Ročník 2012 electroscope Číslo V

Extended Comparison Study on Merging PCAP Files

V. Veselý
Department of Information Systems, Faculty of Information Technology, Brno University of Technology
Božetěchova 2, Brno
E-mail : ivesely@fit.vutbr.cz

Abstract:
Different formats of PCAP (Packet CAPtures) files are nowadays widely used for storing computer network communications. This paper outlines information about PCAP formats with focus on packets precise timing and order. In thjs paper we compare capabilities of different open-source tools for handling PCAP files and introduce our own tool for merging multiple PCAPs. Finally, we evaluate the performance of the implemented tool and compare it with existing implementations.

INTRODUCTION

Traffic monitoring is an essential task for network administrators, ISPs or law enforcing agencies. Unfortunately still no standard exists for packet traces exchange. The most widely accepted PCAP formats are **LibPCAP**, **PCAP Next Generation** and **Microsoft NetMon**.
The computer communication is often load-balanced and then available traffic captures come from multiple monitoring probes. Because of that the traffic from one host could be spread across different PCAP files. We face a problem how to "put together" captures correctly whenever we want to successfully trace and reconstruct a particular traffic flow.
The structure of the paper is following. First we will try to provide an overview of basic concepts, theory and state of the art in the area of joining PCAP files. In the next chapter we will discuss our contribution to the topic – our own software solution called *PCAPMerger*. Following section will show results of validation/verification tests and comparison with other tools. We will conclude this paper with final remarks and we will briefly mention ideas for future work.

STATE OF THE ART

In this section we will introduce relevant information about different PCAP file's formats. We will also describe some tools for handling multiple PCAP files – either to simply **concatenate** their content or to **merge** their content (sort them according to timestamp). And lastly we mention issue regarding handling of timestamp information in PCAP files.

LibPCAP

LibPCAP format [1] is formerly defined as a part of the library with the same name [2]. It is oldest but predominant PCAP format, mostly because it is default for applications like *tcpdump* or *Wireshark*.

It uses just one *Global Header (GH)* where general information about traffic capture (namely little/big endianess detector, the correction to UTC time, snapshot length, data link type for all frames) is stored. After follows frame layout which consists by turns of *Packet Header (PH)* (including timestamps and data sizes) and *Packet Data (PD)*.
Typical structure of LibPCAP is depicted on Fig. 1.

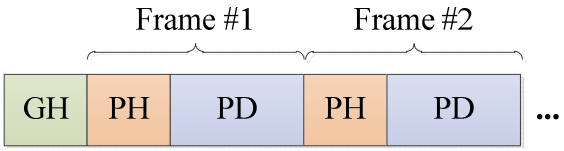

Fig. 1: LibPCAP file structure and frame layout

Multiple extensions (e.g. nanoseconds precision resolution) or vendor specific (e.g. Nokia, RedHat, SuSE) variants of this format were introduced during the years. Nevertheless basic version defined above is the only one generally accepted by all applications.

PCAP Next Generation

PCAP Next Generation (PCAPng) format for storing network communication never became more than IETF draft and currently is maintained outside any IETF working group [3].
A PCAPng file consists of multiple blocks sharing the same common format. Blocks could be categorized into four different groups according to rules of their presence in file: *Mandatory* (at least one block must be present), *Optional* (blocks may appear), *Obsolete* (usage of blocks is depreciated) and *Experimental* (usage is not yet firmly defined but these blocks could be somehow helpful). Following blocks are the most important for this paper:

- *Section Header Block (SHB)* – this is a mandatory block. It defines the most important parameters of PCAP file (length of section, byte-order and options).
- *Interface Description Block (IDB)* – it is mandatory and describes characteristics of sniffing interface (link type, snapshot length, IP

address, MAC address, interface speed, timestamp resolution options with time zone information, applied traffic filters).

- *Enhanced Packet Block (EPB)* – it is optional and contains single captured packet or its portion (frame) with all relevant information like interface ID, timestamp, captured length and packet length, packet data, etc.
- *Simple Packet Block (SPB)* – it is optional and contains single captured frame or its portion (frame), with a minimal set of information about it (just packet length and data).

PCAPng blocks form a tree structure. The physical layout of each PCAP file consists of at least one SHB, with one IDB and corresponding EPB and SPB for packets sniffed on the interface. Typical PCAPng file could have the same structure as it is depicted on Fig. 2.

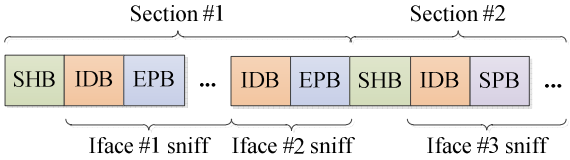

Fig. 2: Structure and layout of PCAPng blocks

Microsoft NetMon

The most complex and also the most advanced features are offered by Microsoft NetMon (MS NetMon) capture file format, currently version 2.3. This format was introduced in **Microsoft Network Monitor (MNM)** [4] traffic analyzing tool. A MNM PCAP file is divided into sections storing the following data:

- *Frame Table* – Simple list where each record represents starting offset of captured frame.
- *Frame Layout* – Actual frames are stored here. Each one of them starts with *Frame Header (FH)* which consists of time offset, real and stored frame size, raw frame data and additional information.
- *Process Info Table* – When traffic capturing is done on an end host then operating system could prepend relevant information about a target or source process of the frame. Such information consists of application path, icon, unique process identifier, source/destination port or source/destination IP address.
- *Comment Info* – Any frame could have also additional textual comments, which are stored in this section.
- *Extended Info* – Since version 2.0 MS NetMon is capable recording another time information. Additional to time and delta offset provided by NetMon library it includes also FILETIME timestamps provided by Windows kernel process. *Extended Info* also holds time zone information for each frame so it is possible to join traffic captures from different places on Earth without any additional time recalculations.

Every Microsoft NetMon file starts with *Capture File Header (CFH)* which acts as signposts containing starting offsets (byte address) to previously mentioned sections.
Fig. 3 shows the whole structure and illustrates offset pointer logic.

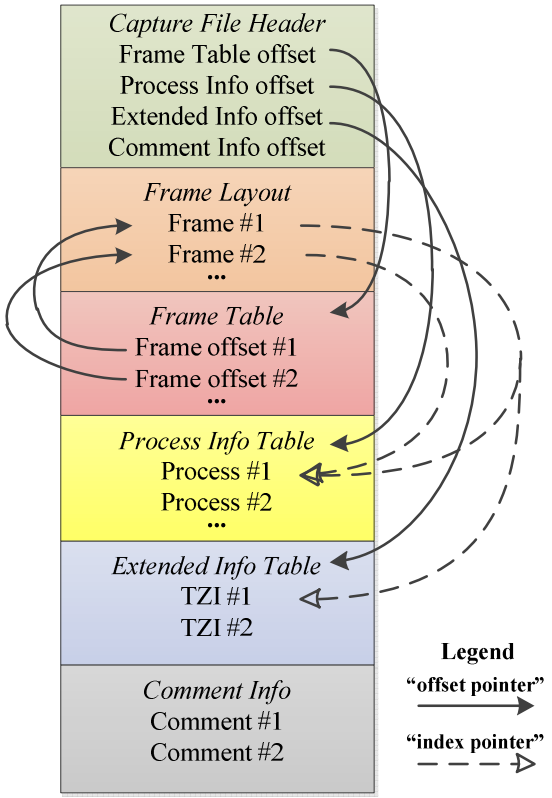

Fig. 3: MS NetMon structure and passing of pointers

Existing Tools

There exists a variety of tools for handling PCAP files differing in what API they use or in which languages they are written. Some of them are even subparts of deep packet analyzing programs like Wireshark or MNM. For instance *capinfo* (part of Wireshark's installation) is a useful program for displaying all important information about one PCAP file. The most known and widely used tools for merging multiple PCAP files into the one output file are Wireshark's *Mergecap* [5], FreeBSD's *tcpslice* [6] or MNM's *NMcap* [7].
Unfortunately only *NMCap* is capable of merging PCAP files in any of previously mentioned formats, the others support only a limited set. Table 1 summarizes this information.

Table 1: Supported formats

Program	***LibPCAP***	***PCAPng***	***MS NetMon***
Mergecap	YES	YES	NO
tcpslice	YES	NO	NO
NMCap	YES	YES	YES

Time Order Issue

In this section we introduce term **time order issue**. We describe how PCAP files with this problem look like, how it happens and how it influences existing tools for merging. But first please note the following table which summarizes how time information is stored in different formats above:

Table 2: Time information in PCAP formats

Format	*Description*
LibPCAP	Time stored per frame in timestamp with two UINT32 fields – 1st number of seconds and 2nd number of microseconds since 1/1/1970. Achievable precision is 1 µs.
PCAPng	Time also stored per frame in timestamp as one UINT64 field measuring units since 1/1/1970 with adjustable precision according to *if_tsresol* Option settings. Default precision is 1 µs.
MS NetMon	The CFH contains initial timestamp marking the beginning of packet capture. It is represented as 16 B SYSTEMTIME variable enabling for 1 ms precision. Then each frame stores only (even negative) offset value as UINT64 measuring number of units (0.1 µs increments, a.k.a. **ticks**) since initial timestamp. Since version 2.4 each frame stores another timestamp as independent UINT64 FILETIME variable with 0.1 µs precision.

Now assume that we have a PCAP file where timestamps are not growing incrementally in list of all **frame containers** (e.g. LibPCAP's PH, MS NetMon's FH, PCAPng's EPB). Fig. 4 contains Wireshark's screenshot that illustrates this instance – timestamps of packets #1, #2, #6 and #7 are correct, but timestamps of #3, #4 and #5 are not in order.

No.	Time	Length	Protocol	Info
1	0.000000	60	TCP	sslp > ms-wbt-serv
2	0.099089	1451	TPKT	Continuation
3	-0.009980	91	TPKT	[TCP Retransmissio
4	-0.000823	2987	TPKT	[TCP Retransmissio
5	-0.009452	60	TCP	sslp > ms-wbt-serv
6	0.103430	316	SSDP	NOTIFY * HTTP/1.1
7	0.102208	380	SSDP	NOTIFY * HTTP/1.1

Fig. 4: Snapshot of PCAP file with Time order issue

This kind of PCAPs with time order issue could be a result of the packet capture obtained in one of the following ways:

- Sniff is performed on the multiple interfaces – that usually means that frame containers are stored chronologically per each interface section but not altogether.
- PCAP is created by a simple concatenation of two or even more PCAPs – the last timestamp of a file is not necessary preceding timestamps of packets in the next file.
- Frame containers are disordered – either by purposely exporting them from the one PCAP in to another or because of delayed packet processing by capturing engine.

In general, packets can be received simultaneously on different interfaces or even on different capturing machines and can have (nearly) same timestamp. However relevant timestamps are not stored in chronological order in PCAP files with time order issue.

Now if we use any previously mentioned tool (*Mergecap*, *NMcap* or *tcpslice*) and try to merge input PCAPs into the chronologically sorted PCAP then we will receive the wrong output. Those programs take from each input PCAP file always the first unprocessed frame container and compare their timestamps between each other. Hence described algorithm just preserves bad time order of frame containers in the resulting output PCAP. Existing tools expect only the basic physical layout of input files – timestamps are growing incrementally without any exception.

CONTRIBUTION

We have decided to solve previously mentioned issue with our own tool, called *PCAPMerger*. This section introduces some of the basic implementation and design notes.

PCAPMerger is implemented in C# language as a console application for .NET framework version 4.

Previous version of *PCAPMerger* was based on MNM API and its C# wrapper. Unfortunately resulting application was proven to be ineffective. Hence, we have decided to reimplement it to be independent on any API and to use a low-level binary file access – direct file stream reading and writing instead of indirect access to content via API. Without going to unnecessary details *PCAPMerger* works as follows where time complexity is based on total number of frames n in all input PCAP files:

1) `ParseArgs()` – Tries to parse input arguments from console and setup appropriately application's behavior.
2) `CheckAndOpenSfsFiles()` – Check input PCAP files existence and open them for binary read operations.
3) `ParseSfsFiles()` – Parse binary content of PCAPs, namely all important general information (e.g. GH, SHB, CFH, frame headers). Initialize abstract data type collection named *Frames* for each PCAP files that stores frame relevant data (e.g. timestamps, frame length). Time complexity $O(n)$.

4) `VerifyMediaTypes()` – Function verifies that all frames in PCAPs are of supported L2 type. Following types are accepted Ethernet, FDDI, RawIP, IEEE 802.11, ATM, any others are removed.
5) `SortFrames()` – Concatenate all *Frames* to one giant collection and sort it according to timestamp information using own delegate sorting algorithm. Time complexity varies upon sorting algorithm – application offers to use either QuickSort $O(n^2)$or HeapSort $O(n \log n)$.
6) `CreateOutput()` – Creates resulting PCAP file for binary writing. The file includes all frames from input PCAPs but sorted chronologically and it is in LibPCAP format. Time complexity $O(n)$.
7) `CloseFiles()` – Close input PCAPs.

Thanks to used sorting mechanisms and initial preprocessing of data, our tool is capable of effective merging of PCAPs even thou they suffer with time order issue.
PCAPMerger supports all three previously described PCAP formats and is easily upgradable to any new future format to come. Besides that we can also merge mixture of different PCAP files on the input.

PERFORMANCE TESTING

Scenario and settings

Only *PCAPMerger* is capable to correctly merge PCAP files inside which total time order are not kept. Nevertheless in this section we show performance of our application in use-case where time order of frames is ensured – merging ordinary PCAP files without time order issue and motivation behind is to compare performance of our solution with the existing ones.
To prove *PCAPMerger* effectiveness we have conducted series of tests focused on measuring CPU and memory requirements and I/O operations. Among tested tools there is also our previous version of *PCAPMerger old* from the beginning of year 2012 that was based on NetMon API.
All programs participated in tests, namely, *Mergecap*, *NMcap*, previous version called *PCAPMerger old* and our brand new *PCAPMerger* – are compared on the same PCAP testing set in LibPCAP format. It is because (see Table **1**), this is the only format that is supported by all programs. Testing set consists of one referential big PCAP file format with communication recorded on the backbone network of Brno University of Technology. A size of this file is 1 GB, which reflects usual sizes of real-world PCAP files. The big PCAP is split into ten chronologically consecutive smaller input PCAP files with the approximately 100 MB each.
All measurements were performed on computer machine with Intel Core i7 CPU quad core with 3.6 GHz, HDD 500GB with NTFS, installed 16 GB of DDR3 RAM and running Windows 7 x64.
Smaller input files are passed to applications in chronologically reversed order to test the worst case scenario for tested routines. Table **3** and Table **4** summarize measured values of tested tools done by Windows Performance Monitor [8] and Windows Performance Analyzer [9].
The meaning of numbers in Table 3 is following:

- ***Time*** = duration of application run;
- ***Memory*** = total size of memory pages touched by process during application run;
- ***I/O Read/Write*** = the first line is the number of read/write operations, the second line is total number of bytes read/write during I/O operations, the third line is average amount of data processed per one operation.

Table **4** on the next page provides a comparison of *PCAPMerger* with other test participants.

Table 3: Absolute comparison of measured parameters during tested programs runs

Program	*Time*	*Memory*	*I/O Read*	*I/O Write*
PCAPMerger	8.6 s	52 MB	1,578,758 ops 5,583,342,861 B c. 3,537 B/op	306,992 ops 1 257 436 288 B c. 4,096 B/op
Mergecap	3.3 s	6.3 MB	288,986 ops 1,183,609,240 B c. 4,096 B/op	306,992 ops 1,257,436,288 B c. 4,096 B/op
NMCap	293.9 s	794 MB	36,744 ops 1,204,307,836 B c. 32,776 B/op	3,159 ops 2,301,226,389 B c. 728,466 B/op
PCAPMerger old	61.1 s	112 MB	73,365 ops 2,403,996,078 B c. 32,767 B/op	5,913 ops 3,697,550,024 B c. 625,325 B/op

Table 4: Relative comparison of measured parameters during tested programs runs

PCAPMerger* vs. *Program	***Run speed***	***Memory requirements***	***Bytes read***	***Bytes written***
Mergecap	c. 2.6× slower	c. 8.3× more	c. 4.7× more	equal
NMCap	c. 34.2× faster	c. 15.3× less	c. 4.6× more	c. 1.8× more
PCAPMerger old	c. 7.1× faster	c. 2.1× less	c. 2.3× more	c. 2.9× more

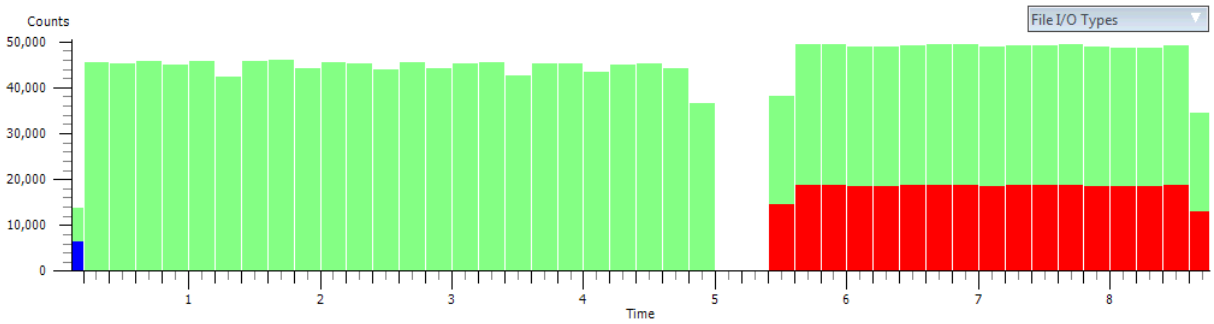

Fig. 5: Arrangement of I/O operation types for *PCAPMerger*

Discussion

Each output was automatically compared with the original big PCAP file. No content differences (none missing or malformed frames) were detected although sizes of output files slightly varied according to applications approaches to storing information (e.g. *NMCap* uses MS NetMon as default output format instead of LibPCAP).

Let us briefly discuss results of individual tools. NMCap achieved the lowest performance. It needs much more resources to complete the same task than others. It highly utilizes CPU and what is more severe its memory consumption is not scalable. Additional tests with *NMCap* revealed that trying to merge ten 1 GB large PCAPs together depletes all available memory and OS ends up with excessive swapping and reallocating memory pages. Hence, *NMCap* seems to be not suitable for merging large PCAP files.

We can observe a significant improvement in speed and more than twofold decrease of memory consumption when comparing current and old version of *PCAPMerger*. Hence, disengaging program from NetMon API had positive impact on overall performance.

The results show us that *Mergecap* outperforms our tools in the speed of merging task. That is because it uses the same approach based on the direct binary access to PCAP files and it has very simplified sorting logic. Hence, it needs nearly no memory because it doesn't need to cache anything. Still *Mergecap* is unable to deal with time order issue as all other competitors of *PCAPMerger*.

The last notable measured parameters are I/O reading and writing operations during run of each application. A comparison of the total number of operations does not clearly reflect effectiveness of the implementation – on the one hand a program could use just a few operations each obtaining a large block of data. On the other hand it could use many quicker operations each obtaining a smaller part of data. Tested disk file system is NTFS which has 4 KB as the default block size. Because of that we consider 4 KB of data transferred per operation as optimum. From all testing subjects only *PCAPMerger* and *Mergecap* reach this transferring speed. Notice excessive amount of data read by *PCAPMerger* – nearly 5.5 GB. Unfortunately this overhead is connected with .NET framework implementation of binary access to files. Related basic C# methods for reading, writing and seeking in file stream are "safe" so that they have exception catchers and fail safes which cannot be overridden and which introduce additional I/O Read operations. This illustrates chart on Fig. 5 where green bars represent portion of I/O Read operations and red bars represents I/O Write operations. During `SortFrames()` functions starting at time 5.4 second only I/O Write should occur but inside .NET framework those methods are connected with some additional I/O Read methods.

CONCLUSION AND FUTURE WORK

In this paper we summarized information about PCAP formats – namely structure of file, usage of timestamps in the frame of captured packets timing and ordering. We provided analysis of existing freely available tools for merging/concatenating multiple PCAPs into the one file and introduce our own solution.

We compared performance of our *PCAPMerger* on the testing set and proved that it is superior to our previous version based on NetMon API PCAP manipulation and the other tool called *NMCap*. *PCAPMerger*'s performance is comparable to *Mergecap* and thus it could be used as an equivalent

or even as a replacement in cases of merging files in PCAPng or MS NetMon format.

Deeper profiling of *PCAPMerger* reveals that it spends nearly 40% of execution time in `SortFrame()` function. Hence, we want to work on improving sorting algorithm in near future, thus making *PCAPMerger* even faster. We also want to make it functional part of bigger framework for handling PCAP files.

Source codes of current version of *PCAPMerger* application importable to Visual Studio 2010 could be downloaded from: http://www.fit.vutbr.cz/~ivesely/prods.php.en.

ACKNOWLEDGEMENT

This work was partially supported by the BUT FIT grant MV-VG20102015022 “Modern Tools for Detection and Mitigation of Cyber Criminality on the New Generation Internet” and in frame of ESF project CZ.1.07/2.3.00/09.0067 “TeamIT – Building Competitive Research Teams in IT”.

REFERENCES

[1] G. Harris. (2011, March) *Development/LibpcapFileFormat*. [Online]. Available from WWW: <http://wiki.wireshark.org/Development/LibpcapFileFormat/>

[2] T. Carstens. (2012, February) TCPDump&libpcap. [Online]. Available from WWW: <http://www.tcpdump.org/>

[3] L. Degioanni, F. Risso, and G. Varenni. (2009, July) *PCAP Next Generation Dump File Format*. [Online]. Available from WWW: <http://www.winpcap.org/ntar/draft/PCAP-DumpFileFormat.html>

[4] Microsoft. (2012, February) *Network Monitor - Site Home*. [Online]. Available from WWW: <http://blogs.technet.com/b/netmon/>

[5] S. Renfro and B. Guyton. (2012, February) *mergecap - The Wireshark Network Analyzer 1.5.0*. [Online]. Available from WWW: <http://www.wireshark.org/docs/man-pages/mergecap.html>

[6] B. Fenner. (2012, February) *The tcpslice project.* [Online]. Available from WWW: <http://sourceforge.net/projects/tcpslice/>

[7] P. Long. (2006, October) *NMCap: the easy way to Automate Capturing*. [Online]. Available from WWW: http://blogs.technet.com/b/netmon/archive/2006/10/24/nmcap-the-easy-way-to-automate-capturing.aspx>

[8] Microsoft. (2012, February) *Performance and Reliability Monitoring Step-by-Step Guide for Windows Server 2008*. [Online]. Available from WWW: http://technet.microsoft.com/en-us/library/cc749249.aspx

[9] Microsoft. (2012, July) *Windows Performance Analyzer*. [Online]. Available from WWW: <http://msdn.microsoft.com/en-us/library/windows/desktop/ff191077(v=vs.85).aspx>