# COMPARISON STUDY ON MERGING PCAP FILES

**Vladimír Veselý**

Doctoral Degree Programme (3), FIT BUT

E-mail: xvesel38@stud.fit.vutbr.cz

Supervised by: Miroslav Švéda

E-mail: sveda@fit.vutbr.cz

**Abstract**: PCAP is nowadays a widely used file format for storing computer network communications. This paper outlines information about PCAP Next Generation format with focus on packets precise timing and order. Paper also compares capabilities of different open-source tools for handling PCAP files and it introduces our own tool for merging multiple PCAPs into the one.

**Keywords**: merging PCAP files, sorting algorithm, Network Monitor API, PCAPMerger

## 1 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 accepted format nowadays is **PCAP** – formerly defined as a part of *libpcap* library. [1]

Often the computer communication is load-balanced and then available traffic captures come from multiple monitoring probes. We challenge problem how to "put together" captures correctly whenever we want to successfully trace particular traffic in this kind of environment.

This paper describes basic theory behind, existing tools, current issues and our software contribution to this topic.

## 2 STATE OF THE ART

In the following section we briefly mention some important notes on PCAP file format. We 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.

### 2.1. PCAP NEXT GENERATION FORMAT

PCAP Next Generation format for storing network communication never became more than IETF draft and currently is maintained outside any IETF working group. (Degioanni, Risso, & Varenni, 2009)

PCAP 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). The most important for this paper are following blocks:

- **Section Header Block (SHB)** – it is mandatory and 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, snaplength, 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).

PCAP blocks form tree structure. Physical layout of each PCAP file consists of at least one SHB, with one IDB and corresponding EPB and SPB packets sniffed on the interface. Typical PCAP file could have same structure as it is depicted on Figure 1.

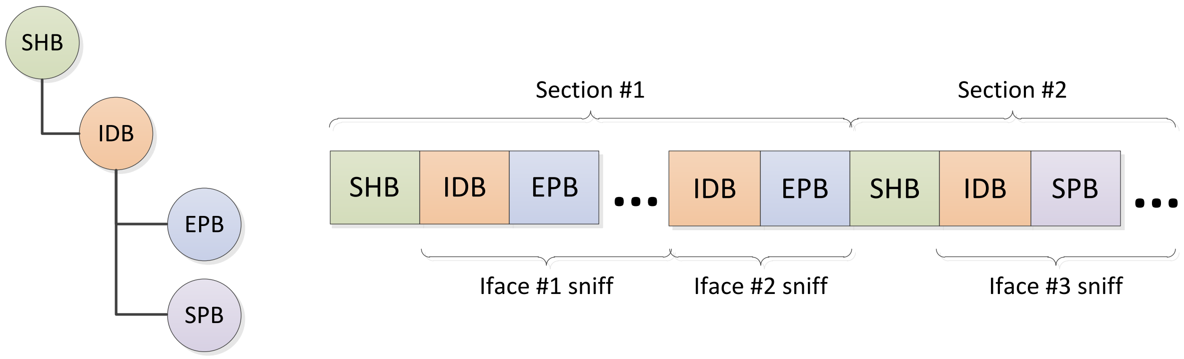

**Figure 1:** Tree structure and physical layout of PCAP blocks

## 2.2. Existing tools

There exists 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 Microsoft Network Monitor (MNM). For instance *capinfo* (part of Wireshark's installation) is 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* (Renfro & Guyton, 2012), FreeBSD's *tcpslice* (Fenner, 2012) or MNM's *NMcap* (Long, 2006).

## 2.3. Time Order Issue

Please note that time stamp is carried in PCAP Next Generation format only in EPB. Time information is stored in EPB in the two 32bit long fields which are interpreted according to the settings in IDB. They measure time since the beginning of UNIX epoch (1$^{st}$ January 1970) either in the number of units or in the number of seconds and milliseconds. In any case maximum achievable precision is 1 millisecond.

Assume that we have PCAP where timestamps in EPB are not growing incrementally. This kind of PCAP could be created usually in the following ways:

- More than one IDB is present – usually when sniff is done on the multiple interfaces.
- More than one SHB is present – PCAP was created by just simply concatenating two or even more PCAPs together.
- EPBs are disordered – either by purposely exporting them from the one PCAP in to another or because of delayed packet processing by capturing engine.

Packets can be received simultaneously on different interfaces or even on different capturing machines and can have (nearly) same timestamp. However relevant EPB are not stored in chronological order because of the physical layout in PCAP file.

Now if you use any previously mentioned tool (*Mergecap*, *NMcap* or *tcpslice*) and try to merge input PCAPs into the chronologically sorted PCAP then you will receive wrong output. Those programs take from each input PCAP file always the first unprocessed EPB and compare their timestamps between each other. Hence described algorithm just preserves bad time order of EPBs in resulting output PCAP. Existing tools simply expect only the basic physical layout of input files – one SHB, one IDB, and consecutive EPB.

## 3 CONTRIBUTION

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

Among existing APIs for manipulation with PCAP files (e.g. *libpcap/WinPCap*, *libnet*, *jNetCap*, etc.) we have chosen to use *NetMon* which is API build for MNM tool. (Microsoft, 2012)

*PCAPMerger* is programmed in C# language as a console application. It uses C# wrapper around *NetMon* API written in C++. With help of this wrapper we can directly enforce some API alternations. This comes out very useful when we bend API to handle significantly larger PCAP files – instead of default 500 MB we are now able to work with files limited only by file system size restrictions.

Without going to unnecessary details *PCAPMerger* runs as follows:

1) It processes input file's list as one of its execution arguments. It checks file's existence and verifies that it is in PCAP format. Then it simply concatenates content of all files into memory. Time complexity of this part: $O(n)$.
2) It closes input files and creates abstract data type frame vector from concatenated data. Time complexity of this part: $O(n)$.
3) It sorts frame vector according to timestamps of each frame using own delegates sorting function. Time complexity of this part: $O(n \log n)$.
4) It writes output PCAP file by exporting frames from concatenated data according to their position in sorted frame vector. Time complexity of this part: $O(n)$.

Thanks to the used sorting mechanism and initial preprocessing of data, our tool solves problem of the total time order among merging PCAPs.

## 4 PERFORMANCE TESTING

Only *PCAPMerger* is capable to correctly merge PCAP files inside which total time order are not kept. Nevertheless in this section we would like to show performance of our application in general use-case – merging ordinary PCAP files 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.

All participating programs – *Mergecap*, *NMcap* and our *PCAPMerger* – are compared on the same PCAP testing set. Testing set consists of the one big PCAP file with communication recorded on the backbone of the Brno University of Technology. Size of the big file is 1 GB to reflect "usual" PCAP file handling lengths. Big PCAP is split into ten chronologically consecutive smaller input PCAP files with the approximately 100 MB each.

All measurements are performed on "average machine" with Intel Core i5 CPU quadcore with 2.4 GHz, installed 4 GB of DDR2 RAM and running Windows 7 x64.

Smaller input files are passed to applications in chronologically reversed order to test the worst possible sorting case during testing routine. Following charts (Figures 2, 3 and 4) show results of running the application plotted by Windows Performance Monitor (Microsoft, 2012).

Red thick solid line shows CPU usage, blue thin solid line shows Memory usage and green dashed line shows I/O data rate (in bytes per second).

Results below are only from the one testing set. Although in reality we conducted more tests and their results are part of program documentation.

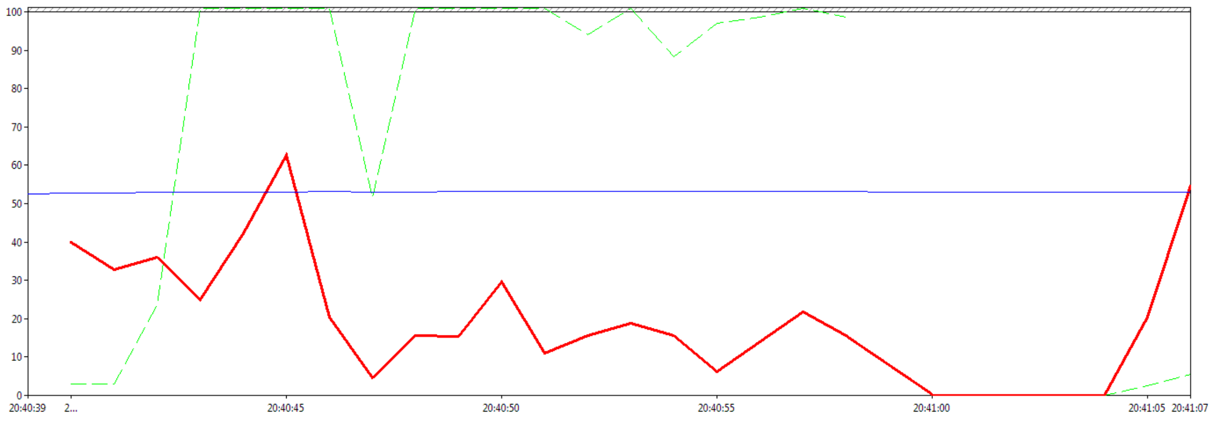

**Figure 1:** *Mergecap* performance results

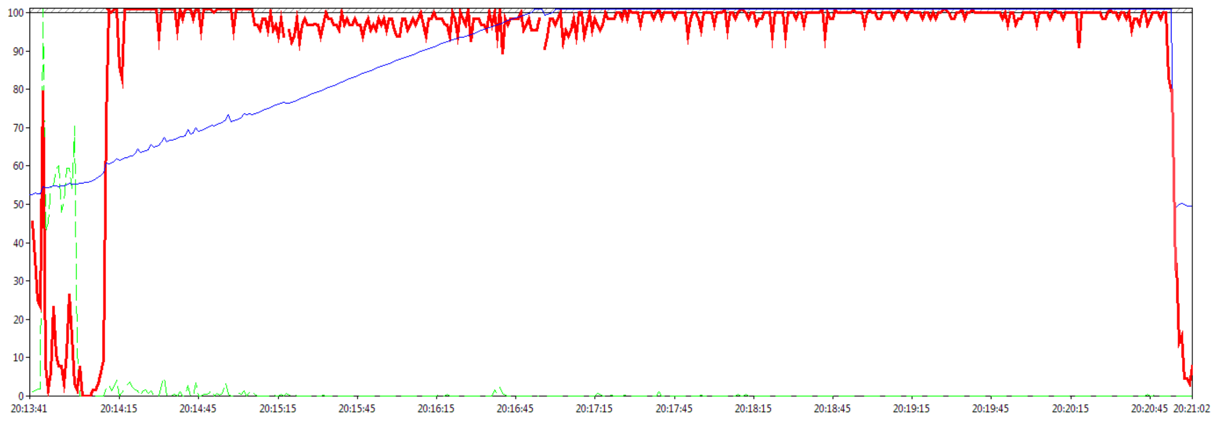

**Figure 2:** *NMcap* performance results

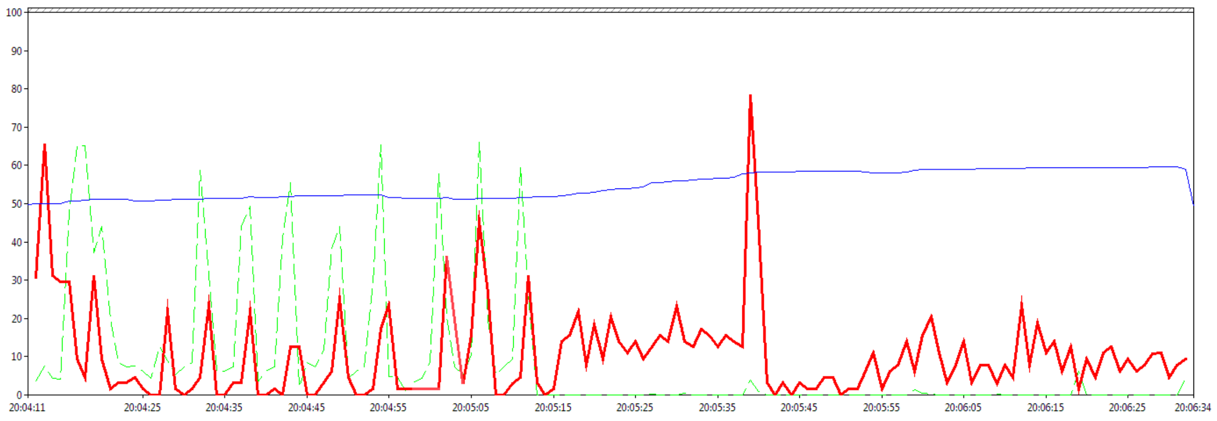

**Figure 3:** *PCAPmerger* performance result

Running time of each application is: *Mergecap* ≈ 27 seconds, *NMcap* ≈ 7 minutes 21 second and *PCAPMerger* ≈ 2 minutes 23 seconds. From all three applications only *NMcap* consumes whole CPU resources and computer's memory during run time. Subsequently *NMcap* forces OS to swap to page file thus radically decreasing performance. *Mergecap* has the most excessive I/O data rate.

Content of each output was compared with referential big PCAP file after every test finished. No differences were detected although sizes of output files slightly varied according to applications approaches to storing information (e.g. usage of PCAP-NG compression block).

On the one hand tests show that *Mergecap* finishes PCAP merging task quicker, on the other hand *NMcap* is significantly slower than other two applications. It is because of *Mergecap* simple sorting logic and well optimized *wiretap-1.0.0* library. But unfortunately *Mergecap* still cannot be used when dealing with time order issue. Although *NMcap* and *PCAPMerger* use the same MNM API, it seems that *PCAPMerger* is far more efficient. To conclude whole section *PCAPMerger* offers balanced performance (average CPU consumption with I/O data rate and nearly the same memory requirements as *Mergecap*) and additionally also solution for previously described problem.

## 5 CONCLUSION AND FUTURE WORK

In this paper we summarize information about PCAP Next Generation format – namely structure of file, usage of timestamps in the frame of captured packets timing and ordering. We provide analysis of existing free tools for merging/concatenating multiple PCAPs into the one file. With respect to issues of the previous tools we also introduce our own software for PCAP files merging. We compare performance of our *PCAPMerger* on the testing set and prove that it is superior to *NMcap* which is based on the same API.

We plan to continue our work on *PCAPMerger* performance. We expect improvements with other alternations of the Network Monitor API or further experiments with different sorting algorithm approaches. Our long term goal is to make *PCAPMerger* independent on any API and write own PCAP handling methods to provide best possible performance. We also plot to include *PCAPMerger* functionality into more general application working above network flows.

Source codes 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”.

## REFERENCES

[1] Carstens, T.: *TCPDump&libpcap*. [Online]. HYPERLINK "http://www.tcpdump.org/" http://www.tcpdump.org/ . February, 2012.

[2] Degioanni, L.; Risso, F.; Varenni, G.: *PCAP Next Generation Dump File Format*. [Online]. HYPERLINK "http://www.winpcap.org/ntar/draft/PCAP-DumpFileFormat.html" http://www.winpcap.org/ntar/draft/PCAP-DumpFileFormat.html . July, 2009.

[3] Renfro, S.; Guyton, B.: *Mergecap - The Wireshark Network Analyzer 1.5.0*. [Online]. HYPERLINK "http://www.wireshark.org/docs/man-pages/mergecap.html" http://www.wireshark.org/docs/man-pages/mergecap.html . February, 2012.

[4] Fenner, B.: *The tcpslice project*. [Online]. HYPERLINK "http://sourceforge.net/projects/tcpslice/" http://sourceforge.net/projects/tcpslice/ . February, 2012.

[5] Long, P.: *NMCap: the easy way to Automate Capturing*. [Online]. HYPERLINK "http://blogs.technet.com/b/netmon/archive/2006/10/24/nmcap-the-easy-way-to-automate-capturing.aspx" http://blogs.technet.com/b/netmon/archive/2006/10/24/nmcap-the-easy-way-to-automate-capturing.aspx . October, 2006.

[6] Microsoft: *Network Monitor - Site Home*. [Online]. HYPERLINK "http://blogs.technet.com/b/netmon/" http://blogs.technet.com/b/netmon/ . February, 2012.

[7] Microsoft: *Performance and Reliability Monitoring Step-by-Step Guide for Windows Server 2008*. [Online]. HYPERLINK "http://technet.microsoft.com/en-us/library/cc749249.aspx" http://technet.microsoft.com/en-us/library/cc749249.aspx . February, 2012.