Network Working Group
Internet-Draft
Expires: September 2, 2004

L. Degioanni
CACE Technologies
F. Risso
Politecnico di Torino
G. Varenni
CACE Technologies
March 2004

TOC

# PCAP Next Generation Dump File Format
# PCAP-DumpFileFormat

## Status of this Memo

## Copyright Notice

## Abstract

This document describes a format to dump captured packets on a file. This format is extensible and it is currently proposed for implementation in the libpcap/WinPcap packet capture library.

## Updates

- [27 Jul 2009] Guy Harris: added some missing reserved block types in Appendix B.
- [27 Jul 2009] Guy Harris: fixed a typo in Appendix B. The range of standardized blocks are in the range 0x00000000-0x7FFFFFFF.
- [ 8 Feb 2008] Gianluca Varenni: better documentation for the format of the timestamps. Renamed the if_tsaccur option into if_tsresol.
- [22 Oct 2007] Gianluca Varenni: added a note related to 64-bit alignment. Specified that the option length field is the length without padding. typos here and there. Added some option examples.
- [17 Oct 2007] Ulf Lamping: Major review: "Interface ID" in "ISB" now 32 bits. isb_starttime/isb_endtime depends on if_tsaccur. Lot's of other editing ...
- [ 8 Oct 2007] Ulf Lamping: Fixed several typos. Grouped the block types into mandatory, optional, experimental, obsolete.
- [14 Sep 2006] Gianluca Varenni: Added the block type code for Arinc 429 in AFDX Encapsulation Information Block
- [23 May 2006] Gianluca Varenni: Added the block type code for IRIG Timestamp Block
- [23 Apr 2006] Gianluca Varenni: Cleaned up Appendix C a bit: we should use the LINKTYPE_xxx values from libpcap, not the DLT_xxx ones. Fixed the introduction to the appendix and added some comments.
- [21 Mar 2006] Gianluca Varenni: Added a preliminary version of Appendix C, detailing the

Standardized Link Types.
- [21 Mar 2006] Gianluca Varenni: Added a preliminary version of Appendix B, detailing the Standardized Block Type codes.
- [21 Mar 2006] Gianluca Varenni: Added the Enhanced Packet Block in section 2.2. Fixed a typo in the list: it's Interface Statistics Block, and not Capture Statistics Block.
- [21 Mar 2006] Gianluca Varenni: Fixed some minor typos in the document.
- [21 Mar 2006] Gianluca Varenni: Fixed an error in Packet Block: option pack_hash should have code 3.
- [21 Mar 2006] Gianluca Varenni: Added the definition of the Enhanced Packet Block.
- [12 Mar 2006] Gianluca Varenni: Added option if_tsoffset in the Interface Description Block.

## Table of Contents

## 1. Objectives

TOC

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 does not fit for some of the nowadays applications particularly from the extensibility point of view.

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

- Extensibility: aside of some common functionalities, third parties should be able to enrich the information embedded in the file with proprietary extensions, which will be ignored by tools that are not able to understand 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. General File Structure

TOC

### 2.1. General Block Structure

TOC

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**.

```
     0                   1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                          Block Type                           |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                      Block Total Length                       |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    /                          Block Body                           /
    /          /* variable length, aligned to 32 bits */            /
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                      Block Total Length                       |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

**Figure 1: Basic block structure.**

The fields have the following meaning:

- Block Type (32 bits): unique value that identifies the block. Values whose Most Significant Bit (MSB) is equal to 1 are reserved for local use. They allow to save private data to the file and to extend the file format. The list of currently defined types can be found in **Appendix B**
- Block Total Length: total size of this block, in bytes. For instance, the length of a block that does not have body is 12 bytes.
- Block Body: content of the block.
- Block Total Length: total size of this block, in bytes. This field is duplicated for permitting 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 dump 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 non understand them can simply ignore their content.

### 2.2. Block Types

TOC

The currently standardized Block Type codes are specified in **Appendix B**, they have been grouped in the following four categories:

MANDATORY blocks must appear at least once in each file:

- **Section Header Block**: it defines the most important characteristics of the capture file.
- **Interface Description Block**: it defines the most important characteristics of the interface(s)

used for capturing traffic.

OPTIONAL blocks can appear in a file:

- **Enhanced Packet Block**: it contains a single captured packet, or a portion of it. It represents an evolution of the original **Packet Block**.
- **Simple Packet Block**: it contains a single captured packet, or a portion of it, with only a minimal set of information about it.
- **Name Resolution Block**: it defines the mapping from numeric addresses present in the packet dump and the canonical name counterpart.
- **Interface Statistics Block**: it defines how to store some statistical data (e.g. packet dropped, etc) which can be useful to undestand the conditions in which the capture has been made.

OBSOLETE blocks should not appear in newly written files (but left here for reference):

- **Packet Block**: it contains a single captured packet, or a portion of it. It should be considered OBSOLETE, and superseded by the **Enhanced Packet Block**.

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

## 2.3. Logical Block Hierarchy

TOC

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":

```
Section Header
|
+- Interface Description
|  +- Simple Packet
|  +- Enhanced Packet
|  +- Interface Statistics
|
+- Name Resolution
```

**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.

## 2.4. Physical File Layout

TOC

The file must begin with a Section Header Block. However, more than one Section Header Block can be present on the dump, each one covering the data following it till 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. This is a mandatory requirement that must be maintained in future versions of the block format.

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

```
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | SHB v1.0  |                      Data                           |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

**Figure 3: File structure example: Typical configuration 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.

```
   |--   1st Section   --|--   2nd Section   --|--  3rd Section  --|
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | SHB v1.0  |  Data   | SHB V1.1  |  Data   | SHB V1.0  |  Data |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

**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).

```
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | SHB | IDB | EPB | EPB |    ...    | 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, and also includes some Name Resolution Blocks (NRB) and an Interface Statistics Block (ISB).

```
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | SHB | IDB | IDB | IDB | EPB | EPB | NRB |    ...    | EPB | ISB | NRB | EPB | EPB |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

**Figure 6: File structure example: more 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.

---

## 2.5. Options

TOC

All the block bodies have the possibility to 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.

Skipping all the optional fields at once is straightforward because most of the blocks are made of a first part with fixed format, and a second optional part. Therefore, the Block Length field (present in the General Block Structure, see **Section 2.1**) can be used to skip everything till the next block.

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

- Option Type (2 bytes): it 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). In case of vendor-specific extensions that have to be identified uniquely, vendors must request an Option Code whose MSB is equal to zero.
- Option Length (2 bytes): it contains the actual length of the following 'Option Value' field without the padding bytes.
- Option Value (variable length): it contains the value of the given option, aligned to a 32-bit boundary. The actual length of this field (i.e. without the padding bytes) is specified by the Option Length field.

Options may be repeated several times (e.g. an interface that has several IP addresses associated to it) TODO: mention for each option, if it can/shouldn't appear more than one time. The option list is terminated by a Option which uses the special 'End of Option' code (opt_endofopt).

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

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Option Code              |         Option Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                       Option Value                            /
/          /* variable length, aligned to 32 bits */            /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/                                                               /
/                 . . . other options . . .                     /
/                                                               /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Option Code == opt_endofopt  |  Option Length == 0          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

**Figure 7: Options format.**

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

| Name | Code | Length | Description | Example(s) |
|---|---|---|---|---|
| opt_endofopt | 0 | 0 | It delimits the end of the optional fields. This block cannot be repeated within a given list of options. | |
| opt_comment | 1 | variable | A UTF-8 string containing a comment that is associated to the current block. | "This packet is the beginning of all of our problems" / "Packets 17-23 showing a bogus TCP retransmission, as reported in bugzilla entry 1486!" / "Captured at the southern plant" / "I've checked again, now it's working ok" / ... |

## 2.6. Data format

TOC

Endianess

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

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 dumps.

Please note: The endianess is indicated by the **Section Header Block**. As this block can appear several times in a pcapng file, a single file can contain both endianess variants!

Alignment

Most (all?) fields of this specification uses 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 bytes (marked in this document e.g. with "aligned to 32 bits") should be filled with zero bytes (TODO: is this requirement a good idea for the sake of performance / do we want to allow bogus bytes here?).

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 should be taken when reading and writing such values. TODO: the spec is not too consistent wrt how 64-bit values are saved. in the Packet blocks we clearly specify where the low and high 32-bits of a 64-bit timestamp should be saved. In the SHB we do use the endianess of the machine when we save the section length.

TODO - Maybe we have to specify something more here. Is what we're saying enough to avoid any kind of ambiguity?.

## 3. Block Definition

TOC

This section details the format of the body of the blocks currently defined.

## 3.1. Section Header Block (mandatory)

TOC

The Section Header Block is mandatory. It identifies the beginning of a section of the capture dump 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 8**.

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +---------------------------------------------------------------+
 0 |                   Block Type = 0x0A0D0D0A                     |
   +---------------------------------------------------------------+
 4 |                      Block Total Length                       |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 8 |                      Byte-Order Magic                         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
12 |          Major Version         |         Minor Version        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
16 |                                                               |
   |                          Section Length                       |
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
24 /                                                               /
   /                      Options (variable)                       /
/                                                               /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Block Total Length                       |
+---------------------------------------------------------------+
```

**Figure 8: 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 "\r\n\n\r" (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 ("\r\n\n\r") 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 endianess of the section. The endianess is recognized by reading the Byte Order Magic, that is located 8 bytes after the Block Type.
- Block Total Length: total size of this block, as described in **Section 2.1**.
- Byte-Order Magic: 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.
- Major Version: number of the current mayor version of the format. Current value is 1. This value should change if the format changes in such a way that tools that can read the new format could not read the old format (i.e., the code would have to check the version number to be able to read both formats).
- Minor Version: number of the current minor version of the format. Current value is 0. This value should change if the format changes in such a way that tools that can read the new format can still automatically read the new format but code that can only read the old format cannot read the new format.
- Section Length: 64-bit value specifying the length in bytes 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. Section Length equal -1 (0xFFFFFFFFFFFFFFFF) 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 -1), its value is always aligned to 32 bits, as all the blocks are aligned to 32-bit boundaries. Also, special care should be taken in accessing this field: since the alignment of all the blocks in the file is 32-bit, this field is not guaranteed to be aligned to a 64-bit boundary. This could be a problem on 64-bit workstations.
- Options: optionally, a list of options (formatted according to the rules defined in **Section 2.5**) can be present.

Adding new block types or options would not necessarily require that either Major or Minor numbers be changed, as code that does not know about the block type or option could 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 2.5**, the following options are valid within this block:

| **Name** | **Code** | **Length** | **Description** | **Example(s)** |
|---|---|---|---|---|
| shb_hardware | 2 | variable | An UTF-8 string containing the description of the hardware used to create this section. | "x86 Personal Computer" / "Sun Sparc Workstation" / ... |
| shb_os | 3 | variable | An UTF-8 string containing the name of the operating system used to create this section. | "Windows XP SP2" / "openSUSE 10.2" / ... |
| shb_userappl | 4 | variable | An UTF-8 string containing the name of the application used to create this section. | "dumpcap V0.99.7" / ... |

TOC

## 3.2. Interface Description Block (mandatory)

The Interface Description Block is mandatory. This block is needed to specify the characteristics of the network interface on which the capture has been made. In order to properly associate the captured data to the corresponding interface, the Interface Description Block must be defined before any other block that uses it; therefore, this block is usually placed immediately after the Section Header Block.

An Interface Description Block is valid only inside the section which it belongs to. The structure of a Interface Description Block is shown in **Figure 9**.

```
     0                   1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +---------------------------------------------------------------+
  0 |                    Block Type = 0x00000001                    |
    +---------------------------------------------------------------+
  4 |                      Block Total Length                       |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  8 |           LinkType            |           Reserved            |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 12 |                            SnapLen                            |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 16 /                                                               /
    /                      Options (variable)                       /
    /                                                               /
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                      Block Total Length                       |
    +---------------------------------------------------------------+
```

**Figure 9: 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 2.1**.
- LinkType: a value that defines the link layer type of this interface. The list of Standardized Link Layer Type codes is available in **Appendix C**.
- SnapLen: maximum number of bytes dumped from each packet. The portion of each packet that exceeds this value will not be stored in the file. (TODO: Is there a need to signal "no limit"?)
- Options: optionally, a list of options (formatted according to the rules defined in **Section 2.5**) can be present.

Interface ID: Tools that write / read the capture file associate a progressive 16-bit number (starting from '0') to each Interface Definition Block. This number is unique within each Section and uniquely identifies the interface (inside the current section); therefore, two Sections can have interfaces identified by the same identifiers. This unique identifier is referenced by other blocks (e.g. Packet Block) to point out the interface the block refers to (e.g. the interface that was used to capture the packet). (TODO - It would be nice, to have a "invalid Interface ID" defined, e.g. 0xFFFFFFFF)

In addition to the options defined in **Section 2.5**, the following options are valid within this block:

| **Name** | **Code** | **Length** | **Description** | **Example(s)** |
|---|---|---|---|---|
| if_name | 2 | Variable | A UTF-8 string containing the name of the device used to capture data. | "eth0" / "\Device\NPF_{AD1CE675-96D0-47C5-ADD0-2504B9126B68}" / ... |
| if_description | 3 | Variable | A UTF-8 string containing the description of the device used to capture data. | "Broadcom NetXtreme" / "First Ethernet Interface" / ... |
| if_IPv4addr | 4 | 8 | Interface network address and netmask. This option can be repeated multiple times within the same Interface Description Block when multiple IPv4 addresses are assigned to the interface. | 192 168 1 1 255 255 255 0 |
| | | | Interface network address and prefix length (stored in the last byte). This | |

| | | | | |
|---|---|---|---|---|
| if_IPv6addr | 5 | 17 | option can be repeated multiple times within the same Interface Description Block when multiple IPv6 addresses are assigned to the interface. | 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 | 6 | 6 | Interface Hardware MAC address (48 bits). | 00 01 02 03 04 05 |
| if_EUIaddr | 7 | 8 | Interface Hardware EUI address (64 bits), if available. | TODO: give a good example |
| if_speed | 8 | 8 | Interface speed (in bps). | 100000000 for 100Mbps |
| if_tsresol | 9 | 1 | Resolution of timestamps. If the Most Significant Bit is equal to zero, the remaining bits indicates the resolution of the timestamp as as a negative power of 10 (e.g. 6 means microsecond resolution, timestamps are the number of microseconds since 1/1/1970). If the Most Significant Bit is equal to one, the remaining bits indicates the resolution as 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). | 6 |
| if_tzone | 10 | 4 | Time zone for GMT support (TODO: specify better). | TODO: give a good example |
| if_filter | 11 | variable | The filter (e.g. "capture only TCP traffic") used to capture traffic. The first byte 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, ...) | 00 "tcp port 23 and host 10.0.0.5" |
| if_os | 12 | variable | 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 3.1**) because the capture can have been done on a remote machine. | "Windows XP SP2" / "openSUSE 10.2" / ... |
| if_fcslen | 13 | 1 | An integer value that specified the length of the Frame Check Sequence (in bits) for this interface. For link layers whose FCS length can change during time, the Packet Block Flags Word can be used (see **Appendix A**). | 4 |
| if_tsoffset | 14 | 8 | A 64 bits 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 | 1234 |

highly syncronized capture systems?

## 3.3. Enhanced Packet Block (optional)

TOC

An Enhanced Packet Block 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 dump generation. The format of an Enhanced Packet Block is shown in **Figure 10**.

The Enhanced Packet Block is an improvement over the original **Packet Block**:

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

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +---------------------------------------------------------------+
 0 |                    Block Type = 0x00000006                    |
   +---------------------------------------------------------------+
 4 |                      Block Total Length                       |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 8 |                         Interface ID                          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
12 |                        Timestamp (High)                       |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
16 |                        Timestamp (Low)                        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
20 |                         Captured Len                          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
24 |                          Packet Len                           |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
28 /                                                               /
   /                          Packet Data                          /
   /          /* variable length, aligned to 32 bits */            /
   /                                                               /
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   /                                                               /
   /                      Options (variable)                       /
   /                                                               /
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                      Block Total Length                       |
   +---------------------------------------------------------------+
```

**Figure 10: 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 2.1**.
- Interface ID: it 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 3.2**) of this field.
- Timestamp (High) and Timestamp (Low): high and low 32-bits of a 64-bit quantity representing the timestamp. The timestamp is a single 64-bit unsigned integer representing the number of units since 1/1/1970. The way to interpret this field is specified by the 'if_tsresol' option (see **Figure 9**) of the Interface Description block referenced by this packet. Please note that

differently from the libpcap file format, timestamps are not saved as two 32-bit values accounting for the seconds and microseconds since 1/1/1970. They are saved as a single 64-bit quantity saved as two 32-bit words.
- Captured Len: number of bytes captured from the packet (i.e. the length of the Packet Data field). It will be the minimum value among the actual Packet Length and the snapshot length (defined in **Figure 9**). The value of this field does not include the padding bytes added at the end of the Packet Data field to align the Packet Data Field to a 32-bit boundary
- Packet Len: actual length of the packet when it was transmitted on the network. It can be different from Captured Len if the user wants only a snapshot of the packet.
- Packet Data: the data coming from the network, including link-layer headers. The actual length of this field is Captured Len. The format of the link-layer headers depends on the LinkType field specified in the Interface Description Block (see **Section 3.2**) and it is specified in **Appendix D**.
- Options: optionally, a list of options (formatted according to the rules defined in **Section 2.5**) can be present.

In addition to the options defined in **Section 2.5** and in the **Packet Block**, the following options are valid within this block:

| Name | Code | Length | Description | Example(s) |
|---|---|---|---|---|
| epb_flags | 2 | 4 | A flags word containing link-layer information. A complete specification of the allowed flags can be found in **Appendix A**. | 0 |
| epb_hash | 3 | variable | This option contains a hash of the packet. The first byte specifies the hashing algorithm, while the following bytes contain the actual hash, whose size depends on the hashing algorithm, and hence from the value in the first bit. The hashing algorithm can be: 2s complement (algorithm byte = 0, size=XXX), XOR (algorithm byte = 1, size=XXX), CRC32 (algorithm byte = 2, size = 4), MD-5 (algorithm byte = 3, size=XXX), SHA-1 (algorithm byte = 4, size=XXX). 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. (TODO: the text above uses "first bit", but shouldn't this be "first byte"?!?) | TODO: give a good example |
| epb_dropcount | 4 | 8 | A 64bit integer value specifying the number of packets lost (by the interface and the operating system) between this packet and the preceding one. | 0 |

## 3.4. Simple Packet Block (optional)

TOC

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

A Simple Packet Block is similar to a Packet Block (see **Section 3.5**), but it is smaller, simpler to process and contains only a minimal set of information. This block is preferred to the standard Packet Block when performance or space occupation are critical factors, such as in sustained traffic dump applications. A capture file can contain both Packet Blocks and Simple Packet Blocks: for example, a capture tool could switch from 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 11** shows the format of the Simple Packet Block.

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +---------------------------------------------------------------+
 0 |                    Block Type = 0x00000003                    |
   +---------------------------------------------------------------+
 4 |                    Block Total Length                         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 8 |                        Packet Len                             |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
12 /                                                               /
   /                        Packet Data                            /
   /          /* variable length, aligned to 32 bits */            /
   /                                                               /
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                    Block Total Length                         |
   +---------------------------------------------------------------+
```

**Figure 11: 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 2.1**.
- Packet Len: actual length of the packet when it was transmitted on the network. Can be different from captured len if the packet has been truncated by the capture process.
- Packet Data: the data coming from the network, including link-layers 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 Packet Len (present in this header).

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 bytes requires only 16 bytes of overhead, which corresponds to an efficiency of more than 86%.

## 3.5. Packet Block (obsolete!)

TOC

The Packet Block is marked obsolete, better use the Enhanced Packet Block instead!

A Packet Block is the standard container for storing the packets coming from the network. The 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 dump generation. The format of a packet block is shown in **Figure 12**.

```
     0                   1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +---------------------------------------------------------------+
  0 |                    Block Type = 0x00000002                    |
    +---------------------------------------------------------------+
  4 |                    Block Total Length                         |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  8 |         Interface ID          |          Drops Count          |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 12 |                        Timestamp (High)                       |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 16 |                        Timestamp (Low)                        |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 20 |                         Captured Len                          |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 24 |                          Packet Len                           |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 28 /                                                               /
    /                          Packet Data                          /
    /          /* variable length, aligned to 32 bits */            /
    /                                                               /
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    /                                                               /
    /                      Options (variable)                       /
    /                                                               /
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                      Block Total Length                       |
    +---------------------------------------------------------------+
```

**Figure 12: 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 2.1**.
- Interface ID: it 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 3.2**) of this field.
- 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 already defined in the Enhanced Packet Block (**Section 3.3**).
- Captured Len: number of bytes captured from the packet (i.e. the length of the Packet Data field). It will be the minimum value among the actual Packet Length and the snapshot length (SnapLen defined in **Figure 9**). The value of this field does not include the padding bytes added at the end of the Packet Data field to align the Packet Data Field to a 32-bit boundary
- Packet Len: actual length of the packet when it was transmitted on the network. Can be different from Captured Len if the user wants only a snapshot of the packet.
- Packet Data: the data coming from the network, including link-layer headers. The format of the link-layer headers depends on the LinkType field specified in the Interface Description Block (see **Section 3.2**) and it is specified in **Appendix D**. The actual length of this field is Captured Len.
- Options: optionally, a list of options (formatted according to the rules defined in **Section 2.5**) can be present.

In addition to the options defined in **Section 2.5**, the following options are valid within this block:

| Name | Code | Length | Description | Example(s) |
|---|---|---|---|---|
| pack_flags | 2 | 4 | Same as epb_flags of the enhanced packet block. | 0 |
| pack_hash | 3 | variable | Same as epb_hash of the enhanced packet block. | TODO: give a good example |

**Table 1**

## 3.6. Name Resolution Block (optional)

TOC

The Name Resolution Block 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, this prevents the need of a name resolution in a delayed time, when the association between names and addresses can be different from the one in use at capture time. Moreover, the Name Resolution Block avoids the need of issuing a lot of DNS requests every time the trace capture is opened, and allows to have name resolution also when reading the capture with a machine not connected to the network.

A Name Resolution Block is normally placed at the beginning of the file, but no assumptions can be taken about its position. Name Resolution Blocks can be added in a second time by tools that process the file, like network analyzers.

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

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +---------------------------------------------------------------+
 0 |                    Block Type = 0x00000004                    |
   +---------------------------------------------------------------+
 4 |                      Block Total Length                       |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 8 |      Record Type              |         Record Length         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
12 /                       Record Value                            /
   /          /* variable length, aligned to 32 bits */            /
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   .                                                               .
   .                  . . . other records . . .                    .
   .                                                               .
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |  Record Type == end_of_recs   |  Record Length == 00          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   /                                                               /
   /                      Options (variable)                       /
   /                                                               /
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                      Block Total Length                       |
   +---------------------------------------------------------------+
```

**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 2.1**.

This is followed by a zero-terminated list of records (in the TLV format), each of which contains an association between a network address and a name. There are three possible types of records:

| Name | Code | Length | Description | Example(s) |
|---|---|---|---|---|
| nres_endofrecord | 0 | 0 | It 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. | |
| nres_ip4record | 1 | Variable | Specifies an IPv4 address (contained in the first 4 bytes), followed by one or more zero-terminated strings containing the DNS entries for that address. | 127 0 0 1 "localhost" |
| nres_ip6record | 2 | Variable | Specifies an IPv6 address (contained in the first 16 bytes), followed by one or more zero-terminated strings containing the DNS entries for that address. | TODO: give a good example |

**Table 2**

Each Record Value is aligned to a 32-bit boundary. The corresponding Record Length reflects the actual length of the Record Value.

After the list of Name Resolution Records, optionally, a list of options (formatted according to the rules defined in **Section 2.5**) can be present.

In addiction to the options defined in **Section 2.5**, the following options are valid within this block:

| Name | Code | Length | Description | Example(s) |
|---|---|---|---|---|
| ns_dnsname | 2 | Variable | A UTF-8 string containing the name of the machine (DNS server) used to perform the name resolution. | "our_nameserver" |
| ns_dnsIP4addr | 3 | 4 | The IPv4 address of the DNS server. | 192 168 0 1 |
| ns_dnsIP6addr | 4 | 16 | The IPv6 address of the DNS server. | TODO: give a good example |

TODO: Add an "Interface ID" option, if the name resolution is only valid for a specific interface?

TODO: Does it make sense to have two "optional mechanisms" (records vs. options) here?

## 3.7. Interface Statistics Block (optional)

TOC

The Interface Statistics Block 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**.

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +---------------------------------------------------------------+
 0 |                   Block Type = 0x00000005                     |
   +---------------------------------------------------------------+
 4 |                      Block Total Length                       |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 8 |                         Interface ID                          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
12 |                        Timestamp (High)                       |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
16 |                        Timestamp (Low)                        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
20 /                                                               /
   /                      Options (variable)                       /
   /                                                               /
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                      Block Total Length                       |
   +---------------------------------------------------------------+
```

**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 2.1**.
- Interface ID: it 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 3.2**) of this field. Please note: in former versions of this document, this field was 16 bits only. As this differs from its usage in other places of this doc and as this block was not used "in the wild" before (as to the knowledge of the authors), it seems reasonable to change it to 32 bits!
- Timestamp: time this statistics refers to. The format of the timestamp is the same already defined in the Enhanced Packet Block (**Section 3.3**).
- Options: optionally, a list of options (formatted according to the rules defined in **Section 2.5**) 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 addiction to the options defined in **Section 2.5**, the following options are valid within this block:

| **Name** | **Code** | **Length** | **Description** | **TODO: give a good example** |
|---|---|---|---|---|
| isb_starttime | 2 | 8 | Time in which the capture started; time will be stored in two blocks of four bytes each. The format of the timestamp is the same already defined in the Enhanced Packet Block (**Section 3.3**). | TODO: give a good example |
| isb_endtime | 3 | 8 | Time in which the capture ended; ; time will be stored in two blocks of four bytes each. The format of the timestamp is the same already defined in the Enhanced Packet Block (**Section 3.3**). | TODO: give a good example |
| isb_ifrecv | 4 | 8 | Number of packets received from the physical interface starting from the beginning of the capture. | 100 |
| isb_ifdrop | 5 | 8 | Number of packets dropped by the interface due to lack of resources starting from the beginning of the capture. | 0 |
| isb_filteraccept | 6 | 8 | Number of packets accepted by filter starting from the beginning of the capture. | 100 |
| isb_osdrop | 7 | 8 | Number of packets dropped by the operating system starting from the beginning of the capture. | 0 |
| isb_usrdeliv | 8 | 8 | 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 lay in the OS buffers when the capture ended. | 0 |

All the fields that refer to packet counters are 64-bit values, represented with the byte 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. Experimental Blocks (deserved to a further investigation)

TOC

### 4.1. Alternative Packet Blocks (experimental)

TOC

Can some other packet blocks (besides the ones described in the previous paragraphs) be useful?

### 4.2. Compression Block (experimental)

TOC

The Compression Block is optional. A file can contain an arbitrary number of these blocks. A Compression Block, as the name says, is used to store compressed data. Its format is shown in **Figure 15**.

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +---------------------------------------------------------------+
   |                        Block Type = ?                         |
   +---------------------------------------------------------------+
   |                      Block Total Length                       |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |  Compr. Type  |                                               |
   +-+-+-+-+-+-+-+-+                                               |
   |                                                               |
   |                       Compressed Data                         |
   |                                                               |
   |              /* variable length, byte-aligned */              |
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                      Block Total Length                       |
   +---------------------------------------------------------------+
```

**Figure 15: Compression Block format.**

The fields have the following meaning:

- Block Type: The block type of the Compression Block is not yet assigned.
- Block Total Length: total size of this block, as described in **Section 2.1**.
- Compression Type: specifies the compression algorithm. Possible values for this field are 0 (uncompressed), 1 (Lempel Ziv), 2 (Gzip), other?? Probably some kind of dumb and fast compression algorithm could be effective with some types of traffic (for example web), but which?
- Compressed Data: data of this block. Once decompressed, it is made of other blocks.

## 4.3. Encryption Block (experimental)

TOC

The Encryption Block is optional. A file can contain an arbitrary number of these blocks. An Encryption Block is used to store encrypted data. Its format is shown in **Figure 16**.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------------------------------------------------------+
|                        Block Type = ?                         |
+---------------------------------------------------------------+
|                      Block Total Length                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Encr. Type  |                                               |
+-+-+-+-+-+-+-+-+                                               |
|                                                               |
|                       Encrypted Data                          |
|                                                               |
|              /* variable length, byte-aligned */              |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Block Total Length                       |
+---------------------------------------------------------------+
```

**Figure 16: Encryption Block format.**

The fields have the following meaning:

- Block Type: The block type of the Encryption Block is not yet assigned.
- Block Total Length: total size of this block, as described in **Section 2.1**.
- Encryption Type: specifies the encryption algorithm. Possible values for this field are ??? (TODO) NOTE: this block should probably contain other fields, depending on the encryption algorithm. To be defined precisely.

- Encrypted Data: data of this block. Once decripted, it originates other blocks.

## 4.4. Fixed Length Block (experimental)

TOC

The Fixed Length Block is optional. A file can contain an arbitrary number of these blocks. A Fixed Length Block can be used to optimize the access to the file. Its format is shown in **Figure 17**. A Fixed Length Block stores records with constant size. It contains a set of Blocks (normally Packet Blocks or Simple Packet Blocks), of wihich it specifies the size. Knowing this size a priori helps to scan the file and to load some portions of it without truncating a block, and is particularly useful with cell-based networks like ATM.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------------------------------------------------------+
|                        Block Type = ?                         |
+---------------------------------------------------------------+
|                      Block Total Length                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Cell Size            |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               |
|                                                               |
|                        Fixed Size Data                        |
|                                                               |
|              /* variable length, byte-aligned */              |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Block Total Length                       |
+---------------------------------------------------------------+
```

**Figure 17: Fixed Length Block format.**

The fields have the following meaning:

- Block Type: The block type of the Fixed Length Block is not yet assigned.
- Block Total Length: total size of this block, as described in **Section 2.1**.
- Cell size: the size of the blocks contained in the data field.
- Fixed Size Data: data of this block.

## 4.5. Directory Block (experimental)

TOC

If present, this block contains the following information:

- number of indexed packets (N)
- table with position and length of any indexed packet (N entries)

A directory block must be followed by at least N packets, otherwise it must be considered invalid. It can be used to efficiently load portions of the file to memory and to support operations on memory mapped files. This block can be added by tools like network analyzers as a consequence of file processing.

## 4.6. Traffic Statistics and Monitoring Blocks (experimental)

TOC

One or more blocks could be defined to contain network statistics or traffic monitoring information. They could be use to store data collected from RMON or Netflow probes, or from other network monitoring tools.

## 4.7. Event/Security Block (experimental)

TOC

This block could be used to store events. Events could contain generic information (for example network load over 50%, server down...) or security alerts. An event could be:

- skipped, if the application doesn't know how to do with it
- processed independently by the packets. In other words, the applications skips the packets and processes only the alerts
- processed in relation to packets: for example, a security tool could load only the packets of the file that are near a security alert; a monitoring tool could skip the packets captured while the server was down.

## 5. Recommended File Name Extension: .pcapng

TOC

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

At least in the "Windows world", 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. 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 (like the current usage of .cap for a pletora of different capture file formats) other file name extensions than .pcapng should be avoided!

## 6. How to add Vendor / Domain specific extensions

TOC

TODO - explain the preferred way to add new block types and new options for existing blocks in more detail.

## 7. Conclusions

TOC

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 dump 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.

## Appendix A. Packet Block Flags Word

TOC

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

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). |
| | FCS length, in bytes (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 |

| | |
|---|---|
| 5-8 | change during time. |
| 9-15 | Reserved (must be set to zero). |
| 16-31 | 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?). |

## Appendix B. Standardized Block Type Codes

TOC

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

As pointed out in **Section 2.1**, Block Types 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. A request should be sent to the authors of this document to add a new Standard Block Type code to the specification.

Here is a list of the Standardized Block Type Codes.

| Block Type Code | Description |
|---|---|
| 0x00000000 | Reserved ??? |
| 0x00000001 | **Interface Description Block** |
| 0x00000002 | **Packet Block** |
| 0x00000003 | **Simple Packet Block** |
| 0x00000004 | **Name Resolution Block** |
| 0x00000005 | **Interface Statistics Block** |
| 0x00000006 | **Enhanced Packet Block** |
| 0x00000007 | IRIG Timestamp Block (requested by Gianluca Varenni <gianluca.varenni@cacetech.com>, CACE Technologies LLC) |
| 0x00000008 | Arinc 429 in AFDX Encapsulation Information Block (requested by Gianluca Varenni <gianluca.varenni@cacetech.com>, CACE Technologies LLC) |
| 0x0A0D0D0A | **Section Header Block** |
| 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 | Reserved. Used to detect trace files corrupted because of file transfers using the FTP protocol in text mode. |

## Appendix C. Standardized Link Type Codes

TOC

NOTE: we should decide if we want to have this list here or in a separate document. It may make sense to have this list in a separate document and describe the format of a frame for each different linktype, or specify that the frame format is proprietary of a company and not public. There are a large number of encapsulations that are really vague and unspecified at all (even the name does not make any sense). Moreover, we should decide if we want to have *all* the linktypes (LINKTYPE_XXX) defined by libpcap, or just a subset of them, thus trying to remove the big mess and confusion of similar headers.

Here is a list of the Standardized Link Type Codes.

| Link Type Name | Link Type Code | Description |
|---|---|---|
| LINKTYPE_NULL | 0 | No link layer information. A packet saved with this link layer contains a raw L3 packet preceded by a 32-bit host-byte-order AF_ value indicating the specific L3 type. |
| LINKTYPE_ETHERNET | 1 | D/I/X and 802.3 Ethernet |
| LINKTYPE_EXP_ETHERNET | 2 | Experimental Ethernet (3Mb) |
| LINKTYPE_AX25 | 3 | Amateur Radio AX.25 |
| LINKTYPE_PRONET | 4 | Proteon ProNET Token Ring |
| LINKTYPE_CHAOS | 5 | Chaos |
| LINKTYPE_TOKEN_RING | 6 | IEEE 802 Networks |
| LINKTYPE_ARCNET | 7 | ARCNET, with BSD-style header |
| LINKTYPE_SLIP | 8 | Serial Line IP |
| LINKTYPE_PPP | 9 | Point-to-point Protocol |
| LINKTYPE_FDDI | 10 | FDDI |
| LINKTYPE_PPP_HDLC | 50 | PPP in HDLC-like framing |
| LINKTYPE_PPP_ETHER | 51 | NetBSD PPP-over-Ethernet |
| LINKTYPE_SYMANTEC_FIREWALL | 99 | Symantec Enterprise Firewall |
| LINKTYPE_ATM_RFC1483 | 100 | LLC/SNAP-encapsulated ATM |
| LINKTYPE_RAW | 101 | Raw IP |
| LINKTYPE_SLIP_BSDOS | 102 | BSD/OS SLIP BPF header |
| LINKTYPE_PPP_BSDOS | 103 | BSD/OS PPP BPF header |
| LINKTYPE_C_HDLC | 104 | Cisco HDLC |
| LINKTYPE_IEEE802_11 | 105 | IEEE 802.11 (wireless) |
| LINKTYPE_ATM_CLIP | 106 | Linux Classical IP over ATM |
| LINKTYPE_FRELAY | 107 | Frame Relay |
| LINKTYPE_LOOP | 108 | OpenBSD loopback |
| LINKTYPE_ENC | 109 | OpenBSD IPSEC enc |
| LINKTYPE_LANE8023 | 110 | ATM LANE + 802.3 (Reserved for future use) |
| LINKTYPE_HIPPI | 111 | NetBSD HIPPI (Reserved for future use) |
| LINKTYPE_HDLC | 112 | NetBSD HDLC framing (Reserved for future use) |
| LINKTYPE_LINUX_SLL | 113 | Linux cooked socket capture |
| LINKTYPE_LTALK | 114 | Apple LocalTalk hardware |
| LINKTYPE_ECONET | 115 | Acorn Econet |
| LINKTYPE_IPFILTER | 116 | Reserved for use with OpenBSD ipfilter |
| LINKTYPE_PFLOG | 117 | OpenBSD DLT_PFLOG |
| LINKTYPE_CISCO_IOS | 118 | For Cisco-internal use |
| LINKTYPE_PRISM_HEADER | 119 | 802.11+Prism II monitor mode |
| LINKTYPE_AIRONET_HEADER | 120 | FreeBSD Aironet driver stuff |
| LINKTYPE_HHDLC | 121 | Reserved for Siemens HiPath HDLC |
| LINKTYPE_IP_OVER_FC | 122 | RFC 2625 IP-over-Fibre Channel |
| LINKTYPE_SUNATM | 123 | Solaris+SunATM |
| LINKTYPE_RIO | 124 | RapidIO - Reserved as per request from Kent Dahlgren <kent@praesum.com> for private use. |
| LINKTYPE_PCI_EXP | 125 | PCI Express - Reserved as per request from Kent Dahlgren <kent@praesum.com> for private use. |
| LINKTYPE_AURORA | 126 | Xilinx Aurora link layer - Reserved as per request from Kent Dahlgren <kent@praesum.com> for private use. |
| LINKTYPE_IEEE802_11_RADIO | 127 | 802.11 plus BSD radio header |
| LINKTYPE_TZSP | 128 | Tazmen Sniffer Protocol - Reserved for the TZSP encapsulation, as per request from Chris Waters <chris.waters@networkchemistry.com> TZSP is a generic |

| | | |
|---|---|---|
| | | encapsulation for any other link type, which includes a means to include meta-information with the packet, e.g. signal strength and channel for 802.11 packets. |
| LINKTYPE_ARCNET_LINUX | 129 | Linux-style headers |
| LINKTYPE_JUNIPER_MLPPP | 130 | Juniper-private data link type, as per request from Hannes Gredler <hannes@juniper.net>. The corresponding DLT_s are used for passing on chassis-internal metainformation such as QOS profiles, etc.. |
| LINKTYPE_JUNIPER_MLFR | 131 | Juniper-private data link type, as per request from Hannes Gredler <hannes@juniper.net>. The corresponding DLT_s are used for passing on chassis-internal metainformation such as QOS profiles, etc.. |
| LINKTYPE_JUNIPER_ES | 132 | Juniper-private data link type, as per request from Hannes Gredler <hannes@juniper.net>. The corresponding DLT_s are used for passing on chassis-internal metainformation such as QOS profiles, etc.. |
| LINKTYPE_JUNIPER_GGSN | 133 | Juniper-private data link type, as per request from Hannes Gredler <hannes@juniper.net>. The corresponding DLT_s are used for passing on chassis-internal metainformation such as QOS profiles, etc.. |
| LINKTYPE_JUNIPER_MFR | 134 | Juniper-private data link type, as per request from Hannes Gredler <hannes@juniper.net>. The corresponding DLT_s are used for passing on chassis-internal metainformation such as QOS profiles, etc.. |
| LINKTYPE_JUNIPER_ATM2 | 135 | Juniper-private data link type, as per request from Hannes Gredler <hannes@juniper.net>. The corresponding DLT_s are used for passing on chassis-internal metainformation such as QOS profiles, etc.. |
| LINKTYPE_JUNIPER_SERVICES | 136 | Juniper-private data link type, as per request from Hannes Gredler <hannes@juniper.net>. The corresponding DLT_s are used for passing on chassis-internal metainformation such as QOS profiles, etc.. |
| LINKTYPE_JUNIPER_ATM1 | 137 | Juniper-private data link type, as per request from Hannes Gredler <hannes@juniper.net>. The corresponding DLT_s are used for passing on chassis-internal metainformation such as QOS profiles, etc.. |
| LINKTYPE_APPLE_IP_OVER_IEEE1394 | 138 | Apple IP-over-IEEE 1394 cooked header |
| LINKTYPE_MTP2_WITH_PHDR | 139 | ??? |
| LINKTYPE_MTP2 | 140 | ??? |
| LINKTYPE_MTP3 | 141 | ??? |
| LINKTYPE_SCCP | 142 | ??? |
| LINKTYPE_DOCSIS | 143 | DOCSIS MAC frames |
| LINKTYPE_LINUX_IRDA | 144 | Linux-IrDA |
| LINKTYPE_IBM_SP | 145 | Reserved for IBM SP switch and IBM Next Federation switch. |
| LINKTYPE_IBM_SN | 146 | Reserved for IBM SP switch and IBM Next Federation switch. |

## Appendix D. Link Layer Headers

TOC

The Packet Data field of the Packet Blocks won't start with the actual network data captured, but with some link type specific "meta data". The format of this meta data depends on the link type used. TODO: mention example code in libpcap that lists these link headers.

## Authors' Addresses

TOC

Loris Degioanni
CACE Technologies
1949 Fifth Street #103
Davis, CA 95616
United States
**Phone:** +1 530 758 2790
**Email:** loris.degioanni@cacetech.com
**URI:** http://www.winpcap.org/loris/

Fulvio Risso
Politecnico di Torino
Corso Duca degli Abruzzi, 24
Torino 10129
Italy
**Phone:** +39 011 564 7008
**Email:** fulvio.risso@polito.it
**URI:** http://staff.polito.it/fulvio.risso/

Gianluca Varenni
CACE Technologies
1949 Fifth Street #103
Davis, CA 95616
United States
**Phone:** +1 530 758 2790
**Email:** gianluca.varenni@cacetech.com
**URI:** http://www.winpcap.org/gianluca/

TOC

## Full Copyright Statement

## Intellectual Property

## Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.