of another block; for storing custom data as part of another block, see Section 3.5.1. The Custom Block is optional, can be repeated any number of times, and can appear before or after any other block except the first Section Header Block which must come first in the file. Different Custom Blocks, of different type codes and/or different Private Enterprise Numbers, may be used in the same pcapng file. The format of a Custom Block is shown in Figure 18.

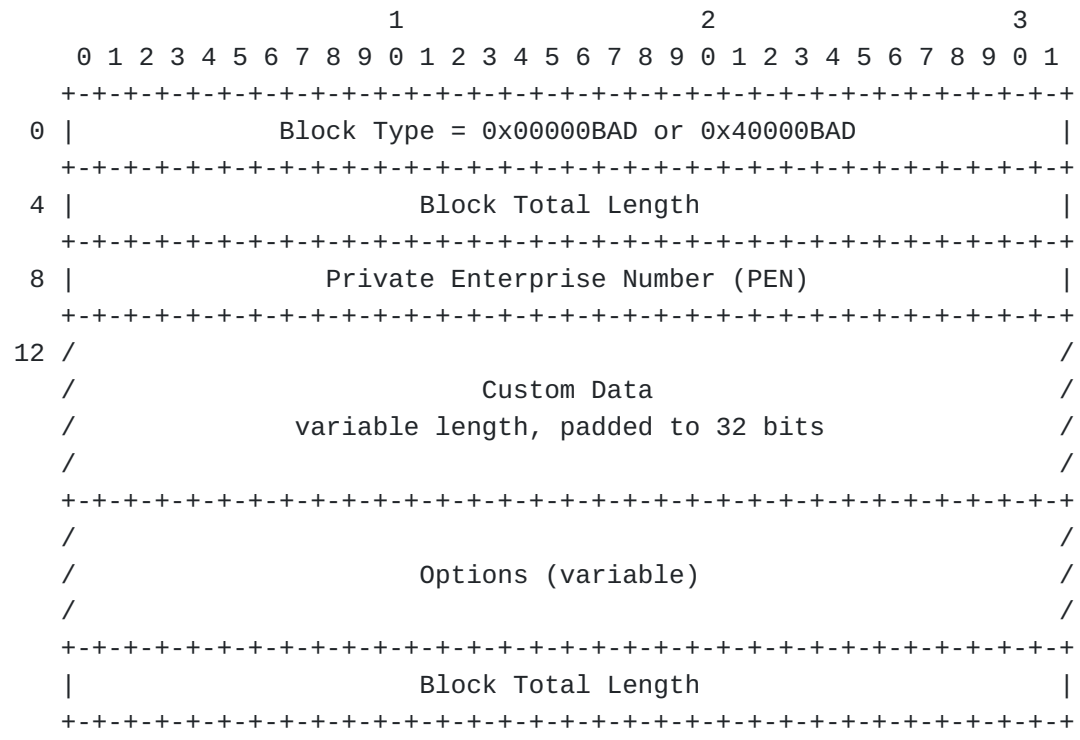


Figure 18: Custom Block Format

The Custom Block uses the type code 0x00000BAD (2989 in decimal) for a custom block that pcapng re-writers can copy into new files, and the type code 0x40000BAD (1073744813 in decimal) for one that should not be copied. See Section 5.2 for details.

The Custom Block has the following fields:

*Block Type: The block type of the Custom Block is 0x00000BAD or 0x40000BAD, as described previously.

*Block Total Length: total size of this block, as described in Section 3.1.

*Private Enterprise Number (32 bits): An IANA-assigned Private Enterprise Number identifying the organization which defined the Custom Block. See Section 5.1 for details. The PEN MUST be encoded using the same endianness as the Section Header Block it is within the scope of.

*Custom Data: the custom data, padded to a 32 bit boundary.

*Options: optionally, a list of options (formatted according to the rules defined in Section 3.5) can be present. Note that custom options for the Custom Block still use the custom option format and type code, as described in Section 3.5.1.

5. Vendor-Specific Custom Extensions

This section uses the term "vendor" to describe an organization which extends the pcapng file with custom, proprietary blocks or options. It should be noted, however, that the "vendor" is just an abstract entity that agrees on a custom extension format: for example it may be a manufacturer, industry association, an individual user, or collective group of users.

5.1. Supported Use-Cases

There are two different supported use-cases for vendor-specific custom extensions: local and portable. Local use means the custom data is only expected to be usable on the same machine, and the same application, which encoded it into the file. This limitation is due to the lack of a common registry for the local use number codes (the block or option type code numbers with the Most Significant Bit set). Since two different vendors may choose the same number, one vendor's application reading the other vendor's file would result in decoding failure. Therefore, vendors SHOULD instead use the portable method, as described next.

The portable use-case supports vendor-specific custom extensions in pcapng files which can be shared across systems, organizations, etc. To avoid number space collisions, an IANA-registered Private Enterprise Number (PEN) is encoded into the Custom Block or Custom Option, using the PEN that belongs to the vendor defining the extension. Anyone can register a new PEN with IANA, for free, by filling out the online request form at http://pen.iana.org/pen/PenApplication.page.

5.2. Controlling Copy Behavior

Both Custom Blocks and Custom Options support two different codes to distinguish their "copy" behavior: a code for when the block or option can be safely copied into a new pcapng file by a pcapng manipulating application, and a code for when it should not be copied. A common reason for not copying a Custom Block or Custom Option is because it depends on other blocks or options in some way that would invalidate the custom data if the other blocks/options were removed or re-ordered. For example, if a Custom Block's data includes an Interface ID number in its Custom Data portion, then it cannot be safely copied by a pcapng application that merges pcapng files, because the merging application might re-order or remove one or more of the Interface Description Blocks, and thereby change the Interface IDs that the Custom Block depends upon. The same issue arises if a Custom Block or Custom Option depends on the presence of, or specific ordering of, other standard-based or custom-defined blocks or options.

Note that the copy semantics is not related to privacy - there is no guarantee that a pcapng anonymizer will remove a Custom Block or Custom Option, even if the appropriate code is used requesting it not be copied; and the original pcapng file can be shared anyway. If the Custom Data portion of the Custom Block or Custom Option contains sensitive information, then it should be encrypted in some fashion.

5.3. Strings vs. Octets

For the Custom Options, there are two Custom Data formats supported: a UTF-8 string and a binary data payload. The rationale for this separation is that a pcapng display application which does not understand the specific PEN's Custom Option can still display the data as a string if it's a string type code, rather than as hex-ascii of the octets.

5.4. Endianness Issues

Implementers writing Custom Blocks or binary data Custom Options should be aware that a pcapng file can be re-written by machines using a different endianness than the original file, which means all known fields of the pcapng file will change endianness in the new file. Since the Custom Data payload of the Custom Block or the binary data Custom Option might be an arbitrary sequence of unknown octets to such machines, they cannot convert multi-octet values inside the Custom Data, or in the Options section of a Custom Block,into the appropriate endianness.

For example, a little-endian machine can create a new pcapng file and add some binary data Custom Options to some non-Custom Block(s) in the file. This file can then be sent to a big-endian host, which will convert the Option Code, Option Length, and PEN fields of the options to big-endian format if it re-writes the file. However, if the software reading the file does not understand the contents of all of the Custom Options, it will leave the Custom Data payload of the options alone (as little-endian format). If this file then gets sent to a little-endian machine, then, when that little-endian machine reads the file, it will, if the software reading the file understands the contents of all the Custom Options, it will detect that the file format is big-endian, and swap the endianness while it parses the file - but that will cause the Custom Data payload to be incorrect since it was already in little-endian format.

In addition, a little-endian machine can create a pcapng file and write some binary data Custom Blocks, containing options, to the file. The file can then be sent to a big-endian host, which, if the software reading the file does not understand the contents of the Custom Blocks, will leave the Custom Data and Options alone (as

little-endina format). If this file then gets sent to a little-endian machine, then, when that little-endian machine reads the file, it will, if the software reading the file understands the contents of all the Custom Blocks, it will detect that the file format is big-endian, and swap the endianness while it parses the file - but that will cause the Custom Data payload, the Option Code and Option Length values in the Options, and the PEN in any Custom Options to be incorrect since they were already in little-endian format.

Therefore, the vendor should either encode the Custom Data of their Custom Blocks and Custom Options, the Option Code and Option Length fields of options in Custom Blocks, and the PEN field of Custom Options in Custom Blocks in a consistent manner, such as always in big-endian or always in little-endian format, regardless of the host platform's endianness, or should encode some flag in the Custom Data payload to indicate in which endianness the rest of the payload is written.

The PEN field of a Custom Block, or of a Custom Option not contained in a Custom Block, MUST be converted by code that reads pcapng files, so this is not an issue for that field, except for Custom Options in Custom Blocks. This is also not an issue for the Custom Data payload of UTF-8 string Custom Options.

6. Recommended File Name Extension: .pcapng

The recommended file name extension for the "PCAP Next Generation Capture File Format" specified in this document is ".pcapng".

On Windows and macOS, files are distinguished by an extension to their filename. Such an extension is technically not actually required, as applications should be able to automatically detect the pcapng file format through the "magic bytes" at the beginning of the file, as some other UN*X desktop environments do. However, using name extensions makes it easier to work with files (e.g. visually distinguish file formats) so it is recommended - though not required - to use .pcapng as the name extension for files following this specification.

Please note: To avoid confusion (such as the current usage of .cap for a plethora of different capture file formats) file name extensions other than .pcapng should be avoided.

7. Conclusions

The file format proposed in this document should be very versatile and satisfy a wide range of applications. In the simplest case, it can contain a raw capture of the network data, made of a series of Simple Packet Blocks. In the most complex case, it can be used as a

repository for heterogeneous information. In every case, the file remains easy to parse and an application can always skip the data it is not interested in; at the same time, different applications can share the file, and each of them can benefit of the information produced by the others. Two or more files can be concatenated obtaining another valid file.

8. Implementations

Some known implementations that read or write the pcapng file format are listed on the pcapng GitHub wiki.

9. Security Considerations

TBD.

10. IANA Considerations

TBD.

[Open issue: decide whether the block types, option types, NRB Record types, etc. should be IANA registries. And if so, what the IANA policy for each should be (see RFC 5226)]

10.1. Standardized Block Type Codes

Every Block is uniquely identified by a 32-bit integer value, stored in the Block Header.

As pointed out in Section 3.1, Block Type codes whose Most Significant Bit (bit 31) is set to 1 are reserved for local use by the application.

All the remaining Block Type codes (0x00000000 to 0x7FFFFFFF) are standardized by this document. Requests for new Block Type codes, Option Type codes, and Secrets Type codes should be made by creating a pull request to update this document at github.com/pcapng/pcapng. The pull request should add a description of the new block, option, or secret type to Section 4. The pull request description should contain a clear request for a new type code assignment.

The following is a list of the Standardized Block Type Codes:

Block Type Code	Description
0x00000000	Reserved ???
0x00000001	Interface Description Block (Section 4.2)
0x00000002	Packet Block (Appendix A)
0x00000003	Simple Packet Block (Section 4.4)
0x00000004	Name Resolution Block (Section 4.5)

Block Type Code	Description
0x00000005	Interface Statistics Block (Section 4.6)
0x00000006	Enhanced Packet Block (Section 4.3)
0x00000007	IRIG Timestamp Block (requested by Gianluca Varenni <gianluca.varenni@cacetech.com>, CACE Technologies LLC); code also used for Socket Aggregation Event Block
0x00000008	ARINC 429 in AFDX Encapsulation Information Block (requested by Gianluca Varenni <gianluca.varenni@cacetech.com>, CACE Technologies LLC)
0x00000009	[systemd Journal Export Block][I-D.richardson-opsawg-pcapng-extras]
0x0000000A	Decryption Secrets Block (Section 4.7)
0x00000101	Hone Project Machine Info Block (see also Google version)
0x00000102	Hone Project Connection Event Block (see also Google version)
0x00000201	Sysdig Machine Info Block
0x00000202	Sysdig Process Info Block, version 1
0x00000203	Sysdig FD List Block
0x00000204	Sysdig Event Block
0x00000205	Sysdig Interface List Block
0x00000206	Sysdig User List Block
0x00000207	Sysdig Process Info Block, version 2
0x00000208	Sysdig Event Block with flags
0x00000209	Sysdig Process Info Block, version 3
0x00000210	Sysdig Process Info Block, version 4
0x00000211	Sysdig Process Info Block, version 5
0x00000212	Sysdig Process Info Block, version 6
0x00000213	Sysdig Process Info Block, version 7
0x00000BAD	Custom Block that rewriters can copy into new files (Section 4.8)
0x40000BAD	Custom Block that rewriters should not copy into new files (Section 4.8)
0x0A0D0D0A	Section Header Block (Section 4.1)
0x0A0D0A00-0x0A0D0AFF	Reserved. Used to detect trace files corrupted because of file transfers using the HTTP protocol in text mode.
0x000A0D0A-0xFF0A0D0A	Reserved. Used to detect trace files corrupted because of file transfers using the HTTP protocol in text mode.
0x000A0D0D-0xFF0A0D0D	Reserved. Used to detect trace files corrupted because of file transfers using the HTTP protocol in text mode.
0x0D0D0A00-0x0D0D0AFF	

Block Type Code	Description
	Reserved. Used to detect trace files corrupted because of file transfers using the FTP protocol in text mode.
0x80000000-0xFFFFFFFF	Reserved for local use.

Table 9: Standardized Block Type Codes

[Open issue: reserve 0x40000000-0x7FFFFFFF for do-not-copy-bit range of base types?]

11. Contributors

Loris Degioanni and Gianluca Varenni were coauthoring this document before it was submitted to the IETF.

12. Acknowledgments

The authors wish to thank Anders Broman, Ulf Lamping, Richard Sharpe and many others for their invaluable comments.

13. References

13.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

13.2. Informative References

[I-D.richardson-opsawg-pcapng-extras] "*** BROKEN REFERENCE ***".

[LINKTYPES] The Tcpdump Group, "the tcpdump.org link-layer header types registry", <http://www.tcpdump.org/linktypes.html>.

Appendix A. Packet Block (obsolete!)

The Packet Block is obsolete, and MUST NOT be used in new files. Use the Enhanced Packet Block or Simple Packet Block instead. This section is for historical reference only.

A Packet Block was a container for storing packets coming from the network.

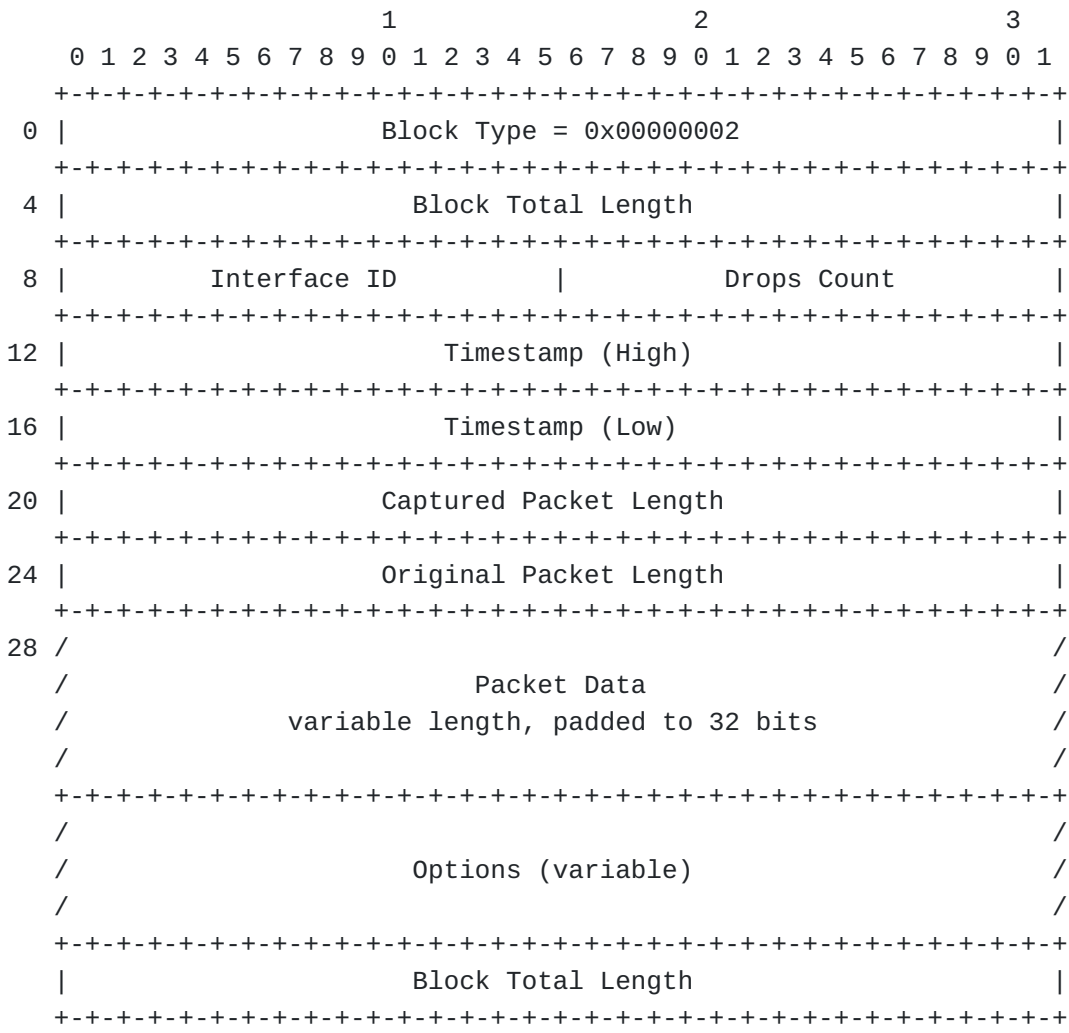


Figure 19: Packet Block Format

The Packet Block has the following fields:

*Block Type: The block type of the Packet Block is 2.

*Block Total Length: total size of this block, as described in Section 3.1.

*Interface ID: specifies the interface this packet comes from; the correct interface will be the one whose Interface Description Block (within the current Section of the file) is identified by the same number (see Section 4.2) of this field. The interface ID MUST be valid, which means that an matching interface description block MUST exist.

*Drops Count: a local drop counter. It specifies the number of packets lost (by the interface and the operating system) between this packet and the preceding one. The value xFFFF (in

hexadecimal) is reserved for those systems in which this information is not available.

*Timestamp (High) and Timestamp (Low): timestamp of the packet. The format of the timestamp is the same as was already defined for the Enhanced Packet Block (Section 4.3).

*Captured Packet Length: number of octets captured from the packet (i.e. the length of the Packet Data field). It will be the minimum value among the Original Packet Length and the snapshot length for the interface (SnapLen, defined in Figure 10). The value of this field does not include the padding octets added at the end of the Packet Data field to align the Packet Data field to a 32-bit boundary.

*Original Packet Length: actual length of the packet when it was transmitted on the network. It can be different from Captured Packet Length if the packet has been truncated by the capture process.

*Packet Data: the data coming from the network, including link-layer headers. The actual length of this field is Captured Packet Length plus the padding to a 32-bit boundary. The format of the link-layer headers depends on the LinkType field specified in the Interface Description Block (see Section 4.2) and it is specified in the entry for that format in [LINKTYPES].

*Options: optionally, a list of options (formatted according to the rules defined in Section 3.5) can be present.

In addition to the options defined in Section 3.5, the following options were valid within this block:

Name	Code	Length	Multiple allowed?
pack_flags	2	4	no
pack_hash	3	variable	yes

Table 10: Packet Block Options

pack_flags:
The pack_flags option is the same as the epb_flags of the enhanced packet block.

Example: '0'.

pack_hash:
The pack_hash option is the same as the epb_hash of the enhanced packet block.

Examples: '02 EC 1D 87 97', '03 45 6E C2 17 7C 10 1E 3C 2E 99 6E C2 9A 3D 50 8E'.

Authors' Addresses

Michael Tuexen (editor)
Muenster University of Applied Sciences
Stegerwaldstrasse 39
48565 Steinfurt
Germany

Email: tuexen@fh-muenster.de

Fulvio Risso
Politecnico di Torino
Corso Duca degli Abruzzi, 24
10129 Torino
Italy

Email: fulvio.risso@polito.it

Jasper Bongertz
Airbus Defence and Space CyberSecurity
Kanzlei 63c
40667 Meerbusch
Germany

Email: jasper@packet-foo.com

Gerald Combs
Wireshark Foundation
339 Madson Pl
Davis, CA 95618
United States of America

Email: gerald@wireshark.org

Guy Harris

Email: gharris@sonic.net

Eelco Chaudron
Red Hat
De Entree 238
1101 EE Amsterdam
Netherlands

Email: eelco@redhat.com

Michael C. Richardson
Sandelman Software Works

Email: mcr+ietf@sandelman.ca

URI: http://www.sandelman.ca/